Neural networks to predict clinical events from cytometry data

Dissertation zur Erlangung des

Doktorgrades der Naturwissenschaften (Dr. rer nat.)

der Fakultät für Biologie und Vorklinische Medizin

der Universität Regensburg

vorgelegt von

Gunther Glehr

aus

Regensburg

im Jahr

2021

Der Promotionsgesuch wurde eingereicht am:

21.12.2021

Die Arbeit wurde angeleitet von:

Prof. Dr. Rainer Spang

Unterschrift:

Gunther Glehr

Abstract

From a computational perspective, clinical decision making requires classifying a patient into classes that respond similarly to treatment.

Cytometry characterises individual cells in patient specimens by size, complexity or surface markers. The result is a matrix per specimen with an unordered number of rows reflecting the cells and a defined number of cell-parameters as columns. These data can be used for classification, hence also for clinical decision making.

Classically, predictive modelling uses cell subpopulation quantities as a predictor to classify each sample. The cell subpopulations are defined by binning cells with similar characteristics. Sequential gating, manually defining cuts in the parameters, or automated clustering are possibilities for binning.

In contrast to this two-step procedure, this thesis investigates the application of neural networks to classify cytometry samples without prior cell subpopulation identification.

I provide a Cell Cloud Classification python package (CCC): A modular framework for neural networks to classify and investigate cytometry samples. The investigation includes a novel, decision-tree based concept to explain the predictions.

I study the impact of hyperparameters by classifying public cytomegalovirus samples. The impact was measured using the classification AUC. Here, using the mean proved superior to maximum pooling. There were an optimum number of layers, but the number of nodes was irrelevant. The number of drawn cells, but neither replacement nor focus on rare cells were relevant. Surprisingly, network initialisation affected performance, but validation and test performance correlated.

I study CCC in binary and multiclass simulations. Additionally, classifying T-cell based flow cytometry samples of (non-) starved human blood resulted in reliable but non-classical discriminative cell subpopulations.

In summary, I present a python package to classify cytometry samples using neural networks and show its application to real and simulated datasets.

Zusammenfassung

Aus der Sicht von computerunterstützer Analyse erfordert die klinische Entscheidungsfindung die Einteilung von Patienten in Klassen welche ähnlich auf Behandlungen ansprechen.

Zytometrie charakterisiert einzelne Zellen von Patientenproben nach Größe, Komplexität oder Oberflächenmarkern. Daraus ergibt sich eine Matrix pro biologischer Probe mit einer ungeordneten Anzahl von Zeilen. Die Zeilen dieser Matrix spiegeln die gemessenen Zellen der Probe wieder und die Spalten eine definierte Anzahl von Zellparametern. Diese Daten können zur Klassifikation und damit auch zur klinischen Entscheidungsfindung verwendet werden.

Klassischerweise verwendet prädiktive Modellierung Zellsubpopulationshäufigkeiten als Prädiktoren, um jede biologische Probe zu klassifizieren. Die Zellsubpopulationen werden durch das Eingruppieren von Zellen mit ähnlichen Eigenschaften definiert. Sequentielles gating, das manuelle definieren von Grenzwerten der Zellparametern, oder automatisiertes Clustering sind Gruppierungsmöglichkeiten.

Im Gegensatz zu diesem zweistufigen Verfahren untersucht diese Arbeit die Verwendung neuronaler Netze zur Klassifizierung von Zytometrieproben ohne vorherige Identifizierung der Zellsubpopulationen.

Ich präsentiere das Python Paket CCC Cell Cloud Classification (Zell-Wolken-Klassifikation): Ein modulares Framework für neuronale Netze zur Klassifizierung und Untersuchung von Zytometrieproben. Die Untersuchung beinhaltet ein neuartiges, entscheidungsbaumbasiertes Konzept zur Erklärung der Vorhersagen des neuronalen Netzes.

Ich untersuche den Einfluss von Hyperparametern, indem ich öffentlich verfügbare biologische Proben mit oder ohne CMV klassifiziere. Anhand der klassifierungs AUC wurde die Performance beurteilt. Zusammenfassen der Zellen mithilfe des Mittelwerts war der Nutzung des Maximums überlegen. Für das neuronale Netz gab eseine optimale Anzahl von Schichten, aber die Anzahl der Knoten spielte keine Rolle. Die Anzahl der gezogenen Zellen, aber weder (nicht) wiederholtes ziehen noch der Fokus auf seltene Zellen waren relevant. Überraschenderweise beeinflusst die Netzwerkinitialisierung die Performance, aber Validierung und Testperformance korrelierten.

Ich untersuche CCC in Binär- und Mehrklassensimulationen. Darüber hinaus führte die Klassifizierung von T-Zell-basierten Durchflusszytometrieproben von (nicht) gefasteten Blutproben zu zuverlässigen, aber nicht klassisch diskriminierenden Zellpopulationen.

Zusammenfassend stelle ich ein Python-Paket vor, um Zytometrieproben mit neuronalen Netzen zu klassifizieren und ich zeige die Anwendung auf reale und simulierte Datensätze.

Summary

From a computational perspective, clinical decision making requires classifying a patient into classes that respond similarly to treatment.

Cytometry characterises individual cells in patient specimens by size, complexity or surface markers. The result is a matrix per specimen with an unordered number of rows reflecting the cells and a defined number of cell-parameters as columns. These data can be used for classification, hence also for clinical decision making.

Classically, predictive modelling uses cell subpopulation quantities as a predictor to classify each sample. The cell subpopulations are defined by binning cells with similar characteristics. Sequential gating, manually defining cuts in the parameters, or automated clustering are possibilities for binning.

In contrast to this two-step procedure, this thesis investigates the application of neural networks to classify cytometry samples without prior cell subpopulation identification.

I build on the results from point cloud classification[1] which was applied successfully in cytometric data analysis by CellCNN[2] and DeepLearningCyTOF[3]. CellCNN[2] showed the successful application of shallow neural networks in flow and mass cytometry data. DeepLearningCyTOF[3] expanded the idea to deep neural networks and showed their superiority in a mass cytometry dataset predicting latent cytomegalovirus (CMV) in peripheral blood samples. Additionally, DeepLearningCyTOF[3] proposed a decision tree based algorithm to explain the predictions of the neural networks.

There are some limitations in the current literature when applying neural networks to cytometry data. First, there is no ready-to-use package for neural network application on cytometry data. Second, explaining the predictions of the neural network is still an issue. Third, neural networks are heavily dependent on manually set hyperparameters, but no extensive investigation of their impact has been provided yet regarding cytometry data.

In this work, I propose the *FeatureLearner*, *Pooler* and *Predictor* concept (FPP). This concept formalises the structure of neural networks applicable for cytometry data which are unordered cells with a defined number of cell-parameters.

The FPP concept is realised in the python package CCC, which is currently available as gitlab repository in the group of Prof. Dr Spang. I provide the Cell Cloud Classification python package (CCC). A modular framework for neural networks to classify and investigate cytometry samples.

The investigation inside CCC includes a novel, decision-tree based concept to explain the predictions. The important internal features of the neural network are identified. In contrast to a single built decision tree for the complete network[3], multiple decision trees are built of each important feature. Multiple trees enable a

more nuanced explanation of the behaviour of the neural network.

I study the impact of hyperparameters by classifying public CMV samples. The impact was measured using the classification AUC. Here, using the mean proved superior to maximum pooling. There were an optimum number of layers, but the number of nodes was irrelevant. The number of drawn cells, but neither replacement nor focus on rare cells were relevant. Surprisingly, network initialisation affected performance, but validation and test performance correlated.

I study CCC in binary and multiclass simulations. Additionally, classifying T-cell based flow cytometry samples of (non-) starved human blood resulted in reliable but non-classical discriminative cell subpopulations.

In summary, I introduce FPP, a structural concept for neural networks in cytometry and present a python package applying FPP to classify cytometry samples. I propose a novel, decision tree based explanatory concept for neural networks in cytometry and show the impact of neural network hyperparameters. Finally, I show the application of CCC in two simulated and two real datasets.

Acknowledgements

This work was carried out in the Department of Statistical Bioinformatics of the Institute of Functional Genomics at the University of Regensburg in the Regensburg International Graduate School of Life Sciences.

I would like to thank my supervisor, Rainer Spang, for giving me the great opportunity to work on many different interesting topics and for his guidance over the last few years. In his group, I was able to gather experience in many ways, from teaching over writing to many opportunities of presenting at conferences.

I also want to thank my second supervisor Wolfram Gronwald, whose calm but well elaborated ideas greatly helped, especially during times of searching for a topic.

I am grateful for my third supervisor, Holger Fröhlich, whose brilliant ideas, hints and suggestions proved invaluable. Without those, this work would look quite different.

I am thankful to RIGeL and its team, as their uncomplicated but well-organised program proved valuable during the whole time and especially in the final days of submitting the thesis.

I am grateful for Katharina Limm, who, especially in the early stages of the PhD was never tired of explaining to me the technical and biological details when I probably asked every other day why measurements cannot be done in another way.

I am also thankful for the fact that Peter Oefner, Katharina Limm and Wolfram Gronwald proposed all variants of topics when I was in desperate need of something to build my thesis on.

Also Michael Altenbuchinger was always easy-to-reach and lent an ear anytime I needed advice or wanted to discuss; thanks for that!

For over two years now, James Hutchinson showed me the secrets of flow cytometry and academia which I am very grateful for. I am very happy to collaborate soon further and am excited for our further path.

Katharina Kronenberg, from James' group, helped me out big time with additional data necessary to test my algorithms and ideas, I hope I was not too demanding in the process.

Of course, I have to thank the people of the Spang-group! Foremost Sharon Peterson, for who was able to make paperwork seem like a piece of cake. Marian Schön for his invaluable assistance with discussions, proofreading, beer-measuring, going-for-one-beer (ahh, the times before kids) and his determination to take all of the teaching obligations from me. Tobias Schmidt for his great input and the discussions, the name FPP and obviously the short-notice availability. Lena Buck for her willingness for discussions and elaboration on my stupid math-related questions. Michael Huttner for his great help with shipping, containers and anything technical-related. That helped me out of desperation multiple times. Jakob Simeth, Rudolf Schill, Kevin Rupp, Zahra Nozari and Thorsten Rehberg for the discussions and

their input - I miss BIC times. Claudio Lottaz and Christian Kohler for being the best room colleagues I ever had. On top of that, for their ultra-helpful knowledge about existing data, code and the server infrastructure at our institute. Nicole Seifert, for great conversations, excellent proofreading and regular advice. It's great this still works out even with Corona and the distance! Also, Christoph Kopp for advice on Latex, general coffee-fellow and Leberkaese-buddy.

Also, thanks to the Oefner-group with Karja Dettmer-Wilde, Wolfram Gronwald, Katharina Limm, Fadi Fadil, Simon Heckscher, Lisa Ellmann, Elke Perthen, Claudia Bogner, Raffaela Berger, Xueni Sun and Leo Kramer - I loved that when I went into the laboratory, I always felt welcome and all of you helped me wherever you could! Thanks for the time and the regular great cake-times.

I would like to express my gratitude to all named and unnamed, supervisors, colleagues, friends and family who supported me in my endeavours while writing this dissertation. Whether through supervising and co-authoring, listening and discussing, encouraging and inspiring, or just being there whenever a challenge presented itself, your efforts were greatly appreciated and helpful to its successful completion.

Finally, I want to thank my mother for her support, strength and care for me. However difficult times were, she was always there for me. Mama, I love you.

Contents

Abstract iii

Zusammenfassung v

Summary vii

Acknowledgements ix

Contents xi

Glossary xv

List of Figures xxi

List of Tables xxiii

I. Cell cloud classification 1

1. Introduction 3
- 1.1. Aim of CCC . . . 3
- 1.2. Predictive modelling . . . 3
- 1.3. What is cytometry data? . . . 7
  - 1.3.1. Flow cytometry . . . 7
- 1.4. How to analyse cytometry data? . . . 10
  - 1.4.1. Preprocessing . . . 11
    - 1.4.1.1. Compensation . . . 11
    - 1.4.1.2. Transformation . . . 13
    - 1.4.1.3. Filtering . . . 14
    - 1.4.1.4. Batch correction . . . 15
    - 1.4.1.5. Subsampling . . . 15
  - 1.4.2. Cell subpopulation identification . . . 17
    - 1.4.2.1. Sequential manual gating . . . 17
    - 1.4.2.2. Supervised cell population identification . . . 17
    - 1.4.2.3. Unsupervised cell population identification . . . 19
  - 1.4.3. Predictive modelling . . . 19
- 1.5. Cytometry data are high dimensional point clouds . . . 20
- 1.6. Supervised learning . . . 21
- 1.7. Neural networks . . . 23
  - 1.7.1. Logistic regression leads to neural networks . . . 23
  - 1.7.2. Single layer perceptron . . . 26

1.7.3. Training a neural network . . . 27
1.7.4. Multi layer perceptron . . . 29
1.7.5. Convolutional neural network . . . 30
1.7.6. Batch normalisation . . . 31

2. Results **33**
2.1. FPP concept . . . 33
2.2. CCC workflow . . . 35

3. Discussion **39**

II. CCC design and performance 43

1. Introduction **45**

2. Methods **47**
2.1. CMV dataset . . . 47
2.2. Tested hyperparameters of neural networks in cytometry . . . 47
2.2.1. Subsampling methods . . . 47
2.2.1.1. Fixed and unfixed . . . 48
2.2.1.2. Draw randomly or based on outlierness . . . 48
2.2.2. Network architecture . . . 48
2.2.3. Weight initialisation . . . 49
2.2.4. Two separate experiments . . . 49
2.2.5. Training, validating and testing . . . 49

3. Results **51**
3.1. Overview over the two experiments . . . 51
3.2. Mean pooling performs better than max pooling . . . 52
3.3. Weight initialisation is crucial for the final performance . . . 54
3.4. Performance depends on network architecture . . . 55
3.5. Subsampling affects performance ranges . . . 56

4. Discussion **59**

III. Identify discriminative cell subpopulations 63

1. Introduction **65**
1.1. How to describe decisions of a FPP network . . . 65
1.1.1. Shapley values . . . 66
1.1.2. Shapley values in machine learning . . . 68
1.1.3. SHAP values . . . 70
1.1.4. Deep SHAP . . . 71

2. Methods **75**
2.1. Binary classification simulation . . . 75

2.2. Multiclass classification simulation . . . 75

3. Results **77**
3.1. Simulation . . . 77
3.1.1. Binary classification . . . 77
3.1.2. Multiclass classification . . . 77
3.2. Gate finding . . . 78
3.2.1. Relevant subset identification . . . 80
3.2.2. Gating directions . . . 82
3.2.3. Replace *FeatureLearner* with an decision tree . . . 85
3.2.4. Find the gating by traversing the decision trees . . . 85
3.3. Apply final gatings . . . 88

4. Discussion **89**

IV. Application of CCC 93

1. Introduction **95**

2. Methods **97**
2.1. Simulations . . . 97
2.2. Binary classification simulation . . . 97
2.3. Multiclass classification simulation . . . 97
2.3.1. FPP network settings for binary and multiclass classification 97
2.4. Identifying fasting blood samples with CCC . . . 98
2.4.1. ANTE vs POST samples . . . 98
2.4.2. FPP network settings . . . 99

3. Results **101**
3.1. Binary classification simulation . . . 101
3.2. Multiclass classification simulation . . . 104
3.2.1. Theoretical performance boundaries for multiclass classification simulation . . . 104
3.2.2. Multiclass simulation results . . . 104
3.3. Identifying fasted blood samples with CCC . . . 110

4. Discussion **115**

Bibliography **121**

Appendices **139**
A. Flow cytometry data: FCS files and . . . 139
B. Flow cytometry data transformation . . . 141
C. Performance experiment 1: All hyperparameter's effects on performance . . . 144

D. Performance experiment 2: All parameter and network architecture effects on performance 145
E. Performance experiment 2: All parameter and network architecture effects on the AUC-range 146
F. Multiclass classification simulation: All decision trees and gatings 147
G. Fasting data: Full gating graph 150

Glossary

[Cell] subsampling Cell subsampling subsamples n cells of a biological sample. 15, 46–49, 56, 59–62, 97, 104, 115

- **fixed** Fixed subsampling subsamples n cells from a biological sample once before using it in predictive modelling. 47–49, 52, 56, 60, 61, 99
- **unfixed** Unfixed subsampling subsamples n cells from a biological sample every time it is used. In contrast to fixed subsampling, the same subsampled biological sample only contains the same cells if the total number of cells N in the biological sample is n (when sampling without replacement). 47–49, 52, 56, 60, 61, 97, 99
- **random** n cells are randomly chosen from a biological sample, can be fixed or unfixed. 15, 46, 47, 49, 52, 56, 60, 61
- **outlierness-based** n cells are chosen according to their outlierness from a biological sample, can be fixed or unfixed. 16, 46, 47, 49, 52, 56, 60, 61

ADC Analog-to-digital converter xvi, 8, 140, 141

bin An interval on the x-axis of a histogram. 140

CCC design and performance nomenclature In part II where I investigate the impact of hyperparameters on classification performance I need to differ between variants and runs.

- **variant** One variant is one set of hyperparameters defining the network architecture and how cells are subsampled as network input. xv, 47, 49, 51–57, 59, 60, 62, 115, 144, 145
- **run** One specific run includes multiple variants, where all variants were initialised with the same internal weights. Therefore each run refers to a certain seed for the pseudo-random number generator. Even when setting all hyperparameters, the classification performance of a neural network will be different depending on the randomly initialised internal weights. Therefore to ensure reproducibility you can manually set a seed which defines the pseudo-random initialised weights. xv, 47, 49, 51–57, 59, 60, 114, 115, 144–146

CD Cluster of differentiation 8, 9

cell subpopulation Cells with similar properties are binned into cell subpopulations. xvii, 3–7, 10–12, 15, 17, 19–21, 39, 41, 42, 45, 59–61, 65–67, 78, 85, 86, 88–92, 95, 96, 98, 109, 115–119

- **cell subpopulation identification** The process of identifying cell subpopulation based on the cell parameters ... 10, 11, 19, 20, 39, 66
  - **gating** ... through manually defined cutoffs. E.g. a cell with parameter$_A$ "high" and parameter$_B$ "low" would be in the $A^+|B^-$ cell subpopulation. iii, vii, 4, 5, 10, 11, 15, 17–20, 36, 37, 42, 65–67, 78, 85, 86, 88, 90, 91, 95,

102, 103, 106, 108–114, 116–119, 139, 147, 148, 150

supervised ... through supervised learning where the cell subpopulation label of some cells is already known. 18

unsupervised ... through unsupervised learning, e.g. k-means clustering. 5, 19, 45, 65, 95

cell subpopulation identification The process of quantifying cell subpopulations in a biological sample. This results in absolute counts of cells in each subpopulation or relative values of how large one subpopulation is inside its parent subpopulation

cell subpopulation label The label of a cell subpopulation. From biology this is T-cell or B-cell, from unsupervised clustering this is cluster_1 or cluster_2.

discriminative cell subpopulations A cell subpopulation differing in quantity between two classes of samples. 10, 91, 95, 116, 117

cell-property Anything that defines a cell: Size, density, complexity, weight, viability, enzymatic activity, ... xvii, xix, 3, 7, 8, 11, 12, 39, 40, 45, 59, 65, 89, 95, 115

channel An ADC with $b \in \mathbb{N}$ bits can represent an analog value with 2^b digital values. One single of these values is a channel. 140

class xvii, xxii, 3–6, 10, 11, 19–21, 23, 33, 35, 36, 39–41, 45, 59, 65, 75, 77–80, 82–85, 88–90, 95–98, 101–113, 115–119, 147, 148

of a patient The class a patient is inside. E.g. High or low risk, young or old, cancer stage I or II. The class is not necessarily constant over time: E.g. Fasting vs a non-fasting sample of a single patient.

of a sample The class of a sample. It does not change and is recorded at the time of measuring that sample.

classification Process of assigning elements to pre-establishes classes. xv, xxi, 3, 5, 21, 23, 33, 36, 39–42, 45, 46, 48, 53, 60, 61, 66, 78, 90, 101, 102, 115–117, 119

binary classification ...when there are only two possible classes. Special case of multiclass classification. 33, 34, 40, 41, 47, 49, 60, 66, 75, 77, 80–82, 84, 86, 88, 95–97, 101, 104, 106, 110, 115–117, 119

multiclass classification ...when there are only more than two possible classes. xiii, 18, 33, 40, 66, 75, 77, 78, 80, 84, 85, 95, 97, 104, 106, 116, 118, 119, 147

cluster of differentiation Protocol to identify and investigate cell surface marker molecules for immunophenotyping cells. E.g. CD3, which is a T-cell marker.

CMV Latent Cytomegalovirus

compensation The process of removing spillover of a flow cytometry sample. 10–13, 15, 139–141

doublet An event where two cells are detected at the same time. 9, 11, 14

event [in a flow cytometry sample] A particle detected by the flow cytometer. This is not necessarily a single cell as doublets or debris might be detected. 5, 9, 11, 12, 14, 139

FCS Flow Cytometric Data File Standard 10, 139

feature A descriptive property of an element. xvii, 5, 11, 20, 21, 23, 26–28, 30, 32–34, 41, 66, 68–72, 82

of a cell Artificially created value of a cell created from combining cell-parameters, possibly describing a cell-property. xvii, 33, 36, 40–42, 45, 48, 56, 59, 62, 65, 66, 78–91, 95, 98, 99, 101–103, 106, 108–112, 115, 116, 118, 119, 148, 150

of a sample A value of a sample usable in an algorithm. E.g. The number of cells in a sample, the percentage of a cell subpopulation, phenotypic data like age, sex or smoking status.

filtering Filtering a flow cytometry sample removes irrelevant cells, e.g. dead cells and doublets. 10, 11, 35

fluorochrome A fluorochrome is a fluorescent dye which can be used to stain biological material. xx, xxi, 8, 11–13, 46, 98, 99

FPP concept A modular concept to organise neural networks for point cloud data. It consists of three parts: The *FeatureLearner*, *Pooler* and *Predictor* xxi, 3, 33, 35, 41, 45, 59, 60, 65, 66, 75, 87–92, 95, 97, 101, 115–117, 119

FeatureLearner The input is a matrix with cells as rows and all cell parameters as columns. Results in a new matrix with (usually) the same amount of rows but new, artificially created cell-features. xvii, 3, 33, 34, 36, 40, 42, 45, 48, 49, 53, 55, 56, 59–62, 65, 66, 78–80, 85, 87, 89–91, 95, 97, 99, 101, 104, 106, 110, 115–119, 144–146, 148,

Pooler Takes a matrix with cells as rows and cell features as columns. Aggregates each feature across all cells. xvii, 3, 33, 34, 40, 45, 46, 48, 49, 52, 54, 55, 59, 62, 65, 78, 89, 95, 97–99, 115, 118,

Predictor Takes a vector of features to predict a sample. Usually this vector is the aggregation-result of the Pooler. xvii, 3, 33, 34, 40, 41, 45, 49, 53, 55, 56, 59–62, 65, 80, 82, 89, 90, 95, 97–99, 101, 104, 115–118, 144–146,

gating direction The combination of pro/contra and the lower/higher direction results in the gating direction of one specific input feature of the *Predictor*, the pooled cell-features. Each cell-feature plus pooling combination is the input for the *Predictor*. High(or low) values of a cell-feature can speak for (pro) or against (contra) each possible class which is found by SHAP values. xvii, 79, 82, 84, 85, 88, 90, 91, 102, 103, 108, 110, 112, 116, 118, 148

HDF5 Hierarchical Data Format 5 139

ImmPort Data warehouse for immunological data. 47

LMD List-mode data 139, 140

mAb Monoclonal antibody 8, 9

MIFlowCyt Minimum Information about a Flow Cytometry Experiment[4] 139

neural network (Artificial) neural network. Machine learning method combining input values in multiple nodes over multiple layers to get a data-based prediction. i, xi, xii, 3, 6, 18, 23, 26, 28–34, 37, 39–42, 45–47, 49, 54, 59–62, 65, 66, 70–72, 75, 80, 88–92, 95, 97, 98, 101, 115–117, 119

activation function The function applied after the linear combination of a node. A common example is the ReLU ($f(x) = max(0, x)$). 23, 26–30, 35, 45, 62, 72, 73

architecture Defines the number of nodes, layers, activations and how they are interconnected in a neural network. xiv, xv, 23, 27, 45–49, 51, 52, 54, 55, 59,

60, 62, 97, 115, 144–146

batch size The $n \leq N$ number of total N samples used to apply the neural network on and calculate the gradient. 32, 49, 98, 99

epoch One epoch is when all N samples are used once in calculating the gradient during training. 36, 49, 61, 98

gradient While training a neural network, the weights are optimised to represent the data best. To find those weights, the gradient given a dataset is calculated, and the weights get modified such that the loss function becomes minimal. 27, 29

hyperparameter Setting of a neural network that does not get optimised during training. E.g. How each FPP-part is built internally by defining the number of layers, the number of nodes in each layer, the choice of the optimisation algorithm or activation functions. xiii, xv, 23, 27, 35, 45–47, 51, 52, 55, 56, 59–62, 90, 97, 99, 100, 110, 111, 115, 118, 144, 145

layer 6, 23, 29, 30, 32, 34, 35, 40, 45, 46, 48, 49, 53, 55, 56, 59–62, 72, 97–99, 101, 104, 115–119, 144–146

- **batch normalisation** Layer which standardises the previous layer for each batch. 32, 34, 48, 49, 56, 98, 99, 118, 144
- **convolutional layer** Layer where instead of one weight for every input, weights are shared across a defined structure of multiple inputs. The FPP-FeatureLearner implements p weights for p input features, shared across all input cells. 30
- **dense layer** Layer where all nodes from the previous layer are connected to all nodes in the current layer. 29, 30, 42, 45, 46, 48, 49, 62, 73
- **hidden layer** Layer of nodes between inputs and output. 29, 30, 49, 53, 62
- **input layer** Contains the raw data, each input is a node.
- **output layer** Layer of the final classification problem. A single node in binary classification, k nodes for a multiclass classification problem with k classes. 33, 84, 119
- **pooling layer** Layer aggregating multiple nodes into one value. The FPP-Pooler enables permutation invariant functions like the mean or maximum to aggregate all 6, 30, 101, 116, 117

multi layer perceptron Neural network consisting of multiple layers of linear combinations followed by various activation functions. 27, 29, 30, 34, 49

node A mathematical operation applying a linear combination using all inputs and passing the result through an activation function. 23, 34, 35, 40, 45, 46, 48, 49, 53, 56, 59, 60, 62, 80, 97–99, 106, 115, 117, 118, 144, 145

seed A seed in pseudo-random experiments enables reproducible "random" values. 46, 47, 49, 54, 60, 62

single layer perceptron Neural network consisting of no hidden layer, only a linear combination of inputs followed by an activation function to get

the prediction. 23, 26–29

weight The weights for each node applied at the linear combination. xv, 23, 26–28, 31, 34, 42, 45–47, 49, 51, 54, 55, 59–61, 115

one-hot encoding For each unique value of multiple classes a new binary column is created. 98

outlierness The outlierness of a cell is proportional to the euclidean distance to other cells. xv, 16, 17, 46, 48, 51, 61, 144

parameter 3, 33–35, 139–141

in flow cytometry The parameter of a flow cytometry measurement, e.g. forward/side scatter height, width or area, but fluorescence markers: FL1 (fluorescence 1), FL2, ... , height, width or area.

of a cell A measured cell-property. xvii, 3–8, 10–21, 31, 33, 34, 36, 37, 39–41, 45–49, 59, 61, 65, 66, 75, 77, 85–91, 95–99, 101, 103, 109, 110, 112, 115–119

of a model A configuration variable of a model. 22–29, 32

permutation invariance Cytometry and point cloud data are unordered cells/items with multiple features. The unordered nature of items can be coped with by introducing a function invariant to the permutation of items. Symmetric functions like the mean or maximum are intrinsically permutation invariant. 33, 48,

point cloud A set of points in space, usually 3D coordinates. xi, 20, 21, 33, 35–37, 40, 60

prediction xxi, 6, 34, 35, 37, 40–42, 45, 53, 56, 57, 60–62, 65, 68–71, 82–84, 89–91, 95, 98, 102, 106–108, 111, 116, 118, 146

of a sample's class The predicted class of a sample, based on the measurements of the sample.

predictive modelling Predictive modelling uses data to predict the data's class affiliation. xv, 3–6, 10, 11, 15, 19–21, 35, 39, 41, 45, 46, 48, 59, 61, 65, 89, 95, 115

python packages

hydra Hydra is an open-source Python framework that simplifies the development of research and other complex applications. 35, 36, 62

python A high-level general-purpose programming language. 3, 35, 40, 41, 73, 95, 115, 117–119, 139

pytorch An open source, optimised tensor library for deep learning using GPUs and CPUs based on torch. 30, 31, 35, 49, 98

pytorch-lightning The lightweight PyTorch wrapper for high-performance AI research. 35

tensorboard TensorBoard is a suite of web applications for inspecting and understanding your TensorFlow runs and graphs. 35

sample

biological sample One measurable instance of a patient xv, 3, 4, 7, 10, 11, 15, 19–21, 31, 33–37, 39–42, 45, 46, 48, 53, 56, 57, 59–62, 65, 66, 75, 77, 78, 80–85, 88–92, 95–98, 101–112, 115–119, 148

flow cytometry sample One data instance of a patient, measured with a flow cytometer. Usually a matrix of many thousand rows (=cells) with up

to 50 columns (parameters)

SHAP SHapley Additive exPlanations. A game theoretic approach to explain the output of any machine learning model. xvii, 66, 70–73, 80, 82–84, 90, 91, 95, 101, 102, 106, 108, 110, 111, 116, 118, 119

Shapley value A solution concept in cooperative game theory. 66, 68–71, 80

spillover Spillover is when a fluorochrome adds brightness to another fluorochrome's measurement. Fluorochrome 1 "spills over" into fluorochrome 2. 11–13, 139

supervised learning A machine learning concept to learn a function based on exemplary labelled examples. 20, 21, 23, 34, 66

AUC Area under the curve. The area under the ROC-curve. A perfect classifier has an AUC of 1, a random classifier an AUC of 0.5. 49, 51–57, 60–62, 101, 106, 107, 110, 117, 119, 144

decision tree A decision tree in machine learning builds a tree of splits from a sample's features resulting in leaves representing the final predicted class. xiii, xiv, xxi, xxii, 18, 36, 37, 65–67, 78, 79, 85, 86, 88–91, 95, 102, 103, 106, 108–112, 115, 116, 119, 147, 148

one-vs-all One-vs-all is a concept in multiclass classification to enable binary classification methods and measures for multiclass problems. Each of the multiple classes transforms into a binary problem: Class I vs all others, class II vs all others, ...

ROC Receiver operating characteristic curve. A plot shows the true positive on the y-axis and the false positive rate on the x-axis at various threshold settings for a binary classification problem. 101, 102, 106, 110, 117, 119

splitting data In machine learning data is used to train and evaluate the algorithm.

test data After the final model was trained with the optimal hyperparameters, the final model's performance must be assessed on an additional dataset.

training data An algorithm is initially fit on the training data set.

validation data To prevent overfitting, the algorithm performance must be assessed on a separate dataset. This data is used to tune model hyperparameters.

List of Figures

1.1. Partitioning and clustering concepts . . . 7
1.2. Fluid-based flow cytometer schematic[43] . . . 8
1.3. Schema of the measurement of a single particle[46] . . . 9
1.4. Flow cytometry analysis workflow . . . 10
1.5. Normalised emission spectrum of fluorochromes FITC and PE . . . 13
1.6. Cytometry transformations . . . 14
1.7. Cell subsampling variants . . . 17
1.8. Example gating of naive Tcells . . . 18
1.9. Cell population identification . . . 20
1.10. Single layer perceptron . . . 26
1.11. Single layer perceptron, pseudo-bias . . . 27
1.12. Single layer perceptron, functional form . . . 28
1.13. Multi layer perceptron . . . 30
1.14. Convolutional layer . . . 31

2.1. FPP concept . . . 34
2.2. Exemplary FeatureLearner . . . 34
2.3. Exemplary Predictor . . . 34
2.4. CCC workflow . . . 37

3.1. Initialisation AUC differences . . . 51
3.2. AUCs of experiment 1 . . . 51
3.3. Color codings for network variants . . . 52
3.4. Mean vs max pooling . . . 52
3.5. AUCs of experiment 2 . . . 53
3.6. AUCs of experiment 2, multiple predictions of each sample . . . 53
3.7. AUC variant-boxplots per run . . . 54
3.8. AUC and loss learning paths . . . 55
3.9. AUC validation and test performances . . . 55
3.10. AUC for different numbers of layers . . . 55
3.11. AUC ranges for multiple predictions of each sample . . . 57

1.1. Gating is a decision tree . . . 67
1.2. SHAP: Linear network . . . 73
1.3. SHAP: Linear network plus activation . . . 73

3.1. Binary classification simulation . . . 77
3.2. Multiclass classification simulation . . . 78
3.3. Gate finding concept . . . 79
3.4. Pooled cell features, binary classification . . . 81
3.5. SHAP beeswarm plot . . . 82

3.6. SHAP value accumulation 83
3.7. Cell feature decision trees for binary classification 86
3.8. Cell feature 2d histogram 87
3.9. Binary node gatings, exemplary 88

3.1. Binary classification simulation 101
3.2. Binary simulation: ROC curves 102
3.3. Binary simulation: SHAP scores 102
3.4. Binary simulation: Cell-feature trees 103
3.5. Binary simulation: Counts from applied gatings 103
3.6. Multiclass classification simulation 104
3.7. Multiclass simulation subsampled classes 105
3.8. Multiclass simulation, true and subsampled counts of cells 105
3.9. Multiclass simulation: ROC curves 107
3.10. Multiclass simulation: Predicted probabilities per class and sample 107
3.11. Multiclass simulation: SHAP values 108
3.12. Multiclass simulation: Cell-feature trees 109
3.13. Multiclass simulation: Counts from applied gatings 109
3.14. Fasting: ROC curves 111
3.15. Fasting: SHAP values 111
3.16. Fasting: Cell-feature trees 112
3.17. Fasting: Counts from applied gatings 113
3.18. Fasting: Repeated gatings graph 114
3.19. Fasting: Most prevalent gating counts 114

A 1. Flow cytometer electronics, analog vs digital 141
B 1. Cytometry transformations 143
C 1. CCC performance: Experiment 1, all parameter effects on performance 144
D 1. CCC performance: Experiment 2, all parameter and network architecture effects on performance 145
E 1. CCC performance: Network parameter + architecture on AUC-range 146
F 1. Multiclass simulation: Gatings and trees for all classes 148
G 1. Fasting: Repeated gatings full graph 150

List of Tables

1.1. Shapley example 68

3.1. SHAP-based gating directions 82
3.2. Gating direction nomenclature 82

2.1. Real data fluorochromes 99

3.1. Binary simulation: Gatings 103
3.2. Multiclass simulation: Gatings 108
3.3. Fasting: Gatings 112

Part I.

Cell cloud classification

1. Introduction

1.1. Aim of CCC

The aim of cell cloud classification (CCC) is to enable predictive modelling using cytometry data from biological samples.

Cytometry characterises cells in a biological sample by cell-properties like size, complexity or surface markers[5]. A measured cell-property is called a cell-parameter, which defines each cell by numerical values suitable for further analysis. Therefore, the resulting data of a biological sample is a matrix with a sample-specific number of unordered cells as rows and a defined number of parameters as columns.

Predictive modelling in cytometry assigns a class label per biological sample based on this data matrix. After the cells of an experiment are unordered, the class of the biological sample can only be defined through the presence and quantity of cells but not through an assessment of which cell was measured first or last.

Therefore, the classical way to apply predictive modelling on cytometry data first defines cell subpopulations leveraging similarities in the cell-parameters. Then each cell subpopulation is quantified for each biological sample resulting in a composition vector per biological sample. Finally, this composition vector can then be used to find differences between classes of biological samples.

In contrast, recent advances[2,3,6] use the data matrix directly. The advantage here is that defining the cell subpopulations and using their quantities in an additional predictive algorithm are not separated anymore. Neural networks are able to use the data matrix directly[2,3]. However, there is no ready-to-use package to apply neural networks on cytometry data.

In the following introduction, I will explain how to do predictive modelling with cytometry data and neural networks. In the results, I propose a novel way to modularise neural networks suitable for cytometry data: The *FeatureLearner*, *Pooler*, *Predictor* (FPP) concept. Finally, I will describe CCC, a python package for predictive modelling on cytometry samples using neural networks based on the FPP concept.

1.2. Predictive modelling

Predictive modelling uses measured biological samples to predict their belonging class. This class enables clinical patient decisions when the class is known to respond to a certain treatment. Any clinical decision needs experienced evaluation of the patient information. A patient's information can be questionnaires, expert assessment or measurable properties of a biological sample like blood, urine or tissue. Classically, experienced evaluation means: A doctor with prior knowledge from medical school and the experience from previous cases decides on further

steps either implicitly or explicitly classifying the patient. Similarly, predictive modelling uses this very same patient information, prior knowledge by defining an initial model, and experience by training the model with data from previous cases to predict the class

Mathematically speaking, predictive modelling is a function $f : \mathbb{R}^p \to C$ which maps p values $x \in \mathbb{R}^p$ to a class label $y \in C$. The task is now to find this function f such that it predicts the classes of your samples. All samples together result in the mathematical problem assigning the class labels to all N measured samples. The data for this is then a matrix with N rows and p features: $X \in \mathbb{R}^{N \times p}$.

In cytometry, it is a bit different: Cytometry characterises individual cells of a biological sample, and predictive modelling gives a class for the whole sample of many cells. The cells are characterised by size, complexity or surface markers[5]. A whole measured sample results in a matrix with an unordered number of cells as rows and a defined number of cell-parameters as columns.

There are two major obstacles when analysing cytometry data: First, each biological sample $i \in \mathbb{N}$ has a different amount of cells $n_i \in \mathbb{N}$. Second, the cells are not ordered.

Now, mathematically, predictive modelling with cytometry data becomes a bit more complex:

$$f : \mathbb{N}^{\mathbb{R}^p} \to C \tag{1.1}$$

A biological sample of cells becomes a matrix $X \in \mathbb{R}^{n_i \times p}$: Each sample i might have a different number of cells n_i with p measured cell-parameters each. These cell-parameters technically span a p dimensional space. Due to the unordered nature of cells, reordering (permuting) the rows of X does not change the contained information of the sample.

There are three major concepts to solve the two obstacles and enable predictive models:

1. Divide the p dimensional space into parts and summarise the cells in each part.

2. Identify cell subpopulations and summarise the cells in each cell subpopulation.

3. Use the cells directly but incorporate their unordered nature and the different number of cells.

In this thesis, I will present and examine an algorithm from the third concept.

You can summarise cells by just counting them or based on statistical measures like mean, median, variance, interquartile range or by fitting bimodal distributions[7]. Additionally, when hierarchical structures like manual gates were established, it is common to report the absolute counts of a cell subpopulation *and* its percentage inside the parent cell subpopulation[5]. For example, B-cells cells can be differentiated into activated and non-activated B-cells. If there are 1000 cells in total, 100 B-cells, 70 activated and 30 non-activated B-cells, the reported percentage for activated is 70% and 30% for non-activated B-cells.

Apart from these three concepts, a recent approach[8] solves a special problem outside the three mentioned concepts: With a given gating scheme as a starting

point, optimise the rectangular gate positions simultaneously with the sample classes based on cell proportions inside each gate.

For better readability, in the following, each new algorithm is introduced with (C_{x-y}): x is the concept number, and y is the algorithm number inside that concept.

Concept 1: Partitioning Divide your p dimensional cell-parameter-space into parts and summarise the cells in each part, see a conceptual visualisation in Figure 1.1. Then apply any predictive model using the summarised parts as features.

For example, you measured two cell-parameters p_1 and p_2 of a sample with 1000 cells. Cut the range of each cell-parameter into nine equally sized pieces. This results in a grid of 9×9 parts. Each part gets a name: $part_{1,1}, part_{1,2}, \ldots, part_{1,9}, part_{2,1}, \ldots, part_{9,9}$. Each of these 81 parts should now hold the number of cells inside that part, and can be used as features for predictive modelling.

This concept is realised in ($C_{2-0\ldots2}$) 2DhistSVM, DREAM-A and fivebyfive[7]. (C_{1-3}) The idea as described in the example was extended by probability binning[9,10] where each part's size is defined such that it contains the same number of events. (C_{1-4}) Cydar[11] counts cells in a hypersphere around positions of randomly selected cells. They then statistically test for differences between conditions with edgeR[12], accounting for overlapping hyperspheres.

Concept 2: Subpopulation identification There are two ways to identify cell subpopulations: Either you use classical sequential gating, or you use unsupervised cell subpopulation identification. Classically, cytometry data is analysed by sequential gating: Sequentially choose cutoffs resulting in a hierarchy to define cell subpopulations. Unsupervised cell subpopulation identification uses algorithms to identify cell subpopulations with similar properties.

To get a defined number of values as features for predictive modelling, the identified cell subpopulations can be summarised. The summarised value of each cell subpopulation is then a useable feature.

For unsupervised cell subpopulation identification, there are two possibilities, shown in Figure 1.1 (B) and (C): Either cluster each sample by itself, followed by meta-clustering to merge each sample's clustering results. Or use all cells from all samples and cluster them together. Note that the data can be over-clustered ("high-resolution clustering"[13]): Extract way more clusters than actually present in the data to prevent merging rare cell subpopulations into big clusters.

Extensive reviews exist[7,13,14], but I will give a chronological overview for the algorithms: (C_{2-1}) One early approach[15] used clustering with SamSPECTRAL[16] followed by FeaLect[17], a feature selection technique similar to Bolasso[18]. (C_{2-2}) SPADE-clustering[19], followed by a combination of earth mover's distance (EMD)[20] and nearest neighbour classification, was published in 2012[21]. (C_{2-3}) flowType with RchyOptimyx[22–24] which clusters with flowMeans[25] and applies univariate (cluster-wise) hypothesis tests with multiple testing correction. (C_{2-4}) RchyOptimyx[23] uses the list of univariate tested clusters to distil the identified cell subpopulations in their simplest possible gating form. (C_{2-5}) Citrus[26] uses hierarchical clustering followed by regularised linear models. (C_{2-6}) FloReMi[27] extended flowType[22,24] to survival data. (C_{2-7}) Competitive SWIFT[28] extends SWIFT[29] with competing cell subpopulation templates per class. (C_{2-8}) MetaCyto[30] goes the other way around

by finding all cell clusters, matching them across samples and identifying the effect size of sample-wise factors. (C_{2-9}) MASC[31] also needs clustered cells but then uses mixed-effect models to model the affiliation of each cell to its cluster. MASC models with and without the class-covariate of a sample in addition to batch-effects and clinical parameters. Each cluster's class coefficient informs where the class-covariate significantly improved the model fit, indicating the cluster-membership of a cell is dependent on it. ($C_{2-9\dots12}$) CyTOF workflow[32] and flowGraph[33] are similar approaches, as well as MIMOSA[34] and COMPASS[35] who use a Bayesian hierarchical framework. (C_{2-13}) diffcyt[13] clusters with FlowSOM[36] and uses edgeR[12], limma[37] or voom[38] for differential analysis of all clusters between classes. They additionally differ between cell type markers, which are used for clustering, and cell state markers, to test differential states within clusters. (C_{2-14}) VoPo[39] uses unsupervised clustering per sample as a starting point, followed by multiple independently calculated meta-clusterings. Finally, an unsupervised Laplacian score based feature selection algorithm[40] finds the most useful meta-clusters, which are the input to a random forest for predictive modelling.

Concept 3: Learn directly on cells The third concept to learn directly on cells does not summarise cells before predictive modelling.
(C_{3-1}) CellCnn[2] uses a neural network approach with one single layer. This single layer represents a linear combination of the cell-parameters per cell, followed by a pooling layer like the mean or maximum over all cells of a sample, concluded with another single layer to use the pooled values predicting the class.
(C_{3-2}) CytoDx[6] regards each cell as a small instance of a sample. The effect of each cell is estimated by maximising a cell-parameter-regularised model over all cells in all samples. The final predictor for one single sample is then the mean over all estimated cell effects of that sample.
(C_{3-3}) DeepLearningCyTOF[3] use the same approach as CellCnn but extend it in three ways: First, they propose deep convolutional neural networks, so multiple instead of a single layer. Second, the possibility of including further patient information like sex, age, or batch of the patient's sample. Third, they propose a permutation-based way to describe the cell subpopulations responsible for the predictions.

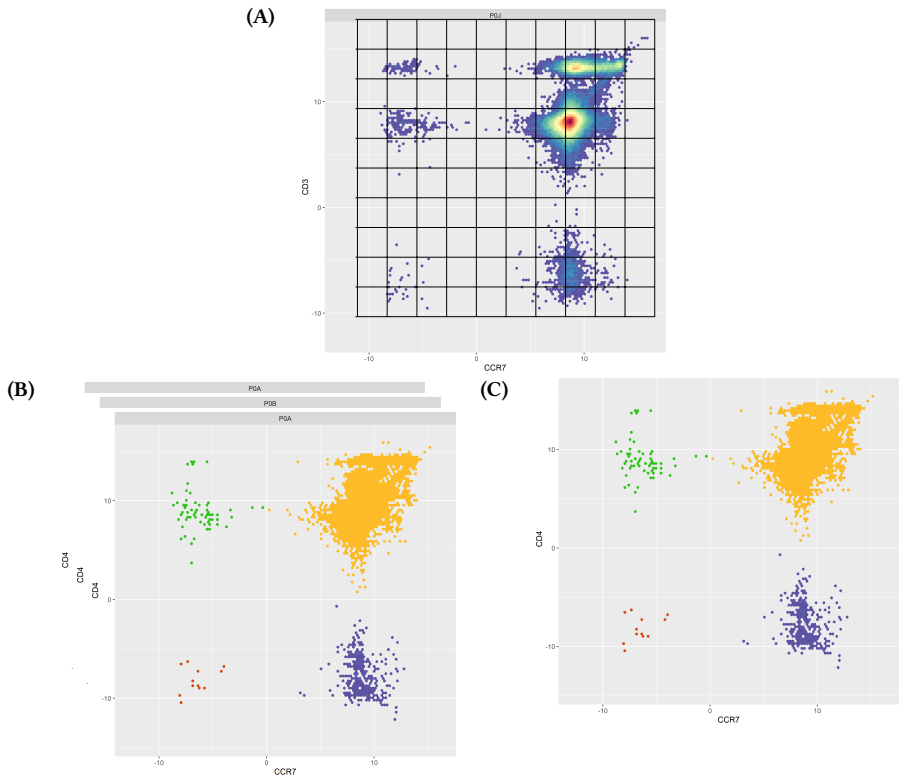


Figure 1.1.: Partitioning and clustering symbolic images. CD3 and CCR7 of one sample are always shown. (A) 2D-histogram of counts with one possible partition where each marker is divided into ten equally spaced parts which result in 10×10 parts. (B) shows the clustering results. Each sample was clustered by itself and must be merged. (C) shows the clustering results. All cells from all samples were clustered together; only one sample is shown.

1.3. What is cytometry data?

Cytometry characterises individual cells of a biological sample. A biological sample contains millions of cells, and each of these cells has a certain size, certain density, specific weight, some enzymatic activity and many more cell properties. Anything that defines a cell is a cell-property, but only some of those cell-properties are measurable. These cell-properties can then be used to characterise a cell. In the following, I use cell-parameter as a measured cell-property quantified through a real number.

Mathematically speaking, the result of a cytometry experiment becomes a matrix $X \in \mathbb{R}^{n \times p}$ with n cells and p cell-parameters.

There are two notable peculiarities with this kind of data: First, the number of cells varies from sample to sample. Second, the order of cells is irrelevant. A variable number of cells is ubiquitous but no big problem: We can downsample or stop measuring the cells to get a fixed amount. The unordered nature of cells is more interesting: The cells are measured in no particular order, so all information about a biological sample is contained in the presence of cells or the quantity of cell subpopulations!

1.3.1. Flow cytometry

Flow cytometry is the measurement of cell-properties in a fluid phase. The cells might come from blood but can also be dissociated solid tissue cells[41]. Flow cy-

tometry enables measuring and visualising anatomic, morphologic, and functional cell-properties[42]. It enables rapid characterisation of thousands to millions of cells with over 40 cell-parameters per cell.

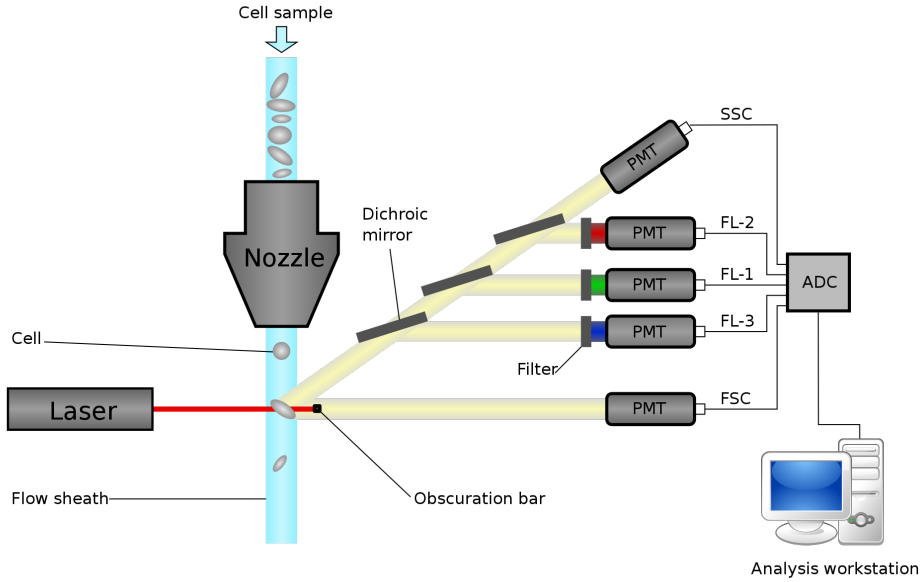


Figure 1.2.: Fluid-based flow cytometer schematic[43]. The nozzle focuses cells in suspension from a sample by creating a laminar flow. One or more lasers excite fluorochrome-labelled antibodies on cells[44]. Photodiodes then detect 1) the forward light scatter informing about cell size and refraction index, 2) (orthogonal) side scatter informing about granularity and 3) the fluorescence signals informing about their connected antibodies[41]. An analog-to-digital converter (ADC) converts the analogue signals into digital numbers to be analysed.

Flow cytometry consists of three key components[44]: The flow cytometer instrument, antibodies and fluorochromes.

1. The flow cytometer instrument. A schematic of the instrument is shown in Figure 1.2. A nozzle focuses cells in suspension from a sample by creating a laminar flow. This laminar flow is enabled by a fast inner fluid stream (of cells) and a slow outer fluid stream (sheath). Both streams run in parallel, guiding cells through the laser(s). One or more lasers excite fluorochrome-labelled antibodies on cells[44]. Photodiodes then detect 1) the forward light scatter informing about cell size and refraction index, 2) (orthogonal) side scatter informing about granularity and 3) the fluorescence signals informing about their connected antibodies[41]. The dichroic mirrors split the different light wavelengths to measure them separately. An analog-to-digital converter (ADC) converts the analogue signals into digital numbers to be analysed.
2. Antibodies. Antibodies recognise a unique molecule, the antigen, at a specific binding site of the antibody. Thus, an antibody binds to a cell if it expresses the antigen on the cell surface.
3. Fluorochromes. A fluorochrome is a fluorescent dye that can be used to stain biological material. You conjugate a fluorochrome to the antibody, mix it with your biological material and wash off unbound antibodies. Now, the flow cytometer can measure the fluorescence from each cell which corresponds to the number of dyed antibodies bound to this cell.

A cluster of differentiation (CD) is assigned to a cluster of monoclonal antibodies

(mAbs) which recognise the same cell surface molecule. MAbs are cloned antibodies binding to only one specific antigen. Different monoclonal antibodies can react with the same cell surface molecule. The Human Leukocyte Differentiation Antigens (HLDA) Workshops defined the CD nomenclature: A non-descriptive CD number is assigned to those mAbs that recognise the same cell surface molecule[45]. CD3 for a protein complex on T-cells and designates both this protein complex as well as the related mAbs.

In flow cytometry, you measure events instead of cells. In flow cytometry, an event is any particle detected by the flow cytometer. This particle, however, can be debris, multiple cells at a time (called doublet if there are two cells measured a time) or single cells. Preprocessing flow cytometry data removes all particles which are not single cells. Therefore I will refer to measured cells as the result of a flow cytometry experiment except during the explanation of preprocessing.

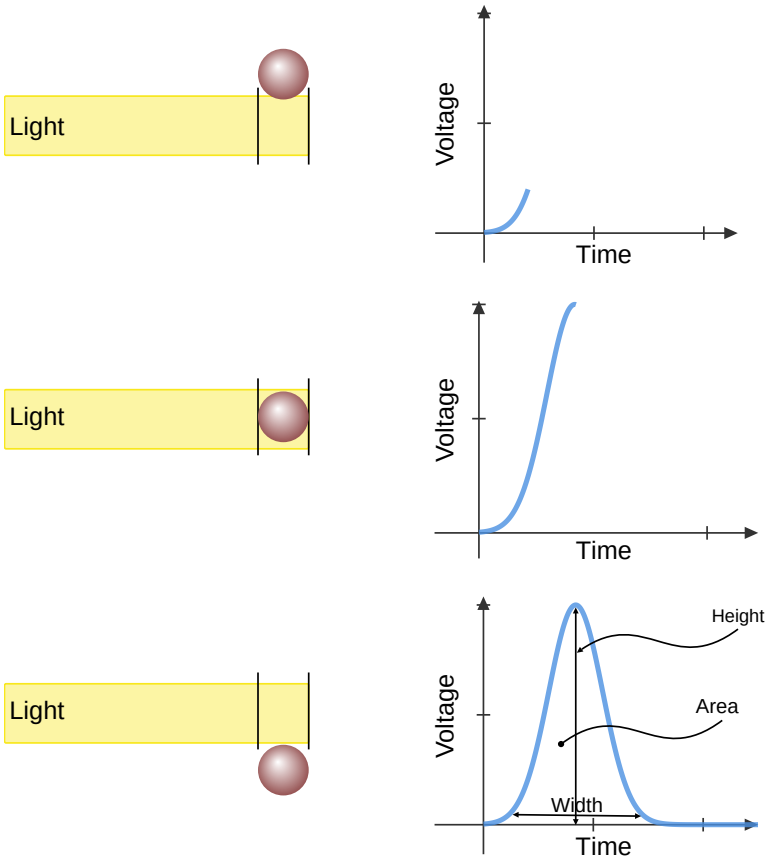


Figure 1.3.: Schema of the measurement of a single particle[46]. A particle traverses the light beam, and the optical sensors detect a voltage depending on the particle's position. When the particle is centred in the light beam, illumination is at its maximum. Consequently, the intensity of the scatter or fluorescence signal reaches its peak. Note that "particle" is most times a single cell, but multiple cells may stick together. Thus, the entire generation (from baseline to baseline voltage) is termed an "event". The height of anevent refers to the maximum intensity of the pulse. The width reflects the time the particle spent in the light beam. The area is the integral of intensity over time. Additionally, an intensity threshold excludes weak pulses of dust or debris.

1.4. How to analyse cytometry data?

In this section, I will explain the usual workflow when analysing cytometry data.

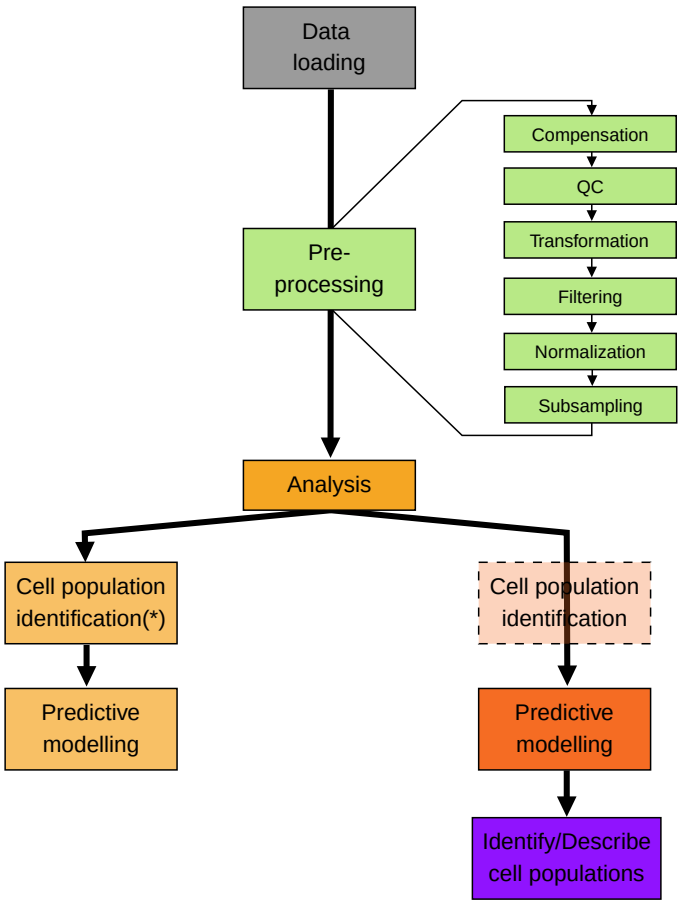


Figure 1.4.: Flow cytometry analysis workflow: After a successful flow cytometry experiment, the data are stored in FCS files which need to be loaded. After that, preprocessing prepares the data for further analysis. This can include compensation, cell-parameter transformation, filtering of irrelevant cells, normalisation and subsampling of cells. Then the actual analysis of the data starts where two broad possibilities exist. Shown in the left branch and most commonly used: Cell subpopulations are identified via manual gating, automated gating or clustering algorithms. Then, predictive modelling uses the summarised cells per biological sample and cell subpopulation to predict the biological sample's class In contrast, shown in the right branch: Directly use the preprocessed cells for predictive modelling. Note that cell subpopulation identification is here shown dashed branch as some algorithms include automated gating or clustering. Contrary to the left branch, where the cell subpopulations are known and identified before using them in predictive models, in the right branch, the relevant cell subpopulations must be identified afterwards. For more information on cell subpopulation identification see Figure 1.9.

(Flow) cytometry analysis can typically answer three questions:

1. How many and which cell subpopulations are present?
2. How many cells are in a specific cell subpopulation?
3. Which cell subpopulation discriminates classes of biological samples?

The last question includes the previous two: To be able to find discriminative cell subpopulationss, you must be able to define and count those subpopulations per biological sample. Predictive modelling is the following application of last question's result: Predict the class of a new biological sample based on discriminative cell subpopulationss.

There are two workflows to analyse cytometry data: Either in two steps, describing all cell subpopulations per biological sample, followed by predictive modelling - shown in the left branch of Figure 1.4. Or in a "single" step: Let the models use the cytometry data directly to find and predict the class of a biological sample - shown in the right branch of Figure 1.4.

Both workflows start by loading and preprocessing the data. Loading the commonly used data, stored in the `.fcs` data format, is described in the appendix, section A. Preprocessing includes: Compensation for fluorescence signal spillover (see subsubsection 1.4.1.1). Transformation of the measured values, some variant of applying the logarithm, as cell-parameter values are log-normal distributed (see subsubsection 1.4.1.2). Filtering of irrelevant events, note specifically "event" in contrast to "cell", so debris, doublets, dead cells are excluded, even when subsequent automated analyses are planned. Finally, normalisation might be necessary to correct for batch-effects introduced through different places, experiment times, instruments, reagents or handlers.

After that, the two-step workflow, shown in the left branch, describes all cell subpopulations per biological sample by identifying and summarising the cell subpopulations. Identify the cell subpopulations: Classically, they are defined by manual gating, but also automated gating or clustering algorithms can be used. Summarise the cells of each cell subpopulation per. Here it is possible to use the total number of cells in a cell subpopulation, their relative frequency in the total dataset or also their relative frequency regarding their parent cell subpopulation: E.g. how many percent of CD3+ cells, the parent gate, are then either CD4-/CD8- (double negative), CD4+/CD8+ (double-positive), CD4+/CD8- or CD4-/CD8+. Also, more complex summary statistics derived from the number and cell-parameter value of the cells are possible, like the mean or variance of a cell-parameter inside a cell subpopulation. Finally, use the cell subpopulations summaries as features of the samples to test for differences between classes or apply predictive modelling.

In contrast to describing cell subpopulations first and extracting the predictive cell subpopulation second, the right branch of algorithms directly classifies biological samples based on single-cell input data. Within this class of algorithms, some include automated gating or clustering internally, hidden from the final applicant, so I dashed the cell subpopulation identification node. Successful model building means that it has predictive power. However, it might be necessary to describe the cell subpopulations responsible for this predictive power. This can be either included in the algorithm or be an additional task. In section 1.2 I already introduced the three concepts of this right branch: Partitioning, cell subpopulation identification and learning directly on cells.

Many packages implementing cytometry workflows exist in Bioconductor (Bioconductor) and on github[47–53]. Some packages also include guided user interfaces (GUI) for analysis[54–63].

1.4.1. Preprocessing

1.4.1.1. Compensation

Multiple cell-properties can be measured with flow cytometry by using multiple fluorochromes. Fluorochromes are fluorescent dyes that can be used to stain an-

tibodies which then bind to the respective surface elements of a cell. By using different fluorochromes for different antibodies, multiple cell-properties can be measured.

However, fluorochromes do have spectral emission overlaps, see Figure 1.5 for two exemplary fluorochromes. Light sources with a specific wavelength are used to excite the fluorochromes, e.g. FITC with blue at 488nm and PE with yellow at 561nm[44]. The actual measurement of light is limited to a specified bandwidth using an optical bandpass filter, denoted by grey-shaded rectangles in the figure. When the emission spectra of fluorochromes overlap, spillover happens: The measured intensity of the bandpass related to one fluorochrome includes not only the light of that fluorochrome but also from all other fluorochromes with emissions in that bandwidth.

To retrieve the actual signal of one fluorochrome by itself, spectral overlap compensation or short just "compensation" has been introduced[64]. Mathematically, spectral overlap can be described as follows:

$$O = KS + A \tag{1.2}$$

where $O \in \mathbb{R}^{p \times n}$ are the observed measurements for p cell-parameters and n cells. $K \in \mathbb{R}^{p \times p}$ is the crossover, spectral overlap or spillover matrix. $S \in \mathbb{R}^{p \times n}$ are the true values per cell-parameter and cell. A is a matrix with n identical columns reflecting an autofluorescence vector $a \in \mathbb{R}^{p}$ which adds a constant value on each cell-parameter for all cells: $A = a \left(1 \ \dots \ 1\right)_n \mathbb{R}^{p \times n}$. K_{ij} denotes the proportion of the i^{th} fluorochrome spilling over to the observed value of the j^{th} fluorochrome.

To extract the true values S for all n events, you solve the previous equation for S. This process is called "to apply compensation":

$$S = K^{-1}(O - A) \tag{1.3}$$

After you already have your observed measurements O, you still need K and A. Note that the compensation-process uses the *compensation matrix* K^{-1} which is the matrix inverse of the spillover matrix K[65].

To obtain the spillover matrix, we need compensation beads or cells. Part of compensation beads must be stained with a single fluorochrome. Another part must be uncoated as a negative control cell subpopulation. The single-colour staining enables the possibility to measure how much spillover from this single fluorochrome spills into the other detectors. The percentage of signal spillover is then stored in the spillover matrix[44].

It became common[66] to disregard the autofluorescence matrix A because the impact on the results was minor. Therefore compensation becomes simpler:

$$S = K^{-1}O \tag{1.4}$$

However, after I investigated inconsistencies in my data, I found that for .fcs-files from Navios systems from Beckman-coulter Life Sciences, autofluorescence was included as follows: If the autofluorescence is not stored in custom keywords, it is defined to a constant value of roundly 0.0309 (for the area measurement). This comes from the fact that the area from the Navios is measured with 20 bits (so

maximum is $2^{20} = 1048576$), and the assumed autofluorescence value is 324. Thus, the default autofluorescence *percentage* is $\frac{324}{2^{20}} \cdot 100 \approx 0.0309$. This number changes for height (stored with 18 bits) and width (stored with 10 bits).

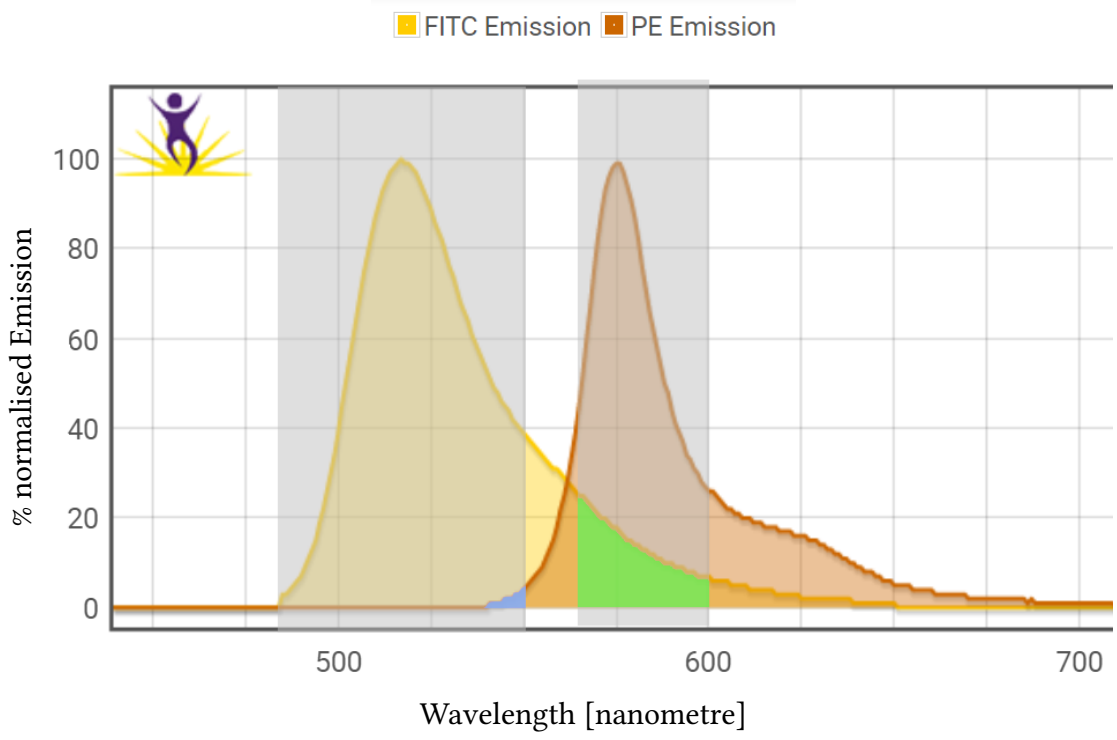


Figure 1.5.: Normalised emission spectrum of fluorochromes FITC and PE. Gray shaded areas: Exemplary wavelengths detected by the sensors. Green area: Additional detected emission from FITC by PE-sensors. Blue area: Additional detected emission from PE by FITC-sensors. These additional emissions are called spillover. Modified from[67]

1.4.1.2. Transformation

In flow cytometry, the values of cell-parameters are approximately normally distributed on a logarithmic scale[68]. Thus for visualisation and further analysis, they are usually log-transformed.

However, compensation can lead to negative values (see subsubsection 1.4.1.1) where the logarithm is undetermined. There are different ways to still transform the data even when negative values are present: Logicle (sometimes called biexponential)[69], HyperLog[70], box-cox[71], linlog, generalised arcsinh (= $\sinh^{-1}(a + bx)$)[72] and flowVS which can automatically estimate suitable configuration parameters for the arcsinh[73]. More recently, rank transformation was proposed to mitigate batch effects, but they strongly recommend combining it with usual preprocessing steps - in particular compensation and debris removal[6]. See section B for the mathematical description and visualisation of all the transformations.

Selecting the correct transformation and its parameters still remains data-dependent[72,74]. Therefore I decided to stay as close as possible to the approach used in the laboratory I got the data from: Hyperbolic arcsine, $\sinh^{-1}(a + bx)$. Here x are the measured values after compensation and $b \in \mathbb{R}$ is a manually chosen parameter and $a = 0$.

Figure 1.6 shows the values of a sample stained with CD4 and CD8. The sample was properly compensated the subplot show: (A) no transformation (B) log-transformation where all values < 0 are set to zero (C) arcsinh-transformation with "cofactor" 150, the default value in flow cytometry, $= \sinh^{-1}(\frac{x}{150})$ (D) arcsinh-transformation with expert-chosen parameters for CD4 and CD8: $b_{CD4} = \frac{1}{3500}$ and $b_{CD8} = \frac{1}{5000}$.

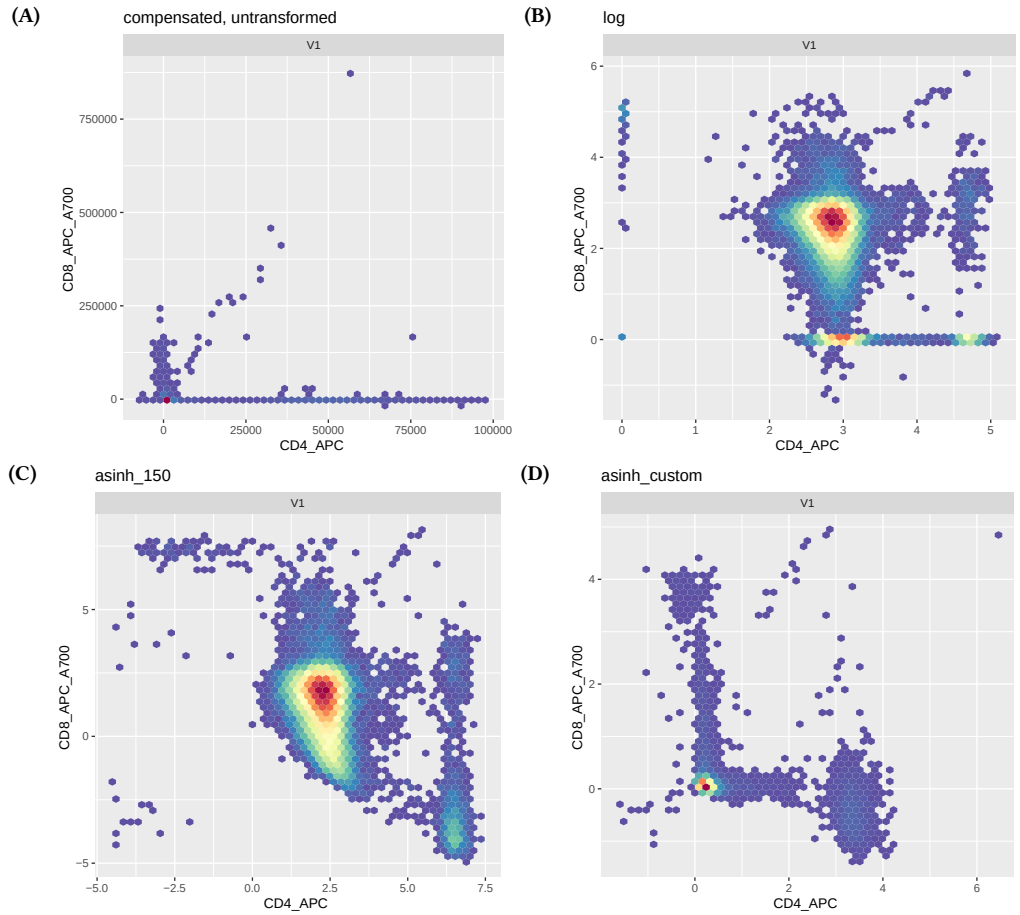


Figure 1.6.: Cytometry transformations. Every plot shows CD4 on the x-axis and CD8 on the y-axis. Plot (A) shows the untransformed, compensated measurements as reference; all other plots show the transformed values. The colour denotes the number of cells at that position where blue is at least one cell. In each plot are 50000 cells. I traverse the plot from top left to bottom right. (A) Raw, compensated data (B) log, negative values set to 0 (C) arcsinh, both CD4 and CD8 are transformed with $\sinh^{-1}$, $a = 0, b = \frac{1}{150}, c = 0$ (D) arcsinh, CD4 and CD8 are transformed with $a = 0$ but $b_{CD4} = \frac{1}{3500}$ and $b_{CD8} = \frac{1}{5000}$ according to expert knowledge from the laboratory

1.4.1.3. Filtering

Any cytometry experiment requires cleaning the events including debris and artefact removal[49]. Doublets, two cells passing the sensor at the same time, dead cells and debris are easily filtered by checking the size and complexity cell-parameters. Event bursts, a sudden spike in observed events as a result of flow rate changes or bubbles, can be identified by plotting the number of acquired cells against the measurement time per cell, and staining artefacts might be removed[5]. A doublet event is a pair of cells measured together, resulting in a larger area but a smaller height of the forward-scatter cell-parameter than single cells.

Dead and damaged cells are excluded because they increase background signal by unspecific antibody-binding and autofluorescence. It is possible to use DNA dyes impermeable to the plasma membrane, so viable cells have no fluorescence. Common dyes to stain DNA are PI, DAPI, 7-AAD or EtBR. Even if suboptimal, light-scatter properties (FSC, SSC) of cells can be used to discriminate from damaged from viable cells[5]. This becomes even more difficult with frozen and consequently thawed sampled, especially to differentiate debris from cells[75].

There are some packages to automatically clean samples from unintended cells, but this analysis is often done through manual gating: flowCut[76], PeacoQC[77], flopR[78] or denoisingCTF[79].

1.4.1.4. Batch correction

Batch correction removes systematic technical variation across batches of biological samples. One single batch can be samples measured at one single day, from a single person, at a specific machine or location, or reagents of a certain age. The experiment-side can assure consistent measurements to some extent: Alignment particles to check instrument precision, reference and calibration particles for instrument sensitivity, compensation standards for multicolour fluorescence analyses and biological control samples depending on your measure of interest[68].

A thorough study showed that fluorescent calibration beads could control some technical variability[80]. Calibration beads are polystyrene particles with defined particle sizes, fluorescence intensities and amount of beads in a given volume.

Computational batch correction is heavily dependent on the data at hand. I used the arcsinh-transformation described in subsubsection 1.4.1.2 which stabilises within-population variance aligning cell subpopulation over samples[73]. There are more sophisticated methods[36,73,81–91] for batch-correction, but my data was consistent after that transformation.

1.4.1.5. Subsampling

Flow cytometry data has a variable number of cells for each measured biological sample. Nevertheless, many algorithms in predictive modelling need an equal number of cells for all biological samples. Therefore I here introduce three subsampling schemes.

For each sample i, there is a matrix of measurements $X^{(i)} \in \mathbb{R}^{n_i \times p}$ with $n_i \in \mathbb{N}$ cells and $p \in \mathbb{N}$ cell-parameters. Row-wise (= cell-wise) subsampling of this matrix results in the final matrix $X' \in \mathbb{R}^{n \times p}$ to analyse.

$$\mathbb{R}^{n_i \times p} \xrightarrow{subsample} \mathbb{R}^{n \times p} \tag{1.5}$$

Random Subsample cells randomly to a fixed number n of cells.

Each cell has the same probability of being included in the final matrix X'. It is straightforward to implement and does not change the underlying distribution of cells. However, rare cell subpopulations will be lost: If you subsample 10000 cells, but your cell subpopulation is only present once in 10^7 cells, this cell subpopulation will be lost.

Density-based Subsample cells density-based, equalising the density over the spanned space of p cell-parameters.

SPADE[19] established the concept: Estimate the local density (LD) for each cell, defined as the number of cells within each cell's neighbourhood. Target density (TD) and outlier density (OD) define which cells are kept. Local densities below OD are discarded, up to TD all cells are taken, and finally, high-density areas are downsampled to a local density of TD. OD and TD are per default defined as 0.01 and 0.05 percentile of the previously calculated local densities per cell of one sample.

$$prob(keepcelli) = \begin{cases} 0 & \text{if } LD_i \leq OD \\ 1 & \text{if } OD < LD_i \leq TD \\ \frac{TD}{LD_i} & \text{if } LD_i > TD \end{cases} \tag{1.6}$$

The local density is calculated by the following steps:

1. Get all cells of one sample
2. Estimate the median minimum distance between a cell and its nearest neighbour, med_min_dist: Pick 2000 cells, calculate the minimum distance from each cell to its neighbour and take the median.
3. The neighbourhood distance is then defined by this estimated median times α:
$$dist\_threshold = med\_min\_dist \cdot \alpha \tag{1.7}$$
4. The local density of cell j of sample i, $X_j^{(i)}$ is then defined by the number of cells in an open ball.
$$LD_j = \left| \left\{ x \in X^{(i)} | d(x, X_j^{(i)}) < dist\_threshold \right\} \right| \tag{1.8}$$

Outlierness-based Subsample cells based on their outlierness.

The outlierness of a cell is proportional to the euclidean distance to other cells. You can draw your cells according to that outlierness: The higher, the more likely you choose that cell. Additionally, you are encouraged to randomly sample a part of the cells and only the other part according to the outlierness.[2] already use the concept of outlierness based on[92].

To calculate the outlierness in a computationally feasible way,[92] propose to draw a specified number of reference cells randomly and calculate the distance of all cells to these drawn. Then the outlierness for cell c with p markers is defined as

$$outlierness_c = [\min_{c' \in \mathbf{C}}(dist(\mathbf{c}, \mathbf{c}'))]^{pow} \tag{1.9}$$

where $\mathbf{C}$ is the set of randomly drawn reference cells from the current sample.

Note that I introduce a power (pow) to this outlierness: The higher this power, the higher the probability of including distant cells.[2] used a different approach (so, $pow = 1$) but used only the top 10% cells based on outlierness when sampling cells according to it. This has a similar impact as increasing pow.

The probability to draw cell c is then $\frac{outlierness_c}{\sum_c^{\text{all cells}} outlierness_c}$.

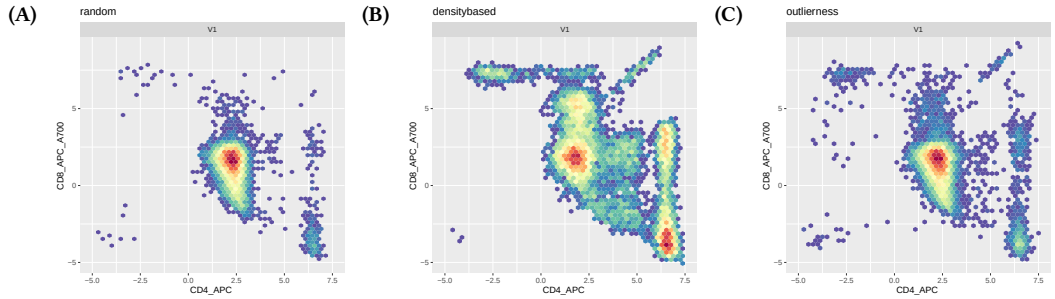


Figure 1.7.: Cell subsampling/downsampling variants. The exemplary sample had originally 552492 cells, subsampled to 10000. Every plot shows all 10000 cells, CD4 on the x-axis and CD8 on the y-axis. (A) A random sample. (B) Density-based[19] subsampling with $OD = 0.01$, TD is found (inside spade) such that the number of cells after downsampling is approximately 10000. Achieved by finding the density such that the sum of cells below that density plus the expected value of cells retained above that density equals approximately 10000. (C) outlierness-based[2,92]. 50% (= 5000 cells) are drawn randomly, the rest based on the calculated outlierness, $pow = 1.5$. To calculate the distance to all cells, 1000 randomly drawn cells reference cells are used.

1.4.2. Cell subpopulation identification

The goal of cell subpopulation identification is to assign a discrete label to each cell. This label can be broad cell types like T-cell or B-cell or fine grain distinction of deep cell subpopulations inside functional cell hierarchies. These labels are assigned utilising similarities in the cell-parameters.

Figure 1.9 shows an overview classifying cell subpopulation identification into manual gating, supervised and unsupervised analysis. Supervised learning needs either pre-gated cells or gating templates. Unsupervised learning finds cell subpopulations without previous data. There are some excellent reviews and benchmarks[5,7,14,32,74,93–110].

1.4.2.1. Sequential manual gating

Sequential manual gating uses manually chosen single values, rectangles, circles or polygons as cutoffs to group cells into cell subpopulations. For this, you visually explore one or two-dimensional cell-parameter-plots and place the gates properly. Each of the cell subpopulations can then again be gated further to get even finer resolution cell subpopulations. These cell-parameters depend on the cell subpopulation you want to "gate". Leucocytes express CD45[111], which is commonly used as cell-parameter, so gating CD45+ cells selects leucocytes. In the exemplary gating in Figure 1.8, CD3+ cells (T-cells) are gated inside the leucocytes, and the CD4 and CD8 expression of the cells are shown in a 2D-histogram. An often criticised problem, especially in the computational analysis community, is that manual gating is subjective, time-consuming, hard to reproduce and difficult with high-dimensional data (10 is already high-dimensional)[5,74,95,104,106].

There are many coding environments to help gating cells manually and efficiently with many samples[48,49,51–53,56,60,63,112–120]. However, subjectivity and reproducibility are still problematic: Measuring in two places or even only on two different machines is rarely directly comparable because of shifts in the measured values.

1.4.2.2. Supervised cell population identification

A natural extension for manual gating is to take samples with previously labelled cells and apply the defined rules to new samples. Applying a defined gating to

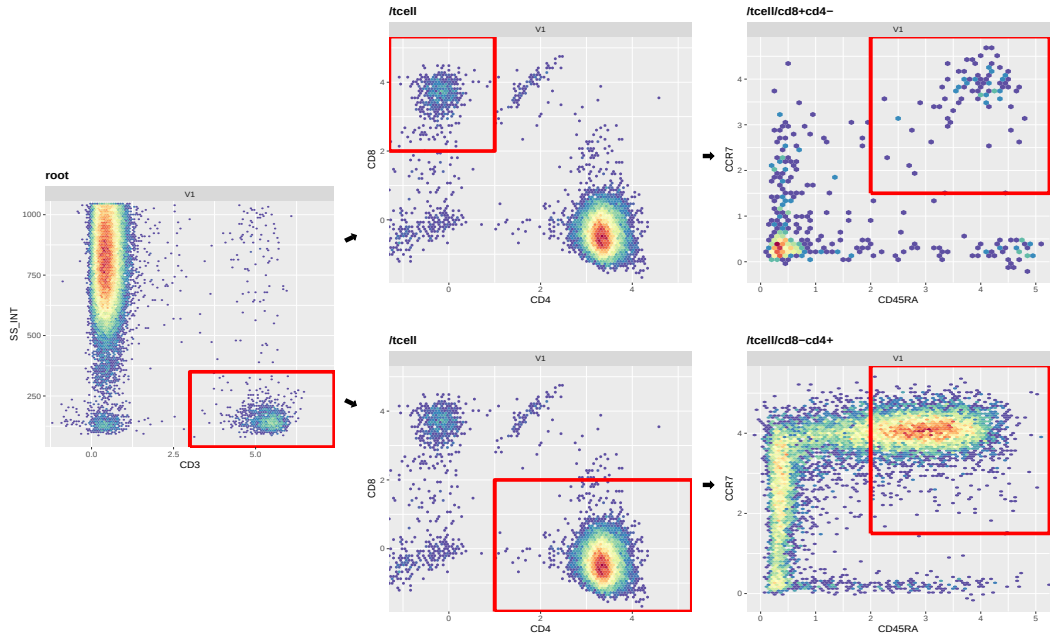


Figure 1.8.: The exemplary gating starts with a pre-gated sample of CD45+ singlets (pre-gates not shown). In the leftmost picture, only CD3+ cells reflecting T-cells are chosen. In the upper row, CD4- and CD8+ cells of the previously selected CD3+ cells are chosen and finally CD45RA+/CCR7+ (naive) cells inside the rectangle. In the lower row, CD4+ and CD8- cells of the first selected CD3+ cells are chosen and finally CD45RA+/CCR7+ (naive) cells inside the rectangle.

new samples is possible, but there are also impressive advances in algorithmic approaches. Supervised machine learning algorithms rely on annotated datasets, so you need either labelled training cells or a predefined gating template to be used to train the algorithm. After the learning phase, the algorithms can predict the type or label of new, unlabelled cells.

Supervised cell subpopulation identification identification can be split into two types:

Cell-parameter based Use cell-parameter values to predict each cell's label directly. Publications applying this principle are a support vector machine approach[121], DeepCyTOF[122], flowLearn[123], a linear discriminant analysis (LDA) approach[124] and optimalFlow[125]. Apart from these algorithms, explicitly built for cytometric data, most multiclass classification algorithms like random forest, decision tree, linear classification, naive Bayes classifier, or neural networks can be used as the setting is always a given matrix of data with n cells and p features ($X \in \mathbb{R}^{n \times p}$) related to some label vector $y \in \mathbb{N}^n$.

Template based Use gating templates and automatically find the best gating strategy based on the cell-parameter values of all cells. This approach is applied in OpenCyto[114], flowDensity[126], ACDC[127] and a bayesian tree variant of ACDC[128]. For OpenCyto[114] and flowDensity[126] a gating template is a sequence of 2D gates defined by the markers and their position (positive or negative). For example, the gating template CD45 and CD3 combined with positive and positive (CD45+, CD3+) yields T-cells based on optimised cutoffs. ACDC[127] and its variant[128] need a marker table similar to the gating template: Also defined by the markers and their position. Contrary, the position can be 1 (positive,present, +), −1 (negative, absent, -) or

0 (don't consider). ACDC[127] creates landmarks based on a scoring scheme and community detection[129] followed by semi-supervised classification. The variation of it[128] builds a tree-structured underlying model and solves the problem of learning the gating trees in a Bayesian framework.

1.4.2.3. Unsupervised cell population identification

Unsupervised cell subpopulation identification identifies cell subpopulations without any prior knowledge but based on cell-parameter similarities: You are clustering cells. After assigning each cell to a cell subpopulation, these cell subpopulations need to be biologically described in their composition. After cytometry data can be high-dimensional, dimensionality reduction is a commonly used approach to prepare data for further unsupervised analysis.

Clustering can be performed either per sample, each possibly containing a variable number of cells, or on all cells from all samples at once[101]. There are many variants of clustering[130] and extensive reviews are already present[7,14,32,98,99,101,102,104–106,131]. Some papers compare algorithms quantitatively in addition to a pure review: Liu et al.[106], with a complete collection of algorithms, using one sample of peripheral blood mononuclear cells publicly available from cytobank[132]. There is also one review on multiple datasets and with emphasis on rare cell subpopulation detection[101]. The FlowCAP challenges[7,133,134] also compared the performance of many algorithms on multiple datasets.

The most popular methods according to number of citations[106] are ACCENSE[135] and DensVM[136]. ACCENSE first reduces dimensionality with t-SNE[137] followed by detecting density-peak based clusters. DensVM additionally incorporates a support vector machine classifier to predict cells distant from the found peaks.

More recent approaches are GigaSOM, a Julia adaption of FlowSOM[36] applicable to billions of cells, SCHNEL[138] which uses hierarchical stochastic neighbour embedding (HSNE)[139] as the basis for Louvain community detection[140] and a recent finite mixture of skew normal factor analyser approach[141].

Most algorithms are applied per sample. Thus, a follow-up meta-clustering is necessary to match the found clusters across samples. Here one approach is to cluster each sample, followed by clustering the cluster centres[73,142] into meta-clusters. Alternatively, cells from all samples can be pooled and clustered together[143,144]. QFMatch[145] proposes adaptive binning on beforehand clustered cells.

After cell subpopulation identification is finished, it is necessary to describe the resulting clusters[106]. MEM[146] or flowCL[147] can identify feature signatures for each cluster and annotate those with cell type ontogenies. Otherwise, a manual investigation of each cluster to find overexpressed markers is necessary.

1.4.3. Predictive modelling

After data loading, preprocessing and possibly cell subpopulation identification, we are ready to apply predictive modelling to predict the class of a biological sample. I explained predictive modelling and its variants for flow cytometry in section 1.2.

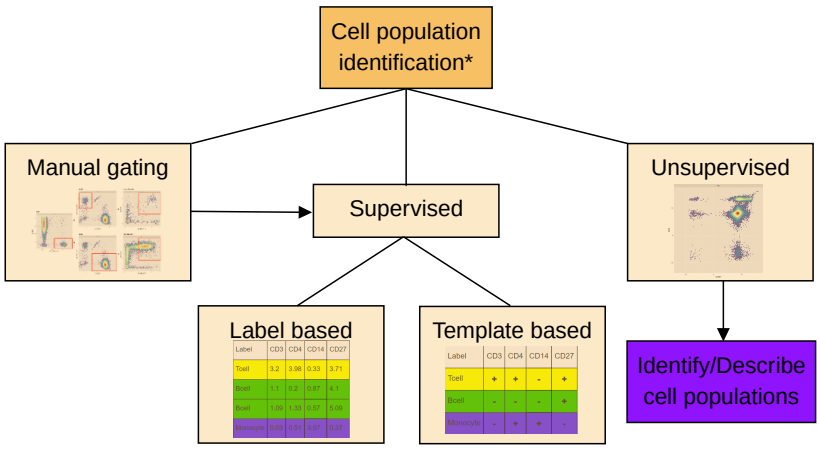


Figure 1.9.: Cell subpopulation identification options. Classical manual gating is the basis for supervised learning. Supervised learning is split into 1) label based learning where the training data consists of cells with their respective label and 2) template-based learning where the "training data" are cell subpopulation templates which are used to find optimal splits to find the positive (+) or negative (-) cell subpopulations. Unsupervised learning identifies and quantifies cell subpopulations without any prior knowledge based on cell-parameter similarities.

1.5. Cytometry data are high dimensional point clouds

In cytometric data, many cells are measured with a defined number of cell-parameters. This can be mathematically described as matrix $X \in \mathbb{R}^{n \times p}$ with n cells and p cell-parameters. Predictive modelling then uses this matrix to predict a class for this complete matrix of all cells.

Point clouds are a special type of data where each point has a defined amount of features but no defined order. Features can be 3D coordinates or colour and temperature of each point. There is no order in the 3D example of point clouds because each point only defines a position in space. This position is fixed, regardless of the order in which points are examined. Mathematically, point clouds can be described as matrix $Z \in \mathbb{R}^{n \times f}$ for n points and f features. Predictive modelling can then be used to classify the object that complete point cloud represents, e.g. a chair or a mug.

Comparing cytometry data X and point clouds Z, we see the only difference is: Cells have cell-parameters, and points have features. Mathematically they are identical; only the number of cell-parameters is usually higher than the number of features. Therefore cytometry data of one biological sample can be described as a high-dimensional point cloud where each point reflects a cell.

Cytometry data have a variable number of acquired cells per biological sample, which is also the case for point clouds. A biological sample has only a limited amount of cells, but this amount varies from sample to sample. Thus there is a natural limit of cells to measure! In contrast, a point cloud has no natural limit: You can set the number of acquired points by defining the resolution you want to achieve and the time you have to measure each object. However, in principle, the point cloud of your first chair might contain a different number of points than your second chair. Therefore the variable number of rows is present in both data types.

The unordered nature of the matrix rows is present for point clouds and cytometry

data. For point clouds, there is a technical necessity to measure each point one at a time. But if we measured the same object at a different time or in a different order of screened surface elements, the object would always stay the same. For cytometry, it is also a technical necessity to measure cell by cell, and it is generally assumed that the order of cells holds no further information about the biological sample. If we were able to measure the same biological sample again, only the cell order would be completely different.

Also, the aim of predictive modelling is similar; just the target is different: In cytometry, we classify biological samples into specific types of patients. In point cloud classification, we classify the presented object. But also here, the mathematical task is the same: Assign a single label to a complete matrix of values. In other words: Assign the entirety of cells or points from one sample to a class All points together could form an aeroplane or a chair; all cells together could describe the tumour stage of a patient. Interestingly, this type of question is essentially multiple instance learning.

Formally[148], multiple instance learning can be described by an instance x in the instance space $\mathbb{X}$, usually $\mathbb{X} \subseteq \mathbb{R}^d$. Each instance is defined by a vector with d real-valued elements. A bag $X \in \mathbb{N}^{\mathbb{X}}$ includes at least one instance, possibly duplicated instances and bags are allowed to overlap. The goal of multiple instance learning is to find a function

$$f : \mathbb{N}^{\mathbb{X}} \to Y \tag{1.10}$$

which finds an association from each bag to an outcome $y \in Y$. When we understand cells or points as instances, a bag of instances represents a cell or point cloud.

In reality, there are no duplicated cells, and overlapping bags cannot exist in cytometry: Each sample has its own unique cells. However, two biological samples can overlap when you think of the measured cell-parameters and the cell subpopulations the reflect: Both measured samples could have activated B-cells defined by specific cell-parameter values, but sample 1 also has non-activated B-cells, defined by other cell-parameter values. Now there are overlapping cells (the activated B-cells) across the two samples (=bags)!

1.6. Supervised learning

The concept of supervised learning aims to find a mapping from input variables, also called features, to an outcome. Given a set of training samples

$$\left\{\mathbf{s}^{(i)}, i = 1, \ldots, n\right\} = \left\{(\mathbf{y}^{(i)}, \mathbf{x}^{(i)}), i = 1, \ldots, n\right\} \tag{1.11}$$

which include the outcome(s) $\mathbf{y}^{(i)}$ for each sample i and the features $\mathbf{x}^{(i)}$ which should be used to model the outcome. These training samples are then used to find that mapping from input $\mathbf{x}$ to output $\mathbf{y}$.

But let's start more rigorously[149]: A **sample** with **sample size** n is defined by the set

$$\left\{\mathbf{s}^{(i)}, i = 1, \ldots, n\right\} \tag{1.12}$$

where $\mathbf{s}$ is a vector of m elements. This sample of size n has been drawn from the

total population of N samples. The basic assumption is that there exists a true, underlying stochastic mechanism: The data generating process (DGP), which could have generated this observed sample. The stochastic properties of this sample are completely described by their joint probability density of all n realisations:

$$f_{\mathbf{s}^{(1)},\mathbf{s}^{(2)},\ldots,\mathbf{s}^{(n)}}(\mathbf{s}^{(1)},\mathbf{s}^{(2)},\ldots,\mathbf{s}^{(n)}) \tag{1.13}$$

When we assume an independently and identically distributed (iid) sample $\left\{\mathbf{s}^{(i)}, i = 1,\ldots,n\right\}$, all marginal probability densities are identical:

$$f_{\mathbf{s}^{(1)},\mathbf{s}^{(2)},\ldots,\mathbf{s}^{(n)}}(\mathbf{s}^{(1)},\mathbf{s}^{(2)},\ldots,\mathbf{s}^{(n)}) = \prod_{i=1}^{n} f_{\mathbf{S}}(\mathbf{s}^{(i)}) \tag{1.14}$$

We are usually only interested in describing the endogenous variables through the exogenous. For this, let

$$\mathbf{s}^{(i)} = \begin{pmatrix}\mathbf{y}^{(i)}\\ \mathbf{x}^{(i)}\\ \mathbf{w}^{(i)}\end{pmatrix} = \begin{cases}\text{to explain, response, dependent, outcome or endogenous variable}\\ \text{explanatory, independent or exogenous variable}\\ \text{variables without effect on } \mathbf{y}^{(i)} \forall i \in \{1,\ldots,N\}\end{cases} \tag{1.15}$$

To distinguish between $\mathbf{w}$ and $\mathbf{x}$ is often not possible but the task of an algorithm. Using this separation of $\mathbf{s}$, we can factorize $f_{\mathbf{S}}(\mathbf{s}^{(i)})$:

$$f_{\mathbf{S}}\left(\mathbf{s}^{(i)}\right) = f_{\mathbf{W}|\mathbf{Y},\mathbf{X}}\left(\mathbf{w}^{(i)} \mid \mathbf{y}^{(i)},\mathbf{x}^{(i)}\right) f_{\mathbf{Y}|\mathbf{X}}\left(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}\right) f_{\mathbf{X}}\left(\mathbf{x}^{(i)}\right) \tag{1.16}$$

We see now that to explain $\mathbf{y}^{(i)}$ given the explanatory variables $\mathbf{x}^{(i)}$ it is sufficient to find the conditional density $f_{\mathbf{Y}|\mathbf{X}}\left(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}\right)$. We want to find a model which resembles this density.

So we try to find a function (or model) $g(\mathbf{s}^{(i)};\Theta)$ with a model-parameter vector $\Theta \in \mathbb{R}^p$ such that the resulting probability density is the same as the probability density of the process, which generated the data in the first place.

$$f_S^{model}\left(\mathbf{s}^{(i)};\Theta_{true}\right) = f_S\left(\mathbf{s}^{(i)}\right) \tag{1.17}$$

If we have a model whose model-parameters allow the above equation to be true, the model is "correctly specified". To find these correct model-parameters is a different question, though!

Generally speaking, we have a model with a set of p model-parameters whose true values we cannot know exactly but would like to find out. Here one approach is to use the maximum likelihood estimation. For this, we need to define the joint likelihood

$$L(\Theta;\mathbf{y},\mathbf{x}) = \prod_{i=1}^{n} f(\mathbf{y}^{(i)}|\mathbf{x}^{(i)},\Theta) = \prod_{i=1}^{n} l(\Theta;\mathbf{y}^{(i)},\mathbf{x}^{(i)}) \tag{1.18}$$

which we want to maximise, so we want to find the model-parameters Θ with the highest likelihood that they generated the observed data. $l(\cdot)$ is the likelihood contribution for each observation i and $f(\cdot)$ is the probability density. This problem

can be rephrased into an optimisation problem of the following form:

$$arg\max_{\Theta} \frac{1}{n} \sum_{i=1}^{n} \log f(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}, \Theta) \tag{1.19}$$

which is the result of the expected value from $\log(L(\Theta; \mathbf{y}, \mathbf{x}))$.

Linear regression, as one example for supervised learning, uses an additional assumption: g is linear in its model-parameters, and only the model-parameters must be estimated

$$g(\mathbf{x}^{(i)}, \beta) = \beta_1 \mathbf{x}_1^{(i)} + \beta_2 \mathbf{x}_2^{(i)} + \cdots + \beta_p \mathbf{x}_p^{(i)} \tag{1.20}$$

where $\beta_j \in \mathbb{R}$ are the linear model-parameters for each of the p elements in $\mathbf{x}$. For a single outcome, this linear model assumes that the data generating process looks as follows:

$$y^{(i)} = \beta_0 + \beta^T \mathbf{x}^{(i)} + \epsilon^{(i)} \tag{1.21}$$

With the additional assumption that $\epsilon^{(i)}$ is $iid \sim \mathcal{N}(0, \sigma^2)$, the probability density becomes

$$f(y \mid \mathbf{x}; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\left(y - g(\mathbf{x}^{(i)}, \Theta)\right)^2} \tag{1.22}$$

where now Equation 1.19 can be solved. Note that solving this analytically becomes impossible for more complex models but must be optimised using gradient descent.

In the use-case of this thesis, I only apply supervised classification. Everything stays the same, just the outcome $y \in Y$ are classes. Thus, the outcome space becomes c discrete values where each value corresponds to one class: $Y = 0, 1, \ldots, c_{classes}$.

1.7. Neural networks

The goal of neural networks is to find some true function $f : \mathcal{X} \to \mathcal{Y}$. The network itself approximates this function, $\hat{f} : \mathcal{X} \to \mathcal{Y}$ which maps an input $\mathbf{x} \in \mathcal{X}$ to some output $\mathbf{y} \in \mathcal{Y}$. For simplicity, I will refer to the approximated function $\hat{f}$ only if necessary. Otherwise, I will use f and assume that $\hat{f} = f$ due to some optimisation algorithm.

The function $\hat{f}$ is actually defined by a model-parameter set Θ. $\Theta := (\Theta_W, \Theta_H)$ includes internal weights of the network Θ_W but also hyperparameters Θ_H like network architecture with its number of nodes, number of layers, activation functions, optimiser choices and many more. In contrast to the hyperparameters Θ_H, weights can be inferred from the data during model training.

1.7.1. Logistic regression leads to neural networks

I here start with logistic regression as it can be seen as a single layer perceptron. Logistic regression is a supervised method to find a linear combination of features to classify a sample into either the “0”-class or “1”-class.

Assume that

$$Pr(Y = 1|X = x) = p(x; \Theta) \tag{1.23}$$
$$Pr(Y = 0|X = x) = 1 - p(x; \Theta) \tag{1.24}$$

for some function p with a set of model-parameters Θ. In logistic regression, all model-parameters can be inferred from the data. For a sequence of n Bernoulli trials $y^{(1)}, \ldots, y^{(n)}$ of a constant probability of success p, the likelihood is

$$\prod_{i=1}^{n} p^{y^{(i)}} (1 - p)^{1-y^{(i)}} \tag{1.25}$$

When we assume n independent and identically distributed samples, the conditional likelihood function can be written as

$$\prod_{i=1}^{n} Pr(Y = y^{(i)}|X = x^{(i)}) = \prod_{i=1}^{n} p(x; \Theta)^{y^{(i)}} (1 - p(x; \Theta))^{1-y^{(i)}} \tag{1.26}$$

Now, the model we are looking for tells us that p is *not* constant for all i but dependent on the values $x^{(i)}$! Thus the first likelihood becomes

$$\prod_{i=1}^{n} (p^{(i)})^{y^{(i)}} (1 - p^{(i)})^{1-y^{(i)}} \tag{1.27}$$

where we introduce $p^{(i)} = p(x^{(i)}; \Theta)$ which claims that $p^{(i)}$ should be the same for identical $x^{(i)}$.

A logistic model (or logistic transformation) $g : X \to Y$ is the log ratio of the probabilities for each class:

$$g(\mathbf{x}) := \log \frac{P(Y = 1|X = x; \Theta)}{P(Y = 0|X = x; \Theta)} \tag{1.28}$$

By solving Equation 1.28 for $P(Y = 1|X = x; \Theta) = p(x; \Theta)$, we get

$$g(\mathbf{x}) = \log \frac{p}{1 - p} \tag{1.29}$$
$$\frac{p}{1 - p} = e^{g(\mathbf{x})} \tag{1.30}$$
$$p = e^{g(\mathbf{x})} - pe^{g(\mathbf{x})} \tag{1.31}$$
$$p = \frac{e^{g(\mathbf{x})}}{1 + e^{g(\mathbf{x})}} \tag{1.32}$$
$$p = \frac{1}{1 + e^{-g(\mathbf{x})}} = \sigma(g(x)) \tag{1.33}$$

when $g(\mathbf{x})$ is linear, this is the classical logistic regression model:

$$g(\mathbf{x}) = \log \frac{P(Y=1|X=x)}{P(Y=0|X=x)} = \beta_0 + \sum_{j=1}^{k} \beta_j x_j \tag{1.34}$$

$$p = \frac{1}{1 - e^{-\left(\beta_0 + \sum_{j=1}^{k} \beta_j x_j\right)}} \tag{1.35}$$

By rewriting that we end up with:

$$p(x; \Theta = \beta) = \frac{1}{1 - e^{-\left(\beta_0 + \sum_{j=1}^{k} \beta_j x_j\right)}} \tag{1.36}$$

Now, this is pretty, but we still need to estimate the model-parameters β somehow. Here we can use maximum likelihood estimation, where the likelihood becomes (for *iid* samples):

$$L\left(\Theta|(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\right) = \prod_{i=1}^{n} p(\mathbf{x}^{(i)})^{y^{(i)}} (1 - p(x^{(i)}))^{1-y^{(i)}} \tag{1.37}$$

resulting in the following negative log-likelihood with Equation 1.30

$$-l\big(\Theta|(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\big) = \tag{1.38}$$

$$= -\sum_{i=1}^{n} \left[y^{(i)} \log p(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log \left(1 - p(x^{(i)})\right)\right] \tag{1.39}$$

$$= -\sum_{i=1}^{n} \left[y^{(i)} \log \left(\frac{p(x^{(i)})}{1 - p(x^{(i)})}\right) + \log\left(1 - p(x^{(i)})\right)\right] \tag{1.40}$$

$$= -\sum_{i=1}^{n} \left[y^{(i)} \log e^{g(\mathbf{x}^{(i)})} + \log\left(1 - \frac{1}{1 + e^{-g(\mathbf{x}^{(i)})}}\right)\right] \tag{1.41}$$

$$= -\sum_{i=1}^{n} \left[y^{(i)} g(\mathbf{x}^{(i)}) + \log\left(\frac{1}{1 + e^{g(\mathbf{x}^{(i)})}}\right))\right] \tag{1.42}$$

$$= \sum_{i=1}^{n} \left[\log\left(1 + e^{g(\mathbf{x}^{(i)})}\right) - y^{(i)} g(\mathbf{x}^{(i)})\right] \tag{1.43}$$

Now we would usually like to solve that by differentiating with respect to each model-parameter β_i, set this to zero, differentiate it again to make sure we found a minimum and use the resulting equations to find an analytical solution - but here no closed form solution exists. Thus we have to approximate it numerically, e.g. using gradient-descent[150]. The derivative however will be useful later, so I show it already here:

$$\frac{\partial l(\Theta)}{\partial \beta_j} = \sum_{i=1}^{n} [y^{(i)} - \sigma(g(x^{(i)}))] x_j^{(i)} \tag{1.44}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$

We can use the chain-rule to easily differentiate this formula using Equation 1.33, Equation 1.34 and Equation 1.39. For simplicity I calculate it for a single sample i

to be able to omit all supersets $^{(i)}$ and I omit all (x) where possible:

$$\frac{\partial l(\Theta)}{\partial \beta_j} = \frac{\partial l(\Theta)}{\partial p} \cdot \frac{\partial p}{\partial g} \cdot \frac{\partial g}{\partial \beta_j} \tag{1.45}$$

$$= \left[\frac{y}{p} + \frac{1-y}{p-1}\right] \cdot \sigma(g(x))[1-\sigma(g(x))] \cdot x_j \tag{1.46}$$

$$= \left[\frac{y}{p} - \frac{1-y}{1-p}\right] \cdot p[1-p] \cdot x_j \tag{1.47}$$

$$= [y-p]\, x_j \tag{1.48}$$

$$= [y-\sigma(g(x))]\, x_j \tag{1.49}$$

Then we can update each model-parameter using gradient descent[150]:

$$\beta_t^{new} = \beta_t^{old} - \eta \frac{\partial l(\beta)}{\partial \beta_t} \tag{1.50}$$

$$= \beta_t^{old} - \eta \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\beta_0 + \sum_{j=1}^{k} \beta_j x_j^{(i)}) \right] x_t^{(i)} \tag{1.51}$$

where $\eta \in \mathbb{R}$ is the step size. All model-parameters are then updated until convergence.

1.7.2. Single layer perceptron

The perceptron[151] with a single output is the easiest variant of a neural network: f maps an input of multiple values $x \in \mathcal{X} \subseteq \mathbb{R}^k$ to a single value $y \in \mathcal{Y} \subseteq \mathbb{R}$. This setting is visualised in Figure 1.10. For simplicity, I will omit the bias in the future, as in principle you can imagine it like an additional input x_0 with the constant value 1 where the weight is equivalent to the previously introduced bias β_0, see Figure 1.11.

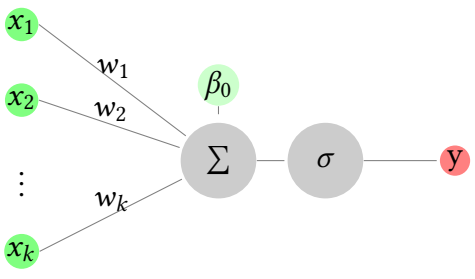


Figure 1.10.: Single layer perceptron for k input features $x_1, \ldots, x_k$, a bias β_0 and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. The input feature values $\mathbf{x}$ are multiplied with the respective weights $\{w_i \in \mathbb{R} | i \in \{1, \ldots, k\}\}$. Then they are summed plus a bias: $z := \beta_0 + \sum_{i=1}^{k} x_i w_i$ and activated to get the final result $y = \sigma(z)$.

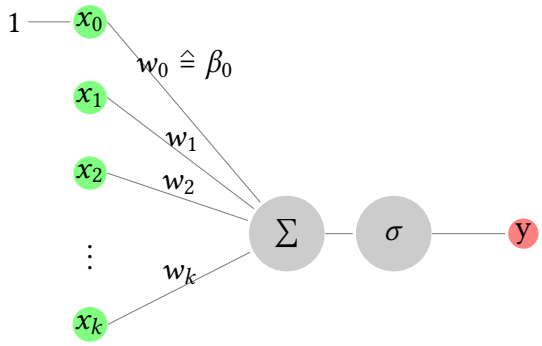


Figure 1.11.: Single layer perceptron for k input features $x_0, x_1, \ldots, x_k$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. The input feature values $\mathbf{x}$ are multiplied with the respective weights $\{w_i \in \mathbb{R} | i \in \{0, \ldots, k\}\}$ and the 'bias-weight' β_0 for the constant $x_0 = 1$. Then they are summed: $z := \sum_{i=0}^{k} x_i w_i$ and activated to get the final result $y = \sigma(z)$.

With this, the function f becomes

$$f : \mathbb{R}^k \to \mathbb{R} \tag{1.52}$$
$$x \mapsto \sigma(\sum_{i=1}^{k} x_i w_i) = \sigma(Wx)$$

with $W \in \mathbb{R}^{1 \times k}$ and $\forall i \in \{1, \ldots, k\} : W_{1i} = w_i$. If σ is a linear transformation between finite vector spaces, there is always a matrix S representing that function. Then we can simplify f:

$$f : \mathbb{R}^k \to \mathbb{R} \tag{1.53}$$
$$x \mapsto \sigma(\sum_{i=1}^{k} x_i w_i) = SWx$$
$$= W^* x$$

This result shows that σ should be non-linear. Otherwise there would be always be a matrix of network weights $W*$ which can replace SW. This becomes even more important with the multi layer perceptron, see subsection 1.7.4.

After we designed the network architecture, we need a way to find the weight matrix W. The model-parameters Θ here consist of the internal, trainable weights $\Theta_W = \{W\}$ and the hyperparameter which activation function $\Theta_H = \{\sigma\}$ to choose.

We can define σ based on the distribution of $\mathcal{Y}$. For this simple network, this actually results in logistic regression[152]. For this, let $\sigma(z) := \frac{1}{1+e^{-z}}$, then the complete network translates to

$$f(x) = \frac{1}{1 + e^{-Wx}} = \frac{1}{1 + e^{-\sum_{i=1}^{k} w_i x_i}} \tag{1.54}$$

which is identical to Equation 1.36 when replacing w_i with β_i (and without intercept).

1.7.3. Training a neural network

After the hyperparameters Θ_H are set, we want to estimate the trainable model-parameters Θ_W. This is usually done by gradient-based optimisation where some

objective function is optimised[153]. When explicitly minimizing it, it is usually called cost, loss or error function:

$$loss(f(x), \hat{f}(x; \Theta)) \tag{1.55}$$

for the true function $f : \mathcal{X} \to \mathcal{Y}$ and its estimate $\hat{f}$. $\hat{f}$ depends on the model-parameter set Θ. Supervised learning claims that your data are pairs of values: $(y^{(i)}, \mathbf{x}^{(i)})$ and I write $\hat{y}^{(i)} = \hat{f}(\mathbf{x}^{(i)}; \Theta)$ as the predicted value.

I rewrite the network structure as shown in Figure 1.12, omitting the bias for the sake of simplicity. The following introduction is roughly based on a recently published book[154]. It becomes clear that a neural network is just a composition of (possibly complex) functions.

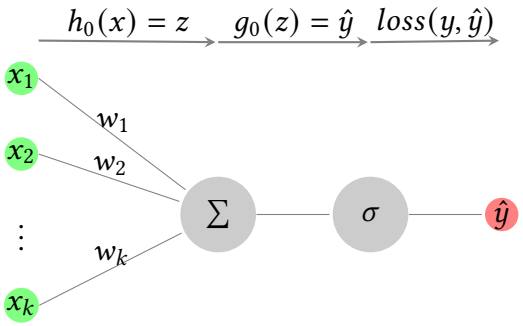


Figure 1.12.: Single layer perceptron for k input features $x_0, x_1, \ldots, x_k$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. The input feature values $\mathbf{x}$ are multiplied with the respective weights $\{w_i \in \mathbb{R} | i \in \{0, \ldots, k\}\}$. Then they are summed: $z := \sum_{i=0}^{k} x_i w_i$ and activated to get the final result $\hat{y} = \sigma(z)$. The complete network is a composition, $\hat{f}(x) = (g_0 \circ h_0)(x)$. Including the loss function, this can be written as composition of three functions: $(loss \circ g_0 \circ h_0)(x)$.

When we choose a simple loss, e.g. based on the loss-function of the squared difference between predicted and true value:

$$E := loss(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^{n} (y - \hat{y})^2 \tag{1.56}$$

We optimise the model-parameters $w_t \in \Theta_W$ by gradient-descent[150] and now have to differentiate this loss with respect to each weight:

$$w_t^{new} = w_t^{old} - \eta \frac{\partial E}{\partial w_t} \tag{1.57}$$

$$= w_t^{old} - \eta \sum_{i=1}^{n} \frac{\partial E^{(i)}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_t} \tag{1.58}$$

With the formulation as a composition, further applying the chain rule results in

$$= w_t^{old} - \eta \sum_{i=1}^{n} \frac{\partial E^{(i)}}{\partial g_0} \frac{\partial g_0}{\partial h_0} \frac{\partial h_0}{\partial w_t} \tag{1.59}$$

Now I go through the calculations step by step for a single instance i (Thus I write

E when I should write $E^{(i)}$):

$$\frac{\partial E}{\partial g_0} = \frac{\partial E}{\partial \hat{y}} = \frac{1}{2} 2(y - \hat{y})(-1) \tag{1.60}$$

$$\frac{\partial g_0}{\partial h_0} = \frac{\partial \sigma(z)}{\partial z} = \sigma(z)[1 - \sigma(z)] \tag{1.61}$$

$$\frac{\partial h_0}{\partial w_t} = x_t \tag{1.62}$$

$$\Rightarrow \qquad = w_t^{old} - \eta \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)}) \cdot (\sigma(z^{(i)})[1 - \sigma(z^{(i)})]) \cdot (x_t) \tag{1.63}$$

which is very similar to Equation 1.51 plus the slope of the sigmoid function σ.

Back-propagation is an algorithm that computes this chain rule in a very efficient order[153]. If we calculate the complete derivative $\frac{\partial E^{(i)}}{\partial w_t}$ for each t, we soon observe that some subexpressions (Equation 1.60 and Equation 1.61) are identical regardless of t. The back-propagation algorithm computes the gradients one layer at a time and iterates backwards from the last layer. This backward iteration enables the efficient calculation of all gradients without redundant calculations of intermediate gradients. In this particular example, even if you have the two functions (and therefore two derivatives) g_0 and h_0, you would still call that a single layer because a layer is usually the aggregation of values (the matrix multiplication) and a subsequent activation function (g_0).

In principle, the only necessity here is that every intermediate function (h_0, g_0) and differentiable. However, the ReLU-function[155] is a prominent example that this is not even strictly necessary as it is not differentiable at 0.

Then the calculated gradients can be used in gradient-descent[150] to find the optimal model-parameters.

1.7.4. Multi layer perceptron

We here use feedforward networks (so, no circles in the neural network), also called multi layer perceptrons (MLP)[153]. As in the single layer perceptron, f maps an input of multiple values $x \in \mathcal{X} \subseteq \mathbb{R}^k$ to a single value $y \in \mathcal{Y} \subseteq \mathbb{R}$. Now the difference is that there is not only one layer but multiple so called hidden layers. This setting is visualised in Figure 1.13. In this figure, each layer is called dense layer with subsequent activation function. Thus, each part of the function can be written as matrix-multiplication with a subsequent (non-linear!) activation, see Figure 1.13.

In Figure 1.13:

$$h_i(z) := W_i z \tag{1.64}$$

where $W_i \in \mathbb{R}^{2\times 2}$, e.g. for the first layer $i = 1$

$$h_1(x) = \begin{pmatrix} w_{11} & w_{13} \\ w_{12} & w_{14} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{1.65}$$

The complete equation becomes

$$\hat{f}(x) = g_2(W_2 g_1(W_1 x)) \tag{1.66}$$

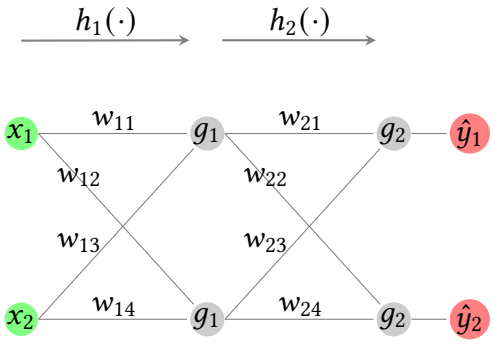


Figure 1.13.: Multi layer perceptron for k input features $x_0, x_1, \ldots, x_k$ with two hidden layers which are defined by the functions h_1 and h_2 where each function is followed by an activation function g_1 or g_2 respectively. The result is a two-dimensional outcome vector $\hat{y}$ which can be written as $\hat{y} = \hat{f}(x) = (g_2 \circ h_2 \circ g_1 \circ h_1)(x)$

And we can optimise it the same way as shown for a single layer by going backwards through the network with gradient descent.

1.7.5. Convolutional neural network

Convolutional neural networks[156] are neural networks that use convolution in at least one of their layers instead of the previously described matrix multiplication[153]. Most times, convolutional neural networks consist of convolutional layers followed by a pooling layer and further dense layers.

The classical setting for convolutional layers resides in 2D images which might consist of 50×30 pixels where each pixel has 3 colours (red, green, blue). To follow the nomenclature of pytorch[157], your (single) input image is of shape $(N \times C_{in} \times H \times W)$ where $N = 1, C_{in} = 3, H = 50$ and $W = 30$. A convolutional kernel, or receptive field[153] is defined by pytorch[157] as:

$$out(N_i, c_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) * input(N_i, k) \tag{1.67}$$

for the 2D cross-correlation operator. You can imagine it as a field of a specific size (the kernel) sliding over the inputs to generate a new feature map. In pytorch's nomenclature, this convolution transforms an input to an output with the following dimensions:

$$\mathbb{R}^{N \times C_{in} \times H_{in} \times W_{in}} \rightarrow \mathbb{R}^{N \times C_{out} \times H_{out} \times W_{out}} \tag{1.68}$$

defined by

$$H_{out} = \lfloor \frac{H_{in} + 2 \cdot padding_H - dilation_H(kernel_H - 1) - 1}{stride_H} + 1 \rfloor \tag{1.69}$$

$$W_{out} = \lfloor \frac{W_{in} + 2 \cdot padding_W - dilation_W(kernel_W - 1) - 1}{stride_W} + 1 \rfloor \tag{1.70}$$

which you can imagine as sliding over the whole image. There are a few possible configurations:

Padding Adds values to the boundaries of your image, thus your effective image size becomes bigger.

Dilation Denotes the spacing between every kernel element.

Stride Denotes the stride between the application of the kernel. When you set this to 1, the kernel is applied to each pixel after each other. If bigger than one, pixels are skipped.

for each configuration, separate settings for height $_H$ and width $_W$ are possible. Importantly, the weights of the kernel are the same, regardless of the position where it is applied. Therefore, gradients have to be aggregated across all applied positions during backpropagation!

We can apply convolution to cytometry data: For an input $X \in \mathbb{R}^{n \times p}$ of n cells and p cell-parameters, the data can be described as $\mathbb{R}^{n \times p}$ matrix for every biological sample. Thus in pytorch-nomenclature we can write this as $(N \times C_{in} \times H_{in} \times W_{in}) = (N \times 1 \times n \times p)$. We choose the kernel as $1 \times p$, then weights are shared across cells but not across cell-parameters, see Figure 1.14. Additionally, this convolution is chained with some nonlinearity, I use the ReLU-function[155]: $relu(x) = max(0, x)$. The resulting matrix after the convolution is $(N \times k \times n \times 1)$ when k different kernels are used. Each of the kernels refers to its own set of weights combining the p cell-parameters. Configure padding as 0, dilation and stride as 1. Then the convolution becomes a special case as it can be replaced with a matrix multiplication. The convolution results in the activation matrix per sample of the form $A \in \mathbb{R}^{n \times k}$:

$$A_{ij} = \sum_{m=1}^{p} X_{im} \cdot w_m^j \tag{1.71}$$

where w_m^j is is the m-th weight of kernel j. We see that this is nothing else than a matrix multiplication:

$$A = XW \tag{1.72}$$

where $W \in \mathbb{R}^{p \times k}$.

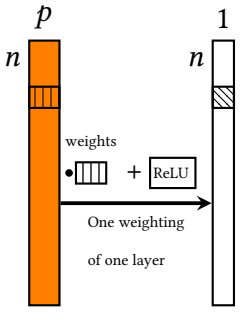


Figure 1.14.: Exemplary convolution over the complete row (=cell) of an input matrix from cytometric data with n cells and p markers.

1.7.6. Batch normalisation

Batch normalisation in neural networks is a method to speed up training the network and allows to be less careful about initialisation[158].

During training a neural network, stochastic gradient descent does not use all samples simultaneously to update the internal weights. Instead, only part of all

training samples are used step by step. The samples used in one part are called a batch of samples, and the number of samples in one batch is called batch size.

Internal covariate shift is a phenomenon[158] where each layer's input values have a different distribution of values from batch to batch. This is a problem because the layers need to adapt continuously to the new distribution.

Batch normalisation is a layer placed before any other layer inside the neural network. For a layer with f input features $x \in \mathbb{R}^f$, batch normalisation can be described mathematically in two steps. First each input feature is normalised separately:

$$x_i' = \frac{x_i - E[x_i]}{\sqrt{Var[x_i]}} \tag{1.73}$$

so the value of the i-th input feature becomes normalised according to its expected value and variance computed over the training data set.

Second, a transformation step is added to restore the representation power of the network:

$$y_i = \gamma_i x_i' + \beta_i \tag{1.74}$$

where γ_i and β_i are trainable model-parameters. y_i is then the output of the batch normalisation layer for the i-th input feature.

The features now have the same distribution, which now offers a promise of faster and less initialisation dependent model training[158].

2. Results

2.1. FPP concept

I propose the *FeatureLearner*, *Pooler*, *Predictor* (FPP) concept, a general layout for neural networks, applicable on point cloud, single cell or more general multiple-instance data (see section 1.5). For clarity I will use solely cell instead of point or instance. One biological sample consists of multiple cells. This data is special as each sample $X \in \mathbb{R}^{n \times p}$ with n cells and p parameters per cell where the rows are unordered.

In Figure 2.1 I visualise the FPP concept which consists of three parts:

1. The *FeatureLearner*, a neural network, uses a matrix $X \in \mathbb{R}^{n \times p}$ with n cells as rows and p parameters as columns as the input. The *FeatureLearner* transforms the input X into the matrix $A \in \mathbb{R}^{m \times k}$ where usually, but not necessarily, $m = n$.

2. The *Pooler* part aggregates all rows from the *FeatureLearner* result matrix A for each column k using $q \in \mathbb{N}_{\geq 1}$ permutation invariant function(s).

3. The *Predictor*, again a neural network, uses these $\mathbb{R}^{q \times k}$ pooled values to predict the class of the biological sample. The classification problem at hand (e.g. binary classification or multiclass classification) defines the form of the output layer of the *Predictor*.

The *FeatureLearner* is a function which finds and identifies predictive cell-features for the classification problem. The resulting cell-features are defined through the internal *FeatureLearner* parameters which describe how the measured cell-parameters are merged.

A symmetric function is per definition permutation-invariant, as a function $f : A^n \to B$ is symmetric for two sets A and B if and only if for all permutations π and $\forall i \in \{1, \ldots, n\}\, x_i \in A$

$$f(x_1, x_2, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}) \tag{2.1}$$

This holds, for example, for the mean, sum, maximum or product of all values x_i.

See Figure 2.2 and Figure 2.3 for exemplary network structures for a binary classification problem.

The *Predictor* itself solves a supervised problem. The $\mathbb{R}^{q \times k}$ matrix from the *Pooler* will be vectorised in one long vector $\mathbb{R}^{q \cdot k}$ as input for the *Predictor*. After the *Pooler* there is no more multi-instance problem: For each sample i, there is a class y and a vector of features $x \in \mathbb{R}^{q \cdot k^+ a}$.

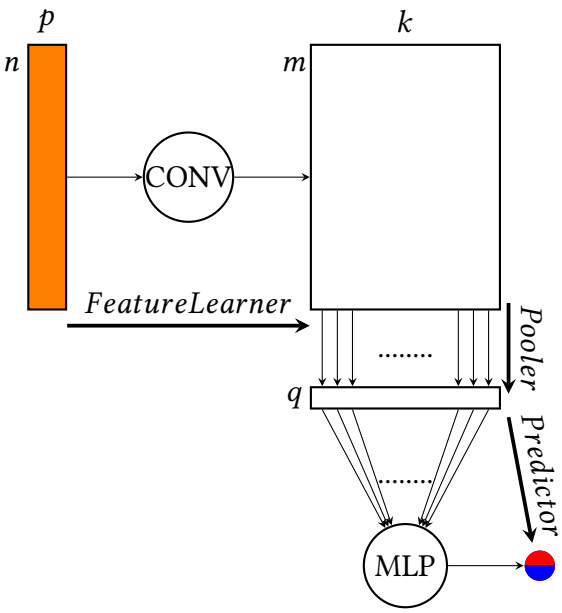


Figure 2.1.: *FeatureLearner*, *Pooler*, *Predictor* (FPP) concept: A general layout for a neural network applicable on single cell data. A matrix $X \in \mathbb{R}^{n \times p}$ with n cells as rows and p cell-parameters as columns is the input to the *FeatureLearner*, a neural network. The *FeatureLearner* transforms the input X into the matrix $A \in real^{m \times k}$ where usually, but not necessarily, $m = n$. After transformation, the *Pooler* part aggregates all rows for each column k using $q \in \mathbb{N}_{\geq 1}$ permutation invariant function(s). The *Predictor* uses these $\mathbb{R}^{q \times k}$ pooled values for prediction, again with a neural network. The data dependent problem defines the mathematical form of the prediction and therefore the final layer of the *Predictor*.

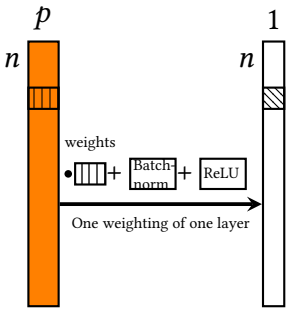


Figure 2.2.: One exemplary weighting inside one layer of the *FeatureLearner*. One layer consists of multiple such weightings. Each weight refers to its respective parameter from left to right. This weighting is followed by batch normalisation, and a ReLU-function is used as nonlinearity.

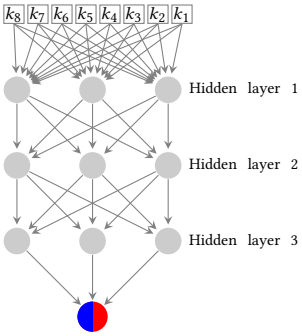


Figure 2.3.: Exemplary *Predictor*, here shown as a multi layer perceptron. The input to the *Predictor* are the pooled values over all cells from the *Pooler* and possibly additional information related to the biological sample. Each of the pooled values is regarded as one self-contained feature of the biological sample. The output is here shown as single node for a binary classification problem but adaptable to any supervised learning problem.

2.2. CCC workflow

The python package CCC contains a routine `cccrunner.run()` which handles preprocessing, training, validation and testing on new data for you. It makes use of the FPP concept (see section 2.1) and is guided by pytorch-lightning[159], a pytorch[157] wrapper, and hydra[160] to set hyperparameters. Here I will give an overview of the core-functionality and conceptional layout of the package, closely following Figure 2.4

CCC follows the flow cytometry analysis workflow explained in section 1.4 and visualised in Figure 1.4. First, preprocessing: Transformation, filtering, normalisation and subsampling can be performed inside CCC. Second, predictive modelling using the FPP concept directly on the preprocessed cell cloud data. Third, investigation of the final model and its predictions.

CCC consists of three major pipelines: Training, testing and inference, see Figure 2.4.

Each pipeline starts by creating MetaData objects consisting of the filenames, class labels and additional information about each of your biological sample. Each file must include all cells and their parameters for one sample. After the MetaData objects are created, either separate MetaData objects are given for training, validation and test set, or one MetaData object is split according to the `data`-configuration.

The resulting MetaData parts are used to generate the actual data (point clouds) where transformations are defined in the `gen_pointclouds`-config. Transformations include subsampling of cells, actual mathematical transformations like *asinh* or log or more complex methods and combinations. Pre-Pre-transforms and pre-transforms special transformations that are only applied when loading the files the first time and are persistent. Thus pre-transformations are useful because the resulting data is saved for fast access. In contrast, transforms (mind the missing "pre-") are applied every time a sample is used and are not persistent. For example, given a sample of 50000 cells, when randomly drawing 1000 cells during pre-transformation, you can subsequently only use exactly those chosen 1000 cells. In contrast, when randomly drawing 1000 during transformation, each time the sample is used, they are drawn again randomly from the total 50000 cells. This example can be used for subsampling: Usually, samples consist of a high, variable number of cells. By pre-transforming each sample to a random subset of 50000 cells, you restrict every sample to have exactly this amount of cells. The nomenclature of transformations is based on the usage in pytorch[157].

If in training or test mode, all possible feature combinations are per default plotted for one exemplary sample: Raw plots use the complete data without any transformation. Processed plots show pre-transformed data with and without transformation.

Training During training, the `model`-configuration defines which network, how many layers, nodes, which activation functions, etc., are used. Also, verbosity is an option: Intermediate plots and values can be shown in (e.g.) Tensorboard - however, mind that intermediate plots increase the computational burden. If a GPU is in use, this burden worsens because plotting needs the data on the CPU. Consequently, much data is regularly swapped between GPU and CPU.

The training -configuration defines the actual training process: Learning rates, number of epochs, which validation value(s) to monitor, GPU-usage or how the Dataloader behaves during each epoch. The final best model based on the validation performance is saved as a checkpoint-file into a directory defined by hydra[160].

Testing During testing, the directory of a previously learned and saved model must be given. This model is then loaded and applied to the given test data.

Afterwards, many investigative plots can be automatically generated, configured by the investigation -configuration. Apart from ROC curves, activation plots and SHAP-plots, three important files should be mentioned. First, part_meta_dict . pickle includes the MetaData objects used for training, validation and test data. Second, CleanTestOutput. pickle holds most of the intermediate values from the network in a nicely prepared form for further use in plots or other subsequent investigations. Third, if decision trees and gatings were computed, dtrees_gatingnodes . pickle holds the decision trees and gatings for each sample class

SHAP-plots explain the importance of each cell-feature from the *FeatureLearner*. Decision trees and gatings replace the *FeatureLearner* with a comprehensive approximation how the network finds the important cells for classification. SHAP-plots, decision trees and gatings will be explained in Part III.

A convenience function check_final_models () enables application and comparison of multiple models, easy exchange of test-datasets and overwriting of configurations. This configuration can enable or disable functionalities of the investigations.

Inference During training, the *FeatureLearner* can be approximated with multiple decision trees, explained in detail in Part III. These decision trees are saved in dtrees_gatingnodes . pickle and use the original p cell-parameters. Each decision trees cuts the cell-parameters to approximate each of the k generated cell-features from the *FeatureLearner*. During training, discriminant decision tree leaves are identified. Those specific leaves then have a different number of cells for each class of biological samples.

During inference, you do not apply the trained network but apply the saved decision trees. The number of cells inside each discriminant leaf then "predicts" the class of the biological sample. Before applying those gates, the MetaData is - as during training and testing - used to generate the actual point clouds, including ((pre-)pre-)transformations.

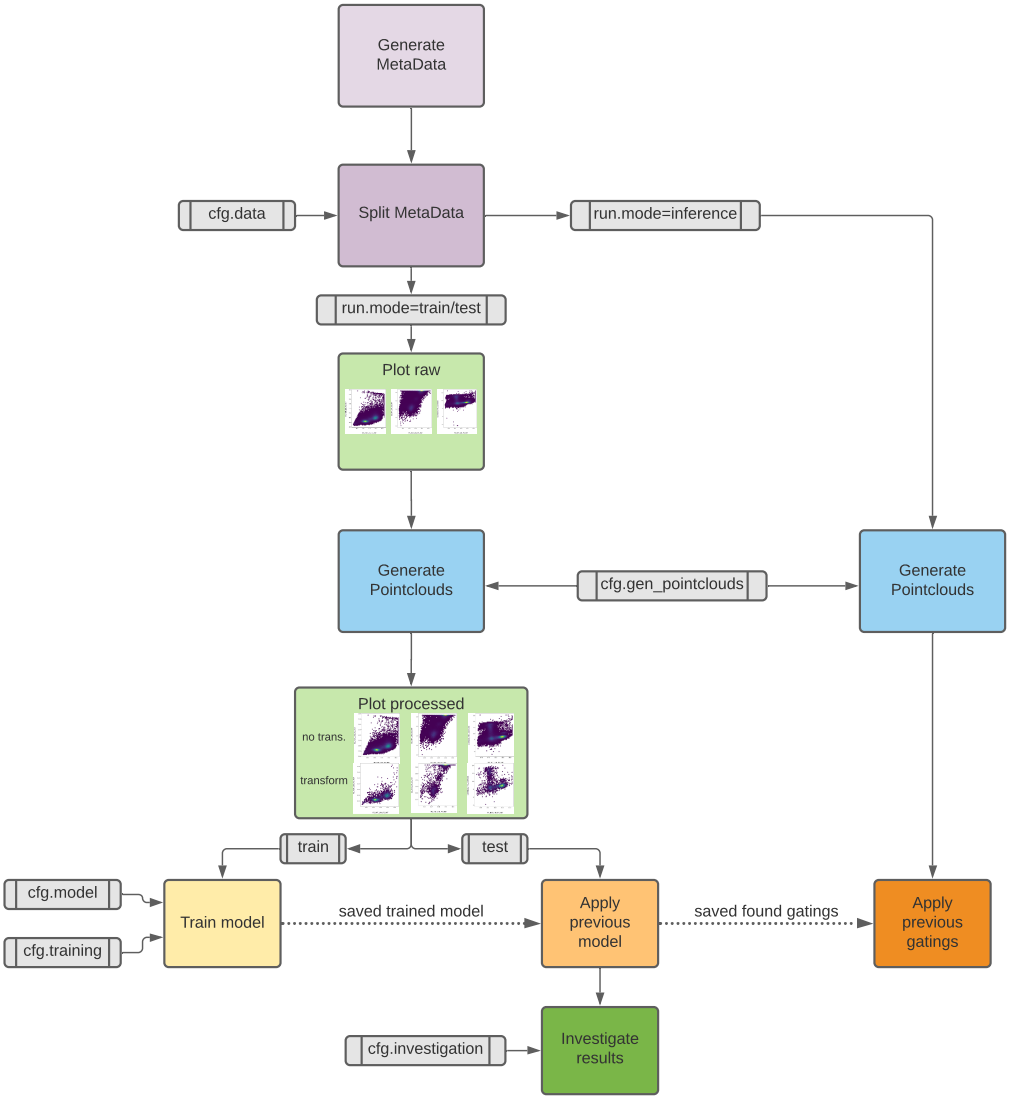


Figure 2.4.: CCC workflow. Start your analysis by creating MetaData object(s), which are consequently split into training, validation and test part. If `run.mode` is either train or test, all possible combinations of raw feature values for one sample are plotted. Then the point clouds are generated according to the `gen_pointclouds`-configuration. Again, all possible combinations of cell-parameters for one biological sample are plotted, always with pre-transforms but with and without transforms. Training behaviour is defined by the `model`-configuration defining the model and the `training`-configuration defining the training process itself. The final model, intermediate values and plots are saved. Afterwards, this saved model can be tested: Proceed through the whole process again but apply the model to the test data without retraining. Many investigations of these results are switchable through `investigation`-configuration. Decision trees and gatings can be saved. Finally, another subsequent call of this workflow in the inference mode applies the previously found decision trees to new data without training nor applying the neural network for fast and explainable prediction of new samples.

3. Discussion

Cytometry characterises individual cells of a biological sample. A cell is defined through all of its cell-properties, but only part of them are measured in a cytometry experiment. Those measured cell-parameters are used to characterise the cell. A biological sample contains millions of cells, and their characterisation can be used to define cohesive cell subpopulations which reflect certain anatomic, morphologic or functional types of cells[42].

Flow and mass cytometry share the same concept of measuring cells: Different dyed antibodies are mixed with the cell suspension, bind to their respective cell surface markers, and the dye can be measured. In flow cytometry, there are fluorescent dyes, where mass cytometry uses metallic dyes. It is possible to characterise millions of cells from a biological sample with up to 50 measurements per cell[5]. These measurements can be written as a matrix of unordered cells in the rows, and some cell-parameters per cell in the columns defines the biological sample.

In this thesis, I use cytometry data for predictive modelling, where a class is assigned according to all cells of a biological sample together. This class enables clinical patient decisions when the class is known to respond to a certain treatment. Due to the unordered nature of cytometry measurements, only the presence or quantity of cells define the class of a biological sample but not which cell was measured first.

There are multiple approaches for predictive modelling in cytometry which can be grouped into three concepts. First, partition and count cells based on the marker values followed by predictive modelling[7,9–11]. Second, apply unsupervised cell subpopulation identification followed by predictive modelling[7,13–17,19,22–24,26–28,30–32,39]. Third, use all cells without previous summarisation of the cells[2,3,6].

In this work, I focus on predictive modelling based on the third concept where all cells are used directly. I use a neural network which uses a defined number of cells per biological sample, weights their cell-parameters, summarises the weighted cells and finally yields the class label of the biological sample. In contrast to splitting cell subpopulation identifications and predictive modelling, where all cell subpopulations must be found and the predictive model has to select the ones relevant for the classification, the neural network directly searches only for the cell subpopulation(s) relevant for the classification. Therefore there is no more restriction to prior knowledge about cell subpopulations nor potential loss of relevant cell subpopulations as all populations are looked at simultaneously.

The first application of neural networks to flow cytometry was in 2017[2]. Multiple linear combinations of the cell-parameters were followed by an aggregation per linear combination across all cells. This aggregation was either the mean or the maximum, which resulted in one single value per sample and linear combination. Finally, a linear combination of these values followed by a sigmoid function predicted the binary class of a biological sample.

More recently[3], using neural networks in cytometry was improved. Instead of having only linear combinations of the cell-parameters and also the aggregated values, introducing multiple layers improved the performance. Multiple layers enable more complex, non-linear combinations of cell-parameters and aggregated values. In consequence, multiple layers improve the capability of the network to model more complex relationships between cells and classes of biological samples.

A related concept to neural networks in cytometry is point cloud classification. Multiple points defined through 3D coordinates in space form a point cloud, and this point cloud can have a specific shape. Point cloud classification aims to classify the shape of a given point cloud. There are multiple approaches to classify point clouds with neural networks[1,161–172]. Again: A point cloud is a number of unordered points where three values define each point, and this complete point cloud has a certain shape (=class). Cytometry data is a number of unordered cells where p cell-parameters define each cell, and all cells together have a certain class. From a mathematical point of view, there is no difference between point clouds and cytometry data. Both are a matrix with a number of unordered items as rows and a defined number of values per item as columns. Thus, we can use advances from the field of point cloud classification in cytometry! A small caveat here might be that the number of cell-parameters is usually higher than the number of values of each point, so some advances in point cloud classification might not work equivalently.

In this part, I introduce a modular concept organising neural networks for cytometry data and a python package applying the concept. For this I build on the works in cytometry[2,3] and point cloud classification[1,161–172]. My *FeatureLearner*, *Pooler* and *Predictor* (FPP) concept consists of three modular parts. First, the *FeatureLearner* transforms the measured matrix of a biological sample into a new matrix with new, predictive cell-features. Second, the *Pooler* aggregates these cell-features into a composition vector of values per biological sample. Third, the *Predictor* uses this composition vector to predict the final class. Modularity comes from the fact that each of these three parts can be replaced by any neural network structure as long as it fulfils the same purpose! Exemplary, the *Pooler* can be the mean per cell-feature but also the maximum, or even both together, resulting in two pooled values per cell-feature. The *Predictor* can be easily replaced according to what your prediction should look like: A sigmoid function as output for binary classification, soft-max function for multiclass classification or a single output node for regression problems.

A predictive cell-feature describes some cell-property that is present only in informative cells. If only T-cells were informative, only those would get a high cell-feature value, and all other cells get the value zero in that specific cell-feature. Thus when aggregating with the mean, biological samples with no T-cells have a mean of zero, but biological samples with some T-cells have a positive mean. Subsequently, the *Predictor* can leverage this difference. The narrative behind "informative" is: If the *FeatureLearner* built only uninformative cell-features, the performance after pooling could not possibly be good. The opposite is not necessarily true, though: If the performance of the *Predictor* is good, it means only that some combinations of some of the cell-features were informative. Still, neither all cell-features have to be used by the *Predictor*, nor does one cell-feature alone have to predict the class of a

biological sample.

In addition to the pooled cell-features, any sample-wise information can be included by just extending the pooled vector by the number of additional features[3]: The input for the *Predictor* becomes all pooled cell-features ***and*** these informations. Additional features can be any sample-wise information like age, sex or also batch numbers of the respective experiment. However, for their[3] specific binary classification dataset, the classification performance did not improve by incorporating additional information per biological sample.

With CCC (Cell Cloud Classification), I built a python package that implements the FPP concept. It is easy to use and directly applicable to new data with two or more classes of biological cytometry samples. The package's functionality is optimised towards binary and multi-class classification problems. To date, the package is available on the local GitLab-repository from Prof. Dr Rainer Spang.

Apart from classification problems, regression problems are already possible in CCC; just the investigative follow-up plots are not designed accordingly. An additional, more complex issue is modelling survival of biological samples because of censored survival times. This should be a minor problem though, as modelling survival is already established for neural networks[173–184].

In summary, I provide a Cell Cloud Classification python package (CCC). It is a modular framework for neural networks to classify and investigate cytometry samples. The investigation includes a novel, decision-tree based concept to explain the predictions.

The CCC package is ready-to-use and can be used for predictive modelling with classification and regression problems. It may be interesting to see the performance of the FPP concept not only in flow and mass cytometry but also in higher-dimensional data like single-cell RNA-seq (scRNA-seq). These hold around 20000 cell-parameter per cell[185] where the curse-of-dimensionality must be regarded as distances are not reliable anymore[186]. However, unreliable distances are ubiquitous for most algorithms regarding scRNA-seq. An initial dimension reduction is often proposed as the first step to avoid this issue[185].

Incorporating confounders of a biological sample might be useful. The *Predictor* can take additional sample-specific values like age, sex, smoking or other risk factors but also technical batch-effect information like executive personnel, place or date of the measurement. Another publication[3] showed that there was no classification performance improvement for their dataset when using demographic data: Age, sex and race groups. However, including demographic and other sample-specific confounders might lead to better results in other datasets.

In line with incorporating confounders, including classical cell subpopulation counts directly as sample-specific values might enhance CCC. Classically, cells from cytometry measurements are binned into cell subpopulations. Afterwards, the number of cells inside each cell subpopulation of a biological samples are used as feature for predictive modelling. When these cell-numbers are known, the neural network can use them directly and does not need to emulate them in a cell-feature. Consequently, only unknown cell subpopulations must be found additionally as cell-feature. This might improve classification performance and interpretability of the cell-features.

Pre-training the neural network might be a potential solution for limited amounts of data and might improve interpretability. Assume the cell subpopulation for each cell is known (manual, supervised or unsupervised) gating. A neural network can be used to predict each cell's cell subpopulation. This network could just be multiple dense layers predicting the probability of each cell subpopulation. This can be understood as adding a cell subpopulation-prediction layer directly after the *FeatureLearner*. If this network trained successfully, its *FeatureLearner*-part can be used as a starting point for the actual classification problem at hand. There are multiple advantages of this approach:
1) A biological sample could be used twice as the optimisation target from the classification, and cell subpopulation prediction problem are unrelated.
2) Cell subpopulation prediction can be trained on unrelated samples without classification label information.
3) As the *FeatureLearner* has a cell subpopulation-specific starting point after the first training, the final *FeatureLearner* will be closer to real cell subpopulations, hence possibly more interpretable.
4) The *FeatureLearner* weights can be frozen but also unfrozen during the actual classification problem, thus the cell-feature can still be adjusted. If the cell subpopulation information is unavailable, the supervised cell subpopulation prediction might be replaced with an autoencoder whose first part is then used as the *FeatureLearner*.

Part II.

CCC design and performance

1. Introduction

Cytometry characterises cells of a biological sample by cell-properties like size, complexity or surface markers. A measured cell-property is called a cell-parameter, which defines each cell by numerical values suitable for further analysis. Up to 50 measured cell-parameters characterise each cell[5], and the resulting data is a matrix of unordered cells in the rows, and some cell-parameters as columns. A biological sample contains millions of cells, and their characterisation can be used to define cohesive cell subpopulations which reflect certain anatomic, morphologic or functional types of cells[42].

Predictive modelling in cytometry assigns a class label to a biological sample according to all measured cells of that sample. This class enables clinical patient decisions when the class is known to respond to a certain treatment. Due to the unordered nature of cytometry measurements, only the presence or quantity of cells define the class of a biological sample but not which cell was measured first.

There are multiple approaches for predictive modelling in cytometry which can be grouped into three concepts. First, partition and count cells based on the cell-parameters followed by predictive modelling[7,9–11]. Second, apply unsupervised cell subpopulation identification followed by predictive modelling[7,13–17,19,22–24,26–28,30–32,39]. Third, use all cells without previous summarisation of the cells[2,3,6].

In the first part of this work, I introduced the FPP concept, a concept to organise neural networks for predictive modelling in cytometry. FPP resides in the third concept. Multiple cells of a biological sample are used together to predict the class the biological sample. FPP consists of three parts. First, the *FeatureLearner* transforms the measured matrix of a biological sample into a new matrix with new, predictive cell-features. Second, the *Pooler* aggregates these cell-features into a composition vector of values per biological sample. Third, the *Predictor* uses this composition vector to predict the final class.

Any neural network has a certain architecture which must be defined through hyperparameters before training. Training then uses the data to optimise the weights. The architecture together with the weights then yield a prediction for each biological sample. The FPP concept can be implemented by using dense layers for the *FeatureLearner* and *Predictor*. In that case, all nodes from one layer are connected to all nodes from the following layer plus some non-linear activation function. Therefore, the number of nodes, layers and activation functions are architectural choices before training. Additionally, the *Pooler* must be chosen between such that all cells are aggregated into a single value without taking the order of cells into account.

In addition to architectural choices for neural networks, there is a further pseudo-hyperparameter: Network initialisation. Neural networks are defined by the connections between nodes and the weights of these connections. These weights are found by optimising a loss function suitable for your classification problem: In

essence, the data defines the weights. However, to optimise those weights, they need some initial values. It is impossible to set all initial values to zero due to the optimisation. State of the art is to initialize dense layers by drawing the weights from a uniform distribution $\mathcal{U}(-k, k)$ with $k = \frac{1}{features_{in}}$ for all layers[157]. Thus the initial seed for drawing random values can be understood as pseudo-hyperparameter.

There was a recent, successful application of neural networks to CyTOF data[6]. CyTOF is similar to flow cytometry, but instead of fluorochromes, metallic dyes are used for the binding antibodies, which can be measured with a time-of-flight mass spectrometer. 472 blood samples from multiple studies were used to predict latent cytomegalovirus (CMV). Based on this data and their analysis, the best architecture was a neural network with two dense layers of three nodes each to combine the cell-parameters, mean as *Pooler* and a single dense layer with three nodes.

A general consideration when using cytometry data in predictive modelling is if and how to subsample cells. In section 1.4.1.5 I introduced three ways to subsample cells: 1) Random where a defined number of cells are randomly subsampled. 2) Density-based subsampling of cells, equalising the density over the spanned space of all cell-parameters. 3) Outlierness-based, where the outlierness of a cell is proportional to the euclidean distance to other cells. Depending on your data and the related classification problem, different subsampling variants might be optimal.

In this part of the thesis, I investigate architectural hyperparameters, subsampling variants and the initialisation pseudo-hyperparameter with respect to their performance impact on the classification of non-/CMV biological samples.

2. Methods

2.1. CMV dataset

I used 472 samples in the ImmPort database[187] from nine studies including SDY112, SDY113, SDY305, SDY311, SDY315, DSY472, SDY478, SDY515, and SDY519[188–190]. These studies mainly examined influenza in plasmablasts and T-cells of healthy individuals but also assessed latent cytomegalovirus (CMV) status. Following a recent publication[3], I identified the studies and their samples by querying the ImmPort database for healthy individuals with CyTOF and CMV antibody titer data. I downloaded the data, transformed all cell-parameters with the standard arcsinh-transformation and a cofactor of 5 ($arcsinh(\frac{x}{5})$). I restricted each sample to 27 overlapping cell-parameters and subsampled randomly to 10000 cells. As some samples overlap between some studies, training, validation and test splits were performed accordingly, following a recent publication[3]: The training set consists of the studies SDY112, SDY113, SDY305, SDY311, SDY315, SDY472, and SDY478 as they contain overlapping samples. SDY515 was used as validation and SDY519 as the test set. CMV antibody titer was cut at value 2, where all samples with less than 2.0 are CMV-negative samples resulting in a binary classification problem.

2.2. Tested hyperparameters of neural networks in cytometry

In the following three subsections, I show which subsampling methods and network architectures are investigated and how the networks are initialised. Additionally, I explain the two separate experiments and how the neural networks were trained.

The subsampling methods and network architectures are hyperparameters, and I will denote one unique combination of these hyperparameters as a variant. One variant has an additional pseudo-hyperparameter: The initialisation of internal weight. Therefore the pseudo-hyperparameter is the seed of the random number generator defining the initial internal weights. I compare multiple variants, and to ensure reproducibility I initialise them with the same seed. Those same-seed variants are a run.

2.2.1. Subsampling methods

Due to the neural network architecture, each sample must be subsampled to a defined number n of cells. Here I explain fixed and unfixed as well as random and outlierness-based subsampling.

2.2.1.1. Fixed and unfixed

Fixed subsampling draws a fixed number of cells once from each biological sample and uses those (fixed) cells as all cells from one biological sample.

In contrast, all cells from one biological sample could be used. To still get the defined number of cells n, unfixed subsampling draws n cells without replacement from all cells every time the biological sample is used.

2.2.1.2. Draw randomly or based on outlierness

When subsampling your data, cells can either be drawn n randomly or by leveraging the outlierness-concept from previously published work[2].

The outlierness of a cell is proportional to the euclidean distance to other cells. Cells can be drawn according to that outlierness: The higher it is, the more likely that cell is chosen. An already published work on predictive modelling in cytometry[2] already applies this outlierness-concept which was presented in another publication[92].

To calculate the outlierness, calculate the distance of every cell to all others. This becomes prohibitively computational expensive for large numbers of cells. Therefore, a more feasible way of calculating the outlierness is to draw a small number of reference cells randomly and calculate the distance of each cell to the drawn reference cells[92]. Then the outlierness for cell c with p markers is defined as

$$[\min_{c' \in \mathbf{C}}(dist(\mathbf{c}, \mathbf{c}'))]^{pow} \tag{2.1}$$

where $\mathbf{C}$ is the set of randomly drawn reference cells from the current biological sample.

To increase the probability of selecting distant cells, use only the top 10% cells based on outlierness when sampling cells according to it[2]. In contrast, I introduce a power (pow) to this outlierness which increases the outlierness of distant cells. In effect, the probability of selecting distant cells increases, but without a hard cutoff of 10%.

When sampling based on outlierness, I drew 50% randomly and 50% based on outlierness, 1000 reference cells, and $pow = 1.5$.

2.2.2. Network architecture

The network architecture for classification based on single-cell measurements consists of 3 parts, as shown in Figure 2.1. In the *FeatureLearner*, dense layers combine the p cell-parameters of each of the n input cells into k artificial cell-features. The *FeatureLearner* usually comprises multiple layers. Each layer consists of multiple weightings of the p markers where each weighting is applied to all cells, each followed by batch normalisation[158] and ReLU-nonlinearity[155].

The number of input cell-parameters is fixed based on the input data (see section 2.1) to 27 cell-parameters. I investigate 1,2,3 and 10 consecutive dense layers where all layers have either 3 or 10 nodes.

The *Pooler* uses these new cell-features and aggregates all n cells into one aggregated value per cell-feature. The aggregation function must be permutation

invariant after the cells do not have any intrinsic order, so the mean or the max are suitable options. I investigate the mean and maximum as aggregation functions.

The *Predictor* receives k aggregates which are combined with a multi layer perceptron with a sigmoid function as the last layer to enable binary classifications. Each *Predictor* layer is a dense layer followed by batch normalisation[158] and ReLU-nonlinearity[155]. In my implementation, the number of hidden layers inside the multi layer perceptron is the number of defined *Predictor*-layers minus one. Thus, if a single layer is defined there is no hidden layer and, in consequence, neither batch normalisation[158] nor ReLU-nonlinearity[155], resulting in a weighted sum of the k *Pooler*-aggregates followed by a sigmoid function. I investigate 1,2,3 and 10 of these *Predictor*-layers where all hidden layers (if existing) have either 3 or 10 nodes.

2.2.3. Weight initialisation

After defining the architecture, every neural network needs some initial values for its internal weights. Usually, they are drawn randomly or based on more sophisticated initialisation routines[191], but they also have random elements. I use the defaults of pytorch[157], which initialises dense layer weights by drawing from a uniform distribution $\mathcal{U}(-k, k)$ with $k = \frac{1}{features_{in}}$ for all layers.

I ensure reproducible runs by setting a seed for the random number generator every time before initialising the weights. This seed is the same for all variants inside one run.

2.2.4. Two separate experiments

Experiment 1 considered different subsampling variants: Fixed and unfixed, random and outlierness-based, drawing 2500, 5000, or 9500 cells, and mean or max pooling. The architecture was fixed to a grid-search based optimum as reported previously[3]: A neural network with two dense layers of three nodes each to combine the cell-parameters, mean as *Pooler* and a single dense layer with three nodes.

Experiment 2 focused on network architecture-dependent performance differences: Cells were randomly subsampled and only mean pooling used. Fixed or unfixed subsampling and drawing 2500, 5000, or 9500 cells were used. The architecture was set to 1,2,3 or 10 layers with each 3 or 10 nodes in the *FeatureLearner* and 1,2,3 or 10 layers with each 3 or 10 nodes in the *Predictor*.

2.2.5. Training, validating and testing

Each model was trained with a learning rate of 0.0001 and batch size of 60. Learning was stopped when the monitored validation loss (binary cross-entropy) did not improve by 0.00001 in the last 200 epochs. Performance was measured using the area under the receiver operating characteristic curve (AUC).

3. Results

3.1. Overview over the two experiments

As described in the methods (see subsection 2.2.4), I had two separate experiments: Experiment 1 only varied hyperparameters that do not change the architecture. Therefore, all variants in experiment 1 had the same initial weights and therefore the exact same starting conditions. In contrast, experiment 2 also changed architectural hyperparameters, therefore starting conditions varied between variants, see subsection 2.2.3.

In Figure 3.1, I showed that the AUC varies for a single variant more than 35%. Each dot is one run of the same variant and the dots are the basis for the boxplots, split for training, validation and test data performance.

In Figure 3.2 and Figure 3.5 I showed the same AUC-boxplots but for all tested hyperparameter variants on the x-axis. Each variant is defined through a unique set of hyperparameters, which are color coded - the legend can be found in Figure 3.3.

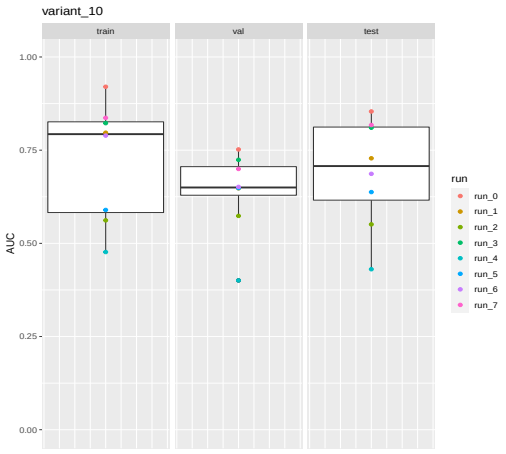


Figure 3.1.: Boxplots of all runs for one exemplary variant for training, validation and test set. In experiment 1, each variant was run 20 times. In experiment 2, each variant was run seven times, shown here. Each point is one run, and the boxplots are generated based on these points.

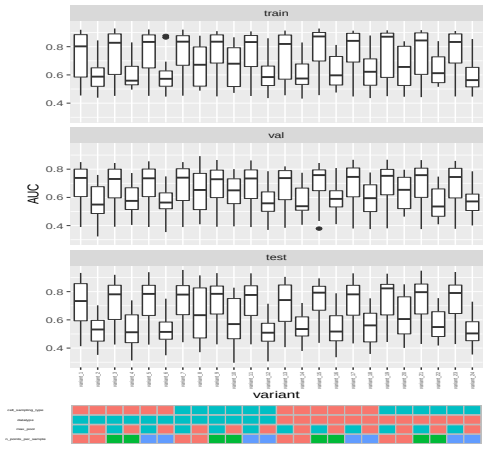


Figure 3.2.: Experiment 1: Boxplots of allruns for each variant. The y-axis of the upper three parts shows the AUC for training, validation and test set. The bottom part is a color-coded information about cell subsampling type (fixed or unfixed), datatype (outlierness or random), pooling (max or mean) and the number of points per sample (2500, 5000, 9500)

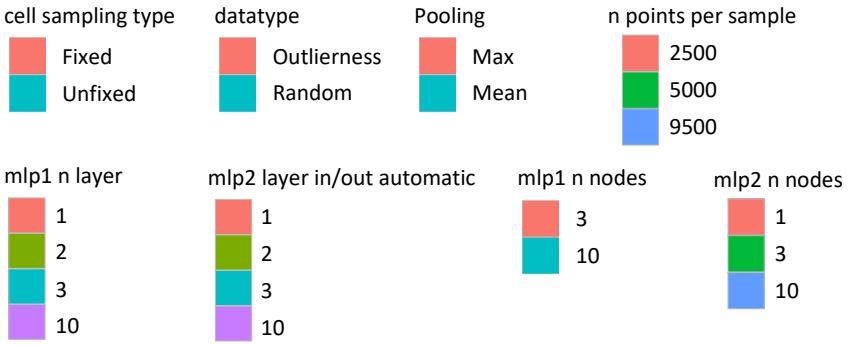


Figure 3.3.: Color codings for Figure 3.2, Figure 3.5 and Figure 3.6.

3.2. Mean pooling performs better than max pooling

In experiment 1, I compared hyperparameters that do not change the architecture. The only hyperparameter with strong impact on the AUC was the decision between mean or maximum as *Pooler*. Here the mean showed consistently superior AUCs, as shown in Figure 3.4. The boxplots include all runs of all tested variants.

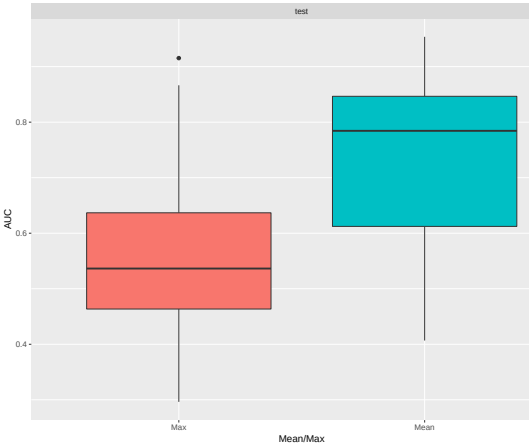


Figure 3.4.: AUC on the y-axis with a boxplot containing all variants and runs, split by the *Pooler* type. Either the mean or the maximum was used to pool all given cells per feature.

All other tested hyperparameters are shown in appendix section C: Fixed and unfixed subsampling, random and outlierness-based subsampling, drawing 2500, 5000, or 9500 cells, and mean or max pooling. However, the shown boxplots in the appendix do not include all runs of all tested variants. Instead, each boxplot includes only the best run for each variant.

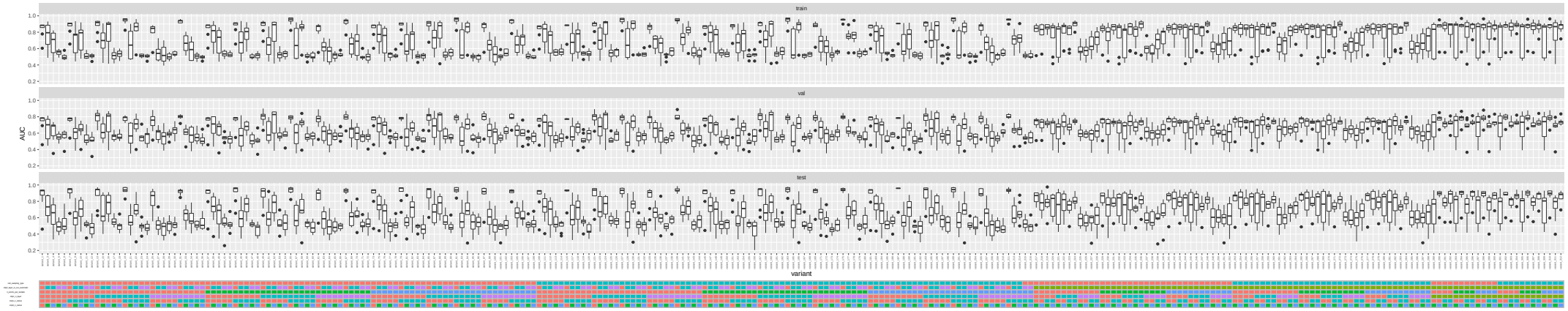


Figure 3.5.: Boxplots of all runs for each variant in experiment 2. The y-axis of the upper three parts shows the AUC for training, validation and test set. The bottom part is a color-coded information about cell sampling type (fixed or unfixed), the number of *Predictor* (mlp2) layers (1,2,3 or 10), the number of cells per biological sample (2500, 5000, 9500), the number of *FeatureLearner* (mlp1) layers (1,2,3 or 10), the number of nodes in each layer of the *FeatureLearner* (mlp1) (3 or 10) and the number of nodes in each layer of the *Predictor* (mlp2) (1, 3 or 10). The *Predictor* (mlp2) layers are special: The number of hidden layers with the respective number of nodes in each layer are the number of *Predictor* (mlp2) layers minus one as always one layer is automatically added to connect the *Predictor* (mlp2) with classification specific output nodes. The legend for the color-coded variant description at the bottom is shown in Figure 3.3.

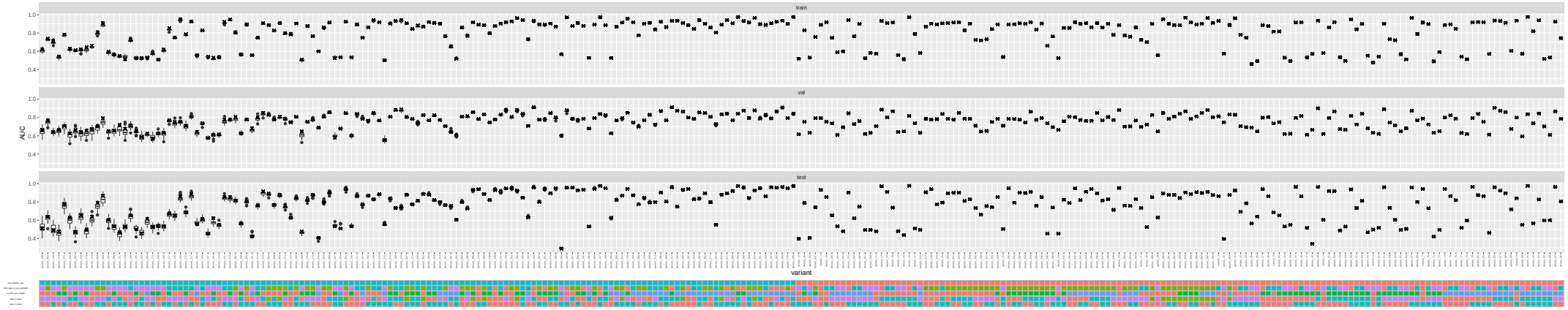


Figure 3.6.: Boxplots of the AUC over 50 predictions in experiment 2 per variant. Each prediction predicted all biological samples but the cells were resampled before prediction. The y-axis of the upper three parts shows the AUC for training, validation and test set. The bottom part is a color-coded information about cell sampling type (fixed or unfixed), the number of *Predictor* (mlp2) layers (1,2,3 or 10), the number of cells per biological sample (2500, 5000, 9500), the number of *FeatureLearner* (mlp1) layers (1,2,3 or 10), the number of nodes in each layer of the *FeatureLearner* (mlp1) (3 or 10) and the number of nodes in each layer of the *Predictor* (mlp2) (1, 3 or 10). The cross denotes the AUC for the mean over all 50 predictions. For all variants where drawing cells is fixed, all predictions are by definition identical. The legend for the colour-coded variant description at the bottom can is shown in Figure 3.3. For each variant, the best run is chosen based on the validation AUC of the first of 50 predictions.

3.3. Weight initialisation is crucial for the final performance

After I had already found that mean pooling performed better than max-pooling, I restricted the initialisation analysis to those variants with mean-pooling. In experiment 1, all variants of one run had the exact same weight initialisation. The network architecture was fixed for all variants as described in the methods according to the best-reported architecture on this CMV dataset[3].

The initialisation impacted the AUC stronger than all variant choices in experiment 1, shown in 3.7 A. Each run reflects a set seed to initialise the starting weights and therefore also the performance of the neural network. The final performance ranged from models where nothing was learned (AUC $\approx$ 0.5) to excellent models (best AUC = 0.9537), but the AUC inside one run differed from 3% up to 20%.

In experiment 2, shown in 3.7 B, the differentruns with their initialisations do not show significant differences.

Even if the initialisations had strong impact on the performance, validation and test AUC correlate strongly ($r = 0.91$, see Figure 3.9 A).

When selecting the best performing run of each variant based on the validation AUC, I can reconsider the previous mean vs max *Pooler* assessment. Comparing the reconsidered Figure 3.9 B and the previous Figure 3.4 showed that the performance difference between mean and maximum was even bigger when selecting the best run for each variant.

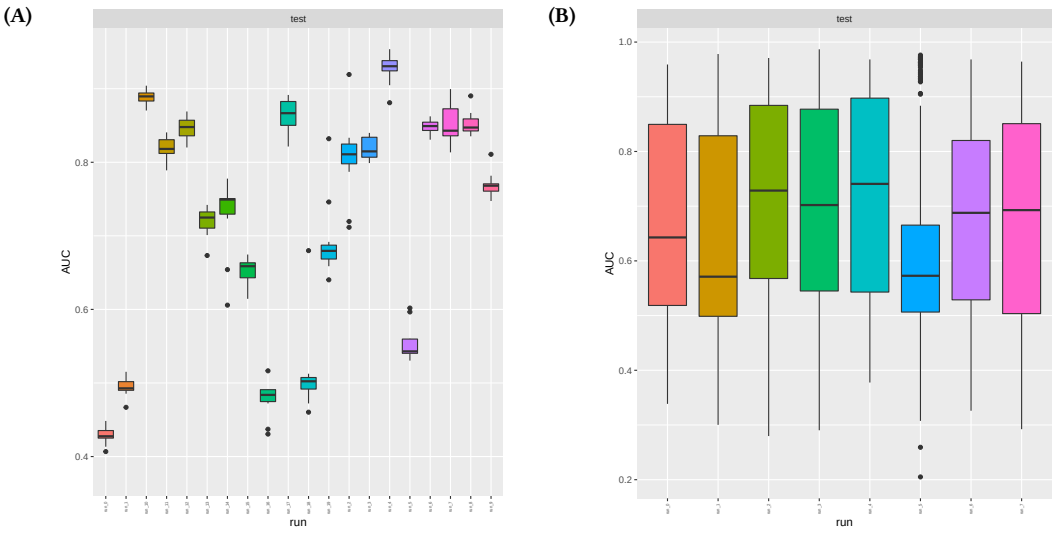


Figure 3.7.: (A) AUC on the y-axis, runs on the x-axis. Each boxplot contains all mean-pooling variants of one run (one single initialisation of the network weights) from experiment 1. (B) AUC on the y-axis, runs on the x-axis. Each boxplot contains all mean-pooling variants of one run from experiment 2, where each variant is initialised differently by design.

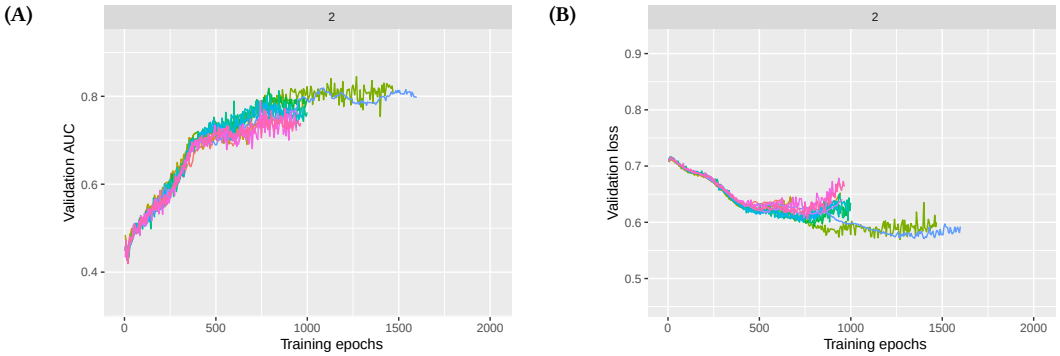


Figure 3.8.: All variants from run 2 of experiment 1, where each line shows the learning path of one variant. (A) AUC on the y-axis, training epochs on the x-axis. Higher values are better. (B) Loss on the y-axis, training epochs on the x-axis. Lower values are better.

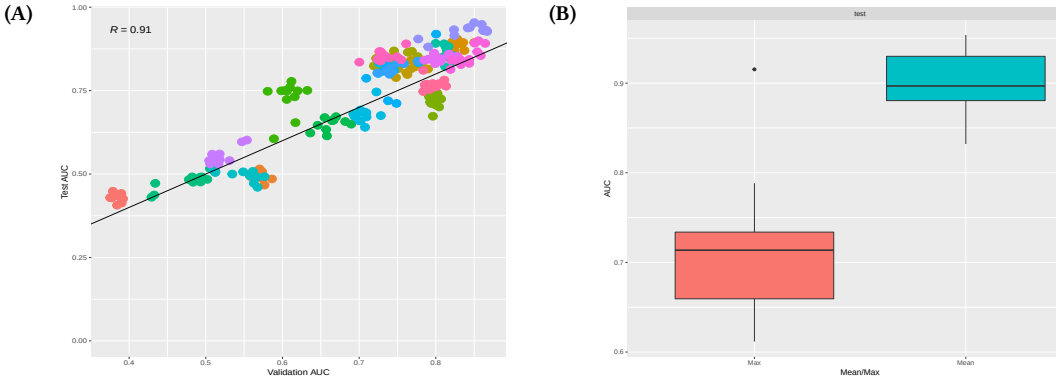


Figure 3.9.: (A) AUC on the test set (y-axis) vs AUC on the validation set (x-axis), both times the results from experiment 1. All variants of one variant are coloured the same. (B) AUC using mean or maximum as aggregation function by the *Pooler* on the test set.

3.4. Performance depends on network architecture

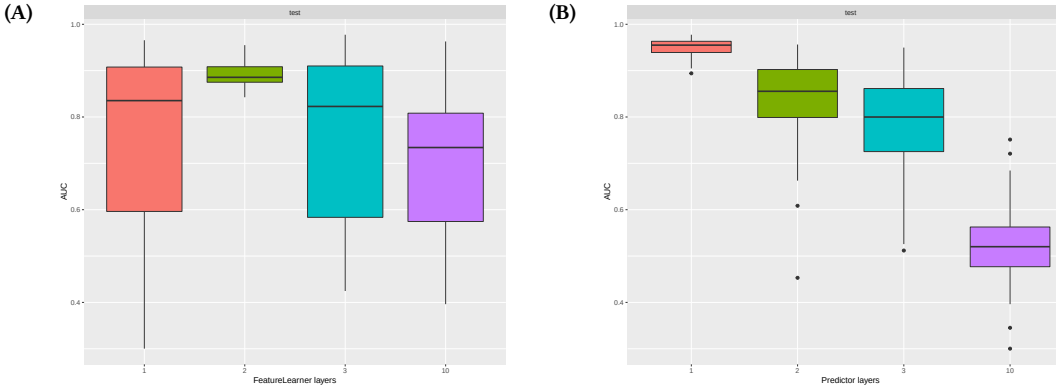


Figure 3.10.: AUC for variable number of (A) *FeatureLearner* and (B) *Predictor* layers compared over allruns and all other hyperparameter variants on the test set.

In the second experiment, I focused on network architecture differences. Every variant has a unique network architecture, resulting in different weight initiali-

sations per variant. Here I only show the impactful hyperparameters. All tested variants of the second experiment can be found in the appendix in Figure D 1.

In the following comparisons, I selected the best performing run based on the AUC of the validation set for each variant.

Figure 3.10 A compares the AUC for 1,2,3 and 10 *FeatureLearner* layers where 2 layers are significantly better.

Figure 3.10 B compares the AUC for 1,2,3 and 10 *Predictor* layers where one predictor layer corresponds to no batch normalisation nor ReLU nonlinearity. Instead, one layer corresponds to a linear combination of all pooled cell-features followed by a sigmoid function predicting the presence of CMV as described in the methods. The plot shows that the AUC is best with this linear combination (one *Predictor* layer) followed by a sigmoid and drops when more layers are used.

The number of nodes had no effect (see Figure D 1 (E) and (F)), neither for the *FeatureLearner* nor for the *Predictor*. Note that the predictor node comparison contains only settings with at least one *Predictor* layer which was already found inferior in Figure 3.10 B.

3.5. Subsampling affects performance ranges

The overall performance of different subsamplings, fixed or unfixed, random or outlierness-based and different numbers of drawn cells did not significantly affect the AUC. (See appendix Figure C 1 (A), (B) and (D) for experiment 1 and Figure D 1 (A) and (C) for experiment 2).

In Figure 3.6 I plot the best run according to the validation AUC of the first prediction per variant. Each boxplot includes 50 AUCs as a result of predicting all biological samples 50 times. The variants are sorted by their test AUC-range from highest (bad) to lowest range (good). The AUC-range is calculated by subtracting the lowest from the highest AUC of the 50 AUCs. The cross denotes the AUC when each prediction is calculated as the mean over all 50 predictions. For all variants where drawing cells is fixed, all predictions are, per definition, identical.

Figure 3.6 shows that training performance generally varied less than validation and test performances. Test performance can vary up to 31% AUC.

The AUC ranges are shown in boxplots, stratified by the parameters. As previously, I selected the best run per variant according to the AUC of the validation set when using the first of 50 predictions of each biological sample. See Figure E 1 for all comparisons; I show the significant differences in Figure 3.11.

More *Predictor* layers lead to higher AUC ranges (B). The number of *FeatureLearner* layers also influences the AUC ranges but is most stable with two layers (A). The more cells per biological sample are chosen, the smaller is the AUC range (C).

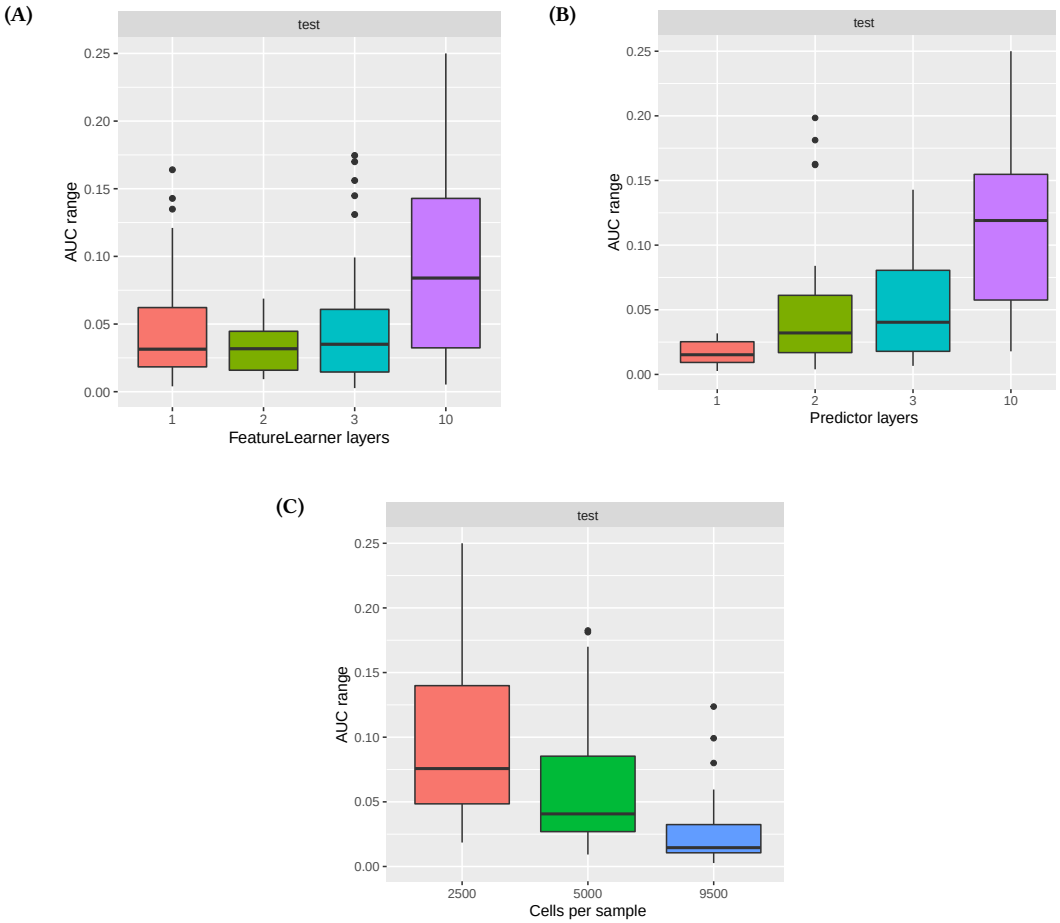


Figure 3.11.: AUC ranges when predicting biological samples multiple times in the unfixed setting of subsampling cells. The boxplots include 50 AUCs as a result of predicting all biological samples 50 times. For each prediction of all biological samples, I calculated the AUC, which then formed the boxplots in Figure 3.6. I calculated the AUC range [for each boxplot] minimum to maximum. This range is shown on the y-axis, each boxplot contains all variants (except for the stratifications). One variant is only the best run per variant according to the AUC of the validation set when using the first of 50 predictions of each biological sample.

4. Discussion

Cytometry characterises cells of a biological sample by cell-properties like size, complexity or surface markers. A measured cell-property is called a cell-parameter, which defines each cell by numerical values suitable for further analysis. Up to 50 measured cell-parameters characterise each cell[5], and the resulting data is a matrix of unordered cells in the rows, and some cell-parameters as columns. A biological sample contains millions of cells, and their characterisation can be used to define cohesive cell subpopulations which reflect certain anatomic, morphologic or functional types of cells[42].

Flow and mass cytometry share the same concept of measuring cells: Different dyed antibodies are mixed with the cell suspension, bind to their respective cell surface markers, and the dye can be measured. In flow cytometry, there are fluorescent dyes, where mass cytometry uses metallic dyes. It is possible to characterise millions of cells from a biological sample with up to 50 measurements per cell[5]. These measurements can be written as a matrix of unordered cells in the rows, and some cell-parameters per cell in the columns defines the biological sample.

In this thesis, I use cytometry data for predictive modelling, where a class is assigned according to all cells of a biological sample together. This class enables clinical patient decisions when the class is known to respond to a certain treatment. Due to the unordered nature of cytometry measurements, only the presence or quantity of cells define the class of a biological sample but not which cell was measured first. Due to the unordered nature, special predictive modelling methods are necessary. Predictive modelling for cytometry data needs special algorithms to factor in the unordered cells.

In the previous part of this thesis, I introduced the FPP concept for predictive modelling with cytometry data. FPP is a modular structure for neural networks containing three parts. First, the *FeatureLearner* transforms the measured matrix of a biological sample into a new matrix with new, predictive cell-features. Second, the *Pooler* aggregates these cell-features into a composition vector of values per biological sample. Third, the *Predictor* uses this composition vector to predict the final class. Therefore, the FPP concept is suitable for predictive modelling with cytometry data.

Predictive modelling with the FPP concept needs multiple user-defined hyperparameters. 1) The architecture of each FPP part defining which layers should be used and how nodes are connected. 2) A pseudo-hyperparameter, the initialisation of the internal weights of the neural network through a seed for the random number generator. 3) The subsampling used by the FPP network. Each unique combination of hyperparameters, excluding the initialisation seed, is called a model variant. Multiple variants which are trained with the same initialisation seed are called a run.

In this part, I investigated the impact of hyperparameters on the classification performance. I used publicly available data from 472 biological samples with or without CMV infection measured with mass cytometry. In the first of two experiments, I used a previously previously published network architecture[3], realized it in the FPP concept and only varied hyperparameters which do not change the initialised weights: Fixed or unfixed subsampling, outlierness-based or random subsampling, 2500, 5000 or 9500 subsampled cells and mean or maximum pooling. In the second experiment, I focused on the impact of architectural hyperparameters. Cells were randomly subsampled and only mean pooling was used. Fixed or unfixed subsampling and drawing 2500, 5000, or 9500 cells were used. The architecture was set to 1,2,3 or 10 layers with each 3 or 10 nodes in the *FeatureLearner* and 1,2,3 or 10 layers with each 3 or 10 nodes in the *Predictor*.

In the first experiment, I varied only hyperparameters which do not change the initialised weights. I observed that mean pooling works better than maximum pooling. I tested multiple initialisation seeds and found that the initialisation itself was crucial for the final performance after training. From excellent binary classification with an AUC of 0.9537 down to random predictions with an AUC of 0.5. In Figure 3.8, I showed that in one run of the first experiment, the initial binary-cross-entropy loss and consequently also the AUC are almost identical across variants. The minimal differences came from the variants with unfixed subsampling of cells.

One initialisation seed ensures equal conditions for all hyperparameter variants, as long as the number and connection of nodes in the neural network do not change. Within one run consisting of many variants initialised with the same seed, the classification performance then only depends on the hyperparameters. This only holds as long as the initialised weights do not change when varying a hyperparameter. Adding nodes or layers anywhere in the network changes the number of weights, therefore classification performance is not only dependent on the hyperparameters but also on the initialisation. In contrast, different subsamplings, and also mean or maximum as special architectures do not change the initial weights.

As the initialisation was so influential, I suggest training each variant multiple times and using the best performing variant based on the performance on the biological samples of the validation dataset. I chose this procedure as the validation AUC correlated highly with the test AUC. The initialisation effect might get smaller when more biological samples are available, but the model performances using multiple seeds for the same hyperparameter settings should still be counter-checked.

Cell subsampling is a central preprocessing step for many algorithms[2,3,6,7,11,13,14]. It is not always necessary to have an equal number of cells for all biological samples, but subsampling is still done to reduce computational costs. Additionally, subsampling cells to a defined amount enables direct comparison of absolute cell numbers in a cell subpopulation across biological samples. In this work, I showed the difference between outlierness-based[2] and random subsampling, where neither showed superior results. However, there are two further methods: SPADE[19] used density-based downsampling, and in classical machine learning, furthest point sampling is proposed for point cloud classification[1]. Apart from these two additional downsampling methods, it might be interesting to train an FPP-based neural

network with a variable amount of cells without restricting it in the first place.

Subsampling can either be fixed or unfixed. Subsampling aims to get a defined number of cells for all biological samples. Fixed subsampling draws cells of each biological sample once before subsequent analysis. Unfixed subsampling draws cells of each biological sample every time this sample is used. Each biological sample is used multiple times during training the FPP neural network as all biological samples in the training set are used in every epoch to estimate the internal weights. After training consists of many epochs, each biological sample is used multiple times.

In addition to fixed and unfixed subsampling, subsampling can be random or based on outlierness. Random subsampling chooses random cells from all existing cells of a biological sample. Outlierness-based subsampling first calculates the outlierness of a cell which is proportional to the euclidean distance to other cells. Consequently, cells with higher outlierness are drawn more likely. Additionally, one part of cells can be subsampled randomly and the other part according to the outlierness. An already published work on predictive modelling in cytometry[2] already uses the concept of outlierness based on another publication[92].

By subsampling cells of biological samples, cells from rare but relevant cell subpopulations might get lost. Rare cell subpopulations are usually regarded as cell subpopulations with frequencies less than 5% or 1%[29,192–196] but even circulating tumour cells (CTCs) with only one to ten CTCs among ten billion normal blood cells are rare cell subpopulation leading to widespread metastasis[197]. Therefore, random cell subsampling of a biological sample loses rare cell subpopulations when the proportion of rare cells is less than the inverse of the number of sampled cells. Losing relevant cell subpopulations can render predictive modelling impossible.

Subsampling bias can avoid the loss of rare cell subpopulations but also increases signal noise. Outlierness-based subsampling increases the probability of selecting cells distant to other cells, therefore if a rare cell subpopulations has some distance to the major cell subpopulations it will be selected during subsampling. However, the signal noise increases as distant cells are not necessarily a cell subpopulation. Distant cells can be measurement errors in the cell-parameters and even if it was an actual cell subpopulation, this population is not necessarily relevant for your classification problem. An already published work on predictive modelling in cytometry[2] already applies the concept of outlierness which was introduced before[92].

In the first experiment I found that fixed or unfixed, random or outlierness-based subsampling of cells and drawing 2500, 5000 or 9500 cells had no significant effect on the AUC.

With unfixed subsampling, each prediction of a biological sample yields slightly different results. A subsample of n cells from a biological sample is never the same. Consequently, the prediction for that biological sample varies. We found that the AUC differences from worst to best AUC (the AUC range) when predicting all biological samples 50 times can be up to 31%. However, most times the AUC ranges were rather small about 5-10%.

The AUC range in the unfixed setting was dependent on the hyperparameters. More *Predictor* layers led to higher AUC ranges. The number of *FeatureLearner* layers also influenced the AUC ranges but was most stable with two layers. The

more cells per biological sample were chosen, the smaller was the AUC range. Training AUC generally varied less than validation and test AUC. If substantial AUC differences occurred, it was most times when the model did not perform well anyway compared to the other variants.

In the second experiment, I focused on the impact of architectural hyperparameters and found 2 *FeatureLearner* layers and no hidden layer for the *Predictor* performing best. It was irrelevant if 3 or 10 nodes were used in either part. Note that in contrast to the fact that neural networks are initialisation dependent and should be retrained multiple times, this specific network structure should be adjusted to your data.

In my current implementation of FPP, architectural hyperparameters are the number of dense layers, how many nodes per layer and the activation function after each layer for the *FeatureLearner* and *Predictor*. I used the mean or maximum over all subsampled cells per cell-feature as *Pooler*.

In the second experiment, the initialisation effect is still present but invisible. In contrast to the first experiment, the network architecture changes across variants. Consequently, the effective initialisation is different even if a manual seed was set beforehand.

In this part, I showed the impact of initialisation, architectural and subsampling hyperparameters. The initialisation had a surprisingly strong impact on the final performance, subsampling only affected the consistency of predictions, and the number of architecture layers was important and mean pooling performed consistently better than maximum pooling.

At the moment, hyperparameters must be searched manually. Here automated hyperparameter estimation methods like Optuna[198] or Nevergrad[199] in combination with already implemented hydra[160] parameter configuration package will be advantageous. A promising approach is a successive halving algorithm[200,201] where a set of hyperparameters is explored initially on a tiny part of the data to retain sets with better performance than the others - those are then trained on a bigger part of the data until arrival on the best hyperparameter setting.

Part III.

Identify discriminative cell subpopulations

1. Introduction

1.1. How to describe decisions of a FPP network

Cytometry characterises cells of a biological sample by cell-properties like size, complexity or surface markers. A measured cell-property is called a cell-parameter, which defines each cell by numerical values suitable for further analysis. Up to 50 measured cell-parameters characterise each cell[5], and the resulting data is a matrix of unordered cells in the rows, and some cell-parameters as columns. A biological sample contains millions of cells, and their characterisation can be used to define cohesive cell subpopulations which reflect certain anatomic, morphologic or functional types of cells[42].

Predictive modelling in cytometry assigns a class label to a biological sample according to all measured cells of that sample. This class enables clinical patient decisions when the class is known to respond to a certain treatment. Due to the unordered nature of cytometry measurements, only the presence or quantity of cells define the class of a biological sample but not which cell was measured first.

There are multiple approaches for predictive modelling in cytometry which can be grouped into three concepts. First, partition and count cells based on the cell-parameters followed by predictive modelling[7,9–11]. Second, apply unsupervised cell subpopulation identification followed by predictive modelling[7,13–17,19,22–24,26–28,30–32,39]. Third, use all cells without previous summarisation of the cells[2,3,6].

In the first part of this work, I introduced the FPP concept, a concept to organise neural networks for predictive modelling in cytometry. Multiple cells of a biological sample are used together to predict the class the biological sample. FPP consists of three parts. First, the *FeatureLearner* transforms the measured matrix of a biological sample into a new matrix with new, predictive cell-features. Second, the *Pooler* aggregates these cell-features into a composition vector of values per biological sample. Third, the *Predictor* uses this composition vector to predict the final class.

The prediction of a biological sample is useful, but explaining the prediction yields additional value. In cytometry, the cell subpopulations responsible for the prediction are interesting. Classical gating makes interpretation easy: Sequential manual gating uses manually chosen single values, rectangles, circles or polygons as cutoffs for the cell-parameters to group cells into cell subpopulations. After finding which cell subpopulations were predictive, the interpretation is known per definition. The identified cell subpopulations can give information about biological processes important for the prediction.

Cell subpopulations derived from sequential manual gating are nodes of decision trees. In Figure 1.1 (A) I show the gating of CD45RA positive, CCR7 positive naive T-cells as introduced in subsubsection 1.4.2.1. This gating can be translated into a decision tree structure by splitting each rectangle in two single cutoffs of the respective cell-parameters. The two cutoffs describe four rectangles, where one

corresponds exactly to the original rectangle. The final chosen cell subpopulation of the decision tree can be described through the leaves of the decision tree, intermediate cell subpopulations by intermediate nodes of the tree.

A decision tree splits the data according to certain cutoffs in the feature. In the case of cell subpopulation identification, each cell traverses the decision tree where the features are the cell-parameters. The cutoffs are obtained based on supervised learning, where the value for some initial cells is known. The value for the cells can be discrete cell subpopulations or also plain numerical values, which leads to a regression problem for the decision tree. In the discrete case, a predicted cell subpopulation is defined as the major cell subpopulation in the training data of the predicted node of the cell. In the case of regression, the predicted value is defined as the mean of all cells inside the predicted node.

After training, I replace each cell-feature from the *FeatureLearner* with a decision tree. Sequential manual gating is well established, decision trees are a suitable replacement, and decision trees are easily trained. In contrast, the neural network of the *FeatureLearner* is not directly understandable. However, the *FeatureLearner* can calculate the output for each cell: The results are k cell-features per cell. I propose to replace each cell-feature with a decision tree. This is only an approximation of the actual neural network but gives an idea about the behaviour for all cells. Based on these decision tree replacements, I can identify gatings. These gatings arise by traversing the decision trees towards nodes with extreme responses.

Possibly, only some of the cell-features are necessary for the final classification of a biological sample. Therefore, I will introduce SHAP values. These SHAP values inform about the importance of a pooled cell-feature. The important cell-features yield consequently the important decision trees.

In this part, I introduce SHAP values and how to derive them for neural networks. I then show how to find and apply gatings given a FPP concept based neural network in simulation for binary and multiclass classification.

1.1.1. Shapley values

Shapley values[202] are a method from coalitional game theory where the goal was to determine the contribution of each player inside a coalition.

Shapley referred to a coalitional game[203] (N, v), where $N = \{1, ..., n\}$ is a set of players and $v : 2^N \to \mathbb{R}$ is a function to assign each coalition of players $S \subseteq N$ a real number. $v(\emptyset) = 0$. Shapley introduced marginal contributions ϕ_i of a player $i \in N$ to a coalition $S \subseteq N \setminus \{i\}$ defined as the difference to the complete coalition's worth after player i joins: $v(S \cup \{i\}) - v(S)$. After many coalitions are possible (the power set of N), Shapley concluded that the value of a player i is the weighted average marginal contribution of i over all possible coalitions including i. The weighting can be interpreted as the probability that a coalition of a certain size forms. Mathematically, this can be expressed by the following formula:

$$\forall i \in N : \phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{s!(n-s-1)!}{n!} \left(v(S \cup \{i\}) - v(S)\right) \tag{1.1}$$

where $s = |S|$ and $n = |N|$.

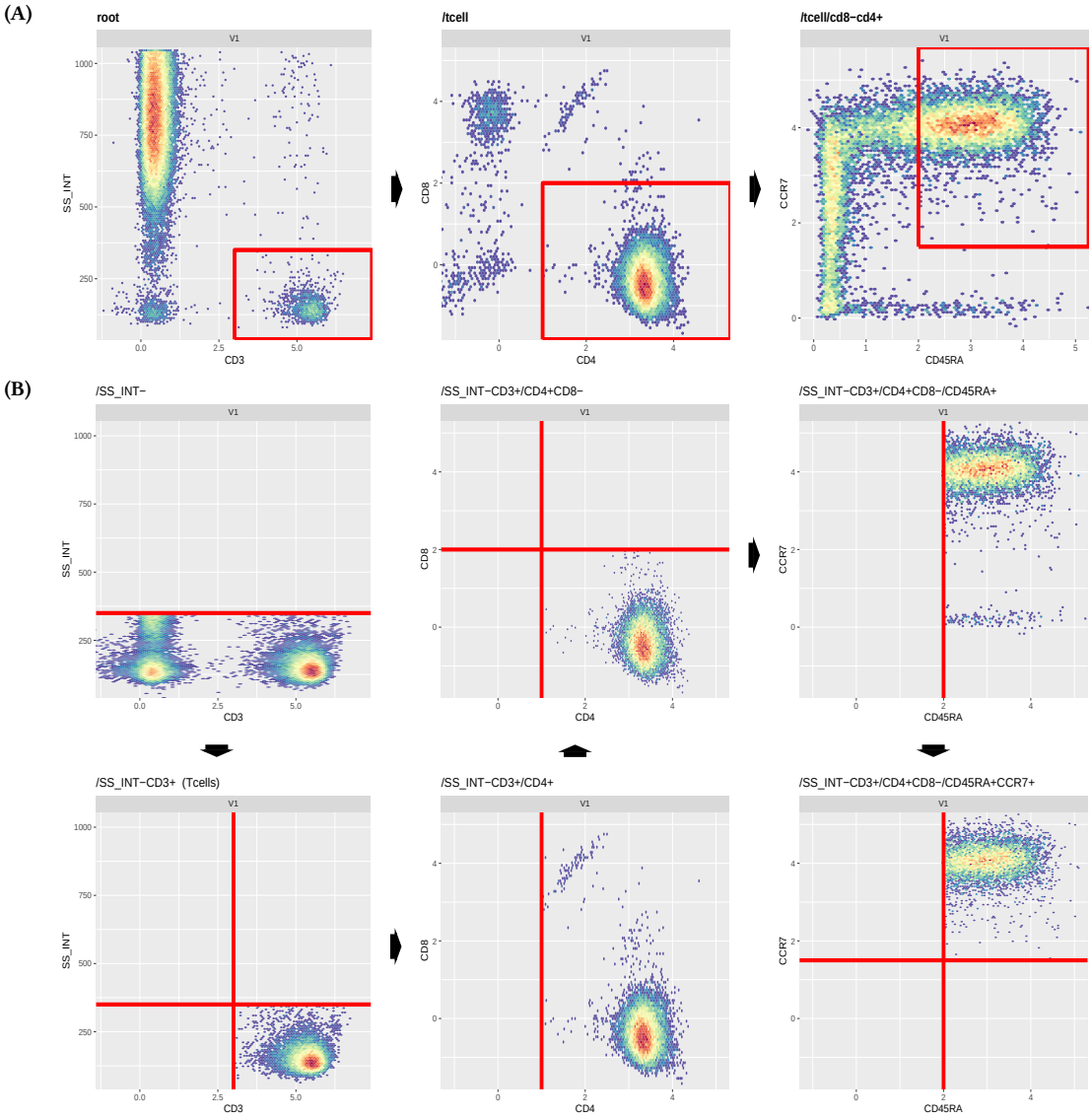


Figure 1.1.: (A) The exemplary gating starts with a pre-gated sample of CD45+ singlets (pre-gates not shown). In the leftmost picture, only CD3+ cells reflecting T-cells are chosen. Next, CD4+ and CD8- cells of the previously selected CD3+ cells are chosen and finally CD45RA+/CCR7+ (naive) cells inside the rectangle.
(B) A decision tree needs binary cutoffs. Rectangle-gatings can be directly replaced with decision trees using multiple steps with the same cutoffs. Follow the arrows for the gating steps. Start by selecting only SS_INT$^-$ cell in the top left plot. Then select CD3$^+$ cells. As in (A), the same cell subpopulation of SS_INT$^-$|CD3$^+$ cells are selected. But now, it is selected by two binary cutoffs instead of a single rectangle. Inside those SS_INT$^-$|CD3$^+$ cells, select CD4$^+$ cells, then CD8$^-$. Now, the same cells as in the second step of (A) are selected. Finally, select CD45RA$^+$ and CCR7$^+$ cells. This selected CD45RA+/CCR7+ (naive) T cells, as in (A).

In Table 1.1 I show a tabular example of how the Shapley values can be calculated for three players A, B and C. In subtable (b), we see each coalition's worth $v(S)$, immutable to the order in which A, B and C entered the coalition. In subtable (a), we see the added value of each player when introduced in all possible orders.

(a)

Order	$v(S) - v(S \setminus \{i\})$		
	First	Second	Third
$C; A; B$	$S = \{C\}\,; i = C$	$S = \{C, A\}\,; i = A$	$S = \{C, A, B\}\,; i = B$
	$v(\{C\}) = 1$	$v(\{C, A\}) - v(\{C\}) = 3 - 1$	$v(\{C, A, B\}) - v(\{C, A\}) = 11 - 3$

(b)

S	$v(S)$
$\{A\}$	1
$\{B\}$	2
$\{C\}$	1
$\{A, B\}$	5
$\{A, C\}$	3
$\{B, C\}$	5
$\{A, B, C\}$	11

(c)

Order	Contribution		
	A	B	C
$A; B; C$	1	5-1=4	11-5=6
$A; C; B$	1	11-3=8	3-1=2
$B; A; C$	5-2=3	2	11-5=6
$B; C; A$	11-5=6	2	5-2=3
$C; A; B$	3-1=2	11-3=8	1
$C; B; A$	11-5=6	5-1=4	1

Table 1.1.: Shapley value example. (b) shows exemplary worth $v(S)$ for each possible set S of the three players A, B and C. (a) shows how to calculate the contribution parts taking the order of players into account. (c) shows the actual contributions per player.

The Shapley values satisfy the properties Efficiency, Symmetry, Dummy and Additivity. Together they can be considered a definition of fair payout[204].

P1 Efficiency: The sum of all Shapley values of a player equals the complete coalition:

$$\sum_{i \in N} \phi_i(v) = v(N) \tag{1.2}$$

P2 Symmetry: If two players contribute equally to all coalitions, their Shapley values are identical:

$$\forall S \subseteq N \setminus \{i, j\} v(S \cup (i)) = v(S \cup (j)) \Rightarrow \phi_i(v) = \phi_j(v) \tag{1.3}$$

P3 Dummy: Also called null-player; A player's without zero effect for all coalitions is zero:

$$\forall S \subseteq N \setminus \{i\} : v(S \cup \{i\}) = v(S) \Rightarrow \phi_i(v) = 0 \tag{1.4}$$

P4 Additivity: For any pair of games v and w, the Shapley values are additive:

$$\forall i \in N : \forall S \subseteq N : (v + w)(S) = v(S) + w(S) \Rightarrow \phi_i(v + w) = \phi_i(v) + \phi_i(w) \tag{1.5}$$

1.1.2. Shapley values in machine learning

In machine learning, the concept of players can be replaced by features and a coalitions worth with the prediction for those features. A nice introduction[205]

uses linear regression as starting point. Let f have the following form with $p \in \mathbb{N}$ features and $x \in \mathbb{R}^p$

$$f(x) := \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \tag{1.6}$$

The contribution of the i-th feature to the prediction for an instance x is the model prediction against the expected prediction when the feature value is unknown. If the feature value is unknown, the best you can do is to replace it by its expected value, thus:

$$\begin{aligned} \phi_i\left(f(x)\right) &= f(x) - f(x|x_i = E[X_i]) && (1.7)\\ &= \left(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p\right) && (1.8)\\ &\quad - \left(\beta_0 + \beta_1 x_1 + \cdots + \beta_i E[X_i] + \cdots + \beta_p x_p\right) \\ &= \beta_i \left(x_i - E[X_i]\right) && (1.9) \end{aligned}$$

However, the independence of all other features is not true for other than additive models. Therefore every subset of features (2^p) must be taken into account[205]:

$$f_S(x) = E[f|X_i = x_i, \forall i \in S] \tag{1.10}$$

where $S \subseteq N = \{1, 2, \ldots, p\}$ is a subset of the set of all features N. f_S defines the predicted value of f when only the subset S of the p features are known. Note that I use the same nomenclature as introduced in the previous section for the plain Shapley values from game theory. So S is always a subset of N. From this, the contribution of each feature subset S can be defined:

$$\Delta_S(x) = f_S(x) - f_\emptyset(x) \tag{1.11}$$

where $f_\emptyset(x) = E[f]$. To bridge to the Shapley values, it is necessary to introduce "interactions" to represent the subset contribution:

$$\Delta_S(x) = \sum_{W \subseteq S} \mathcal{I}_W(x) \tag{1.12}$$

where each interaction $I_S(x)$ can be defined recursively and using Equation 1.11:

$$\mathcal{I}_S(x) = \Delta_S(x) - \sum_{W \subset S} \mathcal{I}_W(x) \tag{1.13}$$

Mind $\subseteq$ in Equation 1.12 but $\subset$ in Equation 1.13. Then the contribution of feature i can be defined by the interaction, split across all participating feature values:

$$\phi_i(x) = \sum_{W \subseteq N \setminus \{i\}} \frac{\mathcal{I}_{W \cup \{i\}}(x)}{|W \cup \{i\}|} \tag{1.14}$$

This finally leads to an explicit definition[205,206]

$$\phi_i(x) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left(\Delta_{S \cup \{i\}}(x) - \Delta_S(x)\right) \tag{1.15}$$

which is the Shapley value for feature i as described earlier for players in Equation 1.1.

All four properties described earlier stay the same, essentially replacing the worth with the prediction and the players with features - except for the efficiency:

P1′ Efficiency: The sum of all Shapley values of all features equals the difference of the prediction and its expected value:

$$\sum_{i \in p} \phi_i = \Delta_N(x) \tag{1.16}$$

$$= f_N(x) - f_\emptyset(x) \tag{1.17}$$

$$= f(x) - E_X\left[f(X)\right] \tag{1.18}$$

P2 Symmetry: If two features have a symmetrical impact across all subsets, their Shapley values are identical:

$$\forall S \subseteq N \setminus \{i, j\} \Delta_{S \cup \{i\}} = \Delta_{S \cup \{j\}} \Rightarrow \phi_i(x) = \phi_j(x) \tag{1.19}$$

P3 Dummy: A feature i without impact on the prediction will be assigned a 0 contribution:

$$\forall S \subseteq N \setminus \{i\} : \Delta_{S \cup \{i\}} = \Delta_S \Rightarrow \phi_i(x) = 0 \tag{1.20}$$

P4 Additivity Contributions are additive across instances x, y

$$\forall i \in N : \forall S \subseteq N : \Delta_S(x + y) = \Delta_S(x) + \Delta_S(y) \tag{1.21}$$

$$\Rightarrow \phi_i(x + y) = \phi_i(x) + \phi_i(y) \tag{1.22}$$

1.1.3. SHAP values

The Shapley values themselves are a theoretical, combinatorial construct, often impossible to feasibly calculate. Therefore, different calculation methods and approximations have been proposed[205,207–211]. A recent publication proposed SHAP-values (SHapley Additive exPlanation) as a unified measure of feature importance.

Let f be the actual predictive model and g an explanation model. Your predictive model might be a very complex neural network, but your explanation model is an interpretable approximation. Local explanation models were proposed in LIME[210], and aim to interpret an individual model prediction based on local approximations around this prediction.

So the aim is to explain $f(x)$ for a single input $x \in \mathbb{R}^p$ of p features, here exemplary all real numbers. The explanation model g might use a "simplified input" x' defined by a mapping function $h_x(x') = x$. Local methods approximate $g(z') \approx f\left(h_x(z')\right)$ when $z' \approx x'$. Then an additive feature attribution method is defined by a explanation model linear in binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{1.23}$$

where $z' \in \{0,1\}^M$ where M is the number of simplified input features and $\phi_i \in \mathbb{R}$. Think of this binary, simplified input feature vector as the coalition vector from Shapley values, M as the maximum coalition size and $\phi_i \in \mathbb{R}$ as the feature attribution of feature i. The binary coalition vector tells if the corresponding feature value is present or absent in the current coalition (I used S for this previously)[204].

Additive feature attribution methods result in a single unique solution with three properties:

S1 Local accuracy: The explanation model $g(x')$ is the actual model $f(x)$ when $h_x(x') = x$

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{1.24}$$

If all features are "enabled" or "present", $x' = \{1\}^p$ and $h_x(x') = x$. If all features are "disabled" or "missing", $x' = \{0\}^p$ and consequently $g(x') = \phi_0$. When $\phi_0 := E_X[f(x)]$ and $x' = \{1\}^p$ this is the same as the efficiency property P1′.

S2 Missingness: A missing feature in the simplified input should have no impact

$$x'_i = 0 \Rightarrow \phi_i = 0 \tag{1.25}$$

S3 Consistency: Let $f_x(z') = f(h_x(z'))$ and $z' \setminus i$ denote $z'_i = 0$. For two models f and f',

$$\forall z' \in \{0,1\}^M : f'_x(z') - f'_x(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i) \Rightarrow \phi_i(f', x) \geq \phi_i(f, x) \tag{1.26}$$

for the marginal contributions of feature i for model f : $\phi_i(f, x)$ and for model f' : $\phi_i(f', x)$ of the instance x.

They[212] also show (in the supplement) that S3 implies Symmetry (P2), Dummy (P3) and Additivity (P4) properties described previously. Finally, the unique explanation model g following the definition of additive feature attribution methods (Equation 1.23) and the properties S1, S2 and S3 are Shapley values:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!\,(M - |z'| - 1)!}{M!} \left(f_x(z') - f_x(z' \setminus i)\right) \tag{1.27}$$

where $|z'|$ is the number of non-zero entries in z'. $(z' \subseteq x')$ represents all z' vectors whose non-zero entries are a subset of the non-zero entries in x'. As defined earlier; x' is the specific simplified input vector such that $h_x(x') = x$. In the context of coalition games, this vector x' would be a vector consisting of only ones (all players are part of the coalition) for each player index $\{1, \ldots, N\}$ of all possible players n.

1.1.4. Deep SHAP

There are multiple ways to calculate exact or estimate these SHAP values - but I use only one, applicable to neural networks. Deep SHAP[212] is based on the connection between Shapley value and DeepLIFT[213]. DeepLIFT is a recursive prediction explanation method for deep learning[211,213] where the explanation comes

from the difference in output from some reference input to the actual input feature value:

$$\sum_{i=1}^{n} C_{\Delta_{x_i}\Delta_o} = \Delta_o \tag{1.28}$$

where $o = f(x)$ is the model output, $\Delta_o = f(x) - f(r)$, $\Delta_{x_i} = x_i - r_i$ and r is the reference input.

When $\phi_i := C_{\Delta_{x_i}\Delta_o}$ and $\phi_0 = f(r)$, we see with Equation 1.23:

$$\sum_{i=1}^{n} C_{\Delta_{x_i}\Delta_o} = \Delta_o \tag{1.29}$$

$$\sum_{i=1}^{n} \phi_i = f(x) - f(r) \tag{1.30}$$

$$\sum_{i=1}^{n} \phi_i = f(x) - \phi_0 \tag{1.31}$$

$$f(x) = \phi_0 + \sum_{i=1}^{n} \phi_i \tag{1.32}$$

$$\tag{1.33}$$

that this is again an additive feature attribution method. In another publication[214], the authors show that in a pure linear neural network (only one hidden layer without activation functions), the SHAP values for every input neuron can be calculated exactly. For a linear neural network with two inputs x_1 and x_2 which are densely connected to two neurons which are again densely connected to one output neuron (see Figure 1.2), the activations can be calculated as follows:

$$h_1^j = w_{1j}^{(1)} x_1 + w_{2j}^{(1)} x_1 \tag{1.34}$$

$$y = w_1^{(2)} h_1^1 + w_2^{(2)} h_1^2 \tag{1.35}$$

The exact SHAP values for an input x_i can be calculated by summing the attributions along all possible paths between that input neuron x_i and the model output y. For simplicity, the authors redefined $\phi(\cdot)$ as the attribution value. $\phi(x_i)$ is then the product of weights along the path with the difference between the feature value f_{x_1} to a background value b_{x_1}:

$$\phi(x_i) = w_2^{(2)} w_{12}^{(1)} (f_{x_1} - b_{x_1}) \tag{1.36}$$

As in the chain rule, the attribution for x_1 can also be written using the intermediate node attributions:

$$\phi(h_1^2) = w_2^{(2)} (f_{h_1^2} - b_{h_1^2}) \tag{1.37}$$

$$\phi(x_1) = \frac{\phi(h_1^2}{f_{h_1^2} - b_{h_1^2}} w_{12}^{(1)} (f_{x_1} - b_{x_1}) \tag{1.38}$$

When a non-linear activation function $g()$ is added as additional layer, shown in

Figure 1.3, the attribution value for $\phi(g)$ becomes $g(f_h) - g(b_h)$ as SHAP values maintain local accuracy and g is a function with a single input. The network activations can be defined as

$$h = \sum_{i=1}^{k} w_i x_i \tag{1.39}$$

$$y = g(h) \tag{1.40}$$

thus the feature attributions become

$$\phi(x_i) = \frac{\phi(h)}{f_h - b_h} w_i (f_{x_i} - b_{x_i}) \tag{1.41}$$

As SHAP values maintain only local accuracy, this is only an approximation but makes 1) backpropagation feasible and 2) at least locally accurate.

These simple network components and their SHAP values are already sufficient to recursively build deep networks and have a (fast) approximation for the complete network when dense layers and non-linear activation functions are combined. For more complex network types, this can also be generalised as long as the SHAP values can be either computed or approximated for the individual component.

Regarding the background value b_{x_i}, they[214] show that SHAP values should be obtained for each baseline in the background distribution and average over all resulting attributions. In the documentation of the python package shap (https://shap-lrjball.readthedocs.io/en/latest/generated/shap.DeepExplainer.html) the authors propose 100 background samples for a good estimate and 1000 samples for a very good estimate of the background values. The samples are chosen randomly from the training dataset due to calculation time. In principle, the complete training set would be usable.

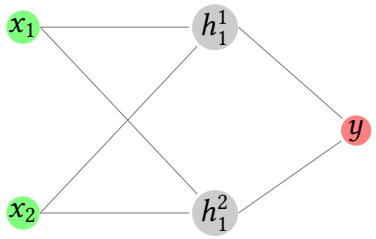


Figure 1.2.: SHAP: Linear network with two inputs x_1 and x_2 and no activation function.

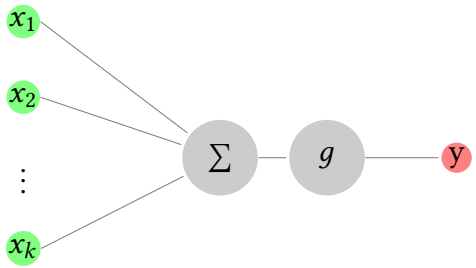


Figure 1.3.: SHAP: Linear network with two inputs x_1 and x_2 and an activation function $g()$.

2. Methods

I introduce a simulated binary and multiclass classification problem. The resulting datasets were used to show the behaviour of my explanation approach for FPP concept based neural networks. A complete analysis of this simulation is shown in Part IV.

2.1. Binary classification simulation

In this simulation, each simulated biological sample contains 10^5 cells with two cell-parameters each. The cells are drawn from the following three multivariate normal distributions with base odds $(1, 1, 3)$ for $(\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3)$.

$$\begin{gathered} \mathcal{N}_1(\begin{pmatrix}4\\0\end{pmatrix}, \sigma^2) \quad \mathcal{N}_2(\begin{pmatrix}4\\4\end{pmatrix}, \sigma^2) \quad \mathcal{N}_3(\begin{pmatrix}0\\0\end{pmatrix}, \sigma^2) \\ \sigma^2 := \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix} \end{gathered} \tag{2.1}$$

The simulation consists of varying the odds of a cell being in $\mathcal{N}_1$ such that the percentage of cells is either 0% (p_0) or 50% ($p_{0.5}$). For each percentage-class I create 100 biological samples of 10^5 cells.

2.2. Multiclass classification simulation

As in the simulation for binary classification, each generated biological sample contains 10^5 cells with two cell-parameters each. The cells are drawn from the same three multivariate normal distributions (see Equation 2.1) but now the percentage of the 10^5 cells inside drawn from $\mathcal{N}_1$ varies over the following values:

$$\{0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.2, 0.5\} \tag{2.2}$$

3. Results

3.1. Simulation

3.1.1. Binary classification

The binary classification explained in section 2.1 simulated biological samples with 10^5 cells drawn from three multivariate normal distributions with base odds $(1, 1, 3)$ for $(\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3)$. Two classes of biological samples were simulated such that the percentage of cells drawn from $\mathcal{N}_1$ is either 0% (p_0) or 50% ($p_{0.5}$).

All simulated cells of one exemplary simulated biological sample per class are shown in Figure 3.1. The two cell-parameters are called f_0 and f_1.

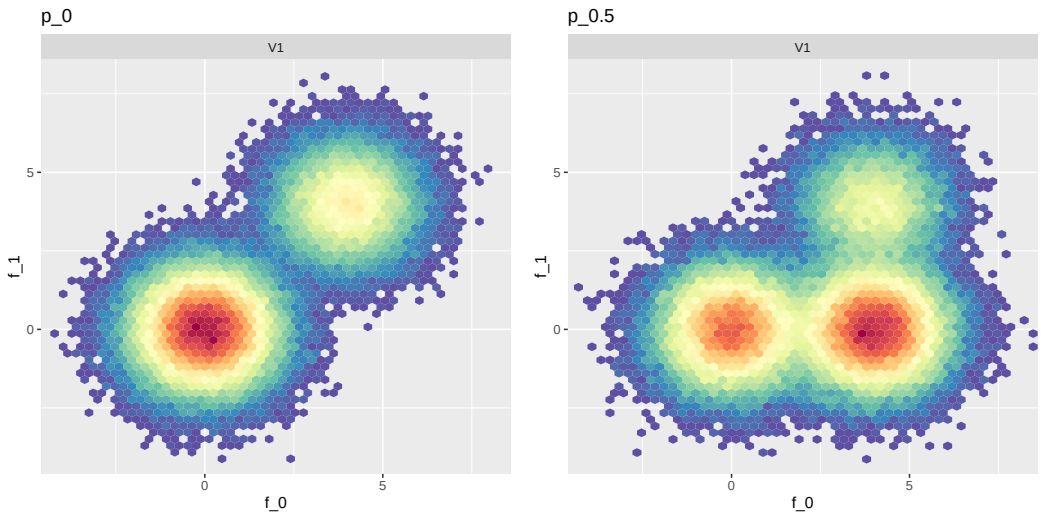


Figure 3.1.: Binary classification simulation data, one exemplary simulated biological sample from class p_0 (left) and one exemplary biological sample from class $p_{0.5}$ (right). All 10^5 cells are shown.

3.1.2. Multiclass classification

The multiclass classification explained in section 2.2 simulated biological samples with 10^5 cells drawn from three multivariate normal distributions. The multiple classes of biological samples with base odds $(1, 1, 3)$ for $(\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3)$. Eight classes of biological samples were generated where the percentage of cells drawn from $\mathcal{N}_1$ was varied.

All simulated cells of one exemplary simulated biological sample per class are shown in Figure 3.2.

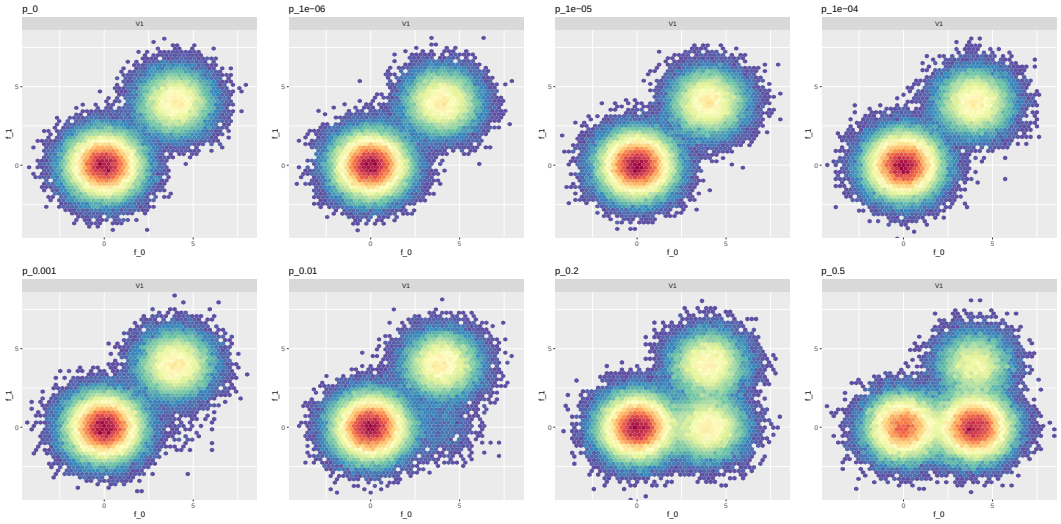
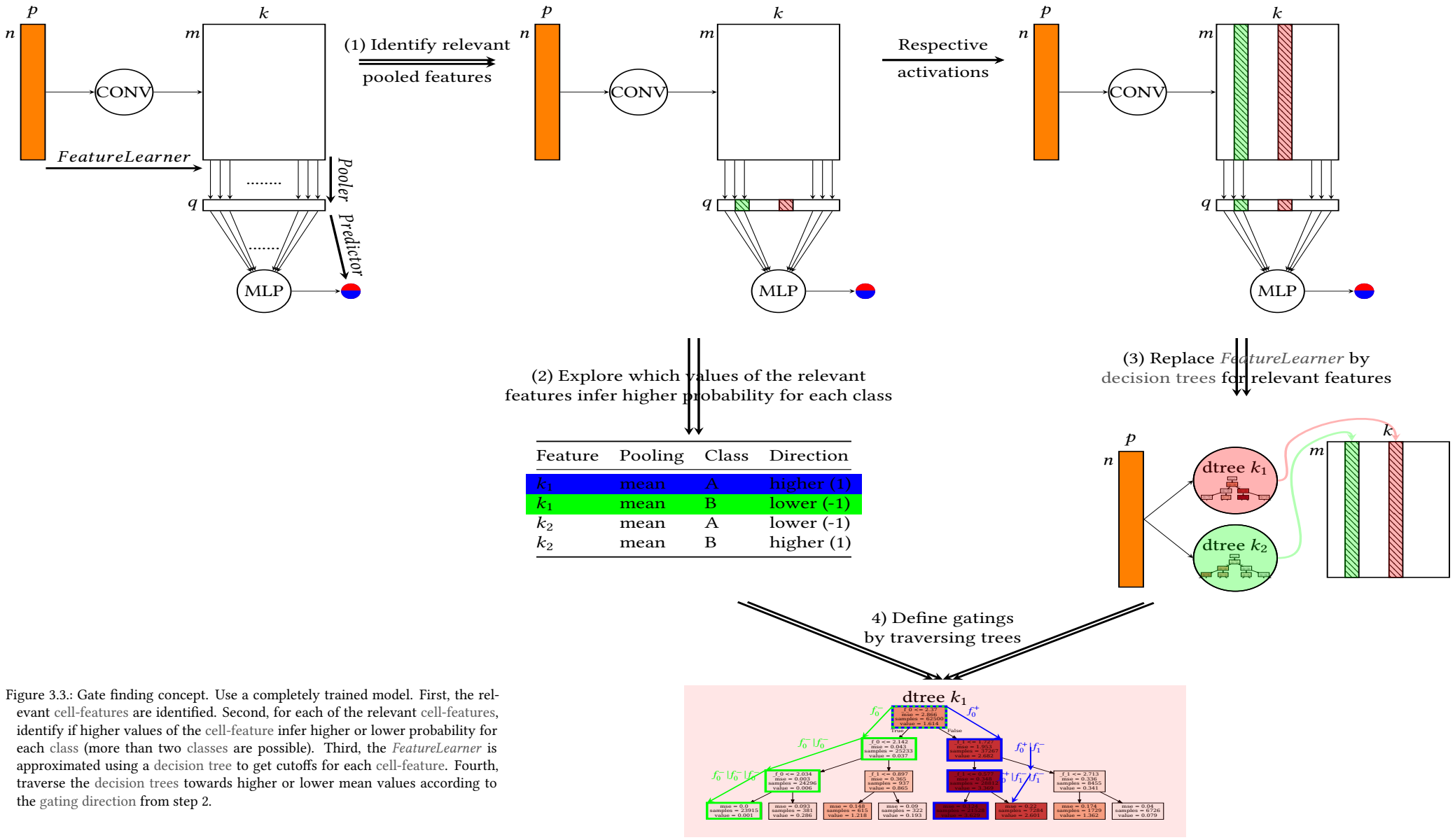


Figure 3.2.: Multiclass classification simulation data, one exemplary biological sample from each class where all 10^5 cells are shown. Going from top-left to bottom-right to higher percentages of $\mathcal{N}_1$.

3.2. Gate finding

After training an FPP model successfully, the cell subpopulation responsible for good performance should be identified. I implemented the following approach for binary and multiclass classification.
At a high level, gate finding includes four steps, see also item 3.2:

1. Identify the relevant subset $R \subseteq \{1, \ldots, k\}$ of k pooled cell-features of the *Pooler*
2. Explore which values (higher or lower) of the respective cell-features of R infer higher probabilities for each class
3. Replace the *FeatureLearner* with a decision tree for all relevant cell-features (R)
4. Define the gating by traversing the decision trees towards higher or lower cell-feature-values identified by 2)



Feature	Pooling	Class	Direction
k_1	mean	A	higher (1)
k_1	mean	B	lower (-1)
k_2	mean	A	lower (-1)
k_2	mean	B	higher (1)

Figure 3.3.: Gate finding concept. Use a completely trained model. First, the relevant cell-features are identified. Second, for each of the relevant cell-features, identify if higher values of the cell-feature infer higher or lower probability for each class (more than two classes are possible). Third, the *FeatureLearner* is approximated using a decision tree to get cutoffs for each cell-feature. Fourth, traverse the decision trees towards higher or lower mean values according to the gating direction from step 2.

3.2.1. Relevant subset identification

To identify the relevant subset := R of pooled cell-features from the *FeatureLearner*, I propose the usage of SHAP values[202,212].

The basis for calculating the SHAP values are the pooled cell-features. In Figure 3.4 I show an example of the pooled cell-features (columns) for each biological sample (rows) from the binary classification simulation. The first two columns are the true and predicted class of the simulated biological samples. After those columns, the actual pooled cell-feature values are shown, and I start counting columns there. The following three columns refer to the cell-features k_0, k_1 and k_2 pooled with the maximum.

Figure 3.4 shows little difference in the maximum pooled values between biological samples of red and blue class. In contrast, the same cell-features pooled with the mean in the next three columns correlate perfectly with class affiliation. For cell-features k_0 and k_2, high values speak for class p_0, for cell-feature k_1 high values speak for class $p_{0.5}$.

I use Deep SHAP[212] which combines DeepLIFT[205,211] with Shapley values[202]. To calculate the SHAP values the distribution of reference biological samples needs to be estimated. I use the pooled cell-features values of all training samples.

The Deep SHAP[212] implementation I use returns a tensor of SHAP values with the same shape as the input for each output node of the neural network. So for an input matrix $P \in \mathbb{R}^{q_{poolings} \times k_{cellfeatures}}$ also the SHAP values are of shape $SHAP \in \mathbb{R}^{q_{poolings} \times k_{features}}$. If there are $c \in \mathbb{N}_{>1}$ output nodes (in multiclass classification)), the same input matrix results in a list of SHAP values $\left\{SHAP^{(i)} | i \in \{1, \ldots, c\}\right\}$ where $SHAP^{(i)}$ reflects the SHAP values of the i-th class.

After the SHAP values are calculated, they can be visualised in a beeswarm plot, see Figure 3.5. Cross check with the pooled values in Figure 3.4: The maxima of all three cell-features (max:k_0, max:k_1 and max:k_2) have the lowest SHAP values. mean:k_1 has the highest SHAP values, so the *Predictor* mainly uses that feature to predict the classes of a biological sample. A point with positive SHAP values denotes that the biological sample' value of this pooled cell-feature increases the probability of that biological sample being in the class $p_{0.5}$.

The most relevant subset of cell-features can be identified by choosing cell-features with high SHAP values across biological samples and classes. Mathematically speaking, I sum the absolute SHAP values over all c classes and subsequently calculate the mean over all N biological samples. Finally, I aggregate them over all poolings per cell-feature j:

$$score_j = \sum_{l=1}^{q} \frac{1}{N} \sum_{m=1}^{N} \sum_{i=1}^{c} |SHAP_{lj}^{(i,m)}| \tag{3.1}$$

I include the highest-scoring cell-features until their score sum $\sum_{j \in R} score_j$ is at least 90% of the total score sum $\sum_j^k score_j$.

In Figure 3.6 I show summary plots for the multiclass classification simulation where subfigure (A) shows the absolute (sample-)mean SHAP values for each pooling : cellfeature combination. Subfigure (B) shows the accumulated values per cell-feature over all poolings as described in Equation 3.1. Based on (B), cell-feature

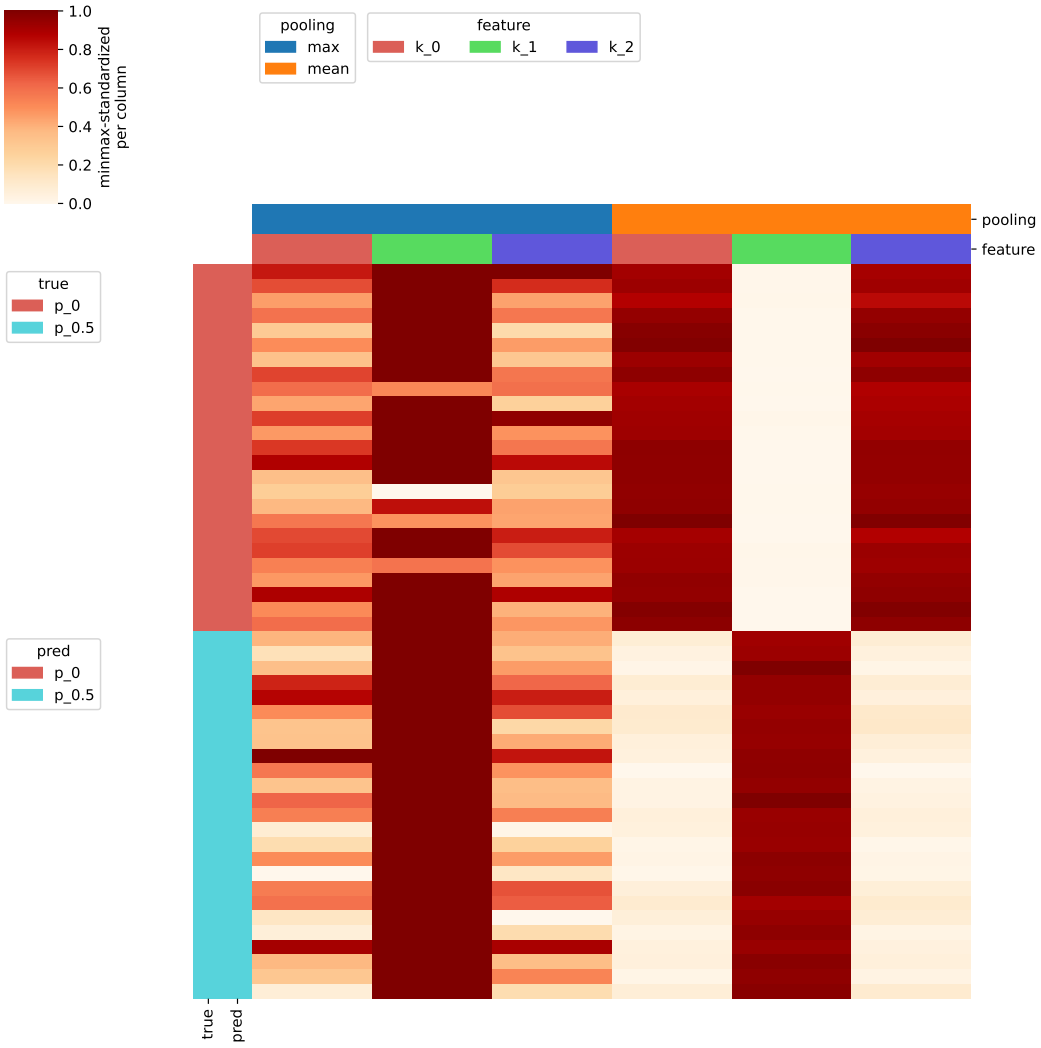


Figure 3.4.: Pooled cell-features for binary classification after training. The first two rows give information about the pooled cell-feature and the applied pooling. In each further row, the respective pooled values for one biological sample are shown. The first two columns show the true and predicted label of each biological sample, all further columns the pooling of each respective cell-feature. Each column is minmax-standardised. In the first three columns, the maxima of k_0, k_1 and k_2, have only little predictive value. The second three columns, the means of k_0, k_1 and k_2, have perfect predictive value as $mean(k_0)$ and $mean(k_2)$ are always higher in the p_0-class and $mean(k_1)$ is always higher in the $p_{0.5}$-class.

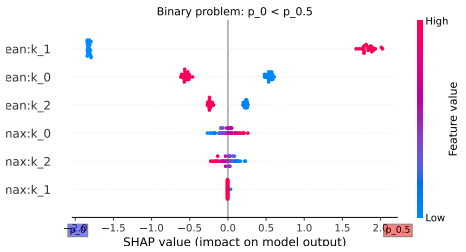


Figure 3.5.: Each row reflects one input to the *Predictor*, so each possible pooling: cellfeature combination. Inside each cell-feature, each point is one biological sample. The colour of each point denotes its pooled cell-feature value. The position on the x-axis, the SHAP-value, describes the impact on the prediction; I show the score before applying the sigmoid function. A point with positive SHAP values denotes that the biological sample' value of this pooled cell-feature increases the probability of that biological sample being in the class $p_{0.5}$.

Table 3.1.: SHAP-based gating directions for simulation A.

Feature	Pooling	Class	Direction
k_0	mean	$p_{0.5}$	lower -1
k_1	mean	$p_{0.5}$	higher 1
k_2	mean	$p_{0.5}$	lower -1
k_0	max	$p_{0.5}$	higher 1
k_1	max	$p_{0.5}$	lower -1
k_2	max	$p_{0.5}$	lower -1

k_1 would be included first, then cell-feature k_0, which is when the 90% cutoff would be reached and no more cell-features included in R.

3.2.2. Gating directions

Which cell-feature-values increase the probability of each class? For this, I introduce the nomenclature shown in Table 3.2. Each cell-feature plus pooling combination is the input for the *Predictor*. If informative, high values of a cell-feature can speak for (pro) or against (contra) each possible class. The combination of pro/contra and the lower/higher direction results in the gating direction of one specific input feature of the *Predictor*, the pooled cell-features. In the following, I show how to identify the gating directions.

Cell-feature	Pooling	Class	pro/contra	Direction
k_0	mean	A	pro	higher
k_0	mean	A	pro	lower
k_0	mean	A	contra	higher
k_0	mean	A	contra	lower

pro/contra	Direction	Explanation
pro	higher	Higher values of $mean(k_0)$ increase the probability for class A
pro	lower	Lower values of $mean(k_0)$ increase the probability for class A
contra	higher	Higher values of $mean(k_0)$ decrease the probability for class A
contra	lower	Lower values of $mean(k_0)$ decrease the probability for class A

Table 3.2.: Gating direction nomenclature, exemplary for feature k_0, mean-pooling of that cell-feature and for class A. Split into two rows to properly fit the page.

To find the gating directions in binary classification, it is necessary to know if pooled cell-feature values at positive SHAP values are high or low. If they are high, high pooled cell-feature values speak **for** the positive class (in the simulated binary classification $p_{0.5}$). If they are low, low pooled cell-feature values speak **for** the

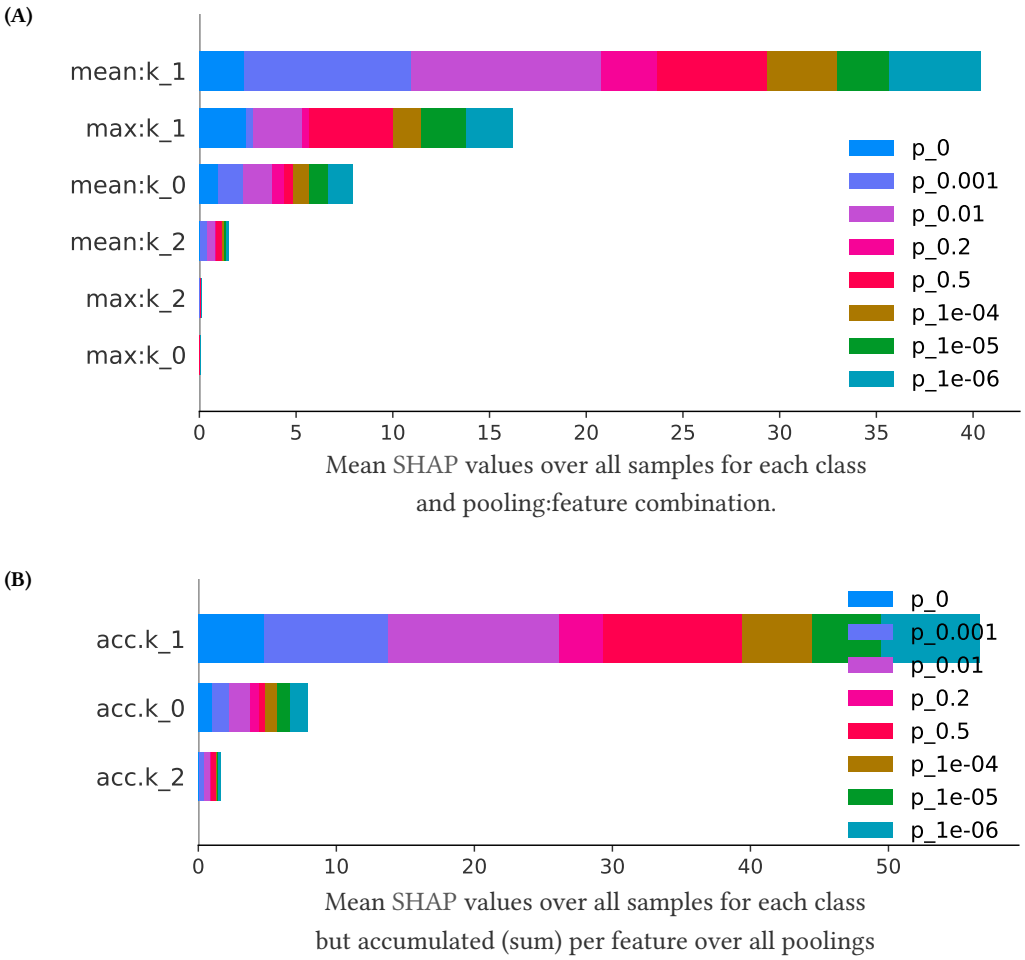


Figure 3.6.: (A) Mean SHAP values over all simulated biological samples for each SHAP and pooling : cellfeature combination. The mean of cell-feature k_1 has the strongest absolute effect summed across all classes, and neither $max(k_0)$ nor $max(k_2)$ have an effect on the prediction. (B) SHAP values from subfigure A, but all poolings are accumulated per Cell-feature ($acc.k_i$). Cell-feature k_1 mainly drives the prediction for all classes, cell-features k_0 and k_2 have a comparably weak influence.

positive class $p_{0.5}$. Each time, the opposite (low/high) speaks for the negative class p_0. To computationally identify these directions, **SHAP split** splits the cell-features according to positive and negative SHAP values and finds which cell-feature means are higher. Alternatively, calculate the **correlation** of SHAP and cell-feature values.

In multiclass classification, **SHAP split** or **correlation** are not sufficient. Exemplary, let a cell-feature's high values speak for class A, so high cell-feature values correspond to positive SHAP values. Let the low cell-feature values correspond to very high, but negative SHAP values, higher than the highest positive SHAP value. From SHAP split or correlation, this cell-feature is pro class A, and the direction is "higher". But after the negative SHAP values actually dominate, the appropriate gating direction would be: The cell-feature speaks **contra** A and the direction is still "higher". Consequently, I identify the maximum absolute SHAP value. If this value is positive, the cell-feature speaks *pro* class. If the value is negative, the cell-feature speaks *contra* class. The direction still remains the correlation.

I visualise the SHAP value to cell-feature correspondence in beeswarm plots. For binary classification I showed Figure 3.5 where I showed how the pooled cell-features influence the predictions, here classes p_0 or $p_{0.5}$. Inside this beeswarm plot, each point is one simulated biological sample. The colour of each point denotes the pooled cell-feature value. The position on the x-axis (the SHAP value) describes the impact on the prediction; I show the score before applying the sigmoid function of the output layer. A point with positive SHAP values increases that point's probability of being in class $p_{0.5}$. A point with negative SHAP values increases that point's probability of being in class p_0. The final prediction of a point is the sum over all SHAP values plus the expected value of the prediction across all biological samples as described in subsection 1.1.2.

In multiclass classification, one beeswarm plot as Figure 3.5 exists for each class Then positive SHAP values denote higher probability and lower values lower probability of the *current plot's* class. In the binary case shown in Figure 3.5, one plot for each class would be possible, but they would only be mirrored. Therefore, the two plots (one for p_0 and one for $p_{0.5}$) can be simplified into one.

Correlation

$$direction = \begin{cases} \text{pro class, positive} & \text{if } r > 0 \\ \text{pro class, negative} & \text{if } r <= 0 \end{cases} \tag{3.2}$$

SHAP split

$$direction = \begin{cases} \text{pro class, positive} & \text{if } a > b \\ \text{pro class, negative} & \text{if } a \leq b \end{cases} \tag{3.3}$$

Absolute SHAP

$$direction = \begin{cases} \text{pro class, positive} & \text{if } s > 0 \wedge r > 0 \\ \text{pro class, negative} & \text{if } s > 0 \wedge r <= 0 \\ \text{contra class, positive} & \text{if } s < 0 \wedge r > 0 \\ \text{contra class, negative} & \text{if } s < 0 \wedge r <= 0 \end{cases} \tag{3.4}$$

where

$$r := corr_{\text{pearson}}(featurevalues, shapvalues) \tag{3.5}$$

$$s := sign\,(\arg\max(|shapvalues|)) \tag{3.6}$$

$$a = mean(featurevalues|shapvalues > 0) \tag{3.7}$$

$$b = mean(featurevalues|shapvalues \leq 0) \tag{3.8}$$

3.2.3. Replace *FeatureLearner* with an decision tree

After identifying the relevant set R of cell-features, the *FeatureLearner* of those cell-features is replaced with decision trees, see item 3.2. The *FeatureLearner* transformed the input $X^{(i)} \in \mathbb{R}^{n \times p}$ into the matrix $A^{(i)} \in \mathbb{R}^{n \times k}$ for each biological sample $i \in \{1, \ldots, N\}$. I aggregate all cells from all biological samples and their respective *FeatureLearner* results:

$$\underline{X} \in \mathbb{R}^{(N \cdot n) \times p} \tag{3.9}$$

$$\underline{A} \in \mathbb{R}^{(N \cdot n) \times k} \tag{3.10}$$

For each cell-feature $j \in R$ ($\underline{A}_{\cdot j}$) based on $N \cdot n$ cells a decision tree can be built. This is a supervised regression model where a value for each cell (the cell-feature) based on all cell-parameters get predicted.

Especially for multiclass classification, I added the possibility to build a decision tree per class In that case, one decision tree is built for every cell-feature and class. For each decision tree, only cells from biological samples of its corresponding class are used to train the decision tree.

I show two visualisations of the resulting decision tree in Figure 3.7: Subfigure A shows a usual decision tree visualisation. The nodes are coloured according to the mean cell-feature value ($\underline{A}$) of all cells inside that node. Subfigure B shows the same decision tree, but shows if cells with certain cell-parameters have higher or lower cell-feature values.

3.2.4. Find the gating by traversing the decision trees

Gatings in flow cytometry are usually denoted as chains of cell-parameter$_i$ positive or negative cell subpopulations. Here I introduce a similar scheme based on the found decision trees. The decision tree gatings always start from the root of a decision tree and descend until a leaf is reached. How to descend the decision tree is defined by the gating direction explained in subsection 3.2.2.

For each relevant cell-feature, there is a decision tree and the information if high or low values of that cell-feature speaks for or against a specific class. To find the final gating, there are two options:

Traverse top-down: Start at the root node of the decision tree and descend to nodes with higher(/lower) cell-feature values according to the "direction" from the previously identified gating directions.

Traverse bottom-up: Start at the leaf node with the highest(/lowest) cell-feature value according to the "direction" from the previously identified gating direction

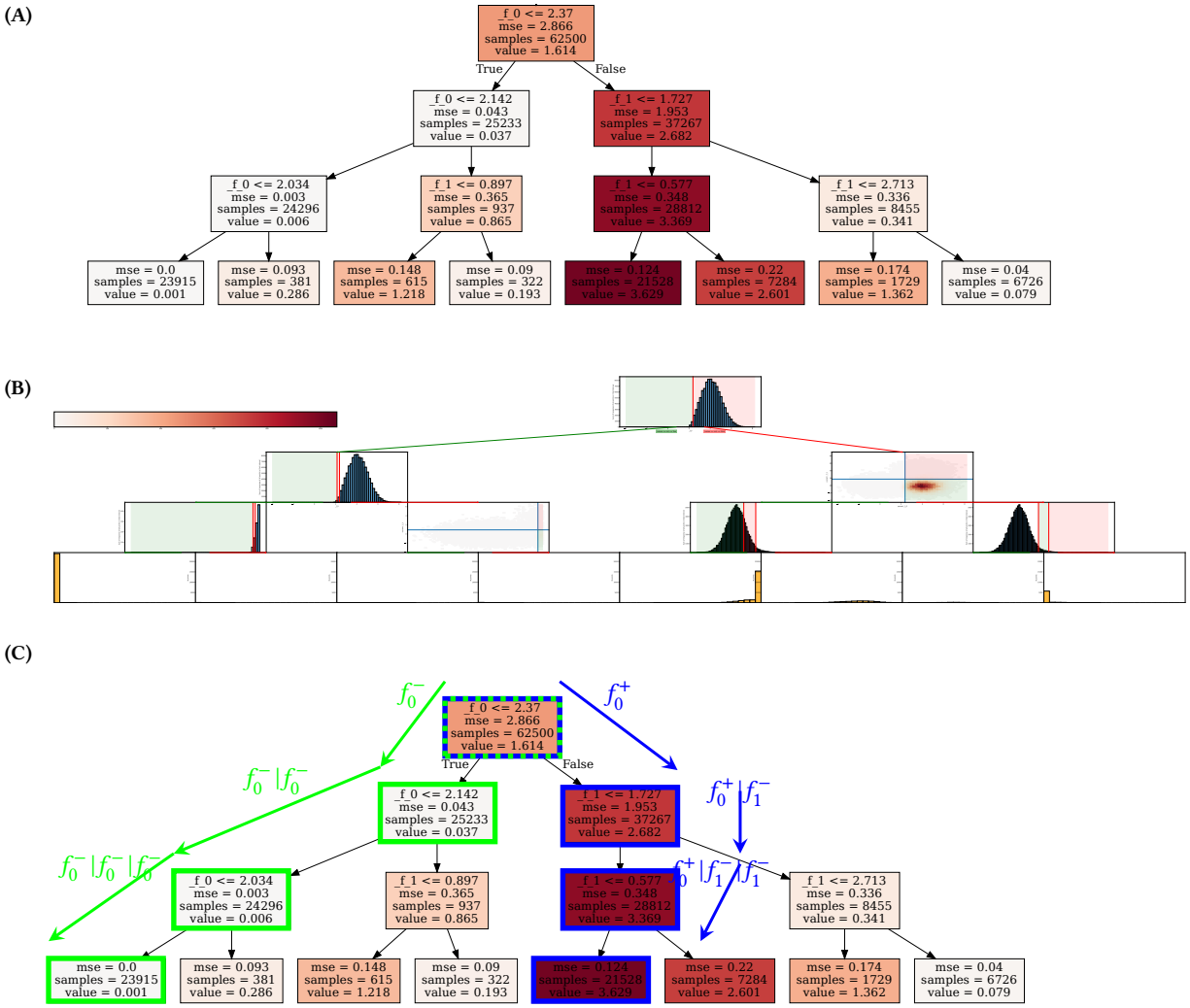


Figure 3.7.: Cell-feature decision trees for binary classification problem.
(A) shows the decision tree for one cell-feature, built as explained in section 3.2. The colour of each node depends on the mean cell-feature value of all cells inside that node.

(B) shows the same decision tree as in (A), but the actual cell-feature values (matrix $\underline{A}$) are shown in 1D or 2D histograms. The root node always shows a histogram of the first decision cell-parameter. The vertical line denotes the cell-parameter's cut-value. Values above the cut (where the decision was false) are in the red-shaded area, values below the cut are in the green-shaded area. All cells in the red shaded area are given to the right node in the next depth. All cells in the green shaded area are given to the left node in the next depth.
Subsequent depths always plot cells in the red and green areas of the parent plot. If the node's cell-parameter is identical to the parent's cut cell-parameter, again a histogram is plotted where the vertical line denotes the cell-parameter's cut value, and red/green areas denote cells that are given to the right/left subsequent node. If the node's cell-parameter is different to the parent's cut cell-parameter, a 2d-histogram is plotted. This 2d-histogram is realised as hexbin plot, which places a hexagonal grid over the cell-parameter space consisting of the parent and current tree cell-parameter. Each grid-element is coloured according to the mean cell-feature (matrix $\underline{A}$) of all cells inside that element. See Figure 3.8 for a conceptual explanation. Cell-feature (k_1) mainly focuses on the bottom right of the three cell subpopulations, see the right plot in the second depth.
The final leaf nodes are again histograms but only show the plain number of cells inside each leaf with the respective cell-parameter-value on the x-axis.

(C) shows the top-down gating paths towards lower values (green) and towards higher values (blue). Here resulting gatings are $f_0^-[2.37]|f_0^-[2.14]|f_0^-[2.03]$ (green) and $f_0^-[2.37]|f_1^-[1.73]|f_1^-[0.58]$ (blue).

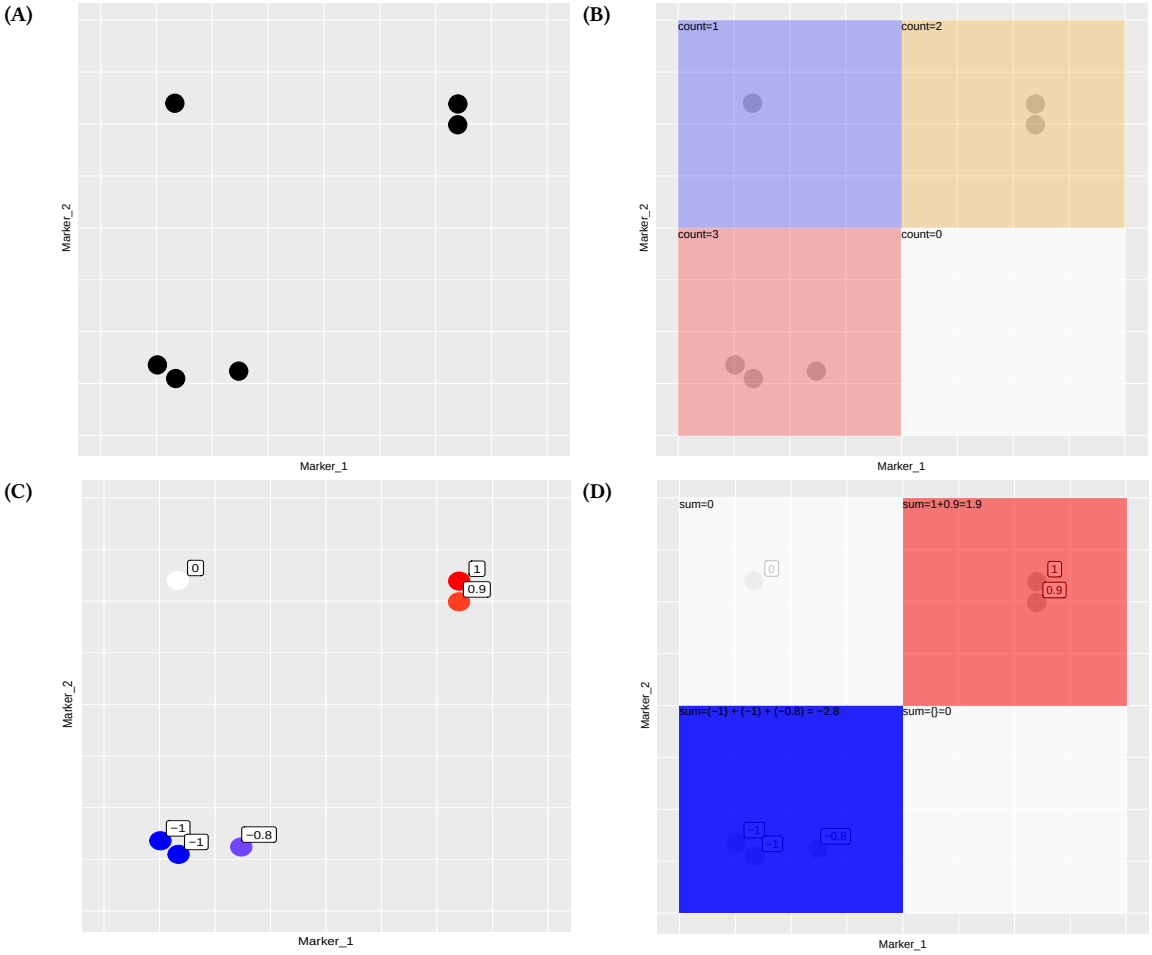


Figure 3.8.: 2d-histogram as classical (B) and cell-feature based (D) visualisation. Each plot shows cell-parameter 1 on the x-axis, cell-parameter 2 on the y-axis and includes the same cells as dots. (A) shows the cell-parameter position of the points. (B) shows the classical way of plotting cells: Place a grid over your space, here a rectangular grid with four grid-elements split at the half of each cell-parameter-axis. The number of cells inside each rectangle defines the colour of that rectangle. (C) some hypothetical function assigns a value to each cell - this is the task of the *FeatureLearner* in the FPP concept, resulting in the cell-feature values. (D) uses the same grid as (B), but instead of colouring the number of points, the colour of each rectangle is according to the total sum of cell-feature values inside.

and ascend to the root node.

After the path was found top-down or bottom-up, the decision tree gatings always start from the root and go downwards. In Figure 3.7 (C) I show two exemplary paths towards higher (blue) or lower (green) values. The resulting gatings are:

green $f_0^-|f_0^-|f_0^-$

blue $f_0^-|f_1^-|f_1^-$

The gatings can also be described additionally with their actual cut-values [in rectangles] after the cell-feature$^{+ \text{ or } -}$ description.

green $f_0^-[2.37]|f_0^-[2.14]|f_0^-[2.03]$

blue $f_0^-[2.37]|f_1^-[1.73]|f_1^-[0.58]$

3.3. Apply final gatings

In the previous section I showed how to find decision tree based gatings denoted as chains of cell-parameter$_i$ positive or negative cell subpopulations.

These gatings can be used directly without previous application of the FPP concept based neural network. Each gating consists of a cell subpopulation and the cell-parameter cutoffs to properly define it. With these cutoffs, each cell's cell subpopulation can be found. Subsequently, count the cells of each biological sample in each cell subpopulation. In CCC I called this "inference", also shown in Figure 2.4.

In Figure 3.9 I counted the cells in the final gating node for the decision tree of cell-feature k_1. The decision tree was built using only class $p_{0.5}$ samples shown in item 3.2. The gating direction was higher cell-feature values in class $p_{0.5}$. Since this example is a binary classification problem, lower cell-feature values speak for p_0. With this, two top-down gatings for k_1 can be identified: $f_0^-|f_0^-|f_0^-$ and $f_0^+|f_1^-|f_0^-$. The numbers in brackets are the actual cut value from the decision tree.

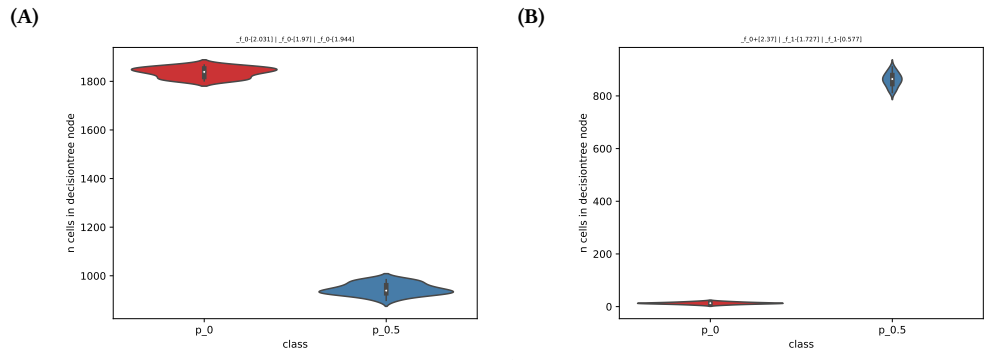


Figure 3.9.: Binary node gatings corresponding to item 3.2. (A) shows the identified gating when going towards lower activation values, so "pro p_0". (B) shows the identified gating when going towards higher activation values, so "pro $p_{0.5}$"

4. Discussion

Cytometry characterises cells of a biological sample by cell-properties like size, complexity or surface markers. A measured cell-property is called a cell-parameter, which defines each cell by numerical values suitable for further analysis. Up to 50 measured cell-parameters characterise each cell[5], and the resulting data is a matrix of unordered cells in the rows, and some cell-parameters as columns. A biological sample contains millions of cells, and their characterisation can be used to define cohesive cell subpopulations which reflect certain anatomic, morphologic or functional types of cells[42].

In this thesis, I use cytometry data for predictive modelling, where a class is assigned according to all cells of a biological sample together. This class enables clinical patient decisions when the class is known to respond to a certain treatment. Due to the unordered nature of cytometry measurements, only the presence or quantity of cells define the class of a biological sample but not which cell was measured first.

In the previous part of this thesis, I introduced the FPP concept for predictive modelling with cytometry data. FPP is a modular structure for neural networks containing three parts. First, the *FeatureLearner* transforms the measured matrix of a biological sample into a new matrix with new, predictive cell-features. Second, the *Pooler* aggregates these cell-features into a composition vector of values per biological sample. Third, the *Predictor* uses this composition vector to predict the final class.

In this part I propose a decision tree replacement for the *FeatureLearner* to explain the class predictions for biological samples. Why would it be useful to replace the *FeatureLearner*? Traverse the FPP neural network backwards:

1. For excellent performance on new biological samples, the complete FPP neural network must perform well.
2. For the complete FPP neural network to perform well, also the *Predictor* must perform well.
3. For the *Predictor* to perform well, there must be at least one predictive pooled cell-features from the *Pooler*.
4. A predictive pooled cell-feature must have different values for biological samples of different classes.
5. For a pooled cell-feature having different values per class, there must be predictive cells which 1) have a different cell-feature value than other cells and 2) have different abundances between classes.
6. The cell-features are the result of the *FeatureLearner* which used only the cell-parameters.
7. If the *FeatureLearner* can be described comprehensibly based on the cell-parameters, the predictive cells are described.

In cytometry, predictive cell subpopulations which enable the classification of biological samples are interesting. Classical gating uses sequentially chosen single values, rectangles, circles or polygons as cutoffs for the cell-parameters to group cells into cell subpopulations. When an algorithm finds which cell subpopulation is predictive, we already know per definition which cell subpopulations were responsible. The identified cell subpopulations can give information about biological processes.

To explain the prediction of a FPP neural network, I propose a gating based on a decision tree replacement for the *FeatureLearner*. Classical gatings in flow cytometry are usually denoted as chains of cell-parameter$_A$ positive or negative cell subpopulations. I showed that cell subpopulations defined by these gatings could be replaced by leaf nodes of decision trees. By descending the cell-parameter-based decision tree the decision cuts should be chained: Which cell-feature was cut at which value and if higher or lower values were chosen. The *FeatureLearner* is a neural network using the cell-parameters to calculate new cell-features. This neural network can be approximated by one decision tree per new cell-feature. The *FeatureLearner* identified predictive cells, and this decision tree is then the basis to describe their cell subpopulations.

Not all cell-features are necessarily predictive, therefore I identify relevant cell-features using SHAP importance values. The number of cell-features is a user-defined hyperparameter and the *Predictor* does not need to use them all. I use Deep SHAP[212] to calculate the SHAP values for the pooled cell-features which are the input for the *Predictor*. Figure 3.4 suggests that high values of mean:k_1 (red) indicate the $p_{0.5}$-class. High values of mean:k_0 and mean:k_2 indicate the p_0-class. This is in concordance to the visual inspection of the pooled cell-feature values from Figure 3.4 described in subsection 3.2.1.

The decision tree for one single Either one decision tree for one single cell-feature is built on cells from all biological samples, or one decision tree is built per class of biological samples. The *FeatureLearner* is applied on the cells to get their cell-feature values and the corresponding cell-parameters. Then, training the decision tree(s) is a classical regression problem where the response (the cell-feature value) of each cell is predicted by its cell-parameters.

For training the decision tree, all cells of your biological samples can be used. Each biological sample in flow cytometry has a variable number of cells which are usually downsampled during preprocessing. However, all available cells (of the training samples) can be used to increase the data basis for decision tree training!

To find the predictive decision tree leaf node, the gating directions is necessary. The decision tree groups cells based on their cell-feature value. To identify the predictive decision tree leaf node, I introduced gating directions based on SHAP values: They tell if high or low cell-feature values correspond to higher or lower probabilities for each class.

There are two options to find the decision tree gating: Top-down and bottom-up. Exemplary, let high cell-feature$_A$ values correspond to higher probability of class$_1$. For top-down gating, start at the root node and descend towards nodes with higher mean cell-feature$_A$ values across cells in the node until reaching a leaf node. For bottom-up gating, start at the leaf node with the highest mean cell-feature$_A$ value across cells in the node and ascend the decision tree until reaching the root node.

A previous publication on neural networks for cytometry data[3] proposed an alternative explanatory way, also based on decision trees: After a neural network was successfully trained, they predict one biological sample multiple times. Now they up-sample each cell of that biological sample by replacing random other cells with that specific cell. The performance difference between original biological sample prediction and the up-sampled prediction quantifies the impact of that specific cell. After applying this for many cells, they build a decision tree to predict these performance differences per cell based on the cell-parameters. They identify predictive cell subpopulation by finding the decision tree node with the highest mean performance difference across cells in that node. The final gating is then descending the decision tree in a top-down manner.

Gatings by decision trees do rarely follow classical gating schemes. The here presented approach uses SHAP importance values followed by building a decision tree. However, the built decision trees and subsequent gatings do not follow classical gating schemes. The mentioned alternative way to build decision trees[3] finds the relevance for each cell and builds one single decision tree for the whole network. This approach seems to have similar problems: The resulting gatings do not directly follow established gating schemes. Here, more work is necessary to find a way of presenting the decisions of the final neural network.

During testing the decision tree based gating, I found top-down being more consistent than bottom-up, especially when building deep trees. The problem with bottom-up gating is that a leaf node might contain only a tiny amount of cells (e.g. 1), then the mean cell-feature value can be extreme. Then actually relevant cell subpopulations which have a lower mean just by chance are overlooked.

In summary, I introduced a way to explain the predictions of FPP concept based neural networks. For this, I explained SHAP values and how to derive them for neural networks. With these SHAP values, the prediction-relevant cell-features and their gating direction can be identified. I showed how to replace the *FeatureLearner* for the relevant cell-features with decision tree(s). The final gatings defining discriminative cell subpopulationss are then the result of traversing these decision trees according to the gating direction. Finally, I showed how to apply these gatings on new biological samples. This enables the prediction of new biological samples without applying the neural network by just counting the number of cells in the identified discriminative cell subpopulationss.

There are some promising options to extend explainability:
1) RchyOptimyx[23] might be a potential lead as they propose a method to find the simplest possible gating form of previously identified phenotypes.
2) Another interesting approach might be attention-networks[215] where the neural network is extended with attention blocks that assign an attention-weight to each cell. These weights could be used as a proxy for cell importance. Attention networks have already been used to predict samples of histopathology imaging datasets[216].
3) Pre-training the FPP neural network to predict given cell subpopulation for each cell might help to get cell-features which better reflect classical gatings.
4) If we already have some algorithm which can predict a cell's cell subpopulation like a random forest or even as a neural network, we can count the number of cells in each cell subpopulation per biological sample. When we use this information as

additional information per biological sample, the FPP neural network does not need to find these known cell subpopulations but can focus on additional information present in the data.

Part IV.

Application of CCC

1. Introduction

Cytometry characterises cells of a biological sample by cell-properties like size, complexity or surface markers. A measured cell-property is called a cell-parameter, which defines each cell by numerical values suitable for further analysis. Up to 50 measured cell-parameters characterise each cell[5], and the resulting data is a matrix of unordered cells in the rows, and some cell-parameters as columns. A biological sample contains millions of cells, and their characterisation can be used to define cohesive cell subpopulations which reflect certain anatomic, morphologic or functional types of cells[42].

Predictive modelling in cytometry assigns a class label to a biological sample according to all measured cells of that sample. This class enables clinical patient decisions when the class is known to respond to a certain treatment. Due to the unordered nature of cytometry measurements, only the presence or quantity of cells define the class of a biological sample but not which cell was measured first.

There are multiple approaches for predictive modelling in cytometry which can be grouped into three concepts. First, partition and count cells based on the cell-parameters followed by predictive modelling[7,9–11]. Second, apply unsupervised cell subpopulation identification followed by predictive modelling[7,13–17,19,22–24,26–28,30–32,39]. Third, use all cells without previous summarisation of the cells[2,3,6].

I introduced the FPP concept, a concept to organise neural networks for predictive modelling in cytometry and its manifestation in my python package CCC. Multiple cells of a biological sample are used together to predict the class the biological sample. FPP consists of three parts. First, the *FeatureLearner* transforms the measured matrix of a biological sample into a new matrix with new, predictive cell-features. Second, the *Pooler* aggregates these cell-features into a composition vector of values per biological sample. Third, the *Predictor* uses this composition vector to predict the final class.

I also introduced a way to explain the predictions of FPP concept based neural networks. The *FeatureLearner* can be replaced by multiple decision trees for the relevant cell-features. The relevant cell-features can be identified through SHAP importance values for the pooled cell-features. Final gatings defining discriminative cell subpopulationss are then the result of traversing these decision trees.

In this part, I apply my python package CCC to multiple datasets and report and interpret the results.

The first dataset is a binary classification simulation where cells of biological sample were simulated with two cell-parameter. Three cell subpopulation were simulated and one cell subpopulation was either present or absent.

The second dataset is a multiclass classification simulation where cells of biological sample were simulated with two cell-parameter. Three cell subpopulation were simulated and the classes of biological samples were defined by the percentage of

cells in one explicit cell subpopulation.

The third dataset is a real binary classification problem where 63 paired blood biological samples of two classes were measured with flow cytometry. In the ANTE class, one patient was measured after starving for 24 hours. After glucose-intake, the same patient was measured after one hour (POST class). The biological samples were measured in two separate batches, and each cell had 19 cell-parameters.

2. Methods

2.1. Simulations

I introduce a simulated binary and multiclass classification problem. The resulting datasets were used to showcase FPP concept based neural networks.

2.2. Binary classification simulation

In this simulation, each simulated biological sample contains 10^5 cells with two cell-parameters each. The cells are drawn from the following three multivariate normal distributions with base odds $(1, 1, 3)$ for $(\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3)$.

$$\begin{gathered} \mathcal{N}_1(\begin{pmatrix}4\\0\end{pmatrix}, \sigma^2) \quad \mathcal{N}_2(\begin{pmatrix}4\\4\end{pmatrix}, \sigma^2) \quad \mathcal{N}_3(\begin{pmatrix}0\\0\end{pmatrix}, \sigma^2) \\ \sigma^2 := \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix} \end{gathered} \tag{2.1}$$

The simulation consists of varying the odds of a cell being in $\mathcal{N}_1$ such that the percentage of cells is either 0% (p_0) or 50% ($p_{0.5}$). For each percentage-class I create 100 biological samples of 10^5 cells.

2.3. Multiclass classification simulation

As in the simulation for binary classification, each generated biological sample contains 10^5 cells with two cell-parameters each. The cells are drawn from the same three multivariate normal distributions (see Equation 2.1) but now the percentage of the 10^5 cells inside drawn from $\mathcal{N}_1$ varies over the following values:

$$\{0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.2, 0.5\} \tag{2.2}$$

2.3.1. FPP network settings for binary and multiclass classification

The FPP network architecture consists of three parts: The *FeatureLearner*, *Pooler* and the *Predictor*. In both simulations, we fixed the hyperparameters such that 50000 cells were subsampled without replacement before further training. Then 2500 cells were the input for the *FeatureLearner*, drawn randomly in each iteration (unfixed subsampling). The *FeatureLearner* consisted of two layers with three nodes each, so the two input cell-parameters f_0 and f_1 were densely connected to three

nodes and those again densely connected to three nodes. Each layer was followed by batch normalisation[158] and after that by a ReLU-function[155]. The *Pooler* aggregated over all 2500 cells with the mean *and* the maximum resulting in $k * q = 3 \times 2 = 6$ pooled cell-features. The *Predictor* used those in one layer with three nodes, also followed by batch normalisation[158] and ReLU-function[155].

I used a batch size of 24, the best validation loss over 2500 epochs defined the final selected model. I used Adam[217] for optimisation with a learning rate of 0.0001, no weight decay, 0.9 as running average gradient coefficient, a coefficient of 0.99 for the squared running average gradient and 10^{-8} as an additive term to the denominator for numerical stability.

Simulation A In the binary problem of simulation A, the *Predictor* had a single output node and a sigmoid function ensured 0/1 output. I used the binary cross-entropy

$$-\left(y_{true}\log\left(\hat{y}_{prob}\right) + (1 - y_{true})\log\left(1 - \hat{y}_{prob}\right)\right) \tag{2.3}$$

with the true 0/1-encoded label y and the predicted probability $\hat{y}_{prob} \in [0, 1]$.

Simulation B In the multiclass problem of simulation B with eight classes ($C = 8$), the *Predictor* had eight output nodes, followed by a softmax-function. I used cross-entropy as described in pytorch[157] (shown in Equation 2.4) to calculate the loss. Pytorch combines the softmax and the negative log-likelihood for computational efficiency. The prediction for one sample y is then a one-hot encoding of C values and $\hat{\mathbf{y}} = \text{softmax}(\hat{\mathbf{y}}')$ where $\hat{\mathbf{y}}'$ are the predicted scores from the neural network before applying the softmax.

$$H(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{C}\sum_{c=1}^{C}\sum_{i=1}^{N} loss\left(\mathbf{y}_c, \text{softmax}(\hat{\mathbf{y}}'_c)\right) \tag{2.4}$$

2.4. Identifying fasting blood samples with CCC

2.4.1. ANTE vs POST samples

I received 63 paired blood biological samples of two classes. In the ANTE class, one patient was measured after starving for 24 hours. After glucose-intake, the same patient was measured after one hour (POST class).

The biological samples were measured in two separate batches.

The measured cell-parameters were: Forward scatter area and width (FSC-A and FSC-Width), side scatter area (SSC-A) and the signal area of multiple fluorochromes. The latter were chosen to differentiate T-cell cell subpopulations[218], see Table 2.1.

The data was preprocessed by the min-max transformation of the linear cell-parameters FSC-A, FSC-Width and SSC-A to a scale of 0-1 ($x_{scaled} = \frac{x - min(x)}{max(x) - min(x)}$). The fluorochrome cell-parameters were transformed using the arcsinh transformation with a cofactor 150 (see subsubsection 1.4.1.2). After transformation, each biological sample was subsampled to a fixed amount of 50000 cells, drawn randomly without replacement.

Marker	Fluorochrome
CCR7	B525-KB520
CD27	B610-ECD
CD8	B690-PerCP
KLRF1	Y585-PE
CD57	Y610-AF594
CD279	Y710-PC5.5
CD161	Y763-PC7
GPR56	R660-APC
CD28	R712-AA700
HLA-DR	R763-AA750
KLRG1	V450-BV421
CD4	V525-BV510
CD45RA	V763-BV785
CD45	UV405-BUV395
CD3	UV675-BUV661
Live-Dead	IR885-ViaKr808

Table 2.1.: ANTE vs POST starving experiment, T-cell cell-parameters their respective fluorochromes.

The network used 2500 randomly sampled cells from these fixed 50000. This results in a combination of fixed and unfixed subsampling as explained in subsection 2.2.1.

2.4.2. FPP network settings

I fixed the hyperparameters such that 2500 cells were put into the *FeatureLearner*. The *FeatureLearner* consisted of two layers with three nodes each. Therefore, 19 cell-parameters were densely connected to three nodes and those again densely connected to three nodes. Each layer was followed by batch normalisation[158] and a ReLU-function[155]. The *Pooler* aggregated the 2500 cells with the mean *and* the maximum resulting in $k * q = 3 \times 2 = 6$ pooled cell-features. The *Predictor* used those in one layer with three nodes, also followed by batch normalisation[158] and ReLU function[155]. Those were then combined into a single node, and a sigmoid function ensured 0/1 output.

For optimisation of the network I used the binary cross-entropy as loss

$$-\left(y_{true}\log\left(\hat{y}_{prob}\right) + (1 - y_{true})\log\left(1 - \hat{y}_{prob}\right)\right) \tag{2.5}$$

with the true 0/1-encoded label y and the predicted probability $\hat{y}_{prob} \in [0, 1]$. I used a batch size of 24, the final model was chosen based on the best validation loss over 2500 epochs. Training took about 5 minutes on a personal laptop with the GPU (NVIDIA GeForce GTX 1650) enabled, an Intel®Core™i5-9300H (2.4GHz) and 32GB RAM. I used Adam[217] for optimisation with a learning rate of 0.0001, no weight decay, 0.9 as running average gradient coefficient, 0.99 as coefficient for the squared running average gradient and 10^{-8} as an additive term to the denominator for numerical stability.

The model was trained 50 times with these hyperparameter settings.

3. Results

3.1. Binary classification simulation

In the binary simulation, I simulated two classes of biological samples. Cells were simulated with two cell-parameters. Class $p_{0.5}$ was defined by drawing 50% of all cells from one of three normal distributions, class p_0 drew no cells from the same distribution. All cells from one exemplary biological sample from each class is shown in Figure 3.1.

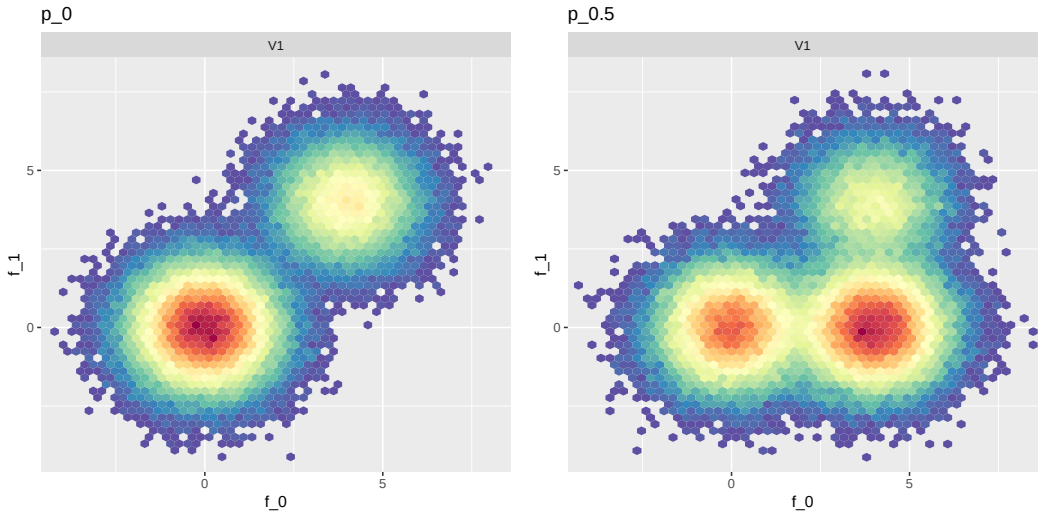


Figure 3.1.: Binary classification simulation data, one exemplary simulated biological sample from class p_0 (left) and one exemplary biological sample from class $p_{0.5}$ (right). All 10^5 cells are shown.

I defined a FPP neural network with two *FeatureLearner* layers, mean and maximum pooling layer and one *Predictor* layer where 2500 cells were the input of one biological sample. I split the data randomly into 40 training, 40 validation and 40 test samples. See section 2.2 and subsection 2.3.1 for a detailed explanation on data generation and training settings.

Training, validation and test performance measured through ROC curves and the AUC were perfect, as shown in Figure 3.2.

In Figure 3.3 I show the results of the SHAP value calculation: Max-pooling of k_1 had essentially no impact on the classification as the SHAP-values were zero for almost all simulated biological samples. Max-poolings of k_0 and k_2 had small, variable strength effects. Mean-poolings had clear, separated effects: High values of the cell-feature means spoke either for (k_1) or against (k_0 and k_2) $p_{0.5}$. The mean of k_1 had the strongest effect according to the SHAP values. It was so strong that the model output from mean:k_1 cannot be changed by the other pooled cell-feature values.

In Figure 3.4 I show the decision tree replacement for the cell-feature k_1 with the highest SHAP values across simulated biological samples. Figure 3.4 (A) shows the decision tree replacement for k_1 and (B) the respective histograms of the cell-feature value over the cells, as explained in Figure 3.7 of Part III.

I found the final gatings, shown in Table 3.1, according to the identified gating directions for mean:k$_1$. Pro $p_{0.5}$ higher resulted in the final gating $f_0^+ \mid f_1^- \mid f_1^-$ for class $p_{0.5}$ by traversing the decision tree towards higher node values. Pro p_0 lower resulted in the final gating $f_0^- \mid f_0^- \mid f_0^-$ for class p_0 by traversing the same decision tree towards lower node values.

Finally, I applied those gatings on the test data and visualised the resulting counts per biological sample in the two respective final gates in Figure 3.5. In subfigure (A) I show that simulated biological samples of class p_0 (red) had more cells in gate $f_0^- \mid f_0^- \mid f_0^-$. In subfigure (B) I show that simulated biological samples of class $p_{0.5}$ had more cells in gate $f_0^+ \mid f_1^- \mid f_1^-$: Almost zero cells were in this gate when the simulated biological samples was actually from class p_0.

The results shown here were directly taken from test_sim05_binary_dtreeTwoclasses, from the CCC implementation, accessible via GitLab from the group of Prof. Dr Spang.

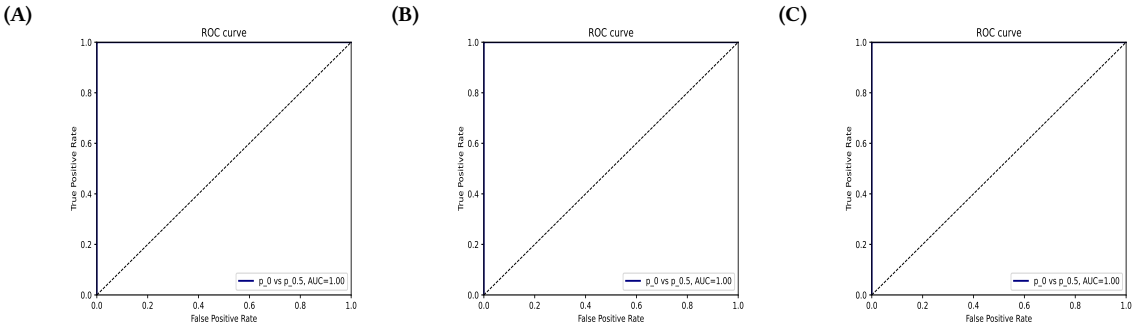


Figure 3.2.: Binary simulation: ROC curves of the final model's predictions on training (A), validation (B) and test (C) simulated biological samples.

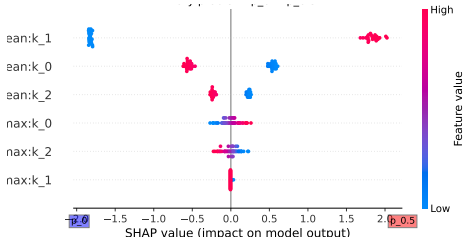


Figure 3.3.: Binary simulation: SHAP scores. Max-pooling of k_1 had essentially no impact on the classification as the SHAP-values were zero for almost all simulated biological samples. Max-poolings of k_0 and k_2 had small, variable strength effects. Mean-poolings had clear, separated effects: High values of the cell-feature means spoke either for (k_1) or against (k_0 and k_2) $p_{0.5}$. The mean of k_1 had the strongest effect according to the SHAP values. It was so strong that the model output from mean:k_1 cannot be changed by any other pooled cell-feature values.

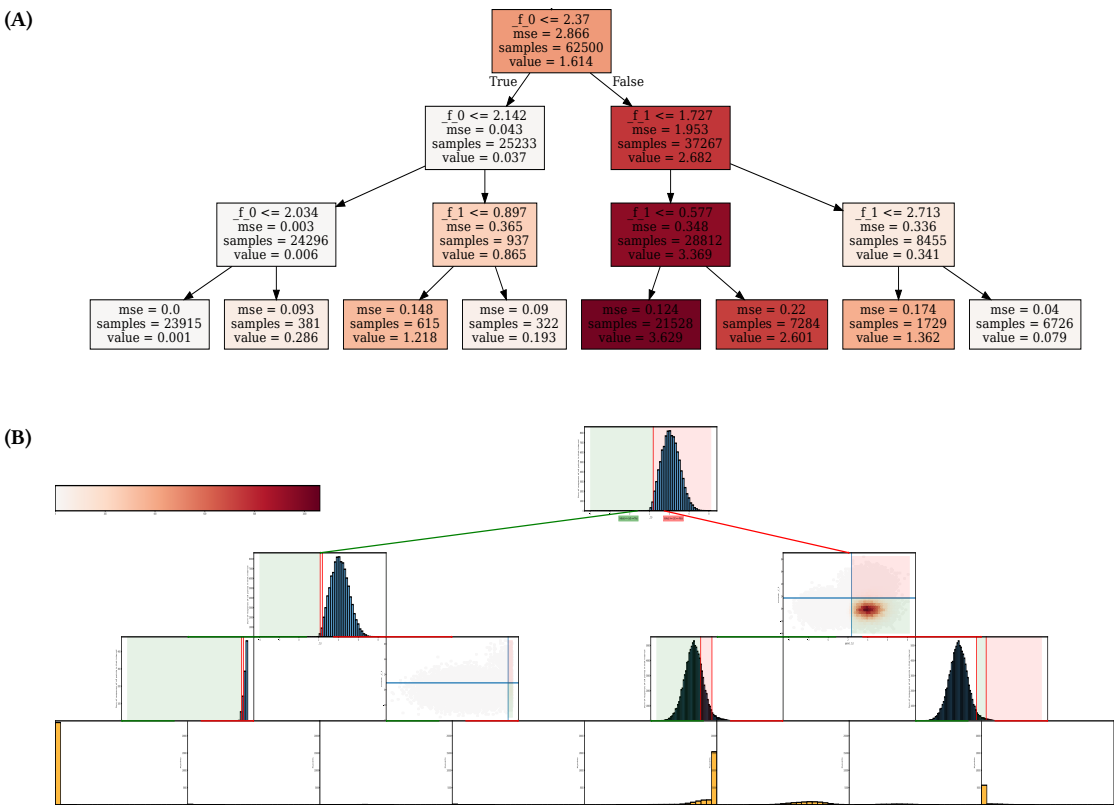


Figure 3.4.: Binary simulation: Cell-feature trees of the final model for the test samples. (A) shows the decision tree, (B) the respective histograms of the cell-feature value over the cells with the cell-parameters as axes. See Figure 3.8 of Part III and the respective section for further explanation as I show the same plots there.

Feature	Pooling	Class	pro/contra	Direction	cut_1	cut_2	cut_3
k_1	mean	$p_{0.5}$	pro	higher	f_0^+	f_1^-	f_1^-
k_1	mean	p_0	pro	lower	f_0^-	f_0^-	f_0^-

Table 3.1.: Binary simulation: Gating based on gating directions derived from Figure 3.3 and the decision tree shown in Figure 3.4

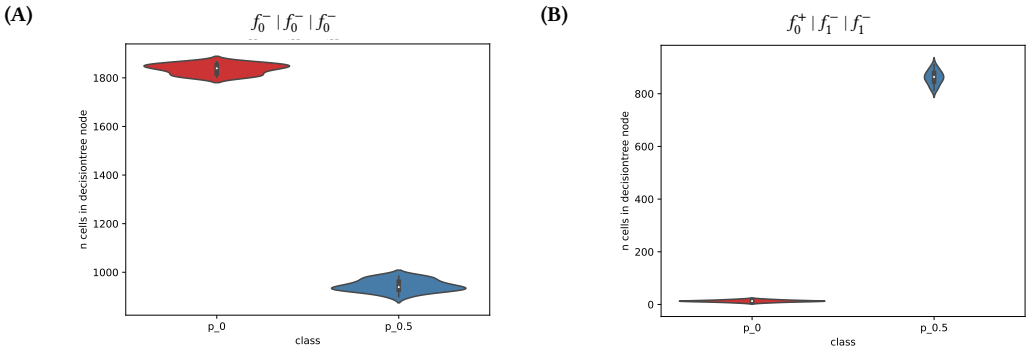


Figure 3.5.: Binary simulation: Applied gatings to the simulated test biological samples. Counts for each simulated biological sample are shown in boxplots for all test samples. The gatings were based on Figure 3.4 and Table 3.1. (A) shows the identified gating when going towards lower cell-feature values, so "pro class p_0". If a simulated biological sample was actually in class p_0 (red) there were more cells in that gate. (B) shows the identified gating when going towards higher cell-feature values, so "pro class $p_{0.5}$". Simulated biological samples of class $p_{0.5}$ had more cells in this gate: Almost zero cells were in this gate when the sample was actually from class p_0.

3.2. Multiclass classification simulation

3.2.1. Theoretical performance boundaries for multiclass classification simulation

In subsection 2.3.1 I described different subsampling methods. In this simulation, I subsample 2500 cells from each sample.

Subsampling 2500 cells makes it impossible to distinguish all simulated classes. All cells of one simulated biological sample per class are shown in Figure 3.6. The subsampled cells of the same simulated biological samples per class are shown in Figure 3.7. In the simulation, for samples with a $\mathcal{N}_1$ percentage of 10^5 there is an expected value of one single cell drawn from the $\mathcal{N}_1$-distribution which will be lost when subsampling 2500 cells. When subsampling 2500 cells, simulated biological sample classes with less than $\frac{1}{2500} = 4 \times 10^{-4}$ cells cannot be distinguished. Therefore, I introduce the *zero-like* class where simulated biological sample classes with less than 4×10^{-4} are inside.

In Figure 3.8 I show the number of cells originating from $\mathcal{N}_1$ of all simulated biological samples: (A) For all cells and (B) for a random subset of 2500 cells. We see that already for class $p_{0.001}$, the *median* number of cells from that $\mathcal{N}_1$ in a simulated biological sample is two. For classes below percentage $p_{0.001}$ the median number is one or even zero. The maximum number of cells in $\mathcal{N}_1$ was 7.

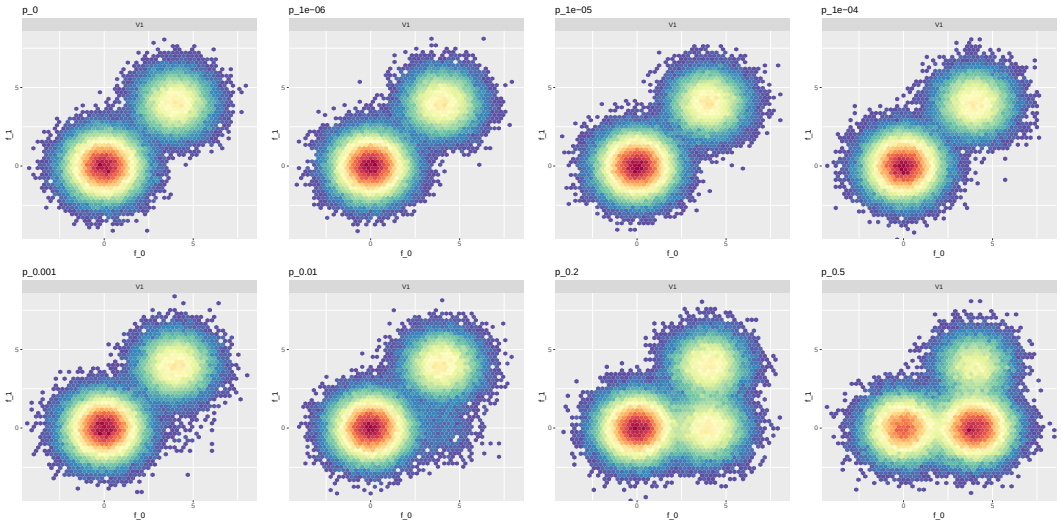


Figure 3.6.: Multiclass classification simulation data, one exemplary biological sample from each class where all 10^5 cells are shown. Going from top-left to bottom-right to higher percentages of $\mathcal{N}_1$.

3.2.2. Multiclass simulation results

In the multiclass simulation, I created eight classes of samples, where each class was defined by the percentage of all cells drawn from one of three multivariate normal distributions ($\mathcal{N}_1$).

As in the binary classification simulation, I used two *FeatureLearner* layers, mean and maximum poolings and one *Predictor* layer where 2500 cells were the input of one sample. In contrast to the binary simulation where I only used one output

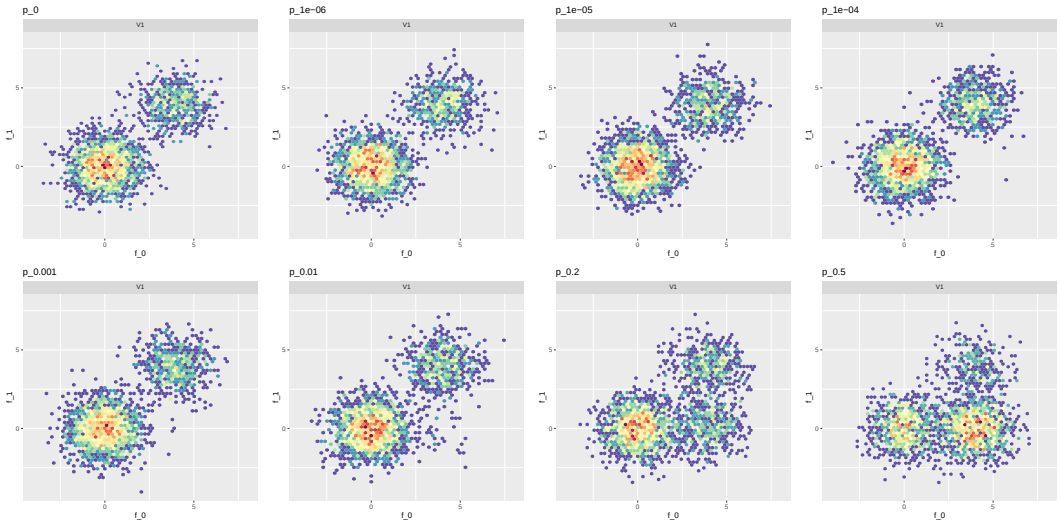


Figure 3.7.: Multiclass simulation data, one exemplary simulated biological sample from each class where only 2500 random of all generated 10^5 cells are shown. Going from top-left to bottom-right to higher percentages of $\mathcal{N}_1$. Samples with a $\mathcal{N}_1$ percentage lower than 0.01 are difficult to distinguish.

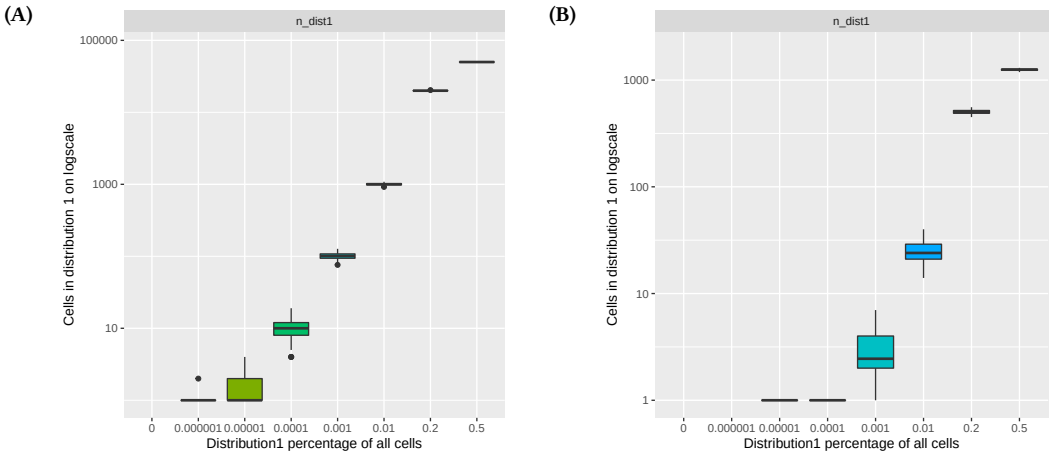


Figure 3.8.: Multiclass simulation data, boxplots of the true amount of cells drawn from $\mathcal{N}_1$ for all 10^5 cells (subfigure A) and 2500 randomly drawn cells (subfigure B). At $\mathcal{N}_1$ percentage of 0.001, there was a maximum of 7 cells in one biological sample, shown in subfigure (B). For class p_0 in subfigure (A) (and in (B) also for class 10^{-6}), no boxplots are shown as no cells were coming from $\mathcal{N}_1$.

node, here I used eight output nodes: One for each class. From the 100 simulated biological samples per class (800 biological samples), I randomly split the data into 480 training, 160 validation and 160 test samples. See section 2.3 and subsection 2.3.1 for a detailed explanation on data generation and training settings.

Training, validation and test performance were measured through one-vs-all ROC curves and the AUC, as shown in Figure 3.9. The test AUCs were perfect for classes $p_{0.5}$ and $p_{0.2}$. The test AUCs had good performance for $p_{0.01}$. Then the AUC drops - but not lower than 0.76 for each class.

Figure 3.10 shows the predicted probabilities of each class for all test simulated biological samples. Samples of classes $p_{0.5}$ and $p_{0.2}$ had almost probability 1 for the respective true class. For Samples from class $p_{0.01}$ there was one misclassification in 19 simulated test samples. Samples from class $p_{0.001}$ were misclassified in 15 of 23 cases. The other (*zero-like*) classes had almost identical probabilities for each class but were finally predicted as $p_{10^{-5}}$.

In contrast to binary classification, in multiclass classification the SHAP values must be visualised in one beeswarm plot per class. I show those plots in Figure 3.11 where cell-feature k_2 showed to be the major source of predictive power, either pooled with the mean or the maximum. In each plot, the current regarded class is written on top of the plot.

Mean and maximum of cell-feature k_2 were always the most important pooled value according to the SHAP values plotted in Figure 3.15. A high maximum of cell-feature k_2 spoke against each respective class. A high mean spoke slightly against each respective class. All other pooling:feature combinations had essentially no effect on the prediction. In subfigure (E), we see that the absolute effect of the mean increased and the impact of the maximum decreased. Starting with (F), the effects became less obvious, but for $p_{0.01}$ and $p_{0.2}$, high values of the maximum spoke for both classes. Simultaneously, a high mean resulted in class $p_{0.2}$, a low mean in $p_{0.01}$. The last class, $p_{0.5}$, can be achieved mainly by having a high mean - but if also the maximum is high, class probability gets low again.

I show in Figure 3.12 the decision tree *FeatureLearner* replacements only for k_2. The shown decision trees were built only on cells from biological samples from classes $p_{0.5}$ or p_0.

The final gatings were found as described in section 3.2 and are shown in Table 3.2. Finally, I apply those gatings on the test data and visualise the resulting counts per biological samples of one exemplary gate in Figure 3.13. All gates can be found in the appendix Figure F 1.

The results shown here are directly taken from the minimum working examples 01_MWE/26_ccc_MWEs/d02_simulations from the CCC implementation accessible via GitLab from the group of Prof. Dr Spang.

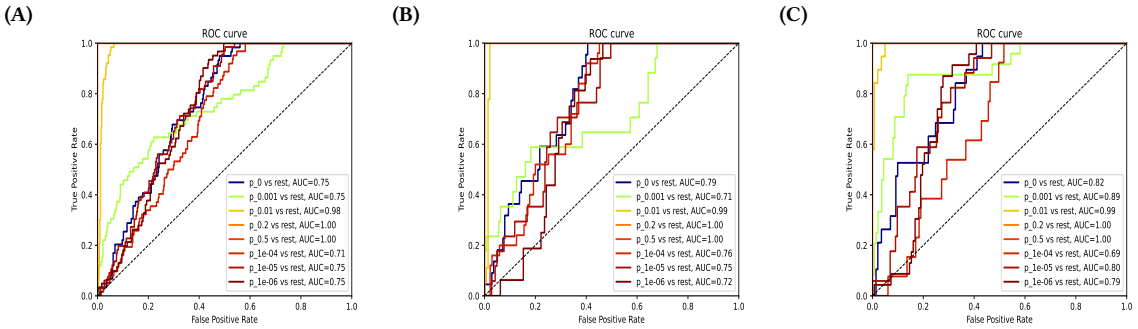


Figure 3.9.: Multiclass simulation: ROC curves of the final model's prediction on training (A), validation (B) and test (C) samples. Focussing on the test AUCs, the classes $p_{0.5}$, $p_{0.2}$ had a perfect AUC of 1. In classes $p_{0.01}$ the AUC already dropped to 0.99. Some of the simulated biological samples in class $p_{0.001}$ were able to be identified, resulting in an AUC of 0.89. Finally, all other classes, previously denoted as *zero-like*, were non-distinguishable. However, due to the nature of one-vs-all, they had an AUC bigger than 0.5 because they *were* actually distinguishable from $p_{0.5}$, $p_{0.2}$ and $p_{0.01}$, only not from each other.

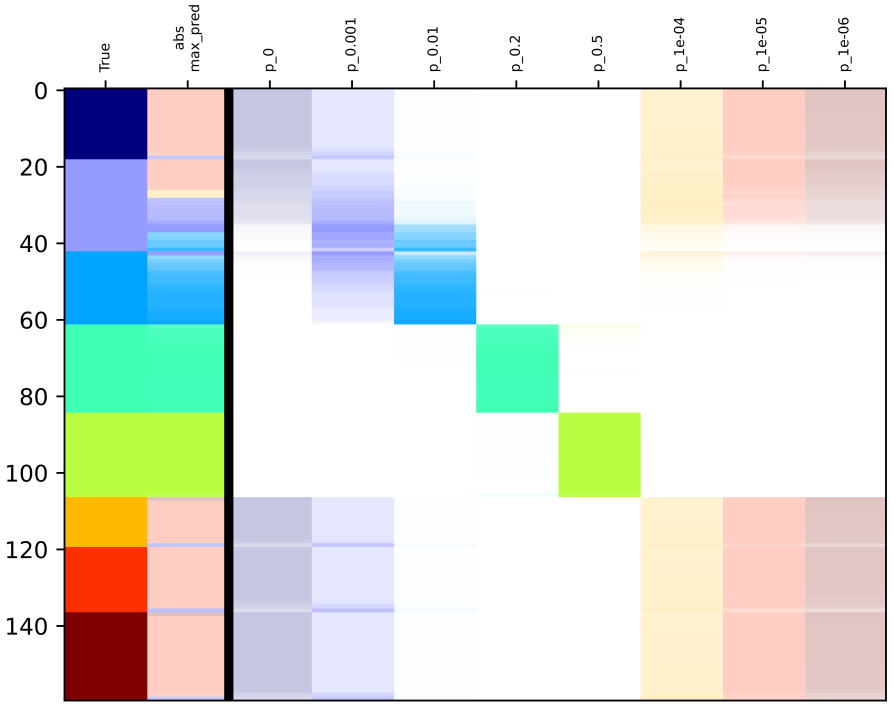


Figure 3.10.: Multiclass simulation: Predicted probabilities per class and simulated biological sample. Each row is one simulated sample, sorted by the true class as coloured in the first column. The second column is coloured according to the class with the highest predicted probability. The colour saturation denotes the probability from zero (white) to one (full colour). For each row, all further columns show the probabilities for the class shown on the x-axis on top. Ideally, the colours would be arranged in diagonal, fully saturated blocks for this simulation. We see that for class $p_{0.5}$ and $p_{0.2}$ this was the case. In class $p_{0.01}$ were already two samples where the probability for another class ($p_{0.001}$) was higher, so predictions were wrong. For class $p_{0.001}$ at least some of the probabilities were high enough to predict the correct class but about half were indistinguishable from the *zero-like* other classes $p_{10^{-4}}$, $p_{10^{-5}}$, $p_{10^{-6}}$ and p_0. For simulated biological sample from those classes we observed three things: First, their probability for the clear $p_{0.5}$ and $p_{0.2}$ classes (and also mostly for $p_{0.01}$) was zero. Second, the probability for each class was almost the same but low: They are indistinguishable from each other. Third, most times a *zero-like* was predicted non-*zero-like*, its predicted class is $p_{0.001}$.

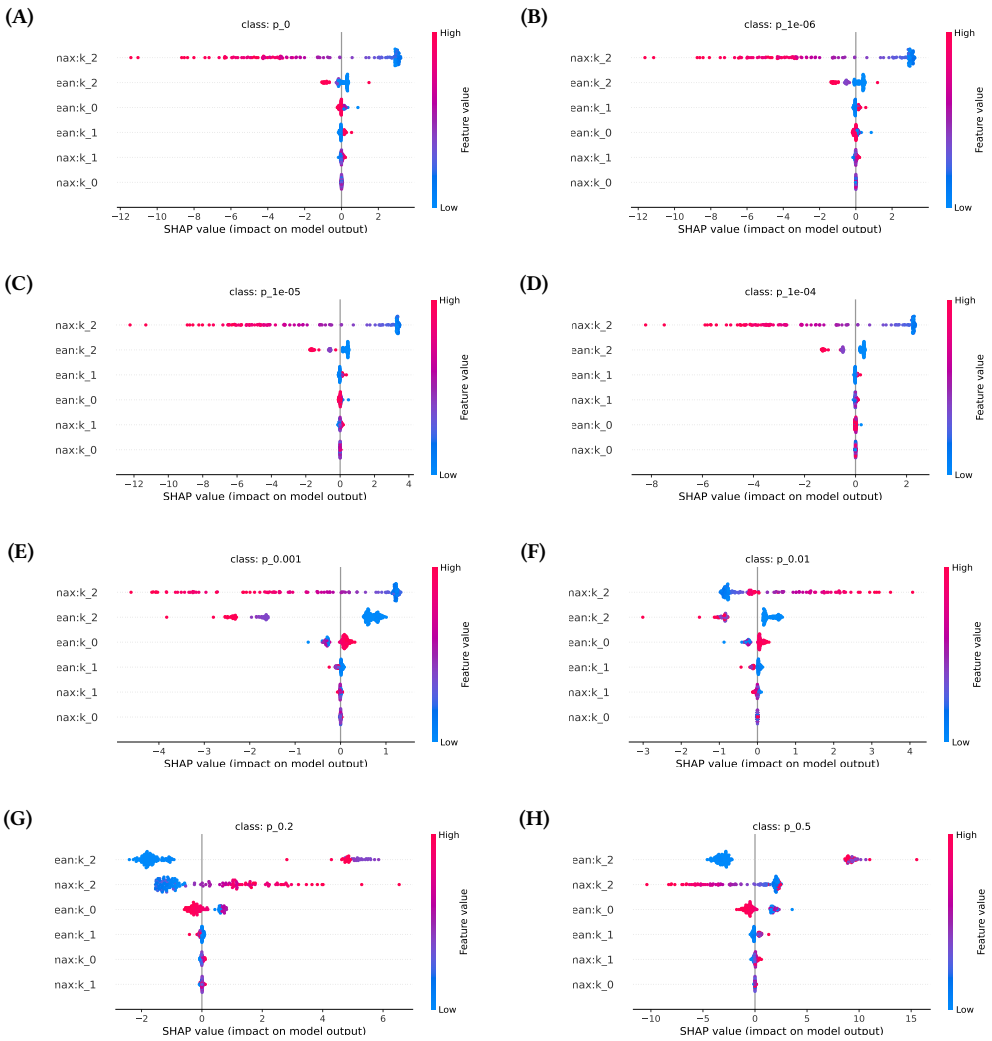


Figure 3.11.: Multiclass simulation: SHAP values. From top-left to bottom-right, the percentage-defined class increases. As the *zero-like* classes were indistinguishable anyway, also the SHAP values looked almost identical in their respective subplots (A)-(D). Only in (D) the x-axis changes. A high maximum of cell-feature k_2 spoke against each respective class. A high mean spoke slightly against each respective class. All other pooling:feature combinations had essentially no effect on the prediction. In subfigure (E), we see that the absolute effect of the mean increased and the impact of the maximum decreased. Starting with (F), the effects became less obvious, but for $p_{0.01}$ and $p_{0.2}$, high values of the maximum spoke for both classes. Simultaneously, a high mean resulted in class $p_{0.2}$, a low mean in $p_{0.01}$. The last class, $p_{0.5}$, can be achieved mainly by having a high mean - but if also the maximum is high, class probability gets low again.

Feature	Pooling	Class	pro/contra	Direction	cut_1	cut_2	cut_3
k_2	mean	p_0	pro	lower	f_0^-	f_1^-	f_1^-
k_2	mean	p_{1e-06}	pro	lower	f_0^-	f_1^-	f_1^-
k_2	mean	p_{1e-05}	pro	lower	f_0^-	f_1^-	f_1^-
k_2	mean	p_{1e-04}	pro	lower	f_0^-	f_1^-	f_1^-
k_2	mean	$p_{0.001}$	pro	lower	f_0^-	f_1^-	f_1^-
k_2	mean	$p_{0.01}$	pro	lower	f_0^-	f_1^-	f_1^-
k_2	mean	$p_{0.2}$	pro	higher	f_0^+	f_1^-	f_0^+
k_2	mean	$p_{0.5}$	pro	higher	f_0^+	f_1^-	f_0^+

Table 3.2.: Multiclass simulation: Gatings based on gating directions from Figure 3.11 and the decision tree shown in Figure 3.12. Even if trained only on cells from simulated biological samples from the respective class, there are only two different final gatings: $f_0^- \mid f_1^- \mid f_1^-$ for *zero-like*, $p_{0.001}$ and $p_{0.01}$ classes and $f_0^+ \mid f_1^- \mid f_0^+$ for $p_{0.2}$ and $p_{0.5}$

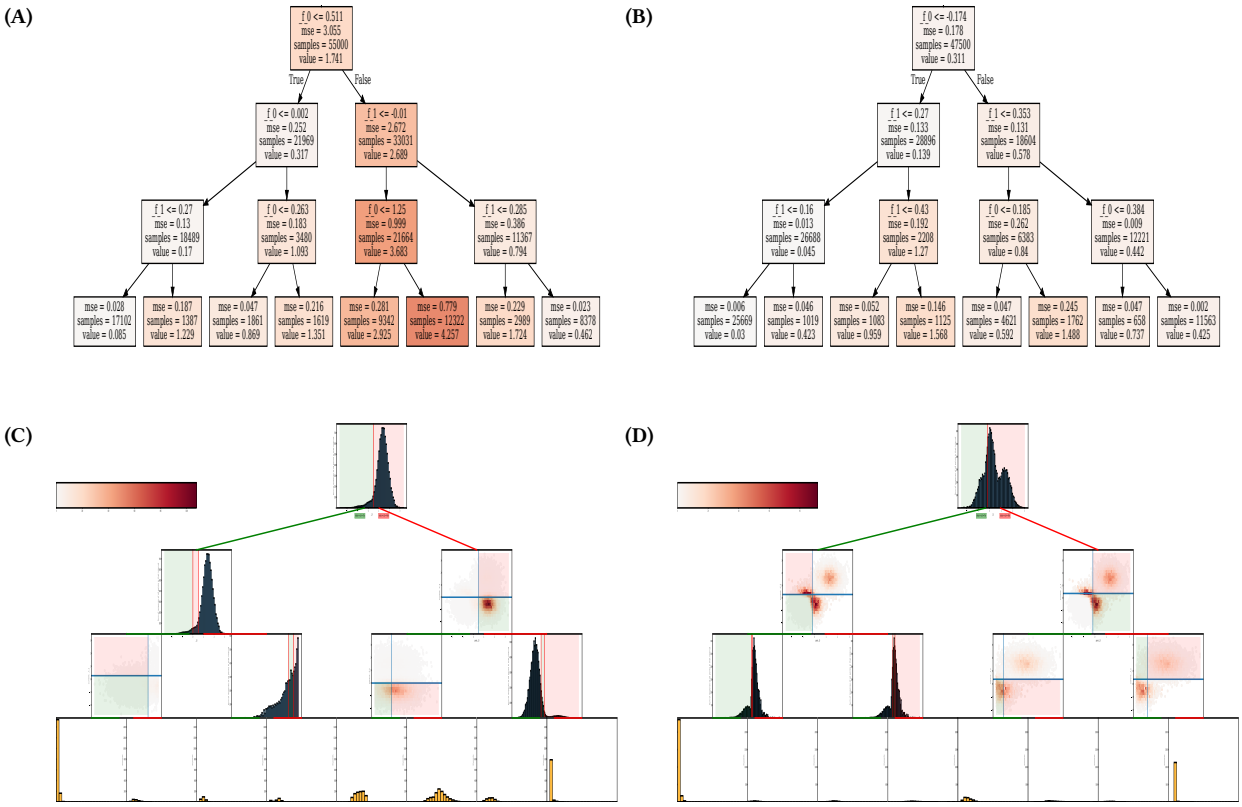


Figure 3.12.: Multiclass simulation: Cell-feature trees of the final model for the simulated test biological samples. (A) shows the decision tree for class $p_{0.5}$, (C) the respective histograms of the cell-feature value over the cells with the cell-parameters as axes. (B) shows the decision tree for class p_0, (D) the respective histograms of the cell-feature value over the cells with the cell-parameters as axes. (C) shows that high cell-feature values focussed on the bottom right cell subpopulation drawn from $\mathcal{N}_1$ and in (D) that high cell-feature values focussed on cells drawn from $\mathcal{N}_2$ and $\mathcal{N}_3$.

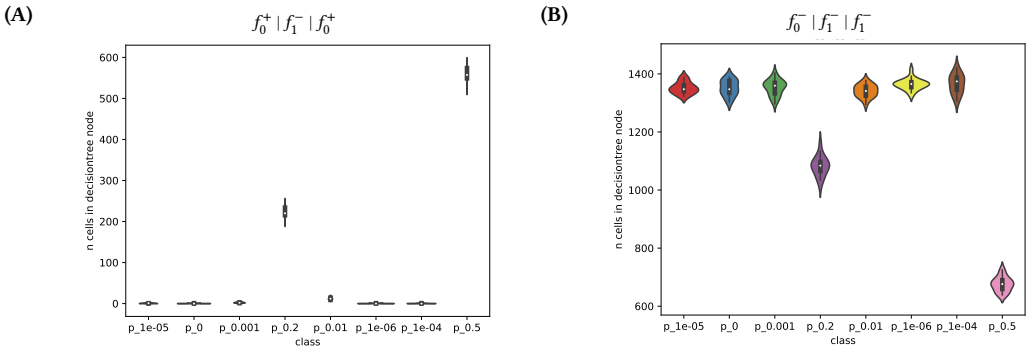


Figure 3.13.: Multiclass simulation: Applied gatings to the simulated test biological samples. Counts for each sample are shown in boxplots for all test samples and each class on the x-axis. The gatings were based on Figure 3.12 and Table 3.2. (A) shows the gating towards higher values, correlating with higher probability. $p_{0.5}$ had the highest counts, followed by $p_{0.2}$ and also $p_{0.01}$ had visibly elevated counts compared to the other classes. (B) shows the gating towards lower values, correlating with higher probability. Class $p_{0.5}$ had the lowest counts, followed by $p_{0.2}$. $p_{0.1}$ is slightly decreased where the other classes were indistinguishable by eye.

3.3. Identifying fasted blood samples with CCC

I classified biological samples using flow cytometry blood sampled measured with T-cell cell-parameters. As explained in the methods (section 2.4), a blood sample was measured before (ANTE) or after (POST) glucose-intake after 24 hours of fasting. The data are stored with the group of Prof. Dr Spang. Out of the 126 biological samples, 30 were measured with the second plate and used as a separate test set. The leftover 96 biological samples were randomly split into training (60%) and validation (40%) biological samples resulting in 30 ANTE and 27 POST biological samples in the training set, 17 ANTE and 21 POST biological samples in the validation set and 15 ANTE and 15 POST biological samples in the test set. See section 2.4 for a detailed explanation of the data.

Training the model was repeated 50 times with random network initialisations. See subsection 2.4.2 for the hyperparameter settings.

Training, validation and test performance, measured through ROC curves, and the related AUC were perfect 44 of 50 times. When I show a result from a single of the 50 models, I always use the identical model chosen from the 44 well-performing models. The performance in the six imperfect models also dropped in the validation set. I show the perfect ROC curves and AUCs in Figure 3.14.

Cell-features k_1 and k_2 have the highest absolute SHAP values, therefore I replace only them with a decision tree. The SHAP values are shown in Figure 3.15.

Based on these SHAP values, the following gating directions were identified. High values of the cell-features mean of k_1 and k_2 spoke for class POST. In Figure 3.15 this is visualised: High values of a biological sample in red had always positive SHAP values towards the class POST.

The decision tree replacements of the *FeatureLearner* for k_1 and k_2 are shown in Figure 3.16. Figure 3.16 shows (A) the decision tree replacement for cell-feature k_1, (B) the decision tree replacement for cell-feature k_2. (C) and (D) the respective histograms of the cell-feature value over the cells, as explained in Figure 3.7 of Part III.

I found the final gatings, shown in Table 3.3, according to the identified gating directions for mean:k_1 and mean:k_2. For both cell-feature means, there are two possible gates because of the binary classification problem.

- Pro ANTE lower of k_1 resulted in the final gating $CD4^- \mid CCR7^- \mid KLRG1^+$ by traversing Figure 3.4 (A) towards lower node values.
- Pro POST higher of k_1 resulted in the final gating $CD4^+ \mid KLRF1^- \mid FSC\text{-}A^-$ by traversing Figure 3.4 (A) towards higher node values.
- Pro ANTE lower of k_2 resulted in the final gating $CCR7^- \mid CD4^- \mid KLRG1^+$ by traversing Figure 3.4 (B) towards lower node values.
- Pro POST lower of k_2 resulted in the final gating $CCR7^+ \mid KLRF1^- \mid CD4^+$ by traversing Figure 3.4 (B) towards higher node values.

Finally, I applied those four gatings on the test data and visualised the resulting counts per biological sample in the four respective final gates in Figure 3.17. Biological samples of class ANTE had more cells in $CD4^- \mid CCR7^- \mid KLRG1^+$ (A).

Biological samples of class POST had more cells in $CD4^+ \mid KLRF1^- \mid FSC\text{-}A^-$ (B).
Biological samples of class ANTE had more cells in $CCR7^- \mid CD4^- \mid KLRG1^+$ (C).
Biological samples of class POST had more cells in $CCR7^+ \mid KLRF1^- \mid CD4^+$ (D).

Not all 50 models resulted in the same decision tree gatings, there were only two gatings occurring ten out of 50 times. The model was trained with the same hyperparameters using 50 different initialisations. All resulting 3×50 gatings (used cell-feature $\times$ trained models) were calculated on the validation samples and regardless of the SHAP values. All gatings are shown in Figure G 1. In Figure 3.18 the full graph of Figure G 1 was restricted to gating paths where at least 10 gatings traverse. Here only two gatings reached the final decision tree nodes: $CD57^- \mid KLRF1^- \mid CCR7^+$ reaching POST and $CD57^+ \mid CD27^- \mid KLRF1^+$ reaching ANTE.

When I applied those two gatings on the test biological samples, the number of cells inside those gatings reliably differentiated the two classes ANTE and POST. In Figure 3.19 I show the cell counts of these test samples. Biological samples with high counts in $CD57^- \mid KLRF1^- \mid CCR7^+$ were identified as class POST. Biological samples with high counts in $CD57^+ \mid CD27^- \mid KLRF1^+$ were identified as class ANTE.

The results shown here were directly taken from the minimum working examples `01_MWE/26_ccc_MWEs/01_FastingGlucose` from the CCC implementation, accessible via GitLab from the group of Prof. Dr Spang.

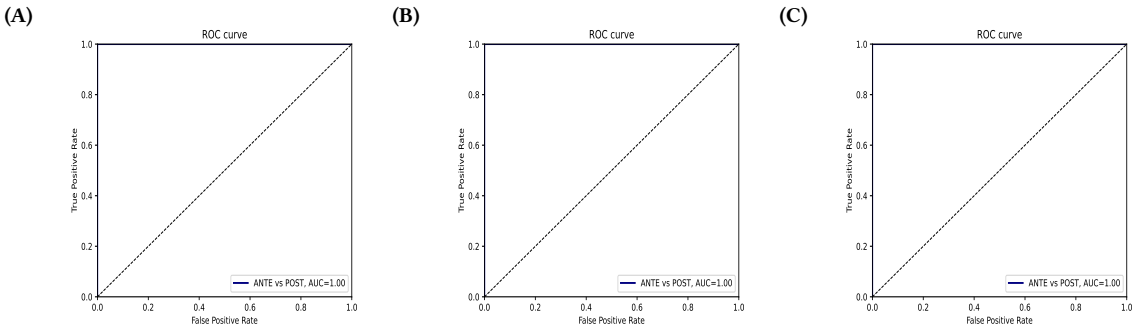


Figure 3.14.: Fasting: ROC curves of one final model's predictions on training (A), validation (B) and test(C) samples.

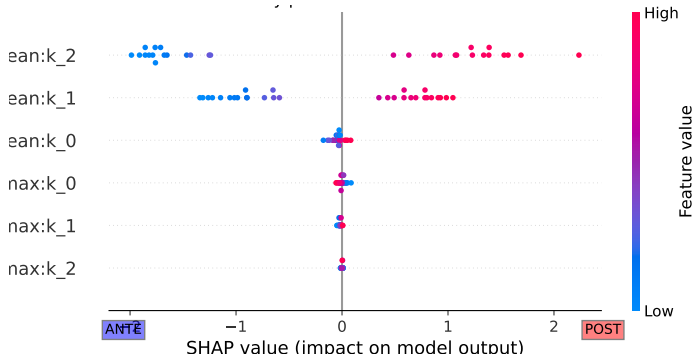


Figure 3.15.: Fasting: SHAP values. The SHAP values of all maximum poolings were close to zero. Mean poolings had clear, separated effects: High values of the cell-features mean of k_1 and k_2 spoke for class POST. The SHAP values of k_0 were close to zero.

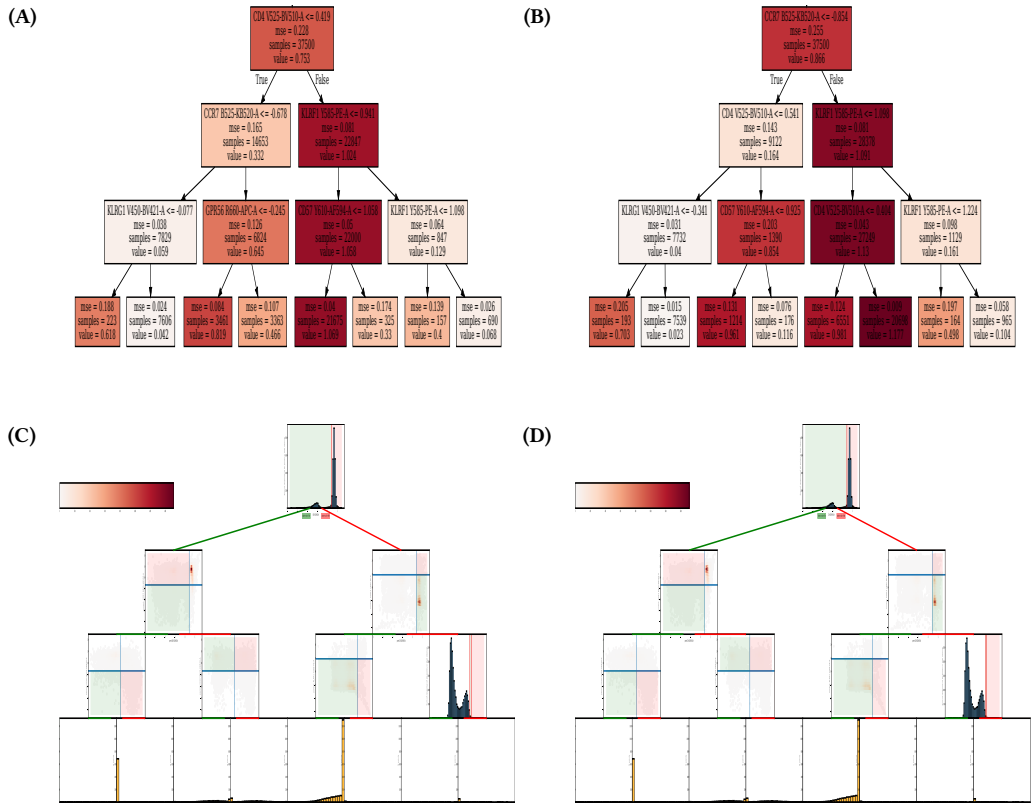


Figure 3.16.: Fasting: Cell-feature decision trees of the final model for the test biological samples. (A) shows the decision tree for cell-feature k_1 (C) the respective histograms of the cell-feature value over the cells with the cell-parameters as axes. (B) shows the decision tree for cell-feature k_2 (D) the cells fit into that decision tree. I only show trees trained on cells from biological samples of the ANTE class as POST class trees were almost identical. See Part III for a detailed explanation of the plots.

Feature	Pooling	Class	pro/contra	Direction	cut_1	cut_2	cut_3
k_1	mean	ANTE	pro	lower	$CD4^-$	$CCR7^-$	$KLRG1^+$
k_1	mean	POST	pro	higher	$CD4^+$	$KLRF1^-$	$FSC\text{-}A^-$
k_2	mean	ANTE	pro	lower	$CCR7^-$	$CD4^-$	$KLRG1^+$
k_2	mean	POST	pro	higher	$CCR7^+$	$KLRF1^-$	$CD4^+$

Table 3.3.: Fasting: Gatings based on gating directions from Figure 3.15 and the decision trees shown in Figure 3.16

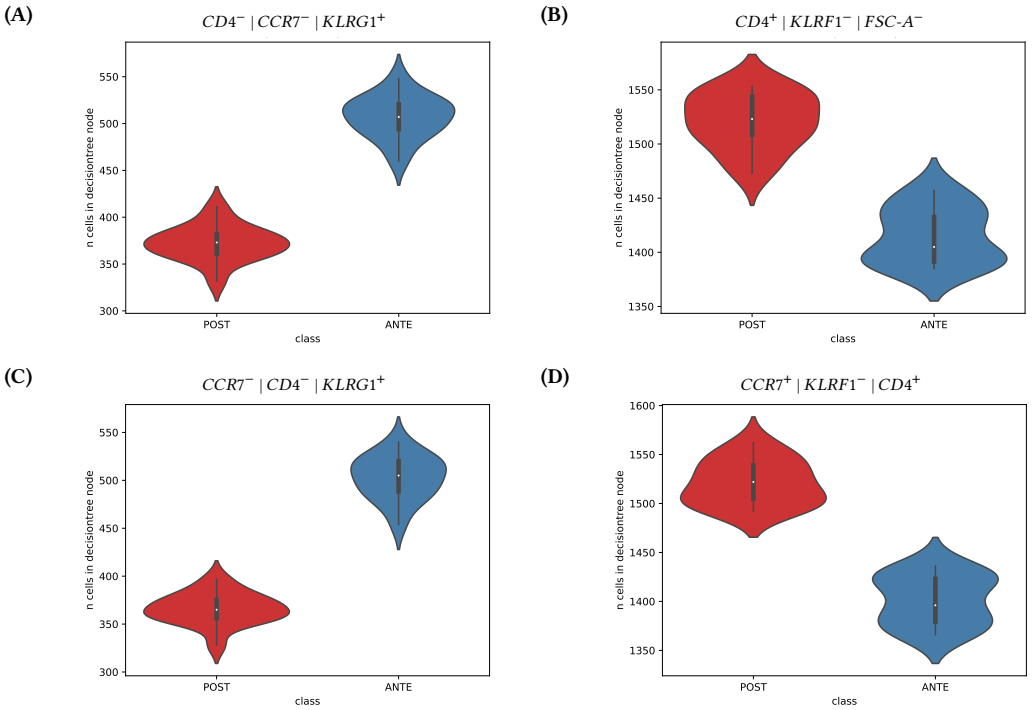


Figure 3.17.: Fasting: Applied gatings to the test samples. Counts for each sample are shown in boxplots for all test samples. The gatings are based on Figure 3.16 and Table 3.3. (A) (k_1) and (C) (k_2) show the identified gating for k_1 when going towards lower activation values, so "pro ANTE". We see that if a sample is actually in class ANTE (blue), there are more cells in that gate. (B) (k_1) and (D) (k_2) show the identified gating when going towards higher activation values, so "pro POST". We see that POST samples have more cells in this gate. Besides, we see that the gatings identified for k_1 and k_2 are almost identical here.

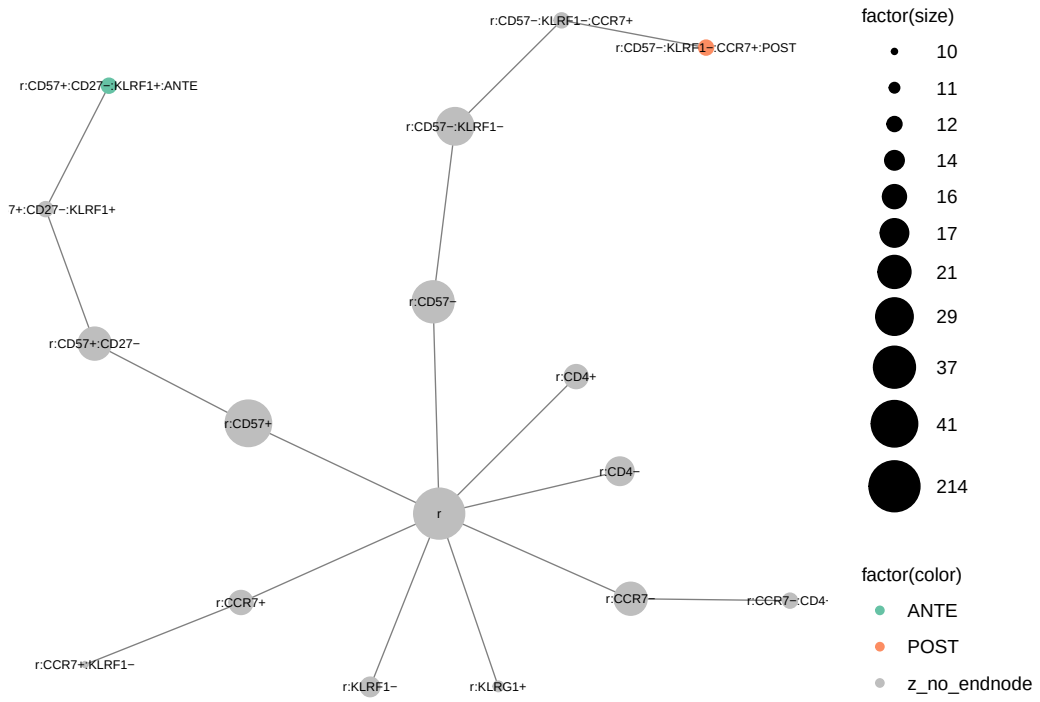


Figure 3.18.: Fasting: Repeated gatings graph. This graph shows the gatings from 50runs and over all (possibly chosen) 3 pooled features. Every gating starts from the root in the centre r. Each gating can be read by traversing the graph going to the outside and finally reaching a green or orange node which reflects the final node for an ANTE or POST gating, respectively. To not overcrowd this plot, I only show here nodes where at least ten gatings match that path. Therefore some gatings are not finished, e.g. r:CD4+ stops after that node because the ten contained gatings split up into less frequent than ten gatings afterwards. We see that only one gating reaches each ANTE and POST at least ten times: $CD57^- \mid KLRF1^- \mid CCR7^+$ reaching POST and $CD57^+ \mid CD27^- \mid KLRF1^+$ reaching ANTE. For the complete gating graph see the Figure G 1

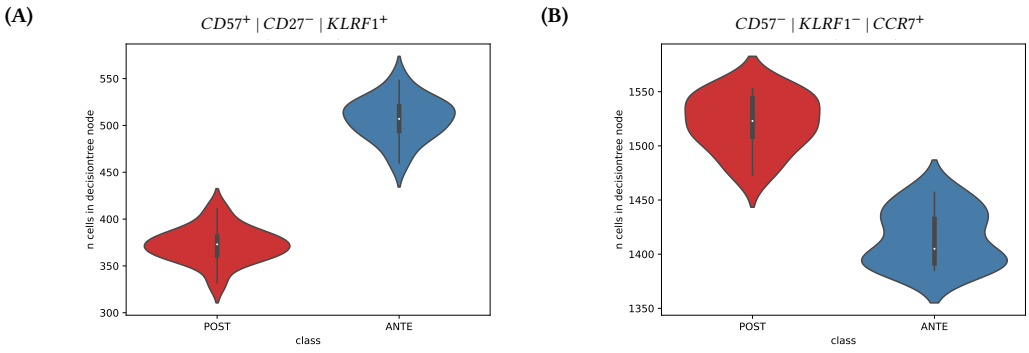


Figure 3.19.: Fasting: Most prevalent gating counts. Counts on the test set for the most prevalent gatings identified in Figure 3.18

4. Discussion

Cytometry characterises cells of a biological sample by cell-properties like size, complexity or surface markers. A measured cell-property is called a cell-parameter, which defines each cell by numerical values suitable for further analysis. Up to 50 measured cell-parameters characterise each cell[5], and the resulting data is a matrix of unordered cells in the rows, and some cell-parameters as columns. A biological sample contains millions of cells, and their characterisation can be used to define cohesive cell subpopulations which reflect certain anatomic, morphologic or functional types of cells[42].

In this thesis, I use cytometry data for predictive modelling, where a class is assigned according to all cells of a biological sample together. This class enables clinical patient decisions when the class is known to respond to a certain treatment. Due to the unordered nature of cytometry measurements, only the presence or quantity of cells define the class of a biological sample but not which cell was measured first. Due to the unordered nature, special predictive modelling methods are necessary. Predictive modelling for cytometry data needs special algorithms to factor in the unordered cells.

In the previous part of this thesis, I introduced the FPP concept for predictive modelling with cytometry data. FPP is a modular structure for neural networks containing three parts. First, the *FeatureLearner* transforms the measured matrix of a biological sample into a new matrix with new, predictive cell-features. Second, the *Pooler* aggregates these cell-features into a composition vector of values per biological sample. Third, the *Predictor* uses this composition vector to predict the final class. Therefore, the FPP concept is suitable for predictive modelling with cytometry data.

Predictive modelling with the FPP concept needs multiple user-defined hyperparameters. 1) The architecture of each FPP part defining which layers should be used and how nodes are connected. 2) A pseudo-hyperparameter, the initialisation of the internal weights of the neural network through a seed for the random number generator. 3) The subsampling used by the FPP network. Each unique combination of hyperparameters, excluding the initialisation seed, is called a model variant. Multiple variants which are trained with the same initialisation seed are called a run.

In this part, I applied the FPP concept using my python package CCC on two simulated and one real dataset and show the explanatory power from decision tree replacements of the neural network.
The binary classification simulation had 100 simulated biological sample from two classes where one of three cell subpopulations defined by two cell-parameters was either present with 50% of all cells or completely missing. After training on a part of the samples, the classification of the left out samples was perfect. Also

the discriminative cell subpopulations, cells drawn from $\mathcal{N}_1$, was found through decision tree gatings.
The multiclass classification simulation had 100 simulated biological sample from eight classes. Each class was defined by the percentage of cells drawn from one of the three cell subpopulations. After training on a part of the samples, the classification was in line with subsampling-related theoretical boundaries. Also, the discriminative cell subpopulations, cells drawn from $\mathcal{N}_1$, was found through decision tree gatings.
The real dataset were 63 paired flow cytometry blood samples from a starved (ANTE) patient or after glucose-intake (POST). This binary classification problem was perfectly classifiable after training on a part of the samples. Decision tree gating identified four cell subpopulations where all four were predictive when counting the cells inside for the left out biological samples.

In the binary simulation, I simulated two classes of biological samples. 10^5 cells with two cell-parameters were simulated per biological sample. Class $p_{0.5}$ was defined by drawing 50% of all cells from one of three normal distributions, class p_0 drew no cells from the same distribution.

A simple FPP neural network with 2500 input cells trained on 40 simulated biological samples was sufficient to gain perfect classification performance on the 40 test samples. The network was defined with two *FeatureLearner* layers, mean and maximum pooling layer and one *Predictor* layer where 2500 cells were the input of one biological sample.

Cell-feature k_1 drove the final prediction, as in Figure 3.3 its SHAP values had the highest absolute value. The SHAP-values in Figure 3.3 showed that the *mean* of k_1 mainly drove the final prediction. All maximum poolings were unimportant. Because k_1 was most important, the *FeatureLearner* was only replaced for cell-feature k_1.

Cell-feature k_1 identified cells coming from the variable cell subpopulation of cells drawn from $\mathcal{N}_1$. The generated decision tree identified cells with high cell-feature values with the gating $f_0^+ \mid f_1^- \mid f_1^-$. This is exactly the cell subpopulation of cells drawn from $\mathcal{N}_1$. Compare the simulated biological samples in Figure 3.1 to the 2D-histogram in the right branch, first depth of Figure 3.4. On the x-axis is f_0 and on the y-axis f_1, cells on the bottom right ($f_0^+ \wedge f_1^-$) were the cells receiving high cell-feature values. These are the cells which were drawn from $\mathcal{N}_1$.

When high values of a cell-feature speak for one class, low values of that cell-feature speak for the other class in a binary classification problem. If a cell-feature speaks for or against a class is defined by the gating direction, which is identified through SHAP values. Therefore, each important cell-feature results in two gatings. Importance of a cell-feature for the prediction is also found by the SHAP values.

From cell-feature k_1, the first generated gating $f_0^+ \mid f_1^- \mid f_1^-$ spoke for class $p_{0.5}$, and its second, converse gating $f_0^- \mid f_0^- \mid f_0^-$ spoke for class p_0. I applied those gatings on the test data and counted the cells per biological sample per gating. Simulated biological samples of $p_{0.5}$ had more cells in gating $f_0^+ \mid f_1^- \mid f_1^-$: Almost zero cells were in this gating when the simulated biological samples was actually from class p_0. Simulated biological samples of class p_0 had more cells in gating $f_0^- \mid f_0^- \mid f_0^-$, but samples of class had also cells inside. This is in concordance

to the simulation as the first gating identified the either present or completely missing cell subpopulation. In contrast, the second gating identified cells from a cell subpopulation in both classes.

The second gating, $f_0^- \mid f_0^- \mid f_0^-$, identified cells from cell subpopulations present in both classes. This might seem confusing, but it is still a discriminative cell subpopulations because the total number of generated cells was always the same for both classes of simulated biological samples. If the percentage of cells drawn from $\mathcal{N}_1$ is either 50% or 0%, also the percentage of cells in the other two multivariate normal distributions changes. Therefore not only $\mathcal{N}_1$ itself is a discriminative cell subpopulations but also $f_0^- \mid f_0^- \mid f_0^-$.

With this simulation, I showed how to apply the python package CCC on binary classification problems. The simulation is comparably easy, and any classification algorithm should be able to classify in this setting perfectly, but here I show the general analysis path and the related reasoning. Except for Figure 3.1 where I show the simulated data itself, all plots were the direct output of the CCC package.

In the multiclass simulation, I created eight classes of samples. 10^5 cells with two cell-parameters were simulated per biological sample. Each class was defined by the percentage of all cells drawn from one of three multivariate normal distributions ($\mathcal{N}_1$).

A simple FPP neural network with 2500 input cells trained on 480 simulated biological samples was sufficient to gain good classification performance on the 160 test samples. The network was defined with two *FeatureLearner* layers, mean and maximum pooling layer and one *Predictor* layer with eight output nodes where 2500 cells were the input of one biological sample.

The classification performance was in line with theoretical boundaries. The classification performance was measured using one-vs-all ROC curves and the resulting AUC. The test AUCs were perfect for classes $p_{0.5}$ and $p_{0.2}$ (AUC = 1). The test AUCs had good performance for $p_{0.01}$ (AUC = 0.99). Some of the simulated biological samples in class $p_{0.001}$ can be identified, resulting in an AUC of 0.89. Then the AUC drops - but not lower than 0.76 for each class. This is perfectly in line of simulation due to the reason that the *zero-like* biological samples are not distinguishable from each other but fairly distinct from $p_{0.5}$, $p_{0.2}$ and $p_{0.01}$. This becomes also visible in Figure 3.10 where the predicted probabilities for each class are shown. The probabilities for $p_{0.5}$, $p_{0.2}$ and $p_{0.01}$ are clear cut from the other classes but the *zero-like* classes have almost the same probability for every *zero-like* class. It is not surprising that biological samples falsely predicted as non-*zero-like* were most times predicted as class $p_{0.001}$, the class most similar to the *zero-like* classes.

zero-like biological samples are not distinguishable from each other. For biological samples with a $\mathcal{N}_1$ percentage of 10^{-5} there is an expected value of one single cell drawn from the $\mathcal{N}_1$-distribution. Thus even a perfect algorithm cannot distinguish the *zero-like* classes $p_{10^{-5}}$, $p_{10^{-6}}$ and p_0. Even more difficult: The network drew only 2500 random cells when predicting a single biological sample. Thus the network cannot reliably distinguish biological samples with less than $\frac{1}{2500} = 4e-4$, including it in the *zero-like* class. To visualise this problem, I showed the same plots as in Figure 3.2 but only 2500 cells from each biological sample were shown in Figure 3.7.

In Figure 3.8 we see the number of cells truly drawn from $\mathcal{N}_1$ for all cells and a random subset of 2500 cells. We see that already for class 0.001, the *median* number of cells from $\mathcal{N}_1$ distribution in a biological sample is 2, rendering also this class very hard to distinguish from *zero-like* even if the theoretical limit is not reached.

Cell-feature k_2 was most important for the prediction of all classes. Multiclass classification results in SHAP values for each predicted class, here mean or maximum pooling of cell-feature k_2 were always most important. Cell-feature k_2 was so strong that no observed value of the other two cell-features was able to overthrow the prediction of k_2.

There are eight gatings for cell-feature k_2. The gatings were based on the gating direction calculated on the mean-pooled cell-feature values. The resulting gatings were very similar across classes. $f_0^- \mid f_1^- \mid f_1^-$ for *zero-like*, $p_{0.001}$ and $p_{0.01}$ classes and $f_0^+ \mid f_1^- \mid f_0^+$ for $p_{0.2}$ and $p_{0.5}$. Comparing to the simulated biological sample visualization in Figure 3.6 we see that $f_0^- \mid f_1^- \mid f_1^-$ focusses on the bottom-left cell subpopulation which is present in all classes but varies in percentage of cells. Similarly, $f_0^+ \mid f_1^- \mid f_0^+$ for $p_{0.2}$ and $p_{0.5}$ focusses on the bottom-right cell subpopulation drawn from $\mathcal{N}_1$, exactly the cell subpopulation I actively changed in percentage.

Applying the two gatings to the 160 simulated test biological sample we see clear differences in the number of cells per gating and class except for the *zero-like* classes as expected. Cell counts are shown in boxplots for all test samples and each class in Figure 3.17. $f_0^+ \mid f_1^- \mid f_0^+$, the gating towards higher (="Direction") cell-feature k_2 values, correlated with higher probability towards high-percentage classes $p_{0.2}$ and $p_{0.5}$. Class $p_{0.5}$ had the highest counts, followed by $p_{0.2}$ and also $p_{0.01}$ had slightly elevated counts compared to the other classes which reflects the simulation. $f_0^- \mid f_1^- \mid f_1^-$, the gating towards lower (="Direction") cell-feature k_2 values, spoke for classes *zero-like*, $p_{0.001}$ and $p_{0.01}$. I saw the mirrored picture the for the other gating: *zero-like*and $p_{0.001}$ had almost identical, high number of cells in that gating, $p_{0.01}$ had slightly decreased counts, $p_{0.2}$ had clearly decreased counts and $p_{0.5}$ had the lowest counts.

With this simulation, I showed how to apply the python package CCC on multiclass classification problems. I showed the general analysis path and the related reasoning. Except for Figure 3.2 and Figure 3.7 where I show the simulated and subsampled data, all plots were the direct output of the CCC package.

In the real data, biological blood samples were measured with flow cytometry of T-cell cell-parameters. 63 paired blood biological samples of two classes were measured. In the ANTE class, one patient was measured after starving for 24 hours. After glucose-intake, the same patient was measured after one hour (POST class).

I fixed the hyperparameters such that 2500 cells were put into the *FeatureLearner*. The *FeatureLearner* consisted of two layers with three nodes each. Therefore, 19 cell-parameters were densely connected to three nodes and those again densely connected to three nodes. Each layer was followed by batch normalisation[158] and a ReLU-function[155]. The *Pooler* aggregated the 2500 cells with the mean *and* the maximum resulting in $k * q = 3 \times 2 = 6$ pooled cell-features. The *Predictor* used those in one layer with three nodes, also followed by batch normalisation[158] and ReLU function[155]. Those were then combined into a single node, and a sigmoid function ensured 0/1 output.

A simple FPP neural network with 2500 input cells trained on 57 biological samples was sufficient to gain excellent classification performance. 38 biological samples were used as validation data and final classification performance was assessed with ROC-curves and their AUC on 30 biological samples. The 30 test samples were measured on a separate plate. The network got 19 cell-parameters as input per cell and was defined with two *FeatureLearner* layers, mean and maximum pooling, and one output layer.

The model training was repeated 50 times to assess the algorithm's stability. 44 of those trained models classified all test samples perfectly (AUC = 1), and the other six models could have been excluded as their validation performance was also imperfect.

I chose one exemplary of the 44 perfect models and its identified gatings. cell-features k_1 and k_2 were most important according to the calculated SHAP values. Therefore, decision tree replacements for k_1 and k_2 were built. For both cell-features, high values spoke for the class POST. The resulting gatings $CD4^+ \mid KLRF1^- \mid FSC\text{-}A^-$ and $CCR7^+ \mid KLRF1^- \mid CD4^+$ spoke for POST. The gatings $CD4^- \mid CCR7^- \mid KLRG1^+$ and $CCR7^- \mid CD4^- \mid KLRG1^+$ spoke for ANTE. The hierarchical order switched, but $CD4^+$ and $KLRF1^+$ cells spoke for POST in both pro-POST gatings.

To assess the stability of the identified gatings, all gatings of all 50 training repetitions were identified, regardless of their cell-feature's SHAP values. Here only two gatings were present at least 10 times in 150 gatings (3 cell-features and 50 repetitions). $CD57^- \mid KLRF1^- \mid CCR7^+$ spoke for class POST. $CD57^+ \mid CD27^- \mid KLRF1^+$ spoke for class ANTE.

All identified gatings do not follow classical gating schemes. Usually $CD45^+$ cells (Leukocytes) and $CD3^+$ (T-cells) are identified first, followed by $CD4^+$ or $CD8^+$ T-cells which provide different functions[5]. However, after the classification performance is near perfect, it seems that the network does not need the information if a cell is a T-cell or not but starts directly differentiating the cell-parameter $CD4$. The two overlapping gatings start differentiating $CD57$, which is found on a subset of cells with natural killer activity, but it does not tell which cell it actually is in a classical sense. However, these gatings might lead to new insights which cell subpopulations are relevant.

All six gatings were applied to the test biological samples and showed clear differences in the number of cells in each gating from POST to ANTE biological samples.

With this analysis, I showed the application of the python package CCC on a real-world flow cytometry binary classification problem. All plots were the direct output of the CCC package.

In summary, I applied the python package CCC to two simulated and one real dataset and investigated the results. The classification performances were as good as expected for the binary classification and multiclass classification simulations and also great for the real fasting-dataset where fasted and non-fasted blood biological sample were differentiated. The found gatings in the simulations correctly identified the varied cell subpopulations. The found gatings for the fasting dataset were able to differentiate between the two classes but did not resemble classical gating schemes in flow cytometry. The results are promising, and it will be interesting to see future

applications of CCC.

Bibliography

1. Qi, C. R., Yi, L., Su, H. & Guibas, L. J. *PointNet++: deep hierarchical feature learning on point sets in a metric space* 2017. http://dl.acm.org/ft_gateway.cfm?id=3295263&type=pdf.
2. Arvaniti, E. & Claassen, M. Sensitive detection of rare disease-associated cell subsets via representation learning. *Nature Communications* **8,** 14825. ISSN: 2041-1723. https://www.nature.com/articles/ncomms14825.pdf (2017).
3. Hu, Z., Tang, A., Singh, J., Bhattacharya, S. & Butte, A. J. A robust and interpretable end-to-end deep learning model for cytometry data. *Proceedings of the National Academy of Sciences* **117,** 21373–21380. ISSN: 1091-6490. https://www.pnas.org/content/117/35/21373 (2020).
4. Lee, J. A. *et al.* MIFlowCyt: the minimum information about a Flow Cytometry Experiment. *Cytometry Part A* **73,** 926–930. ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/full/10.1002/cyto.a.20623 (2008).
5. Cossarizza, A. *et al.* Guidelines for the use of flow cytometry and cell sorting in immunological studies (second edition). *European journal of immunology* **49,** 1457–1973 (2019).
6. Hu, Z., Glicksberg, B. S. & Butte, A. J. Robust prediction of clinical outcomes using cytometry data. *Bioinformatics (Oxford, England)* **35,** 1197–1203 (2019).
7. Aghaeepour, N. *et al.* Critical assessment of automated flow cytometry data analysis techniques. *Nature Methods* **10,** 228–238. ISSN: 1548-7105. https://www.nature.com/articles/nmeth.2365.pdf (2013).
8. Ji, D. *et al.* Machine Learning of Discriminative Gate Locations for Clinical Diagnosis. *Cytometry Part A* **97,** 296–307. ISSN: 1552-4930. https://pubmed.ncbi.nlm.nih.gov/31691488/ (2020).
9. Roederer, M., Moore, W., Treister, A., Hardy, R. R. & Herzenberg, L. A. Probability binning comparison: a metric for quantitating multivariate distribution differences. *Cytometry* **45,** 47–55. ISSN: 0196-4763. https://onlinelibrary.wiley.com/doi/10.1002/1097-0320(20010901)45:1%3C47::AID-CYTO1143%3E3.0.CO;2-A (2001).
10. Roederer, M. & Hardy, R. R. Frequency difference gating: A multivariate method for identifying subsets that differ between samples. *Cytometry* **45,** 56–64. ISSN: 0196-4763. https://onlinelibrary.wiley.com/doi/10.1002/1097-0320(20010901)45:1%3C56::AID-CYTO1144%3E3.0.CO;2-9#bib2 (2001).
11. Lun, A. T. L., Richard, A. C. & Marioni, J. C. Testing for differential abundance in mass cytometry data. *Nature Methods* **14,** 707–709. ISSN: 1548-7105. https://www.nature.com/articles/nmeth.4295.pdf (2017).

12. McCarthy, D. J., Chen, Y. & Smyth, G. K. Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Research* **40,** 4288–4297. ISSN: 0305-1048. https://academic.oup.com/nar/article/40/10/4288/2411520 (2012).

13. Weber, L. M., Nowicka, M., Soneson, C. & Robinson, M. D. diffcyt: Differential discovery in high-dimensional cytometry via high-resolution clustering. *Communications Biology* **2,** 1–11. ISSN: 2399-3642. https://www.nature.com/articles/s42003-019-0415-5.pdf (2019).

14. Mair, F. *et al.* The end of gating? An introduction to automated analysis of high dimensional cytometry data. *European journal of immunology* **46,** 34–43. https://onlinelibrary.wiley.com/doi/full/10.1002/eji.201545774 (2016).

15. Zare, H. *et al.* Automated analysis of multidimensional flow cytometry data improves diagnostic accuracy between mantle cell lymphoma and small lymphocytic lymphoma. *American journal of clinical pathology* **137,** 75–85. ISSN: 1943-7722. https://pubmed.ncbi.nlm.nih.gov/22180480/ (2012).

16. Zare, H., Shooshtari, P., Gupta, A. & Brinkman, R. R. Data reduction for spectral clustering to analyze high throughput flow cytometry data. *BMC Bioinformatics* **11,** 403. ISSN: 1471-2105. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-403 (2010).

17. Zare, H., Haffari, G., Gupta, A. & Brinkman, R. R. Scoring relevancy of features based on combinatorial analysis of Lasso with application to lymphoma diagnosis. *BMC Genomics* **14 Suppl 1,** S14 (2013).

18. Bach, F. *Model-Consistent Sparse Estimation through the Bootstrap* 2009. https://arxiv.org/pdf/0901.3202.

19. Qiu, P. *et al.* Extracting a cellular hierarchy from high-dimensional cytometry data with SPADE. *Nature biotechnology* **29,** 886–891. ISSN: 1087-0156 (2011).

20. Rubner, Y. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision* **40,** 99–121. ISSN: 1573-1405. https://link.springer.com/article/10.1023/a:1026543900054 (2000).

21. Qiu, P. Inferring phenotypic properties from single-cell characteristics. *PLOS ONE* **7,** e37038. ISSN: 1932-6203 (2012).

22. Aghaeepour, N. *et al.* Early immunologic correlates of HIV protection can be identified from computational analysis of complex multivariate T-cell flow cytometry assays. *Bioinformatics (Oxford, England)* **28,** 1009–1016. https://academic.oup.com/bioinformatics/article/28/7/1009/211200 (2012).

23. Aghaeepour, N. *et al.* RchyOptimyx: cellular hierarchy optimization for flow cytometry. *Cytometry Part A* **81,** 1022–1030. ISSN: 1552-4930 (2012).

24. O'Neill, K., Jalali, A., Aghaeepour, N., Hoos, H. & Brinkman, R. R. Enhanced flowType/RchyOptimyx: a BioConductor pipeline for discovery in high-dimensional cytometry data. *Bioinformatics (Oxford, England)* **30,** 1329–1330 (2014).

25. Aghaeepour, N., Nikolic, R., Hoos, H. H. & Brinkman, R. R. Rapid cell population identification in flow cytometry data. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **79,** 6–13 (2011).

26. Bruggner, R. V., Bodenmiller, B., Dill, D. L., Tibshirani, R. J. & Nolan, G. P. Automated identification of stratifying signatures in cellular subpopulations. *Proceedings of the National Academy of Sciences* **111,** E2770–E2777. ISSN: 1091-6490. https://www.pnas.org/content/pnas/111/26/E2770.full.pdf (2014).

27. van Gassen, S., Vens, C., Dhaene, T., Lambrecht, B. N. & Saeys, Y. FloReMi: Flow density survival regression using minimal feature redundancy. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **89,** 22–29 (2016).

28. Rebhahn, J. A. *et al.* Competitive SWIFT cluster templates enhance detection of aging changes. *Cytometry Part A* **89,** 59–70. ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/full/10.1002/cyto.a.22740 (2016).

29. Mosmann, T. R. *et al.* SWIFT-scalable clustering for automated identification of rare cell populations in large, high-dimensional flow cytometry datasets, part 2: biological evaluation. *Cytometry Part A* **85,** 422–433. ISSN: 1552-4930 (2014).

30. Hu, Z. *et al.* MetaCyto: A Tool for Automated Meta-analysis of Mass and Flow Cytometry Data. *Cell Reports* **24,** 1377–1388. ISSN: 2211-1247. https://www.sciencedirect.com/science/article/pii/S2211124718310805 (2018).

31. Fonseka, C. Y. *et al.* Mixed-effects association of single cells identifies an expanded effector CD4+ T cell subset in rheumatoid arthritis. *Science translational medicine* **10** (2018).

32. Nowicka, M. *et al.* CyTOF workflow: Differential discovery in high-throughput high-dimensional cytometry datasets. *F1000Research* **6,** 748. https://f1000research.com/articles/6-748/v1 (2017).

33. Yue, A., Chauve, C., Libbrecht, M. & Brinkman, R. R. Automated identification of maximal differential cell populations in flow cytometry data. *bioRxiv,* 837765. https://www.biorxiv.org/content/10.1101/837765v5 (2019).

34. Finak, G. *et al.* Mixture models for single-cell assays with applications to vaccine studies. *Biostatistics (Oxford, England)* **15,** 87–101. https://academic.oup.com/biostatistics/article/15/1/87/244741 (2014).

35. Lin, L. *et al.* COMPASS identifies T-cell subsets correlated with clinical outcomes. *Nature Biotechnology* **33,** 610–616. ISSN: 1546-1696. https://pubmed.ncbi.nlm.nih.gov/26006008/ (2015).

36. van Gassen, S. *et al.* FlowSOM: Using self-organizing maps for visualization and interpretation of cytometry data. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **87,** 636–645 (2015).

37. Ritchie, M. E. *et al.* limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research* **43,** e47. ISSN: 0305-1048. https://academic.oup.com/nar/article/43/7/e47/2414268 (2015).

38. Law, C. W., Chen, Y., Shi, W. & Smyth, G. K. voom: Precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology* **15,** R29. ISSN: 1474-760X. https://genomebiology.biomedcentral.com/articles/10.1186/gb-2014-15-2-r29 (2014).

39. Stanley, N. *et al.* VoPo leverages cellular heterogeneity for predictive modeling of single-cell data. *Nature Communications* **11,** 3738. ISSN: 2041-1723. https://www.nature.com/articles/s41467-020-17569-8 (2020).

40. He, X., Cai, D. & Niyogi, P. Laplacian Score for Feature Selection. *Advances in Neural Information Processing Systems* **18.** https://proceedings.neurips.cc/paper/2909-laplacian-score-for-feature-selection (2005).

41. Tuchin, V. V. *Advanced Optical Flow Cytometry: Methods and Disease Diagnoses* ISBN: 9783527409341 (John Wiley & Sons, Hoboken, 2012).

42. Panina, Y., Karagiannis, P., Kurtz, A., Stacey, G. N. & Fujibuchi, W. Human Cell Atlas and cell-type authentication for regenerative medicine. *Experimental & molecular medicine* **52,** 1443–1451 (2020).

43. Kierano. *Schematic diagram of a flow cytometer, from sheath focusing to data acquisition.* File: figures/img/48-flowCyto/1920px-Cytometer.svg.png. https://en.wikipedia.org/wiki/Flow_cytometry#/media/File:Cytometer.svg. (CC BY 3.0 CC BY).

44. Goetz, C., Hammerbeck, C. & Bonnevier, J. *Flow Cytometry Basics for the Non-Expert* ISBN: 9783319980713 (Springer International Publishing, Cham, 2018).

45. Engel, P. *et al.* CD Nomenclature 2015: Human Leukocyte Differentiation Antigen Workshops as a Driving Force in Immunology. *The Journal of Immunology* **195,** 4555–4563. ISSN: 1550-6606 (2015).

46. Nha, B. D. *Mathcha: Online Mathematics editor, a fast way to write and share mathematics* https://www.mathcha.io/.

47. Ferran Cardoso & Xiao Qin. *CyGNAL: Pipeline for analysing and visualising CyTOF datasets, with a focus on PTM signalling networks.* 2021. https://github.com/TAPE-Lab/CyGNAL.

48. Scott White. FlowKit: Intuitive Python framework for flow cytometry analysis and visualization, including GatingML support. *Google Scholar.* https://github.com/whitews/FlowKit (2020).

49. Burton, R. J. *et al.* CytoPy: An autonomous cytometry analysis framework. *PLOS Computational Biology* **17,** e1009071. ISSN: 1553-7358 (2021).

50. Alexander Kononov. *cytofBrowser: Analysis and visualisation single-cell proteomic data (CyTOF)* 2021. https://github.com/AlexanderKononov/cytofBrowser.

51. Amy Fox. *cytotypr: Pipeline to analyze flow cytometry data* 2021. https://github.com/aef1004/cytotypr.

52. Jim Java. *flowpipe: flow&mass cytometry pipeline* 2021. https://github.com/priscian/flowpipe.

53. Dillon Hammill. *cytoSuite: Compensation & Gating Routines for Analysis of Flow Cytometry Data* 2018. https://github.com/ClaudiaMY/cytoSuite.

54. Dominik Orliński. *cytometry_web_app: Tool for analyzing cytometric data* 2018. https://github.com/Dominnio/cytometry_web_app.

55. GitHub, s. *plateypus: Processes and plots high throughput cytometry experiments through a GUI* 2021. https://github.com/soorajachar/plateypus.

56. David Novak. *tidycell: Framework for discovery of aberrant populations in cytometry data* 2020. https://github.com/davnovak/tidycell.

57. Yincong Zhou & Jitong Xue. *CytoSEE: A Web-based toolkit for automatic computation and evaluation of cytometry data* 2019. https://github.com/mingchen-lab/cytosee.

58. Mirek Kratochvil & Abhishek Koladiya. *panelbuilder: Panel optimization and unmixing tool for multicolor and spectral cytometry* 2021. https://github.com/exaexa/panelbuilder.

59. Christina Bligaard Pedersen. *cyCombine: Robust Integration of Single-Cell Cytometry Datasets* 2021. https://github.com/biosurf/cyCombine.

60. Greg Finak & Jake Wagner. *cytoverse: collections of tools for cytometry data analysis* 2020. https://github.com/RGLab/cytoverse.

61. Ashhurst, T. M. *et al.* Integration, exploration, and analysis of high-dimensional single-cell cytometry data using Spectre. *Cytometry Part A*. ISSN: 1552-4930 (2021).

62. Dillon Hammill & Christopher Hall. *CytoExploreR* 2021. https://github.com/DillonHammill/CytoExploreR.

63. Pier Federico Gherardini. *panorama: R package for interactive visualization and analysis of single-cell data* 2018. https://github.com/ParkerICI/panorama.

64. Bagwell, C. B. & Adams, E. G. Fluorescence spectral overlap compensation for any number of flow cytometry parameters. *Annals of the New York Academy of Sciences* **677,** 167–184. ISSN: 0077-8923 (1993).

65. Nguyen, R., Perfetto, S., Mahnke, Y. D., Chattopadhyay, P. & Roederer, M. Quantifying spillover spreading for comparing instrument performance and aiding in multicolor panel design. *Cytometry Part A* **83,** 306–315. ISSN: 1552-4930 (2013).

66. Anderson, E. Beckman Coulter Life Sciences. *.fcs compensation. E-mail message to author.* (2021).

67. Glehr, B. G. *Screenshot from Fluorescence Spectra Analyzer with two fluorophores* File: figures/img/48-flowCyto/spilloverBiolegend.png. https://www.biolegend.com/en-us/spectra-analyzer.

68. Shapiro, H. M. *Practical flow cytometry* 4. ed. ISBN: 9780471411253 (Wiley-Liss, Hoboken, NJ, 2003).

69. Parks, D. R., Roederer, M. & Moore, W. A. A new "Logicle" display method avoids deceptive effects of logarithmic scaling for low signals and compensated data. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **69,** 541–551. https://onlinelibrary.wiley.com/doi/full/10.1002/cyto.a.20258 (2006).

70. Bagwell, C. B. Hyperlog-a flexible log-like transform for negative, zero, and positive valued data. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **64,** 34–42. https://pubmed.ncbi.nlm.nih.gov/15700280/ (2005).

71. Bickel, P. J. & Doksum, K. A. An Analysis of Transformations Revisited. *Journal of the American Statistical Association* **76,** 296 (1981).

72. Finak, G., Perez, J.-M., Weng, A. & Gottardo, R. Optimizing transformations for automated, high throughput analysis of flow cytometry data. *BMC Bioinformatics* **11,** 546. ISSN: 1471-2105 (2010).

73. Azad, A., Rajwa, B. & Pothen, A. Immunophenotype Discovery, Hierarchical Organization, and Template-Based Classification of Flow Cytometry Samples. *Frontiers in Oncology* **6** (2016).

74. Montante, S. & Brinkman, R. R. Flow cytometry data analysis: Recent tools and algorithms. *International journal of laboratory hematology* **41 Suppl 1,** 56–62 (2019).

75. Reardon, A. J. F., Elliott, J. A. W. & McGann, L. E. Fluorescence as an alternative to light-scatter gating strategies to identify frozen-thawed cells with flow cytometry. *Cryobiology* **69,** 91–99 (2014).

76. Meskas, J., Wang, S. & Brinkman, R. *flowCut — An R package for precise and accurate automated removal of outlier events and flagging of files based on time versus fluorescence analysis* (2020).

77. Annelies Emmaneel. *PeacoQC: Peak-based selection of high quality cytometry data: R package version 1.2.0* 2021. https://github.com/saeyslab/PeacoQC.

78. UCL Computational Systems and Synthetic Biology. *FlopR: An R package for the processing of plate reader and flow cytometry data. This includes: normalisation and calibration of plate reader data, and removing debris and doublets and calibration of flow cytometry data.* 2021. https://github.com/ucl-cssb/flopr.

79. Maria Fernanda Senosain. *denoisingCTF: a CyTOF denoising R package* 2019. https://github.com/msenosain/denoisingCTF.

80. Dendrou, C. A. *et al.* Fluorescence intensity normalisation: correcting for time effects in large-scale flow cytometric analysis. *Advances in bioinformatics,* 476106. ISSN: 1687-8027 (2009).

81. Jake Wagner *et al. flowStats: algorithms for flow cytometry data analysis using BioConductor tools* 2021. https://github.com/RGLab/flowStats.

82. Hahne, F. *et al.* Per-channel basis normalization methods for flow cytometry data. *Cytometry Part A* **77,** 121–131. ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/10.1002/cyto.a.20823 (2010).

83. Gherardini, P. F. *premessa: R package for pre-processing of mass and flow cytometry data* 2020. https://github.com/ParkerICI/premessa.

84. Nikolas Pontikos. *flowBeads: Bioconductor package for working with calibration beads in flow cytometry.* 2016. https://github.com/pontikos/flowBeads.

85. Castillo-Hair, S. M. *et al.* FlowCal: A User-Friendly, Open Source Software Tool for Automatically Converting Flow Cytometry Data from Arbitrary to Calibrated Units. *ACS Synthetic Biology* **5,** 774–780 (2016).

86. Schuyler, R. P. *et al.* Minimizing Batch Effects in Mass Cytometry Data. *Frontiers in Immunology* **10,** 2367. ISSN: 1664-3224. https://www.frontiersin.org/articles/10.3389/fimmu.2019.02367/full (2019).

87. Benny Lo & Timothy Keyes. *Cytofin: An R package for CyTOF data integration* 2021. https://github.com/bennyyclo/Cytofin#additional-information.

88. Molania, R., Gagnon-Bartsch, J. A., Dobrovic, A. & Speed, T. P. A new normalization for Nanostring nCounter gene expression data. *Nucleic Acids Research* **47,** 6073–6083. ISSN: 0305-1048. https://academic.oup.com/nar/article/47/12/6073/5494770 (2019).

89. Trussart, M. *et al.* Removing unwanted variation with CytofRUV to integrate multiple CyTOF datasets. *eLife Sciences Publications, Ltd.* https://elifesciences.org/articles/59630 (2020).

90. van Gassen, S., Gaudilliere, B., Angst, M. S., Saeys, Y. & Aghaeepour, N. CytoNorm: A Normalization Algorithm for Cytometry Data. *Cytometry Part A* **97,** 268–278. ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/10.1002/cyto.a.23904 (2020).

91. Korsunsky, I. *et al.* Fast, sensitive and accurate integration of single-cell data with Harmony. *Nature Methods* **16,** 1289–1296. ISSN: 1548-7105. https://www.nature.com/articles/s41592-019-0619-0 (2019).

92. Sugiyama, M. & Borgwardt, K. M. *Rapid Distance-Based Outlier Detection via Sampling* in (2013), 467–475.

93. Herzenberg, L. A., Tung, J., Moore, W. A., Herzenberg, L. A. & Parks, D. R. Interpreting flow cytometry data: a guide for the perplexed. *Nature immunology* **7,** 681–685. ISSN: 1529-2908. https://www.nature.com/articles/ni0706-681 (2006).

94. Bashashati, A. & Brinkman, R. R. A survey of flow cytometry data analysis methods. *Advances in bioinformatics,* 584603. ISSN: 1687-8027 (2009).

95. Lugli, E., Roederer, M. & Cossarizza, A. Data analysis in flow cytometry: the future just started. *Cytometry Part A* **77,** 705–713. ISSN: 1552-4930. https://pubmed.ncbi.nlm.nih.gov/20583274/ (2010).

96. O'Neill, K., Aghaeepour, N., Spidlen, J. & Brinkman, R. Flow cytometry bioinformatics. *PLOS Computational Biology* **9,** e1003365. ISSN: 1553-7358 (2013).

97. Kvistborg, P. *et al.* Thinking outside the gate: single-cell assessments in multiple dimensions. *Immunity* **42,** 591–592 (2015).

98. Chester, C. & Maecker, H. T. Algorithmic Tools for Mining High-Dimensional Cytometry Data. *The Journal of Immunology* **195,** 773–779. ISSN: 1550-6606 (2015).

99. Saeys, Y., van Gassen, S. & Lambrecht, B. N. Computational flow cytometry: helping to make sense of high-dimensional immunology data. *Nature Reviews Immunology* **16,** 449–462. ISSN: 1474-1741. https://vpn-ur.uni-regensburg.de/proxy/22159cca/https/www.nature.com/articles/nri.2016.56 (2016).

100. Ghaleb, T. A., Mohammed, M. A. & Ramadan, E. *Automated analysis of flow cytometry data: a systematic review of recent methods* in *2016 2nd International Conference on Open Source Software Computing (OSSCOM)* (IEEE, 2016), 1–7. ISBN: 978-1-5090-4580-8.

101. Weber, L. M. & Robinson, M. D. Comparison of clustering methods for high–dimensional single–cell flow and mass cytometry data. *Cytometry Part A* **89,** 1084–1096. ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/full/10.1002/cyto.a.23030 (2016).

102. Kimball, A. K. *et al.* A Beginner's Guide to Analyzing and Visualizing Mass Cytometry Data. *The Journal of Immunology* **200,** 3–22. ISSN: 1550-6606 (2018).

103. Pedreira, C. E. *et al.* From big flow cytometry datasets to smart diagnostic strategies: The EuroFlow approach. *Journal of Immunological Methods* **475,** 112631. ISSN: 0022-1759. http://www.sciencedirect.com/science/article/pii/S0022175919301164 (2019).

104. Palit, S., Heuser, C., de Almeida, G. P., Theis, F. J. & Zielinski, C. E. Meeting the Challenges of High-Dimensional Single-Cell Data Analysis in Immunology. *Frontiers in immunology* **10,** 1515 (2019).

105. Todorov, H. & Saeys, Y. Computational approaches for high-throughput single-cell data analysis. *The FEBS journal* **286,** 1451–1467 (2019).

106. Liu, P. *et al.* Recent Advances in Computer-Assisted Algorithms for Cell Subtype Identification of Cytometry Data. *Frontiers in Cell and Developmental Biology* **8,** 234. ISSN: 2296-634X (2020).

107. Ferrer-Font, L. *et al.* High-Dimensional Data Analysis Algorithms Yield Comparable Results for Mass Cytometry and Spectral Flow Cytometry Data. *Cytometry Part A.* ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/full/10.1002/cyto.a.24016 (2020).

108. Rybakowska, P., Alarcón-Riquelme, M. E. & Marañón, C. Key steps and methods in the experimental design and data analysis of highly multi-parametric flow and mass cytometry. *Computational and Structural Biotechnology Journal* **18,** 874–886. ISSN: 2001-0370. http://www.sciencedirect.com/science/article/pii/S2001037019303848 (2020).

109. Cheung, M. *et al.* Current trends in flow cytometry automated data analysis software. *Cytometry Part A.* ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/full/10.1002/cyto.a.24320 (2021).

110. Baumgaertner, P. *et al.* Unsupervised Analysis of Flow Cytometry Data in a Clinical Setting Captures Cell Diversity and Allows Population Discovery. *Frontiers in Immunology* **12,** 633910. ISSN: 1664-3224. https://www.frontiersin.org/articles/10.3389/fimmu.2021.633910/full (2021).

111. Nakano, A., Harada, T., Morikawa, S. & Kato, Y. Expression of leukocyte common antigen (CD45) on various human leukemia/lymphoma cell lines. *Acta pathologica japonica* **40,** 107–115. ISSN: 0001-6632. https://pubmed.ncbi.nlm.nih.gov/2140233/ (1990).

112. Jiang, M. *cytolib* 2017.

113. Greg Finak, M. J. *flowWorkspace* 2017.

114. Finak, G. *et al.* OpenCyto: an open source infrastructure for scalable, robust, reproducible, and automated, end-to-end flow cytometry data analysis. *PLOS Computational Biology* **10,** e1003806. ISSN: 1553-7358. https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003806 (2014).

115. Finak, G., Jiang, W. & Gottardo, R. CytoML for cross-platform cytometry data sharing. *Cytometry Part A* **93,** 1189–1196. ISSN: 1552-4930 (2018).

116. Geoff Ivison. *boolbins: A library for performing boolean gating on flow cytometry data* 2017. https://github.com/BlishLab/boolbins.

117. Dai, Y. *et al.* CytoTree: an R/Bioconductor package for analysis and visualization of flow and mass cytometry data. *BMC Bioinformatics* **22,** 138. ISSN: 1471-2105. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-04054-2 (2021).

118. Eugene Yurtsev, Guillaume Pelletier, Paul Nobrega & Michael Jared Lumpe. *FlowCytometryTools: A python package for visualization and analysis of high-throughput flow cytometry data* 2021. https://github.com/eyurtsev/FlowCytometryTools.

119. Jacob Frelinger & Adam J Richards. *fcm: A python library for working with flow cytometry data.* 2015. https://github.com/jfrelinger/fcm.

120. Brian Teague. *cytoflow: A Python toolbox for quantitative, reproducible flow cytometry analysis* 2021. https://github.com/cytoflow/cytoflow.

121. Quinn, J. *et al.* A statistical pattern recognition approach for determining cellular viability and lineage phenotype in cultured cells and murine bone marrow. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **71,** 612–624. https://onlinelibrary.wiley.com/doi/10.1002/cyto.a.20416 (2007).

122. Li, H. *et al.* Gating mass cytometry data by deep learning. *Bioinformatics (Oxford, England)* **33,** 3423–3430. https://academic.oup.com/bioinformatics/article/33/21/3423/3964381 (2017).

123. Lux, M. *et al.* flowLearn: fast and precise identification and quality checking of cell populations in flow cytometry. *Bioinformatics (Oxford, England)* **34,** 2245–2253. https://academic.oup.com/bioinformatics/article/34/13/2245/4860364 (2018).

124. Abdelaal, T. *et al.* Predicting Cell Populations in Single Cell Mass Cytometry Data. *Cytometry Part A* **95,** 769–781. ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/10.1002/cyto.a.23738 (2019).

125. Del Barrio, E., Inouzhe, H., Loubes, J.-M., Matrán, C. & Mayo-Íscar, A. optimalFlow: optimal transport approach to flow cytometry gating and population matching. *BMC Bioinformatics* **21,** 479. ISSN: 1471-2105. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-020-03795-w (2020).

126. Malek, M. *et al.* flowDensity: reproducing manual gating of flow cytometry data by automated density-based cell population identification. *Bioinformatics (Oxford, England)* **31,** 606–607. https://pubmed.ncbi.nlm.nih.gov/25378466/ (2015).

127. Lee, H.-C., Kosoy, R., Becker, C. E., Dudley, J. T. & Kidd, B. A. Automated cell type discovery and classification through knowledge transfer. *Bioinformatics (Oxford, England)* **33,** 1689–1695 (2017).

128. Disi Ji, Eric Nalisnick, Yu Qian, Richard H. Scheuermann & Padhraic Smyth. Bayesian Trees for Automated Cytometry Data Analysis. *Machine Learning for Healthcare Conference,* 465–483. ISSN: 2640-3498. http://proceedings.mlr.press/v85/ji18a.html (2018).

129. Girvan, M. & Newman, M. E. J. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America* **99,** 7821–7826. ISSN: 0027-8424. https://www.pnas.org/content/99/12/7821 (2002).

130. Xu, D. & Tian, Y. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science* **2,** 165–193. ISSN: 2198-5812. https://link.springer.com/article/10.1007/s40745-015-0040-1 (2015).

131. Mair, F. Gate to the Future: Computational Analysis of Immunophenotyping Data. *Cytometry Part A* **95,** 147–149. ISSN: 1552-4930. https://onlinelibrary.wiley.com/doi/full/10.1002/cyto.a.23700 (2019).

132. Kotecha, N., Krutzik, P. O. & Irish, J. M. Web-based analysis and publication of flow cytometry experiments. *Current protocols in cytometry / editorial board, J. Paul Robinson, managing editor ... [et al.]* **Chapter 10,** Unit10.17. ISSN: 1934-9297. https://pubmed.ncbi.nlm.nih.gov/20578106/ (2010).

133. Qiu, P. Computational prediction of manually gated rare cells in flow cytometry data. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **87,** 594–602 (2015).

134. Aghaeepour, N. *et al.* A benchmark for evaluation of algorithms for identification of cellular correlates of clinical outcomes. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **89,** 16–21 (2016).

135. Shekhar, K., Brodin, P., Davis, M. M. & Chakraborty, A. K. Automatic Classification of Cellular Expression by Nonlinear Stochastic Embedding (ACCENSE). *Proceedings of the National Academy of Sciences* **111,** 202–207. ISSN: 1091-6490. https://www.pnas.org/content/111/1/202 (2014).

136. Becher, B. *et al.* High-dimensional analysis of the murine myeloid cell system. *Nature Immunology* **15,** 1181–1189. ISSN: 1529-2916. https://www.nature.com/articles/ni.3006 (2014).

137. van der Maaten, L. & Hinton, G. Visualizing Data using t-SNE. *Journal of Machine Learning Research* **9,** 2579–2605. ISSN: 1533-7928 (2008).

138. Abdelaal, T., de Raadt, P., Lelieveldt, B. P. F., Reinders, M. J. T. & Mahfouz, A. SCHNEL: scalable clustering of high dimensional single-cell data. *Bioinformatics (Oxford, England)* **36,** i849–i856. https://academic.oup.com/bioinformatics/article/36/Supplement_2/i849/6055909 (2020).

139. Pezzotti, N., Höllt, T., Lelieveldt, B., Eisemann, E. & Vilanova, A. Hierarchical Stochastic Neighbor Embedding. *Computer Graphics Forum* **35,** 21–30. ISSN: 1467-8659. https://onlinelibrary.wiley.com/doi/10.1111/cgf.12878 (2016).

140. Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* **2008,** P10008. ISSN: 1742-5468. https://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/meta (2008).

141. Lee, S. X., McLachlan, G. J. & Pyne, S. in *Mathematical, computational and experimental T cell immunology* (eds Molina-París, C. & Lythe, G.) 281–294 (Springer, Cham, Switzerland, 2021). ISBN: 978-3-030-57203-7.

142. Pyne, S. *et al.* Automated high-dimensional flow cytometric data analysis. *Proceedings of the National Academy of Sciences* **106,** 8519–8524. ISSN: 1091-6490. https://www.pnas.org/content/106/21/8519 (2009).

143. Lee, S. X., McLachlan, G. J. & Pyne, S. Modeling of inter-sample variation in flow cytometric data with the joint clustering and matching procedure. *Cytometry* **89,** 30–43. ISSN: 0196-4763. https://onlinelibrary.wiley.com/doi/10.1002/cyto.a.22789 (2016).

144. Dundar, M., Akova, F., Yerebakan, H. Z. & Rajwa, B. A non-parametric Bayesian model for joint cell clustering and cluster matching: identification of anomalous sample phenotypes with random effects. *BMC Bioinformatics* **15,** 1–15. ISSN: 1471-2105. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-15-314 (2014).

145. Orlova, D. Y. *et al.* QFMatch: multidimensional flow and mass cytometry samples alignment. *Scientific Reports* **8,** 1–14. ISSN: 2045-2322. https://www.nature.com/articles/s41598-018-21444-4.pdf (2018).

146. Diggins, K. E., Ferrell, P. B. & Irish, J. M. Methods for discovery and characterization of cell subsets in high dimensional mass cytometry data. *Methods (San Diego, Calif.)* **82,** 55–63 (2015).

147. Courtot, M. *et al.* flowCL: ontology-based cell population labelling in flow cytometry. *Bioinformatics (Oxford, England)* **31,** 1337–1339. https://academic.oup.com/bioinformatics/article/31/8/1337/212753 (2015).

148. Herrera, F. *et al. Multiple Instance Learning: Foundations and Algorithms* ISBN: 9783319477589 (Springer-Verlag, s.l., 2016).

149. Rolf Tschernig. Methoden der Ökonometrie - Handout. *Universität Regensburg* (2018).

150. M. Augustine Cauchy. Methode generale pour la resolution des systemes d'equations simultanees. *Comptes Rendus de l'Academie des Science* **25,** 536–538. https://ci.nii.ac.jp/naid/10026173737/ (1847).

151. White, B. W. & Rosenblatt, F. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. *The American Journal of Psychology* **76,** 705. ISSN: 00029556. http://www.jstor.org/stable/1419730 (1963).

152. Cox, D. R. The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society. Series B (Methodological)* **20,** 215–242. ISSN: 0035-9246. http://www.jstor.org/stable/2983890 (1958).

153. Ian Goodfellow, Yoshua Bengio & Aaron Courville. Deep Learning. *MIT Press.* http://www.deeplearningbook.org (2016).

154. Skansi, S. *Introduction to deep learning: From Logical Calculus to Artificial Intelligence* ISBN: 978-3-319-73004-2. http://dspace.agu.edu.vn/handle/AGU_Library/13575 (Springer, Cham, 2018).

155. Vinod Nair & Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf.

156. LeCun, Y. *et al.* Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* **1,** 541–551. ISSN: 0899-7667 (1989).

157. Paszke, A. *et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library* 2019. https://arxiv.org/pdf/1912.01703.

158. Sergey Ioffe & Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *International Conference on Machine Learning,* 448–456. ISSN: 1938-7228. http://proceedings.mlr.press/v37/ioffe15.html (2015).

159. Falcon, W. A. PyTorch Lightning. *GitHub.* https://github.com/PyTorchLightning/pytorch-lightning (2019).

160. Omry Yadan. *Hydra - A framework for elegantly configuring complex applications* 2019. https://github.com/facebookresearch/hydra.

161. Soilán, M., Lindenbergh, R., Riveiro, B. & Sánchez-Rodríguez, A. POINTNET FOR THE AUTOMATIC CLASSIFICATION OF AERIAL POINT CLOUDS. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* **IV-2/W5,** 445–452 (2019).

162. Uy, M. A., Pham, Q.-H., Hua, B.-S., Nguyen, D. T. & Yeung, S.-K. *Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data* 2019. https://arxiv.org/pdf/1908.04616.

163. Qi, C. R., Liu, W., Wu, C., Su, H. & Guibas, L. J. *Frustum PointNets for 3D Object Detection from RGB-D Data* 2017. https://arxiv.org/pdf/1711.08488.

164. Zhang, Z., Hua, B.-S. & Yeung, S.-K. *ShellNet: Efficient Point Cloud Convolutional Neural Networks Using Concentric Shells Statistics* in (2019), 1607–1616. http://openaccess.thecvf.com/content_ICCV_2019/papers/Zhang_ShellNet_Efficient_Point_Cloud_Convolutional_Neural_Networks_Using_Concentric_Shells_ICCV_2019_paper.pdf.

165. Zhang, M., You, H., Kadam, P., Liu, S. & Kuo, C.-C. J. *PointHop: An Explainable Machine Learning Method for Point Cloud Classification* 2019. https://arxiv.org/pdf/1907.12766.

166. Wang, Y. *et al.* Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics (TOG)* **38,** 146. ISSN: 0730-0301. http://dl.acm.org/ft_gateway.cfm?id=3326362&type=pdf (2019).

167. Qi, C. R., Su, H., Mo, K. & Guibas, L. J. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation* 2016. https://arxiv.org/pdf/1612.00593.

168. Li, Y. *et al. PointCNN: Convolution On X-Transformed Points* in (2018), 820–830. http://papers.nips.cc/paper/7362-pointcnn-convolution-on-x-transformed-points.pdf.

169. Liu, M., Sheng, L., Yang, S., Shao, J. & Hu, S.-M. *Morphing and Sampling Network for Dense Point Cloud Completion* 2019. https://arxiv.org/pdf/1912.00280.

170. Bello, S. A., Yu, S. & Wang, C. *Review: deep learning on 3D point clouds* 2020. https://arxiv.org/pdf/2001.06280.

171. Liu, W., Sun, J., Li, W., Hu, T. & Wang, P. Deep Learning on Point Clouds and Its Application: A Survey. *Sensors (Basel, Switzerland)* **19** (2019).

172. Guo, Y. *et al.* Deep Learning for 3D Point Clouds: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PP,** 1. ISSN: 0162-8828 (2020).

173. Faraggi, D. & Simon, R. A neural network model for survival data. *Statistics in medicine* **14,** 73–82. ISSN: 0277-6715 (1995).

174. Yu, C.-N., Greiner, R., Lin, H.-C. & Baracos, V. Learning Patient-Specific Cancer Survival Distributions as a Sequence of Dependent Regressors. *Advances in Neural Information Processing Systems* **24** (2011).

175. Fornili, M., Ambrogi, F., Boracchi, P. & Biganzoli, E. *Piecewise Exponential Artificial Neural Networks (PEANN) for Modeling Hazard Function with Right Censored Data* in *Computational intelligence methods for bioinformatics and biostatistics* (eds Formenti, E., Tagliaferri, R. & Wit, E.) (Springer, Berlin, 2014), 125–136. ISBN: 978-3-319-09042-9.

176. Luck, M., Sylvain, T., Cardinal, H., Lodi, A. & Bengio, Y. *Deep Learning for Patient-Specific Kidney Graft Survival Analysis* 2017. https://arxiv.org/pdf/1705.10245.

177. Changhee Lee, William Zame, Jinsung Yoon & Mihaela van der Schaar. DeepHit: A Deep Learning Approach to Survival Analysis With Competing Risks. *Proceedings of the AAAI Conference on Artificial Intelligence* **32.** ISSN: 2374-3468. https://ojs.aaai.org/index.php/AAAI/article/view/11842 (2018).

178. Ching, T., Zhu, X. & Garmire, L. X. Cox-nnet: An artificial neural network method for prognosis prediction of high-throughput omics data. *PLoS computational biology* **14,** e1006076 (2018).

179. Katzman, J. L. *et al.* DeepSurv: personalized treatment recommender system using a Cox proportional hazards deep neural network. *BMC medical research methodology* **18,** 24 (2018).

180. Fotso, S. *Deep Neural Networks for Survival Analysis Based on a Multi-Task Framework* 2018. https://arxiv.org/pdf/1801.05512.

181. Gensheimer, M. F. & Narasimhan, B. A scalable discrete-time survival model for neural networks. *PeerJ* **7,** e6257. ISSN: 2167-8359 (2019).

182. Håvard Kvamme and Ørnulf Borgan and Ida Scheel. Time-to-Event Prediction with Neural Networks and Cox Regression. *Journal of Machine Learning Research* **20,** 1–30. ISSN: 1533-7928. https://jmlr.org/papers/v20/18-424.html (2019).

183. Kather, J. N. *et al.* Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study. *PLoS medicine* **16,** e1002730 (2019).

184. Kvamme, H. & Borgan, Ø. *Continuous and Discrete-Time Survival Prediction with Neural Networks* 2019. https://arxiv.org/pdf/1910.06724.

185. Luecken, M. D. & Theis, F. J. Current best practices in single–cell RNA–seq analysis: a tutorial. *Molecular Systems Biology* **15,** 360. ISSN: 1744-4292 (2019).

186. Beyer, K., Goldstein, J., Ramakrishnan, R. & Shaft, U. *When Is "Nearest Neighbor" Meaningful?* in *Database Theory - ICDT'99* (eds Beeri, C. & Buneman, P.) (Springer, Berlin and Heidelberg, 1999), 217–235. ISBN: 978-3-540-49257-3.

187. Bhattacharya, S. *et al.* ImmPort, toward repurposing of open access immunological assay data for translational and clinical research. *Scientific data* **5,** 180015 (2018).

188. Kronstad, L. M., Seiler, C., Vergara, R., Holmes, S. P. & Blish, C. A. Differential Induction of IFN-a and Modulation of CD112 and CD54 Expression Govern the Magnitude of NK Cell IFN-g Response to Influenza A Viruses. *The Journal of Immunology* **201,** 2117–2131. ISSN: 1550-6606. https://www.jimmunol.org/content/201/7/2117 (2018).

189. Miron, M. *et al.* Human Lymph Nodes Maintain TCF-1hi Memory T Cells with High Functional Potential and Clonal Diversity throughout Life. *The Journal of Immunology* **201,** 2132–2140. ISSN: 1550-6606 (2018).

190. Alpert, A. *et al.* A clinically meaningful metric of immune age derived from high-dimensional longitudinal monitoring. *Nature Medicine* **25,** 487–495. ISSN: 1546-170X. https://www.nature.com/articles/s41591-019-0381-y (2019).

191. Hanin, B. & Rolnick, D. *How to Start Training: The Effect of Initialization and Architecture* 2018. https://arxiv.org/pdf/1803.01719.

192. Jindal, A., Gupta, P., Jayadeva & Sengupta, D. Discovery of rare cells from voluminous single cell expression data. *Nature Communications* **9,** 4719. ISSN: 2041-1723. https://www.nature.com/articles/s41467-018-07234-6 (2018).

193. Wegmann, R. *et al.* CellSIUS provides sensitive and specific detection of rare cell populations from complex single-cell RNA-seq data. *Genome Biology* **20,** 142. ISSN: 1474-760X. https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1739-7 (2019).

194. Jiang, L., Chen, H., Pinello, L. & Yuan, G.-C. GiniClust: detecting rare cell types from single-cell gene expression data with Gini index. *Genome Biology* **17,** 144. ISSN: 1474-760X. https://link.springer.com/article/10.1186/s13059-016-1010-4 (2016).

195. Dong, R. & Yuan, G.-C. GiniClust3: a fast and memory-efficient tool for rare cell type identification. *BMC Bioinformatics* **21,** 158. ISSN: 1471-2105. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-020-3482-1 (2020).

196. Hedley, B. D. & Keeney, M. Technical issues: flow cytometry and rare event analysis. *International journal of laboratory hematology* **35,** 344–350. https://onlinelibrary.wiley.com/doi/full/10.1111/ijlh.12068 (2013).

197. Ting, D. T. *et al.* Single-cell RNA sequencing identifies extracellular matrix gene expression by pancreatic circulating tumor cells. *Cell Reports* **8,** 1905–1918. ISSN: 2211-1247 (2014).

198. Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. *Optuna: A Next-generation Hyperparameter Optimization Framework* 2019. https://arxiv.org/pdf/1907.10902.

199. Rapin, J. & Teytaud, O. *Nevergrad - A gradient-free optimization platform* https://GitHub.com/FacebookResearch/Nevergrad. 2018.

200. Kevin Jamieson & Ameet Talwalkar. Non-stochastic Best Arm Identification and Hyperparameter Optimization. *Artificial Intelligence and Statistics,* 240–248. ISSN: 1938-7228. http://proceedings.mlr.press/v51/jamieson16.html (2016).

201. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research 18.* https://arxiv.org/pdf/1603.06560 (2018).

202. Shapley, L. S. H. W. K. & Tucker, A. W. Contributions to the Theory of Games. *Annals of Mathematics studies* (1953).

203. Algaba, E., Fragnelli, V. & Sánchez-Soriano, J. *Handbook of the Shapley Value* ISBN: 9781351241410 (Chapman and Hall/CRC, 2019).

204. Molnar, C. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable* https://christophm.github.io/interpretable-ml-book/ (2019).

205. Štrumbelj, E. & Kononenko, I. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems* **41,** 647–665. ISSN: 0219-3116 (2014).

206. Erik Å trumbelj & Igor Kononenko. An Efficient Explanation of Individual Classifications using Game Theory. *Journal of Machine Learning Research* **11,** 1–18. ISSN: 1533-7928. https://jmlr.org/papers/v11/strumbelj10a.html (2010).

207. Bach, S. *et al.* On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLOS ONE* **10,** e0130140. ISSN: 1932-6203. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140 (2015).

208. A. Datta, S. Sen & Y. Zick. *Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems* in *2016 IEEE Symposium on Security and Privacy (SP)* (2016), 598–617.

209. Lipovetsky, S. & Conklin, M. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry* **17,** 319–330. ISSN: 1524-1904 (2001).

210. Ribeiro, M. T., Singh, S. & Guestrin, C. *"Why Should I Trust You?"* in *KDD2016* (eds Krishnapuram, B. *et al.*) (Association for Computing Machinery Inc. (ACM), New York, NY, 2016), 1135–1144. ISBN: 9781450342322.

211. Shrikumar, A., Greenside, P. & Kundaje, A. Learning Important Features Through Propagating Activation Differences. *PMLR 70:3145-3153.* https://arxiv.org/pdf/1704.02685 (2017).

212. Lundberg, S. & Lee, S.-I. *A Unified Approach to Interpreting Model Predictions* 2017. https://arxiv.org/pdf/1705.07874.

213. Shrikumar, A., Greenside, P., Shcherbina, A. & Kundaje, A. *Not Just a Black Box: Learning Important Features Through Propagating Activation Differences* 2016. https://arxiv.org/pdf/1605.01713.

214. Chen, H., Lundberg, S. & Lee, S.-I. *Explaining Models by Propagating Shapley Values of Local Components* 2019. https://arxiv.org/pdf/1911.11888.

215. Feng, M., Zhang, L., Lin, X., Gilani, S. Z. & Mian, A. *Point Attention Network for Semantic Segmentation of 3D Point Clouds* 2019. https://arxiv.org/pdf/1909.12663.

216. Ilse, M., Tomczak, J. M. & Welling, M. *Attention-based Deep Multiple Instance Learning* 2018. https://arxiv.org/pdf/1802.04712.

217. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2014. https://arxiv.org/pdf/1412.6980.

218. Kronenberg, K., Riquelme, P. & Hutchinson, J. A. Standard protocols for immune profiling of peripheral blood leucocyte subsets by flow cytometry using DuraClone IM reagents (2021).

219. Murphy, R. F. & Chused, T. M. A proposal for a flow cytometric data file standard. *Cytometry* **5,** 553–555. ISSN: 0196-4763 (1984).

220. Data file standard for flow cytometry. Data File Standards Committee of the Society for Analytical Cytology. *Cytometry* **11,** 323–332. ISSN: 0196-4763 (1990).

221. Dean, P. N., Bagwell, C. B., Lindmo, T., Murphy, R. F. & Salzman, G. C. Introduction to flow cytometry data file standard. *Cytometry* **11,** 321–322. ISSN: 0196-4763 (1990).

222. Seamer, L. C. *et al.* Proposed new data file standard for flow cytometry, version FCS 3.0. *Cytometry* **28,** 118–122. ISSN: 0196-4763 (1997).

223. Spidlen, J. *et al.* Data File Standard for Flow Cytometry, version FCS 3.1. *Cytometry Part A* **77,** 97–100. ISSN: 1552-4930 (2010).

224. R Core Team. *R: A Language and Environment for Statistical Computing* R Foundation for Statistical Computing (Vienna, Austria, 2021). https://www.R-project.org/.

225. Hahne, F., Huber, W., Ruschhaupt, M., Toedling, J. & Barry, J. *prada: Data analysis for cell-based functional assays* R package version 1.63.0 (2019).

226. Ellis, B. *et al. flowCore: Basic structures for flow cytometry data* R package version 2.0.1 (2020).

227. Mike Jiang Greg Finak, N. G. *ncdfFlow: A package that provides HDF5 based storage for flow cytometry data.* R package version 2.34.0 (2020).

228. Van Rossum, G. & Drake, F. L. *Python 3 Reference Manual* ISBN: 1441412697 (CreateSpace, Scotts Valley, CA, 2009).

229. Yurtsev, E. *fcsparser* https://github.com/eyurtsev/fcsparser. 2015. (MIT License).

230. Yurtsev, E. *FlowCytometryTools* https://github.com/eyurtsev/FlowCytometryTools. 2013. (MIT License).

231. White, S. *FlowKit* https://github.com/whitews/FlowKit. 2018. (BSD-3-Clause License).

232. White, S. *FlowIO* https://github.com/whitews/FlowIO. 2013. (BSD-3-Clause License).

233. Dresden, Z. *fcswrite* https://github.com/ZELLMECHANIK-DRESDEN/fcswrite. 2016. (BSD-3-Clause License).

234. Kundert, K. *FCMcmp* https://github.com/kalekundert/fcmcmp. 2016. (MIT License).

235. Jeremy Jacobs. *xfcs: Extract metadata and data from FCS (3.0, 3.1) files* 2017. https://github.com/j4c0bs/xfcs.

236. Anderson, E. Beckman Coulter Life Sciences. *Question regarding FCS2 vs FCS3. E-mail message to author.* (2019).

237. Qian, Y. *et al.* FCSTrans: an open source software system for FCS file conversion and data transformation. *Cytometry. Part A : the journal of the International Society for Analytical Cytology* **81,** 353–356 (2012).

238. J. Spidlen., N. G. *flowUtils* 2017.

239. Moore, W. A. & Parks, D. R. Update for the logicle data scale including operational code implementations. *Cytometry Part A* **81,** 273–277. ISSN: 1552-4930 (2012).

240. Lo, K., Brinkman, R. R. & Gottardo, R. Automated gating of flow cytometry data via robust model-based clustering. *Cytometry Part A* **73,** 321–332. ISSN: 1552-4930 (2008).

Appendices

A. Flow cytometry data: FCS files and

A flow cytometry experiment results in n events with p parameters. The data of these experiments are usually stored as LMD files, where FCS became a de-facto standard how to organize these list-mode files.

Apart from the actual event and parameter values, additional information is often necessary to properly describe a flow cytometry experiment. Here, MIFlowCyt has been introduced to describe a flow cytometry experiment in a standardised way: It includes an overview of the experiment, sample, instrument and data analysis details. Except for the data analysis, all are quite straightforward as you would introduce an experiment to someone new to your task. The data analysis details include the actual data (or how to get it). The data are usually supplied as LMD files, e.g. FCS files. Additionally, compensation (see subsubsection 1.4.1.1) description must be given. Next, every data transformation must be given. Finally, if applicable, all gating (see subsubsection 1.4.2.1) details are necessary.

FCS is a file format to standardise reading and writing LMD files. As technology improved, multiple updates of that format were necessary: FCS 1.0[219], FCS 2.0[220,221], FCS 3.0[222] and FCS 3.1[223]. To read those data files, there exist packages in R[224]: prada (deprecated)[225], flowCore[226] and ncdfFlow (HDF5-based, extends flowCore[226], use for huge datasets)[227]. For python[228] there exists a variety of packages[85,229–235] where FlowKit's FlowIO or fcsparser package seem most used. Take note that sometimes one `.LMD`- or `.fcs`-file contains multiple datasets: For example multiple samples (deprecated[223]) or once in FCS2.0 and once in FCS3.x format (still allowed).

FCS3.1 files essentially consist of:

1. HEADER segment: Describes the location of the other segments.
2. TEXT segment: Contains keyword-value pairs describing the data set, e.g. feature names, number of events, type of the following DATA, spillover matrix (see subsubsection 1.4.1.1), transformations, etc.
3. DATA segment: Contains the raw data.
4. (Optional) ANALYSIS segment: Contains the result of data processing, e.g. cell subsetting, cell cycle analysis, etc.
5. (Optional) CRC Value: Redundancy check-value for the data.
6. (Optional) OTHER segments: Anything.

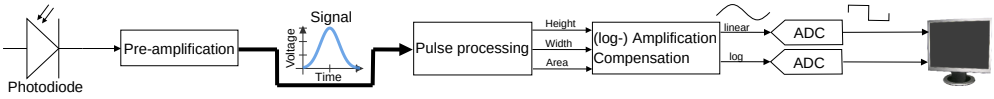
Even if it strongly advised against using the FCS2.0 format in your analysis, it is sometimes still used (or the only thing available) and especially present in the

.LMD-files we received from collaborators. This old standard[220,221] seems to lack only the CRC value compared to FCS3.1. But actually, FCS3.x and FCS2.0 must be interpreted completely different. The main difference resides in the extraction of DATA using TEXT information. Flow cytometry data must usually be compensated (see subsubsection 1.4.1.1) and transformed (see subsubsection 1.4.1.2) before analysis. Ernie Anderson[236] was of great help to me to differ between FCS2.0 and FCS3.x files: In FCS2.0-files, the data is already transformed and compensated. The respective TEXT transformation and compensation information reflect what **has been done**. That's in contrast to FCS3.x files where the TEXT information explains what **should be done**.

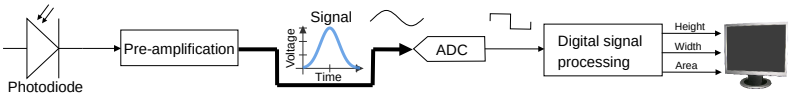
I add a bit of background and terminology[236]. The data format difference results from (old) analogue and (nowadays) digital compensation and transformation. See also Figure A 1 which condenses information from Shapiro [68].

At a high level, measuring a cell works the same both times: Use a light source, measure stray light by electrical means and convert these electrical signals to digital values recognisable by a PC. The older analogue design used logarithmic amplifiers and performed compensation manipulating the voltage of the signals before digitising it with the ADC. This was advantageous as low-resolution ADC can be used: They are rated by the resolution (number of output lines, between 6 and 24 "bits") and the sampling rate (up to many MHz). The ADCs that time were expensive, and only low-resolution, so many circuits were done analogue. Nowadays, they are cheap and have a high resolution and sampling rate. One downside of the analogue world is that compensation and transformation are fixed as the resolution is not high enough to change the values.

Regarding terminology: Each signal-processed measurement like height, width or area of forward scatter (FS), side scatter(SS), or fluorescence (FL.n) is called a parameter in flow cytometry. The ADC's resolution refers to the maximum possible binary number generated by the given number of bits: 2^{bits}. It was generally unnecessary for analogue cytometers to have a resolution higher than $2^{10} = 1024$, thus values from 0 to 1023. Newer systems have 24 or more bit-ADCs. Each of the possible 2^{bits} values is termed a channel. Sometimes, bin is used interchangeably. This confusion resulted from a common visualisation of flow cytometry data in histograms: Those histograms bin the 2^{bits} values, but in early days the number of bins was the same as the number of channels (e.g. both times 1024), leading to a sloppy usage of terminology.



(A) The (fluorescence) scattered light is detected by an photomultiplier tube (PMT). This signal needs to be "pre-amplified", increasing the detected electrical current and converting it to a voltage. After that, the signal is processed by analog circuits resulting in height, width and area of the pulse. These three signals are linearly and logarithmically amplified (read "transformed") and compensated (see subsubsection 1.4.1.1). The ADC converts the analog to a computer-readable digital signal.[46]



(B) The (fluorescence) scattered light is detected by a photomultiplier tube (PMT). This signal needs to be "pre-amplified", increasing the detected electrical current and converting it to a voltage. Then directly, the ADC converts the analog into a digital signal. specialised digital signal processors convert the signal into height, width and area. Note that the data are neither compensated nor transformed, this is left to the analyst.[46]

Figure A 1.: Flow cytometer electronics, analog vs digital

B. Flow cytometry data transformation

In flow cytometry, the values of parameters are approximately normally distributed on a logarithmic scale[68]. However, compensation leads to negative values where the logarithm is undetermined.

Therefore, multiple transformations have been proposed to properly include negative values: Logicle (sometimes called biexponential)[69], HyperLog[70], box-cox[71], linlog, generalised arcsinh ($= \sinh^{-1}(x)$)[72] and flowVS which can automatically estimate suitable parameters for the arcsinh[73]. More recently, rank transformation was proposed to mitigate batch effects, but they strongly recommend combining it with usual preprocessing steps - in particular compensation and debris removal[6].

I show the formula for different transformations and stick mostly to the notation from flowCore[226]. Comparisons of the transformations have been published[72,74,237]. Except for linlog (flowUtils[238]), rank transformation and flowVS, all transformations are implemented in flowCore[226], see also the vignettes in Bioconductor for further information. If you work with FlowJo, you might want to look at the Bioconductor package flowWorkspace.

Proper data transformation is not straightforward, and its choice and parametrisation remain data-dependent[72,74]. The examples in Figure B 1 use mainly the default settings and should be adjusted manually to your data at hand.

log(arithm)

$$f(x) = \begin{cases} log(x)\frac{r}{d} & \text{if } x > 0 \\ 0 & else \end{cases}$$

arcsinh[226]

$$f(x) = \sinh^{-1}(a + bx) + c$$

linlog[72]

$$f(x) = \begin{cases} \frac{x-a}{a} + \log(a) & \text{if } x \leq a \\ \log(x) & y > a \end{cases}$$

biexponential[226]

$f^{-1}(x) = ae^{bx} - ce^{dx} + h$

logicle[69,239]

$$f^{-1}(x) = \begin{cases} T10^{-(M-W-A)}\left(10^{x-W-A} - p^2 10^{-(x-W-A)/p} + p^2 - 1\right) & \text{if } x \geq W + A \\ -T10^{-(M-W-A)}\left(10^{W+A-x} - p^2 10^{-(W+A-x)/p} + p^2 - 1\right) & \text{if } x < W + A \end{cases}$$

The parameters are chosen from the data: T is the maximum data value, M is the range of the data in decades, W is the number of decades in the approximately linear region, A is the number of decades of negative values to be included.

generalised box-cox[71,240]

$f(x) = \frac{sgn(x)|x|^a - 1}{a}, a > 0$

HyperLog[70]

$$f(x) = root\ (EH(y) - x) \text{ where } \begin{cases} EH(y) = 10^{(\frac{y}{a})} + \frac{b*y}{a} - 1 & \text{if } y >= 0 \\ EH(y) = -10^{(\frac{-y}{a})} + \frac{b*y}{a} + 1 & else \end{cases}$$

find $root$ such that $x = EH(y)$

Rank

Inside each sample i and per measured feature j, sort all cells and replace the measurement with its percentile. $f(x_{ij}) = \frac{rank(x_{ij} \text{| in sample } i)}{\#\text{cells in sample } i} \cdot 100$

flowVS[73]

$f(x) = \sinh^{-1}(\frac{x}{c})$ where the "cofactor" c is estimated per feature by using Bartlett's likelihood-ratio test.

where $f(x)$ are compensated, transformed data and a, b, c, d, e, h, r, d are additional parameters. For the biexponential and logicle transformations only the inverse f^{-1} has a closed form, so $f(x)$ must be solved numerically. The sinh and thus also arcsinh can be generalised to the biexponential function[69]. The logicle scale is the inverse of a modified biexponential function with the "logicle condition" that the second derivative $\frac{\partial^2 f(x)}{\partial x^2} = 0$ at the (chosen) data zero position ($y = W + A$). With the additional restriction to a scale range of $[0, 1]$, a, b, c, d, h of the biexponential function can be fixed.

Figure 1.6 shows exemplary parametrisations of the different transformations on a sample stained with CD4 and CD8. Note that I show the transformed values on the x- and y-axis. In cytometry, sometimes the *untransformed* values are shown, but on the respective transformation scale.

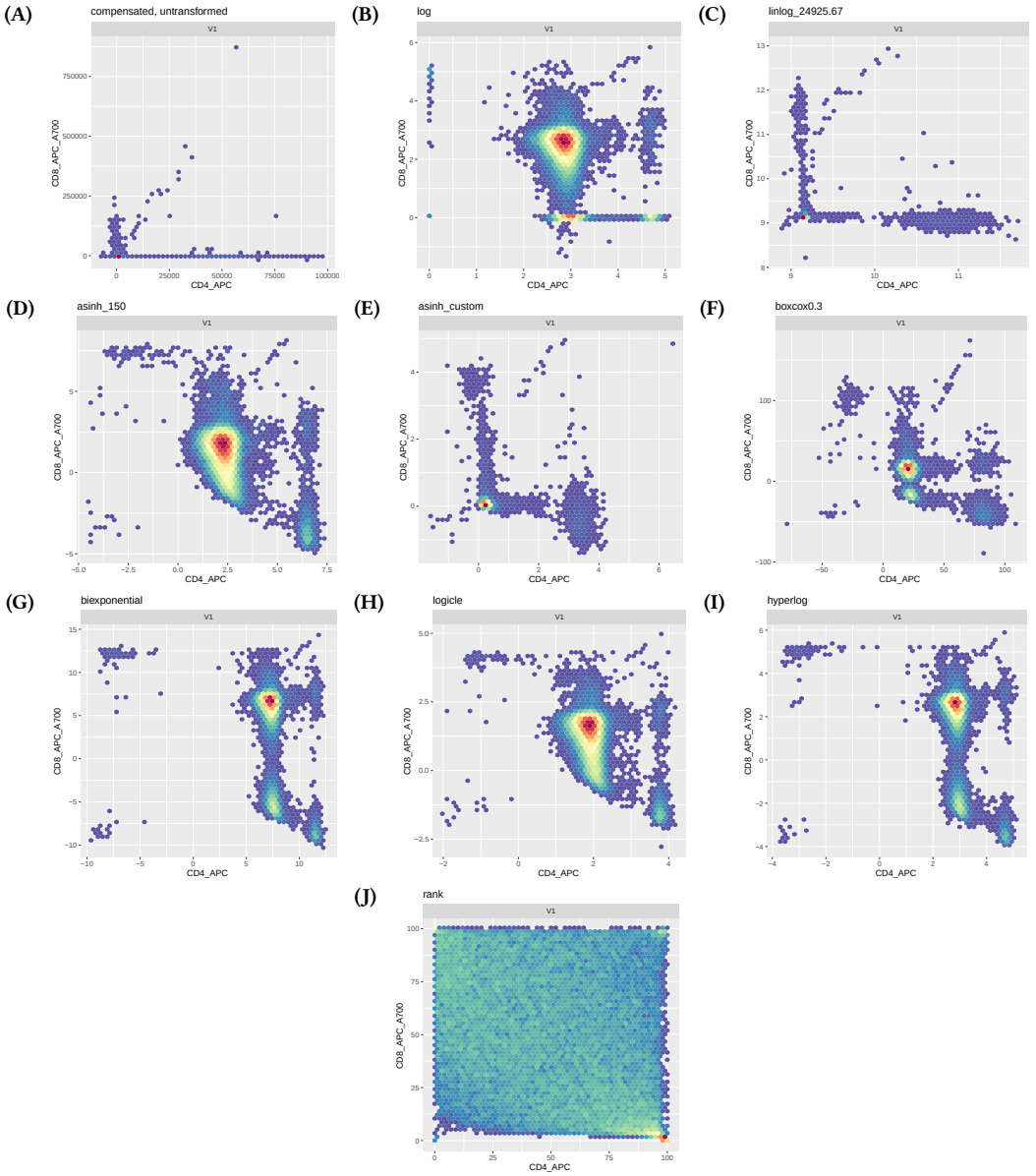


Figure B 1.: Cytometry transformations. Every plot shows CD4 on the x-axis and CD8 on the y-axis. Plot (A) shows the untransformed, compensated measurements as references; all other plots show the transformed values. The colour denotes the number of cells at that position where blue is at least one cell. In each plot are 50000 cells. I traverse the plot row by row. (A) Raw, compensated data (B) log, negative values set to 0 (C) linlog, a=24925.67, optimal estimation from flowUtils[238] (D) arcsinh, both CD4 and CD8 are transformed with $\sinh^{-1}$, $a = 0, b = \frac{1}{150}, c = 0$ (E) arcsinh, CD4 and CD8 are transformed with $a = 0, c = 0$ but $b_{CD4} = \frac{1}{3500}$ and $b_{CD8} = \frac{1}{5000}$ according to expert knowledge from the laboratory (F) box-cox, $a = 0.3$ (G) biexponential, $a = 0.5, b = 1, c = 0.5, d = 1, h = 0, w = 0$ (H) logicle, $W = 0.5, T = 262144, M = 4.5, A = 0$ (I) HyperLog, $a = 1, b = 1$ (J) rank transformation.

C. Performance experiment 1: All hyperparameter's effects on performance

AUCs for all tested hyperparameters in experiment 1. Fixed and unfixed, drawing cells randomly and by outlierness, drawing 2500, 5000, or 9500 cells, and mean or max pooling. The architecture was fixed to a grid-search based optimum as reported previously[3]. There are 2 *FeatureLearner* layers with 3 nodes each and 1 *Predictor* layer with 3 nodes. Each layer is followed by batch normalisation[158] and a ReLU-function[155]. After the 3-node *Predictor* layer, the final layer consisting of a single node reflects the binary problem. This final layer has neither batch normalisation nor ReLU-function but a sigmoid-function.

Batch-normalisation[158] was used after each *FeatureLearner* and *Predictor* layer, except the last *Predictor* layer (after the 3-nodes)

The shown boxplots do not include all runs of all tested variants. Instead, each boxplot includes only the best run for each variant.

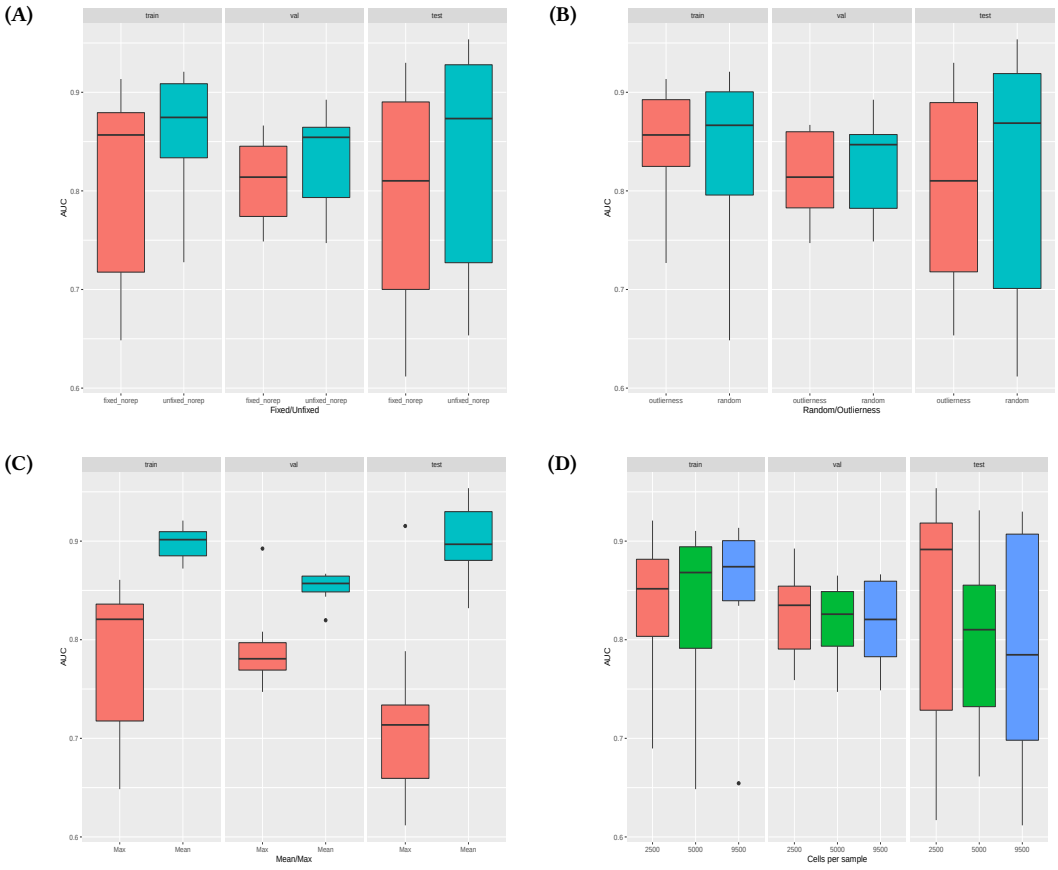


Figure C 1.

D. Performance experiment 2: All parameter and network architecture effects on performance

Performance differences for all tested hyperparameters in experiment 2. For this, I fixed drawing cells random and used only mean pooling. Regarding subsampling, I used fixed or unfixed without replacement (A) and drew 2500, 5000, or 9500 cells (C). The architecture was set to 1,2,3 or 10 layers (D) with each 3 or 10 nodes (E) in the *FeatureLearner* and 1,2,3 or 10 layers (B) with each 3 or 10 nodes (F) in the *Predictor*.

The shown boxplots do not include all runs of all tested variants. Instead, each boxplot includes only the best run for each variant.

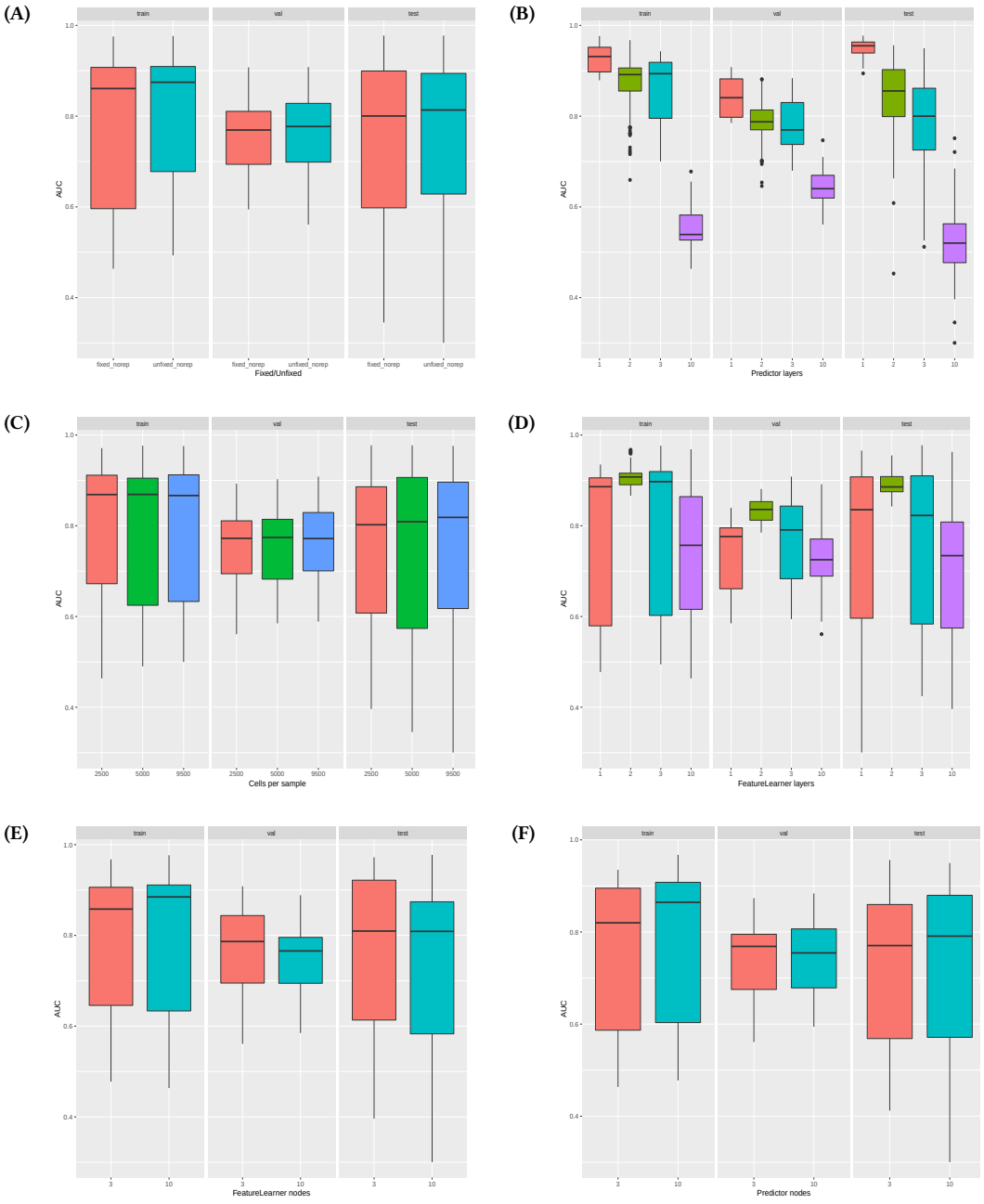


Figure D 1.

E. Performance experiment 2: All parameter and network architecture effects on the AUC-range

The unfixed setting draws a certain number of cells ("cells per sample") from all cells in that sample each time this sample is inspected. An "inspection" happens during training but also during inference, so the prediction of a sample can change. Here I show the results of how strong this prediction variability changes the final AUC, stratified for the parameters. The boxplots include 50 AUCs due to predicting all samples 50 times. For each prediction of all samples, I calculated the AUC, which then formed the boxplots in Figure 3.6. From these predictions, shown as boxplots, I calculated the AUC range minimum to maximum. This range is shown on the y-axis, each boxplot contains all parameter variantruns (except for the stratified).

For this, I fixed drawing cells random and used only mean pooling. Regarding subsampling, I show only unfixed without replacement as fixed would mask the multiple-prediction effect. I show 2500, 5000, or 9500 cells (E). The architecture was set to 1,2,3 or 10 layers (A) with each 3 or 10 nodes (C) in the *FeatureLearner* and 1,2,3 or 10 layers (B) with each 3 or 10 nodes (D) in the *Predictor*.

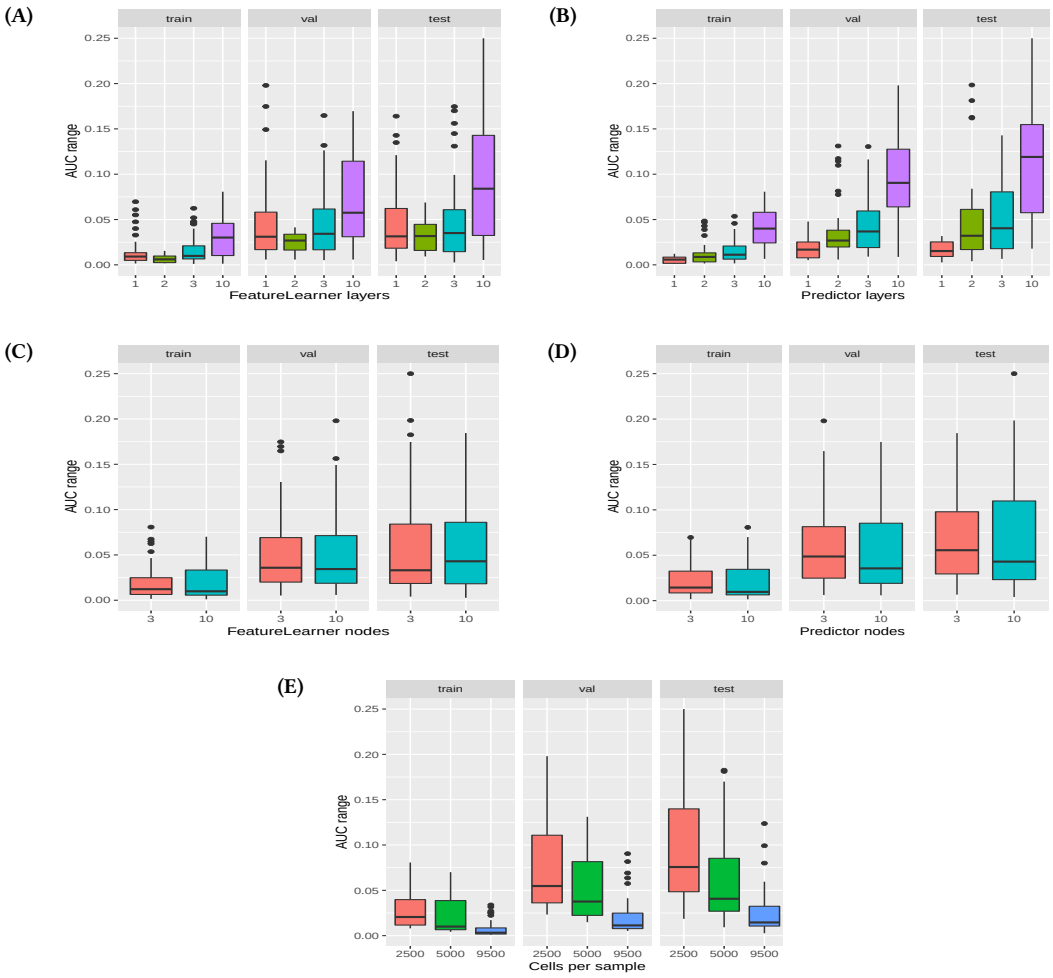


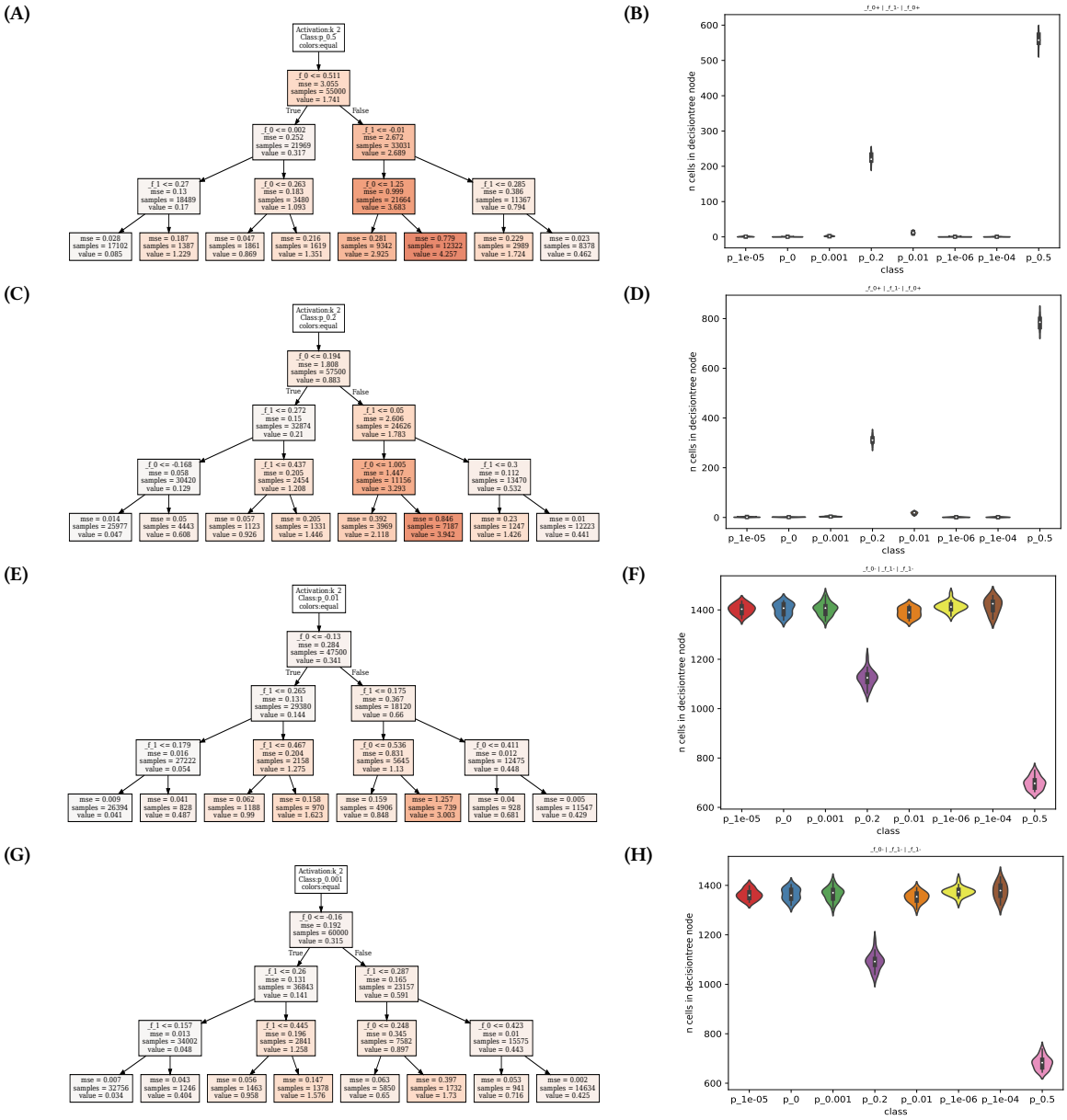
Figure E 1.

F. Multiclass classification simulation: All decision trees and gatings

In the multiclass classification setting, it is possible that the decision trees built on cells from each distinct class of samples are completely different for every class.

Therefore, I here show them for all eight classes in this multiclass classification problem. The plot is continued on the next page. Interestingly, in this comparably easy simulation, there are essentially only two types of decision trees and associated gatings:

- $f_0^- \mid f_1^- \mid f_1^-$ for *zero-like*, $p_{0.001}$ and $p_{0.01}$ classes
- $f_0^+ \mid f_1^- \mid f_0^+$ for $p_{0.2}$ and $p_{0.5}$ classes



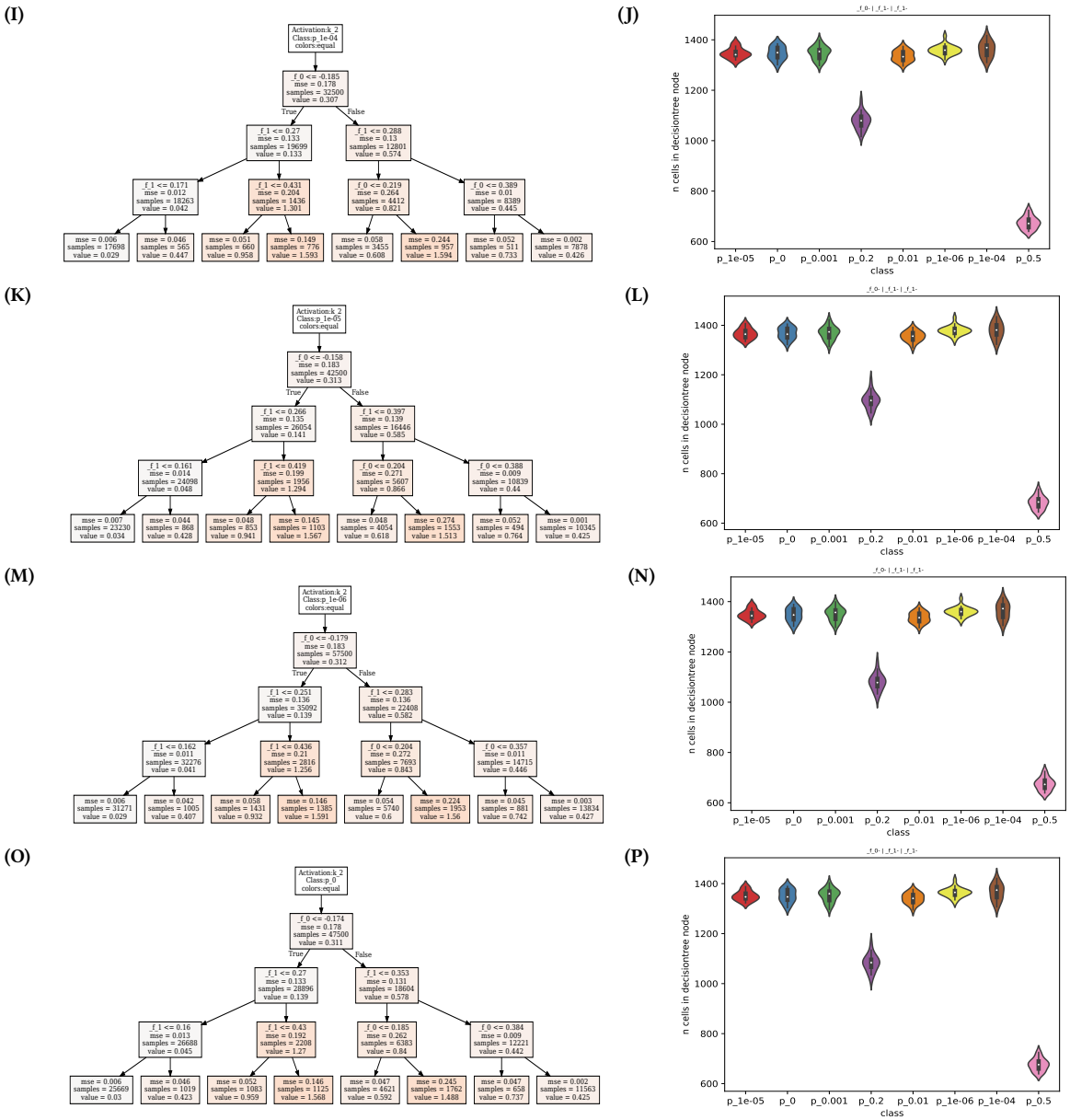


Figure F 1.: Multiclass simulation: Gatings and trees for all classes. Here we see the decision tree built to replace the *FeatureLearner* for its cell-feature k_2, distinctly learned on cells from simulated biological samples of one class. On the left side, always the decision tree is shown. On the right side, the respective node counts from the resulting gating of the test samples. For the gating directions see Table 3.2. We see that there are essentially two types of gates: First, $p_{0.5}$ and $p_{0.2}$ where high counts speak for classes with higher $\mathcal{N}_1$ percentage. Second, all others where high node counts speak for classes with lower $\mathcal{N}_1$ percentage.

G. Fasting data: Full gating graph

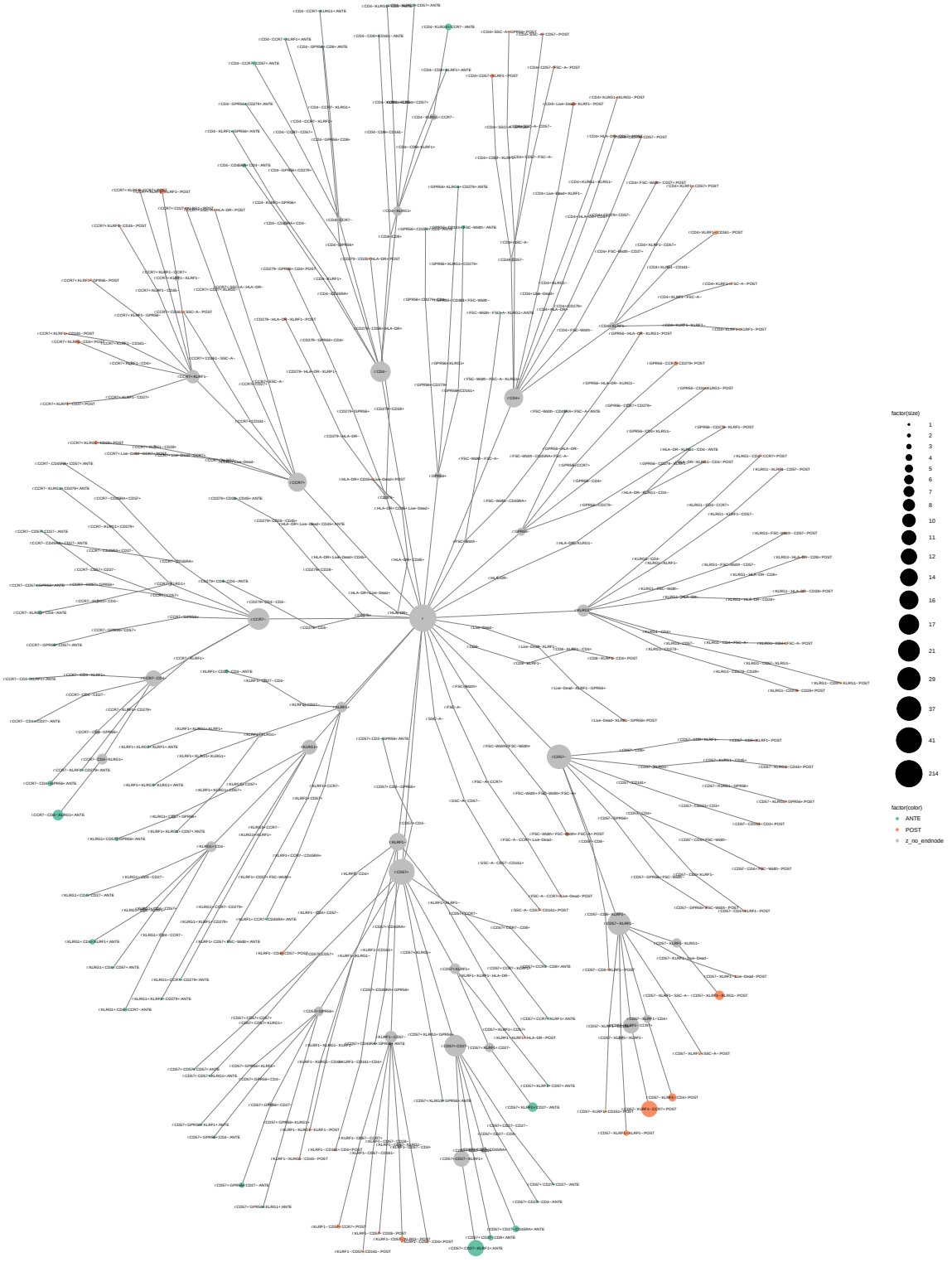


Figure G 1.: Fasting: Repeated gatings full graph. This graph shows the gatings from 50 initialisations and over all 3 pooled cell-features. Every gating starts from the root in the center r. Each gating can be read by traversing the graph going to the outside and finally reaching a green or orange node which reflects the final node for an ANTE or POST gating, respectively.