Universiteit Antwerpen

Predicting Amino Acid Sequence Interactions Based on Pairwise Comparisons of Physicochemical Properties

Joey De Pauw

Principal Adviser:	Prof. Dr. Serge Demeyer
Principal Co-Adviser:	Prof. Dr. Kris Laukens
Assistant Advisers:	Dr. Pieter Meysman M. Sc. Pieter Moris

Dissertation Submitted in June 2019 to the
Department of Mathematics and Computer Science
of the Faculty of Sciences, University of Antwerp,
in Partial Fulfillment of the Requirements
for the Degree of Master of Science.

Contents

List of Figures

List of Tables

List of Acronyms

AUC area under the curve

CDR3 complementarity-determining region 3

CSV comma-separated values

CTL cytotoxic T-cell

EBI European Bioinformatics Institute

FPR false positive rate

GAP global average pooling

HLA human leukocyte antigen

MHC major histocompatibility complex

PPI protein-protein interaction

PR precision-recall

TCR T-cell receptor

TPR true positive rate

Nederlandstalige Samenvatting

Een belangrijk deel van het immuunsysteem is antigen presentatie door MHC molecules en de daaropvolgende herkenning door T-cell receptors (TCRs). Hoewel er veel technieken zijn voor het voorspellen van de binding tussen MHC molecules en epitopen, blijft het voorspellen van TCR binding een open probleem. Dit kan grotendeels toegekend worden aan het voormalige gebrek aan voldoende data van degelijke kwaliteit. Dankzij next generation sequencing methodes bestaan er tegenwoordig wel verschillende datasets die de binding tussen TCR en epitoop beschrijven, zoals bijvoorbeeld de VDJdb dataset [30].

We introduceren een nieuwe techniek voor het encoderen van features die gebaseerd is op het paarsgewijs vergelijken van fysicochemische eigenschappen van aminozuren. Het idee achter deze techniek is om de binding van de aminozuren waaruit de eiwitpartners zijn opgebouwd te "simuleren"; meer specifiek de binding tussen het CDR3 deel van een T-cell receptor en het bijhorende epitoop. Daarnaast werd de techniek ook getest om interacties tussen proteïnen in het algemeen te voorspellen. Artificiële neurale netwerken werden gebruikt als classificatie methode.

In een eerste set experimenten met de VDJdb dataset bestuderen we de impact van het gebruik van verschillende eigenschappen en "operators" voor het encoderen van features. Met deze resultaten vergelijken we vervolgens vier verschillende modellen, waaronder het state-of-the-art NetTCR model [15] dat ook gebaseerd is op neurale netwerken. Ten slotte voeren we enkele experimenten uit om verschillende scenario's te vergelijken. Uit deze reeks experimenten kunnen we afleiden dat onze methode positieve resultaten haalt die vergelijkbaar zijn met de state-of-the-art. Verder merken we op dat deze experimenten

de indicatie geven dat ons model beter slaagt in het generaliseren van de relatie tussen epitoop en T-cell receptor dan het referentiemodel.

Vervolgens werden experimenten uitgevoerd voor de tweede probleemstelling van interacties tussen algemene humane proteïnen. Hieruit blijkt dat onze techniek minder aangewezen is voor dit soort data vanuit praktische overwegingen. Door de kwadratische complexiteit van het algoritme en de langere sequenties in de dataset is het een uitdaging om modellen te ontwikkelen die zowel in memory passen als voldoende snel kunnen trainen. Uit de beperkte set experimenten in deze probleemstelling, waarin we onze techniek vergelijken met het state-of-the-art model DNN-PPI [20], krijgen we dezelfde indicatie dat ons model beter scoort als het aankomt op het generaliseren van de gezochte relatie.

In conclusie stellen we dat onze methode voor het encoderen van features positieve resultaten haalt die te vergelijken zijn met de state-of-the-art voor het voorspellen van epitoop-TCR bindingen. Onze intuïtie, dat de keuze van methode voor het encoderen van features een grote rol speelt in de uiteindelijke performantie van een classifier, wordt experimenteel bevestigd. Een methode die enkel gebaseerd is op het paarsgewijs vergelijken van fysicochemische eigenschappen lijkt beter voor het leren en generaliseren van de binding tussen aminozuursequenties.

Acknowledgements

I would first like to thank Pieter Moris and Pieter Meysman for their supervision. Thanks to our weekly meetings I always had a goal to work towards and when I got overwhelmed by all the required biological background, they would happily take their time to explain it to me. In addition their continuous feedback and new ideas helped shape this thesis.

Secondly I would like to thank professor Kris Laukens and professor Serge Demeyer for supervising my thesis.

Lastly I would like to thank Seppe Wouters for always being there for me and for creating helpful and informative visualizations on request. His interest in my work kept me motivated.

The experiments presented in this work were executed on the high performance computing core facility CalcUA. If not for their excellent support and flexible policies, it would not have been possible to perform equally extensive experiments.

The positive only HI PPI dataset was provided by the Center for Cancer Systems Biology (CCSB) at the Dana-Farber Cancer Institute which is supported by the following funding organizations:

- The National Human Genome Research Institute (NHGRI) of NIH
- The Ellison Foundation, Boston, MA
- The Dana-Farber Cancer Institute Strategic Initiative

Abstract

Tools for predicting epitope recognition by T-cell receptors can prove to be a valuable asset for studies of the adaptive immune system, with applications in auto-immunity, tumour susceptibility and vaccine design [7]. State-of-the-art methods however are still limited to only predicting known epitopes [9]. Attempts to create general models are making rapid progress, but their relatively low accuracies still limits widespread use because they are not reliable enough.

We present a new feature encoding technique that relies on pairwise comparisons of physicochemical properties on the amino acid level. The idea behind this encoding is to only feed pairwise interactions as inputs to a classifier, in the hope that it will be forced to learn the actual "binding" relation this way.

Our experiments show promising results that indicate a strong relation between feature encoding and the generalization capabilities of a classifier. When applying our approach to the similar problem of predicting protein-protein interactions (PPIs), we could derive the same conclusions, increasing our confidence in the proposed technique.

CHAPTER 1

Introduction

A central part of the adaptive immune system is antigen presentation by MHC molecules and subsequent binding by T-cell receptors (TCRs). Although many, high accuracy methods are available for predicting MHC to epitope binding like NetMHCpan [14; 24], NetMHCcons [16] and NetMHC [2], the accurate prediction of TCR specificities is still an open problem in the field of bioinformatics. This can mainly be attributed to the absence of sufficient and reliable data until recently. Next generation sequencing methods allow for the collection of data describing the TCR to epitope binding and have onset the rise of publicly available, curated datasets like the VDJdb dataset [30].

We present a new feature encoding technique for peptides based on the pairwise comparison of physicochemical properties of amino acids. The idea behind this technique is to "simulate" the binding between two proteins, in this case between the CDR3 part of a TCR and the epitope that it binds. Furthermore this technique is also applied in the more general case of predicting protein-protein interactions to test its effectiveness.

This feature encoding technique was specifically designed to work with artificial neural network classifiers. Although in theory it could be used in combination with any supervised machine learning method as a classifier, we chose neural networks because they are known to work well with two-or-higher-dimensional matrices as input, which is exactly what our technique encodes the features into.

Typical classifiers rely on a vector encoding per amino acid, for example with the BLOSUM50 matrix. Our intuition is that, by restricting a classifier to only learn from combinations of amino acids, we can achieve a better generalization of the "binding" relation between them. Previous studies also suggest that TCRs sharing a common target are characterized by sharing, to some degree, a common motif [10; 6; 15]. Our technique generalizes this idea in the sense that models can be trained to recognize more complex patterns through the physicochemical properties of amino acids, since we also incorporate the knowledge that some amino acids are more similar to each other than others into the encoding.

Better predictive models for TCR specificities can help reduce the cost of TCR repertoire analysis, in turn facilitating studies of auto-immunity, tumour susceptibility and vaccine design [7].

This document is structured as follows. Chapter 2 provides the necessary background in both computer science and biology for the full understanding of our work. In Chapter 3 we discuss related work to our contribution. Then Chapter 4 gives an overview of the experimental setup and in Chapter 5 we discuss the results. Finally Chapter 6 lists threats to validity and Chapter 7 concludes.

However before continuing to the essence of this work, we would like to remark on how it came to be. Section 1.1 contains a brief, chronological explanation of the process that resulted in this thesis.

1.1 Workflow

We dedicated the first few weeks to setting up the project and getting familiar with the domain field. This included tasks like collecting data, setting up our working environment and reading up on biological background information. After this, we started a more exploratory phase, where we visualized the data, looked for appropriate tools and frameworks and also performed some preliminary experiments. Since the early stages we always held weekly meetings to follow up on our progress and plan the next steps.

Following the exploratory phase was an experimentation stage. At this time we had enough background information and also more confidence in the feasibility of our approach, that we could start the more structured experimentation process. We developed and optimized our models by trying different variations

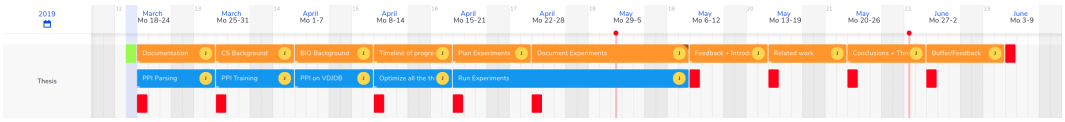


Figure 1.1: Capture of planning for last three months.

and evaluating their effect on the resulting predictive performance. Through this iterative, heuristic process we created the final models that are discussed in this thesis.

Although we always worked with weekly sprints, we decided to make them more explicit for the final three months. As can be seen in the example in Figure 1.1, we created a weekly planning, complete with milestones and subtasks. Orange bars indicate a "writing" task and blue bars were used for more "implementation" related tasks. This ensured a healthy weekly balance between the two to provide enough variation. Naturally the order and content of this planning was updated continuously and as expected, it also turned out to be on the optimistic side, which we had planned for. Even though our planning was very dynamic, it still proved to be a useful tool to keep an overview and ensure we did not take on more work than what we effectively could finish.

The final stage was mostly dedicated to running experiments and documenting them, as well as writing the other sections of the thesis. Access to the CalcUA supercomputer infrastructure helped tremendously. Both to schedule experiments and to be able to run larger models on bigger datasets. However, despite their GPU nodes with two NVIDIA Tesla GPUs with 16GB video RAM each, we still ran into some practical limitations for the second use case of prediction protein-protein interactions. In the end we were still able to perform some interesting experiments for this use case, albeit with more preliminary results.

CHAPTER 2

Background

Due to the interdisciplinary nature of this thesis, the background is split into two parts. We first cover a very minimal background in biology with essential information that is necessary for the full understanding of this work (Section 2.1). Then an introduction in computer science is given which focuses on classification with neural networks and the interpretation of results (Section 2.2).

2.1 Background in Biology

This section discusses some background in biology, more specifically in immunology, which will help provide context to the thesis.

2.1.1 Immunology

All concepts relating to immunology were grouped in Figure 2.1. The first part of this section explains some biological terms that are used in the figure. The second part discusses how an immune response works.

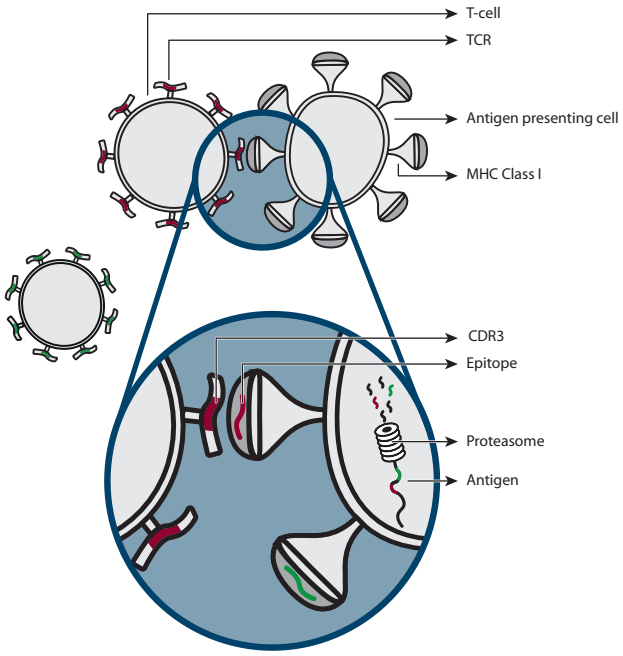


Figure 2.1: Schematic overview of immunological background.[1]

Vocabulary and Definitions

Protein
A large biomolecule consisting of one or more long chains of amino acid residues. Proteins can carry out various functions in biological systems, ranging from transport and signaling to enzymatic and immunological processes.

Peptide
A short chain of amino acids. Proteins can be cut into peptides.

Amino acid
Organic compound with an amine (-NH2) and carboxyl (-COOH) group along with a side chain which determines its type. About 500 types are known, but only 20 of them appear in the genetic code of living organisms.

Antigen
A protein for which an immune response should be triggered. The name is a contraction of **anti**body **gen**erator. Most antigens have the potential to be bound by multiple TCRs, each of which is specific to one of the antigen's epitopes.

[1]Credit: Seppe Wouters.

Epitope
Epitope refers to the part of an antigen that binds with a TCR. An antigen can contain multiple epitopes.

Major histocompatibility complex (MHC)
MHC molecules bind to epitopes and display them on the cell surface for recognition by the appropriate T-cells. There are two main types: MHC class I and MHC class II. The class determines which type of T-cell can bind to it.

T-cell receptor (TCR)
The part of a T-cell that binds to a specific epitope. Epitopes have to be presented by MHC molecules.

Cytotoxic T-cell (CTL)
This type of T-cell binds to MHC class I. It is mainly used to fend of internal threats like cancer cells and cells infected by viruses.

T helper cell
Another type of T-cell. This one is specific to MHC class II. They are mostly used to kill threats that exists outside of cells, like bacteria for example. A macrophage first engulfs and digests the threat and then presents its epitopes using MHC class II molecules.

Complementarity-determining region 3 (CDR3)
This part of the TCR is responsible for binding to an epitope. Like CDR1 and CDR2 it is a highly variable region in the molecule. It is what gives the TCR its specificity.

Antigen Presentation

The first step in an immune response is antigen presentation and detection. There are two main actors involved in this process: a T-cell and an antigen presenting cell. When a cell is invaded by a foreign protein (the antigen), it dissects the protein into smaller peptides, called epitopes. It is the responsibility of MHC proteins to bind to these epitopes and present them on the surface of the cell. In this context the cell is often referred to as the antigen presenting cell. The immune system of the organism subsequently decides what action to take depending on the presented epitopes. As can be seen in Figure 2.1 the epitope is (quite literally) presented by the MHC like a sausage in a hotdog.

Note that not only the epitopes of foreign proteins are presented. As part of the function of a cell, proteins are constantly being produced and broken down. These proteins are also presented on the surface of the cell. T-cells only bind to specific epitopes, in particular to the ones that indicate a threat

to the organism. One would expect this to be a one-to-one relationship, but the process is not quite lock and key. We know that the TCR of a T-cell in general only binds to one epitope, but there are often multiple different TCRs that bind to the same epitope.

Figure 2.1 shows that a cell typically has many different MHC molecules, each presenting a specific epitope, whereas a T-cell only has one type of TCR that is responsible for binding to a specific epitope. This binding is mainly determined by the CDR3 region of the TCR. In some cases, this binding is very tight. In others, it is weak and short-lived, but the binding always shows great specificity [1].

On top of that, every TCR consists of an alpha and a beta chain. Both chains have to bind to the epitope in order to trigger an immune response. Unfortunately most modern sequencing techniques aren't able to determine which alpha chain corresponds to which beta chain. At least not in an efficient and affordable enough way to enable the collection of a reasonable amount of data. For now, experimental results only indicate the binding between one of the chains to an epitope.

2.1.2 Properties of Amino Acids

Protein interaction or protein binding is a complex phenomenon which is not yet fully understood. Many factors are involved, including shape, relative position, charge, mass, hydrophobicity and many others. Some of these features depend on others and are difficult to measure. Others are more fundamental and easy to characterize.

In this thesis we focus on the fundamental properties of amino acids. Five properties are considered: *Charge*, *Hydrophobicity*, *Polarity*, *Mass* and *Hydrophilicity*. Their respective values per amino acid are included in the Appendix in Table A.1. Figure 2.2 shows their pairwise Pearson correlation coefficients and correlation plots.

The following sections explain the intuition behind each of the features and why they are relevant in this context. All values were accessed through the *biopython*[2] and *pyteomics*[3] [11; 19] Python packages, which based them on peer reviewed publications, as cited per feature.

[2] `https://biopython.org/`
[3] `https://pyteomics.readthedocs.io/`

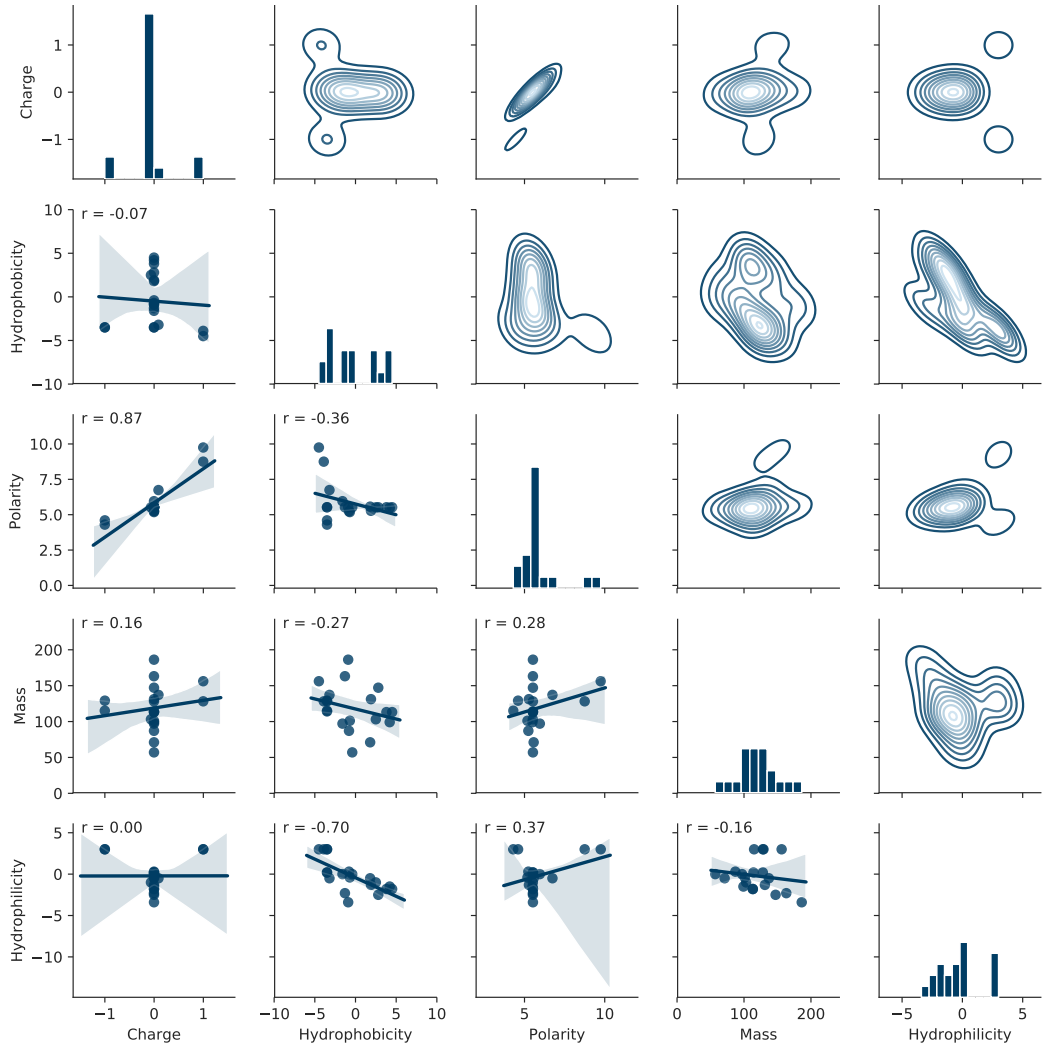


Figure 2.2: Distributions and correlations of amino acid properties. The Pearson correlation coefficient (r) is displayed for each combination.

Charge

Charge is a chemical property that simply indicates whether the molecule has a positive or negative electrical charge. Opposite charges attract each other and molecules with the same charge repel each other. As can be seen in Figure 2.2 and Table A.1 most amino acids are neutral (they have next to no charge) and only a few of them are positively or negatively charged.

Values were derived from the following studies: [3; 23; 18].

Hydrophobicity

Hydrophobicity indicates how much a molecule is repelled by water. Since hydrophobe molecules try to get away from water, they tend to get close to other hydrophobe molecules in order to minimize their contact with water.

Values were derived from the following studies: [17].

Hydrophilicity

Where hydrophobicity is a measure for how much a molecule is repelled by water, this one indicates how much a molecule is attracted to water. It is hence no surprise that their correlation coefficient r is -0.70, indicating that they are, to a certain extent, inversely proportional. Hydrophile molecules are often found together with a layer of water acting as the "glue" between them.

Values were derived from the following studies: [13].

Polarity

Polarity indicates how asymmetric the charge is distributed on a molecule. A molecule is apolar if it does not have a clear positive and negative part. Polar molecules are more positive or negative on a specific side. A high correlation coefficient (0.87) is found between the values for polarity and charge which is in line with their chemical meaning. In contrast to charge, two polar molecules or two apolar molecules attract each other and apolar ones repel polar ones and vice versa.

Values were derived from the following studies: [5; 4].

Mass

In comparison to the other properties, mass is simpler to define and calculate because every atom has an atomic mass value (expressed in Daltons). Molecular mass can be calculated by simply summing up the mass of each atom. It is important for binding because of its impact on the shape and position of each molecule.

2.2 Background in Computer Science

Some concepts in computer science that are essential for the full understanding of this thesis are explained in this section.

2.2.1 Classification

Assigning a label to a sample is more formally known as classification. With sample we refer to a single instance in a dataset, which is also commonly called an observation. In our case we perform binary classification on the interaction between two proteins. They can either interact with each other or they do not. Classification is a form of machine learning where we try to train a model that not only represents, but also generalizes some dataset.

The main application of these models is to predict the labels of unseen instances. Take for example the simple case of handwritten character recognition. A model is trained by feeding it many samples of handwritten characters along with the characters they represent. The goal of the model is to learn the relation between the handwritten characters (which are pixel maps) and the actual characters they represent. If the model succeeds to do this, we can then feed it unseen samples of handwritten characters and obtain the estimated characters they represent.

An important caveat to beware of, is the pitfall of overfitting. A naive model may succeed very well in classifying its training data, but stil fail at classifying unseen instances. If this happens, the model has simply learned the exact mapping between each input sample and its label rather than generalizing the relation between the two. This phenomenon is called overfitting.

2.2.2 Verification of Classifiers

There are various techniques and metrics to assess the predictive performance of classifiers. A first limitation to overcome however, is to make these metrics independent of overfitting. It is trivial to "train" a classifier which scores perfectly on its training dataset. That is why we calculate and report all metrics on a validation set. This set of samples is never used during training and represents the unseen samples.

A trivial method to split some dataset in a training and validation partition is the random split. It simply takes $x\%$ of the data for training randomly and leaves the remaining $1 - x\%$ for validation. Note that this approach may not be quite representative enough for actual unseen samples due to two reasons:

- There may be a strong correlation between the samples in the training set and the validation set.
- Not all samples were considered in the validation set. A large amount of particularly hard (or easy) to classify samples can end up in the validation set, which would skew the results.

The first issue can be solved by not sampling randomly but stratified. During the sampling we can take known, strong correlations into account to try and minimize this effect. Similarly we can also change the distribution of samples per class to account for an undesired class imbalance in the dataset.

The second point is solved by using K-folds cross-validation. This technique first splits the dataset in K partitions (folds). Then in K iterations each time one of the folds is left out to be the validation set and a model is trained on the $K - 1$ remaining folds. In the end, every fold has been in the validation set and we obtained K sets of metrics on which further statistical analysis can be performed (e.g.: calculate average and standard deviation). Typically 3 to 10 folds are used depending on the application.

The set of metrics used to evaluate a model also depends on the application. In the following sections we describe the most common and descriptive metrics that are used for binary classification and how to interpret them. Unless specified otherwise, all metrics are calculated on the validation data. A few abbreviations are used:

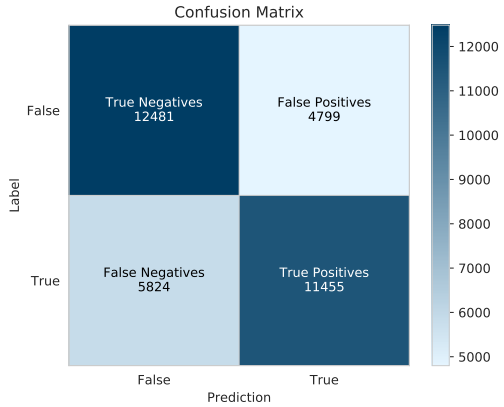
P: Positives
= The amount of positive samples.

N: Negatives
= The amount of negative samples.

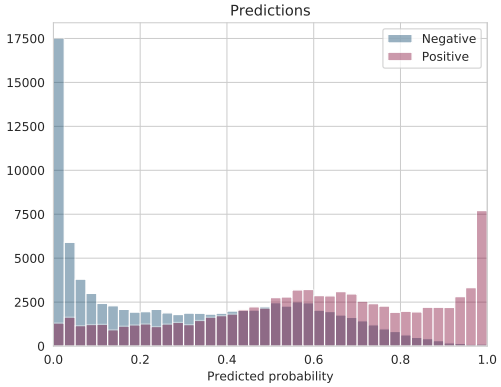
TP: True positives
= The amount of positive samples that are labeled positive.

TN: True negatives
= The amount of negative samples that are labeled negative.

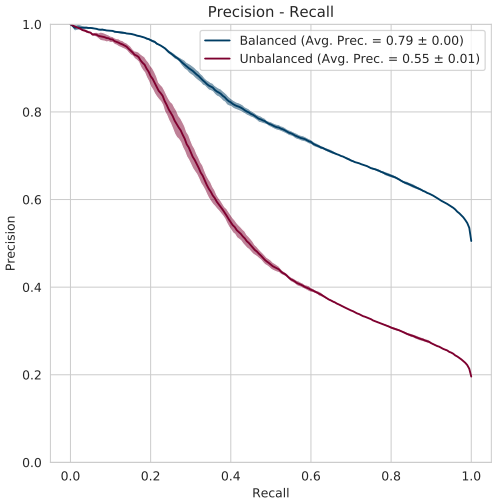
FP: False positives
= The amount of negative samples that are labeled positive.



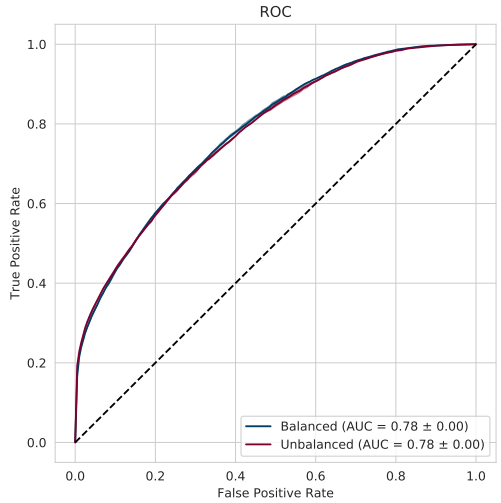
(a) Confusion matrix for the experiment with balanced data and threshold 0.5.



(b) Predictions histogram for the experiment with balanced data.



(c) Example precision-recall curves for balanced and unbalanced datasets.



(d) Example of two ROC curves for balanced and unbalanced data.

Figure 2.3: Examples of classification metrics. Two reference experiments were performed to generate these visualizations, one with a balanced dataset (50% positive) and one with an unbalanced dataset (20% positive).

FN: False negatives
= The amount of positive samples that are labeled negative.

TPR: True positive rate
= TP / P

FPR: False positive rate
= FP / N

In this context we indicate by "labeled positive" that the classifier assigned the sample a probability of being positive that is higher than some threshold value. Samples with probabilities lower than this threshold are labeled negative. Indeed we are not limited to using 50% as a fixed threshold to label samples. This enables us to, for example, define a more conservative classifier by simply raising the threshold and hence only allowing the samples we are more certain of to be labeled positive.

These metrics can also intuitively be visualized in a confusion matrix where the actual labels are set out against the predicted labels. Figure 2.3a shows an example of this. Positives and negatives are calculated by taking the row-wise sum. More complex metrics, like true and false positive rate for example, can be found by dividing values with their row- or column-wise sum. Beware that a confusion matrix shows the distribution of labels for a specific threshold only and can hence give a skewed image of the overall performance of a classifier.

Balanced Accuracy

This metrics is derived from the simple accuracy ($\frac{TP+TN}{P+N}$), but also takes class imbalance into account. It is calculated by normalizing with respect to the classes:

$$(\frac{TP}{P} + \frac{TN}{N})/2$$

Naturally a score of 100% represents a perfect classifier and a score of 50% represents a random classifier. Less intuitive is that a score of 0% would also indicate a very powerful classifier, since it is just as difficult to get all labels wrong as it is to get them all right.

This metric may be very intuitive, however it does not suffice to describe the overall behavior of a classifier since it only considers a single threshold to label samples. The next metrics can be made invariant of the threshold and are hence more informative.

Precision / Recall

Precision and recall are used to estimate how well a model can classify positive samples. Precision indicates how many of the samples that are labeled positive by the classifier, are actually positive ($\frac{TP}{TP+FP}$). It represents accuracy within the samples that are predicted positive. Recall on the other hand, calculates the fraction of positive samples that are effectively labeled positive by the classifier ($\frac{TP}{TP+FN}$). In other words it is the accuracy within the positive samples or simply the true positive rate. Average precision gives a summary of these metrics by taking the weighted average of precision values with respect to their corresponding recall for varying threshold values.

Depending on the goal of the classifier, one can choose to train the model more towards either precision or recall. In a medical setting for example it may be more important to reduce false positives and hence optimize for precision.

A precision-recall curve (PR curve) shows the relation between these values for varying thresholds. An example is given in Figure 2.3c. Ideally the curve would be as high as possible, maintaining a high precision for increasing recall. Note however that the first point is always at $(0, 1)$ and the final point at $(1, pos\_ratio)$. The class imbalance can hence directly be seen on the curve, which is an interesting property when the actual class balances are known. When this is not the case however, precision-recall curves can give a skewed representation of the performance of a classifier.

ROC curve

A ROC curve (which stands for "Receiver operating characteristic curve") is a plot which sets out the true positive rate (TPR) against the false positive rate (FPR) for varying thresholds. It always starts in $(0, 0)$ and ends in $(1, 1)$ since for a threshold of zero, nothing is marked positive and hence the TPR and FPR are both zero. Similarly with a threshold of one both the TPR and FPR are one.

Unlike a precision-recall curve, a ROC curve visualizes the predictive power of a classifier independent of class distributions. This can be seen in Figure 2.3d. Every combination of TPR and FPR on the curve can be obtained by setting the corresponding threshold (which is not visible on the curve) for the classifier.

A metric that is commonly used together with the ROC curve is the area under the curve (AUC). It is calculated as the surface under the ROC curve and represents the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

Predictions Histogram

Perhaps the simplest visualization to represent a classifier is a histogram of the predicted probabilities per label. An example can be found in Figure 2.3b. It makes the raw results of the classifier very clear and it is easy to interpret.

One can intuitively derive the ROC curve from a predictions histogram by "sliding" the threshold from right to left and counting the samples on the right of the line as being classified positive. From this the true and false positive rates can be calculated to form the points of an ROC curve.

Similarly it is possible to derive a precision-recall curve by again varying the threshold from high to low and calculating the resulting precision and recall values. In this case recall would be calculated in a non-decreasing fashion, so the precision-recall curve is formed from left to right.

There are however some drawbacks to this visualization. There is no trivial way to display multiple series with it and there is no simple metric to "summarize" it (compared to the other plots). Combined with the fact that it has a strong relation with the ROC curve and precision-recall curve, the latter two visualizations are usually preferred.

2.2.3 Artificial Neural Networks

Artificial neural networks (or neural networks for short) were inspired by the system of neurons in the brain. A neural network can be used to learn the complex function that maps samples to labels and as such, act as a classifier. In essence a neural network is nothing more than a composition of perceptrons, where each perceptron is a non-linear function (activation function) applied to some weighted sum of its inputs. In recent years they have gained in popularity, in part due to their success in image classification.

In comparison to other machine learning algorithms, neural networks arguably have the most parameters. The entire structure of the network determines the model and there are limitless possibilities of arranging the perceptrons (also called neurons in this context). There are however some parts that are the same for all neural networks. Namely they all have a set of input neurons and a set of output neurons with any combination of "hidden" layers in between. A layer describes a set of neurons that depend on the output from the preceding layer.

The training of a neural network is achieved by calculating the output based on the given inputs and comparing it to the desired output (the label). From the difference between the real value and the desired value, a gradient can be calculated and applied to each neuron individually. This is done for all samples in a back propagation fashion. Often the samples are also grouped in batches to yield a faster and more stable training process, since training on single samples sometimes introduces "noise", which is smoothed out by taking the average over a batch. The process of feeding the neural network training samples is repeated for the entire dataset. An epoch indicates a complete iteration of the training set, after which a validation step is performed.

To conclude this section on artificial neural networks, we give a brief explanation for all of the layers that were used in our models or in the reference

models. The goal of this background information is to provide sufficient context to understand the idea behind the models explained in Section 4.4. It is not intended to be a complete overview, and will hence not go into detail about the underlying mathematical foundations of these layers.

Convolutional Layer

Two of the things neural networks excel at are feature extraction and deep learning. In the example of image recognition, images are processed by convolutional layers. A convolutional layer is a set of relatively small kernels that slide over the image to generate a new image, where a kernel is just a matrix of weights that the input is multiplied with before taking the sum. Due to this recursive nature, convolutional layers can be applied hierarchically, each one learning from the outputs of the preceding layer. This is called deep learning.

When a convolutional layer is used on a two-dimensional matrix, there are two ways to apply the kernel. On the one hand, the kernel can slide in both dimensions, which is mostly used in the context of image recognition. On the other hand, for features that have a more linear nature, it is better to create a kernel with the same width as the input and to only slide it in one dimension, over the the length.

If multiple kernels are used per layer, this acts as an extra dimension to the output. For two-dimensional convolutions (which can also be applied to three-dimensional inputs), the full depth is considered and each kernel creates a layer in the output. Similarly in one-dimensional convolutions the aforementioned length is the accumulation of vectors generated by different kernels. In this context it makes more sense to think of the length dimension as being the depth.

Dense Layer

Where convolutional layers are typically associated with feature extraction, dense layers are often considered to be the “deciders” of the network. A dense layer is simply a set of neurons that are all fully connected to the neurons of the preceding layer. In other words, every neuron outputs a value based on the weighted sum of the nodes in the preceding layer. They are mostly used at the end of the network to form a decision based on the features provided by preceding convolutional layer(s). For binary classification, we always have a single output neuron with the sigmoid activation function, mapping positive samples to one and negative samples to zero.

Max Pool Layer

Like a convolutional layer, the max pool layer also uses a sliding window over its input. However this layer aggregates the values rather than transforming them. A max pool layer outputs the maximum value in its window and usually slides the window with a step size larger than one, hence decreasing the size of the output. For example, a max pool layer with step size of two by two will decrease the size of its input to one fourth of its original size. Again there is a one- and two-dimensional variant depending on over how many axes the window can slide.

Dropout

When neural networks are given more input features and more computational capacity (by expanding the architecture), they also become more prone to overfitting. This is because they also gain the ability to learn the exact mapping between each input and output combination. To counteract this phenomenon, dropout is often used. With dropout you disable part of the network randomly during training.

The effect is that different neurons learn on different batches, making it harder or even impossible for the network to learn the exact mapping for every sample. Rather it will be forced to learn the more general mapping by combining the things it learned in different neurons from different batches.

It should be clear that dropout is only applied during training. The full network is used for the actual classification, for example in the validation step.

Global Max/Average Pool Layer

Another technique to counteract overfitting, is the use of a global max pool or global average pool layer. They behave similarly to a max pool layer, but rather than using a window, they aggregate the entire input using either the max or average operator.

The idea, as introduced by Lin et al. [21], is to replace dense layers, which are prone to overfitting and depend heavily on dropout regularization, with global pooling layers that are less general and ensure a stronger correspondence between feature maps and labels. Additionally they provide the benefit that their output size is fixed by the depth of the preceding layer, rather than the dimensions of the input. This gives the advantage that the model can be applied to inputs of various sizes.

Batch Normalization

This is one of the more technical layers. Put simply, this layer normalizes the output signals with respect to the current batch. It increases the overall speed, performance, and stability of the network by minimizing the interdependence between internal layers. When the weights of a layer change, the distribution of its outputs change as well. Normally all succeeding layers have to compensate for this change, but by using batch normalization, the network does not need to take this into account anymore, resulting in a faster converging training process and more stable results.

In theory batch normalization also regularizes the network, decreasing the need for dropout. When using the two regularizers in combination, it is of course best to perform batch normalization after dropout, such that only the neurons that participate in the batch are used in the calculation of the normalization parameters.

LSTM

LSTM, which stands for long short-term memory, is essentially a memory cell that can be used in neural networks. Much like a convolutional layer, it has the ability to combine multiple features from the preceding layer. Only here it is achieved by storing intermediate values and updating them accordingly. They are mostly used in the context of continuous data like handwriting or speech recognition, but can also be applied to text sequences or images.

Embedding

An embedding layer is used to transform discrete encodings of categoric variables into a continuous vector space. For example, we could assign numbers from 1 to 20 to each of the 20 amino acids in alphabetical order and train a classifier with this discrete encoding. There is however no apparent reason in this encoding for why the distance between *Alanine (A)* and *Histidine (H)* is 6 for example, i.e. this encoding holds no meaning with respect to the amino acids.

By first embedding them in a continuous vector space, we can ensure that "similar" variables are mapped closer together and hence that the encoding reflects their meaning. The notion of "similarity" however depends on the data that is used to train the embedding.

CHAPTER 3

Related Work

To illustrate the effectiveness of our new feature encoding technique for amino acid sequences, we focused on two use cases: predicting TCR to epitope binding and predicting protein-protein interactions (PPIs). Related work for the first use case is discussed in Section 3.1 where we first give an overview of the context and then discuss specific studies. Similarly for the second use case, Section 3.2 provides a brief overview of related work in predicting protein-protein interactions.

3.1 TCR-Epitope Binding

The prediction of epitope recognition by TCRs is still a very young problems in the field of bioinformatics. Reasonable amounts of data of sufficient quality have only recently become available. One example is the publication of the VDJdb dataset in 2017, which is a curated database of TCR sequences with known antigen specificities [30]. Due to the relative immaturity of this problem statement, there is also not a varied spectrum of related work available. We selected three papers with the strongest link to our method and discuss them in detail.

Previous work by De Neuter et al. [7] demonstrated the feasibility of predicting T-cell receptor to epitope binding using random forest classifiers. Where they built a classifier per epitope, our approach targets the more general problem of predicting binding between arbitrary TCRs and arbitrary epitopes. More concretely the key difference is that our approach accepts both a TCR CDR3 sequence and an epitope as input compared to their approach where only CDR3 sequences are provided to a range of models, one for each supported epitope. Similarly to our technique, they also based most of their features on physicochemical properties. This was done per position and also by taking the average over the entire sequence.

Based on this work by De Neuter et al., a web tool called TCRex was subsequently implemented by Gielis et al. (from the same research group) [9]. Their application supports 49 different epitopes: 44 viral epitopes and 5 cancer epitopes. Its main use case is to facilitate TCR repertoire analysis, where the user can upload a TCR sequence data file and obtain predictions for which T-cell receptors are likely to bind to which epitopes. Although multiple tools exist to predict epitopes and their binding with MHC molecules, TCRex is one of the only, publicly available tools for the prediction of epitope recognition by TCRs.

In another recent study, Jurtz et al. propose a neural network to predict epitopes recognized by cytotoxic T-cells [15]. Their approach encodes epitope and CDR3 sequences with the BLOSUM50 matrix and passes them to separate inputs followed by convolutional layers of different sizes which are then summarized and concatenated. The result is connected to a global max pooling layer and successively fed to a dense layer of 10 nodes before continuing to the final sigmoid predictor. Section 4.4.4 dives further into the technical aspect of their model. Though a different dataset is used, their study is still very similar to this one, which is why we chose to use their classifier as a reference model.

A similar problem to this one is the prediction of epitope to MHC binding. MHCflurry by O'Donnell et al. [25] and DeepSeqPan by Liu et al. [22] are two examples of recent studies in this field. In these studies they also make the distinction between general models and models trained per epitope or per MHC allele. DeepSeqPan presents a single, general model and MHCflurry provides a range of models, each specific to one MHC allele.

3.2 Protein-Protein Interactions

The main contribution of our work is the new feature encoding technique for amino acid sequences. Aside from TCR-epitope binding, another possible application for this technique is in the problem of predicting protein-protein interactions (PPIs). Since this problem is only used to illustrate the performance of our technique, we limit the scope of this related work to an overview paper to provide context, a reference implementation that is also based on neural networks and finally some work in the related field of predicting kinase-specific interactions.

An overview paper from 2013 by Zahiri et al. divides methods for predicting protein-protein interactions in four main categories: methods based on genomic context and structural information, methods that use network topology to predict protein-protein interaction, methods that detect protein-protein interaction by using text mining and literature mining (or database search) and, finally, methods based on machine learning algorithms utilizing heterogeneous genomic/proteomic features [33]. Naturally our approach is situated in the fourth category. They also mention the need for a gold standard dataset to evaluate performance where the most complex (and influential) part is selecting negative samples. We experimented with two different datasets, using different methods to select negative samples.

Li et al. [20] present a deep learning approach for predicting protein-protein interactions called DNN-PPI. The structure is similar to that of Jurtz et al., but it uses an embedding layer rather than the BLOSUM50 matrix for encoding and an LSTM layer is used after the convolutions, replacing the functionality of the global max pooling layer. Their approach was used as a reference for our PPI experiments and it is further discussed in Section 4.4.6.

There are various other classification problems in molecular biology that deal with a similar problem. For example, predicting kinase-specific interactions [27; 32]. While these different fields each have their own unique challenges, there are also many commonalities and several insights gained here into machine learning techniques could prove fruitful there.

CHAPTER 4

Experimental Setup

Before discussing the results, we first explain the context of our experiments. To begin, Section 4.1 gives an overview of how the amino acid properties were used for feature encoding. Then Section 4.2 discusses the different datasets that were used in our experiments. In Section 4.3 we explain how the data is split and processed into the correct format. Section 4.4 gives an overview of the six different neural network models that we tested and finally, in Section 4.5 we discuss our experiment plan, listing which experiments were performed and why.

4.1 Features

Feature selection and encoding is a fundamental aspect in every classification task. It can greatly influence the performance, training speed and generalization ability of the resulting classifier. The way we encode features is what characterizes our approach.

In all of our models, we work with a single input that encodes the information of two amino acid sequences in an "interaction matrix". This matrix is calculated from the pairwise combinations of amino acids in the sequences.

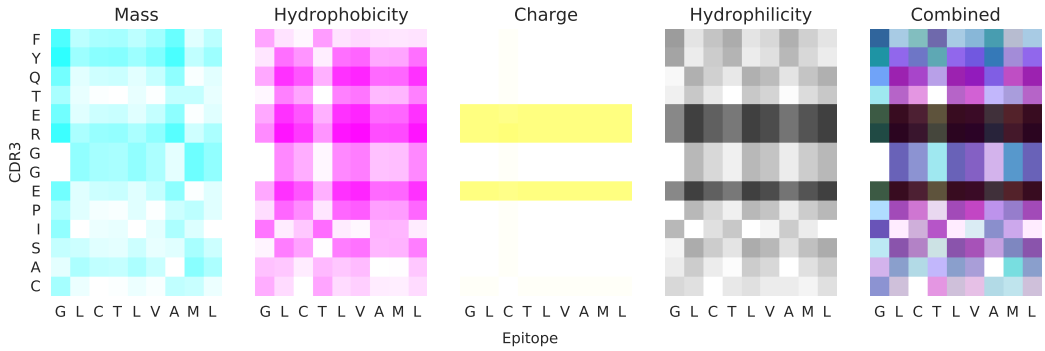


Figure 4.1: An epitope and CDR3 sequence encoded with four physicochemical properties combined with the abs diff operator. The rightmost image shows the CMYK color encoding, which is the result of combining the first four layers.

Naturally there are many ways to combine and transform a pair of amino acids into a number. We chose for the most straight forward approach of first converting each amino acid to a number and then combining them using some mathematical operator.

This of course leaves the choice of number mapping and of operator open. Since we are working on a very low, fundamental level, comparing one amino acid to another, we chose the most basic physicochemical amino acid properties that are known to play a role in protein binding, namely: *charge*, *hydrophobicity*, *hydrophilicity*, *polarity* and *mass*. An overview of these properties and their corresponding scales is given in Section 2.1.2.

As for the operator, we experimented with four different options: *product*, *difference*, *absolute difference* and *"layers"*. The first three are quite trivial and evident choices. The latter operator *"layers"* refers to the idea of not applying any operator ourselves, but rather having the model figure out the best way to combine the values for each amino acid. It works by transforming the two input vectors of numbers into three matrices:

- The first matrix is the result of replicating the values of the first vector along the horizontal axis.
- The second one replicates the values of the second vector along the vertical axis.
- The third matrix is calculated by taking the matrix product of both vectors (same as the *product* operator).

These matrices were strategically chosen such that when they are overlayed, all possible combination are present through the first two layers and the final

layer contains their product, which is otherwise not trivially computable by a convolutional layer. This ensures that the network has enough information to figure out the ideal combination.

Luckily we are not limited to a single choice of property and operator. Convolutional neural networks support a variable number of "channels", which correspond to the depth of a three-dimensional matrix. The term channels is often used because of the link with image classification. A colored image consists of multiple channels, one for each base color, and their combination forms the resulting composite color.

Similarly we can represent our feature encoding as an image encoded in the CMYK color scheme. An example input matrix for the four features *mass*, *hydrophobicity*, *charge* and *hydrophilicity* with the absolute difference operator (abs diff) is displayed in Figure 4.1. Unfortunately the CMYK format only supports four layers (cyan, magenta, yellow and "key", or simply black) which limits our visualization capacities, but neural networks can of course accept an arbitrary amount of layers.

This encoding was partly inspired by sequence alignment algorithms like for example the Smith-Waterman algorithm [31]. In these algorithms two sequences are typically placed on the edges of a "scoring matrix" that contains an alignment score for each combination. High scoring diagonal stretches then represent parts of the sequences that align well with each other. Similarly in our technique we aim to find diagonals after applying some complex calculation on the features, since a diagonal would indicate that the two sequences "match" in a sense. Furthermore other patterns could also prove interesting. Anti-diagonals for example would represent that the sequences match when one is (partly) turned such that its inverse aligns with the other sequence.

Considering that we want to detect patterns in three-dimensional matrices, the choice of using convolutional neural networks was evident. This machine learning technique excels at pattern recognition in images, which is exactly what our feature encoding technique has reduced the problem of predicting binding between amino acid sequences to.

4.2 Datasets

Two types of datasets were used. The first one contains experimental data on the interaction between TCRs and epitopes presented by MHC molecules. It

is discussed in Section 4.2.1. The second type consists of interactions between proteins in general. Sections 4.2.2 and 4.2.3 describe two instances of this type of dataset.

4.2.1 VDJdb

VDJdb is a curated database of TCR sequences with known antigen specificities. It was released in 2017 with the primary goal to facilitate access to existing information on known epitopes presented by known MHC class I and II molecules [30]. Continuous updates are provided by the scientific community when new discoveries are made.

It is accessible in CSV format and contains the following columns (among others):

gene
: Either *TRA* or *TRB*. As explained in Section 2.1, most T-cell receptors consists of an alpha and a beta chain. This field indicates whether the CDR3 sequence came from an alpha (TRA) or beta (TRB) chain.

cdr3
: Sequence for the CDR3 part of the TCR.

species
: The VDJdb dataset includes entries from three different species: *Homo Sapiens* (human), *Macaca Mulatta* (rhesus macaque, a type of primate), *Mus Musculus* (house mouse).

antigen.epitope
: Sequence of the epitope.

mhc.class
: Either *MHCI* or *MHCII*. This indicates the class of MHC that presented the epitope.

mhc.a, mhc.b
: Each class of MHC molecules can originate from different genes. These two fields encode the gene for each MHC chain. For Class I there is only one chain and a constant part (Beta-2 microglobulin). HLA-A, HLA-B and HLA-C are the most common class I chains in humans. MHC class II molecules always have two different chains of the same type. For humans the most common types are HLA-DP, HLA-DQ and HLA-DR.

Some other fields like references and more detailed gene information are also available, but they are not relevant in this context.

Table 4.1: Overview of VDJdb dataset as of October 30, 2018.

Species	Gene	Interactions	Epitopes
HomoSapiens	TRA	6,000	91
	TRB	14,941	147
MacacaMulatta	TRA	74	1
	TRB	1,312	3
MusMusculus	TRA	1,413	25
	TRB	1,680	32
Total:	TRA	7,487	117
	TRB	17,933	182

Statistics

The full dataset contains 25,420 interactions as of October 30, 2018. It consists of 24,771 unique cdr3 sequences and 182 unique antigen epitopes. The majority of the data (96%) consists of MHC class I molecules. A more detailed dissection of the data can be found in Table 4.1 and Figure 4.2, where chart (A) of Figure 4.2 corresponds to Table 4.1.

In chart (B) we see the distribution of MHC types per species. The dataset mostly contains HLA-A and HLA-B MHC molecules. Charts (C) and (D) include histograms of the length distributions for the epitope and CDR3 sequences respectively. Finally in chart (E) we can see that the amount of interactions is not uniformly distributed over the different epitopes. Some interactions are more studied than others. The first two epitopes correspond to the human herpesvirus 5 and influenza A virus respectively. Every epitope is also unique to one species and one MHC class.

All dataset visualizations were generated with the Metabase[1] app. This was done by loading the data in a PostgreSQL database and then linking the database with the Metabase webtool.

Another dataset that consists of 500,000 TRB CDR3 sequences from a study by Dean et al. [8] was used as a reference to generate negative samples in some experiments. Only 1,246 out of 17,933 sequences of the VDJdb overlap with this dataset. Given the large amount of CDR3 sequences to sample from, they can be considered to be independent.

[1] `https://metabase.com/`

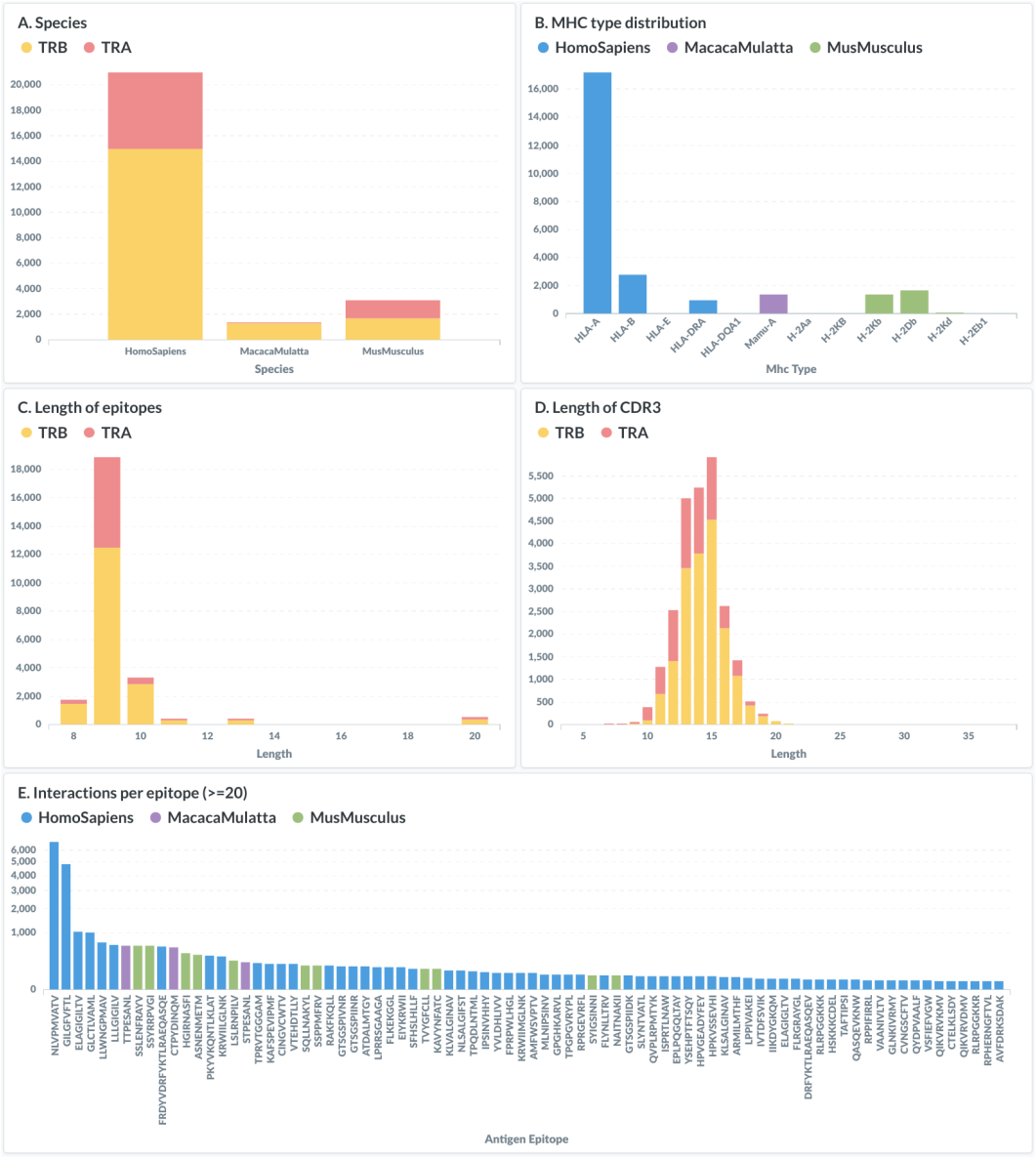


Figure 4.2: Distribution of VDJdb.

A. Amount of interactions per species.
B. Amount of interactions per MHC type per species.
C. Length distribution of epitope sequences.
D. Length distribution of CDR3 sequences.
E. Amount of interactions per epitope. The y-axis is in log scale.

Preprocessing

In our experiments we mostly used a subset consisting only of TRB genes. It is clear that they make up the largest part of the VDJdb dataset and are hence the most documented sequences. The idea is to build a classifier that can generalize the binding between TCRs and epitopes, but since no information is available for both chains, we chose to focus on the TRB chain. An experiment to investigate the impact of this filtering is discussed in Section 5.1.4.

All columns except *cdr3* and *antigen.epitope* were removed because they did not contribute relevant information for training. Though this projection was not technically necessary, it did help speed up the parsing of the file.

4.2.2 Human Interactome (HI) PPI Dataset

Two datasets were used in our protein-protein interactions experiments. The first one is the third generation dataset (HI-III) from the Human Reference Protein Interactome Mapping Project (HuRI)[2]. It contains positive entries for known interactions between human proteins. As of the time it was accessed (April 13, 2019) it was not yet published. Hence the interactions were verified but not yet validated, which should not have an impact for this study since it is only used as a proof of concept for our technique. Unlike the one-to-many relation of the immunological dataset, the interactions described here can be considered to be many-to-many.

Statistics

Figure 4.3 shows some statistical information about the dataset. It is clear from chart (A) that most of the proteins have a length not higher than 1,200 amino acids (97.5%). About 80.6% of the proteins have a length of at most 600 amino acids. From chart (B) we can derive that most proteins take part in multiple interactions, leading to the many-to-many relation. In total the dataset contains 8,112 unqiue proteins, participating in 77,214 interactions.

Preprocessing

In this case only two fields from the dataset were needed: the sequences for each of the interactors. Due to the relative large lengths of these proteins and the fact that they occur in many interactions, the dataset only contains the sequence identifiers and not the actual sequences. We wrote a Python script to asynchronously fetch all sequences from the proteins API of the European Bioinformatics Institute (EBI) and store them in a separate CSV file.

[2] `http://interactome.baderlab.org`

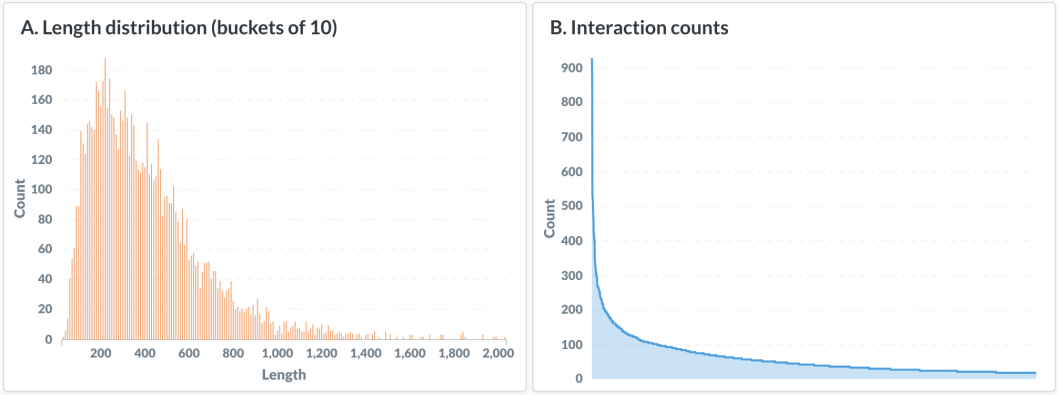


Figure 4.3: Distribution of HI PPI dataset.

A. Distribtion of protein lengths. A bucket size of 10 was used to group the lengths.
B. Amount of interactions per protein.

This file functions as a map from protein identifiers to sequences. No further preprocessing was needed.

4.2.3 Pan's PPI Dataset

The second dataset for predicting protein-protein interactions is based on previous work by Pan et al. [26]. It was selected because it is also used in the study by Li et al. [20] with which we compare our technique. By using the same data, we aim to minimize the potential bias introduced in our comparison.

Unlike the previous dataset, this one contains negative as well as positive samples. The negative samples were generated by pairing proteins found in different subcellular locations, ensuring that possible interactions between them are highly unlikely [20]. Both datasets consist of human proteins.

Statistics

In Table 4.2 we can see a general overview of the dataset. There are about the same amount of positive and negative interactions. However the number of unique proteins they contain differs by a factor of almost five. Overall 1,296 proteins occur in both datasets as indicated by the "Intersection" column.

Figure 4.4 shows the same statistics as for the previous dataset. We find that the protein length is higher on average compared to the first set. About 90.7% and 65.6% of the sequences have a length of at most 1200 and 600 amino acids respectively. The relative distributions of protein length and of

Table 4.2: Overview of Pan's PPI dataset.

Label	Interactions	Proteins	Intersection
Positive	36,590	9,469	1,296
Negative	36,480	2,184	

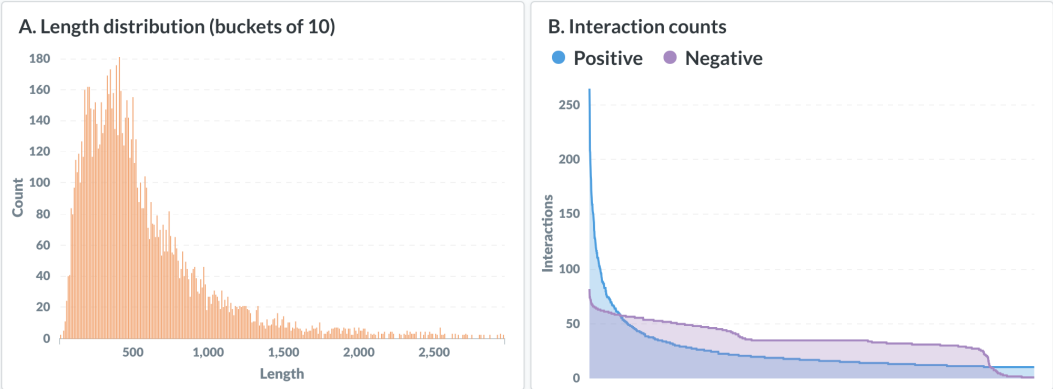


Figure 4.4: Distribution of ppi Pan's PPI dataset.

A. Distribtion of protein lengths. A bucket size of 10 was used to group the lengths.
B. Amount of interactions per protein (limited to 2000).

positive interaction counts do seem to be similar, as is expected from datasets describing the same biological phenomena.

For the negative interaction counts a different distribution is found. This is to be expected and can be explained by the process that generated these samples. The counts are more uniformly distributed which corresponds to the random pairing of proteins from different subcellular locations.

Preprocessing

Parsing this dataset was somewhat more troublesome. Despite numerous standards being available for encoding protein-protein interactions or protein data in general, this dataset was made available in a Microsoft Word document with a custom encoding. After filtering out all redundant information, we were able to parse and transform the data to the same CSV format as described for the previous dataset.

4.3 Data Processing

Training data needs to be in a specific format depending on the model. Some models can handle inputs of variable sizes, where others require a fixed input size. In this section, we describe how we implemented this. Furthermore we also document the process to split the input data into train and validation sets and we touch on how negative samples are generated.

Because of the exploratory nature of this research, we invested in implementing a general framework for data processing, consisting of simple blocks that can be combined into a custom data processing flow. This approach enabled us to reuse a large portion of the code for different models and it also simplified debugging, since the output of each block could easily be inspected and compared to what was expected.

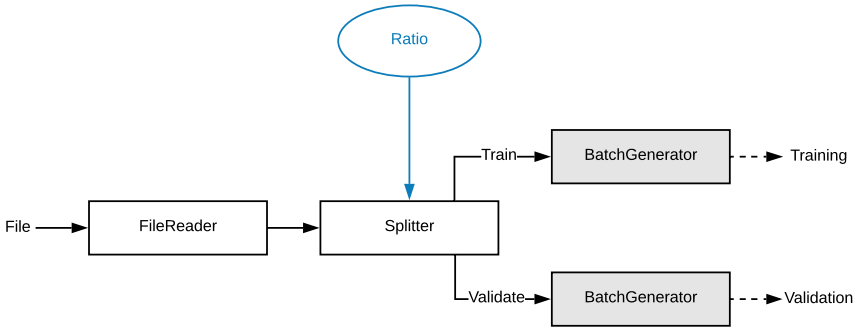
With an online visualization tool[3], we modeled our different data processing flows before implementing them. This both saved time, because operations were first stripped down to their atomic elements in the design phase, and also it made the entire processing flow more conceptually clear. The latter helped in discussing different approaches and spotting misconceptions about the data and algorithms at an early stage.

Figure 4.5 show the general overview of a classification task with either a random split or K-Folds for cross-validation. First the input file is parsed by some *FileReader*. Specialized implementations were made per data source type. Then the available data is split into a train and a validation set either with a random split or by using K-Folds. Unless stated otherwise, 5 folds cross-validation was used. After this, each dataset is processed by a custom *BatchGenerator* which converts the data to features that are used as input to the model.

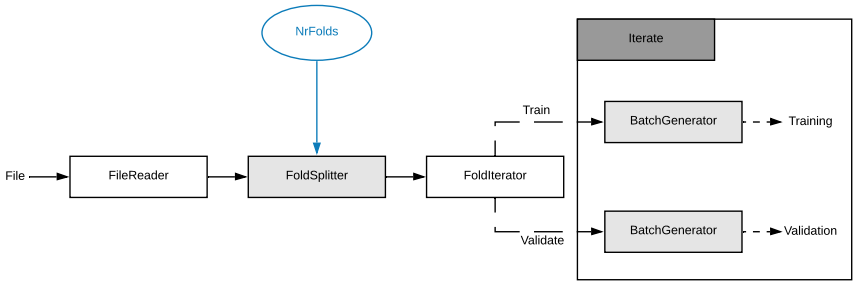
In this visualization, boxes represent operations and blue circles are parameters. Arrows indicate the flow of data. A full arrow means the data is passed all in one go. Dashed arrows represent a set of batches flowing between the operations. Finally dotted arrows indicate that single items are passed between the operations.

The next two sections discuss two *BatchGenerator* implementations, each for their own type of model with specific requirements.

[3] `https://lucidchart.com`



(a) Cross-validation with a random split.



(b) Cross-validation with K-Folds.

Figure 4.5: Two cross-validation approaches: random split and K-Folds.

4.3.1 Grouped Batch Generator

This *BatchGenerator* outputs batches of same-sized images. Models that support variable-size inputs often do require the size to be the same within a batch, which is what this flow realizes.

As shown in Figure 4.6, it does this by first grouping all samples by their shape (the size of each dimension, two in this case). Then sets that are too small to create a full batch are filtered out (as indicated by the *MinAmount* parameter). After this, the features are extracted for all items by generating their images with a specific *FeatureBuilder*. Finally these groups, or sets with a specific shape, are sampled randomly and within each group a batch is sampled. These sampling steps are weighted to make sure the resulting distributions remain the same.

Since most datasets don't contain negative samples, we generate them by shuffling the positive samples. Shuffling happens by first removing the label and then splitting the two peptides (CDR3 and epitope). They are then both independently grouped by their length. From this the *BatchExtender* can request a batch of a specific size (the same one as was randomly selected from the positive branch). The chance of generating a positive match this way is rel-

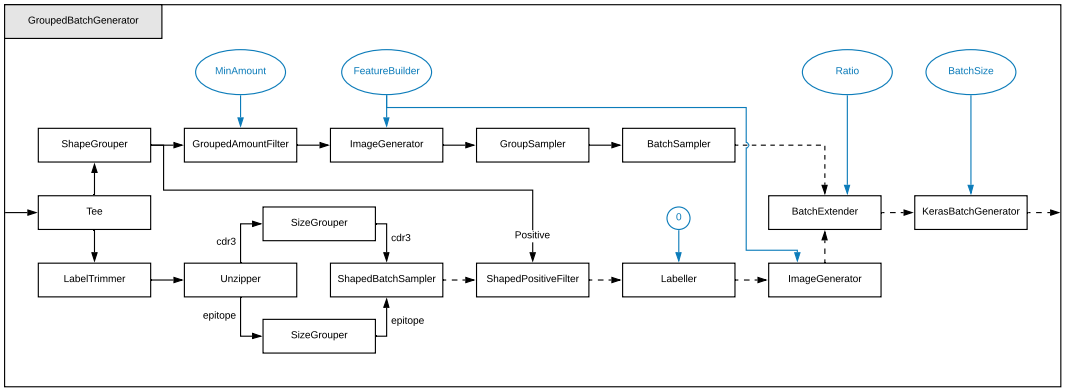


Figure 4.6: Data processing flow for *GroupedBatchGenerator*.

atively small, but it does happen often enough that positive samples need to be filtered. That is why after a batch is selected it is cross referenced with the positive samples to make sure no known positive samples pass through. Finally they are labeled as negative and the image is generated for each sample.

As described by Park et al. [28], shuffling is a commonly used technique to generate a negative dataset when only positive data is available. They do however warn that the sampling should be truly random and uniform over the positive dataset to best be able to generalize to the population level (at least for PPI experiments). With the described method we do in fact sample randomly and hence our results should be representative and generalizable.

With both positive and negative samples combined in a batch (depending on some ratio), all that remains is to transform them to the format used by *Keras*[4] and pass them to the model. Keras is the Python package we used for training neural networks. Despite the ratio parameter being available, we only worked with an equal distribution of positive and negative samples ($Ratio = 50\%$).

When this flow is used for the VDJdb dataset with a *MinAmount* of 32, only about 350 to 400 samples are discarded on average for the training or validation set. This corresponds to about 4.5% of the TRB data which is an acceptable (and unavoidable) penalty to achieve a proper format for training. Note that with K-Folds cross-validation it is possible for discarded samples of one iteration to still be used in other iterations.

[4] `https://keras.io/`

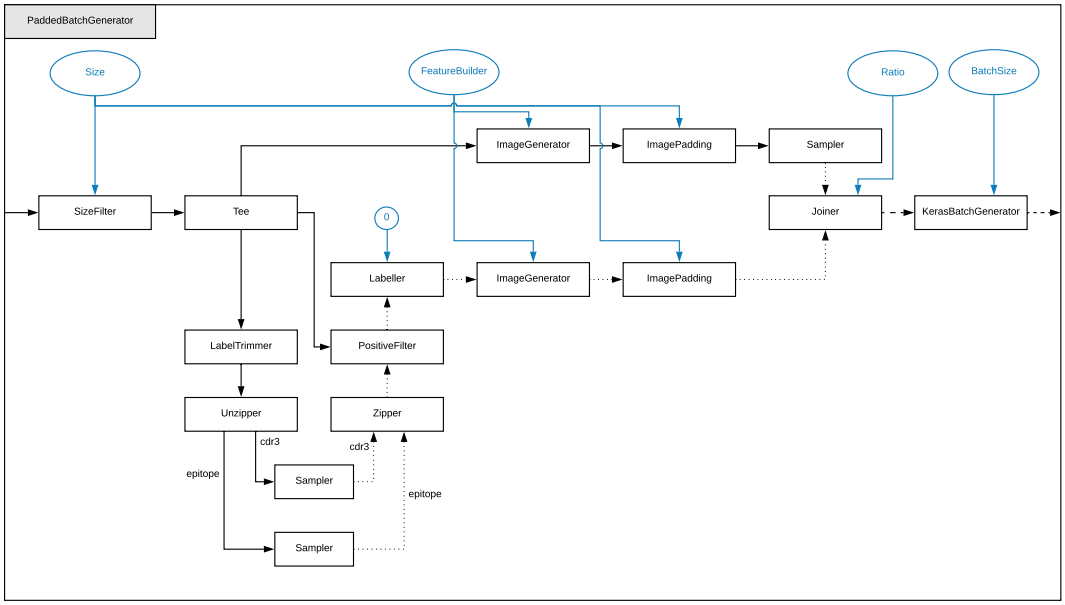


Figure 4.7: Data processing flow for *PaddedBatchGenerator*.

4.3.2 Padded Batch Generator

Models that do not support a variable input size require each item to have the same size. This *BatchGenerator* filters items that are too large and adds padding to smaller items to make every item equal in size. The flow in Figure 4.7 gives a visual representation.

Like in the *Grouped Batch Generator* we repeat the input stream to a positive and negative branch, but first the items that are too small or too large are already filtered out. In the positive branch we can simply generate the images and add padding to them. They are then individually sampled and joined with the negative samples.

Negative samples are again created by shuffling the positive samples randomly. First the label is stripped and then the two peptides are split. Both the CDR3 peptides and antigen epitopes are sampled independently to generate mismatches. If by accident a known positive match is sampled, it is filtered. Finally they are given a negative label and processed in the same way as positive samples.

For the VDJdb dataset the size constraints of 8 to 13 for the epitope and 10 to 20 for the CDR3 part were chosen. It can be seen in Figure 4.2 that these include the bulk of the data. Only 512 samples are dropped with these parameters, which is about 3% of the TRB data.

4.3.3 Post-processing

Various metrics are collected during and after the training of a model, both on the training data and on the validation set. An overview of these metrics and how to interpret them can be found in Section 2.2. At the end of the training process they are all persisted in the form of CSV files.

These CSV files are then post-processed to derive statistical information and generate informative plots. First the values of multiple iterations of the K-Folds algorithm are consolidated, resulting in the mean and standard deviation for each metric. Then they are plotted using the *seaborn*[5] and *matplotlib*[6] Python libraries.

The post-processing scripts were built in a flexible way, supporting the plotting of multiple datasets (or series) either on the same axis or in a grid view to facilitate analysis. This way multiple runs can easily be compared and viewed from different perspectives. The entire process is based on placement of folders. To compare two runs, one simply has to place (or copy) them together into a folder and run the script to get the desired visualizations.

4.4 Models

Six different models were implemented to validate our feature encoding technique. The first four are used for the prediction of TCR to epitope binding and one of these four is the state-of-the-art NetTCR model [15]. The last two models were implemented to predict protein-protein interactions and here DNN-PPI was used as a reference [20]. We explain the architecture for each of the models as well as the intuition behind why we settled on these specific neural network architectures.

4.4.1 Padded Model

Our first model relies on padding of zeroes to generate input matrices of the same size. We will henceforth refer to it simply as the *padded* model. Its architecture, as described in Table 4.3, is quite standard in the context of image classification. We have a "feature extraction" part at the front followed by two dense layers to perform the subsequent classification.

[5] `https://seaborn.pydata.org/`
[6] `https://matplotlib.org/`

Table 4.3: Structured representation of the padded model. Consecutive layers are listed from top to bottom with their respective output sizes (**W**idth, **H**eight and **D**epth). A vertical line indicates no change in size (same value as above) whereas a horizontal line is used to indicate the absence of a dimension.

Layer	**W**	**H**	**D**
Input	13	20	5
Conv 2D ($128 \times 3 \times 3$)	|	|	128
Batch Normalization	|	|	|
Conv 2D ($64 \times 3 \times 3$)	|	|	64
Max Pool 2D (2×2)	6	10	|
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Conv 2D ($128 \times 3 \times 3$)	|	|	128
Batch Normalization	|	|	|
Conv 2D ($64 \times 3 \times 3$)	|	|	64
Max Pool 2D (2×2)	3	5	|
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Flatten	960	—	—
Dense 32	32	—	—
Dense 1	1	—	—

The feature extraction part is made up of two identical parts. Each consists of two convolutional layers followed by a max pool and dropout layer. Batch normalization was added at strategic locations. All convolutional layers use the relu activation function, the hidden dense layers use a tanh activation and the final output neuron was sent through a sigmoid activation to achieve predictions. The RMSProp optimizer was used to learn model weights, as is the case for all of our own models.

During the tuning of this model we experimented with different kernel sizes, varying amounts of kernels, different activation functions, different optimizers and finally placement of batch normalization, dropout and convolutional layers. In our experiments, the presented model architecture achieved the highest performance without much unneeded complexity. For example consider that a model with a depth of 256 in the first convolutional layer could achieve similar, or maybe even slightly better results in some cases, then we would still opt for the smaller, simpler model. This trade off is made to reduce overengineering and it also helps to regulate training times.

4.4.2 GAP Model

Table 4.4: Structured representation of the GAP model. Consecutive layers are listed from top to bottom with their respective output sizes (**W**idth, **H**eight and **D**epth). A vertical line indicates no change in size (same value as above) whereas a horizontal line is used to indicate the absence of a dimension.

Layer	**W**	**H**	**D**
Input	w	h	5
Conv 2D ($256 \times 3 \times 3$)	|	|	256
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Conv 2D ($256 \times 3 \times 3$)	|	|	256
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Conv 2D ($128 \times 3 \times 3$)	|	|	128
Max Pool 2D (2×2)	w/2	h/2	|
Batch Normalization	|	|	|
Conv 2D ($1 \times 1 \times 1$)	|	|	1
Batch Normalization	|	|	|
Global Average Pooling 2D	1	—	—

The second convolutional model we implemented for predicting TCR to epitope binding is similar to the padded model, however rather than using dense layers at the end, here we use a global average pooling (GAP) layer. This allows the model to remain independent of the input size, since the output size of a GAP layer only depends on the depth of the previous layer. Within a batch, all input samples do still need to have the same size however, which is exactly what the *GroupedBatchGenerator* realizes (as explained in Section 4.3.1). Its architecture can be found in Table 4.4.

Again we use the relu activation function for all convolutional layers. Contrary to our expectations, this model performed better with a single intermediate max pool layer. A likely explanation is that the GAP layer also performs an aggregation, and by not max pooling twice, we ensure a large enough resolution remains. If a second max pool layers would be added, only one sixteenth of the original input size would remain compared to one fourth in the current situation.

4.4.3 Dense Model

Table 4.5: Structured representation of the dense model. Consecutive layers are listed from top to bottom with their respective output sizes (**W**idth, **H**eight and **D**epth). A vertical line indicates no change in size (same value as above) whereas a horizontal line is used to indicate the absence of a dimension.

Layer	**W**	**H**	**D**
Input	13	20	5
Flatten	1300	—	—
Dense 512	512	—	—
Batch Normalization	|	—	—
Dense 512	512	—	—
Batch Normalization	|	—	—
Dense 256	256	—	—
Batch Normalization	|	—	—
Dense 1	1	—	—

The third and last model we created has a dense architecture with three hidden layers, as shown in Table 4.5. All hidden layers use the tanh activation function and the final neuron uses a sigmoid activation. There aren't many parameters to tweak for this model. Our experiments indicated no significant increase in performance when further increasing the amount of layers or neurons, so we settled on this architecture.

4.4.4 NetTCR Model

Finally we explain our implementation of the NetTCR model by Jurtz et al. [15]. Its architecture is described in Table 4.6. Compared to the previous models, this one makes use of two separate inputs, each with only two dimensions: length and depth. The idea behind this model is that features are extracted independently from both input sequences before they are concatenated and processed in unison. Sigmoid activation functions are used throughout the entire network. The batch normalization layer was added by us to speed up training convergence. This model was trained with the adam optimizer.

For feature encoding, this technique uses the BLOSUM50 matrix to map each amino acid to a vector of 20 integers. In short the BLOSSUM50 matrix is used to represent evolutionary relations between amino acids and, as such, contains

Table 4.6: Structured representation of the NetTCR model. Consecutive layers are listed from top to bottom with their respective output sizes (**L**ength and **D**epth). In this case the five indented convolutional layers are applied to the input layer rather than to the layer directly above. A vertical line indicates no change in size (same value as above) whereas a horizontal line is used to indicate the absence of a dimension.

Layer	**L**	**D**	**L**	**D**
Input	l_1	20	l_2	20
├─ Conv 1D (100×1)	|	100	|	100
├─ Conv 1D (100×3)	|	100	|	100
├─ Conv 1D (100×5)	|	100	|	100
├─ Conv 1D (100×7)	|	100	|	100
└─ Conv 1D (100×9)	|	100	|	100
Concatenate (depth)	|	500	|	500
(Batch Normalization)	|	|	|	|
Conv 1D (100×1)	|	100	|	100

Layer	**L**	**D**
Concatenate (length)	$l_1 + l_2$	100
Global Max Pooling 1D	100	—
Dense 10	10	—
Dense 1	1	—

an entry for each combination that indicates how likely (or unlikely) they are to be replaced by each other.

4.4.5 PPI Padded Model

To classify protein-protein interactions we decided to expand upon the *padded* model (Section 4.4.1). It can be seen in Table 4.7 that the architecture is similar, with a few key differences. Because the sequences of these proteins are considerably larger, we also have to take some practical limitations into account. That is why the input size is limited to a length of 600. Furthermore the convolutional layers are set to not use padding at the edges, making them reduce their output size by a small constant. There are also much less kernels and only one convolutional layer per max pool layer. These choices were made from practical considerations to make sure the model fit in memory and that training times remained within practical bounds.

Table 4.7: Structured representation of the padded PPI model. Consecutive layers are listed from top to bottom with their respective output sizes (**W**idth, **H**eight and **D**epth). A vertical line indicates no change in size (same value as above) whereas a horizontal line is used to indicate the absence of a dimension.

Layer	**W**	**H**	**D**
Input	600	600	2
Conv 2D ($16 \times 10 \times 10$)	591	591	16
Max Pool 2D (2×2)	295	295	|
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Conv 2D ($16 \times 8 \times 8$)	288	288	16
Max Pool 2D (2×2)	144	144	|
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Conv 2D ($16 \times 5 \times 5$)	140	140	16
Max Pool 2D (2×2)	70	70	|
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Conv 2D ($8 \times 3 \times 3$)	68	68	8
Max Pool 2D (2×2)	34	34	|
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Conv 2D ($8 \times 3 \times 3$)	32	12	8
Max Pool 2D (2×2)	16	16	|
Dropout (0.25)	|	|	|
Batch Normalization	|	|	|
Flatten	2048	—	—
Dense 16	16	—	—
Dense 1	1	—	—

Tuning this model was difficult due to the long time it took to train, even on a small subset of the data. We eventually decided on this model because the performance it achieved was adequate for our use case. It is hence quite possible that this model can be improved further with stronger hardware and more time per training session.

Table 4.8: Structured representation of the DNN-PPI model. Consecutive layers are listed from top to bottom with their respective output sizes (**L**ength and **D**epth). A vertical line indicates no change in size (same value as above) whereas a horizontal line is used to indicate the absence of a dimension.

Layer	**L**	**D**	**L**	**D**
Input	1200	1	1200	1
Embedding (20×128)	|	128	|	128
Conv 1D (10×10)	1191	10	1191	10
Max Pool 1D (2)	595	10	595	10
Conv 1D (10×8)	588	10	588	10
Max Pool 1D (2)	294	10	294	10
Conv 1D (10×5)	290	10	290	10
Max Pool 1D (2)	145	10	145	10
LSTM (80)	80	—	80	—

Layer	**L**
Concatenate (length)	160
Dense 1	1

4.4.6 DNN-PPI Model

Lastly the DNN-PPI model by Li et al. [20] was replicated to act as a reference for our PPI experiments. Table 4.8 displays the architecture of their model in detail. Like NetTCR, they also use two separate inputs to process the input sequences. This of course makes the model more light-weight since its size scales linearly with the size of the input, rather than quadratic like in our PPI model. It uses the adam optimizer for training.

For feature encoding they use a random number for each of the amino acids, which is then passed to an embedding layer that creates a feature vector of size 128. After three convolution/max pool combinations they apply an LSTM layer with 80 cells to give the output a fixed size. The two outputs are then concatenated and passed to a single output neuron. It is interesting to note that, despite the fact that protein-protein interactions are symmetric, this model still mostly uses two separate networks that are only joined at the end.

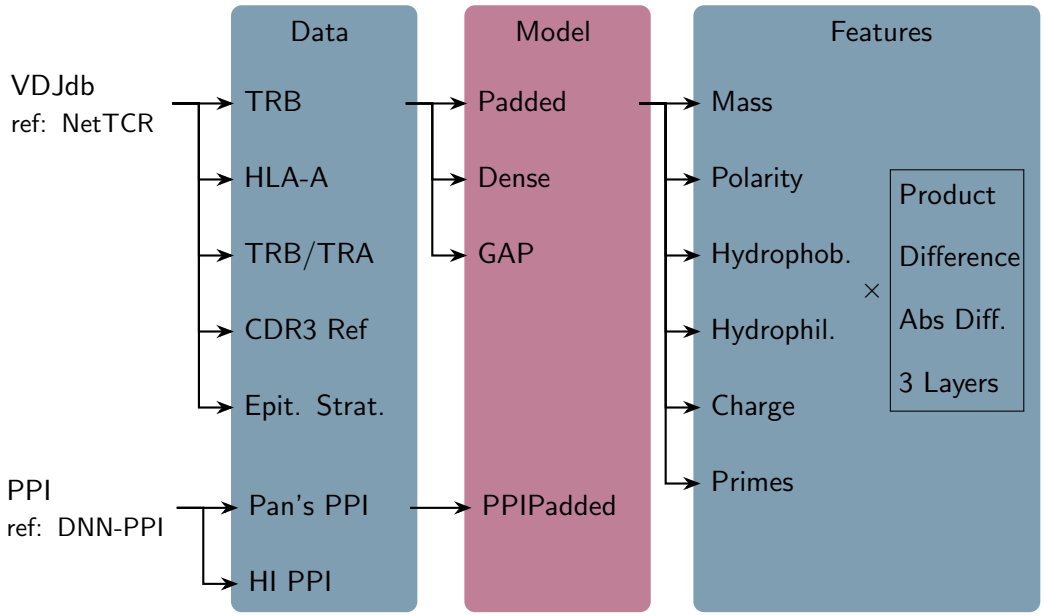


Figure 4.8: Schematic visualization of the experiment plan.

4.5 Experiment Plan

Our main goal is to present a novel feature encoding technique for predicting interactions between two amino acid sequences. We selected two distinct use cases to demonstrate the performance of our technique: T-cell receptor to epitope binding and protein-protein interactions. Within these two classes there are however many more variables that all influence each other. We can largely group these variables in three categories: *data*, *model* and *features* as shown in Figure 4.8.

The presence of these variables prohibit us from simply showing the results of one or two experiments and concluding from this. To ensure the validity of our conclusions, we first investigate the impact these variables have on our results. With this context we then draw proper conclusions by comparing our results to those of the respective reference models we replicated.

Since every variable can potentially influence any of the other variables, we would have to compare every combination of them to achieve the most optimal result. Naturally this is not feasible considering the large amount of variables and relatively long training times. Rather we apply a heuristic to only compare the most interesting combinations. As can be seen in Figure 4.8 we essentially turn the fully connected graph of variables into a tree-like structure. While

we experiment with one of the categories, the variables from the other two categories are kept constant, which is what we represent with the arrows.

For the VDJdb experiments we start by comparing the impact of the different features and their operators. For this total of 24 experiments we only use the *padded* model and only consider the scenario with TRB sequences. These results are then used to determine the best features, which are used for the next set of experiments where we compare the four implemented models. Finally in this branch we take the best model with the best combination of features and see how it performs in some different scenarios (data column).

Similarly for the PPI experiments we compare two models in three scenarios. First we experiment with Pan's PPI dataset, with and without swapping the proteins. Then we look at how these models behave on the Human Interactome (HI), positive only dataset (with swapping).

Unfortunately we were not able to perform equally elaborate experiments for the PPI problem statement due to practical limitations. It was both a challenge to keep our models within practical memory limits and our training times under the reasonable bound of 24 hours (which was imposed by the CalcUA supercomputer).

CHAPTER 5

Results and Discussion

As stated in the experiment plan (Section 4.5), we iterate over three categories of variation. First features, then models and finally datasets. This structure is reflected in the following sections. First we discuss the results concerning the immunological problem statement with the VDJdb dataset in Section 5.1. Then Section 5.2 continues with the PPI experiments.

5.1 VDJdb

All VDJdb experiments were performed with 5-fold cross-validation and a limit of 40 epochs. We decided to use the same amount of epochs for every model and every scenario to facilitate post processing and visualization. The cutoff of 40 epochs was selected because it proved to be sufficient in all experiments. Every metric either stopped improving altogether or plateaued slowly at this hard cutoff, ensuring that the reported results are close to optimal. All metrics were calculated on the validation set with the final model.

5.1.1 Feature Operators

Our first experiments compare the individual features in combination with all four operators. Results were generated with the padded model and with the VDJdb dataset filtered on TRB sequences. There are two goals for these experiments: determine which operator functions best for each feature and identify which features are the most informative.

Compare Operators

For the first goal we inspect the precision-recall and ROC curves for each of the operators per feature. These results can be found in Figures 5.1 and A.1 respectively. The same conclusions can be drawn from both figures. Namely in all of the cases, the absolute difference operator achieves the best results, though only by a small amount.

Our intuition was that the *product* operator would perform better than the other operators for the *charge* and *prime* features. The idea behind *charge* is that opposites attract and that the product operator would help encode this interpretation, since operands with different signs yield a negative result and operands with the same sign always yield a positive result. Additionally molecules without a charge aren't influenced much by charged molecules. This would also nicely be represented by multiplying by zero.

However the diff and abs diff operator also succeeds at encoding this behavior. Values that are further apart result in higher values. The directionality is even removed when taking the absolute value. It is hence not surprising that all four operators achieve similar results.

The prime feature was included to test whether the model would improve by allowing it to discern every possible amino acid combination. We wanted to explore if the performance was "capped" because the features we provided did not give enough information about each of the amino acids. Since there is a lot of overlap in the feature values, it is not always possible to derive which combination of amino acids generated a specific value. By adding an arbitrary encoding to the first 20 prime numbers, we ensure that their product would always be unique, allowing the model to discern all possible combinations. This theory was tested in the next section (5.1.2).

For the other features we expected that the difference operator would perform best because in general we can interpret them as "the closer these values are together, the more likely it is for the amino acids to interact with each other". Again the same information can also be derived with the other operators, so it is not much of a surprise that they all perform almost equally well.

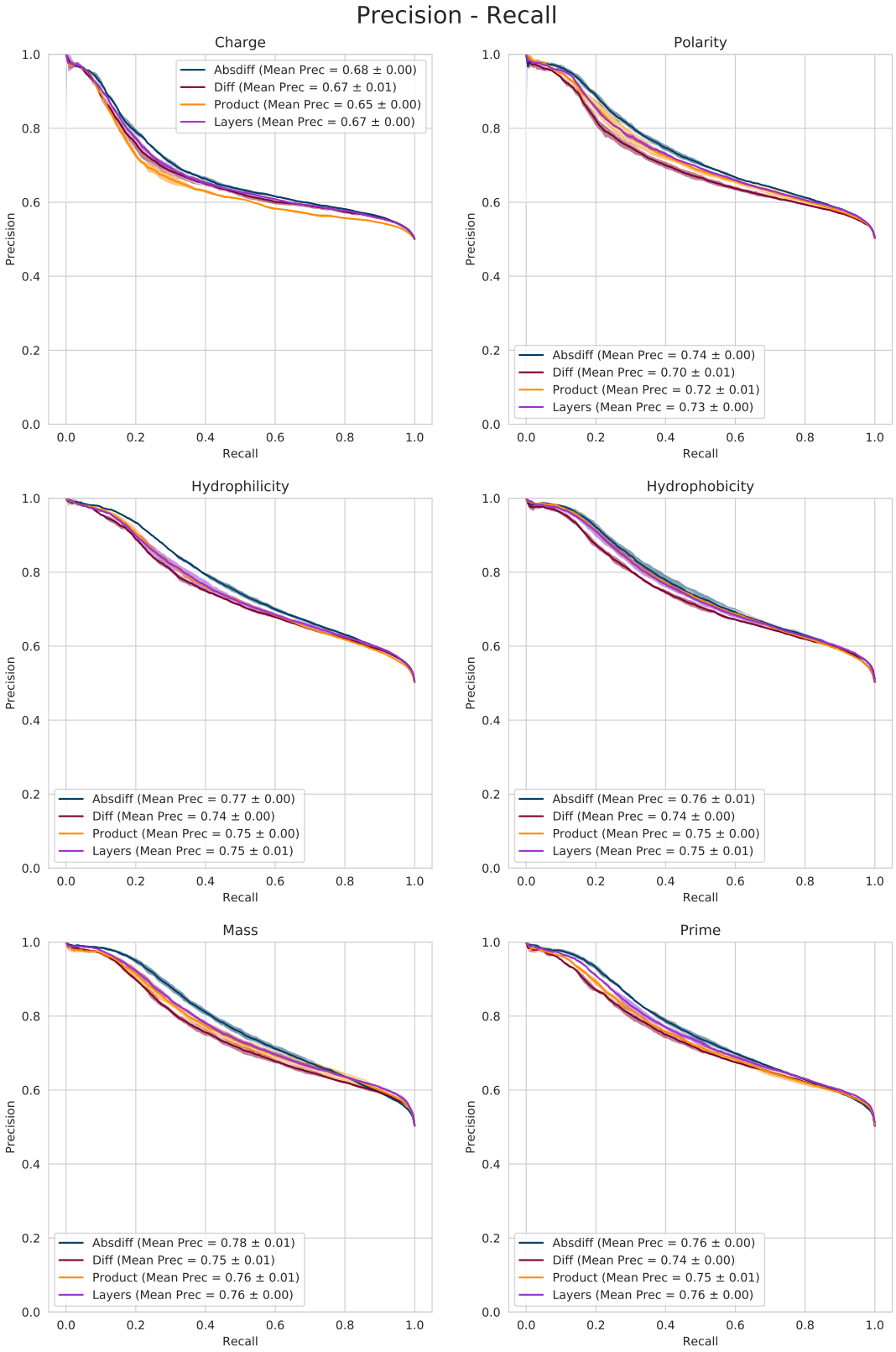


Figure 5.1: Precision-recall curves for padded model in combination with different features and operators. Trained and validated with the VDJdb dataset filtered on TRB sequences. See Figure A.1 for ROC curves.

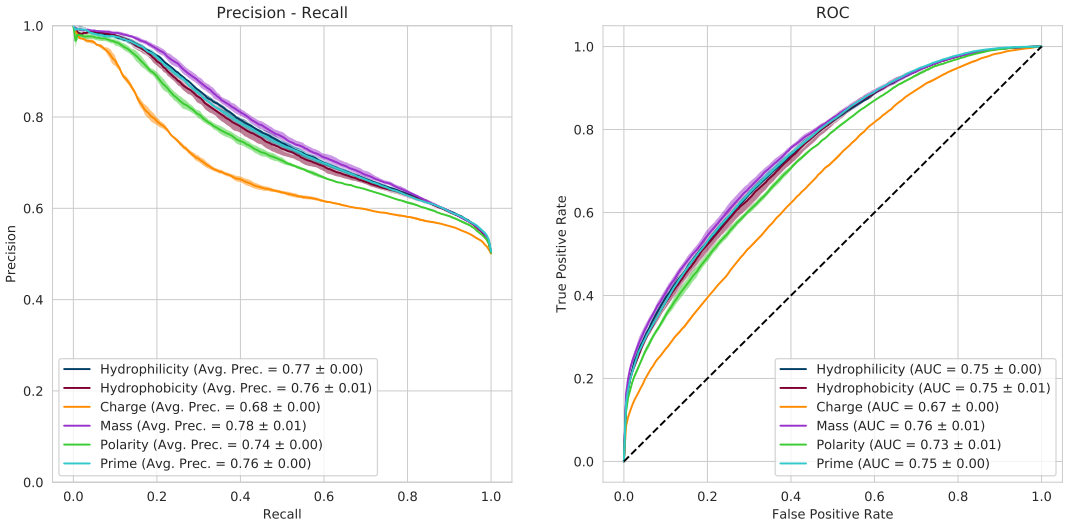


Figure 5.2: Precision-recall and ROC curves for the different features with the abs diff operator. Results originate from the padded model with the VDJdb dataset filtered on TRB sequences.

To conclude we would like to remark that the differences in predictive performance we observe are very small and, within this context, can be considered to be insignificant. These experiments were performed with one model and with one specific dataset. It is hence not unlikely that in a different scenario, the results may also slightly vary, perhaps in favor of a different operator.

We derive from these experiments that the choice of operator has very little impact on the resulting predictive performance of our models. That is why the *abs diff* operator was selected for all features except *prime*. It achieves good result and additionally it appears to converge faster than the other operators (its performance increases faster in the earlier epochs). For the *prime* feature we use the product operator to keep its intended purpose, as explained earlier.

Compare Features

The answer to the second goal, to identify which features are the most informative, can also be derived from these experiments. Figure 5.2 summarizes the previous experiments by including the best performing combination for each of the features. From this figure it is clear that not all features performed equally well. Most scores are close together, except for *charge*, which performs noticeably worse than the others. This is likely because the values associated with *charge* are not very informative. Only values around -1, 0 and 1 are used as can be seen in Figure 2.2.

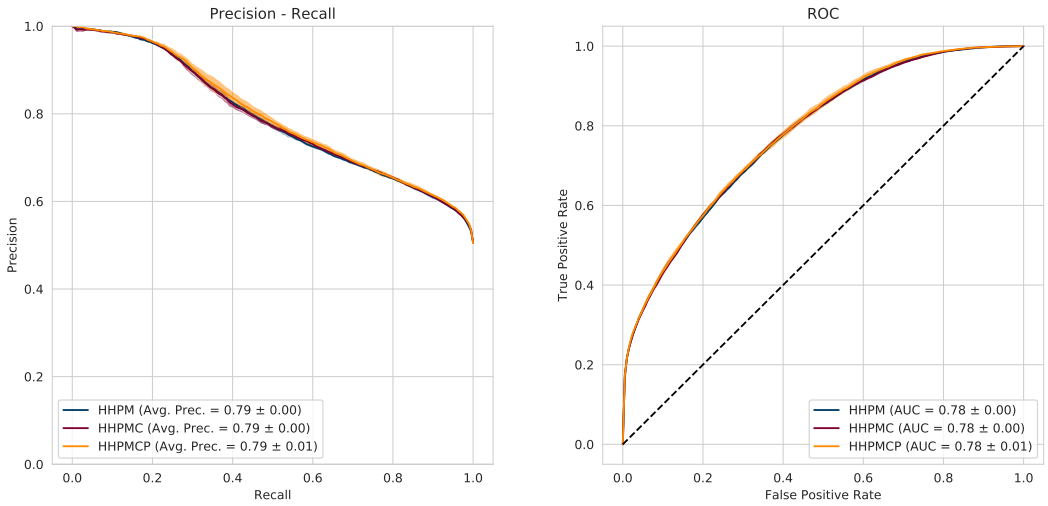


Figure 5.3: Precision-recall and ROC curves for three combinations of features. The letters HHPMCP respectively stand for: ***H****ydrophobicity*, ***H****ydrophilicity*, ***P****olarity*, ***M****ass*, ***C****harge* and ***P****rime*. Results originate from the padded model with the VDJdb dataset filtered on TRB sequences.

However we should not conclude from this that it doesn't matter which features are used in the final model. Though the results may be close together, this does not mean that the same information is learned from each of the features. It is possible that some features excel at identifying certain interactions where other features would miss the interaction entirely. The next section (5.1.2) explores how combinations of features affect the performance.

5.1.2 Features

To determine which combination of features is the most informative, we tested three situations: 1) the best four biological features ***H****ydrophobicity*, ***H****ydrophilicity*, ***P****olarity* and ***M****ass* (HHPM), 2) all five biological features, including ***C****harge* (HHPMC) and finally 3) all features, including ***P****rime* with the product operator (HHPMCP). Again the results were generated with the padded model and with the VDJdb dataset filtered on TRB sequences. Except for *prime*, the abs diff operator was used for feature encoding.

Figure 5.3 shows the precision-recall and ROC curves for these experiments. We can see that there is very little variation between the different scenarios. Compared to the individual features, there is a slight increase in predictive performance when combining them. This increase may seem insignificant, but it is consistent, indicating that there is at least some benefit to combining the features together.

Including the prime feature seems to offer no improvement in performance. Due to this fact and because the feature has no biological meaning, we chose to not include it in any further experiments. It would likely only increase the risk of overfitting. Since there is no apparent reason to limit the amount of features, we opted to use the combination HHPMC for the next experiments.

5.1.3 Models

As explained in Section 4.4, we created three types of models for our experiments. Up until now we have used the *padded* model. Here we compare it with our two other models, one with global average pooling and one using only dense layers. The *NetTCR* model was also replicated to represent the state-of-the-art and act as a reference.

From Figure 5.4 it is clear that the *padded* model does in fact perform best, though only by a small amount. The results are very close together except for the *dense* model. We can see that this model, which does not rely on convolutions, has less generalization power. In theory this architecture is more generic and as such it should be able to learn the same relations as a convolutional neural network. However because it is more generic, it is also more prone to overfitting and in general it also requires more data to learn the correct classification. All these results are in line with what we expected. The metrics we derived from the *NetTCR* model also corresponded to those reported in their paper [15].

In practice the *padded*, *GAP* and *NetTCR* models can be considered to perform equally well. The slight variations in metrics we observe can probably be overcome by tweaking the parameters and overengineering the models to this specific dataset and its features. Without a separate test set to compare these models, the differences are simply too insignificant to draw any conclusions about which one performs best. What we can derive from these experiments is that our technique works at least as good as the state-of-the-art for this specific scenario.

5.1.4 Data

The final aspect that influences our results, is the training data. Where the previous variations could be tweaked freely to improve performance, it is important to keep in mind that changing the training data also fundamentally changes the underlying problem that a classifier is solving. In essence we compare how well our technique is able to solve different problems in this section.

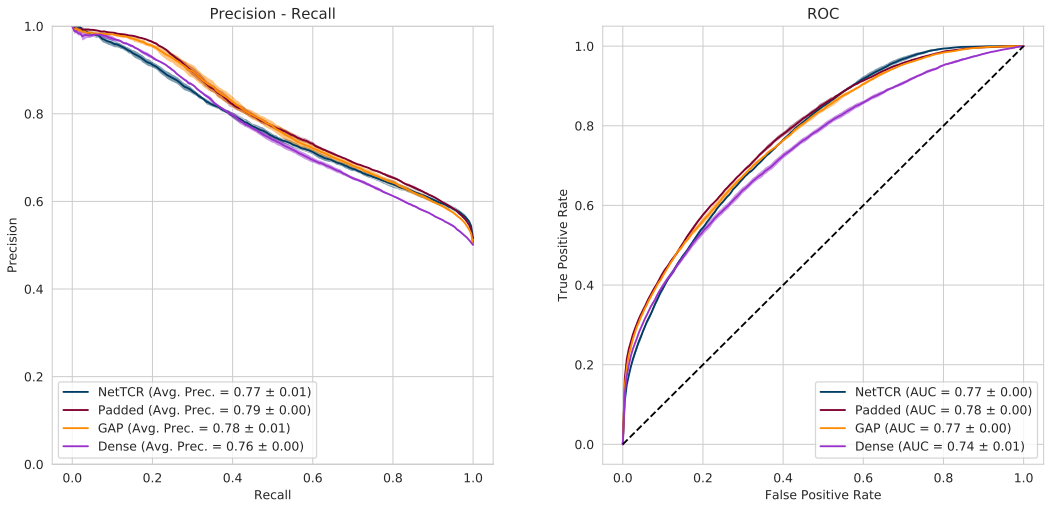


Figure 5.4: Precision-recall and ROC curves for the different models, where NetTCR was used as a reference. Training and validation were performed with the VDJdb dataset filtered on TRB sequences and the combination HHPMC with the abs diff operator was used for feature encoding.

Two different cross-validation methods are considered. First we compare a few interesting subsets of the VDJdb using random sampling (meaning that every fold is a random combination of samples). Then we study the impact of using epitope stratified cross-validation, where all samples of a specific epitope are placed in the same fold.

Random Folds

The previous experiments focused on predicting the binding between an epitope and the β-chain of a T-cell receptor (TRB). In these experiments we also look at the more specific situation where only HLA-A presented epitopes are considered (see 4.2.1 for more information) as well as the more general case where both TRB and TRA sequences are included. We also investigate the impact of using a separate reference dataset of TRB CDR3 sequences to generate the negative samples.

Figure 5.5 shows the results of these experiments. There appears to be a correlation between the amount of samples and the performance. With the smallest subset, filtering on HLA-A genes, we see a decrease in AUC of 9% and in average precision of 8%. The full dataset on the other hand achieves a higher performance than filtering on TRB sequences (about 2% higher AUC and average precision). A logical explanation for this is that the TRA and TRB sequences bind in a similar enough way for the model to benefit from the

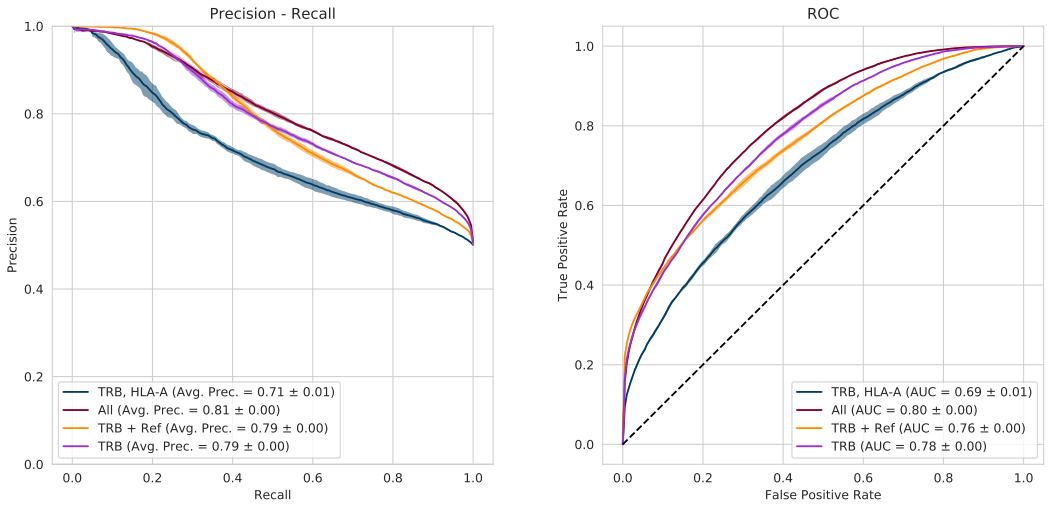


Figure 5.5: Precision-recall and ROC curves for different datasets. Training and validation were performed with the padded model and the combination HHPMC with the abs diff operator was used for feature encoding.

increase in data without being "confused" by the different interactions.

As mentioned earlier however, one should be careful when interpreting these results, since in essence we are comparing how effective our technique is at solving different problem statements. The validation datasets are different for each of these scenarios, meaning that the reported results are only relevant within the context of the data. For example another possible explanation would be that TRA sequences are much simpler to classify than TRB sequences. Then by adding them to the training and validation sets, it would also be possible to achieve overall higher performances when in fact, the model would be no better at classifying TRB sequences. We know however that this is not the case because TRB and TRA sequences are highly similar and about equally difficult to classify with this setup.

In the case of using reference TRB CDR3 sequences we find that the results are quite similar to the normal case. There is however an interesting difference in that the respective precision-recall curves cross each other somewhere around a recall of 50%. To investigate this further we included the respective prediction histograms in Figure 5.6. There it becomes clear why their average precision scores are the same, but the AUC score is lower for the reference sequences. It appears that the normal model "favors" positive samples, classifying more of them correctly, whereas the model with reference sequences gets more negative samples right. Since a ROC curve only takes positive samples into account, it is to be expected that the TRB model has a higher AUC value.

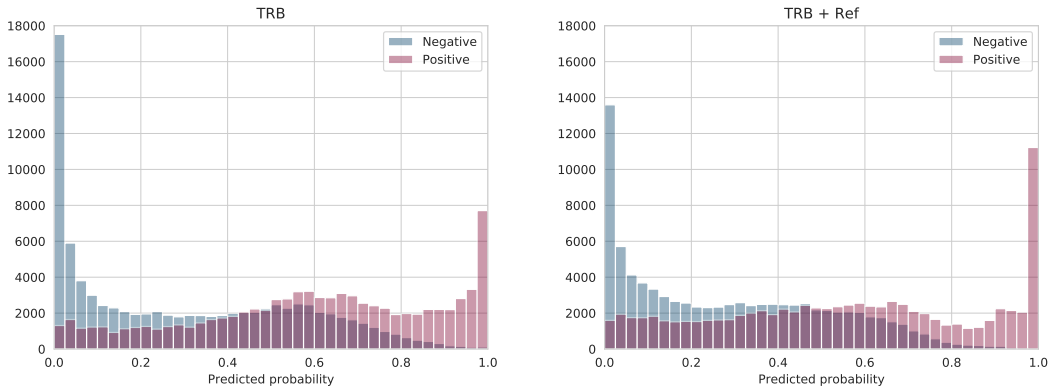


Figure 5.6: Prediction histograms for two experiments with VDJdb filtered on TRB sequences. On the left without using reference sequences and on the right with reference TRB CDR3 sequences. Training and validation were performed with the padded model and the combination HHPMC with the abs diff operator was used for feature encoding.

A possible explanation for this phenomenon is that the model with reference sequences is also trained on a more diverse set of TCR sequences. It is hence not unlikely that it is also better at classifying the negative samples in the validation set correctly, since it came into contact with a wider and more diverse range of examples in the training set than the other model, which only used shuffling.

Epitope Stratified Folds

All experiments up until now were cross-validated with random sampling. Intuitively this can best be interpreted as "predicting binding for known epitopes" because most of the epitopes we used in the validation step, were also seen during training. To evaluate a truly generic model that can predict the binding between any TCR and epitope, seen or unseen, we have to use epitope stratified cross-validation. This ensures that during validation the model is only tested on previously unseen epitopes. Where random sampling gives an overestimate of the performance, this strategy reports an underestimate of the actual performance since in reality, the impact of known epitopes should also be taken into account.

These experiments were performed with the VDJdb dataset filtered on TRB sequences. Contrary to the other experiments, here only 4-fold cross-validation was used. This is because with a higher number of folds, there would either be unbalanced folds or ones that only contained one epitope, which would make it impossible to generate negative samples. Despite the unfortunate discrepancy

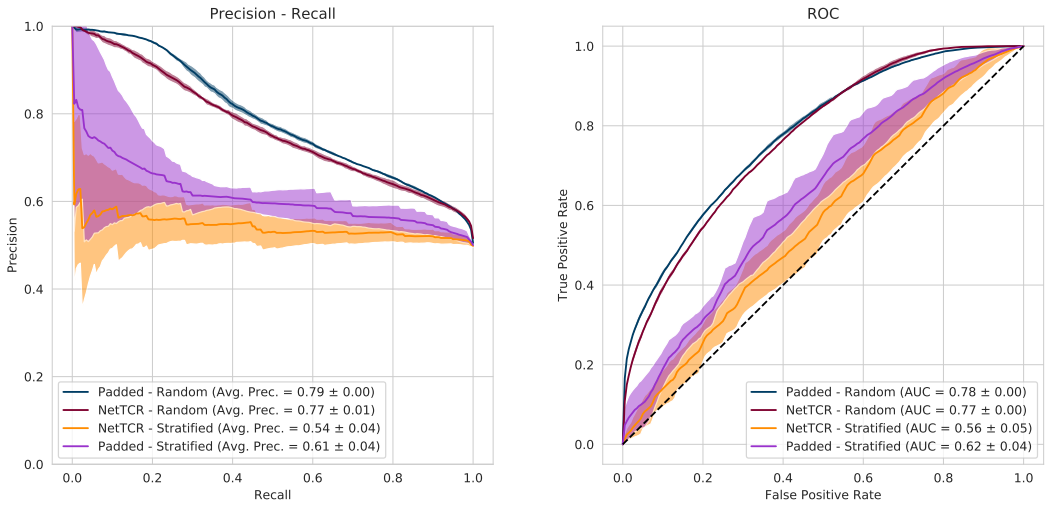


Figure 5.7: Precision-recall and ROC curves for epitope stratified cross-validation. Training and validation were performed on the VDJdb dataset filtered on TRB sequences with the padded model. The combination HHPMC with the abs diff operator was used for feature encoding. The results of the previous experiments with random sampling were also included in this figure to facilitate comparison.

with our other experiments, four folds still suffices to provide consistent and representative results.

With random cross-validation we could not discern much difference between our best model (*padded*) and the state-of-the-art (*NetTCR*). In this scenario however the results do show a larger difference. As can be seen in Figure 5.7 there is a difference in both average precision and AUC of about 6%. It is clear that overall the predictive performance in the epitope stratified case is lower, which is intuitive. There is also a lot more variability in the results, especially for thresholds with a low recall. This would probably improve if more data were available and if more folds could be used.

A possible explanation for these results is that our model succeeds better at generalizing the binding relation because it combines the features at an earlier point. The *NetTCR* model focuses heavily on feature extraction for the two inputs independently before combining them, which only happens at the final few layers. Our intuition is that, because of this, the model is limited to learning the "binding" relation on a global level, rather than also on the underlying levels.

We conclude from this range of experiments with the VDJdb dataset that our feature encoding technique (in combination with the *padded* or *GAP* model)

succeeds at accurately predicting the binding between TCRs and epitopes. We learned that it is important to select informative features with a logical operator and that the choice of operator can help guide the training process. Furthermore it appears that higher predictive performances can be achieved for this problem statement if more data were available. Finally our technique also seems to have a better generalization capacity than the state-of-the-art *NetTCR* model.

5.2 PPI

The second branch of our experiment plan pertains to protein-protein interactions. Unfortunately we were not able to perform equally elaborate experiments for this use case due to practical limitations. It was both a challenge to keep our neural network models within practical memory limits and the training times under 24 hours. That is why these experiments were only cross-validated with three folds. Limiting training to 40 epochs may also have been too restrictive for this use case. Our results are hence a bit more preliminary and should be interpreted more carefully.

First we explain our experiments with Pan's PPI dataset, which contained negative samples. The impact of our practical tradeoffs is discussed as well as the impact of swapping the two inputs. Then we compare these results to those of the Human Interactome (HI) PPI dataset which only consisted of positive samples. Here negative samples were generated by recombining the proteins randomly and filtering out positive matches.

5.2.1 Pan's PPI Dataset

Figure 5.8 shows the results of five experiments conducted with the second PPI dataset. In green and blue we have our model with and without swapping respectively. These models were trained on proteins with a maximum length of 600 amino acids and the mass and polarity features were used for feature encoding. Again due to practical limitations we could not use more than two features, so we selected these two arbitrarily. Then we also have two instances of the DNN-PPI model in red and orange with and without swapping respectively. The same effect can be seen, namely that swapping the inputs, and hence making the model more general, reduces the overall predictive performance. However the effect seems smaller for the DNN-PPI model. Lastly we

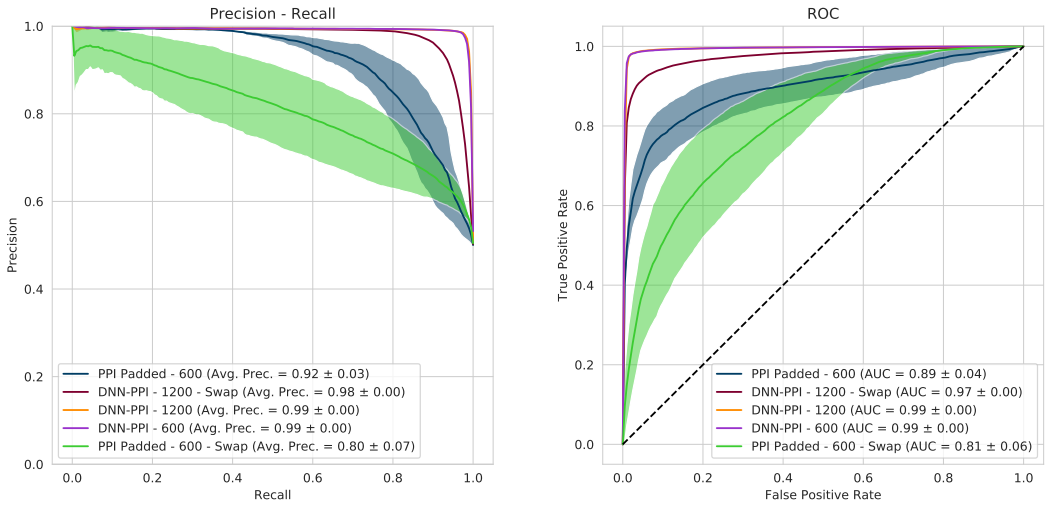


Figure 5.8: Precision-recall and ROC curves for Pan's PPI dataset. The *PPI Padded* model used a maximum length of 600 and features were encoded with the mass and polarity features.

also trained the DNN-PPI model with a maximum length of 600 amino acids to investigate whether this limitation would impact the results. It is clear from the figure that this is not the case. The purple and orange models score equally well.

From these experiments we can derive that the DNN-PPI model is clearly better at classifying Pan's PPI dataset. These results also correspond to what was reported in the paper describing DNN-PPI [20]. It is highly unlikely that our model will ever reach the same performance, even with more features and more training time. This is also logical because our technique is inherently based on protein binding on a physicochemical level. It is infeasible to accurately predict binding in general for sequences with a length of 600 amino acids or longer. Most other techniques are based on finding motifs (short, common stretches of amino acids in a protein) to help with prediction [20]. However our technique should be better at generalizing what it means for proteins to bind, exactly because it does not rely on motifs.

5.2.2 Human Interactome (HI) PPI Dataset

In the next set of experiments, where the HI PPI dataset is used, we notice a different trend. As can be seen in Figure 5.9, the DNN-PPI model is not able to learn anything form this setup. We think this difference might be caused by a bias in Pan's PPI dataset, which evidently was also used in the paper that described the DNN-PPI model [20]. However this is just an intuition. It

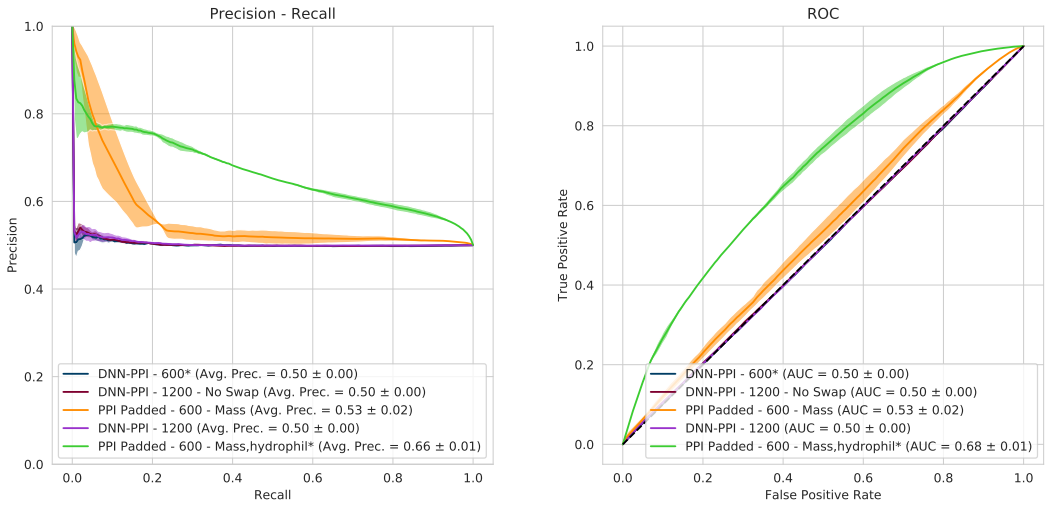


Figure 5.9: Precision-recall and ROC curves for HI PPI dataset. The *PPI Padded* model used a maximum length of 600 and features were encoded either only with the mass feature or with the mass and hydrophilicity features. In the experiments marked with an asterisk, a subset of the data was used. All experiments used random swapping of the inputs, except the DNN-PPI model labeled with "No Swap".

is most likely not an implementation error since a variation on this model was in fact able to learn something from the VDJdb dataset that was processed in the same way, making the dataset the only variable that could explain these results.

We consider this experimental setup to be more representative because it does not allow any biases. Since there are no negative samples to be learned by heart, a model is forced to learn the actual intended relation from the dataset. Furthermore this is also an independent dataset that we did not engineer ourselves.

From the two experiments with our model it is clear that it is able to generalize the protein-protein interaction to some extent. Remarkably enough the predictive performance increases drastically by adding a second feature. In the first situation, only the mass feature was used. Then we added hydrophilicity because it also has a high performance and is among the least correlated features with mass (see Figures 5.2 and 2.2). However because this dataset is larger, it was no longer able to run within 24 hours. To circumvent this issue we trained and validated this model on a subset of 40,000 samples out of the 77,214. The DNN-PPI model was also trained on this subset to verify that it did not give our model an unfair advantage, which does not seem to be the case.

To conclude our PPI experiments, we remark that both models have their advantages. Depending on the use case, classification based on the primary sequences of proteins might suffice. In other scenarios the generalization that can be achieved by working with physicochemical properties may be required, at the cost of lower accuracies. Our technique does however remain impractical in this scenario because it scales quadratically with respect to the input size.

CHAPTER 6

Threats to Validity

It is important to consider the scope and the limitations of our work. Therefore we look at possible threats to validity and subdivide them into four categories. *Internal Validity* pertains to the validity of the study itself. Validity of generalization is categorized in *External Validity*. *Construct Validity* ensures that inferences made from observations are valid. Finally *Conclusion Validity* discusses whether the right conclusions are drawn with enough evidence.

Internal Validity

- The algorithms developed in this study may have contained errors, leading to false results. Although we can never be sure that there are no bugs in a software system, it is possible to verify the desired behavior. We carefully tested the components of our algorithms manually to verify whether the results were as expected. Furthermore it is clear that the achieved results lie within the same bounds as those of state-of-the-art methods. No inexplicably high or low values were encountered.
- Due to the biological nature of our datasets, there is an inherent risk that they contain errors. These could either originate from sequencing the amino acids or by recording false negatives or false positives. This is of course unavoidable and there is nothing we can do prevent or circumvent this. However we estimate these errors to only have a very small impact on our results, safeguarding our conclusions which were drawn with a large enough margin.

- Biases in the dataset may have skewed the results of our experiments. It is true that overrepresentation of specific pathogens in the data (cfr. Figure 4.2) can bias a classifier to "focus" more on those samples and hence report a higher performance than what is actually achieved independent of the epitope. To counteract this bias and overestimation, we also performed epitope stratified experiments where overrepresentation actually had a negative impact on the results.

External Validity

As mentioned earlier, to truly compare different models after the tuning of hyperparameters, one would have to use an independent test set to evaluate performances. Using the validation set, which was also used during the training of the models, can potentially introduce a bias due to the possibility of overengineering. However, by using K-folds cross-validation, we already mitigated this risk to some extent. Moreover conclusions were only drawn if the differences were large enough, making it implausible that they were caused by a potential bias.

The way negative samples were selected in our experiments forms another potential threat to the generalizability of our results. Although random shuffling to create negative samples is a commonly used technique as described by Park et al. [28], we should note that it is also very susceptible to artificial biases being introduced that could lead to either over- or underestimates of the actual predictive performance of our classifiers. By using truly random sampling, we were able to minimized this risk to the best of our knowledge.

Similarly we also used random sampling to "simulate" iterating over the dataset. This of course leads to a less strict definition of an epoch, since it is possible that some samples are selected multiple times while others may not come up at all in some epoch. Over the course of many epochs however, the law of large numbers dictates that the sampled distribution should approach the expected distribution, which mitigates the impact of this random behavior.

Construct Validity

Artificial neural networks are notorious for being black boxes in the sense that it is hard to understand what they learn and how predictions are formed (unlike with decision trees for example). It is hence also difficult to make inferences from their results. In this study we only stated clear facts and formulated inferences as intuitions and "likely explanations". Indeed, the inferences made from our experiments should be interpreted as preliminary until further studies confirm our findings.

Naturally this is a more general problem that is also inherent to many other studies that perform machine learning experiments. As such, various solutions or workarounds have also been proposed in the literature. One example is the LIME tool[1], which is based on the work presented by Ribeiro et al [29]. It tries to explain how machine learning classifiers achieve a prediction by learning an interpretable model locally around the predicted sample.

Conclusion Validity

Though the exact explanation for our positive results may still be unclear and uncertain, we are still able to draw conclusion from them if our experimental setup is valid. Again by using K-folds cross-validation and by exploring many different scenarios, we minimized this risk.

[1] `https://github.com/marcotcr/lime`

CHAPTER 7

Conclusions

To conclude we first summarize our work, then future work is listed and finally we state our conclusions.

7.1 Summary

Accurately predicting TCR to epitope binding remains an open problem in bioinformatics. Reliable data has only recently become available by innovations in sequencing techniques, allowing for the introduction of publicly available classifiers like NetTCR [15] and TCRex [9].

We introduced a new technique for feature encoding that relies on pairwise comparisons of physicochemical amino acid properties. This technique can not only be used to encode TCR and epitope sequences, but also any pair of amino acid sequences in general. To study the impact and effectiveness of this encoding, we conducted a range of experiments with the VDJdb dataset on the one hand and with two distinct protein-protein interaction datasets on the other hand.

Both scenarios were compared with similar state-of-the-art methods. From our experiments with NetTCR it became clear that a higher predictive performance

could be reached in the case of epitope stratified cross-validation. Similarly when comparing our technique with DNN-PPI, an artificial neural network classifier for predicting PPIs, we found that our method achieved better results in a more constrained setting where generalization plays a key role. However other techniques may prove more practical for predicting protein-protein interactions due to the relatively long sequences and quadratic scalability of our approach.

7.2 Future Work

From a biological perspective, we known that both the alpha and beta chain of a T-cell receptor (TRA and TRB) need to bind to an epitope in order to trigger an immune response. However modern sequencing techniques are not yet able to capture this relation. That is, we do not known which TRA sequence belongs to which TRB sequence. Having this information available can potentially greatly increase the predictive performance of classifiers, since at this point, it is like we are missing a key part of the puzzle.

Similarly we can expect the prediction of "triggering an immune response" to be more accurate than simply predicting the binding part, since there are often multiple TCRs that bind to the same epitope. This natural redundancy can be exploited by an immune response classifier that combines the output of our model to give a more accurate prediction about whether at least one TCR can bind an epitope.

In the case of protein-protein interactions it would be interesting to see whether using a siamese network for the DNN-PPI model could perform better on a dataset where shuffling is used to generate negative samples. This should also help make the classifier more robust with respect to swapping the inputs. A followup study could also test our hypothesis that the model has trouble generalizing because the separate inputs are only concatenated relatively late.

We have limited ourselves to only using features per amino acid and combining them. There are however also features that are defined on combinations of amino acids, like the instability index by Guruprasad et al. [12] or the BLOSUM50 matrix for example. Furthermore most features can also be calculated on longer stretches of amino acids. The impact of using a sliding window of size two or three over the sequences to calculate its values could also be investigated.

7.3 Conclusions

Two separate use cases were studied to assess the effectiveness of our feature encoding technique. From both sets of experiments we learned that our classifiers, based on pairwise interaction matrices, performed better in scenarios where generalization played a key role. In the first case by using epitope stratified cross-validation and in the second case by using shuffling to generate negative samples. We conclude from these experiments that a feature encoding technique that is solely based on the pairwise comparisons between amino acids, likely has a better generalization capacity than techniques that rely on separate inputs and a feature vector per sequence.

This success can most likely be attributed to three factors. Firstly, the classifier is only given information about the combination of two sequences, which mostly prevents it from learning sequences by heart and thus overfitting. Secondly since the sequences are inherently joined from the start, the model is able to learn the actual "binding" relation starting from a low level of abstraction (i.e. from the first convolutional layers). Thirdly by using features with actual biological significance, rather than random numbers or prime encodings, we also incorporate the knowledge that some amino acids are more similar to each other than others into the encoding, which can be of major importance for predicting binding.

Bibliography

[1] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. Molecular biology or the cell. In *Molecular biology or the cell*. 2002.

[2] Massimo Andreatta and Morten Nielsen. Gapped sequence alignment using artificial neural networks: application to the MHC class I system. *Bioinformatics*, 32(4):511–517, 2015.

[3] John N Aronson. The Henderson-Hasselbalch equation revisited. *Biochemical Education*, 11(2):68–68, 1983.

[4] Bengt Bjellqvist, Bodil Basse, Eydfinnur Olsen, and Julio E Celis. Reference points for comparisons of two-dimensional maps of proteins from different human cell types defined in a pH scale where isoelectric points correlate with polypeptide compositions. *Electrophoresis*, 15(1):529–539, 1994.

[5] Bengt Bjellqvist, Graham J Hughes, Christian Pasquali, Nicole Paquet, Florence Ravier, Jean-Charles Sanchez, Séverine Frutiger, and Denis Hochstrasser. The focusing positions of polypeptides in immobilized pH gradients can be predicted from their amino acid sequences. *Electrophoresis*, 14(1):1023–1031, 1993.

[6] Pradyot Dash, Andrew J Fiore-Gartland, Tomer Hertz, George C Wang, Shalini Sharma, Aisha Souquette, Jeremy Chase Crawford, E Bridie Clemens, Thi HO Nguyen, Katherine Kedzierska, et al. Quantifiable predictive features define epitope-specific T cell receptor repertoires. *Nature*, 547(7661):89, 2017.

[7] Nicolas De Neuter, Wout Bittremieux, Charlie Beirnaert, Bart Cuypers, Aida Mrzic, Pieter Moris, Arvid Suls, Viggo Van Tendeloo, Benson Ogunjimi, Kris Laukens, et al. On the feasibility of mining CD8+ T cell receptor patterns underlying immunogenic peptide recognition. *Immunogenetics*, 70(3):159–168, 2018.

[8] Jared Dean, Ryan O Emerson, Marissa Vignali, Anna M Sherwood, Mark J Rieder, Christopher S Carlson, and Harlan S Robins. Annotation of pseudogenic gene segments by massively parallel sequencing of rearranged lymphocyte receptor loci. *Genome medicine*, 7(1):123, 2015.

[9] Sofie Gielis, Pieter Moris, Nicolas De Neuter, Wout Bittremieux, Benson Ogunjimi, Kris Laukens, and Pieter Meysman. TCRex: a webtool for the prediction of T-cell receptor sequence epitope specificity. *bioRxiv*, page 373472, 2018.

[10] Jacob Glanville, Huang Huang, Allison Nau, Olivia Hatton, Lisa E Wagar, Florian Rubelt, Xuhuai Ji, Arnold Han, Sheri M Krams, Christina Pettus, et al. Identifying specificity groups in the T cell receptor repertoire. *Nature*, 547(7661):94, 2017.

[11] Anton A Goloborodko, Lev I Levitsky, Mark V Ivanov, and Mikhail V Gorshkov. Pyteomics-a Python framework for exploratory data analysis and rapid software prototyping in proteomics. *Journal of The American Society for Mass Spectrometry*, 24(2):301–304, 2013.

[12] Kunchur Guruprasad, BV Bhasker Reddy, and Madhusudan W Pandit. Correlation between stability of a protein and its dipeptide composition: a novel approach for predicting in vivo stability of a protein from its primary sequence. *Protein Engineering, Design and Selection*, 4(2):155–161, 1990.

[13] Thomas P Hopp and Kenneth R Woods. Prediction of protein antigenic determinants from amino acid sequences. *Proceedings of the National Academy of Sciences*, 78(6):3824–3828, 1981.

[14] Vanessa Jurtz, Sinu Paul, Massimo Andreatta, Paolo Marcatili, Bjoern Peters, and Morten Nielsen. NetMHCpan-4.0: Improved peptide–MHC class I interaction predictions integrating eluted ligand and peptide binding affinity data. *The Journal of Immunology*, 199(9):3360–3368, 2017.

[15] Vanessa Isabell Jurtz, Leon Eyrich Jessen, Amalie Kai Bentzen, Martin Closter Jespersen, Swapnil Mahajan, Randi Vita, Kamilla Kjærgaard Jensen, Paolo Marcatili, Sine Reker Hadrup, Bjoern Peters, et al. NetTCR: sequence-based prediction of TCR binding to peptide-MHC

complexes using convolutional neural networks. *bioRxiv*, page 433706, 2018.

[16] Edita Karosiene, Claus Lundegaard, Ole Lund, and Morten Nielsen. NetMHCcons: a consensus method for the major histocompatibility complex class I predictions. *Immunogenetics*, 64(3):177–186, 2012.

[17] Jack Kyte and Russell F Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of molecular biology*, 157(1):105–132, 1982.

[18] Albert L Lehninger, David L Nelson, Michael M Cox, Michael M Cox, et al. *Lehninger principles of biochemistry*. Macmillan, 2005.

[19] Lev I Levitsky, Joshua A Klein, Mark V Ivanov, and Mikhail V Gorshkov. Pyteomics 4.0: five years of development of a Python proteomics framework. *Journal of proteome research*, 18(2):709–714, 2018.

[20] Hang Li, Xiu-Jun Gong, Hua Yu, and Chang Zhou. Deep neural network based predictions of protein interactions using primary sequences. *Molecules*, 23(8):1923, 2018.

[21] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[22] Zhonghao Liu, Yuxin Cui, Zheng Xiong, Alierza Nasiri, Ansi Zhang, and Jianjun Hu. DeepSeqPan, a novel deep convolutional neural network model for pan-specific class I HLA-peptide binding affinity prediction. *Scientific reports*, 9(1):794, 2019.

[23] Dexter S Moore. Amino acid and peptide net charges: A simple calculational procedure. *Biochemical Education*, 13(1):10–11, 1985.

[24] Morten Nielsen and Massimo Andreatta. NetMHCpan-3.0; improved prediction of binding to MHC class I molecules integrating information from multiple receptor and peptide length datasets. *Genome medicine*, 8(1):33, 2016.

[25] Timothy J O'Donnell, Alex Rubinsteyn, Maria Bonsack, Angelika B Riemer, Uri Laserson, and Jeff Hammerbacher. MHCflurry: open-source class I MHC binding affinity prediction. *Cell systems*, 7(1):129–132, 2018.

[26] Xiao-Yong Pan, Ya-Nan Zhang, and Hong-Bin Shen. Large-Scale prediction of human protein- protein interactions from amino acid sequence based on latent topic features. *Journal of Proteome Research*, 9(10):4992–5001, 2010.

[27] Luca Parca, Bruno Ariano, Andrea Cabibbo, Marco Paoletti, Annalaura Tamburrini, Antonio Palmeri, Gabriele Ausiello, and Manuela Helmer-Citterich. Kinome-wide identification of phosphorylation networks in eukaryotic proteomes. *Bioinformatics*, 35(3):372–379, 2018.

[28] Yungki Park and Edward M Marcotte. Revisiting the negative example sampling problem for predicting protein–protein interactions. *Bioinformatics*, 27(21):3024–3028, 2011.

[29] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[30] Mikhail Shugay, Dmitriy V Bagaev, Ivan V Zvyagin, Renske M Vroomans, Jeremy Chase Crawford, Garry Dolton, Ekaterina A Komech, Anastasiya L Sycheva, Anna E Koneva, Evgeniy S Egorov, et al. VDJdb: a curated database of T-cell receptor sequences with known antigen specificity. *Nucleic acids research*, 46(D1):D419–D427, 2017.

[31] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[32] Duolin Wang, Shuai Zeng, Chunhui Xu, Wangren Qiu, Yanchun Liang, Trupti Joshi, and Dong Xu. MusiteDeep: a deep-learning framework for general and kinase-specific phosphorylation site prediction. *Bioinformatics*, 33(24):3909–3916, 2017.

[33] Javad Zahiri, Joseph Hannon Bozorgmehr, and Ali Masoudi-Nejad. Computational prediction of protein–protein interaction networks: algorithms and resources. *Current genomics*, 14(6):397–414, 2013.

Appendices

APPENDIX A

Additional Tables and Figures

Table A.1: Features of amino acids. Values are rounded to up to five decimal places.

Amino acid	**Charge**	**Hydro-phobicity**	**Polarity**	**Mass (Da)**	**Hydro-philicity**
A	-0.00202	1.8	5.57001	71.0779	-0.5
C	-0.06399	2.5	5.51813	103.1429	-1
D	-1.00157	-3.5	4.29938	115.0874	3
E	-1.00024	-3.5	4.59930	129.1140	3
F	-0.00202	2.8	5.52496	147.1738	-2.5
G	-0.00202	-0.4	5.52496	57.0513	0
H	0.08889	-3.2	6.74115	137.1393	-0.5
I	-0.00202	4.5	5.52496	113.1576	-1.8
K	0.99769	-3.9	8.75006	128.1723	3
L	-0.00202	3.8	5.52496	113.1576	-1.8
M	-0.00202	1.9	5.27496	131.1960	-1.3
N	-0.00202	-3.5	5.52496	114.1026	0.2
P	-0.00202	-1.6	5.95502	97.1152	0
Q	-0.00202	-3.5	5.52496	128.1292	0.2
R	0.99798	-4.5	9.75006	156.1857	3
S	-0.00202	-0.8	5.24005	87.0773	0.3
T	-0.00202	-0.7	5.18500	101.1039	-0.4
V	-0.00202	4.2	5.49506	99.1310	-1.5
W	-0.00202	-0.9	5.52496	186.2099	-3.4
Y	-0.00287	-1.3	5.52435	163.1732	-2.3

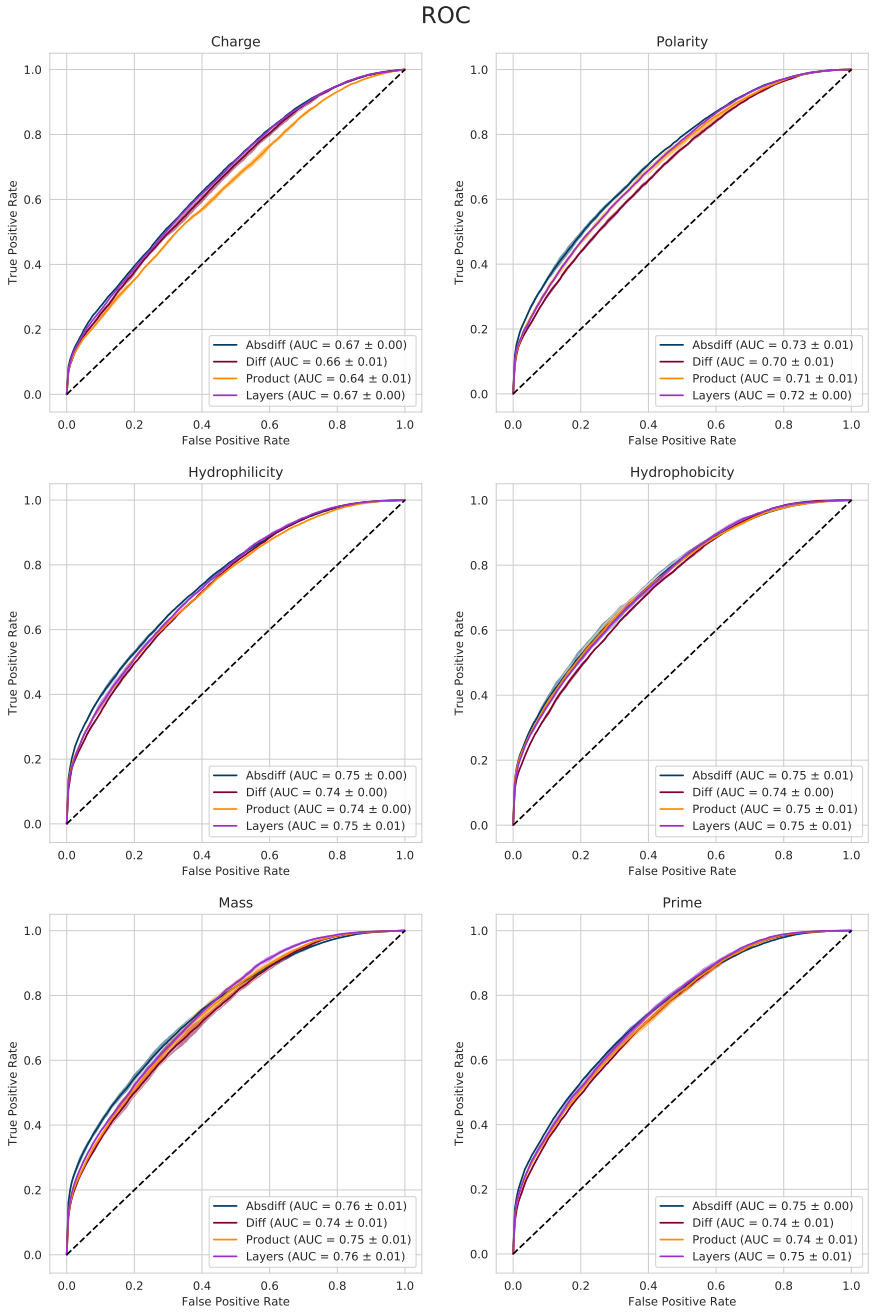


Figure A.1: ROC curves for padded model in combination with different features and operators. Training and validation performed with the VDJdb dataset filtered on TRB sequences. See Section 5.1.1 for interpretation.