Classification of TCRβ Repertoire between CD4$^+$ and CD8$^+$ T-Cell Populations

Lewis Iain Moffat

Primary Supervisor: Prof. Benny Chain
Secondary Supervisor: Prof. John Shawe-Taylor

A dissertation submitted in partial fulfillment

of the requirements for the degree of

MSc Machine Learning

of

University College London.

Department of Computer Science

University College London

September 3, 2017

This report is submitted as part requirement for the MSc Degree in Machine Learning at University College London. It is substantially the result of my own work, except where explicitly indicated in the text.

Code for this report is available at: `https://github.com/innate2adaptive/tcRIP`

Abstract

T Cells recognize antigens using a highly diverse repertoire of antigen specific T Cell Receptors (TCRs). The recent advent of high throughput sequencing has made it possible to gather hundreds of thousands of these TCR sequences, opening up the repertoire to computational analysis. One of the questions yet to be answered is whether the two major types of T Cells, $CD4^+$ and $CD8^+$, show distinct differences in their TCR CDR loops, suggesting a functional difference in how they recognize antigens bound to MHC Class II and Class I respectively.

This work aims to investigate the potential difference between populations by classifying TCRβ chain CDR3s using a variety of feature engineering methods and machine learning classification algorithms. The feature engineering methods focus on extracting local sequencing based features such a small motif usage and amino acid positional probability. A range of classification algorithms were tested, including k-Nearest Neighbors and Recurrent Neural Networks.

The best classification and feature engineering combination did not achieve a high performance, with a test accuracy of 67%. This combination used an XGBoost classifier on the combined CDR1β, CDR2β, CDR3β, and V Gene for each TCR. The amino acids in the sequences were replaced with their five dimensional Atchley vectors. Although it did not achieve a high performance, this approach provides insights into machine learning with TCR chains, and suggests that biologically there is a distinct difference between populations in V gene usage and the CDR3β interaction with bound antigens.

Acknowledgements

Firstly, I would like to thank my primary supervisor Professor Benny Chain from the UCL Division of Infection & Immunity. I am honoured that he accepted me to take on this project, as well as extremely grateful for the insight and assistance he gave throughout. I would also like to greatly thank my secondary supervisor Professor John Shawe-Taylor, Head of the Department of Computer Science UCL, who provided invaluable advice and guidance in completing this project.

Furthermore I wish to greatly acknowledge Professor David T. Jones, UCL department of Computer Science, Bioinformatics group. I also wish to acknowledge Dr. Mark Herbster, UCL Department of Computer Science. Both gave extremely useful advice for different aspects of this project.

I would like to give special thanks to Mazlina Ismail who provided me with help and guidance throughout, particularly in getting started. Finally I would like to thank all the other UCL Academics and staff, without whom I never would have had this wonderful opportunity.

I would like to dedicate this dissertation to my grandmother, Pamela Moffat, who passed away while I worked on this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview of the problem approached in this thesis, including the motivation and primary objectives. It also provides a brief discussion of the field and relevant work. Finally an outline of the structure of this report is given.

1.1 Motivation

The immune system is a host organism's defense system against disease. Vertebrates, including Humans, exhibit a complex immune system that can be split into two broad categories. Innate Immunity provides the generic response to pathogens seeking to cause disease while Adaptive Immunity adds memory and hence provides a powerful second line pathogen specific response. Having an intricate understanding of the adaptive immune system will not only improve our understanding of disease, but also provide the opportunity to develop novel immunotherapeutics [9].

Within adaptive immunity T Cells perform a very important role by carrying out the cell-mediated aspects of adaptive immunity. One of the keys to their functioning are the T Cell Receptors (TCRs). Each receptor acts to recognize a specific antigen that could be presented to the cell. Mature T Cells exhibiting TCRs come from two distinct lineages, those that display the CD4 glycoprotein or CD8 glycoprotein. Both proteins work in tandem with a TCR complex to bind to an antigen and the Major Histocompatibility Complex

(MHC) in which the antigen is being presented.

Due to the advent of High Throughput Sequencing (HTS), large numbers of TCR sequences have become available for computation analysis. Naturally this has led to the use of statistical and machine learning based methods to analyze the massive and highly diverse TCR repertoire [10]. More specifically, a significant amount of research has focused on the Complementary Determining Region 3 (CDR3) of the β chain, as it is the primary sub-sequence in the TCR that dictates the antigen binding specificity.

A relatively recent topic in this research is how the CDR3 varies between CD4 and CD8 T Cell populations. So far very little work has been published that directly asks this question. To our knowledge only Li et al. [11], and our own work [12].

Li et al. showed a distinct difference in amino acid usage and richness of CDR3 sequences between the two populations, and part of their work was to use a machine learning classifier that learned to classify a sequence as originating from either CD4 or CD8 (with $> 80\%$ accuracy). As such the goal of this work is to repeat the same classification task to validate the conclusions of Li et al., and if possible improve upon them, with our own data.

1.2 Objectives

The primary goal of this work is to reaffirm and validate the conclusions drawn by Li et al. on our own data set. Namely, can a machine learning classifier be trained that can accurately assign a CDR3 as belonging to a CD4 T Cell or a CD8 T Cell?

This naturally breaks down into two sub-goals: designing an effective feature engineering method and choosing an appropriate classification algorithm. The feature engineering task consists of first emulating the Li et al. method and then experimenting with a variety of other custom and published techniques to find the optimal approach.

For each of these methods a variety of classification algorithms are tested,

ranging from k-Nearest Neighbours to Recurrent Neural Networks. All with the aim of finding the algorithm best suited to this task.

1.3 Structure of the Thesis

The core of this work begins in Chapter 2 where we first present a brief discussion of Adaptive Immunity, T Cells, and computational analysis of TCRs. This continues on to discuss Protein Classification more generally, which includes feature engineering methods and machine learning classification algorithms, with particular focus on those used in this work.

Chapter 3 goes on to provide an outline of the implementation of this work. This covers the data exploration, feature engineering, and classification tasks. Chapter 4 provides the results for each of the different sub-tasks of this work. This chapter also discusses the results as a whole and their implications. Due to the difficultly of the classification task a significant amount of exploratory work was performed which is also presented. This also includes a short discussion of why the Li et al. machine learning work cannot be fully used as a benchmark.

This thesis then concludes with Chapter 5 which discusses how well the objectives of this work were met. It then continues with an analysis of what work can be improved upon and provides several suggestions for future investigations.

Chapter 2

Background and Related Work

This section establishes a background for the project as well as grounding it in current literature. This pertains not only to the machine learning techniques used but also the fundamentals of the underlying biology. This is done by first exploring the problem domain, the Immunology. The machine learning approach is then discussed through the lens of current protein classification methods in literature.

2.1 Adaptive Immune System

In most organisms the immune system serves as the primary defense against invading organisms and agents that cause disease. It is therefore crucial in maintaining health and ensuring an organism's survival. In vertebrates this system can be roughly split between two categories, innate (non-specific) immunity and acquired (specific) immunity [13].

As the name suggests, innate immunity provides a generic response to pathogens that is not specific to the pathogen and is not long lasting [1]. Unique to vertebrates is the adaptive immune system, which serves as the more sophisticated second line defense to the first line of innate immunity.

More specifically, adaptive immunity is triggered if a pathogen evades the innate immune system. In doing so its presence must generate a threshold level of antigen as well as generate "stranger"/"danger" [14, 15] or damage signals, activating dendritic cells [16]. Antigens are molecules that trigger an

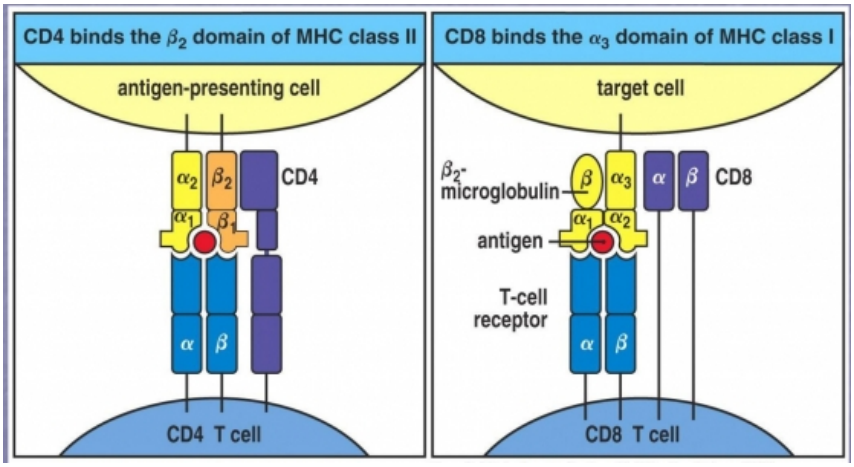


Figure 2.1: Difference in binding of CD4/CD8 T Cells to an Antigen Presenting Cell (APC); Source: [1]

immune response, for example a viral particle or cellular debris from a ruptured bacterium [17].

Broadly speaking the adaptive immune responses are carried out by a specific group of white blood cells called Lymphocytes [18]. These responses can also be separated into two rough categories: antibody responses and cell-mediated immune responses. These are preformed by two distinct classes of cell, B Cells and T Cells, respectively. [1]

The focus on this study lies in the domain of T Cells which respond to antigens primarily through cellular interaction. This includes the activation of phagocytes, antigen specific cytotoxic T Cells, and release of cytokines. This is in contrast to B Cells which function through Humoral immunity i.e. antibody production and the accessory processes surrounding it.

T Cells are identified by the presence of T Cell Receptors (TCRs) on their surface. These T cells react to antigens, being short peptides, bound to Major Histocompatibility Complex (MHC) Class I or Class II molecules found on the surface of Antigen Presenting Cell (APC). Mature T Cells exhibiting the α and β chains on their TCRs are composed of two distinct lineages. Those that interact with MHC Class I and those that interact with MHC Class II. These two lineages are labelled as CD4 T Cells, interacting with MHC Class II, and CD8 T Cells, interacting with MHC Class I. CD4 and CD8 are the names for

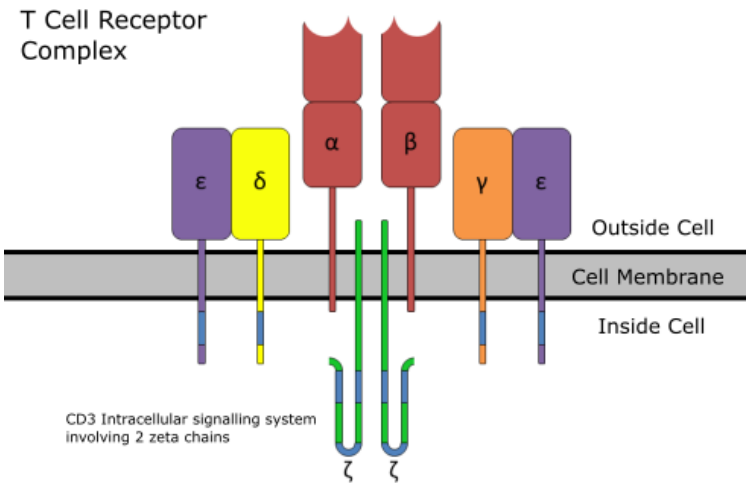


Figure 2.2: T Cell Receptor Complex, greek letters corresponding to different molecular chains.

the surface glycoproteins, which each lineage expresses, that act together with the TCR to recognize the MHC Class I/II molecule [19]. These two cell types make up the basis for the binary classification presented within this work.

2.1.1 T Cell Receptor

As mentioned previously T Cells express TCRs on their surface. These receptors are crucial in that they serve to recognize antigens bound to MHC molecules. Generally, TCRs will recognize and bind to a specific antigen, although this is a low affinity and degenerate binding [20]. Given the importance of this recognition, an understanding of TCRs is crucial to not only being able to understand T Cells and adaptive immunity but also to developing novel therapeutics. This is particularly true given that a significant portion of the emerging immunotherapeutics market consists of engineering and altering TCR specificity or introducing Chimeric Antigen Receptors (CARs) onto T Cells [9].

This being the case, understanding TCR structure and composition is critical to furthering our understanding of adaptive immunity. As previously mentioned, this work aims to design a classifier that can split TCRs into two groups. That is to say, having originated from a CD4 or CD8 T Cell. The data being used for learning is actually the CDR3 subsequence from the overall TCR

protein sequence. This distinction is key given the rather intricate structure of TCRs.

Molecularly, TCRs are trans-membrane peptides consisting of several different protein chains forming a Quaternary structure [21]. This can be seen in Figure 2.2, where each different chain is represented by a different greek letter. Each chain has a different role but they all work in unison to bind to the antigen/MHC complex when it is presented.

The invariant Cluster of Differentiation 3 (CD3) consists of the γ, δ, and two ϵ chains. Together with the two ζ chains the CD3 forms the portion of the TCR that is relatively unchanging. The other portion consists of the α and β chains that are highly variable. These two chains are primarily responsible for specifying what antigen a particular TCR will bind to.

Both variables regions in the α and β chains contain three hypervariable loops called Complementary Determining Regions (CDRs) [20]. CDR3β is the main loop responsible for recognizing the bound antigen, and as such serves as our data upon which a classifier will be built. If successful, as was shown in [11], then it suggests a fundamental difference between how CD4 and CD8 T Cells recognize antigens.

2.1.2 V(D)J Recombination

The mechanism behind how CDR3s are generated is called V(D)J recombination and its discovery merited Susumu Tonegawa the award of a Nobel Prize in 1987 [22]. This mechanism is responsible for the generation and subsequent variability of α and β chains.

During lymphocyte development in the early stages of T-Cell maturity the TCRs undergo this unique form of genetic recombination that hinges on the mixing and combination of several genes as well as deletions and additions of nucleotides. In the case of the α chain VJ recombination occurs, and for the β chain VDJ recombination (See Figure 2.3).

The Variable (V), Diversity (D), and Joining (J) genes all contribute to a somewhat stochastic combination of nucleotide segments that results in the

potential for an extremely diverse repertoire of potential receptor sequences. For a given TCR one V, (D) and J gene undergo recombination. Furthermore, there are random deletions and additions of nucleotides. These further contribute to the variability.

Although a highly complex mechanism the result is that the CDR3 sequences in both chains contain on the left hand side (upstream) several amino acids from a specific V gene, and on the right hand (downstream) several amino acids from a specific J gene. In the case of the β chain it may also contain a portion of a D gene. From a classification perspective this leaves a dataset with a variety of common ends and common beginnings, with a more variable center.

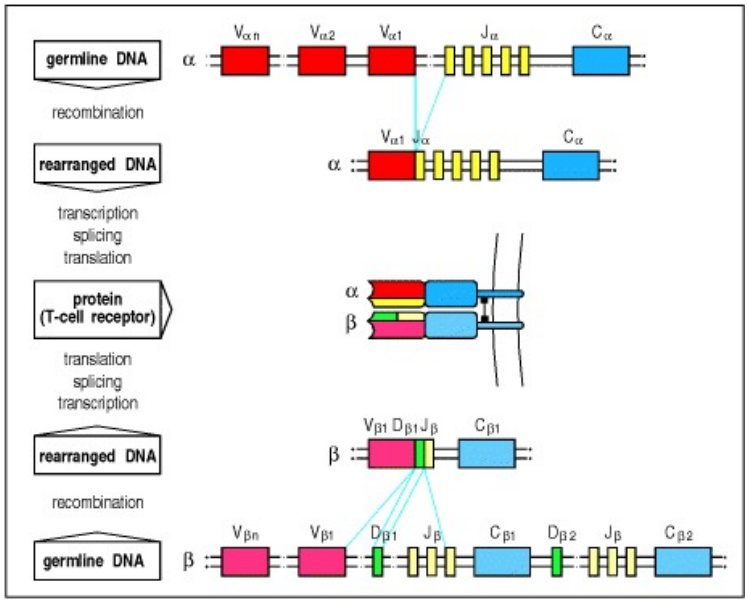


Figure 2.3: T-cell receptor α- and β-chain gene rearrangement and expression. C regions are Constant sequences. Source: Janeway et al. [2]

2.2 Computational TCR Repertoire Analysis

There are approximately 5×10^{11} possible TCRβ chains [23], and including the α chain that makes more than 10^{15} distinct TCRs [24]. Any given individual will exhibit $\sim 10^{12}$ T Cells [25] which suggests that an individual can only contain a small fraction of the potential repertoire. Although small, it is still a very diverse portion, as has been shown extensively [25, 23, 26, 27].

One of the key technologies that has made it possible to begin probing the TCR repertoire is High Throughput Sequencing (HTS) [28] which has significantly helped improve our understanding of the repertoire [29, 30, 31, 32]. That being said, the high diversity combined with factors like unequal polymerase chain reaction (PCR) amplification, sequencing errors, and sampling biases make quantification of the repertoire elusive [33].

As Heather et al. [10] have recently pointed out, there are a variety of tools that have become available for analyzing HTS data from TCR repertoires, each with their own focus. This year, 2017, there have been even more tools published [34]. For a list of those currently available see Table 2.1.

Tool	Extra Details	Reference
IMGT^{TM} Tools	Web based	[35]
Decombinator	Our tool, focus on fast annotations	[36]
iSSAKE	Can handle short reads	[37]
IRmap	Takes in 454 seq. data	[32]
MiTCR	Focus on error correction	[38]
MiXCR	Focus on error correction	[39]
IMSEQ	Focus on error correction	[40]
MiGEC	Handles uniquely barcoded data	[41]
TCRklass	Focus on reading seqs. without CDR3	[42]
RTCR	Focus on rare TCR reads	[43]
ARGalaxy	Web based	[34]
Presto	Handles uniquely barcoded data	[44]
IgBLAST	Originally for Ig sequences	[45]
Vidjil	Unique heuristic search	[46]
IMontior	Uses BLAST and second alignment step	[47]
TRIg	Focus on non-regular TCRs	[48]
LymAnalyzer	Uses short tags for assignment	[49]

Table 2.1: A variety of tools available for the computational analysis of the TCR repertoire

Although not perfect, the use of these tools has allowed our understanding of the TCR repertoire to grow rapidly, in part by expanding the number of observed sequences. This in turn facilitates the usage of statistical and machine learning based methods to broaden our insight into the repertoire's functioning. In a recent review article Heather et al.[10] highlight that com-

putational analysis will be the key to unlocking the information hidden in the repertoire. For example, we can now identify specificity groups in the repertoire [50, 51] and can predict structural elements of a TCR to a reasonable degree of accuracy[52].

One issue that still remains unclear are the differences in CD4 and CD8 TCR populations within the repertoire. Work by Li et al.[8], and to an extent our own work [12], has suggested that the two populations are distinct in both distribution and amino acid usage within healthy patients. Li et al. were able to demonstrate this using a supervised learning classification that could identify which of the two populations a given TCR belonged to using only the CDR3β sequence. This was done with $> 80\%$ classification accuracy.

The classification technique was based on our own previous work (Thomas et al. [7]) which used clustering to distinguish between CDR3s from several populations, each containing a different immunization profile. Our work seeks to validate the conclusions of Li et al. with our own dataset and to explore other feature engineering and classification techniques.

2.3 Machine Learning based Classification

The classification of TCRβ sequences can be also be approached from a machine learning perspective. This sections serves to analyze current approaches to feature engineering of proteins more broadly as well as applied to our specific problem. It also seeks to ground the classification algorithms used and their value in TCR classification.

From a computational point of view proteins are simply sequences of variable length containing symbols from a 20 entry vocabulary. However this simplicity belies the obvious difficulty entailed with using them as a machine learning dataset. This is especially true given the biological information that underpins each symbol and sequence.

An easy comparison to draw is that of sentences in Natural Language Processing (NLP). In order to be used by the vast majority of ML algorithms

the words in a sentence, or in our case amino acids in a protein, must be converted to a numeric value of a static length. This conversion is the feature engineering step, preceding the actual classification.

2.3.1 Feature Engineering

The specific problem presented in this work can be grounded in the bioinformatics domain as a protein classification challenge, a well explored but still burgeoning topic. Many of the current approaches to protein classification feature engineering leverage distance measures, which are based on alignment techniques like BLAST or Smith-Waterson. More advanced approaches have also arisen using methods like HMMs [53, 54] that rely upon remote homology detection [55].

Although they are strong models, the local sequence based approaches often fail to provide informative features [56], such is the case with predicting protein fold conformation [57, 58]. A number of studies have focused on feature extraction from whole sequences as a beginning point for ML approaches [59, 60, 61, 62]. These typically combine elementary biophysical features with engineered representations. This approach is particularly powerful in that the strongest features learned by a classifier typically expose some biologically important information.

In the context of TCR classification, our exact problem was approached by Schoinas [12] using this approach. More specifically, features like triplet and duplet frequency were combined with biophysical metrics (e.g. molecular weight, iselectric point, and other physiochemical properties) to create a final feature vector. Duplets and triplets are non-contiguous subsequences of two or three amino acids respectively. For example, "CA" or "CSA". Validation accuracy's of $\sim 70\%$ were reported however testing performance revealed overfitting with a final accuracy of $\sim 60\%$.

Li et al. [8] instead converted their protein sequences into Atchley vectors which are five dimensional vectors corresponding to [63] five physicochemical properties for each amino acid. These are based on polarity, secondary struc-

ture, molecular volume, codon diversity and electrostatic charge. So a sequence of L amino acids would have a feature engineered length of $5L$, where each amino acid has been replaced by the vector, and the final vector is flattened to be one-dimensional. Classification was then performed on the sequences binned by length. Li et al. posited that the reason for their successful classification was the preferential usage of small (2-3 amino acids) motifs by each population.

Thomas et al. [7] also used Atchley vectors however combined them with p-tuples. These are small amino acids long sub-sequences within the CDR3, where p is the length of the tuple (e.g. $p = 3$). First a selection of observed tuples were converted to atchley vectors and clustered. All tuples from a sequence were then mapped to a cluster and and the frequency of tuples mapped to each cluster was then used as a feature vector. For example, if there $k = 100$ clusters then each sequence would have a k dimensional feature vector. Taking into account both these works would suggest that successful feature engineering methods will hinge on sequence based features that can extract information on small motifs.

Recent work by Cinelli at el. [64] took the tuple concept even further when classifying CDR3s for a response to ovalbumin. The first stage was to generate a tuple frequency vector of all potential tuples within a CDR3 (e.g. if $p = 3$ then $20^3 = 8000$-dimensional vector). The dimensionality of these features was then reduced using a Gaussian Naive Bayes classifier to rank the best features. In this case feature reduction appeared to reduce noise in the data by removing uninformative features and subsequently improving classification. This suggests that our problem could be aided by feature reduction techniques.

To return to the more general case of protein classification, there have also been attempts at unsupervised feature learning. In the case of embeddings this is done before classification. This is in comparison to techniques like Deep Neural Networks that learn hierarchical features while training; taking simple inputs like one-hot encoded vectors.

Recently NLP has seen large improvements coinciding with the advances in Deep Neural networks, and one of the particularly relevant examples of this is word embeddings. Word embeddings are continuous fixed length vectors learned from shallow two layer neural networks that try to construct the linguistic context of words, e.g. Word2Vec [65] and GloVe [66].

These vectors can then replace words in sentences, converting the sentences into numeric vectors of embeddings. This concept has been extended to biological sequences, like in ProtVec [67], and in the case of proteins each word is a small motif much like a p-tuple. This is an especially pertinent example of how features have been learned from protein sequences in a purely unsupervised manner. This relation to p-tuples would also suggest it has value in our current classification task.

Another unsupervised neural network model that has seen much use applied to proteins is the autoencoder (AE). AEs are neural networks that take an input and try and reconstruct it after having put it through a bottleneck. The encoder consists of several consecutively decreasing layers that ends in a layer that produces a low dimensional latent representation of the input vector (See Figure 2.4). The decoder then tries to reconstruct the original input from this latent representation [68]. In proteomics the AE has been used in varying forms and has achieved state-of-the-art results in learning representations [69]. It has also been used to varying degrees of success in denoising and dimensionality reduction [70, 71, 72, 73].

2.3.2 Classification Algorithms

The other side of the machine learning aspect of this problem is the classification algorithm. Protein classification has seen the use of a multitude of different algorithms with varying degrees of success [74, 75]. Some of the most popular being Support Vector Machines (SVM), Neural Networks (NN), k-Nearest Neighbours (k-NN), and ensemble methods. This work has explored the application of these algorithms and their variants to the challenge of TCRβ classification. The following sections describe the individual classification al-

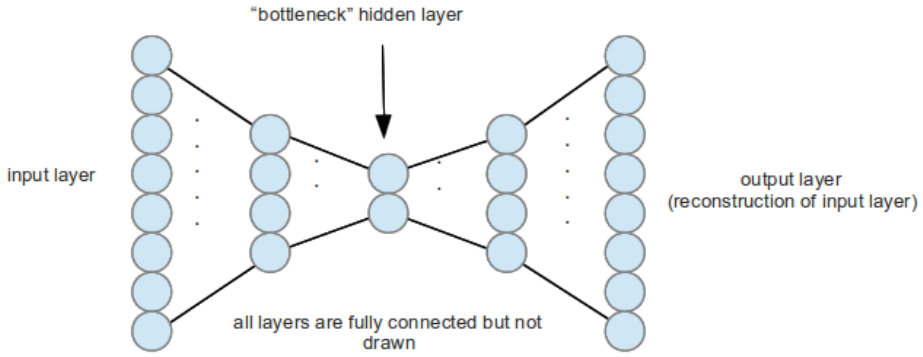


Figure 2.4: General structure of an AutoEncoder Neural Network for unsupervised learning. Source: [3]

gorithms and give a background to their functioning.

2.3.2.1 Support Vector Machine (SVM)

The SVM is a common but powerful discriminative classifier that seeks to find a separating hyperplane between two or more classes. More specifically it is a non-probabilistic binary linear classifier. The linear hyperplane is found by maximizing the margin between the nearest training observations from each class (See Figure 2.5). The observations that define this margin are termed the support vectors, which give the model its name. SVMs use soft margin optimization via slack variables [76] and minimizing typically the hinge loss with the 2-norm regularization of the weight vector.

SVMs can also take advantage of the Kernel-trick to transform the data to a high dimensional feature space which allows the model to separate non-linearly separable data [77].The SVM is also the technique used by Li et al., who used a Radial Basis Function kernel to separate the CD4 and CD8 sequences [11]

2.3.2.2 k-Nearest Neighbours (k-NN)

The k-Nearest Neighbours algorithm is a well explored non-parametric approach to classification that assigns a new point to the same class as the majority of its k nearest neighbours within a feature space [78]. For example, if $k = 1$ then a test point will be mapped to the class of its nearest neighbour

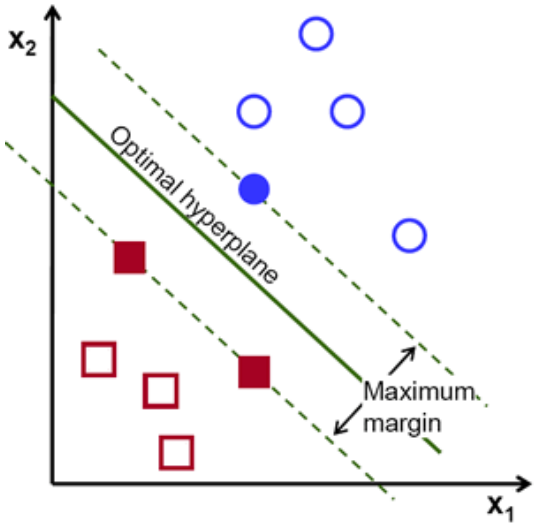


Figure 2.5: Example of an SVM with an optimal separating hyperplane that maximizes the margin. Source: [4]

in the training set. This is an example of lazy learning and is arguable one of the simplest machine learning algorithms [79]. Although not as powerful as some other techniques used it acts as an effective baseline for classification.

2.3.2.3 Deep Neural Network (NN)

Neural Networks, or Artificial Neural Networks, are machine learning algorithms inspired by the structure of a biological neural network. This is done by creating layers of interconnected nodes that pass information forwards, and is optimized by having error propagated backwards. This method of optimization is called backpropagation. The original model, the perceptron, consisted of one hidden layer [80] that approximated a linear transform. Multi-layer perceptrons use several hidden layers for a more powerful model [81].

Recently the use of neural networks with three or more layers, termed deep neural networks, has seen great success in a multitude of fields including bioinformatics [68]. In the case of a classification algorithm these generally contain several layers with a non-linear activation function like a Rectified Linear Unit (ReLU [82]) ending in a layer with a softmax layer. This produces a probability of belonging to one of a specified number of classes.

2.3.2.4 Recurrent Neural Network (RNN)

Recurrent Neural Networks are an adaption on regular feed-forward neural networks to tackle the challenge of temporal/sequential data. The network consists of a cell for each network that acts as layer which produces an output as well as passing information along to the next layer [83] (See Figure 2.7). Given that proteins are sequences of amino acids, it has made the RNN an obvious candidate for use in protein classification [84]. This is usually done by taking the hidden layer of the final cell and passing it through a hidden softmaxed layer.

This is particularly true given the state-of-the-art performance achieved by RNNs using Long Short Term Memory [85] Cells (LSTMs) and Gated Recurrent Units [86] (GRUs). These are cells that contain several different interconnected layers. Another powerful variant is the Bidirectional RNN which passes information both forward and backward [87]. Final hidden units from both directions are concatenated and then passed through a softmax layer for classification.

2.3.2.5 AdaBoost

Adaptive boosting or AdaBoost is a meta-algorithm that uses ensembles of weak learners to build a better overall classifier [88, 89]. A weak learner is an algorithm that can consistently find classifiers that are slightly better than random. For example, in the binary case classifiers that achieve an accuracy of 55%.

Boosting more generally is a method for converting rules of thumb into a highly accurate classifier. A boosting algorithm consists of first creating a program for deriving rules of thumb. Rules are then chose for a subset of the training set. This is repeated T times until finally all the classifiers are combined using weighted majority vote.

The basic concept behind AdaBoost is to maintain a distribution D on the training set and iterative train a weak learner on it. After every round, weights are assigned to the hardest examples which are those that were most

often misclassified by previous weak learners. This ensures that the boosting focuses on the misclassified examples. The algorithm for AdaBoost is provided in Algorithm 1. The most coommon base classifier, and the classifier used in this work, for learning rules of thumn is a Decision Tree.

Algorithm 1: AdaBoost

Input: Training data $S = \{(x_1, y_1), ..., (x_m, y_m)\}$
Init:$D_1(1) = ... = D_1(m) = \frac{1}{m}$
for $t = 1, ..., T$ **do**
Fit a classifier h_t:$R^d \rightarrow \{-1, 1\}$ using dist. D_t
Calculate weighted training error :

$$\epsilon_t = \sum_{i=1}^{m} D_t(i)\mathcal{I}[h_t(x_i) \neq y_i] \tag{2.1}$$

Calculate α using error:

$$\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t} \tag{2.2}$$

Update:
for $i \in \{1, ..., m\}$ **do**

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_t h_t(x_i)}}{Z_t} \tag{2.3}$$

Where Z_t is a normalization constant
end
end
return *Final Classifier:*

$$H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{2.4}$$

2.3.2.6 XGBoost

Extreme Gradient Boosting (XGBoost) is a set of gradient tree boosting algorithms, the one of most interest being the XGBoost classifier. The XGBoost framework came into popularity recently when it achieved winning results in several data science competitions [90]. The classification algorithm itself is a gradient boosted tree, the XGBoost implementation being heavily optimized for both compute and classification performance.

Similar to AdaBoost, gradient boosted trees consist of an ensemble of decision trees. More specifically, classification and regression trees (CARTs)[91]. These are the weak learners of the boosting system. The algorithm is termed 'gradient' boosting because it optimizes around an arbitrary differentiable loss function. Mean Square Error in the case of XGBoost.

By optimizing around these weak learners the algorithm uses an additive model to progressively add layers of weak learners. Each layer is optimized and then frozen before adding another layer. Gradient descent is used to optimize the layer by flowing the gradient through the parameters of the individual weak learners. This then reduces the overall residual loss. This approach has been termed functional gradient descent [92].

XGBoost improves on the standard gradient boosting approach by using several add-ons. Some of the variables that are limited are:

- The number of trees
- Tree depth; for example, to a maximum of 4-8 levels
- Number of nodes and leaves
- Number of observations per split
- Minimum improvement to loss in order to add a split to the true

Another addition to the standard algorithm is using weighted updates. Predictions by each tree are sequentially added together where each tree has a weight attached to it. This weight dictates how much it contributes to the sum. This is also called shrinkage and severs to limit optimization from becoming too aggressive [93]. XGBoost also uses stochastic gradient boosting, which amounts to using random mini-batches sampled from the training set to iteratively train the algorithm. Another add-on is L_1 and L_2 norms on leaf weight values to prevent overfitting. All of these addtions combined with the distributed and compute optimized framework make XGBoost a valuable classification algorithm.

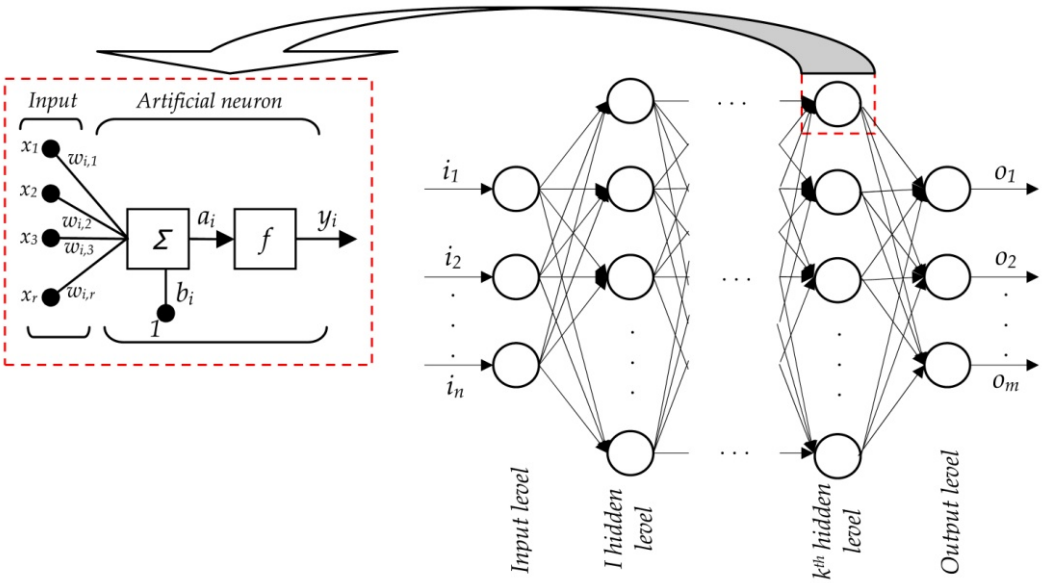


Figure 2.6: Example of a multi-layer neural network. Source: [5]

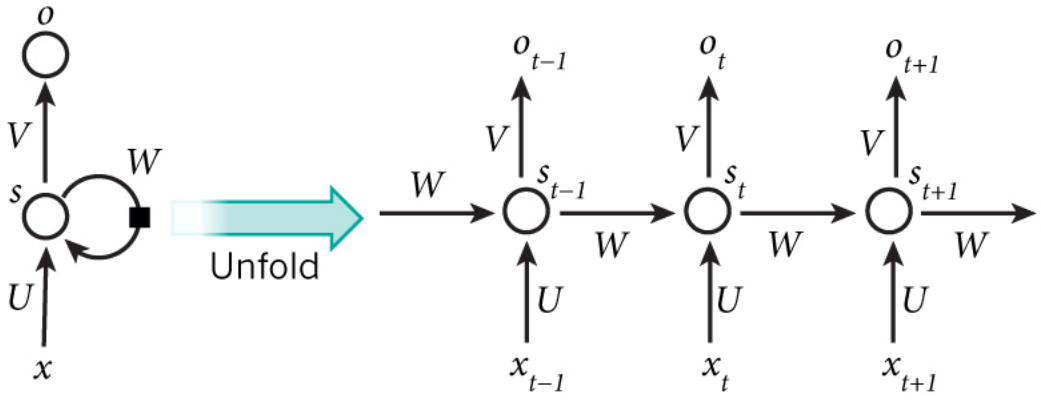


Figure 2.7: Example of an unrolled recurrent neural network. Source: [6]

Chapter 3

Materials and Methods

3.1 Data Retrieval

3.1.1 Study Subjects

TCR data was extracted from three healthy patients. The patients are labelled HV3, HV2, and HV1. Data was obtained primarily from patient HV1 as the other two patient datasets were not available until late in the study. Thus when this work refers to having used the 'single patient' dataset this is the HV1 dataset. When the full/complete dataset is referred to it is the pooled data of all three patients.

3.1.2 TCR Library Construction and Sequencing

The following wet lab work to extract the sequencing information was performed by researchers from the Chain Lab. Peripheral blood samples were taken from patients, separated into CD4 and CD8 subsets by fluorescence-activated cell sorting, and stored in RLT buffer at $-80°C$. The mRNA was then extracted, barcoded by reverse transcription, and amplified for several cycles by PCR. Sequencing was then performed with an Illumina MiSeq^{TM} machine.

3.1.3 TCR Sequencing Data Analysis

Analysis of the data produced by sequencing was performed using the Chain Labs Decombinator Tool[1] v3.1. This involves three main stages: Demultiplexing, Decombinating, and Error-correction.

The demuliplexing step cleans and deconvolutes the sequence data from different samples analyzed simultaneously by the sequencing machine. The decombination step takes these files and identifies rearranged TCRs, and subsequently outputs their 5-part classifiers. These values are the list index of the V gene used, the index of the J gene used, the number of V deletions relative to the germ line, number of J deletions, and the inserted nucleotide sequence. It is important to highlight these five values as they were not only used to produce CDR sequences but were also used during experimentation in the feature engineering portion of this work. The final step is the Collapsing which takes the decombinated output and performs sequence error and frequency correction.

The CDR3s were then extracted from the error corrected files using the CDR3 extractor script provided with the Decombinator. In the case of extracting CDR1/CDR2 sequences an R script was used [94], written by a member of the Chain Group, Mazlina Ismail[2]. This script also took the output of the Decombinator as input.

3.2 Data Exploration

The entirety of data exploration was performed using the the Python programming language [95], as is the rest of the software written for this work. Several modules were also used to assist in the analysis. These include the Natural Language Toolkit [96], Biopython [97], and Scikit-Bio [98]. All visualizations were produced using the Matplotlib python package [99] or the Seaborn package [100]. The majority of statistical methods used are to generate simple

[1] Available at: https://innate2adaptive.github.io/Decombinator/

[2] Available at: https://github.com/MazlinaIsmail/misc-scripts/blob/master/CDR3extractoR_V3.R

figures like heat maps and frequency bar graphs (histograms).

In the case of the t-distributed stochastic neighbor embedding (t-SNE) visualizations, Scikit-learn [101] was used in conjugation with Matplotlib. Typical t-SNE parameters were to first reduce the input using Principal Component Analysis (PCA) and then meet a perplexity of 30. This outputted a 2d vector for each input that could subsequently be plotted. When V and J regions were used in analysis their respective identification indices were extracted from the five part classifier produced by the Decombinator. These integer values were then used to represent specific genes.

3.3 Feature Engineering

Due to the challenging classification task a variety of feature engineering methods were used in an exploratory fashion to find the best potential features. Those presented were the best found and have been roughly split by what underlying technique they are based on.

3.3.1 p-Tuple Methods

This section encompasses the three feature engineering techniques that used p-tuples. All three of these methods represent length invariant approaches to feature engineering and so were able to use the full dataset. In all cases the CDR3 sequences were converted to sequences of overlapping 2-tuples or 3-tuples. The first and most basic technique was to convert the sequences of tuples to a 400D ($p = 2$) or 8000D ($p = 3$) vector were each entry corresponds to the frequency of a potential tuple seen in the sequence (See Figure 3.1).

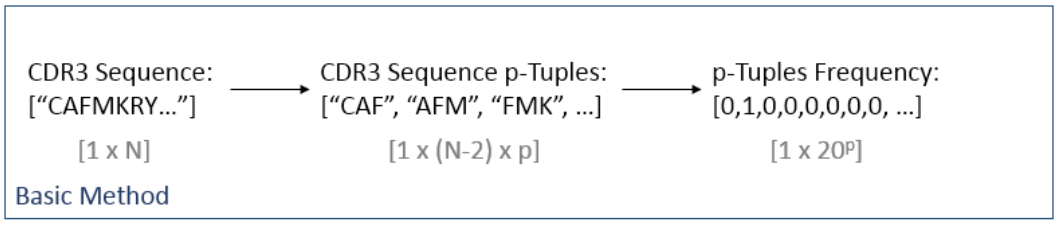


Figure 3.1: Basic p-tuple feature engineering method of tuple frequency counts

The next method used the same approach however it concatenates the tu-

ple frequency vectors of the CDR1 and CDR2 together with the corresponding CDR3 input sequence. So the final vector would have triple the dimensionality of the original basic method (See Figure 3.2).

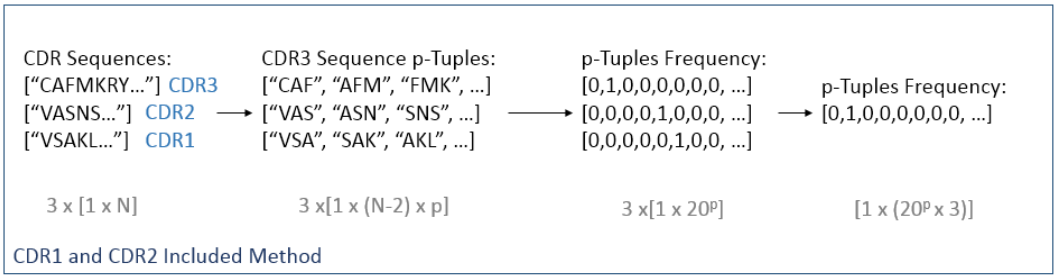


Figure 3.2: Basic p-tuple feature engineering method of tuple frequency counts with CDR1 and CDR2 incorporated

The third method is the method employed by Thomas et al. [7] which takes the p-Tuples of amino acids and instead replaces each amino acid within the tuple with its corresponding 5D Atchley vector. This creates sequences of $5p$ vectors. A set of q randomly selected tuples from all all sequences are clustered using K means. Values for q of 1000 and 10000 were tried. The Scikit-Learn implementation of mini-batch K means was used with $k = 100$ clusters. This resulted in a k-dimensional feature vector. See Figure 3.3 for the full computational pipeline.

3.3.2 Protein Embeddings

Another length invariant technique is using the ProtVec method developed by Asgari and Mofrad [67]. Two variants of this technique were used, one custom and trained on our dataset, and another trained on SwissProt protein sequences. The latter is a skip-gram model pretrained on non-overlapping 3-tuples from 324,018 protein sequences from SwissProt. These 100D vectors are available as supplementary material [3] in the form of a dictionary that maps each possible 3-tuple to its 100D vector.

After converting all overlapping tuples in a CDR3, all vectors are then added together to create a final 100D vector that represents the entire CDR3.

[3] http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0141287

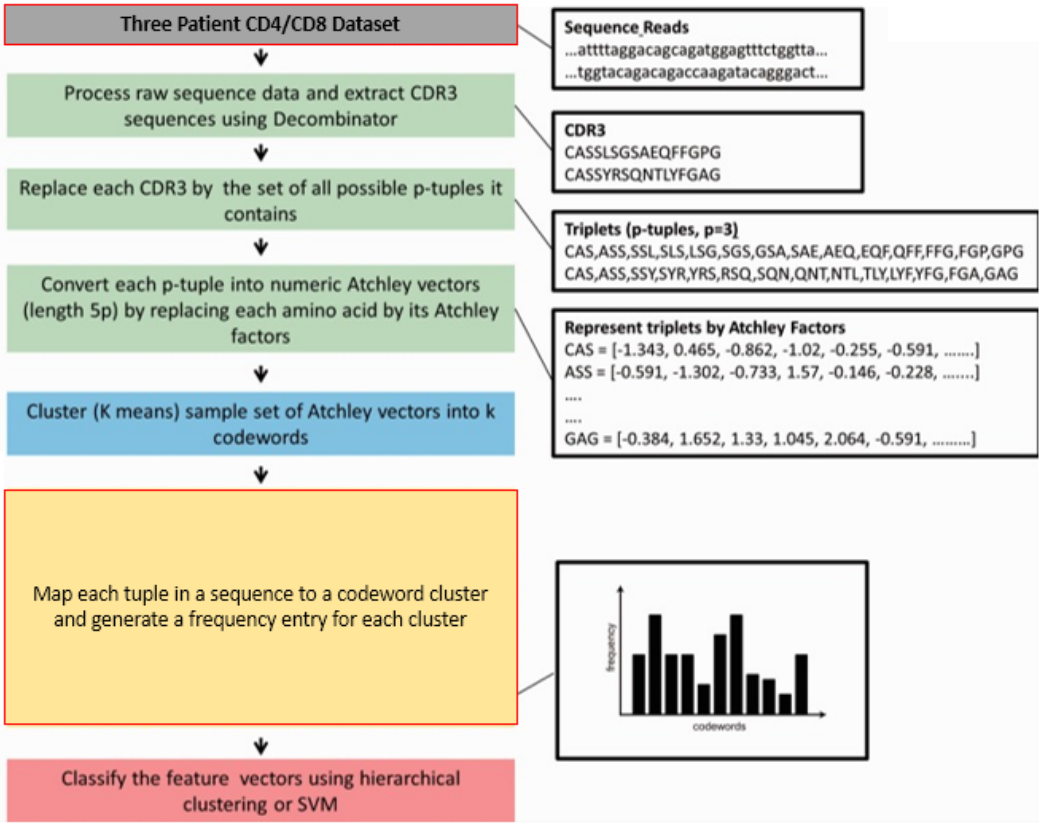


Figure 3.3: Thomas et al. [7] feature engineering method were red outlined boxes show logistic changes from original pipeline to suit our data

Alternatively the vectors are left and fed into an RNN, where each embedding corresponds to one cell. Addition of the vectors, as proposed in Asgari and Mofrad [67], serves to summarize the sequence information into one vector. It has also been pointed out that averaging or taking min/max for each dimension across the embeddings vectors works just as well as summation, if not the same [102].

The second embedding technique is a custom GloVe model trained on the complete CDR3 dataset. This used a skip window of 4, two layers, an Adam optimizer [103], and was trained till convergence. This produced 300D vectors for each potential 3-tuple. This is then finalized with the same process as above of addition before classification or feeding into an RNN classifier. See Pennington et al. [66] for layer implementation details.

3.3.3 Li et al. Feature Engineering

We emulated the method employed by Li et al. [8] which relies on fixed sequence length. For example, using 14 long CDR3s each amino acid is replaced with its Atchley vector and is then subsequently flattened so the final vector is $5 \times length$ which in this case is 70D. See Figure 3.4.

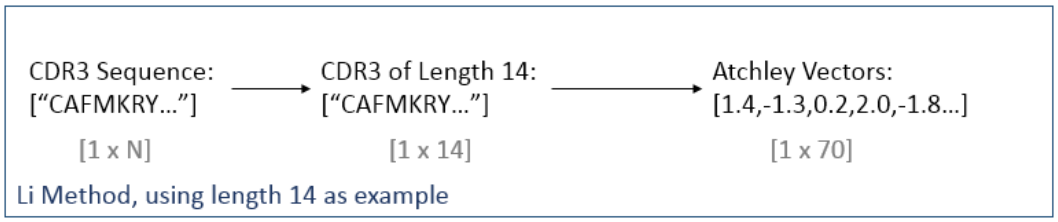


Figure 3.4: Li et al. [8] feature engineering method visualized with 14 long CDR3s

An extension of this method was to concatenate the CDR1 and CDR2 sequences to the beginning of the CDR3 sequence. This left a vector of [CDR1, CDR2, CDR3] flattened out so that it was two dimensional. In essence one long sequence (See Figure 3.5). The vast majority of CDR1s and CDR2s were 5 AAs long and 6 AAs long. Filtering took place to remove the CDR3s who's CDR1/CDR2 did not conform to this length. From the 14 long CDR3 population this removed 16.0% of the total examples. Between the two classes this removal was approximately even. This trend was also consistent with other CDR3 lengths.

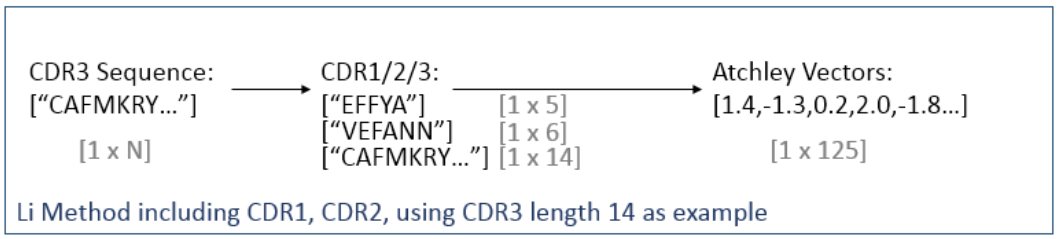


Figure 3.5: Li et al. [8] feature engineering method including CDR1 and CDR2, visualized with 14 long CDR3s

3.3.4 Amino Acid Positional Probability

This method first takes all CDR3s of a fixed length and then calculates the frequency histogram for the use of each amino acid in each position. This

is done for CDR3s in CD4 and CD8 separately. The histograms are then normalized across each position. Each CDR3 is then replaced with two vectors of the same side. The first contains the probability of having seen the amino acid present at each position if the sequence was a CD4. The second is the same except for CD8. These vectors are then concatenated to have a new vector double the size of the original CDR3 sequence (See Figure 3.6).

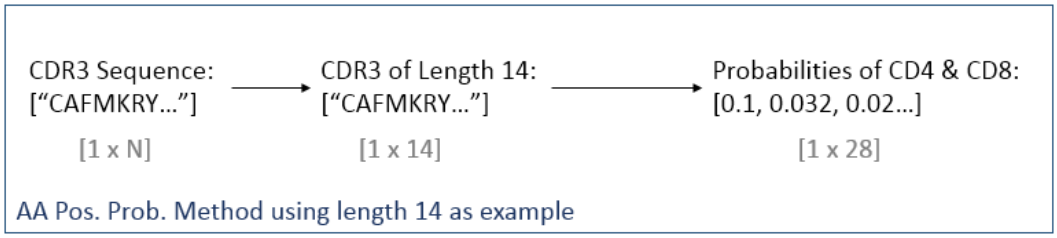


Figure 3.6: Amino Acid Positional Probability feature engineering method, visualized with 14 long CDR3s

3.4 Feature Reduction

3.4.1 Metrics Based Methods

Two methods were used that relied on metrics of individual features. Both were used to select the k best features which were then classified upon. The first metric was the Analysis of Variance (ANOVA) F-value. This was calculated using the Scikit-Learn function[4]. The second method calculated the mutual information between features and chose those with the best values[5]. The mutual information implementation is based on the work in [104] and [105].

3.4.2 AutoEncoder(AE)

The AutoEncoder was implemented using the Tensorflow package [106] and the Keras package [107, 108] for python. This was leveraged for two different particular feature engineering methods. The first is the p-tuple frequency

[4] Available at:
http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif

[5] Available at:
http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif

method that generates either 400D or 8000D vectors (multiplied by three if the CDR1/CDR2 are included). This AE is designed for compression and used the structure detailed in Table 3.1. This is a simple feed forward network trained until convergence using the Mean Square Error (MSE) loss function and an Adam optimizer [103]. The MSE loss function is defined as,

$$MSE = L(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \tag{3.1}$$

Where y is the ground truth, $\hat{y}$ is the predicted label, and m is the number of examples in the mini-batch. This network uses a latent (bottleneck) dimension of 8 units, although several others were tried.

Layer	Units	Activation Function
Dense	512	sigmoid
Dense	128	sigmoid
Dense	32	sigmoid
Dense	8	sigmoid
Dense	32	sigmoid
Dense	128	sigmoid
Dense	512	sigmoid
Dense	8000	sigmoid

Table 3.1: Structure of p-tuple compression AE, were final output units were altered to match input

The second AE was used to compress the data produced by the Li et al. feature engineering method. Due to the very structured and repeating nature of the Atchley vectors it is a good candidate for compression. Two networks were tried, a feed forward network like that in Table 3.1, and a recurrent autoencoder (RAE) using LSTM cells. Only the feed forward network was carried forward as it attained the same results without the added complexity of a recurrent network. The structure is detailed in Table B.1.

All AE networks were trained until convergence after which they were used to transform all samples from the given dataset. This was done by only using the encoder portion of the network to get the 8D latent representation. After this point the dataset was then trained upon as normal.

3.5 Feature Learning

3.5.1 AutoEncoder

The AE is implemented using Tensorflow. Its structure is detailed in Table B.2. The values inputted to the network are the amino acids one-hot encoded, concatenated, and flattened into a vector of size $Length_{seq} \times 20$.

The AE network also minimizes the mean square error using an Adam optimizer. The network was trained until convergence upon which the encoder was used to produce the latent representation of the dataset. This was then trained, optimized, and classified upon. Latent sizes of 2, 4, & 8 were tried. An XGBoost classification algorithm was used as it provided a clear baseline for comparison between different latent sizes.

3.6 Classification

For all dataset configurations the data was shuffled and then an 80%:20% train/test split took place. Validation was performed by 5-fold cross validation. In the case of optimization of hyper-parameters a grid search was performed; line search if there is only one hyper-parameter.

The only classifiers not cross-validated were the neural networks due to the computational intractability. Instead the networks used a 60%:20%:20% train/validation/test split. For a visualization of the pipeline see the Figure 3.7. Note that duplicate CDR3 sequences were removed from the training set in order to emulate Li et al. [8] and improve classification. These accounted for 3.3% of all the data available.

3.6.1 Testing Set

In order to ensure good practice the Test Set is only used for the classification of the final combination of the best classification algorithm and the best feature engineering method. The accuracy of the classification is then measured and along with recall, precision, and support, is used as the metric of success of this work.

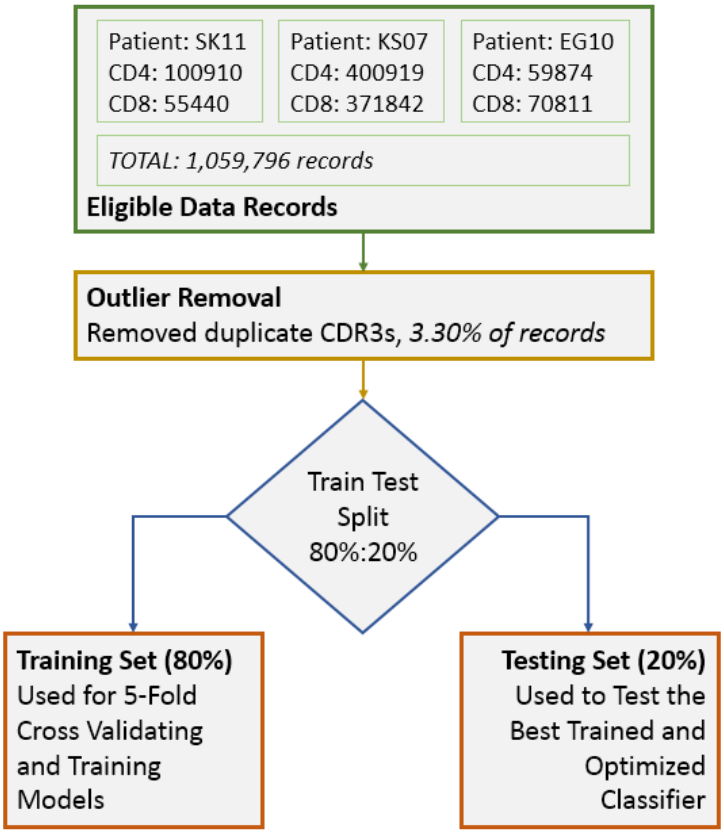


Figure 3.7: Overview of the classification process

Data Record Example:
This is one 'X' value

CDR3	CDR2	CDR1	V Gene Index	J Gene Index	V Deletions	J Deletions
CAWSPKGPQETQYF	SVGIG	GTSNPN	20	6	4	0

Corresponding 'y' Value, One-Hot Encoded:

CD4	CD8
1	0

Most work used only CDR3 as the feature set however some feature engineering techniques leveraged the other values.

Figure 3.8: Example of one Data Record and the feature sets available to be used for classification

3.6.2 Confidence Intervals

Due to the large number of feature engineering method-classification algorithm combinations, validation accuracy's have not been included with a confidence interval for clarity. Instead the validation accuracy's for the best classifiers

from each feature engineering method have been provided with a confidence interval. However more generally, confidence intervals were calculated by running 100 rounds of bootstrap for each combination.

3.6.3 Imbalanced Classes

In the particular case of classifying on individual patients, HV3 exhibited a 33:67 class balance. HV2 and HV1 showed balanced of ∼55:45. In order to correct this balance the commonly used Synthetic Minority Over-sampling Technique (SMOTE [109]) was implemented. This was done using the Imbalanced-Learn [110] python package. Note that is is done only on the training data so as not to artificially affect the accuracy of validation or testing classification

3.6.4 Standard Classifiers

All of the following classifiers are deemed 'standard' in that they were implemented using the Scikit-Learn Library. These are covered in the following list along with what hyper-parameters were optimized. All values of set and optimized parameters used in final models are contained with the Appendix. Any (hyper)parameters not optimized used the default values as provided by Scikit-Learn.

1. k-Nearest Neighbours
   (a) k; within range $[1, 7]$
2. Support Vector Machine
   (a) kernel; between Linear, Gaussian, Polynomial
   (b) Polynomial Kernel Degree; n between $[1, 3]$
   (c) C; within range $[1 \times 10^{-3}, 1 \times 10^{3}]$
   (d) γ; within range $[1 \times 10^{-5}, 1 \times 10^{2}]$
3. AdaBoost
   (a) Number of Estimators; within range $[10, 1000]$,
   (b) Learning Rate; within range $[10, 1000]$

For explicit details of implementation please see the Sci-Kit Learn documentation [6]

3.6.5 XGBoost

The XGBoost classifier was implemented using the XGBoost library for python[7]. To reduce overfitting maximum depth was set to 6, minimum sum of instance hessian needed in a child was set to 2, and the minimum loss required to create a new partition in a leaf node was set to 0.5. The L_2 loss was optimized between $[0, 10]$. All other values were left at default.

3.6.6 Deep Neural Network (NN)

All implementations of NN classifiers were built using Tensorflow and used randomized mini-batches by shuffling after every epoch. Furthermore all dense (fully-connected) layers used batch normalization [111], gradient descent was performed using an Adam optimizer, weights were initialized using an Xavier Initializer [112], biases were initialized at zero, and learning rate was decayed to 99% each epoch. As mentioned previously, these networks were written in python using Tensorflow.

The primary classification network is a standard 5-layer network with ReLU activation functions and progressively decreasing unit sizes by powers of two. The number of units was altered depending on the feature engineering method used. See the appendix for the corresponding architecture to a given feature engineering method (Table B.3 & B.4).

For all networks early stoppage was performed when validation accuracy began to drop during training, preventing overfitting. The hyper-parameters optimized around for both networks are the learning rate, mini-batch size, and number of hidden dimensions.

[6] Available at: http://scikit-learn.org/stable/documentation
[7] Available at: https://github.com/dmlc/xgboost

3.6.7 Recurrent Neural Network (RNN)

The RNN was also implemented using Tensorflow in the python language and its structure is visualized in Figure 3.9. This was fed either one-hot encoded amino acids for each time step, or 3-tuple embeddings for each time step. A convolutional layer was initially placed between the RNN when being fed embeddings however this was removed as it negatively impacted performance. The network was initialized and optimized using the same setup us the standard feed forward neural networks.

The RNNs were trained on all sequences and to account for length differences all sequences were padded up to the maximum sequence length seen in the dataset, 31, with zeros. These zero vectors when inputted were either the same shape as the one-hot encoded vectors or the embeddings depending. Compared to training the model on only fixed length sequences this had a slight effect on increasing accuracy. Clipping longer sequences and lower the maximum allowed length was also tried however this negatively impacted performance.

More accurately the RNN used is a Bi-directional RNN. In the model constructed both final latent vectors from each direction are concatenated and run through two dense layers and a final sigmoid layer. Several architectural choices were optimized before running optimization of hyper-parameters. These are described below:

- LSTM Cell vs GRU Cell?
  - LSTM Cells were chosen as they showed more stability.
- Stacking?
  - Stacking was tried with 2 and three layers however it did not postively effect the performance.
- Dropout and at what level?
  - Dropout was tried between 50% and 80%. It did not help decrease overfitting and it made training unstable.

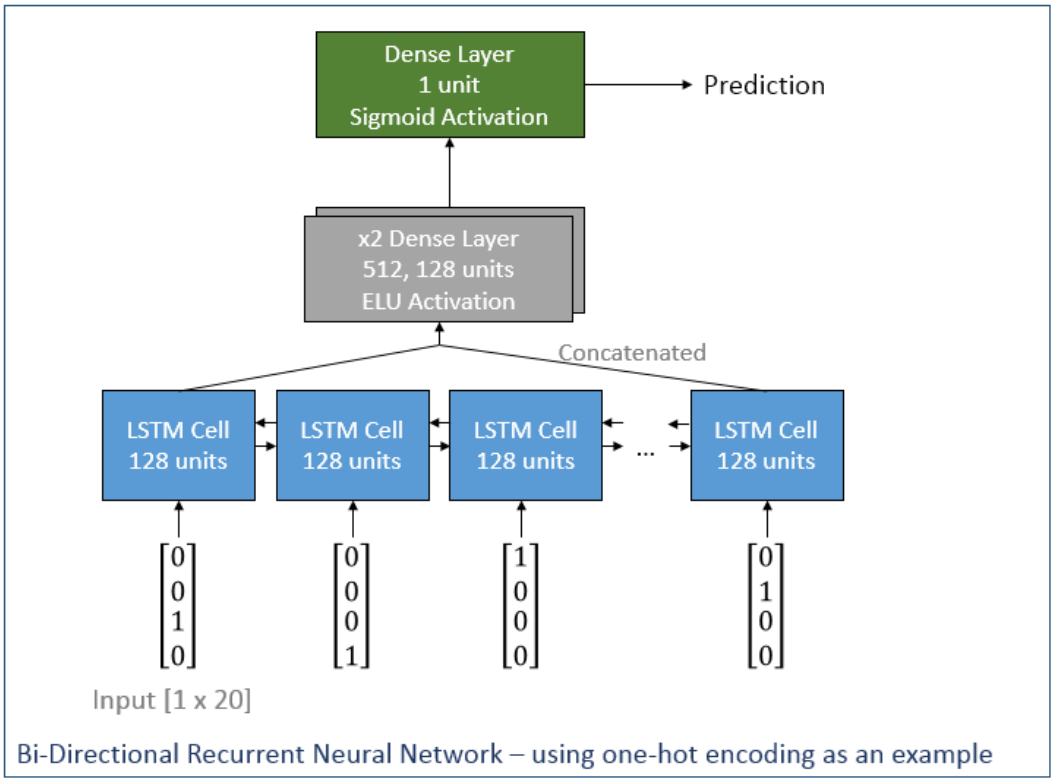


Figure 3.9: Recurrent Neural Network Architecture

- Attention and what size?
  - Attention was added up to 10 units per cell. Addition of attention at whatever size had no appreciable effect on performance and also reduced .
- Batch Normalization on dense layers?
  - This increased stability while training but decreased overall accuracy and so was left out.

After having established the architectural setup of the Bi-RNN the model was trained and optimized by grid search around the learning rate and mini-batch size

3.7 Li et al. Data Extraction

Li et al. provided the data used in their analysis in the form of raw sequencing data in FASTQ format. This is in contrast to the processed CDR3 sequences.

These files were downloaded and unsuccessfully processed using the decombinator (v3). The aim being to test the methods explored in this work on their dataset. Unfortunately the decombinator could not extract any sequence information as the data contained little readable V/J gene information. Li et al. did not provide the software used for extraction, although they do say they use an edited decombinator version 2 that is equivalent to the current version 3.

Other approaches were attempted to extract the CDR3 sequences, all unsuccessfully. These include the decombinator short read script from version 2, RTCR [43], ARGalaxy [34], and MIXCR [39]. We also contacted the researchers for a copy of their CDR3 sequences however the data was not forthcoming.

3.8 Software

The software was maintained by the Git[8] revision control software and backed-up using a public repository on Github at the following address:
`https://github.com/innate2adaptive/tcRIP`

[8] Available at: https://git-scm.com/

Chapter 4

Results

4.1 Data Exploration

This section covers the initial phase of research performed in this work. This was to establish informative statistics of the patient datasets with the end goal of using these values to build effective classifiers and feature extractors. This also guided further exploration of the data when leveraging the different feature engineering techniques.

4.1.1 Dataset Attributes

The first step in analyzing the dataset was to establish the basic statistics of the two populations across the different patient samples. These values have been presented in Table 4.1. From a machine learning perspective this is a very useful dataset in that the complete dataset contains an almost even class balance as well as close to 1 million distinct data points (visualized in Figure 4.1).

Patient:	HV1	HV3	HV2	Complete
CD4 Sequences	58069	98556	361613	494141
CD8 Sequences	69006	53086	331328	430755
Duplicates	1806	2354	22409	34933
Total Sequences[1]	127075	151642	692941	924896
CD4 to CD8 Ratio	46:54	65:35	52:48	53:47

Table 4.1: General statistics for all patient CD4 and CD8 TCRβ populations

The values for the TCRα chains show very similar proportions except

there are uniformly fewer distinct sequences (Appendix Table A.1). When classifying upon the TCRβ chains the size and class weights of the complete dataset make using it preferable over any of the individual patient datasets. It also allows better conclusions to be drawn should the feature engineering or classification reveal any biologically relevant patterns that are not patient specific.

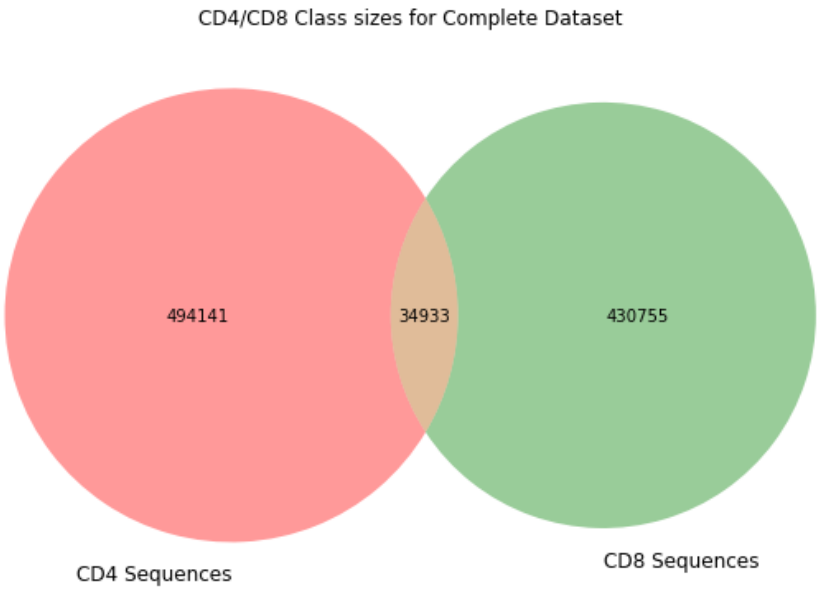
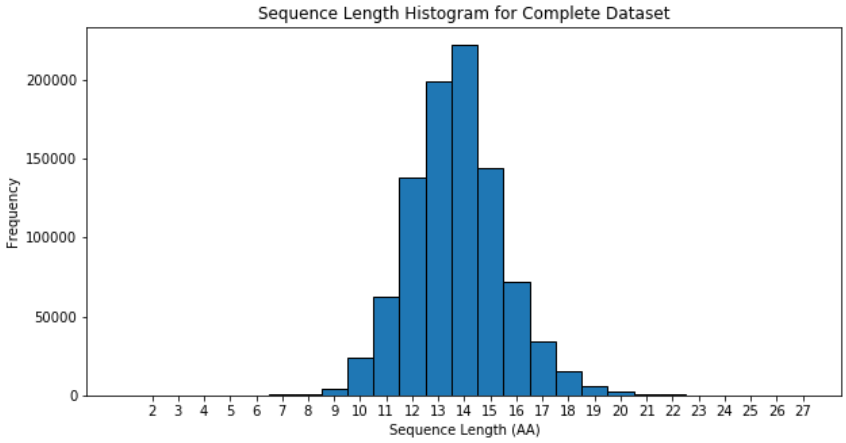


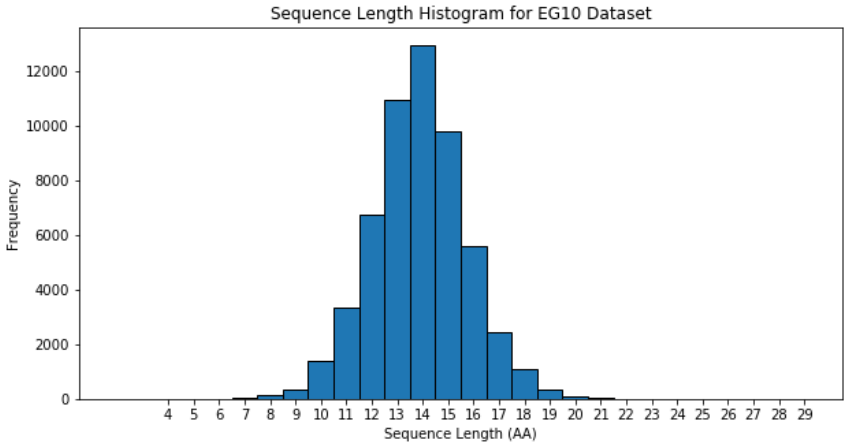
Figure 4.1: Venn diagram showing the shared sequences between CD4/CD8

Another issue that required investigation was sequence length. Dealing with variable sequence lengths is one of the bigger issues in machine learning applied to fields like proteomics and genomics, in that most methods require a static sequence length. Unfortunately it is rarely the case that a proteomics dataset contains all sequences of one length, as is the case here. In Figure 4.2 it can be seen that across all patients most sequences sit between 9 and 20 amino acids long. The most popular length being 14 long with >200,000 sequences. There also appeared to be no significant difference between the CD4 and CD8 population sequence lengths.

Given that there is a relatively large number of sequences that are 14 long many of the length invariant techniques were performed by separating sequences by length. As would be expected, this was primarily upon 14 long



(a) Sequence length distribution for complete dataset



(b) Sequence length distribution for HV1 dataset

Figure 4.2: Sequence length in a single patient population and all populations

sequences before being extended to other lengths. This was also useful in that it allowed direct comparison with other literature like that of Li et al. which used this same binning procedure [11].

4.1.2 V-J Gene Usage

Establishing the usage of V and J genes within the datasets was the first key step in establishing a difference between the CD4 and CD8 populations. This dataset saw the use of 12 distinct J genes and 58 distinct V genes. In most cases these genes are referred to by their index as used by the Decombinator.

From the most simplistic point of view, and potentially the most useful, both CD4 and CD8 populations show the same usage of J genes. This can be

seen clearly in Figure 4.3a. This would tend to suggest that direct usage of J gene associated features is not valuable towards separation. By extensions it also suggests that J gene fragments at the N-terminal part of CDR3s will be less useful in classification.

In slight contrast, some V genes showed preferential use by CD4 or CD8 but by small margins (Figure 4.3b). This suggested that including the V gene used as a feature could be valuable to separation. In order to investigate this further a standard non-linear classifier, an SVM-RBF, was used to classify sequences using only combinations of their V and J gene indexes (Results in Table 4.2).

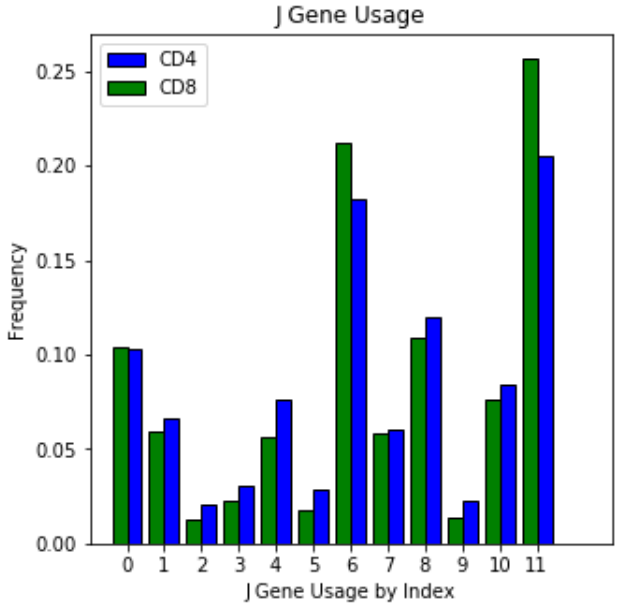
Using the V region as a feature appeared to be equivalent to using both V and J genes as features, whereas the J gene performed only slightly better than random classification. This being the case the V region was tagged for incorporation with other feature engineering methods. This also suggested that there would be preferential usage of some small motifs retained in the V influenced N-terminal region, between CD4 and CD8 populations.

Feat. Engineering Method	Accuracy
V Gene Only	59.0±0.05%
J Gene Only	54.3±0.05%
V and J Gene	59.1±0.05%

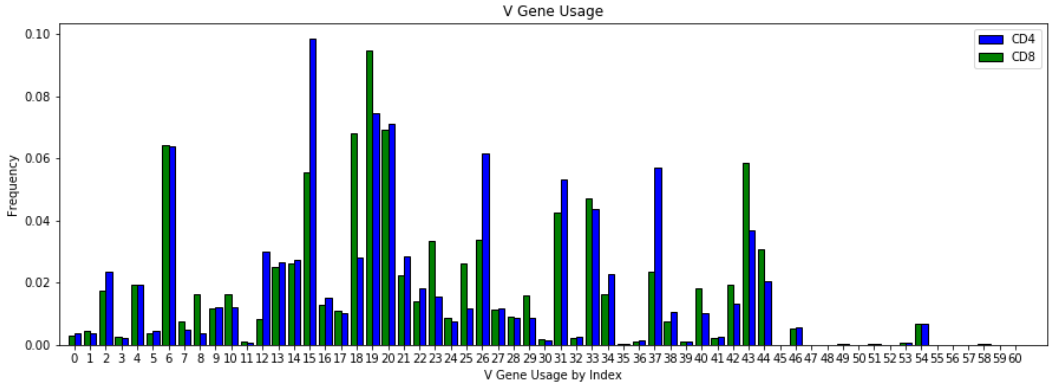
Table 4.2: Classification of CDR3s using only V and J gene indexes as data for the complete dataset using an SVM-RBF classifier

Another technique that lent itself towards understanding the usage of V and J genes was t-SNE. Converting the CDR3s via the Li et al. method to numerical values and then performing t-SNE provided a biologically relevant way to visualize the two populations within a 2D space. Figure 4.4 provides this visualization. From this two main insights were drawn. The first of which is that the two classes are not trivially separably using this feature extraction method, and that there is some form of clustering occurring.

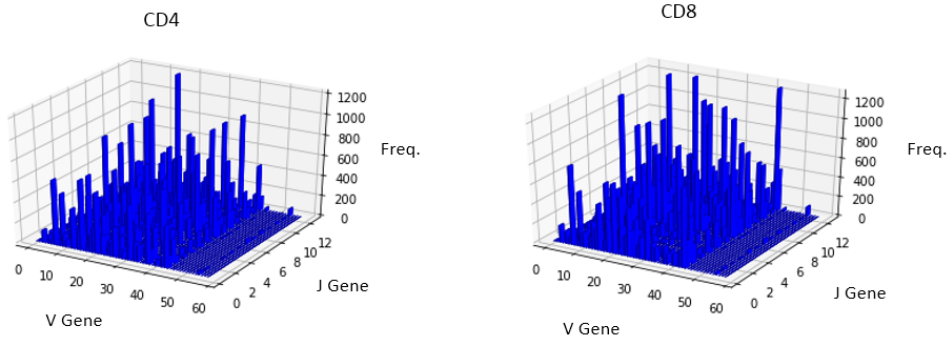
These clusters upon manual investigation proved to be sequences of the same V-J combination. This follows given that sequences from the same com-



(a)



(b)



(c)

Figure 4.3: Usage of V and J genes in CD4 and CD8 populations for the complete dataset

bination will usually have the same or similar amino acids at their C-terminal and N-terminal regions; these being clipped chunks of the original genes. Being that the two populations are distributed between all clusters this also suggested that any separating hyper-planes occurring during classification would have to intersect all clusters.

Given this insight we visualized the most common VJ combination, TRBV2 with TRBJ2-6 by the same method. The overall aim being to search for differences within a particular V-J cluster. As can be seen in Figure C.1 the same two patterns are seen, the two populations are spread and are occupying clusters together. These clusters can be seen well in Figure 4.5.

Upon manual investigation it appeared that many of these clusters occurred because of retention of variable amounts of the original V-J genes. For example one cluster would contain more conserved J region, and so more amino acids in common, than another cluster. Another commonality in the clusters was small motifs within the center of the sequences e.g. 'GA'. It is possible that these smaller clusters within clusters are actually antigen specificity groups, as recently suggested by [51, 50], who saw similar clustering with CDR3 sequences and identified them as such.

To give a concrete example, one cluster contained sequences with the pattern:

$$CAIR****SYEQYF \tag{4.1}$$

Where '$*$' denotes a variable amino acid. This implied that classification within this cluster would occur around the four variable amino acid positions in the center. Given the assertion by Li et al. [8] that good separation was occurring due to the presence of particular small motifs, then by extension these motifs likely exist within the center of the sequence were amino acid variability is more common.

To summarize, the analysis of V and J regions provided several key insights into the dataset. Firstly, that sequences were clustering due to usage of the same motifs at both N and C terminals due to usage of the same V and J

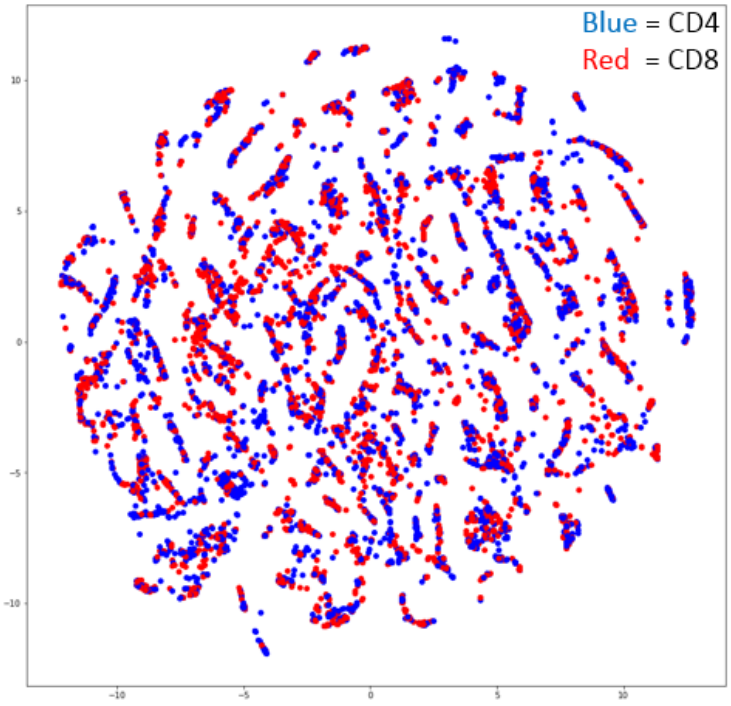


Figure 4.4: 2D t-SNE Embeddings on CDR3s from by CD4 and CD8 populations using the Li et al. feature extraction method for 14 long sequences

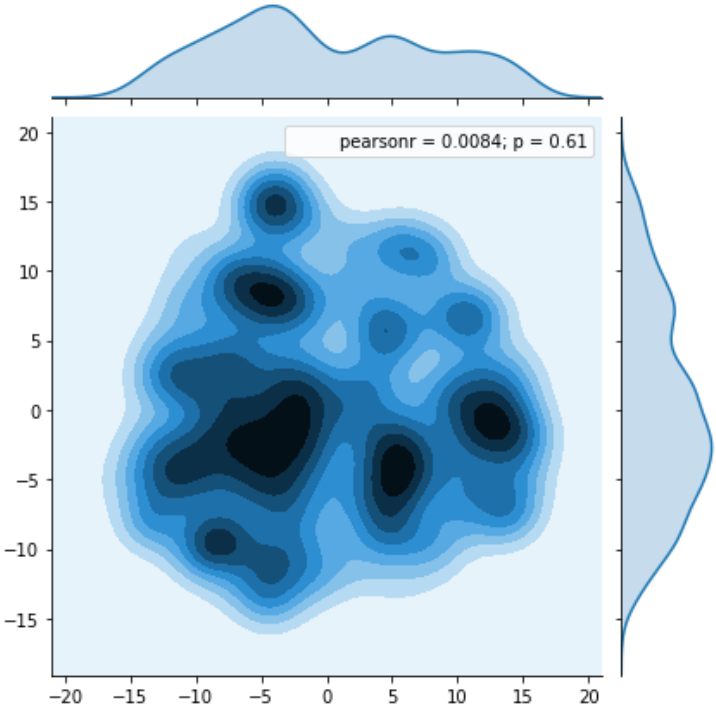


Figure 4.5: 2D t-SNE Embeddings on CDR3s from the most common V-J combination, TRBV2 & TRBJ2-6, using the Li et al. feature extraction method for 14 long sequences

genes. Within these clusters are further clusters containing sequences with similar but highly variable motifs in the center. Secondly, encoding the V region explicitly is a valuable feature. This suggests that some of the clusters will show preference towards CD4 or CD8 sequences, and that the preferential usage of V genes will likely be seen doubly; with the preferential usage of small motifs resulting from the V genes. This guided the next step in data exploration which was analyzing usage of small motifs.

4.1.3 Small Motif Usage

The next step was to analyze the usage of small motifs, called p-tuples. The 10 most common tuples for both CD4 and CD8 were calculated and compared for p in range $[1, 7]$ (See Figure C.2).

All of the most common tuples were found to occupy the C-terminal and N-terminal regions of sequences, again coming from the V and J genes and not the more variable center. For example, a common beginning is "CASS" and a common ending is "TQYF." Furthermore many of the other common tuples were variants of other ends and beginnings, like "EQYF." As would be expected, tuples originating from the J region were more represented than tuples from the V region, the V region being more variable. Given the strong link to V and J genes this tuple usage is another way to view the clustering seen previously.

Between the two populations there appeared to be little significant difference in their tuple usage. Given that the common tuples are dominated by those from the terminals it further supports that any tuples significant in distinguishing CD4/CD8 membership will be towards the center.

This being the case the next step taken was to filter the sequences by removing the first 4 and last 4 amino acids to focus on the center of the sequence. This range was informed by the previous tuple usage and experimentation. Unfortunately both populations showed relatively the same tuple usage. Interestingly, many of the most common tuples in the center showed a heavy usage of Glycine and Alanine. Biologically this could be due to the

flexibility that the smaller amino acids provide the sequence.

It is difficult to draw any implications however it suggests experimenting with classification using clipped sequences might lend further insight. This being the case the next step in analysis was to use a finer grain approach by instead analyzing the usage of amino acids at specific positions. Furthermore this method could also be compared to Li et al. who also analyzed positional amino acid preference.

4.1.4 Preferential Amino Acid Usage

The next technique employed to gain insight is the analysis of positional usage of amino acids. Global usage of amino acids in CDR3 sequences (Figure C.3) matches what has already been seen in literature [113], which led to the question of do the CD4 and CD8 populations show contrasting usage of amino acids at different positions?

Figure 4.6 gives heat maps of the distributions of amino acids in each position for 14 long sequences. From a naïve visual perspective there is little obvious difference between the usage within CD4 and CD8 populations. Analysis of Figure 4.6e reveals there actually is a slight difference between amino acid usage of the two populations.

Namely, the CD8 CDR3s are more likely to use negatively charged amino acids (Asp, Glu), and CD4 CDR3s are more likely to use positively charged amino acids (His, Lys, Arg). This matches what has been reported by Li et al. [11]. Also examining the first four positions suggests a link to the preference of certain V genes between the populations. CD4 CDR3s are more likely to use V genes beginning with the tuples CS, CSA, and $CSAR$. This is in contrast to CD8 CDR3s that prefer V regions beginning with CA, CAS, and $CASS$. This difference is small and as such, was not noticed in the small motif usage analysis (Section 4.1.3). This preference of initial tuples and central amino acid electrostatic charge will likely contribute to the separation if encoded in features. The latter is particularly relevant to the Atchley vector method used by Li et al. as one of the dimensions of the Atchley vector is electrostatic

charge.

Both Figure 4.6a & 4.6b are not remarkably informative in that the heavy usage of V-J associated tuples skews the contrast of the diagram at either end of the sequence. In contrast Figure 4.6c & 4.6d provide the usage in the more variable center of the sequence. What is particularly notable is the heavy usage of glycine at the center of the sequence, also noted during the motif analysis. Although it is not constructive in classifying it provides an insight into the underlying structural biology.

The presence of glycine (and proline) is known to be characteristic of small loops and turns [114]. This could be due to its effect on lowering activation energy for structural conformation. In the case of TCRs this could also be advantageous in establishing binding degeneracy, making the loop more flexible. From a classification perspective this hints that potential differences between CD4/CD8 may lie in the overall tertiary structure of the sequence.

4.2 Feature Reduction

During initial experimentation with classification it became apparent that Feature Reduction could be used to gain a better insight into which features were aiding the most in classification. The results of this experimentation are particularly useful when considered with the insights gleaned during data exploration.

4.2.1 Mutual Information

The first approach was to calculate the mutual information for each feature produced by the Li et al. method. This is a common method for calculating feature importance. Using the Li et al. method method for feature engineering with only the CDR3 sequences showed that the center amino acids had much less dependency (more mutual information; see Figure 4.7).

This further supports the concept that the center amino acids are the key to classification. It should be noted that this operates under the assumption that less dependent features will provide more information to a classification

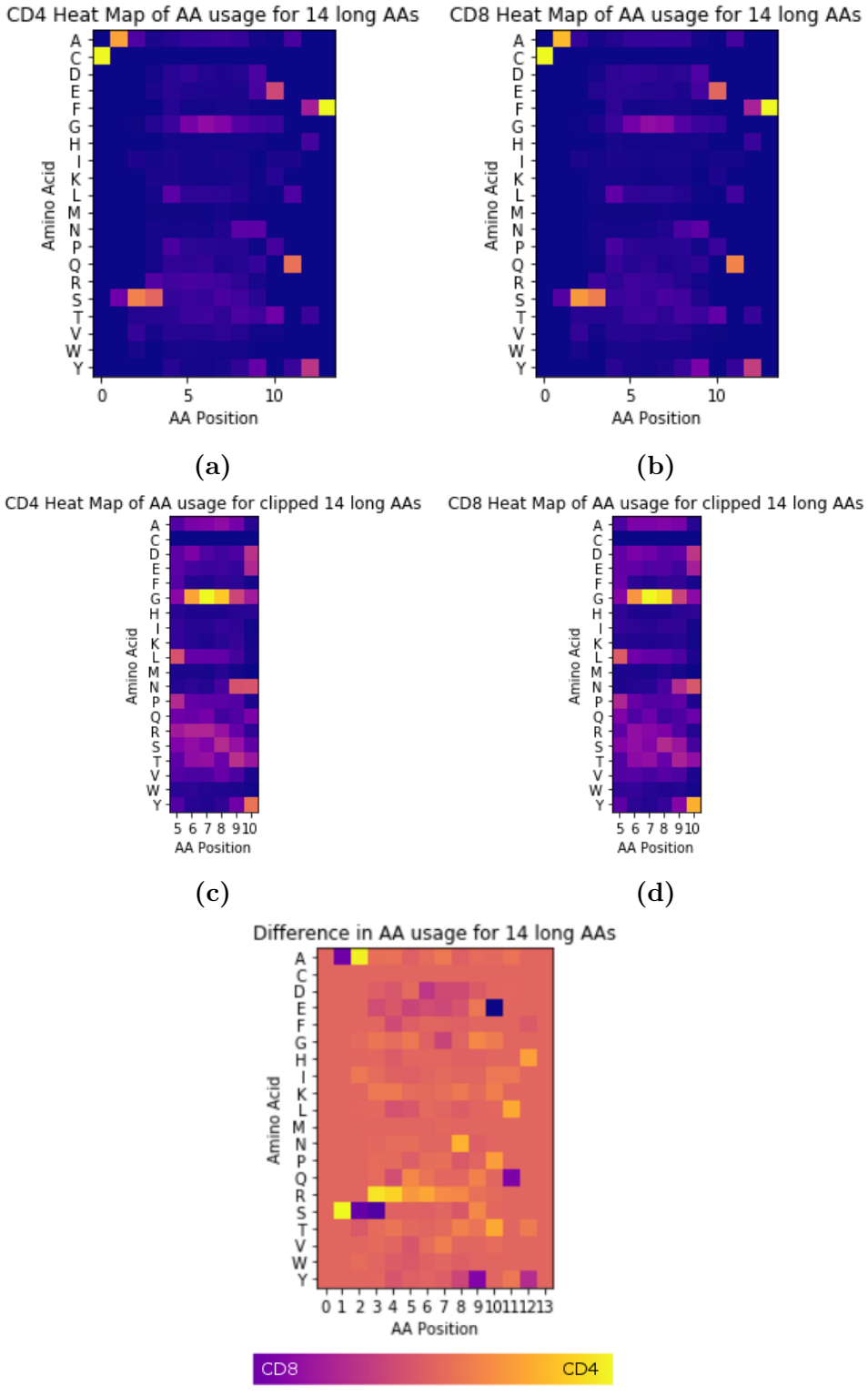


Figure 4.6: Usage of amino acids at different positions in CD4 and CD8 populations for the complete dataset; the clipping removed the first and last four amino acids. The 'difference' sub-figure shows the ratio of CD4 to CD8 converted to a color map.

algorithm.

Including the CDR1 and CDR2 sequences obscured this pattern, and even when analyzed on their own they did not show any distinct positional preference. Using the ANOVA F-value showed the same results as the Mutual Information. Interestingly including the V gene index as a feature proved to be incredibly effective in that it appeared by both metrics to be the most valuable feature.

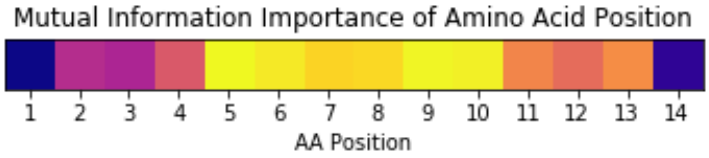


Figure 4.7: Average of Atchley vector mutual information at each amino acid position for the complete dataset

It should be pointed out that it was expected that the beginning of the sequence would provide some information for classification due to the tuples originating from V gene usage. This however appeared not to be the case, likely because the information is more explicitly encoded in the V gene feature (not included in Diagram 4.7).

4.2.2 AutoEncoder

The other feature reduction method used is an AutoEncoder (AE). An AE was used because it allowed for feature compression and an easy analysis of what features were being easily compressed. As can be seen in both Table 4.3 & 4.4 feature compression with an AutoEncoder was not successful. Although the number of features was reduced, the AE did not succeed in reducing noise without lowering the validation accuracy, compared to the non-reduced methods presented in Section 4.4.

That said, the difference in accuracy between the various methods follows what would be expected. For example, the larger latent dimensions generally resulted in a higher classification accuracy. The same is true when comparing methods that only used the CDR3 compared to all three CDRs.

Feat. Engineering Method	Initial Dim.	Latent Dim.	Accuracy
2-tuple frequency	400	4	55.5%
2-tuple frequency	400	8	57.8%
3-tuple frequency	8000	4	55.5%
3-tuple frequency	8000	8	57.2%
Li et al.	70	4	56.8%
Li et al.	70	8	59.3%

Table 4.3: Best results of AE compression on applicable feature engineering methods. 14 long sequences are used for Li et al. methods. Subsequent classification was performed using an XGBoost classifier providing the accuracy metric.

Feat. Engineering Method	Initial Dim.	Latent Dim.	Accuracy
2-tuple frequency	1200	4	55.1%
2-tuple frequency	1200	8	59.6%
3-tuple frequency	24000	4	56.2%
3-tuple frequency	24000	8	57.6%
Li et al.	125	4	57.7%
Li et al.	125	8	60.9%

Table 4.4: Best results of AE compression on applicable feature engineering methods, all including CDR1 and CDR2. The 14,6,5 long sequences are used for Li et al. methods (CDR3, CDR2, CDR1 respectively). Subsequent classification was performed using an XGBoost classifier providing the accuracy metric.

Upon manual investigation it appeared that the AE was accurately compressing features related to the terminals of the sequences. For example, tuples found in V and J regions could be easily reconstructed but not those commonly found in the center. This suggests that after compression the classification was completely relying upon the V and J region associated features for separation. In order to better analyze this phenomenon we instead used an AE to learn to reconstruct sequences based only on their amino acids; the input/output being one-hot encoded vectors. In order to make the distinction clear this has been termed 'Feature Learning.'

4.3 Feature Learning

The results of AutoEncoding the complete dataset and subsequent classification accuracy on a validation set of 20% can be seen in Table 4.5. The

classification took place by encoding the data to the latent dimension and classifying upon that. Using a latent dimension of 2 could produce convincing recreations of the original input such as:

```
Input:  [CASSLLAGFDTQYF]
Output: [CASSAAGGGDTQYF]
```

What is interesting to note is that in most cases the network was able to reconstruct the C-terminal and N-terminal region of the sequence, the regions containing the most conserved V and J gene segments. This confirms what was seen using the compression based AE.

In the center the network tended to rely upon short repeats of several amino acids, namely glycine and alanine. This follows given their already noted commonality in tuples near the center of the sequence. It is likely that the network learned to mainly encode information from the two genes in the latent representation. Considering that the low classification error, it gives further merit to the idea that the best features will be derived from the center of the sequence. It is also interesting that the AE did not show any evidence of encoding the previously seen trend of amino acid preference by electrostatic charge.

Latent Dimension	Mean Square Error	Accuracy
2	1.91×10^{-4}	55.5%
4	1.79×10^{-4}	55.7%
8	1.28×10^{-4}	57.7%

Table 4.5: Results of AE learning a latent representation of one-hot encoded amino acids and classification accuracy using the latent representation. Classification was performed using an XGBoost classifier. Means square error is provided at convergence

The larger latent dimensions showed a better compression based on MSE as well as classification accuracy. Unfortunately this accuracy is still quite poor. However it does suggest that features can be learned that improve upon baseline i.e. random. The sequences reproduced using an 8D latent

representation still relied upon short stretches like the 2D however not as frequently.

In an effort to examine encoding only the center of the sequence, the same autoencoder architecture was trained upon subsets of the sequences. More specifically a window of 8 amino acids from the 2^{nd} to 10^{th} position for each sequence. The results are found in Table 4.6.

Latent Dimension	Mean Square Error	Accuracy
2	2.34×10^{-4}	55.9%
4	1.89×10^{-4}	57.0%
8	1.37×10^{-4}	57.2%

Table 4.6: Results of AE learning a latent representation of clipped 2:10 one-hot encoded amino acids and classification accuracy using the latent representation. Classification was performed using an XGBoost classifier. Mean square error is provided at convergence

As was expected the MSE was higher, reflecting that the Encoder did not have access to the amino acids determining V and J gene usage. The classification accuracy is similar suggesting that the V-J regions were not contributing noise, however this conclusion is tenuous given the already low accuracy. It also suggests that a significant amount of classification relevant information is located in the center of the sequence.

Although exploration has suggest preferential tuple usage, particularly in the N-terminal, it could be that the preferential amino acid usage noted in Section 4.1.4 contributes more to classification. This follows given the Mutual Information analysis suggested that the N-terminal region was contributing the least to classification. If this is the case any information encoded about preferential V gene usage will come from an explicit V gene feature.

4.4 Classification

This section covers the results of classification using the various feature engineering methods. Due to the success of the Li et al. method it was used to perform several exploratory classification experiments. For this reason it is presented first.

4.4.1 Li et al. Feature Engineering

The Li et al. feature engineering method performed the best of all feature engineering methods. Furthermore, the best performance came from using all three CDR sequences and including the V gene index as a feature. In the case of 14 long sequences this is a 126 dimensional vector. Using an XGBoost classifier led to the highest classification accuracy at 68.4% (see Table 4.7).

It is interesting that CDR1 and CDR2 are determined by the V region and their addition improved classification. As was shown during data exploration, the V gene associated features improve classification. This tends to suggest that providing more information related to the V gene improves classification. Future experiments should be performed to incorporate further V gene information, such as the family to which it belongs, and any pertinent biophysical characteristics.

Returning to the classification algorithms, across all variants of this feature engineering method the boosting based classifiers outperformed the others. The best classification result using the XGBoost classifier on sequences of different length was using 12 long sequences. This resulted in a classification accuracy of 69.7% (See Table C.4). Although XGBoost presented the best results, Adaboost results are almost comparable.

Another point of note is that the SVM kernel choice was optimized however the RBF always performed the best. An example is in Table C.5 using the basic Li method. This is also the case for the other feature engineering methods, so only the SVM-RBF results are included.

It is difficult to provide a precise reason as to why boosting was effective however there are a few potential reasons. From a machine learning perspective, boosting helps lower model bias. The ensemble of weak learners gives flexibility to the model structure so that it can bend to fit the data structure [115]. Being that this data contains a very complex structure this is one potential reason why boosting performed well.

It is also very easy to train weak learners on this dataset. K-NN by itself

Classifier	CDR3	CDR1/2/3	CDR1/2/3 + V gene	CDR3 + V gene
SVM-RBF	61.0%	63.8%	65.0%	61.5%
k-NN	56.0%	57.1%	57.3%	56.5%
NN	61.4%	64.6%	64.5%	62.0%
RNN	62.0%	62.3%	64.0%	62.8%
Adaboost	**62.2%***	**66.6%***	67.8%	64.2%
XGBoost	**62.2%***	66.3%	**68.4%***	**64.8%****

Table 4.7: Best validation accuracy of classification algorithms on Li et al. feature engineered 14 amino acid long sequences including variants of the original method.
$^{*} :< \pm 0.1$, $^{**} :< \pm 0.5$

could be called a weak learner given its results (Table 4.7). Empirically boosting has also shown great success in a variety of tasks. This includes data science competitions like Kaggle [90] and TCR based classification [116]. The latter case used Linear Programming Boosting (LPBoost) which should be tried in future experiments, along with other AdaBoost variants like MadaBoost which is more noise resistant [117].

More broadly, due to the relative success of the extended Li feature engineering method, it has been used to probe some portions of the dataset. This was to establish whether there were any other biases between populations that had not yet been seen. For example, if TCRα chains could be better classified than TCRβ or if different patients showed differences in population classification results.

4.4.1.1 Data Efficiency

The first of such inquiries was whether the dataset contained enough records to accurately accomplish the classification task using the Li et al. method. This done by running 3-fold cross validation 100 times for each 10% interval of the total dataset.

Figure 4.8 clearly shows that for both training accuracy and cross-validation accuracy, 10% of the data is sufficient to achieve the same performance as using the full dataset. This suggest two things.

The first is that whatever relevant information is available, from the Li et

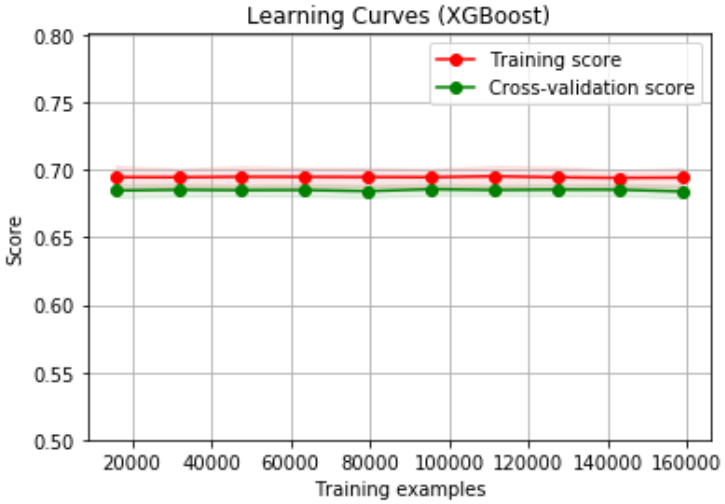


Figure 4.8: Learning curves for Li et al. method using CDR1/2/3 and V Gene for 14 long sequences, with an XGBoost classifier. Score is classification accuracy.

al. features, is learned by the classification algorithm from a small number of samples. This itself suggests that the contributing features are simple or have a strong signal.

This figure also suggests that the Li et al. method itself is not capable of achieving a strong classification performance, or rather that insufficient data is not a problem. Both of these points imply that improvement upon this work should focus on feature engineering with the current dataset, in contrast to testing more classification algorithms.

4.4.1.2 Difference in Patient Datasets

The next baselines that had to be established was if there was a significant difference in classification using patient populations. SMOTE was used in order to balance the more weighted patient datasets, and classification was performed using the CDR3 and V region with an XGBoost Classifier.

Classifier	Patient: HV1	Patient: HV3	Patient: HV2
XGBoost	63.6±0.6%	66.9±0.4%	65.1±0.1%

Table 4.8: Best validation accuracy of XGBoost on Li et al. feature engineered CDR3 sequences + V region using different patient datasets. All using 14 long sequences.

As can be seen in Table 4.9, there is some difference in the classification accuracy's of the patients. The most stable is HV2 which has the largest number of samples by far (14 long CDR3s), $\sim 150,00$ compared to $32,000$ and $29,000$ for HV3 and HV1 respectively. Taking the weighted mean gives an accuracy of 65.2% which, given the confidence, is comparable to the accuracy achieved using all patients, 64.8% (Table 4.7). All considered, this suggests there is no significant difference in patient population classification.

Biologically this is beneficial as it suggests that the conclusions drawn from the classification can be generalized, rather than just being patient specific. From a machine learning perspective these results do not indicate that classification is better off using a patient specific dataset.

4.4.1.3 Difference in TCRα and TCRβ Chains

Our data records also contained access to a dataset of TCRα chains in similar proportions to the TCRβ chains. Classification was ran on the TCRα CDR3 and V region for the complete dataset. This was done with the logic that if the TCRα showed an improvement in classification it could be used to identify relevant features in TCRβ.

Results are presented in Table 4.9. There appears to be no significant difference in classification between the two chains. Repeating this classification with different lengths and classifiers supported this point. Although it does not assist the current classification task it does suggest that any classifiers developed in the future for TCRβ could be generalized to TCRα.

Classifier	TCRα	TCRβ
XGBoost	65.3±0.2%	64.8±0.1%

Table 4.9: Best validation accuracy of XGBoost on Li et al. feature engineered CDR3 sequences + V region using TCRα and TCRβ.

4.4.1.4 Difference in CDR Sequences

Another difference to be established was the capability of classification algorithms to assign a TCR chain based soley on CDR1, CDR2, or the V re-

gion. Performing this classification also provides a better understanding as to why the addition of extra CDRs and V region features improve performance. The investigation was performed early in the study and so only uses the HV1 dataset. In the case of the V region this was classification using only a single feature, the V gene index.

Classifier	CDR3	CDR2	CDR1	V Region
XGBoost	62.2±0.2%	61.9±0.5%	61.4±0.3%	59.4±0.6%

Table 4.10: Best validation accuracy of XGBoost on Li et al. feature engineered CDR1, CDR2, and V region separately.

Interestingly all three CDR sequences achieved similar classification results, and the V gene being only slightly worse. This in and of itself suggests that the V gene is a valuable feature. This is further supported by the performance boost achieved when it is added to the standard Li et al. method. This is strange given that Li et al. claimed no influence of V and J genes on their classification results.

Given that the CDR1 and CDR2 are directly produced according to the V gene it is possible that beneficial features from the two CDRs are encoding information somehow related to the V gene. Taking into account the relatively high performance of the combination of CDRs and V gene, it is possible that the CDR1, CDR2, and V gene all encode different but valuable information regarding the V gene. This further supports the notion that other feature engineering work should aim to include more information regarding the V gene.

4.4.1.5 Boosting Feature Importance

Given the most effective classifier is XGBoost, the underlying decision tree boosted ensemble can be used to distinguish the feature importance values. In the case of the best variant (CDR1/2/3 + V gene) the V gene contributed 21% of total feature importance, the rest of which is spread over the amino acids of the CDRs. This spread, without the V gene feature, is included in Figure 4.9 where the importance for an amino acid is the sum of the five Atchley value

importance's.

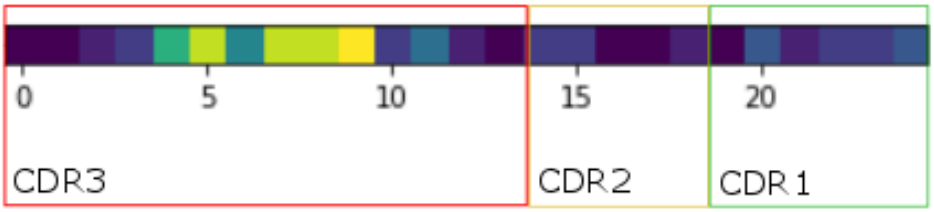


Figure 4.9: Feature Importance from Decision Tree Ensemble in XGBoost Classifier, for the best variant of the Li et al. feature extraction methods, summed for positional value information

As would be expected, the CDR3 commands the most importance and the CDR2 and CDR1 only provide slight assistance in separation. Interestingly the CDR3 importance is very similar to the Mutual Information as presented in Figure 4.7. This is another suggestion that the separation using CDR3 is hinging on the amino acid usage in the center, and by extension its small motif usage in the center.

A more biologically relevant way to view all of the feature importance's is to see the expanded version of Figure 4.9. This is presented in Figure 4.10 with the corresponding Atchley vector meanings.

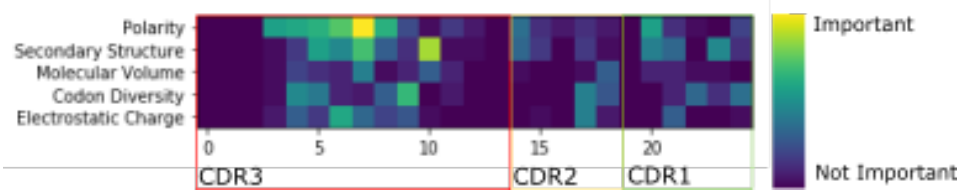


Figure 4.10: Feature Importance from Decision Tree Ensemble in XGBoost Classifier, for the best variant of the Li et al. feature extraction methods

The first notable aspect of this figure is that the first three amino acids show no importance. This would again suggest that any V related tuples are not influencing the classification. Any V related information is likely coming from the CDR1, CDR2, and the V gene index. If this is the case it would explain the performance increase noted with the addition of the extra features; it is encoding information not provided by the amino acids in the V determined N-terminal region.

This figure also suggests, as has been noted before, that the most valuable features aside from the V gene index come from the center of the CDR3. More specifically the polarity and less so the secondary structure are providing the most valuable information for classification. It was previously noted that CD4 and CD8 populations showed differential preference for charged amino acids however it appears that this is not as important compared to the two other mentioned attributes.

To ascertain how important the polarity of the amino acids was to separation, classification was performed using only the 25 polarity features from the three CDRs. This achieved a classification accuracy of $64.5 \pm 0.1\%$. This strengthens the suggestion that the polarity is one of the main contributing factors to the current separation performance.

From a biological perspective, the preference for amino acids by polarity and secondary structure likely lies in the difference in cleft geometry between MHC Class I and Class II. More specifically, that the CD4 TCRs and CD8 TCRs will prefer amino acids that make the CDR loop more conformationally favourable to binding the antigen-MHC complex. Further work is needed to investigate how this difference could leveraged for improved feature engineering.

Given that the XGBoost classifier assigned zero importance to several features, classification was performed again but on the top 50 and top 10 best features. This was done using an SVM-RBF with the logic that taking a non-linear combination of the best features could reduce noise and improve classification. Results are presented in Table 4.11.

Classifier	10 Best Feats.	50 Best Feats.	All Features
SVM-RBF	$59.2 \pm 0.1\%$	$60.0 \pm 0.8\%$	$68.4 \pm 0.5\%$

Table 4.11: Best validation accuracy of SVM-RBF on Li et al. feature engineering method including CDR1, CDR2, and V region, with 10 and 50 best features determined by XGBoost

Although these results show a decrease in classification accuracy, they

provide a valuable insight into the contribution of individual features. Notably, taking the combination of the ten best features is equivalent in performance to just using the V gene as a feature (Table 4.10). Especially given that the V gene is one of those best features. As was expected the other features originate from the polarity and secondary structure attributes of amino acids near the center of the sequence.

Using the best 50 features provides only a small increase beyond using the 10 best features which is still significantly lower than using all 126 (for 14 long sequences). Oddly it is also lower than using only the polarity features ($64.5 \pm 0.1\%$). This suggests that some features are creating noise. Thus the suggestion from these results is that the features are not powerful enough to achieve a better classification and are potentially being affected by some noise.

4.4.1.6 High Confidence Classifications

Using the CDR3 and V region we wished to establish which CDR3s were being confidently assigned to one population or another by the classifier. This was again with the purpose of understanding what features were driving the classification and if any specific type of sequence was being classified better than others.

This unfortunately was not possible due to the highest confidences being in the range of 50.8-51.0%. This in and of itself provides an insight into the classification. Primarily that there are no particularly distinguishing features and that all points sit relatively close to the separating hyperplane. This concept is readily apparent in the t-SNE visualizations as well.

This is inline with what has been seen when examining the contribution of individual CDRs, V gene, and Atchley vector components. All of the 'stronger' features like polarity and the V gene by themselves could achieve reasonable performance but combining them only provided a small boost. This suggesting that none of the features are actually contributing heavily to the separation.

More broadly, this suggests that aside from V gene influence the separation is hinging off a weak preference for amino acids in the center of the CDR3. This

is primarily by polarity and secondarily by charge and secondary structure. Biologically this suggests that the energy of the antigen interaction is the key aspect differentiating CD4 and CD8 populations. This energy of course depending on the conformation and ability to interact between the CDR3 and bound antigen. Assuming this is the case, it is clear that the Li et al. method is not capable of capturing this interaction well enough to achieve a high accuracy. This being the case future feature engineering work should focus on different approaches, such as using Dynamic Bayesian Networks.

4.4.1.7 Comparison with Li et al.

All of the previously discussed work thoroughly explores classification using the Li et al. method however it does not address the large performance difference between this work and that reported by Li et al. [11]. Using both their feature engineering method (only CDR3) and classification algorithm there is more than a 30% difference in classification accuracy. A direct comparison is provided in Table 4.12.

	Our Work	Li et al.
Accuracy	61.0±0.1%	93±0.01%
Classifier	SVM-RBF	SVM-RBF
Feat. Eng.	CDR3	CDR3
Total Records	924896	302899
Class Balance (CD4:CD8)	∼50:50	∼90:10

Table 4.12: Comparison of classification between Li et al. and our work. Class balances are provided for the specific sequence lengths i.e. 14 long.

One of the most notable differences is the class balance. Where this work has an even class balance Li et al. have a class balance heavily skewed to CD4 sequences. For 14 long CDR3s this is particularly bad. From their work, it does not appear they used class balancing methods whatsoever. This raises the question of whether their results are artificially high due to a lack of class balancing. This would also explain why our dataset is more than three times as large and still has a lower accuracy.

Unfortunately they have not made their code available but in the methods

they described the exact SVM classifier function used from the Sci-Kit Learn library. According to current and previous documentation, class balancing is not done by default for this function. Furthermore, only accuracy is provided as a metric and not recall or precision which would identify this problem. Given that their data could not be extracted, the ability to benchmark is left as an open question for future work.

4.4.2 Protein Embeddings

Before the using the ProtVec method for classification the numericized dataset was visualized with t-SNE to see if the clusters were unique to the Li et al. feature engineering method. As can be seen in Figure 4.11 not only are the clusters present but they are also more clearly visible. This suggests that this feature engineering method will also suffer from having to classify within the V-J clusters.

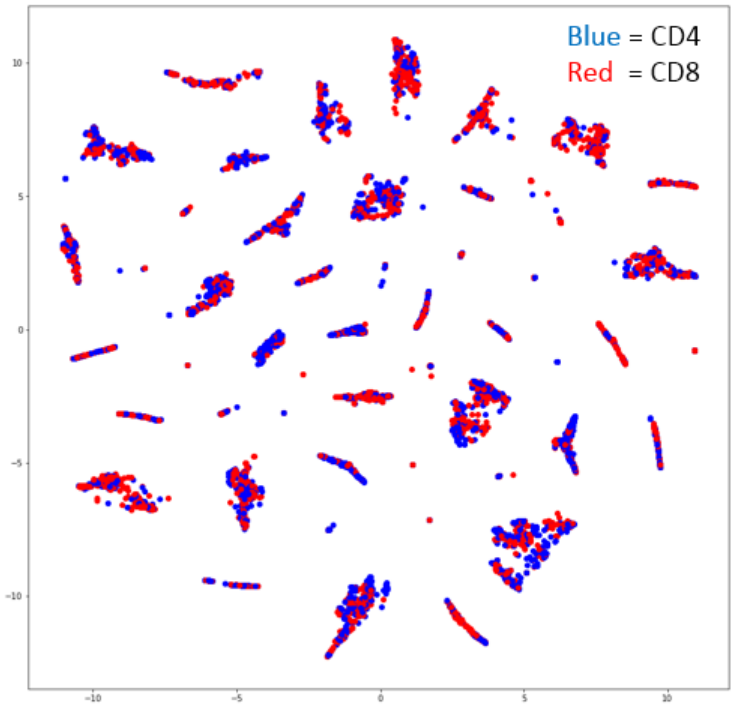


Figure 4.11: t-SNE visualization of complete dataset using the custom ProtVec embeddings feature engineering method.

The results of the different classifiers has been included in Table C.1 and Table C.2 for the SwissProt trained and custom embeddings respectively. Clas-

sification on the SwissProt data showed similar performance for the various classifiers, all being between 59%-60% accuracy on the same validation set. The custom trained protein embeddings were also in the same accuracy range. Given the similarity of results between classifiers there are no implications as to which is more suited to this task.

4.4.3 p-Tuple Methods

As can be seen in Table 4.13 the tuple methods showed similar performance to the Li et al. method albeit with lower accuracy scores across the board, by $\sim 1\%$ averaged across all classifiers. Generally the $p = 2$ method was more successful than the $p = 3$. However the best method, though by a slim margin, is using the all CDRs with an Adaboost classifier for $p = 3$. This resulted in a classification accuracy of 64.9%.

Classifier	p-Value	Best Params.	CDR3 Acc.	CDR1/2/3 Acc.
SVM-RBF	2	$C = 10$, $\gamma = 0.001$	59.7%	64.7%
SVM-RBF	3	$C = 10$, $\gamma = 0.001$	58.7%	64.2%
k-NN	2	$k = 5$	55.0%	60.8%
k-NN	3	$k = 5$	53.4%	60.7%
Adaboost	2	Est= 100	**60.5%****	64.7%
Adaboost	3	Est= 100	58.2%	**64.9%****
XGBoost	2	Est= 50, $L_2 = 5$	59.8%	64.8%
XGBoost	3	Est= 50, $L_2 = 5$	58.3%	63.8%

Table 4.13: Best results of classification algorithms on 2/3-tuple frequency feature engineered sequences with and without CDR1 and CDR2 sequences. * :< ±0.1, ** :< ±0.5

4.4.4 Amino Acid Positional Probability

The amino acid positional probability method performed better than the p-tuple method however it did still did not perform as well as the Li et al. method. Again the boosting classifiers performed the best and the CDR combination was better than the CDR3 alone. The best classification accuracy achieved is 65.6% using an Adaboost classifier (See Table 4.14). This result places the positional probability model as the second best method.

Classifier	Best Params.	CDR3 Acc.	CDR1/2/3 Acc.
SVM-RBF	$C = 10$, $\gamma = 0.001$	56.3%	60.7%
NN	Batch= 64, $\eta = 0.005$	56.8%	58.6%
k-NN	$k = 5$	58.9%	60.6%
Adaboost	Est= 100	61.8%	**65.6%***
XGBoost	Est= 50, $L_2 = 5$	**62.0%****	65.5%

Table 4.14: Best results of classification algorithms on amino acid positional probability feature engineered sequences with and without CDR1 and CDR2 sequences.
$^{*} :< \pm 0.1$, $^{**} :< \pm 0.5$

4.4.5 Overall Classification

Being that the best classification took place with the Li et al. method, the next step was to combine it with the second best method which was the amino acid positional probability. Unfortunately this showed no improvement over the Li et al. method by itself. It is likely that this lack of improvement is because both methods are encoding the same information.

If this is the case it would suggest that separation is occurring because of preferential amino acid usage. This also supports that CDRs will prefer amino acids that assist in lowering the conformational requirements. If this is the case it also explains why the Li et al. method of using Atchley vectors worked the best; it more explicitly encoded the attributes from each amino acid that relate to potential conformational preference e.g. polarity, secondary structure, and charge. This is in contrast to the p-tuple method and protein embeddings which focused more on combinations of several amino acids.

	Precision	Recall	f_1-score
CD4	67%	56%	62%
CD8	69%	77%	73%
Average	68%	67%	67%

Table 4.15: Metrics for classification of the test set using the best classifier and feature engineering method: Li et al with all variants and an XGBoost classifier.

This being the case, the Li et al. method is considered the best result achieved in this work, with a validation accuracy of 68.4%. When applied to

the test set it achieved a final accuracy of **67.7%**. This suggests that there was no significant overfitting. Additional metrics for the classification have been included in Table 4.15. Both the recall and f_1-score are better for CD8s suggesting that the CD8s were more easily identified.

Chapter 5

General Conclusions

The main aim of this work was investigate whether the CD4/CD8 origin of TCRβ sequences could be predicted using a machine learning classification algorithm. In doing so we wished to reaffirm the conclusions drawn by Li et al. [8] on our own data set. These were that CDR3s could be classified with greater than an 80% accuracy by replacing the amino acids with Atchley vectors and using an SVM-RBF classifier.

This developed into two main tasks to achieve this overall classification goal. Creating a successful feature engineering method, and choosing an appropriate classification algorithm. Four primary feature engineering approaches were explored, of which the most successful was the Li et al. method using the three concatenated CDRs combined with the V Gene index. Using an XGBoost classifier, this achieved 67.7% test set accuracy.

It was found that across all feature engineering methods the boosting classification algorithms were the best suited. This is likely being due to the ensemble nature of boosting as well as its empirical success in other classification tasks. This is interesting to note given the recent success of linear programming boosting (LPBoost) in a binary classification using TCRs [116]. Future methods should investigate other types of boosting classifiers like LPBoost, or Adaboost variants like MadaBoost [117].

Unfortunately the performance achieved is much less than the reported 93% validation accuracy (14 long sequences) reported by Li et al. who only

used the CDR3 sequences. What is interesting to note is that Li had an overall class balance of 80% CD4 and 20% CD8, and noted no effect by V Gene usage on classification. We were also unable to determine their class balancing method, if any. Furthermore, they do not appear to have used a test set as only validation accuracy is provided.

In contrast to Li et al, much of the feature engineering and feature importance techniques suggested that the V gene was the most powerful of all features used. Interestingly much of the information common between sequences was conserved motifs from V and J genes. When visualized these appeared as clusters of sequences, each corresponding to a unique V-J combination. Within these clusters existed more clusters of sequences. These usually contained CDR3s with similar short motifs in the center of the sequence. These are potentially specificity groups as has been suggested recently in similar clustering investigations [51, 50].

Due to the commonality of the V-J derived N,C-terminals the center of the CDR3 sequence contained the most variability and thus information available for classification. This was supported by feature importance investigations using Mutual Information, AutoEncoding, and Boosting. It appeared to be the case even when CDR1 and CDR2 were included although they did provide a small improvement.

It appeared that this variable center region contributed to separation using the Li et al. method through preferential amino acid usage between the two populations. Primarily by amino acid polarity but also by charge and secondary structure.

This in itself is a weak way to classify and so suggests that to improve performance largely different feature engineering approaches should be explored. For example using Dynamic Bayesian Networks, which has seen success in protein classification [118]. Alternatively focus on analysis of the V gene as both the V gene index and CDR1/CDR2, which are determined by the V gene, improved classification accuracy when added as features.

Another approach this work did not explore is the combination of naïve biophysical metrics, like those used in a previous work [12], with the local sequence based approaches used in this investigation. This is also left for future work. Furthermore, this work did not explore methods of normalizing the sequences by removing the conserved V and J regions present in CDR3s. This is left to future work, the logic in doing so is that it would provide a stronger conclusion as to the nature of amino acid usage in the center. It could also reduce noise.

Two other approaches also lend themselves to future work. The first is using the recently published approach for judging the value of individual tuples counts as features using a one-dimensional Naïve Bayes classifier [64]. The second is combining CDR3s from paired TCRα and TCRβ chains and classifying upon the the concatenated sequences. Unfortunately, there are no datasets currently available with significant records from both CD4 and CD8, to our knowledge.

Regardless of the classification performance, a 67% classification accuracy indicates there is some distinct difference between the two populations. Aside from preferential V gene usage, the fact that the most important features originate from the center of the CDR3β suggests that this difference lies in the energy of the interaction with the antigen.

Appendix A

Data Exploration

Patient:	EG10	SK11	KS07	Complete
CD4 Sequences	25210	36642	139176	175331
CD8 Sequences	30298	21725	125895	156850
Duplicates	2912	2858	29102	39881
Total Sequences[1]	55508	58367	265071	332181
CD4 to CD8 Ratio	45:55	63:37	53:47	52:48

Table A.1: General statistics for all patient CD4 and CD8 TCRα populations

Appendix B

Methods

Layer	Units	Activation Function
Dense	64	sigmoid
Dense	32	sigmoid
Dense	8	sigmoid
Dense	32	sigmoid
Dense	64	sigmoid
Dense	70	linear

Table B.1: Structure of Li et al. compression AE, were final output units were altered to match input. Uses 14 long CDR3s as an example.

Layer	Units	Activation Function
Dense	256	softplus
Dense	128	softplus
Dense	32	softplus
Dense	8	softplus
Dense	32	softplus
Dense	128	softplus
Dense	256	softplus
Dense	280	softplus

Table B.2: Structure of VAE for feature learning, were final output units were altered to match input

The following is a list of the Scikit-Learn classifiers and their parameters that were not optimized.

1. k-Nearest Neighbours
   (a) algorithm = auto

Layer	Units	Activation Function
Dense	512	ReLU
Dense	256	ReLU
Dense	128	ReLU
Dense	32	ReLU
Dense	2	softmax

Table B.3: Structure of standard five layer classification neural network for all feature methods with an input dimension greater than 400 dimensions

Layer	Units	Activation Function
Dense	256	ReLU
Dense	128	ReLU
Dense	64	ReLU
Dense	32	ReLU
Dense	2	softmax

Table B.4: Structure of standard five layer classification neural network for all feature methods with an input dimension less than 400 dimensions

(b) leaf size = 30

(c) distance metric = minkowski

(d) number of threads = 3

(e) weights = balanced

2. Support Vector Machine

(a) Adam $\beta_1 = 0.9$

(b) Adam $\beta_2 = 0.999$

(c) Adam $\epsilon = 1 \times 10^{-8}$

(d) Adam decay, reduce by $= 1 \times 10^{-6}$ every batch

3. Support Vector Machine

(a) class weights = balanced

(b) shrinking = True

(c) tolerance = 0.001

(d) decision function = One-Versus-Rest

4. Adaboost

(a) Algorithm = SAMME.R

Layer	Units	Activation Function
Dense	128	softplus
Dense	256	softplus
Dense	512	softplus
Dense	280	softmax *

Table B.5: Structure of InfoGAN Generator network, were final output units were altered to match input. Softmax was applied across every 20 units of the output to emulate the one-hot encoding of the input.

Layer	Units	Activation Function
Dense	128	softplus
Dense	256	softplus
Dense	512	softplus
Dense	1/2	sigmoid/softmax

Table B.6: Structure of the InfoGAN Discriminator/Q network.Final layer goes Discriminator/Q

(b) Base Estimator = Decision Trees

(c) learning rate = 1

5. XGboost

(a) Base score = 0.5

(b) gamma = 0

(c) learning rate = 0.1

(d) max. delta step = None

(e) maximum depth = 3

(f) objective = Binary-Logistic

(g) L_1 Regularization = 0

Appendix C

Results

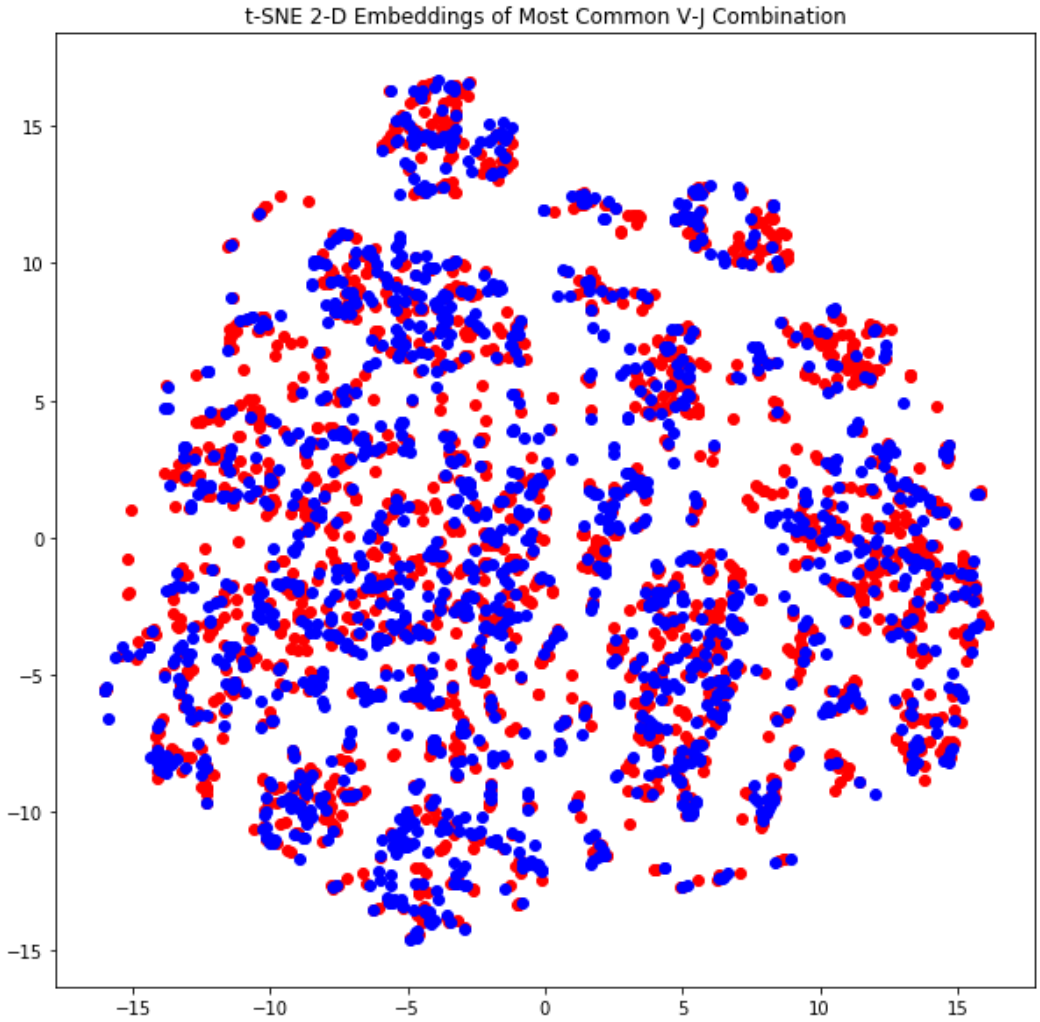


Figure C.1: t-SNE visualization of sequences using the most common V-J gene combination.

CD4, p=1	CD8, p=1	CD4, p=2	CD8, p=2	CD4, p=3	CD8, p=3
['S' '116040']	['S' '148365']	['CA' '59428']	['CA' '44839']	['CAS' '40501']	['CAS' '53065']
['F' '78288']	['F' '102848']	['AS' '55688']	['AS' '43010']	['ASS' '33743']	['ASS' '46222']
['A' '78162']	['A' '99065']	['SS' '53366']	['SS' '39159']	['QYF' '22218']	['QYF' '29119']
['G' '69653']	['G' '84602']	['YF' '30222']	['YF' '23608']	['TQY' '11561']	['TQY' '13997']
['C' '53956']	['C' '68651']	['QY' '29394']	['QY' '22398']	['SSL' '9350']	['EQY' '13926']
['Q' '52980']	['Q' '67877']	['FF' '26890']	['FF' '19217']	['EQY' '9334']	['SSL' '12811']
['T' '50938']	['T' '62005']	['EQ' '26743']	['EQ' '17410']	['QFF' '7979']	['QFF' '12635']
['Y' '45831']	['Y' '60879']	['TQ' '14521']	['TQ' '12114']	['EQF' '7767']	['EQF' '12429']
['E' '40722']	['E' '59242']	['SL' '14077']	['GG' '10579']	['YEQ' '7004']	['YEQ' '10607']
['R' '32186']	['L' '38250']]	['GG' '12890']	['SL' '10317']	['DTQ' '6786']]	['NEQ' '9871']

CD4, p=4	CD8, p=4	CD4, p=5	CD8, p=5	CD4, p=7
['CASS' '33507']	['CASS' '45925']	['CASSL' '9118']	['CASSL' '12457']	['STDTQYF' '1454']
['TQYF' '11554']	['TQYF' '13982']	['YEQYF' '6789']	['YEQYF' '10353']	['SYNEQFF' '1184']
['EQYF' '9331']	['EQYF' '13917']	['DTQYF' '6768']	['NEQFF' '9457']	['NTGELFF' '928']
['ASSL' '9125']	['ASSL' '12462']	['NEQFF' '5872']	['DTQYF' '8890']	
['EQFF' '7766']	['EQFF' '12425']	['CASSP' '5059']	['TEAFF' '6856']	CD8, p=7
['YEQY' '6789']	['YEQY' '10353']	['TEAFF' '4901']	['CASSQ' '6779']	['SYNEQFF' '1950']
['DTQY' '6768']	['NEQF' '9458']	['TDTQY' '4220']	['CASSP' '6055']	['STDTQYF' '1463']
['NEQF' '5872']	['DTQY' '8890']	['ETQYF' '3939']	['TDTQY' '5838']	['NTGELFF' '1049']
['EAFF' '5823']	['EAFF' '8403']			
['ASSP' '5070']]	['TEAF' '6856']]			

Figure C.2: Counts of the most common p-tuples of different sizes for both CD4 and CD8 populations in the single patient dataset

Amino Acid Usage in CD4/CD8 Globally:

CD4:			CD8:		
	['S'	15.2%]		['S'	15.4%]
	['F'	10.3%]		['F'	10.7%]
	['A'	10.2%]		['A'	10.3%]
	['G'	9.1%]		['G'	8.8%]
	['C'	7.1%]		['C'	7.1%]
	['Q'	6.9%]		['Q'	7.1%]
	['T'	6.7%]		['T'	6.5%]
	['Y'	6.0%]		['Y'	6.3%]
	['E'	5.3%]		['E'	6.2%]
	['R'	**4.2%]**		['L'	4.0%]
	['L'	4.0%]		**['D'**	**3.3%]**
	['P'	3.3%]		['P'	3.0%]
	['N'	3.0%]		**['R'**	**3.0%]**
	['D'	**2.6%]**		['N'	2.8%]
	['V'	2.0%]		['V'	1.9%]
	['H'	1.3%]		['H'	1.1%]
	['I'	1.1%]		['I'	0.9%]
	['K'	0.9%]		['K'	0.6%]
	['W'	0.5%]		['W'	0.6%]
	['M'	0.3%]		['M'	0.2%]

Figure C.3: Normalized Counts of amino acids across CD4 and CD8 populations for the complete dataset

Classifier	Best Parameters	Test Accuracy
SVM-RBF	$C = 1$, $\gamma = 0.01$	60.0%
NN	Batch= 256, $\eta = 0.01$	59.8%
RNN	Batch= 64, $\eta = 0.001$	58.0%
Adaboost	Estimator= 100	59.7%
XGBoost	Estimator= 80, $L_2 = 1$	60.2%

Table C.1: Best results of classification algorithms on SwissProt protein embeddings feature engineered sequences.

Classifier	Best Parameters	Test Accuracy
SVM-RBF	$C = 10$, $\gamma = 0.01$	59.8%
NN	Batch= 128, $\eta = 0.01$	59.6%
RNN	Batch= 128, $\eta = 0.0001$	58.9%
Adaboost	Estimator= 50	59.3%
XGBoost	Estimator= 50, $L_2 = 1.5$	59.8%

Table C.2: Best results of classification algorithms on custom protein embeddings feature engineered sequences.

Classifier	Best Parameters
SVM-RBF	$C = 10$, $\gamma = 0.001$
k-NN	$k = 5$
NN	Batch= 64, $\eta = 0.001$
RNN	Batch= 256, $\eta = 0.0001$
Adaboost	Estimator= 120
XGBoost	Estimator= 100, $L_2 = 0.9$

Table C.3: Best general parameters of classification algorithms on Li et al. feature engineered 14 amino acid long sequences.

Length	Test Accuracy
12	**69.7%**
13	68.1%
14	68.4%
15	67.1%
16	67.5%

Table C.4: Best results of classification using Li et al. feature engineered sequences with CDR1, CDR2, CDR3, and V gene for the six most common lengths, using the best classifier: XGBoost

Kernel	Best Parameters	Test Accuracy
Linear	$C = 10$, $\gamma = 0.005$	59.6%
Polynomial, $n = 3$	$C = 1$, $\gamma = 0.05$	58.8%
RBF	$C = 10$, $\gamma = 0.001$	60.5%

Table C.5: Best results of classification using Li et al. feature engineered sequences with CDR3 for the three different SVM kernels, for 14 long sequences.

Classifier	Best Parameters
SVM-RBF	$C = 1$, $\gamma = 0.001$
k-NN	$k = 5$
NN	Batch= 128, $\eta = 0.005$
Adaboost	Estimator= 100
XGBoost	Estimator= 100, $L_2 = 2$

Table C.6: Best general parameters of classification algorithms for the amino acid positional probability feature engineering method

Appendix D

Colophon

This document was set in the Modern Latin typeface using LaTeX and BibTeX, and composed with the ShareLatex[1] editor. The text is double spaced and uses the IEEE citation style. The citations and papers were managed using the Mendeley[2] application.

The LaTeX code used for this report was written using the git[3] revision control system. In doing so the code is published and maintained in a public repository[4] using the GitHub service. The structure is based on the 'UCL Thesis Template' as per the license[5].

[1] https://www.sharelatex.com/
[2] https://www.mendeley.com/
[3] https://git-scm.com/
[4] https://github.com/Groovy-Dragon/COMPGI99
[5] https://www.sharelatex.com/templates/thesis/university-college-london-thesis

Bibliography

[1] J Lewis M. Raff K. Roberts B Alberts, A Johnson and P. Walter. Molecular biology of the cell, 6th edition by B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. In *Biochemistry and Molecular Biology Education*, volume 36, page 171. 2015.

[2] CA Jr Janeway, P Travers, and M Walport. Immunobiology: The Immune System in Health and Disease. 5th edition. *Garland Science*, 2001.

[3] Nghia Ho. RBM AUTOENCODERS, 2012.

[4] G Bradski. OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Dejan TanikicÌĄ and Vladimir Despotovic. Artificial intelligence techniques for modelling of temperature in the metal cutting process. In *Metallurgy-Advances in Materials and Processes*. InTech, 2012.

[6] WildML. 1.Recurrent Neural Networks Tutorial, Part 1 âĂŞ Introduction to RNNs âĂŞ WildML. *Wildml.Com*, pages 1–7, 2015.

[7] Niclas Thomas, Katharine Best, Mattia Cinelli, Shlomit Reich-Zeliger, Hilah Gal, Eric Shifrut, Asaf Madi, Nir Friedman, John Shawe-Taylor, and Benny Chain. Tracking global changes induced in the CD4 T-cell receptor repertoire by immunization with a complex antigen using short stretches of CDR3 protein sequence. *Bioinformatics*, 30(22):3181–3188, 2014.

[8] Hoi Ming Li, Toyoko Hiroi, Yongqing Zhang, Alvin Shi, Guobing Chen, Supriyo De, E Jeffrey Metter, William H Wood, Alexei Sharov, Joshua D Milner, Kevin G Becker, Ming Zhan, and N.-p. Weng. TCR repertoire of CD4+ and CD8+ T cells is distinct in richness, distribution, and CDR3 amino acid composition. *Journal of Leukocyte Biology*, 99(March):0215–071, 2015.

[9] Michaela Sharpe and Natalie Mount. Genetically modified T cells in cancer therapy: opportunities and challenges. *Disease models & mechanisms*, 8(4):337–50, 2015.

[10] James M. Heather, Mazlina Ismail, Theres Oakes, and Benny Chain. High-throughput sequencing of the T-cell receptor repertoire: pitfalls and opportunities. *Briefings in Bioinformatics*, page bbw138, 2017.

[11] Hoi Ming Li, Toyoko Hiroi, Yongqing Zhang, Alvin Shi, Guobing Chen, Supriyo De, E Jeffrey Metter, William H Wood, Alexei Sharov, Joshua D Milner, Kevin G Becker, Ming Zhan, and N.-p. Weng. TCR repertoire of CD4+ and CD8+ T cells is distinct in richness, distribution, and CDR3 amino acid composition. *Journal of Leukocyte Biology*, 99(March):0215–071, 2015.

[12] Athanasios Schoinas. *Classification of CD4 / CD8 T-Cells using T-Cell receptor sequence definitions*. PhD thesis, University College London, 2015.

[13] Paul Grasso. *Essentials of Pathology for Toxicologists*. CRC Press, 2003.

[14] C. A. Janeway. Approaching the asymptote? Evolution and revolution in immunology. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume 54, pages 1–13, 1989.

[15] Polly Matzinger. Tolerance, Danger, and the Extended Family. *Annu. Rev. Immunol*, 12:991–1045, 1994.

[16] Travers P. Walport M. & Shlomchik M. Janeway, C. Immunobiology: The Immune System In Health And Disease. In *Immunobiology: The Immune System In Health And Disease*, page 892. 2001.

[17] US National Library of Medicine. Definition: Antigen, 2015.

[18] J. Hopkins, A. Maton, W. M. Charles, J. Susan, Q. W. Maryanna, L. David, and D. W. Jill. *Human Biology and Health*. 1993.

[19] Ronald N Germain. T-cell development and the CD4-CD8 lineage decision. *Nature reviews. Immunology*, 2(5):309–322, 2002.

[20] Andrew K. Sewell. Why must T cells be cross-reactive? *Nature Publishing Group*, 12(9):669–677, 2012.

[21] Thomas J Kindt, Richard A Goldsby, Barbara A Osborne, and Janis Kuby. *Immunology*, volume 5. 2003.

[22] Nobel Media AB. "The Nobel Prize in Physiology or Medicine 1987", 1987.

[23] Harlan S Robins, Santosh K Srivastava, Paulo V Campregher, Cameron J Turtle, Jessica Andriesen, Stanley R Riddell, Christopher S Carlson, and Edus H Warren. Overlap and effective size of the human CD8+ T cell receptor repertoire. *Science translational medicine*, 2(47):47ra64, 2010.

[24] Mark M. Davis and Pamela J. Bjorkman. T-cell antigen receptor genes and T-cell recognition. *Nature*, 334(6181):395–402, 1988.

[25] T P Arstila, A Casrouge, V Baron, J Even, J Kanellopoulos, P Kourilsky, F. W. Alt, B. Arden, S. P. Clark, D. Kabelitz, T. W. Mak, L. Rowen, B. F. Koop, L. Hood, M. M. Davis, P. J. Bjorkman, M. M. Davis, C. Pannetier, C. Pannetier, J. Even, P. Kourilsky, W. M. C. Rosenberg, P. A. H. Moss, J. I. Bell, M. A. Hall, J. S. Lanchbury, T. Nanki, H. Kohsaka, N. Miyasaka, U. G. Wagner, K. Koetz, C. M. Weyand, J. J. Goronzy,

E. C. Dudley, H. T. Petrie, L. M. Shah, M. J. Owen, A. Hayday, H. von Boehmer, H. J. Fehling, C. Benit, B. Lucas, F. Vasseur, H.-M. Rodewald, H. J. Fehling, L. Gapin, C. Trigueros, P. Borgulya, H. Kishi, Y. Uematsu, H. von Boehmer, C. J. MacMahan, P. J. Fink, E. Padovan, G. J. V. Nossal, J. D. Altman, D. H. Busch, I. Pilip, E. G. Pamer, E. A. Butz, M. J. Bevan, V. P. Argaet, P. J. Lehner, P. A. H. Moss, C. Tanchot, F. A. Lemonnier, B. Pérarnau, A. A. Freitas, B. Rocha, C. Tanchot, B. Rocha, S. P. Berzins, R. L. Boyd, J. F. A. P. Miller, and D. C. Douek. A direct estimate of the human alphabeta T cell receptor diversity. *Science (New York, N.Y.)*, 286(5441):958–61, 1999.

[26] Qian Qi, Yi Liu, Yong Cheng, Jacob Glanville, David Zhang, Ji-Yeun Lee, Richard A Olshen, Cornelia M Weyand, Scott D Boyd, and JÃűrg J Goronzy. Diversity and clonal selection in the human T-cell repertoire. *Proceedings of the National Academy of Sciences of the United States of America*, 111(36):13139–44, 2014.

[27] RenÃľ L. Warren, J. Douglas Freeman, Thomas Zeng, Gina Choe, Sarah Munro, Richard Moore, John R. Webb, and Robert A. Holt. Exhaustive T-cell repertoire sequencing of human peripheral blood samples reveals signatures of antigen selection and a directly measured repertoire size of at least 1 million clonotypes. *Genome Research*, 21(5):790–797, 2011.

[28] Robert A. Holt and Steven J M Jones. The new paradigm of flow cell sequencing. *Genome Research*, 18(6):839–846, 2008.

[29] J. Douglas Freeman, RenÃľ L. Warren, John R. Webb, Brad H. Nelson, and Robert A. Holt. Profiling the T-cell receptor beta-chain repertoire by massively parallel sequencing. *Genome Research*, 19(10):1817–1824, 2009.

[30] Paul L. Klarenbeek, Paul P. Tak, Barbera D C van Schaik, Aeilko H. Zwinderman, Marja E. Jakobs, Zhuoli Zhang, Antoine H C van Kam-

pen, RenÃĬ A W van Lier, Frank Baas, and Niek de Vries. Human T-cell memory consists mainly of unexpanded clones. *Immunology Letters*, 133(1):42–48, 2010.

[31] Wilfred Ndifon, Hilah Gal, Eric Shifrut, Rina Aharoni, Nissan Yissachar, Nir Waysbort, Shlomit Reich-Zeliger, Ruth Arnon, and Nir Friedman. Chromatin conformation governs T-cell receptor Jβ gene segment usage. *Proceedings of the National Academy of Sciences of the United States of America*, 109(39):15865–70, 2012.

[32] Chunlin Wang, Catherine M Sanders, Qunying Yang, Harry W Schroeder, Elijah Wang, Farbod Babrzadeh, Baback Gharizadeh, Richard M Myers, James R Hudson, Ronald W Davis, and Jian Han. High throughput sequencing reveals a complex pattern of dynamic inter-relationships among human T cell subsets. *Proceedings of the National Academy of Sciences of the United States of America*, 107(4):1518–23, 2010.

[33] Paul D. Baum, Vanessa Venturi, and David A. Price. Wrestling with the repertoire: The promise and perils of next generation sequencing for antigen receptors, 2012.

[34] Hanna IJspeert, Pauline A. van Schouwenburg, David van Zessen, Ingrid Pico-Knijnenburg, Andrew P. Stubbs, and Mirjam van der Burg. Antigen Receptor Galaxy: A User-Friendly, Web-Based Tool for Analysis and Visualization of T and B Cell Receptor Repertoire Data. *The Journal of Immunology*, page 1601921, 2017.

[35] Eltaf Alamyar, Patrice Duroux, Marie Paule Lefranc, and VÃĬronique Giudicelli. IMGT® tools for the nucleotide analysis of immunoglobulin (IG) and t cell receptor (TR) V-(D)-J repertoires, polymorphisms, and IG mutations: IMGT/V-QUEST and IMGT/HighV-QUEST for NGS. *Methods in Molecular Biology*, 882:569–604, 2012.

[36] Niclas Thomas, James Heather, Wilfred Ndifon, John Shawe-Taylor, and Benjamin Chain. Decombinator: A tool for fast, efficient gene assignment in T-cell receptor sequences using a finite state machine. *Bioinformatics*, 29(5):542–550, 2013.

[37] RenÃľ L. Warren, Brad H. Nelson, and Robert A. Holt. Profiling model T-cell metagenomes with short reads. *Bioinformatics*, 25(4):458–464, 2009.

[38] Dmitriy A Bolotin, Mikhail Shugay, Ilgar Z Mamedov, Ekaterina V Putintseva, Maria A Turchaninova, Ivan V Zvyagin, Olga V Britanova, and Dmitriy M Chudakov. MiTCR: software for T-cell receptor sequencing data analysis. *Nat Meth*, 10(9):813–814, 9 2013.

[39] Dmitriy A Bolotin, Stanislav Poslavsky, Igor Mitrophanov, Mikhail Shugay, Ilgar Z Mamedov, Ekaterina V Putintseva, and Dmitriy M Chudakov. MiXCR: software for comprehensive adaptive immunity profiling. *Nature Methods*, 12(5):380–381, 2015.

[40] Leon Kuchenbecker, Mikalai Nienen, Jochen Hecht, Avidan U. Neumann, Nina Babel, Knut Reinert, and Peter N. Robinson. IMSEQ-A fast and error aware approach to immunogenetic sequence analysis. *Bioinformatics*, 31(18):2963–2971, 2014.

[41] Mikhail Shugay, Olga V Britanova, Ekaterina M Merzlyak, Maria a Turchaninova, Ilgar Z Mamedov, Timur R Tuganbaev, Dmitriy a Bolotin, Dmitry B Staroverov, Ekaterina V Putintseva, Karla Plevova, Carsten Linnemann, Dmitriy Shagin, Sarka Pospisilova, Sergey Lukyanov, Ton N Schumacher, and Dmitriy M Chudakov. Towards error-free profiling of immune repertoires. *Nature methods*, 11(6):653–5, 2014.

[42] Xi Yang, Di Liu, Na Lv, Fangqing Zhao, Fei Liu, Jing Zou, Yan Chen, Xue Xiao, Jun Wu, Peipei Liu, Jing Gao, Yongfei Hu, Yi Shi, Jun Liu, Ruifen Zhang, Chen Chen, Juncai Ma, George F. Gao, and Baoli Zhu.

TCRklass: a new K-string-based algorithm for human and mouse TCR repertoire characterization. *Journal of Immunology*, 194(1):446–54, 2015.

[43] Bram Gerritsen, Aridaman Pandit, Arno C. Andeweg, and Rob J. De Boer. RTCR: A pipeline for complete and accurate recovery of T cell repertoires from high throughput sequencing data. *Bioinformatics*, 32(20):3098–3106, 2016.

[44] Jason A. Vander Heiden, Gur Yaari, Mohamed Uduman, Joel N H Stern, Kevin C. O'connor, David A. Hafler, Francois Vigneault, and Steven H. Kleinstein. PRESTO: A toolkit for processing high-throughput sequencing raw reads of lymphocyte receptor repertoires. *Bioinformatics*, 30(13):1930–1932, 2014.

[45] Jian Ye, Ning Ma, Thomas L Madden, and James M Ostell. IgBLAST: an immunoglobulin variable domain sequence analysis tool. *Nucleic acids research*, 41(W1):W34–W40, 2013.

[46] Mathieu Giraud, MikaÃńl Salson, Marc Duez, CÃľline Villenet, Sabine Quief, AurÃľlie Caillault, Nathalie Grardel, Christophe Roumier, Claude Preudhomme, and Martin Figeac. Fast multiclonal clusterization of V (D) J recombinations from high-throughput sequencing. *BMC genomics*, 15(1):409, 2014.

[47] Wei Zhang, Yuanping Du, Zheng Su, Changxi Wang, Xiaojing Zeng, Ruifang Zhang, Xueyu Hong, Chao Nie, Jinghua Wu, Hongzhi Cao, and others. IMonitor: a robust pipeline for TCR and BCR repertoire analysis. *Genetics*, 201(2):459–472, 2015.

[48] Sheng-Jou Hung, Yi-Lin Chen, Chia-Hung Chu, Chuan-Chun Lee, Wan-Li Chen, Ya-Lan Lin, Ming-Ching Lin, Chung-Liang Ho, and Tsunglin Liu. TRIg: a robust alignment pipeline for non-regular T-cell receptor and immunoglobulin sequences. *BMC bioinformatics*, 17(1):433, 2016.

[49] Yaxuan Yu, Rhodri Ceredig, and Cathal Seoighe. LymAnalyzer: a tool for comprehensive analysis of next generation sequencing data of T cell receptors and immunoglobulins. *Nucleic acids research*, 44(4):e31–e31, 2015.

[50] Jacob Glanville, Huang Huang, Allison Nau, Olivia Hatton, Lisa E Wagar, Florian Rubelt, Xuhuai Ji, Arnold Han, Sheri M Krams, Christina Pettus, Nikhil Haas, Cecilia S Lindestam Arlehamn, Alessandro Sette, Scott D Boyd, Thomas J Scriba, Olivia M Martinez, and Mark M Davis. Identifying specificity groups in the T cell receptor repertoire. *Nature*, 547(7661):94–98, 7 2017.

[51] Asaf Madi, Asaf Poran, Eric Shifrut, Shlomit Reich-Zeliger, Erez Greenstein, Irena Zaretsky, Tomer Arnon, Francois Van Laethem, Alfred Singer, Jinghua Lu, Peter D Sun, Irun R Cohen, and Nir Friedman. T cell receptor repertoires of mice and humans are clustered in similarity networks around conserved public CDR3 sequences. *eLife*, 6:e22057, 2017.

[52] Thomas Hoffmann, Antoine Marion, and Iris Antes. DynaDom: structure-based prediction of T cell receptor inter-domain and T cell receptor-peptide-MHC (class I) association angles. *BMC Structural Biology*, 17(1):2, 2018.

[53] Tommi Jaakkola, Mark Diekhans, and David Haussler. A Discriminative Framework for Detecting Remote Protein Homologies. *JOURNAL OF COMPUTATIONAL BIOLOGY*, 7(12):95–114, 2000.

[54] Johannes Söding. Protein homology detection by HMM-HMM comparison. *Bioinformatics*, 21(7):951–960, 2005.

[55] Dan Ofer and Michal Linial. NeuroPID: A predictor for identifying neuropeptide precursors from metazoan proteomes. *Bioinformatics*, 30(7):931–940, 2014.

[56] Robert Rentzsch and Christine A. Orengo. Protein function prediction - the power of multiplicity, 2009.

[57] Annabel E. Todd, Russell L. Marsden, Janet M. Thornton, and Christine A. Orengo. Progress of structural genomics initiatives: An analysis of solved target structures. *Journal of Molecular Biology*, 348(5):1–26, 2005.

[58] Lesley H. Greene, Tony E. Lewis, Sarah Addou, Alison Cuff, Tim Dallman, Mark Dibley, Oliver Redfern, Frances Pearl, Rekha Nambudiry, Adam Reid, Ian Sillitoe, Corin Yeats, Janet M. Thornton, and Christine A. Orengo. The CATH domain structure database: New protocols and classification levels give a more comprehensive resource for exploring evolution. *Nucleic Acids Research*, 35(SUPPL. 1), 2007.

[59] Dong Sheng Cao, Qing Song Xu, and Yi Zeng Liang. Propy: A tool to generate various modes of Chou's PseAAC. *Bioinformatics*, 29(7):960–962, 2013.

[60] C. H.Q. Ding and I. Dubchak. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, 17(4):349–358, 2001.

[61] I Dubchak, I Muchnik, S R Holbrook, and S H Kim. Prediction of protein folding class using global description of amino acid sequence. *Proceedings of the National Academy of Sciences of the United States of America*, 92(19):8700–4, 1995.

[62] Dan Ofer and Michal Linial. ProFET: Feature engineering captures high-level protein functions. *Bioinformatics*, 31(21):3429–3436, 2014.

[63] William R Atchley, Jieping Zhao, Andrew D Fernandes, and Tanja Drüke. Solving the protein sequence metric problem. *Proceedings of the National Academy of Sciences of the United States of America*, 102(18):6395–400, 2005.

[64] Mattia Cinelli, Yuxin Sun, Katharine Best, James M. Heather, Shlomit Reich-Zeliger, Eric Shifrut, Nir Friedman, John Shawe-Taylor, and Benny Chain. Feature selection using a one dimensional naïve Bayes' classifier increases the accuracy of support vector machine classification of CDR3 repertoires. *Bioinformatics*, 33(7):951–955, 2017.

[65] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *Arxiv*, pages 1–12, 2013.

[66] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

[67] Ehsaneddin Asgari and Mohammad R.K. Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PLoS ONE*, 10(11), 2015.

[68] Yoshua Bengio. *Learning Deep Architectures for AI*, volume 2. 2009.

[69] Xiaoyong Pan, Yong-Xian Fan, Junchi Yan, and Hong-Bin Shen. IPMiner: hidden ncRNA-protein interaction sequential pattern mining with stacked autoencoder for accurate computational prediction. *BMC genomics*, 17:582, 2016.

[70] James Lyons, Abdollah Dehzangi, Rhys Heffernan, Alok Sharma, Kuldip Paliwal, Abdul Sattar, Yaoqi Zhou, and Yuedong Yang. Predicting backbone C-alpha angles and dihedrals from protein sequences by stacked sparse auto-encoder deep neural network. *Journal of Computational Chemistry*, 35(28):2040–2046, 2014.

[71] Son P Nguyen, Yi Shang, and Dong Xu. DL-PRO: A novel deep learning method for protein model quality assessment. In *2014 International Joint*

Conference on Neural Networks (IJCNN), volume 2014, pages 2071–2078, 2014.

[72] Tong Liu, Yiheng Wang, Jesse Eickholt, and Zheng Wang. Benchmarking Deep Networks for Predicting Residue-Specific Quality of Individual Protein Models in CASP11. *Scientific reports*, 6(January):19301, 2016.

[73] Jian Wei Liu, Guang Hui Chi, Ze Yu Liu, Liu Yuan, Hai En Li, and Xiong Lin Luo. Predicting protein structural classes with autoencoder neural networks. In *2013 25th Chinese Control and Decision Conference, CCDC 2013*, pages 1894–1899, 2013.

[74] Khurshid Ahmad, Muhammad Waris, and Maqsood Hayat. Prediction of Protein Submitochondrial Locations by Incorporating Dipeptide Composition into Chou's General Pseudo Amino Acid Composition. *The Journal of Membrane Biology*, 249(3):293–304, 6 2016.

[75] T L Bailey. *Bioinformatics: Data, Sequence Analysis and Evolution*, volume I. Springer Science, 2016.

[76] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[77] N Cristianini and J Shawe-Taylor. *An introduction to Support Vector Machines*, volume 47. 2000.

[78] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[79] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[80] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

[81] Sankar K. Pal and Sushmita Mitra. Multilayer Perceptron, Fuzzy Sets, and Classification. *IEEE Transactions on Neural Networks*, 3(5):683–697, 1992.

[82] Djork-ArnÃľ Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *Under review of ICLR2016üijŇ _ŕŘåĞžäžĘELU*, (1997):1–13, 2015.

[83] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *Iclr*, (2013):1–8, 2014.

[84] Soren Kaae Sonderby, Casper Kaae Sonderby, Henrik Nielsen, and Ole Winther. Convolutional LSTM networks for subcellular localization of proteins. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9199:68–80, 2015.

[85] Sepp Hochreiter and Jurgen JÃijrgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1–32, 1997.

[86] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv*, pages 1–9, 2014.

[87] M. Schuster and K. K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[88] P L Bartlett and M Traskin. AdaBoost is consistent. *Journal of Machine Learning Research*, 8:2347–2368, 2007.

[89] Robert E. Schapire. Explaining adaboost. In *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 37–52. 2013.

[90] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *KDD*, pages 1–10, 2016.

[91] L Breiman, J H Friedman, R A Olshen, and C J Stone. *Classification and Regression Trees*, volume 19. 1984.

[92] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. *NIPS*, pages 512–518, 1999.

[93] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[94] R R Development Core Team. *R: A Language and Environment for Statistical Computing*, volume 1. 2011.

[95] Python Software Foundation. Python Language Reference, version 3.6, 2013.

[96] Steven Bird and Edward Loper. NLTK: The Natural Language Toolkit. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 1–4, 2004.

[97] P J Cock, T Antao, J T Chang, B A Chapman, C J Cox, A Dalke, I Friedberg, T Hamelryck, F Kauff, B Wilczynski, and M J de Hoon. Biopython, 2009.

[98] Rob Knight, Gavin Huttley, Daniel McDonald, Micah Hamady, and Antonio Gonzalez. Scikit-Bio: Python package for Bioinformatics, 2017.

[99] J. D. Hunter. Matplotlib: A 2D graphic environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[100] Michael Waskom ; Olga Botvinnik ; Paul Hobson ; John B Cole ; Yaroslav Halchenko ; Stephan Hoyer ; Alistair Miles ; Tom Augspurger ; Tal Yarkoni ; Tobias Megies ; Luis Pedro Coelho ; Daniel Wehner ; cynddl ; Erik Ziegler ; diego0020 ; Yury V. Zaytsev ; T and Michael Waskom ; Olga Botvinnik ; Paul Hobson ; John B. Cole ; Yaroslav Halchenko ; Stephan Hoyer ; Alistair Miles ; Tom Augspurger ; Tal Yarkoni ; Tobias

Megies ; Luis Pedro Coelho ; Daniel Wehner ; cynddl ; Erik Ziegler ; diego0020 ; Yury V. Zaytsev ; T. seaborn: v0.5.0. *zenodo*, 2014.

[101] Fabian Pedregosa, GaÃńl Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and ÃĽdouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2012.

[102] Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognition Letters*, 80:150–156, 2016.

[103] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, pages 1–13, 2014.

[104] Alexander Kraskov, Harald Stoegbauer, Ralph G Andrzejak, and Peter Grassberger. Hierarchical clustering using mutual information. *Europhysics Letters*, 70(2):4, 2003.

[105] Brian C. Ross, A Kraskov, H Stögbauer, P Grassberger, I Grosse, P Bernaola-Galván, P Carpena, R Román-Roldán, J Oliver, L Kozachenko, and NN Leonenko. Mutual Information between Discrete and Continuous Data Sets. *PLoS ONE*, 9(2):e87357, 2014.

[106] Ladislav Rampasek and Anna Goldenberg. TensorFlow: Biology's Gateway to Deep Learning? *Cell Systems*, 2(1):12–14, 2016.

[107] FranÃğois Chollet. Keras: Deep Learning library for Theano and TensorFlow. *GitHub Repository*, pages 1–21, 2015.

[108] FranÃğois Chollet. Building Autoencoders in Keras, 2016.

[109] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[110] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[111] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Arxiv*, pages 1–11, 2015.

[112] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:249–256, 2010.

[113] Long Ma, Liwen Yang, Bin Shi, Xiaoyan He, Aihua Peng, Yuehong Li, Teng Zhang, Suhong Sun, Rui Ma, and Xinsheng Yao. Analyzing the CDR3 Repertoire with respect to TCRâĂŤBeta Chain V-D-J and V-J Rearrangements in Peripheral T Cells using HTS. *Scientific Reports*, 6(July):29544, 2016.

[114] Florian Krieger, Andreas Möglich, and Thomas Kiefhaber. Effect of proline and glycine residues on dynamics and barriers of loop formation in polypeptide chains. *Journal of the American Chemical Society*, 127(10):3346–3352, 2005.

[115] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning. *Elements*, 1:337–387, 2009.

[116] Yuxin Sun, Katharine Best, Mattia Cinelli, James M. Heather, Shlomit Reich-Zeliger, Eric Shifrut, Nir Friedman, John Shawe-Taylor, and

Benny Chain. Specificity, privacy, and degeneracy in the CD4 T cell receptor repertoire following immunization. *Frontiers in Immunology*, 8(APR), 2017.

[117] Carlos Domingo and Osamu Watanabe. MadaBoost: A Modification of AdaBoost. *Conference on Computational Learning Theory (COLT)*, pages 180–189, 2000.

[118] Xin-Qiu Yao, Huaiqiu Zhu, and Zhen-Su She. A dynamic Bayesian network approach to protein secondary structure prediction. *BMC bioinformatics*, 9:49, 2008.