5-8-2020

Machine Learning and Deep Learning to Predict Cross-immunoreactivity of Viral Epitopes

Zahra Tayebi

MACHINE LEARNING AND DEEP LEARNING TO PREDICT CROSS-IMMUNOREACTIVITY OF VIRAL EPITOPES

by

ZAHRA TAYEBI

Under the Direction of Pavel Skums, PhD

ABSTRACT

Due to the poor understanding of features defining cross-immunoreactivity among heterogeneous epitopes, vaccine development against the hepatitis C virus (HCV) is trapped. The development of vaccines against HCV and human immunodeficiency virus, which are highly heterogeneous viruses (HIV) is significantly vulnerable due to variant-specific neutralizing immune responses. The novel vaccine strategies are based on some assumptions such as immunological specificity which is strongly linked to the epitope primary structure, by increasing genetic difference between epitopes cross-immunoreactivity (CR) will decline [1]. In this study first, we checked the hamming distance and statistic evaluation associating HVR1 sequence and CR based on the sequence of the immunogen and antigen pairs then generated five different machine learning models. Also, we implemented deep learning models like Convolutional Neural Network to predict CR. As a result we could provide 90% accuracy by using the CNN model.

INDEX WORDS: HCV, Cross-immunoreactivity, Hamming distance, Machine learning, Deep learning, CNN

MACHINE LEARNING AND DEEP LEARNING TO PREDICT CROSS-IMMUNOREACTIVITY OF VIRAL EPITOPES

by

ZAHRA TAYEBI

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2020

MACHINE LEARNING AND DEEP LEARNING TO PREDICT CROSS-IMMUNOREACTIVITY OF VIRAL EPITOPES

by

ZAHRA TAYEBI

DEDICATION

This thesis is dedicated to my husband Vahid and my parents whose unyielding love, support, and encouragement have excited my soul and inspired me to peruse and complete this research.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Doctor Pavel Skums, who guided me throughout this study. I would also like to thank my friends and family who supported me and offered deep insight into the study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Hepatitis C virus (HCV) belongs to the Flaviviridae family [2] and is a single-stranded RNA virus. 3.0% of the world's population is infected by HCV, and it is a significant cause of liver disease in the world [3]. HCV infection growths to chronicity in 70%–85% of infected adults [4]. Per year, it is estimated that 476,000 deaths are because of hepatitis C. As current anti-viral therapy is not much effective for patients [5], there is a big need for HCV vaccine. Genetically, HCV is very heterogeneous and is classified into six genotypes and numerous sub-genotypes [6]. To control infectious diseases, vaccines are the most efficacious. But the problem is that the development of vaccines against highly heterogeneous viruses like HCV and human immunodeficiency virus (HIV) is significantly vulnerable by variant-specific neutralizing immune responses. A major obstacle for formulating broadly protective vaccines is that these viruses have an unlimited capacity to quickly mutate and escape from immune neutralization [7][8]. It is estimated that 130 million and 33 million individuals are infected in the world with HCV and HIV, respectively [3][8]. The number of viral variants circulating in the world is immense because each infected host has a large variety of viral variants. Developing vaccines against this big range of viral variants is challenging. We used hamming distance which is the number of positions at which corresponding symbols are different to see the relationship between hamming distance and CR. Modeling supervised machine learning models like SVM, Naive Bayes, and Logistic Regression are another way to check the accuracy of CR. We also implemented Decision tree model which is implemented by another study to compare our results. Although, we got better performance by using machine learning models, but the main benefit of deep learning techniques is automatic features extraction and regulation of performance in a continuous fashion. So, those has been increasingly used to solve many challenging problems in biology in the past few years.

2 RELATED WORKS

Classical vaccine development approaches are still producing broadly protective vaccines against HCV and HIV [7][9]. Recently, to cope with viral antigenic diversity, novel vaccine strategies developed. These strategies focus on using epitopes with restricted heterogeneity [10], the center of tree variants, or phylogenetic ancestors [11], generating a concoction of heterogeneous epitopes [12][13] or mimotopes [14][15], or predicting consensus sequences. But regard to properties of highly heterogeneous epitopes, these strategies are based on some assumptions such as immunological specificity is strongly linked to the epitope primary structure, by increasing genetic difference between epitopes cross-immunoreactivity (CR) will decline, and that by diversifying evolution resulting from an ‘‘arms race’’, the viral sequence space is shaped [1]. However, the conditional relevance of these assumptions has not been methodically validated.

A quantitative analysis of the HVR1 CR is modeled, which used synthetic peptides and mouse immunization, in conjunction with a network analysis of the HVR1, sequence space showed significant immunological and structural HVR1 convergence. The findings suggest tractability of the HVR1 immunological specificity and offer a novel framework for HCV vaccine development, which is applicable to other heterogeneous viruses [16].

There are several studies which have considered machine learning in classification problems. Based on the type of problem. In our case, based on our data and labels, we need a supervised learning model to do classification and prediction. We focused mainly on five types of classification technics, (i) Support Vector Machine (SVM), (ii) Naive Bayes (NB), (iii) Logistic Regression(LR), (iv) K Nearest Neighbors, and (v) Decision Tree(DT).

Campo DS et al.[16] found a significant functional and structural convergence that shows in the occupied sequence space the same HVR1 properties evolved frequently and independently.

They are tractable and highly amenable to predictive modeling because of their regular occurrence of the limited number of immunological specificities. So, they generated a decision tree associating HVR1 sequence and CR, which, based on the sequence of the immunogen and antigen pairs, can predict CR [16]. They used 10-fold cross-validation and got an average testing accuracy of 92.5%. But they have done oversampling, and they did not mention anything. Also, different manual settings are applied in their decision tree model. They have not explained their model clearly and just reported their results.

3 MATERIALS AND METHODS

3.1 Dataset

We are using serum specimens from mice immunized with 103 synthetic peptides representing different HVR1 variants (referred to as immunogens) to model HVR1 CR. We will use a set of 261 HVR1 peptides as antigens in an enzyme immunoassay. We used the HVR1 variants, which all randomly selected from major branches of a phylogenetic tree (from GenBank). We would have 26,883 unique immunogen antigen pairs in total for training.

Also, we have a matrix of ones (CR happened) and zeros (CR not happened) as our labels, which means we will have a binary classification problem. The number of columns of this matrix is equal to the number of antibodies sequences, and the number of rows is equal to the number of antigens sequences.

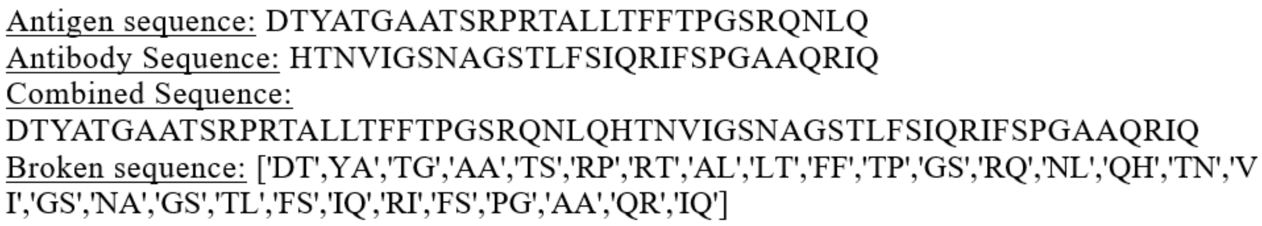
3.2 Implementation

Prediction of cross-reactivity using a decision tree has done before[16]. We tried five different machine learning algorithms, SVM, KNN, LR, NB, and, DT. In addition, we used the advantages of CNN to predict CR. We used CNN to conduct feature extraction and dimensionality compression and classification.

3.2.1 Preprocessing

To implement our models, we need to do preprocessing as a first step. In this stage, we did some preprocessing techniques to prepare our data as an input of machine learning and deep learning models. There is no confident evidence that can show the structure of protein sequences when CR happened, and it was a challenging part of the preprocessing part. So, we assumed some structures and will progress the preprocessing part based on those assumptions. An excellent first step when working with sequences is to split it into words or in terms of sequences, breaking those to the smaller lengths.

We have tried different preprocessing technics to get the best performance. First, we considered a pair of the 29-length sequences from antibodies and antigens datasets and concatenated them, and then we broke those final 58 length sequences to length 2 subsequences. In total, we have 29 subsequences for each concatenated sequence, Figure1.



Antigen sequence: DTYATGAATSRPRTALLTFFTPGSRQNLQ
Antibody Sequence: HTNVIGSNAGSTLFSIQRIFSPGAAQRIQ
Combined Sequence:
DTYATGAATSRPRTALLTFFTPGSRQNLQHTNVIGSNAGSTLFSIQRIFSPGAAQRIQ
Broken sequence: ['DT',YA','TG','AA','TS','RP','RT','AL','LT','FF','TP','GS','RQ','NL','QH','TN','VI','GS','NA','GS','TL','FS','IQ','RI','FS','PG','AA','QR','IQ']

Figure 1:A sample of Preprocessing

Each subsequence is getting a string of zeros and ones which is based on a dictionary of premutation of pair characters.

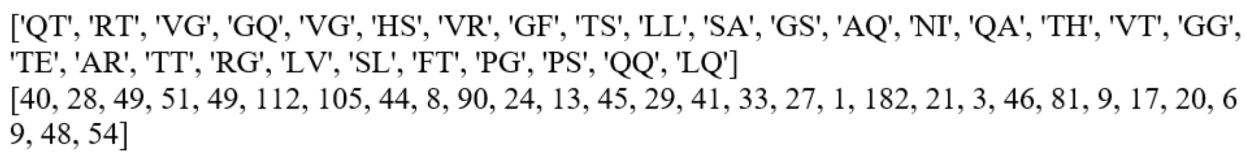
In the second approach, instead of placing each pair of antigen and antibody back to back, we placed one sequence below another and then broke those final 58 length sequences to length 2 subsequences. Then we used the dictionary to assign zeros and ones strings to them.

In the third approach, after getting 29 subsequences in the second stage of preprocessing, the whole dataset is split into two parts, 75 percent of data for training, and 25 percent for testing. Then we used Keras as high-level APIs of neural networks. It supports multiple backend neural network computation engines and is written in Python. Ease of learning, ease of model building, and supporting multiple backend engines are the reasons which we used Keras, and also, it is user-friendly. We used Keras with TensorFlow backend, which is its primary backend to do its low-level operations.

As we need to know if there is any meaning in how the characters are placed side to side, word embedding is very handy for us. We know that word embedding deals with dense vectors so, we should vectorize sequences into a list of integers. We used Keras word embedding method, which is much more accurate than previous methods such as bag of words. Keras word embedding is the library function that allows the use of the embedding for the word semantically and syntactically, and we need here the semantic word embedding.

So, we need to tokenize sequences. By using Keras Tokenizer, the numbers do not represent counts but rather correspond to the word values from the dictionary word index. Each value of the dictionary that encodes the total dataset with the keys in the dictionary maps to one of each integer in our vectorized sequences. Those would be our vocabulary terms. In other words, “fit_on_texts” updates internal vocabulary based on a list of sequences. Here the vocabulary index is creating based on word frequency. So, if we give it something like, "it is rainy." It will create a dictionary, word_index["it"] = 1; word_index["is"] = 2, etc. which is a word index dictionary. So, every subsequence gets a unique integer value. Index 0 is reserved for padding. Lower integer means more frequent subsequence. “texts_to_sequences” of Tokenizer transforms each sequence in sequences to a sequence of integers. So, it takes each subsequence in the sequences and replaces it with its corresponding integer value from the word_index dictionary.

By using Tokenizer, the length of the resulting vectors is equal to the length of each sequence, Figure2.

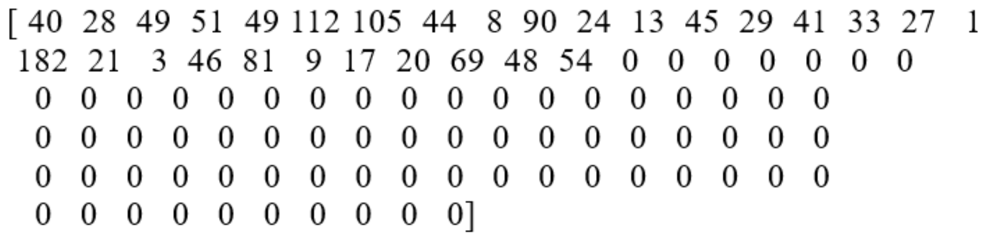


```
['QT', 'RT', 'VG', 'GQ', 'VG', 'HS', 'VR', 'GF', 'TS', 'LL', 'SA', 'GS', 'AQ', 'NI', 'QA', 'TH', 'VT', 'GG',
'TE', 'AR', 'TT', 'RG', 'LV', 'SL', 'FT', 'PG', 'PS', 'QQ', 'LQ']
[40, 28, 49, 51, 49, 112, 105, 44, 8, 90, 24, 13, 45, 29, 41, 33, 27, 1, 182, 21, 3, 46, 81, 9, 17, 20, 6
9, 48, 54]
```

Figure 2:Example of Tokenization result

Keras is doing these two parts separately because we almost always fit once and convert to sequences many times. To convert actual text into sequences and feed them to the network, we will fit on our training corpus one time and will use that same word_index dictionary at training, evaluation, testing, or prediction time.

One problem that we have is that in each sequence, we will have multiple repetitions of subsequences, and we would have different lengths for our integer vectors. To solve this problem, we have used sequence padding, which simply pads the sequence of integers with zeros. We may prepend zeros or append them. As it does not matter whether we prepend or append zeros, we preferred to prepend zeros. We specified the fixed length of our sequences by adding a maximum length parameter. This parameter cuts sequences that surpass that number. So, our data is now ready as our model input, Figure3.

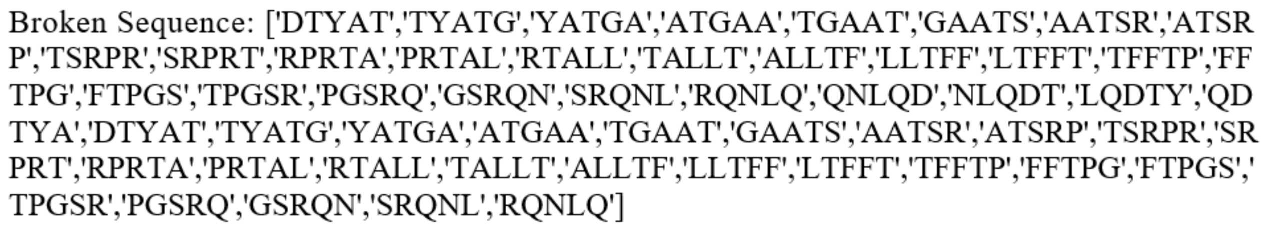


```
[ 40  28  49  51  49 112 105  44   8  90  24  13  45  29  41  33  27   1
 182  21   3  46  81   9  17  20  69  48  54   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

Figure 3:Sequence padding

In the fourth approach, k-mers is implemented to get different subsequences of each concatenated sequence which here is a different combination of k-length subsequences.

Choosing k was another challenging part. We decided to consider k=5, which means different 5-length combinations of each sequence. In total, there are 54 subsequences, Figur4. Then we used Keras Tokenizer to tokenize sequences.



```
Broken Sequence: ['DTYAT','TYATG','YATGA','ATGAA','TGAAT','GAATS','AATSR','ATSR
P','TSRPR','SRPRT','RPRTA','PRTAL','RTALL','TALLT','ALLTF','LLTFF','LTFFT','TFFTP','FF
TPG','FTPGS','TPGSR','PGSRQ','GSRQN','SRQNL','RQNLQ','QNLQD','NLQDT','LQDTY','QD
TYA','DTYAT','TYATG','YATGA','ATGAA','TGAAT','GAATS','AATSR','ATSRP','TSRPR','SR
PRT','RPRTA','PRTAL','RTALL','TALLT','ALLTF','LLTFF','LTFFT','TFFTP','FFTPG','FTPGS','
TPGSR','PGSRQ','GSRQN','SRQNL','RQNLQ']
```

Figure 4:K-mers, Five-mers

In the next step of preprocessing, oversampling is implemented, which is due to imbalanced data. It will duplicate examples from the minority class. In our dataset, we have class one (cross-immunoreactivity) and zero (not cross-immunoreactivity), and we had severe skew in the class distribution. So, we needed to handle this situation to prevent bias in the training dataset. Even SVM and some of other machine learning algorithms can handle unbalanced data but we wanted to find the hiest accuracy. Therfore we decided to use the SMOTE (Synthetic Minority Over-sampling Technique) oversampling method among different methods of oversampling. This method generates synthetic data based on the feature space similarities between existing minority instances. In order to create a synthetic instance, it finds the K-nearest neighbors of each minority instance, randomly selects one of them, and then calculate linear interpolations to produce a new minority instance in the neighborhood [17]. We decided to check how the accuracy metric will change.

3.2.2 Evaluation based on hamming distance

Hamming distance is the measure of dissimilarity between two sequences, i.e., the no. of positions at which the corresponding character is different. So, for each pair of antigen and antibody hamming distance is calculated, and then we checked if the hamming distance is between ten to twenty-four and made a matrix with a size equal to the label matrix. If the value of both were the same in the same positions, it means hamming distance has an important role in CR. However, this method is not applicable and trustable for too much data. So, other methods are implemented.

3.2.3 Classification models

Most of the classifiers, such as decision trees and neural networks, can only take input data as a vector of features. However, there are no explicit features in sequence data. So, it is challenging to find features. First, we built our machine learning models and made a comparison. Then we implemented the deep learning model and compared our results.

3.2.3.1 Support Vector Machine (SVM)

A practical method for sequence classification is SVM [18]. This is because to separate two classes, SVMs are designed to maximize the margin. It means the trained model generalizes well on unseen data. Most other computer programs implement a classifier through the minimization of error occurred in training, which leads to poorer generalization. Because of this, SVMs have been widely applied to many areas of bioinformatics, including protein function prediction, protease functional site recognition, transcription initiation site prediction, and gene expression data classification[19]. In this study, SVM is implemented for all four previous preprocessing methods. The SVM model, which we used, is from sklearn library of python (svm.SVC()), which has different kernels and regularization parameters, e.g., parameter C, which is for minimizing the norm of the weights and controls the trade-off between achieving a low error on the training data. We have tried different kernels and parameter C values by a grid search to find what is the best for each preprocessing method and then the training data is fitted to the SVM model, and for 10-fold we got different accuracies, and the best result belongs to the C = 6.0 and kernel: 'rbf' which was 86.25%. Table 1.

3.2.3.2 Naïve Bayes

Usually, a model is defined based on some assumptions, and the probability distributions are described by a set of parameters. In the training step, the parameters of M are learned. In the classification step, a new sequence is assigned to the class with the highest likelihood. The simplest generative model is the Naive Bayes sequence classifier [20]. It makes the assumption that, given a class, the features in the sequences are independent of each other. The conditional probabilities of the features in a class are learned in the training step. Due to its simplicity, Naïve Bayes has been widely used from text classification [21] and genomic sequences classification [22]. Here we used GaussianNB from sklearn which the likelihood of the features is assumed to be Gaussian. For first and second preprocessing methods, we got 80.9% accuracy, and for third and fourth methods, we got 78.5% and 56.04% accuracy, respectively, Table 1.

3.2.3.3 K Nearest Neighbors (KNN)

In classifying sequences, the major problem is that they are a mixture of characters and words. We need numerical representation of those words to feed them into our K-NN algorithm to compute distances and make predictions. That is the reason when we used third and fourth preprocessing methods, which are tokenizing methods, we got more accuracy. Here we tried a different number for K by grid search to find the best K for each preprocessing model and then applied KNN. The KNN model is from skit-learn (kNeighborsClassifier()), and the best result belongs to the third preprocessing method and K=15, which was 85.21% after 10-folds. Table 1.

3.2.3.4 Logistic Regression

One of the most simple and commonly used Machine Learning algorithms for two-class classification is Logistic Regression. It is easy to implement and has different parameters to tune C=c, solver=Solver, penalty=Penalty, max_iter=Max_iter). We tried different parameters by grid search and found the best parameters for each preprocessing method and then applied the LogisticRegression(), which is from skit-learn library of python. The best accuracy belongs to the third preprocessing method and 'C': 12, 'max_iter': 10, 'penalty': 'l1', 'solver': 'saga', which gave us 81.29% accuracy. Table 1.

3.2.3.5 Decision Tree

Decision tree is one of the classification techniques that uses branches method to illustrate each feasible result of a decision making in each possible outcome. Decision trees are great for their simplicity and interpretation. In this study, we used DecisionTreeClassifier() from skit-learn library of python and tried different max_depth parameters for our different preprocessing methods and find max_depth=12 and the SVM for the third preprocessing method as the best model which provided accuracy equal to 85.22%. Table 1.

Table 1: Accuracy comparison, using different models and preprocessing methods

		Preprocessing		
Models	First method	Second method	Third method	Fourth method
SVM	81.25%	81.24%	86.84%	84.96%
Naïve Bayes	80.97%	80.98%	78.55%	56.04%
K Nearest Neighbor	81.12%	81.12%	85.21%	84.74%
Logistic Regression	81.10%	81.10%	81.29%	81.18%
Decision Tree	81.23%	81.20%	85.22%	84.69%

Based on our results from different machine learning models, the SVM classifier had the best accuracy in CR prediction.

3.2.3.6 CNN

In this section, we will build our model, which is CNN in the first step. CNN can extract features from images and using them in neural networks. So, it has developed image classification and computer vision. The properties that made CNN useful in image processing makes them also useful for sequence processing. We can imagine CNN as a specialized neural network that can detect specific patterns.

Predicting CR is a binary classification problem. We decided to use deep learning models to solve this problem. The protein sequences contain much valuable information. They also contain too much noise. In order to get a more accurate representation, we use the CNN algorithm to extract protein sequences features. In a layer-by-layer manner, CNN can effectively avoid the interference of human factors and extract the advanced features of protein sequences information automatically and objectively. The following, we used Keras to build our deep learning model. This stage was another challenging part of this study for us because we had to understand each part of Keras carefully.

One of the main models in Keras is the Sequential model, which we used here. A linear stack of layers built the Sequential model, and each layer has its own size that is customizable and its activation function. In each step, we can add a layer to our model by using model.add(). The first layer is the embedding layer which has three arguments:

i. ‘input_dim’, which is the size of the vocabulary in the text data. Our data is integer encoded to values between 0-400 so, the size of our vocabulary is 401 words.

ii. 'output_dim' that is the size of the vector space that words are embedded. This layer defines for each word the size of the output vectors, and for our model, it is 29.

iii. 'input_length' that we defined the input layer of our model which its length is equal to the length of our input sequences.

The computations of the Embedding layer is multiplying a one-hot vector with an embedding matrix in constant time. So, one 2D vector with one embedding for each subsequence in the input sequence of subsequences would be the output of the Embedding layer.

The next layer is Conv1D, which is a one-dimensional convolution layer. We work with one-dimensional convolution due to our sequential data. Patterns in the sequence become more sophisticated by adding each convolutional layer, and we want to pick on those patterns and detect specific features. One dimensional convolution considers the size of the filter kernel and starts to take a patch of input features. Then it computes the dot product of the input and weights of the filter, Figure 5. One dimensional ConvNet is helpful for specific patterns in sequences; certain sequences can be recognized at a different position because it is invariant to translations.

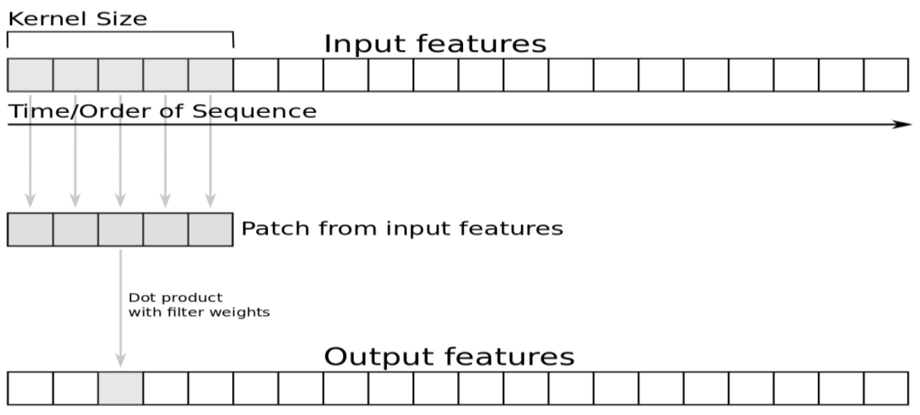


Figure 5:1 Dimensional Convolution

The Conv1D layer has three arguments, the number of filters, the kernel size, and the activation function. In our model architecture, Figure 6, we added three Conv1D layers to our model, and for each of them using 128 filters, kernel size of 3 and ‘ReLU’ activation function.

In the next step, due to our unbalanced dataset, we added a batch normalization layer in which the activations of the previous layer at each batch will become normalized by that, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. The benefits of that are faster network training, which gives us much more quickly convergence, using higher learning rates, easier initialization of weights, having viable activation functions, and as a result, will give better results. We used batch normalization layers after the convolution layer and before max-pooling layers.

After the batch normalization layer, we reduced the dimensionality from 3D to 1D by using the GlobalMaxPooling1D layer, and its output is one response for every feature map. To get a shape that works with dense layers it is often used at the end of the backend of a convolutional neural network.

In the end, we have 2 Dense layers, which the first one had a ReLU activation function and reduce the dimension from 128 to 10. The second fully connected layer reduced the ten dimensions to 1 dimension, which means class 1 or 0 for our classification problem.

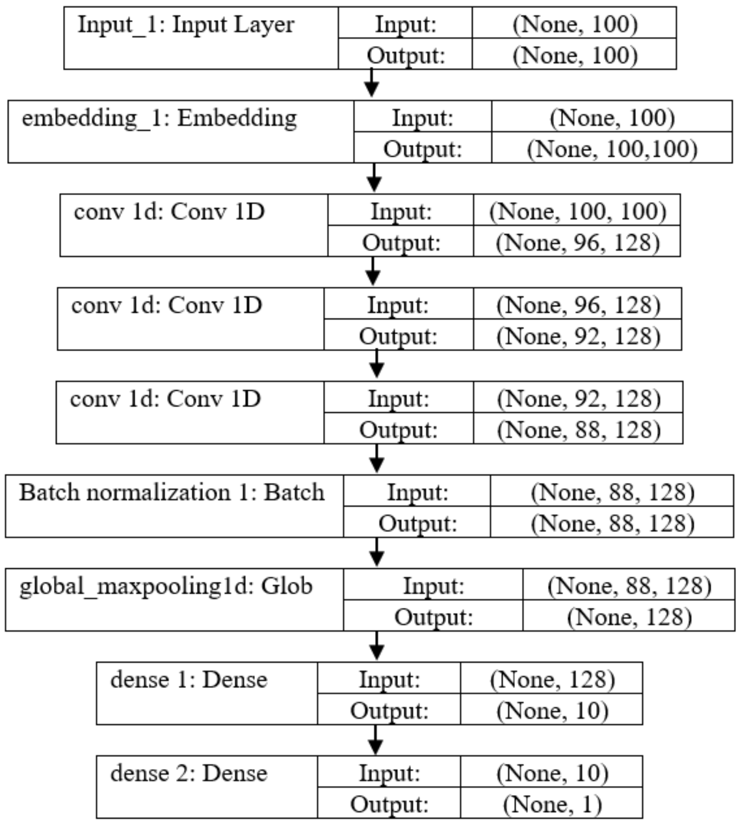


Figure 6:Model Architecture

4 RESULTS

After running the model for 20 epochs and considering the batch sizes equal to 20, the accuracy and loss are two parameters that we considered and evaluated our model. The accuracy of the training set was 97%, and for the testing set was 90%, Figure 7. The loss for the training set was 0.1, and for test set was 0.35, Figure 7.

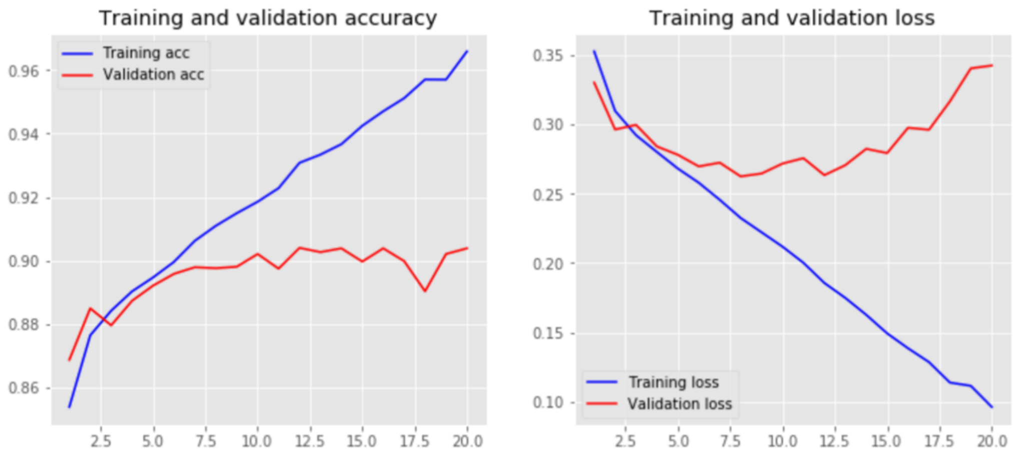


Figure 7: CNN Training and validation accuracy and loss, epochs:20, batch size:20

In this study, the same model CNN is implemented with different numbers of epochs and batch sizes. We have tried epochs = 100 and batch size = 32 and got 90.17% accuracy for the testing set, Figure 8.

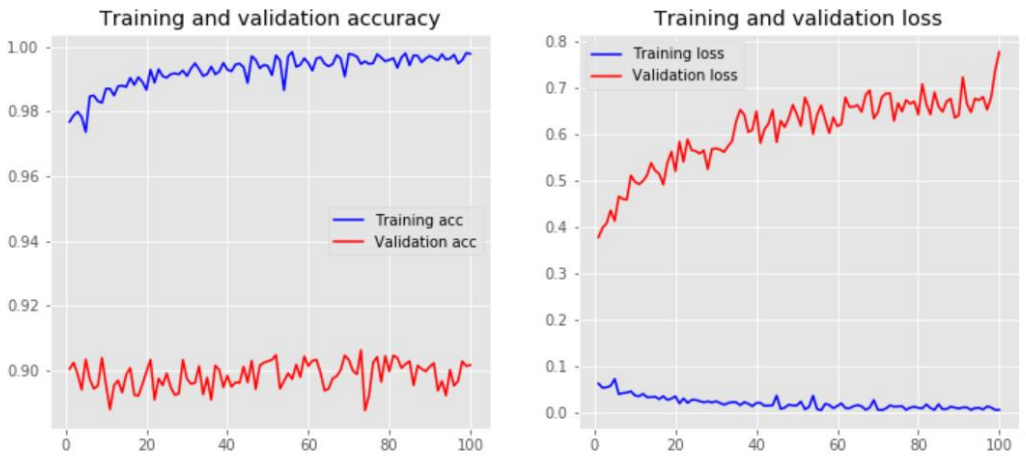


Figure 8: CNN Training and validation accuracy and loss, epochs:100, batch size:32

Another was epochs = 10 and size = 5, and the test accuracy was 89.53%, Figure 9.

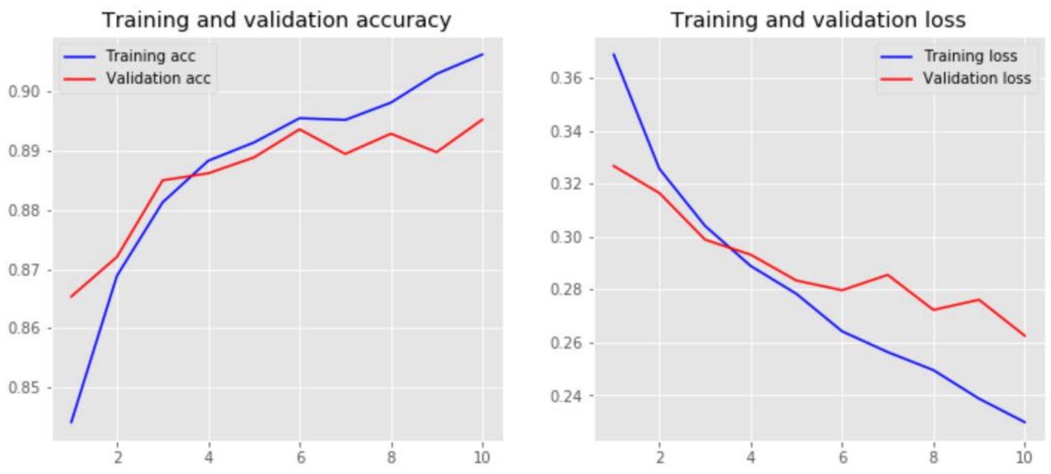


Figure 9: CNN Training and validation accuracy and loss, epochs:10, batch size:5

For epochs = 50 and size = 10 test accuracy was 89.69%, Figure 10.

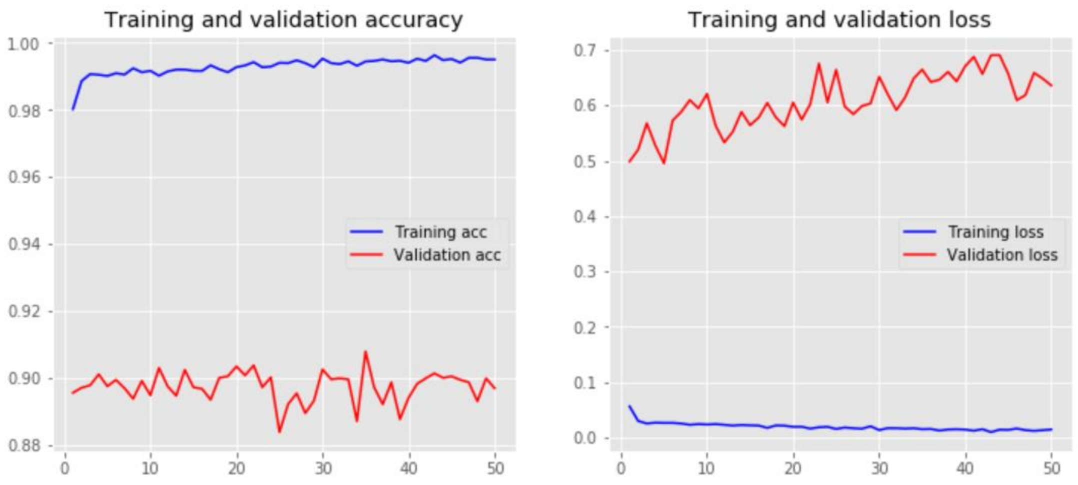


Figure 10: CNN Training and validation accuracy and loss, epochs:50, batch size:10

In the future, we have the plan to implement the LSTM model with attention, tuning hyperparameters more carefully, and Use the parameter grid, which we believe will make this prediction more accurate. Also, we want to extract sequence information by Position Specific Scoring Matrix to see is there any sign to show us how we should bind the sequences in the preprocessing part to get better results.

5 CONCLUSIONS

Elucidation of CR among variable antigenic epitopes is crucial for vaccine development. Most of the previous studies are based on biology assumptions, and they didn’t consider how antigen and antibody sequences bind. Just One study considered the combination of these sequences, and they used a decision tree to model this problem. But they have used manual settings to predict CR, e.g., cut-off = 0.85. In this study, we modeled the HVR1 CR in sequence space by different machine learning algorithms and tuned hyperparameters for each model and preprocessing method. We found SVM and our third preprocessing method as the best, which provides 86.25% accuracy. Then we modeled the HVR1 CR in sequence space by CNN for the first time and got 90.39% accuracy, which is significant.

REFERENCES

[1] I. Sheridan, O. G. Pybus, E. C. Holmes, and P. Klenerman, “High-resolution phylogenetic analysis of hepatitis C virus adaptation and its relationship to disease progression.,” *J. Virol.*, vol. 78, no. 7, pp. 3447–54, Apr. 2004.

[2] G. Kuo *et al.*, “An assay for circulating antibodies to a major etiologic virus of human non-A, non-B hepatitis.,” *Science*, vol. 244, no. 4902, pp. 362–4, Apr. 1989.

[3] J. Torresi, D. Johnson, and H. Wedemeyer, “Progress in the development of preventive and therapeutic vaccines for hepatitis C virus,” *J. Hepatol.*, vol. 54, no. 6, pp. 1273–1285, Jun. 2011.

[4] A. Alberti, L. Chemello, and L. Benvegnù, “Natural history of hepatitis C,” *J. Hepatol.*, vol. 31, pp. 17–24, Jan. 1999.

[5] D. G. Bowen and C. M. Walker, “Adaptive immune responses in acute and chronic hepatitis C virus infection,” *Nat. 2005 4367053*, vol. 436, no. 7053, pp. 946–952, Aug. 2005.

[6] P. Simmonds, “Genetic diversity and evolution of hepatitis C virus - 15 years on,” *J. Gen. Virol.*, vol. 85, no. 11, pp. 3173–3188, Nov. 2004.

[7] M. Houghton, “Prospects for prophylactic and therapeutic vaccines against the hepatitis C viruses,” *Immunol. Rev.*, vol. 239, no. 1, pp. 99–108, Jan. 2011.

[8] S. P. McBurney and T. M. Ross, “Viral sequence diversity: challenges for AIDS vaccine designs,” *Expert Rev. Vaccines*, vol. 7, no. 9, pp. 1405–1417, Nov. 2008.

[9] S. Rerks-Ngarm *et al.*, “Vaccination with ALVAC and AIDSVAX to Prevent HIV-1 Infection in Thailand,” *N. Engl. J. Med.*, vol. 361, no. 23, pp. 2209–2220, Dec. 2009.

[10] O. O. Yang, “Candidate Vaccine Sequences to Represent Intra- and Inter-Clade HIV-1 Variation,” *PLoS One*, vol. 4, no. 10, p. e7388, Oct. 2009.

[11] T. Hamano *et al.*, “Determination of HIV Type 1 CRF01_AE gag p17 and env-V3 Consensus Sequences for HIV/AIDS Vaccine Design,” *AIDS Res. Hum. Retroviruses*, vol. 20, no. 3, pp. 337–340, Mar. 2004.

[12] K. H. Kang *et al.*, “Synthetic Antigens Representing the Antigenic Variation of Human Hepatitis C Virus,” *Viral Immunol.*, vol. 23, no. 5, pp. 497–508, Oct. 2010.

[13] K. Yusim *et al.*, “Genotype 1 and global hepatitis C T-cell vaccines designed to optimize coverage of genetic diversity,” *J. Gen. Virol.*, vol. 91, no. 5, pp. 1194–1206, May 2010.

[14] L. M. R. El-Attar, C. D. Partidos, and C. R. Howard, “A peptide mimotope of hepatitis C virus E2 protein is immunogenic in mice and block human anti-HCV sera,” *J. Med. Virol.*, vol. 82, no. 10, pp. 1655–1665, Sep. 2010.

[15] R. Roccasecca *et al.*, “Mimotopes of the hyper variable region 1 of the hepatitis C virus induce cross-reactive antibodies directed against discontinuous epitopes,” *Mol. Immunol.*, vol. 38, no. 6, pp. 485–492, Dec. 2001.

[16] D. S. Campo *et al.*, “Hepatitis C Virus Antigenic Convergence,” *Sci. Rep.*, vol. 2, no. 1, p. 267, Dec. 2012.

[17] “Fighting Imbalanced Data Set with code Examples - Towards Data Science.” [Online]. Available: https://towardsdatascience.com/fighting-imbalance-data-set-with-code-examples-f2a3880700a6. [Accessed: 09-Mar-2020].

[18] Z. Xing, J. Pei, and E. Keogh, “A brief survey on sequence classification,” *ACM SIGKDD Explor. Newsl.*, vol. 12, no. 1, p. 40, 2010.

[19] Z. R. Yang, “Biological applications of support vector machines.,” *Brief. Bioinform.*, vol. 5, no. 4, pp. 328–338, 2004.

[20] D. D. Lewis, “Naive(Bayes)at forty: The independence assumption in information retrieval,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1998, vol. 1398, pp. 4–15.

[21] S. H. Myaeng, K. S. Han, and H. C. Rim, “Some effective techniques for naive bayes text classification,” *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 11, pp. 1457–1466, 2006.

[22] B. Y. M. Cheng, J. G. Carbonell, and J. Klein-Seetharaman, “Protein classification based on text document classification techniques,” *Proteins Struct. Funct. Genet.*, vol. 58, no. 4, pp. 955–970, Mar. 2005.