Thesis Dissertation

CONTINUOUS INPUT VECTOR REPRESENTATION THROUGH EMBEDDINGS FROM LANGUAGE MODELS FOR PROTEIN STRUCTURE PREDICTION USING CONVOLUTIONAL NEURAL NETWORKS WITH HESSIAN FREE OPTIMISATION

Sotiris Chatzimiltis

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2021

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

Continuous input vector representation through Embeddings from Language Models for Protein Structure Prediction using Convolutional Neural Networks with Hessian Free Optimisation

Sotiris Chatzimiltis

Supervisor
Dr. Chris Christodoulou

Thesis submitted in partial fulfilment of the requirements for the award of bachelor's degree in Computer Science at University of Cyprus

May 2021

Acknowledgements

Firstly I would like to express my opinion on academic research. Academic research provides the opportunity to students to explore a whole new world and develop skills that could be very valuable in the future.

At this point I would like to thank my advisor Dr. Chris Christodoulou for the continuous support on my related research, for his patience, motivation and knowledge of the subject. His guidance played a major role in the completion of this thesis. Moreover I would like to pay my special regards to Dr. Vasilis Promponas for his initial idea about this dissertation and his knowledge in biology that helped me understand crucial parts of this project.

Special thanks to Dr. Michalis Agathokleous for his constant support and alumnus Panayiotis Leontiou that provided to me necessary programs and scripts to implement this project.

Finally I would like to thank my family for all the mental and emotional support throughout my studies.

Abstract

This dissertation is attempting to solve the protein secondary structure prediction problem, a problem that concerns the field of Computer Science and Biology for decades.

Proteins are highly complex substances that are present in all living organisms. Proteins are of great nutritional value and they directly contribute in the chemical processes essential for life. The study proteins (structure and function), can help to improve food supplements, drugs and antibiotics. Moreover, the study of existing proteins could possibly help treat diseases and solve numerous biological problems, such as the covid-19.

There are millions of proteins with known primary structure but only for a small fraction of those we know the secondary and tertiary structure. The reason is that current state-of-the art methods and instruments for protein structure determination are extremely expensive in terms of both money and time. This is incredibly serious, since the primary structure on its own, tells nothing about the actual function of the protein. This emerged the need of a number of computational algorithms and techniques that attempt to predict the secondary and tertiary structure of a protein, given its primary, which do so significantly faster and cheaper.

For the purpose of this dissertation a CNN with a HFO algorithm was utilized in order to predict the secondary structure of proteins based on inputs (embeddings) that were extracted from a language model (BERT). The results obtained for this combination, for the CB513 dataset were an overall Q3 accuracy of 81.76% for a single fold and 93.65% for 10-fold cross validation with ensembles and random forest filtering technique. The SOV score was 75.17 and 89.63 respectively. Moreover for the PISCES dataset, the Q3 accuracy was 81.80% for a single fold and 87.13% for 5-fold cross validation with ensembles and random forest filtering, while the SOV score was 79.29 and 84.28 respectively. In addition the results for the independent testing dataset (CASP13) with the system trained on PISCES was 78.92% for single fold and 90.69% with ensembles and random forest. The SOV score was 72.17 for single fold and 87.94 with ensembles, random forest and external rules. The results for the CASP13 dataset when the system was trained with the CB513 dataset were 78.74% for single fold and 90.54% with ensembles and random forest. The SOV score was 71.33 for single fold and 87.47 for ensembles, random forest and external rules. Additional experiments showed that the bigger the window variable gets, the better the Q3 accuracy of the test set is. The reason is that a larger window will use more neighboring secondary structures to build the post processing datasets, thus being able to capture longer connections between secondary structures leading in higher Q3 accuracy up to 98% and SOV score of almost 97.

Contents

Chapter 1

1 Introduction

1.1 Protein Secondary Structure Prediction problem

Proteins are highly complex substances that are present in all living organisms. Proteins are of great nutritional value and they directly contribute in the chemical processes essential for life. Proteins are made up of amino acid residues linked together by peptide bonds. Several hundred amino acids are found in nature, but only 20 amino acids appear in the human body. Proteins of similar function have similar amino acid composition and sequence [1]. However in order to explain the function of a protein we need to know how amino acids composing a protein interact and fold in a three-dimensional (3D) space. This 3D structure determines the function of each protein. The field of studying about protein structures and functions can contribute to enhance food supplements, drugs and antibiotics.

In general, there are four levels of protein structure. The Primary, Secondary, Tertiary and Quaternary structures, and each one differs from the other by the degree of complexity in the polypeptide chain. The primary structure is the linear sequence of amino acids that form a specific protein. Secondary structure refers to the folding of one polypeptide chain of a protein. There are 8 different types of secondary structures appeared in proteins, but we can group them into 3 wider categories, the alpha (a) helix, the beta (β) pleated sheet, and coil/loops. The tertiary structure is a three-dimensional (3D) representation of the polypeptide chain, that determines the specific function of a protein. Finally, the quaternary structure is the structure of a protein macromolecule formed by interactions of multiple polypeptide chains [2].

At the moment there are millions of proteins with known primary structure, but only for a small fraction of them we know their secondary and tertiary structure. The reason is that methods used to determine secondary and tertiary structures demand a serious amount of time and money. To be precise the cost of structure determination using conventional means, such as X-ray crystallography or Nuclear Magnetic Resonance, can vary between \$100,000 – \$300,000 per protein structure [3]. Thus this process is made the bottleneck of the pipeline for the protein function determination. There was therefore an emerging need to develop methods for predicting the secondary and tertiary structures, which are considerably cheaper and require less time than the experimental methods.

This dissertation tries to predict the secondary structure of proteins using embeddings from Language Models and Convolutional Neural Networks with Hessian Free Optimisation algorithm. Embeddings are extracted from two different language models, ELMo and ProtBERT and are used as inputs to the CNN.

1.2 Importance of PSSP

The solution of the PSSP problem is very important because the secondary structure is essential to determine the tertiary structure, which gives information about how proteins function. The experimental methods used for determining the tertiary structure of proteins are extremely expensive in both time and money, which led to the study of just a small portion of known proteins. As a result, the scientific community has information about the functions of just a small (a few thousands) of proteins, compared to the millions of proteins that exist.

Furthermore, this means that the PSSP can help identify the tertiary structure of a protein with higher accuracy and less effort. It is very important to note that the functions of a protein are based on the 20 amino acids that compose a protein, which is the main reason why the research in this field is very important. Understanding how these molecules fold around space, assemble and function can help to understand why people are getting older, why they suffer from dangerous diseases and viruses (such as cancer), how can a cure for a disease be found (like the cure for covid-19), and other 'difficult to answer' questions.

The proteins' functions are related with their structure, which depends on both the physical and chemical parameters of these molecules. Bioinformatics is an interdisciplinary field that develops methods and software tools for understanding biological data. It combines knowledge from biology, computer science, information engineering, mathematics, and statistics to analyze and interpret biological data.

1.3 Previous Research on PSSP

Predicting the secondary structure of proteins is a problem that researchers have been working on for more than six decades. Multiple machine learning algorithms have been developed over the years that are dedicated to this problem. Algorithms using only machine learning techniques have reported three-state accuracy around 85% whereas algorithms that rely on sequence-based structural similarities have achieved Q3 accuracy over 90% [4].

Figure 1.1 shows the number of publications on the PSSP problem over a period of 40 years, as well as the cumulative number of publications. According to the graph the cumulative number of publications for the PSSP problem increased significantly between 1973 and 2015. Also we can observe that this is a problem that concerns the scientific community since new publications come out every year.

Table 1.1 shows the Q3 accuracy of various methods trying to solve the PSSP problem in chronological order. Those algorithms are better described here [5].

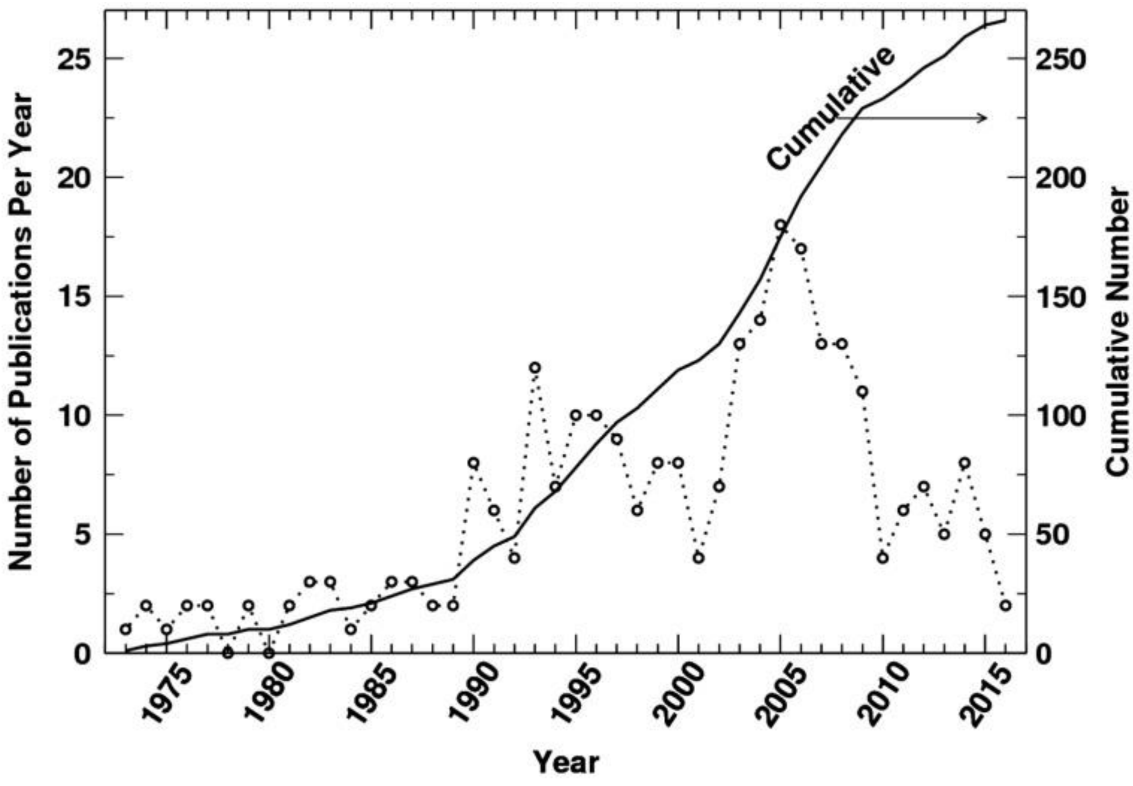


Figure 1.1: Number of publications for PSSP per year [6]

NO.	METHOD	Q3 ACCURACY(%)
1	Feedforward Fully Connected NN (Qian and Sejnowski,1988)	63.30
2	PHD (Rost,2001;Rost and Sander,1993)	71.40
3	NNSSP (Salamov and Soloveyev, 1997)	68.41
4	DSC (King and Sternberg, 1996)	71.95
5	PREDATOR (Frishman and Argos, 1997)	68.60
6	Consensus (Cuff and Barton, 1999)	72.70
7	BRNN–Backpropagation (Baldi et al, 1999)	76.00
8	LAD (Jacek et al. ,2005)	70.60
9	MASSP3 (Giuliano et al., 2005)	76.10
10	Evolutionary method for learning HMM structure (Won et al.,2007)	65.00
11	Two-Stage method (Fadime et al., 22007)	74.10
12	Cascade BRNN (Jinmiao and Narendra, 20070)	74.38
13	Deep Convolutional Neural Field (Wang et al., 2016)	83.00
14	Convolutional Neural Networks (Pavlidis, 2016)	40.00
15	LSTM–BRNN (Heffernan et al., 2017)	84.00
16	MUFold – SS (Fang et al., 2018)	86.49
17	Feed Forward NN with HFO (Charalambous et al., 2020)	80.40
18	Convolutional Neural Network with SVM filtering (Dionysiou et al., 2020)	81.00

Table 1.1: Methods used for PSSP in chronological order

Chapter 2

2 Background

2.1 Biology Background

2.1.1 The biological Role of Proteins

Proteins are macromolecules made of amino acids. They are coded for by our genes and form the basis of living tissues. They also play a central role in biological processes. For example, proteins catalyse reactions in our bodies, transport molecules such as oxygen, keep us healthy as part of the immune system and transmit messages from cell to cell [7].

Food consumption is the main source of proteins for the human body. The digestive system breaks down the consumed food into amino acids, which enter the blood stream. Cells of the human body gather amino acids from the blood stream to create all the essential proteins to perform specific functions. If there is a shortage of amino acids in the blood stream, probably because of a poor diet with less proteins, the immune system will become weak, causing dizziness, exhaustion, or even serious diseases. That happens because to create the necessary proteins for the human body, the cells need enough amino acids, otherwise they will not be able to support the needs of the entire human body.

Understanding the structure of a protein and consequently its function will help in the development for better food supplements, drugs and antibiotics. Research or studies on existing proteins could also help solve numerous biological problems and treat diseases. The advancement of technology made this procedure easier due to the computational power we have.

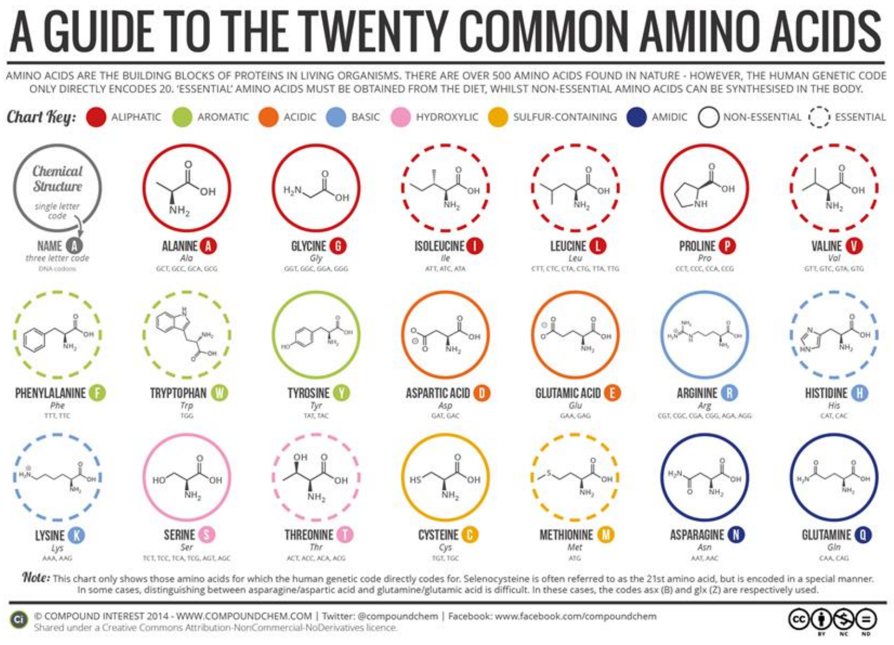
There are over thirty thousand (30.000) different proteins in the human body which perform a wide variety of functions. Table 2.1 describes some of the most important types of proteins and the functions they perform.

Type	Function Description	Example
Enzymes	They build and break down molecules. They are critical for growth, digestion, and many other processes in the cell. Without enzymes, chemical reactions would happen too slowly to sustain life.	Lactase
Structural	Proteins that strengthen cells, tissues, organs and more.	Collagen
Signaling	Proteins that allow cells to communicate with each other.	Insulin
Regulatory	Proteins that bind the DNA to turn genes on and off.	Estrogen
Transport	Transport proteins move molecules and nutrients around the body and in and out of cells.	Hemoglobin
Sensory	Proteins that help us learn about our environment. They help us to detect light, sound, touch, smell, taste, pain and heat.	Opsin
Motor	They keep cells moving and changing shape. They also transport components around inside cells.	Myosin
Defense	Defense proteins help organisms fight infection, heal damaged tissue, and evade predators.	Antibodies
Storage	Proteins that store nutrients and energy-rich molecules for later use.	Casein

Table 2.1: Types of proteins and their function [8]

2.1.2 Amino Acids

Proteins consist of hundreds or even thousands of amino acids, which are organic compounds that contain amine (NH2) and carboxyl (COOH) functional groups. Amino acids and proteins are the building blocks of life. There over 500 amino acids found in nature, but only 20 of those amino acids are needed to make all the proteins found in the human body. Figure 2.1 illustrates the 20 common amino acids with their structure. Amino acids can be divided into two groups, the essential and the non-essential. Essential amino acids have to be obtained from our diet, where as non-essential amino acids can be synthesized from the human body [9].


A GUIDE TO THE TWENTY COMMON AMINO ACIDS

AMINO ACIDS ARE THE BUILDING BLOCKS OF PROTEINS IN LIVING ORGANISMS. THERE ARE OVER 500 AMINO ACIDS FOUND IN NATURE - HOWEVER, THE HUMAN GENETIC CODE ONLY DIRECTLY ENCODES 20. 'ESSENTIAL' AMINO ACIDS MUST BE OBTAINED FROM THE DIET, WHILST NON-ESSENTIAL AMINO ACIDS CAN BE SYNTHESISED IN THE BODY.

Chart Key: ALIPHATIC AROMATIC ACIDIC BASIC HYDROXYLIC SULFUR-CONTAINING AMIDIC NON-ESSENTIAL ESSENTIAL
Chemical Structure single letter code
NAME A three letter code DNA codons
ALANINE A Ala GCT, GCC, GCA, GCG
GLYCINE G Gly GGT, GGC, GGA, GGG
ISOLEUCINE I Ile ATT, ATC, ATA
LEUCINE L Leu CTT, CTC, CTA, CTG, TTA, TTG
PROLINE P Pro CCT, CCC, CCA, CCG
VALINE V Val GTT, GTC, GTA, GTG
PHENYLALANINE F Phe TTT, TTC
TRYPTOPHAN W Trp TGG
TYROSINE Y Tyr TAT, TAC
ASPARTIC ACID D Asp GAT, GAC
GLUTAMIC ACID E Glu GAA, GAG
ARGININE R Arg CGT, CGC, CGA, CGG, AGA, AGG
HISTIDINE H His CAT, CAC
LYSINE K Lys AAA, AAG
SERINE S Ser TCT, TCC, TCA, TCG, AGT, AGC
THREONINE T Thr ACT, ACC, ACA, ACG
CYSTEINE C Cys TGT, TGC
METHIONINE M Met ATG
ASPARAGINE N Asn AAT, AAC
GLUTAMINE Q Gln CAA, CAG

Note: This chart only shows those amino acids for which the human genetic code directly codes for. Selenocysteine is often referred to as the 21st amino acid, but is encoded in a special manner. In some cases, distinguishing between asparagine/aspartic acid and glutamine/glutamic acid is difficult. In these cases, the codes asx (B) and glx (Z) are respectively used.

Figure 2.1: The twenty amino acids found in human body [9]

2.1.3 Protein Structures

As mentioned before, there are four levels of protein structure. The Primary, Secondary, Tertiary and Quaternary structures, and each one differs from the other by the degree of complexity in the polypeptide chain. Knowing only the number and type of amino acids of a protein are not enough, since order and layout of amino acids are needed to determine the three-dimensional structure and consequently the function of the protein. Below we can take a closer look at the four different structures of a protein.

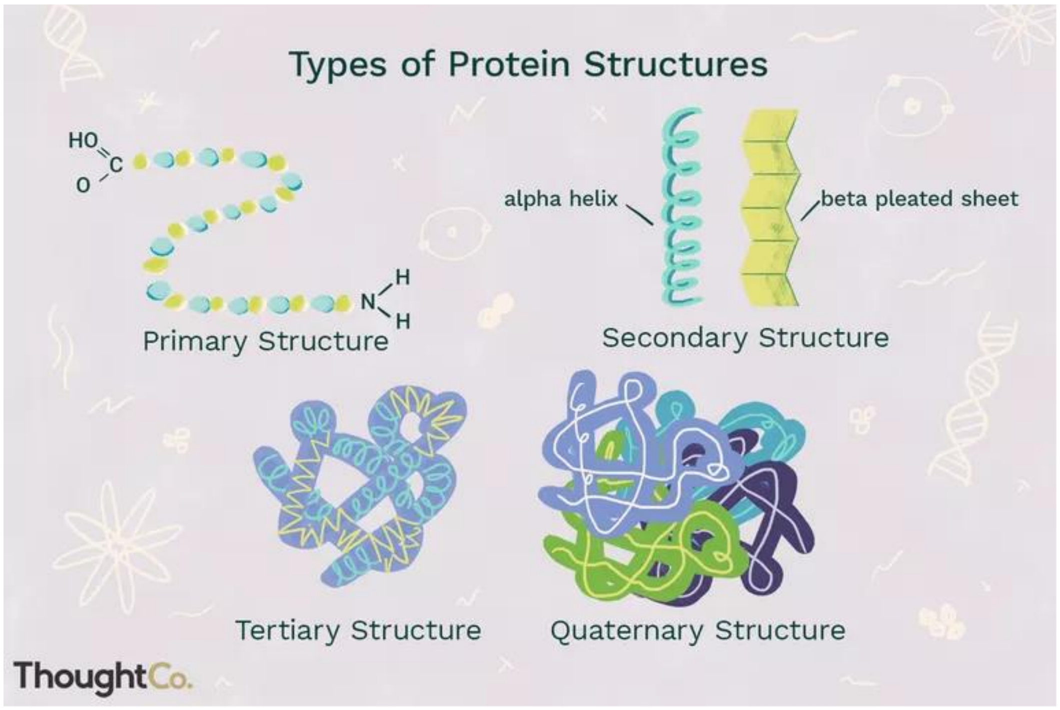


Figure 2.2: Protein structures illustration [2]

2.1.3.1 Primary Structure

Primary structure describes the order in which amino acids are linked together to form a protein. The order of amino acids in a polypeptide chain is unique and specific in each protein. The sequence of amino acids for every protein is determined by the gene encoding . Changing just a single amino acid will generate a different protein [2]. Figure 2.2 illustrates the primary structure of a protein.

2.1.3.2 Secondary Structure

The secondary structure is the local folding of a polypeptide chain due to hydrogen bonds that are formed between the backbone-chain peptide groups. There are two main categories of secondary structures observed in proteins. The first category is the alpha (α) helix structure. This structure resembles a coiled spring and is secured by hydrogen bonding in the polypeptide chain. The second category of secondary structure in proteins is the beta (β) pleated sheet. This structure appears to be folded or pleated and is held together by hydrogen bonding between polypeptide units of the folded chain that lie adjacent to one another [2]. Figure 2.2 show the two main categories of secondary structure.

2.1.3.3 Tertiary Structure

The tertiary structure of a protein refers to its three-dimensional structure, which is represented by the three-dimensional coordinates of each atom of the protein. The tertiary structure of a protein depends on multiple elements of the secondary structure, and is generally the result of side chain interactions between various amino acids. Therefore, the amino acid sequence of a protein (primary structure), forms the secondary structure of the protein and then the local folding of the protein determines the tertiary structure of the protein, thus the final shape and function of the protein.

2.1.3.4 Quaternary Structure

The quaternary structure of a protein is the three-dimensional structure consisting of the aggregation of two or more individual polypeptide chains. Each polypeptide chain can be referred as a subunit. Proteins with quaternary structure, consist of more than one of the same type of protein subunit. For example Hemoglobin contains four subunits, two alpha subunits and two beta subunits.

2.2 Artificial Neural Networks Background

2.2.1 Origins of Artificial Neural Networks

Artificial neural networks are computational systems that try to mimic the behavior of biological neural networks. Biological neural networks are found in almost every living creature that has the ability to adapt in a changing environment. The term “neural” is derived from the basic functional unit of the nervous system called “neuron”. Neurons are located in multiple parts of the human body.

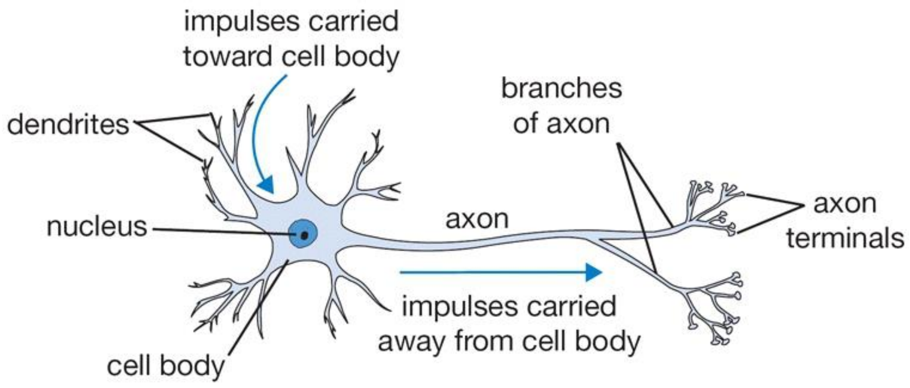

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Figure 2.3: Biological Neuron Illustration [10]

A biological neural network consists of multiple neurons that receive, process and transmit information between each other. A biological neuron consists of three main parts. The dendrites which receive input signals from other neurons. The cell body that adds all the input signals and triggers a response if the sum is higher than a threshold, and the axons that transmit the signal from the cell body to other neurons via synapsis connections. The impulses carried toward or away from the cell body are as shown in Figure 2.3.

An Artificial Neural Network (ANN) has the same architecture with a biological neural network. An ANN has nodes that represent neurons, edges between neurons instead of synaptic connections, and a threshold function that determines the output of the network.

Through the years a variety of ANN have been developed in order to tackle different problems and some of them are going to be discussed in the following section.

2.2.2 Variations of Artificial Neural Networks and Optimizers

2.2.2.1 McCulloch and Pitts (McP)

The first ANN model was suggested by Warren McCulloch and Walter Pitts in 1943. This artificial neuron has a very simple design and was based on a single biological neuron of the human brain. The McP model is a binary threshold unit. It computes a weighted sum of its inputs and fires a one or a zero depending on whether the sum is above or below a certain threshold. The weights of this neuron can be varied enabling it to perform arbitrary logic operations and indeed making the neuron adaptable.

$$y = \begin{cases} 1 & \text{if } w \cdot x > s \\ 0 & \text{otherwise} \end{cases}$$

Equation 2.1: McP equation

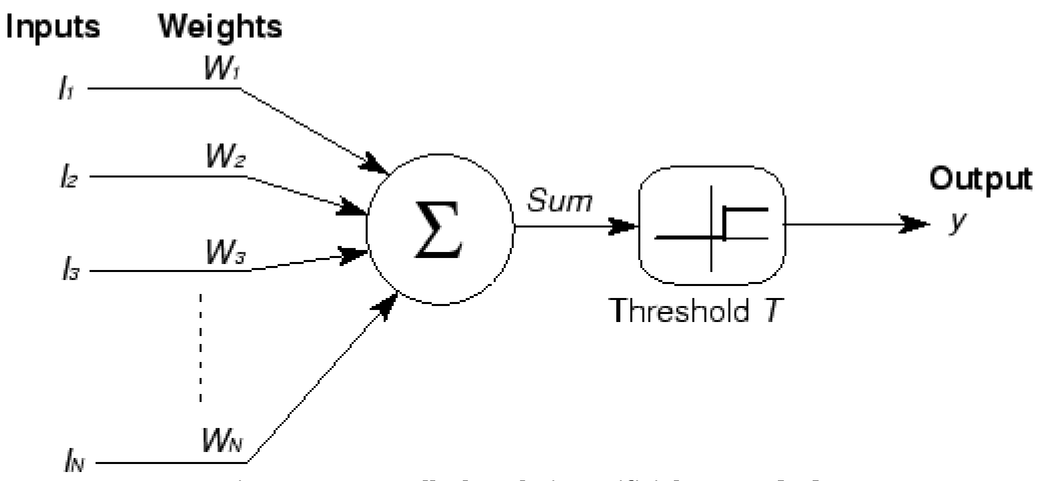


Figure 2.4: McCulloch and Pits artificial neuron [11]

In Figure 2.4 we can see the basic components of a McP artificial neuron. The inputs and the weights of the neuron can be written as two vectors. The dot product of those two vectors is then compared with the threshold to determine if the output is one or zero (Equation 2.1). Weights are key components in ANN since are the ones that make our model capable to learn. Usually weights are initialized with random values and through the training process are altered until the reach an optimal value.

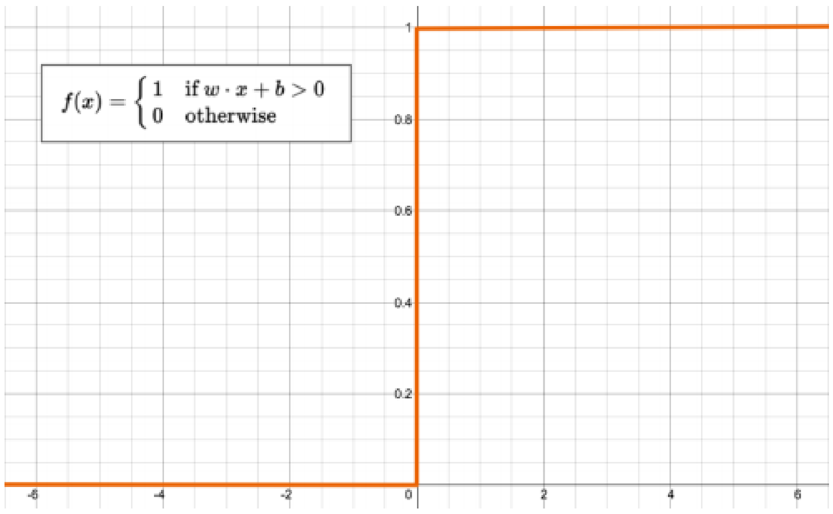
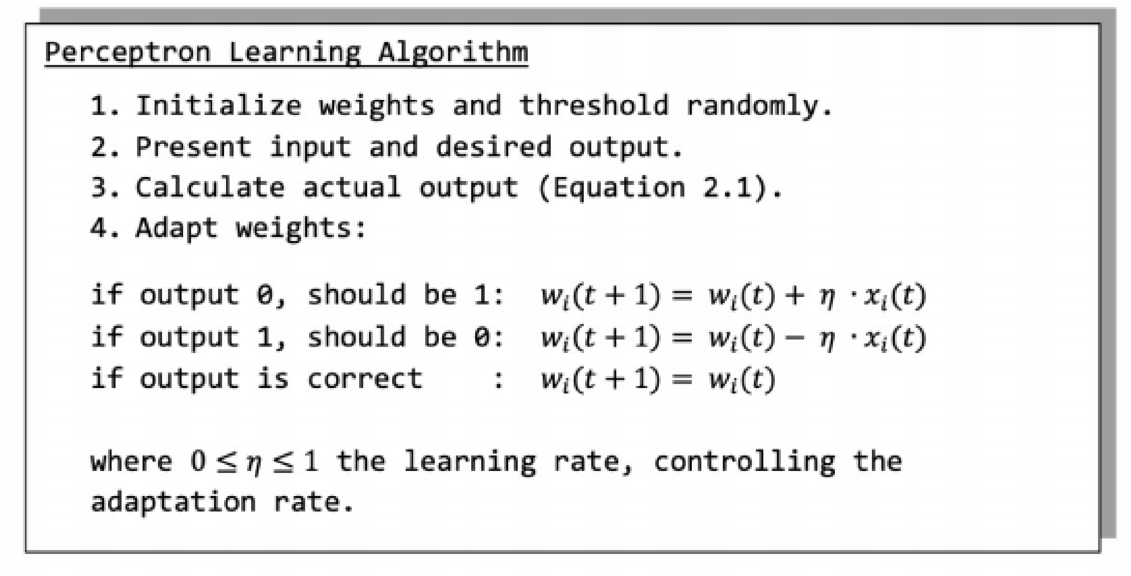


Figure 2.5: The step or heaviside threshold function

McP neurons are trained using the Perceptron Learning Algorithm (Algorithm 1). This algorithm modifies the weights when the target and predicted outputs are different, whereas if target and predicted outputs are the same the weights remain the same. The drawback using the perceptron algorithm was that it could only solve linearly separable patterns.



```
Perceptron Learning Algorithm

   1. Initialize weights and threshold randomly.
   2. Present input and desired output.
   3. Calculate actual output (Equation 2.1).
   4. Adapt weights:

   if output 0, should be 1:  $w_i(t+1) = w_i(t) + \eta \cdot x_i(t)$
   if output 1, should be 0:  $w_i(t+1) = w_i(t) - \eta \cdot x_i(t)$
   if output is correct    :  $w_i(t+1) = w_i(t)$

   where $0 \leq \eta \leq 1$ the learning rate, controlling the
   adaptation rate.
```

Algorithm 1: Perceptron Learning Algorithm [5]

2.2.2.2 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron is a class of feed-forward neural networks. MLP was design in order to solve problems that were not linearly separable. They consist of multiple McCulloch and Pitts neurons which form layered feed forward networks (Figure 2.6).

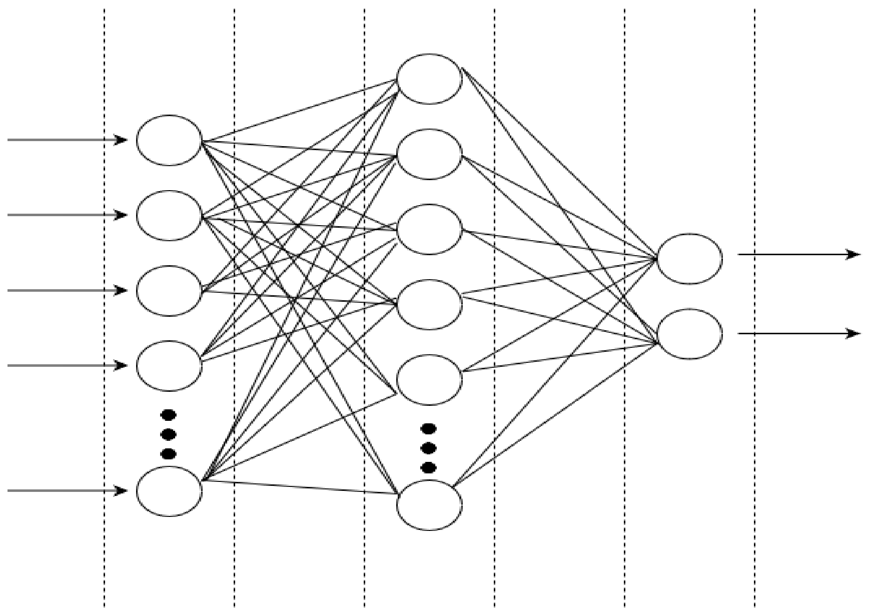


Figure 2.6: Multi-Layer Perceptron with one Hidden Layer Illustration

An MLP neural network consists of an input layer, at least one hidden layer and an output layer. Hidden and output layers are active, whereas the input layer is inactive since it only forwards the data to the network. Each layer has one or more neurons and an independent neuron unit, also known as ‘bias’, which has a constant input value of one (1). The role of the bias unit is to help the network adapt more effectively to the provided data.

Choosing the number of hidden layers and the number of neurons per layer is very important as it specifies the complexity of the network and thus the problems it can solve. The Kolmogorov Theorem says that having three active layers (2 hidden layers and the output layer) are enough to form any arbitrary complex shape that is capable of separating any classes (Figure 2.7). In general every neuron of the first active layer adds a new decision line that separates the data into classes. Every neuron of the second active layer combines the decision lines, from the first active layer, forming convex regions. Finally the third active layer forms the arbitrary complex shapes that separate the classes.

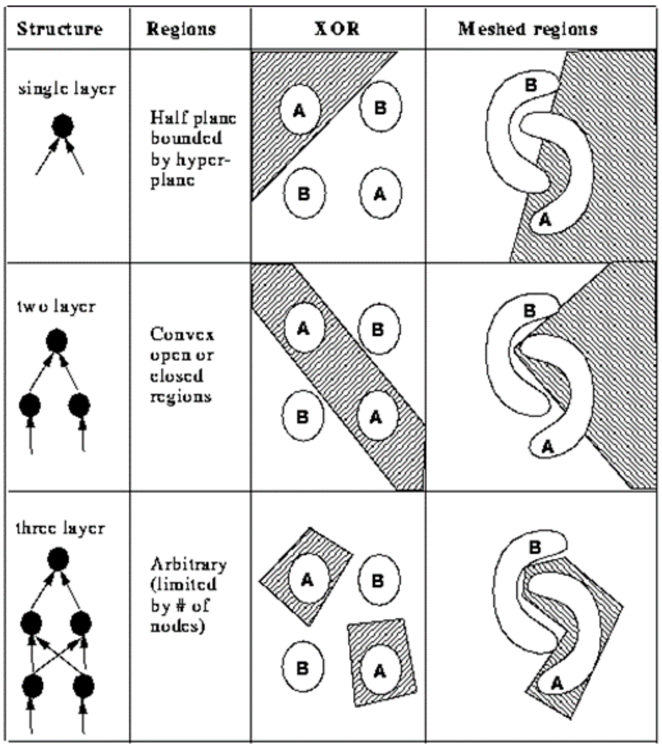

Structure
Regions
XOR
Meshed regions
single layer
Half plane bounded by hyper-plane
A
B
B
A
B
A
two layer
Convex open or closed regions
A
B
B
A
B
A
three layer
Arbitrary (limited by # of nodes)
A
B
B
A
B
A

Figure 2.7: The shape of regions in pattern space that can be separated by a Multi-Layer Perceptron [38]

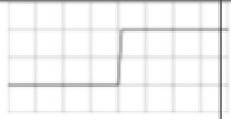
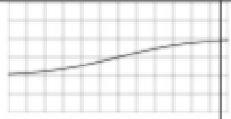
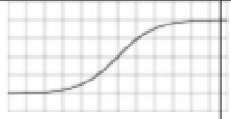
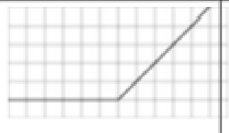
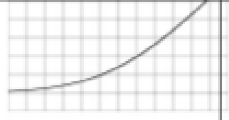
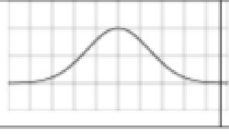
Name	Plot	Equation	Derivative	Range
Heaviside		$f(x) = \begin{cases} 1\ if\ x > 0 \\ 0\ otherwise \end{cases}$	$f'(x) = \begin{cases} 0\ if\ x \neq 0 \\ ?\ if\ x = 0 \end{cases}$	{0,1}
Logistic / Sigmoid		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	(0,1)
TanH		$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	$f'(x) = \frac{1}{x^{2} + 1}$	(-1,1)
Rectified linear unit (ReLU)		$f(x) = \begin{cases} 0\ if\ x < 0 \\ x\ otherwise \end{cases}$	$f'(x) = \begin{cases} 0\ if\ x < 0 \\ 1\ otherwise \end{cases}$	[0,∞)
SoftPlus		$f(x) = \ln(1 + e^{x})$	$f'(x) = \frac{1}{1 + e^{-x}}$	(0,∞)
Gaussian		$f(x) = e^{-x^{2}}$	$f'(x) = -2xe^{-x^{2}}$	(0,1)

Table 2.2: List of activation functions [5]

Another advantage that multilayer perceptron have is the use of a variety of activation functions. McP neurons use a specific threshold activation function (step function) while MLP neurons can use any arbitrary activation function (Table 2.2). This is the reason why McP can only perform binary classification, while MLP can perform regression or classification, depending on the selected activation function. Furthermore, activation functions provide an indication to the network whether the outputs are closer or further of the expected outputs, which helps the network adjust the weights accordingly, to improve predictions.

Almost the same process as McP is followed to produce the output of an MLP network. In each active layer the dot product of the weights and the respective inputs is calculated and then the bias term is added. The value calculated as then passed in an activation function (Equation 2.2). The outputs produced are then fed as inputs to the next active layer if there is one.

$$y = f\left(w^T x + b\right)$$

Equation 2.2: MLP equation that calculates output

Gradient Descent

The main objective when creating an ANN is to adjust the weights of the network in order to make the best predictions possible. To evaluate how good the predictions are, an error function is utilized, like the mean squared error (MSE) that shows how close the predictions are with respect to the target outptus.

$$\mathrm{MSE} = \frac{1}{n}\sum_{k=1}^{n}(t_k - o_k)^2$$

Equation 2.3: Mean square error equation (n is the number of output neurons, t_k is the target output and o_k is the actual output for neuron k)

Gradient descent is a widely used optimization algorithm used for training neural networks. It is an iterative algorithm that is able to minimize an error function by moving in the direction of the steepest descent, which is defined as the negative of the gradient. To minimize this error, weight vectors are adjusted according to the negative of the derivative of the error value, with respect to each weight (Equation 2.4).

$$\Delta w_{ij} = -n\frac{dE}{dw_{ij}}$$

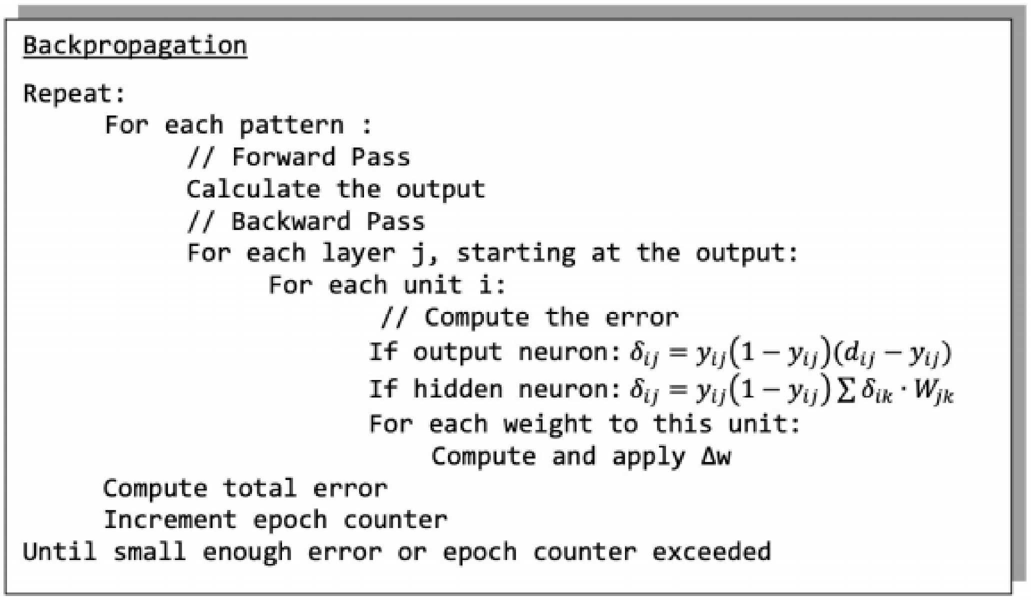
Equation 2.4: Gradient descent change in weights

Backpropagation Algorithm (BP)

Adjusting the weights using the gradient descent algorithm demands knowing the predicted and target outputs. This is feasible only in the output layer where both of the values are known, thus only the weights between the last hidden layer and the output can be adjusted. The backpropagation algorithm, which solves this issue, propagates the error from the output layer back to the last hidden layer, which then does the same until all the weights are updated.

The BP algorithm needs two passes in order to update all the weights. The first pass is a forward pass that based on a given input calculates the predicted output, and a backward pass that calculates and propagates the error to the previous layers (Algorithm 2). This process is repeated until all of the patterns have been passed into the network (one epoch).

The goal is to feed the neural network all the input patterns several times until the error decreases to a specific value or until a number of epochs are completed.



```
Backpropagation

Repeat:
    For each pattern :
        // Forward Pass
        Calculate the output
        // Backward Pass
        For each layer j, starting at the output:
            For each unit i:
                // Compute the error
                If output neuron: $\delta_{ij} = y_{ij}(1 - y_{ij})(d_{ij} - y_{ij})$
                If hidden neuron: $\delta_{ij} = y_{ij}(1 - y_{ij}) \sum \delta_{ik} \cdot W_{jk}$
                For each weight to this unit:
                    Compute and apply $\Delta w$
    Compute total error
    Increment epoch counter
Until small enough error or epoch counter exceeded
```

Algorithm 2: Backpropagation Algorithm (δij is the error signal, yij is the predicted output, dij is the target output of neuron i of layer j. The δik is the same as δij but for the previous iteration of the algorithm).

2.2.2.3 Convolutional Neural Networks (CNN)

A Convolutional Neural Network is a class of deep artificial neural networks, which is most commonly applied to analyze visual imagery. CNN have a wide range of applications, from image and video recognition, recommender systems, image classification to medical image analysis, and natural language processing (NLP). Convolutional Neural Networks, is an extension of traditional MLP, perform convolution instead of matrix multiplication in at least one of their layers.

CNN are trying to solve two major problems that multilayer perceptrons have. Firstly since MLP are fully connected networks, meaning that each neuron in one layer is connected to every neuron in the next layer, making them prone to overfitting. CNNs on the other hand take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns. Secondly CNNs are translation equivariant meaning that it preserves translations [12].

Architecture of Convolutional Neural Networks

CNNs consist of an input and an output layer, and multiple hidden layers. Those hidden layers are classified into convolutional layers and sub-sampling (pooling) layers. The activation function used is Rectified Linear Unit (RELU) and it is applied after a convolutional layer. CNNs end with a fully connected layer, usually a MLP network.

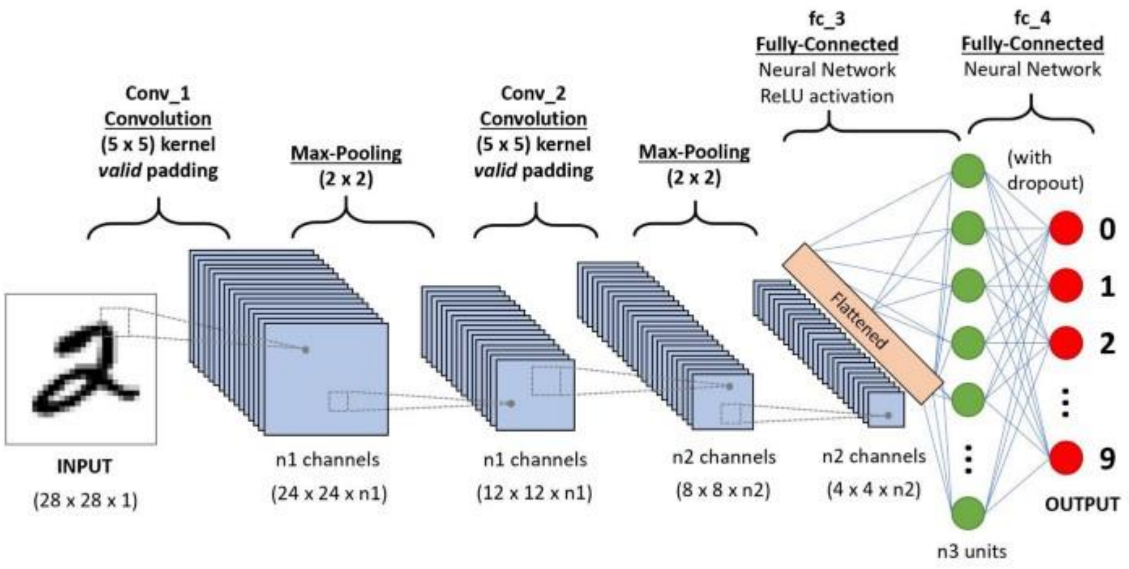


Figure 2.8: A CNN sequence to classify handwritten digits [13]

The input of a convolutional neural network is an image of size d × d × c, where d is the height and width of the image and c is the number of channels the input image has (e.g. RGB c = 3, GRAYSCALE c = 1). A convolutional layer has n filters (kernels) of size k × k × m, where k is smaller than the dimensions of the image and m can be either the same as the number of channels (c) or smaller. Convolutional layers convolve the input, which leads to n feature maps of size smaller or equal than d – k + 1, and pass their output to the next layer. If the next layer is a convolutional layer the same process is repeated, but if the next layer is a pooling layer, the feature maps are sub-sampled, typically averaging or maximizing above the same areas in feature maps of size p × p (p is between 2 and 5 depending on the size of the image).

Figure 2.8 illustrates a CNN which is used to classify handwritten digits. The diagram shows the different layers of a CNN (convolution, pooling, multilayer perceptron) and the feature maps that are extracted from each image (small squares). At the end of the CNN, there is a fully connected network (MLP) which is used to classify the handwritten digit.

Pooling layers are typically placed between two convolutional layers. The main scope of pooling layers is to reduce the dimensions of the feature maps produced by convolutional layers. Reducing the dimension of a feature map, reduces the number of parameters and the complexity of the network, which consequently reduces the total computation time of the network, as well as preventing the network from overfitting. However using sub-sampled layers may remove details that are useful to train the network resulting in poorer overall performance. There are several types of pooling layers such as min pooling, max pooling, average pooling and L2-normalization pooling. Figure 2.9 shows an example of max pooling with kernel size 2 × 2 and stride 2.

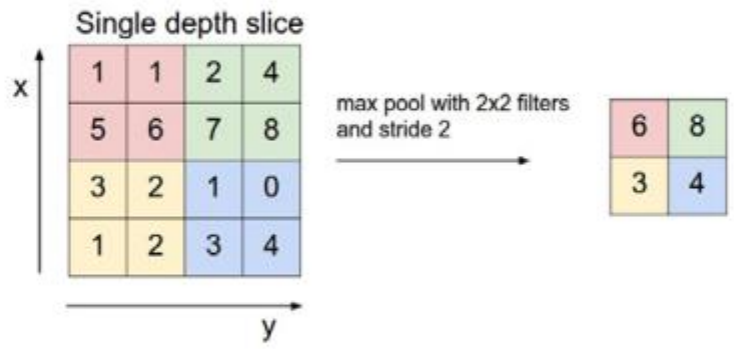
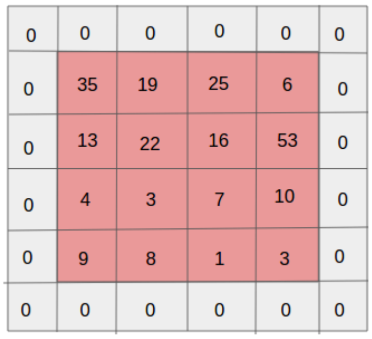


Figure 2.9: Max pooling example

Finally in some cases we may need to apply padding around the image in order to control the dimensions of the outputs produced from the convolution layers. There are several types of padding such as constant padding and zero padding. Figure 2.10 illustrates a 4 × 4 matrix that has zero padding and became a 6 × 6 matrix.



0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

Figure 2.10: Zero padding in a 4 x 4 matrix [14]

2.2.2.4 Line Search

Line search is one of the two basic iterative approaches used to find a minimum x* of an objective function. In ANN x represents the weights of the network and the objective function represents the error function. Line search in each iteration firstly finds a descent direction along which the objective function will be reduced and then computes a step size that determines how far we should move in that direction [15].

$$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

Equation 2.5: Line search update rule (a_k is the step size , p_k is the descent direction)

2.2.2.5 Conjugate Gradient (CG)

Conjugate Gradient is a line search method that in every iteration tries not to undo part of the moves done previously, by conjugating the directions of all previous moves. In an N-dimensional problem, conjugate gradient can converge in an optimal solution making at most N steps [16]. Figure 2.11 illustrates an example of a two-dimensional problem. Conjugate gradient converges in only 2 steps, whereas gradient descent needs more.

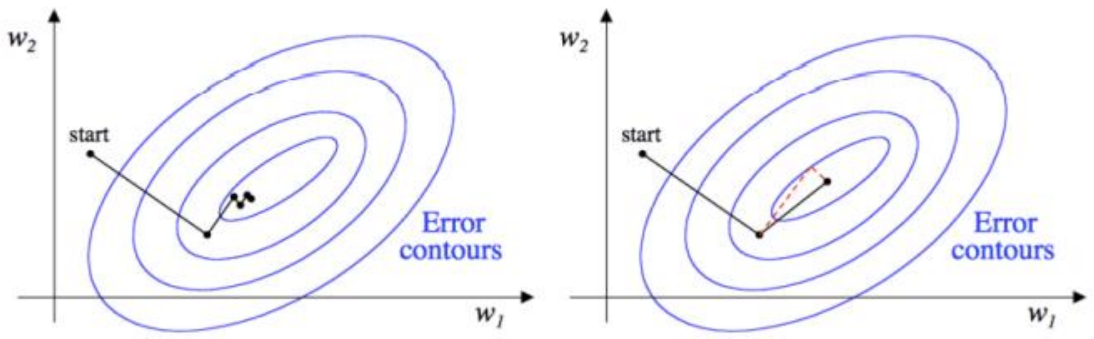


Figure 2.11:Gradient descent (left) vs Conjugate Gradient (right)

2.2.2.6 Newton's Method

An iterative method, originally used to find approximations of the roots of real-valued functions. Newton's Method is currently used in optimisation problems to find the maximum or minimum of a function. This method is considered a second-order optimisation method since it requires information about the second derivative of the optimisation function.

The difference between first and second order optimisation methods is that, while first-order algorithms provide a plane that is tangent to a point on the error surface, second-order methods provide a quadradic surface that hugs the curvature of the error surface. Consequently second-order algorithms can converge to a minimum faster.

A simple approach of Newton's method is by using only the first derivative of a function to find its root. Starting with a sub-optimal initial point x_0 and a function y = f(x), iterate as follows:

1. Set $x_i = x_0$
2. Find the equation for the tangent line at x_i
3. Find the point (x_{i+1}) of intersection between the tangent line and the x-axis
4. Find the projection of x_{i+1} on f(x)
5. Set $x_i = x_{i+1}$ and repeat from 2 until $f(x_i)$ < threshold

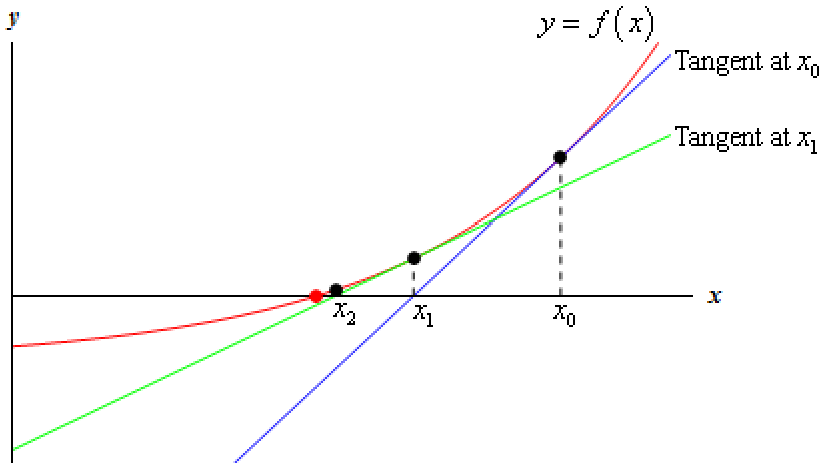


Figure 2.12: Illustration of Newton's method using first-derivative of a function

$$x_{n+1} = x_n - \frac{f(x_n)}{f\prime(x_n)}$$

Equation 2.6: Update rule for Newton's method using first-derivative of a function

Having an intuition about how Newton's method finds an approximation using a first-degree polynomial, we can proceed to examine the Newton's method for a second-degree polynomial function with one variable. A second degree polynomial is quadratic in nature and would need a second-order derivative to work with. Taylor approximation series are used.

Second-order Taylor expansion for x_0 is

$$f(x_0 + x) \approx f(x_0) + f'(x_0)x + f''(x_0)\frac{x^2}{2}$$

To minimize $f(x_0 + x)$ we find the derivative and equate to zero

$$\frac{d}{dx}\left(f(x_0) + f'(x_0)x + f''(x_0)\frac{x^2}{2}\right) = 0$$

$$f'(x_0) + f''(x_0)x = 0$$

$$x = -\frac{f\prime(x_0)}{f\prime\prime(x_0)}$$

In order to find the minimal of 'x' we iterate the process as follows:

$$x_{n+1} = x_n - \frac{f\prime(x_n)}{f\prime\prime(x_n)}$$

$$x_{n+1} = x_n - \left(f''(x_n)\right)^{-1} f'(x_n)$$

Figure 2.13: Update rule for Newton's method using second-order derivative of a function

The above update rule can be generalize when the objective function has multiple dimensions. First derivatives are replaced with gradients, and second derivatives with the Hessian matrix.

$$f'(x) \rightarrow \nabla f(x)$$

$$f''(x) \rightarrow \mathrm{H}(f)(x)$$

Which gives us:

$$x_{n+1} = x_n - (H(f)(x_n))^{-1} \nabla f(x_n)$$

Figure 2.14: Newton's method update rule

Using the Newton's method to find the optimal parameters of our network seems to be a huge advantage since by fitting a curve at a specific point and not a plane we can directly find the minima of the curvature, making the whole process faster [17]. However computing a Hessian matrix for a network with a big number of parameters in every iteration is inefficient, because of the amount of storage and computation needed. A solution to this problem is instead of calculating and storing the entire Hessian matrix, we can calculate an approximation the requires less computational resources and does not have to be stored.

$$
\boldsymbol{H(e)} = \begin{bmatrix} \frac{\partial^2 e}{\partial w_1^2} & \frac{\partial^2 e}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 e}{\partial w_1 \partial w_n} \\ \frac{\partial^2 e}{\partial w_2 \partial w_1} & \frac{\partial^2 e}{\partial w_2^2} & \cdots & \frac{\partial^2 e}{\partial w_2 \partial w_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 e}{\partial w_n \partial w_1} & \frac{\partial^2 e}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 e}{\partial w_n^2} \end{bmatrix}
$$

Figure 2.15: Hessian matrix of the error function with respect to the weights

2.2.2.7 Hessian Free Optimisation (HFO)

As mentioned before, computing the Hessian matrix for a large ANN with thousands to millions of parameters is not always possible due to the extremely high memory requirements and computational resources needed. The HFO method proposes solutions to these memory requirements which enables it to be effective to train NNs [18].

HFO is a variation of Newton's method. This algorithm, instead of calculating and storing the entire Hessian Matrix (H), calculates the dot product of H with an arbitrary vector u (Hu). It takes advantage of mathematical techniques, like finite differences, which computationally costs the same as a single gradient calculation. Usually the Gu product is used, where G is the Gauss-Newton matrix, an approximation of the Hessian Matrix [19]. While it seems pointless to use an approximation instead of the actual matrix, Gauss-Newton bypasses some of the problems that the Hessian may face, which could make the algorithm completely ineffective. In fact, even when those problems do not occur, the use of the G matrix consistently results in better search directions utilizing half the memory and running twice as fast, comparing to the usage of the Hessian matrix. A detailed analysis of the HFO method was described by Charalambous [20].

Chapter 3

3 Data Manipulation

3.1 PSSP Metrics

The main objective of the PSSP problem is to predict the secondary structure of proteins as accurately as possible, based on their respective primary structure. This dissertation utilizes supervised learning methods. Supervised learning demands the input data (primary structure) with their corresponding labels (secondary structure) in order to train an ANN to make predictions.

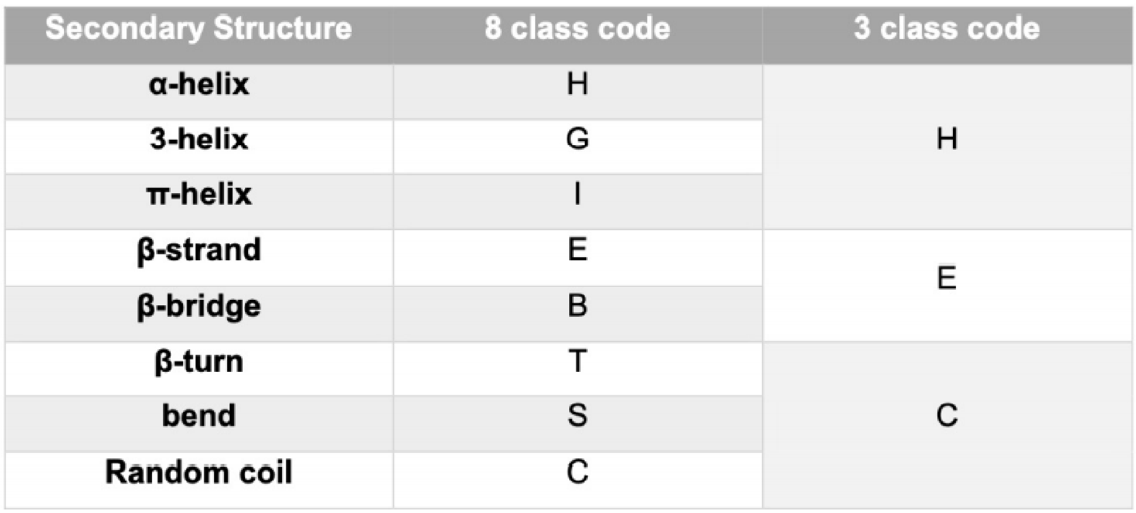
To be able to measure how good the predictions of the trained models, two different metrics were used. First the per residue Q3 accuracy, which measures the number of correctly classified amino acids, divided by the total number of amino acids. Second the Segment Overlap (SOV) score, is used to measure the overall quality of the predicted structure by comparing segments of classes.

For example, if the correct secondary structure of a protein starts with four (4) helices, followed by two (2) coils and then another four (4) helices and the predicted secondary structure is just ten (10) helices in a row, the two metrics will produce different results. The Q3 accuracy will be 80%, since eight out of ten amino acids were predicted correctly, whereas the SOV score would be just 48.

3.2 Protein Databases and DSSP

Documented proteins are organized in various protein databases such as the iProClass (Protein Information Resource), the PDBe (Protein Data Bank in Europe), the PDBj (Protein Data Bank in Japan) and the RCSB (RCSB Protein Data Bank). Protein databases contain various information about millions of proteins. Information includes name, length, structures and other biological information related to proteins. Protein information from the above databases were extracted in order to create the datasets of the PSSP problem.

The Dictionary for Secondary Structure of Proteins (DSSP) defined a standardized format of categorizing the secondary structures of a protein [21]. This format proposes eight (8) different classes of secondary structures, based on their shape and they are represented by a capital letter of the English Alphabet. The classes are, the α-helix (H), 3-helix (G), π-helix (I), β-strand (E), β-bridge (B), β-turn (T), bend (S), and random coil (C) for residues which are not in any of the other conformations (Table 3.1). The above eight (8) categories are usually grouped into three (3) broader classes that describe the nature of the shape of the specific local segment of the protein. For the purpose of this dissertation the 3-class classification is used. This includes the helix (H) conformations that contain the first three categories (H, G, I), the sheet (E) conformations, containing the next two categories (E,B), and finally Coil (C) conformations which contain everything else (T,S,C).

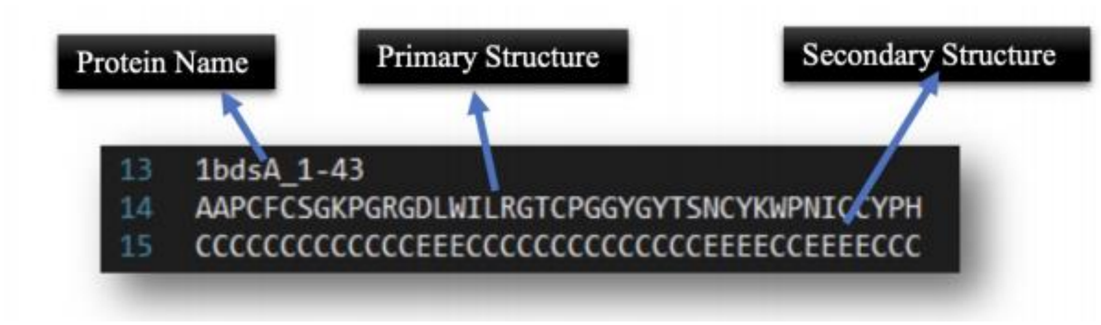


Secondary Structure	8 class code	3 class code
α-helix	H	H
3-helix	G	
π-helix	I	
β-strand	E	E
β-bridge	B	
β-turn	T	C
bend	S	
Random coil	C	

Table 3.1: Protein secondary structures abbreviations, grouped in 8 and 3 classes

3.3 Dataset Format

All the datasets used for this dissertation had records of a 3-line format per protein. The first line of each triplet has the protein name, the second line has the primary structure and the third line the original secondary structure of the protein. This project makes use of the primary structure of a protein, in order to extract the embeddings that will be used as inputs to the CNN. Moreover, the original secondary structure of the protein will be compared with the predicted secondary structure obtained from the trained model, to evaluate the model. An example of the triplet is shown in Figure 3.1.



```
1bdsA_1-43
AAPCFCSGKPGRGDLWILRGTCPGGYGYTSNCYKWPNICCYPH
CCCCCCCCCCCCCEEECCCCCCCCCCCCCCEEEECCEEEECCC
```

Figure 3.1: Example of a protein representation

3.4 Extracting embedding from Natural Language Processing

The innovating part of this dissertation was the use of embeddings, as inputs to the CNN, obtained from different language models (LMs). Two different Natural Language Processing models that were adapted to extract protein embeddings were used, the Sequence-to-Vector (SeqVec) [22] embedder and the Protein Bidirectional Encoder Representations from Transformers (ProtBERT) embedder [23]. The former was inspired from the ELMo language model [24] and the latter from the BERT [25] language model.

However, the final experiments were performed on the embeddings that were extracted from ProtBERT. The choice of ProtBERT was based on a preliminary testing of a Convolutional Neural Network (CNN) with the Subsampled Hessian Newton (SHN) optimization [26] (that we are using in this project), which resulted to a Q3 accuracy of 78% with the ProtBert embedding whereas with the SeqVec embedding the Q3 accuracy was around 71%. This is due to how the embedders extract the embeddings. As shown in Figure 3.2, ELMo can be considered a shallow bidirectional model whereas BERT is deep bidirectional. Thus ProtBert extracts stronger representations for every amino acid because it takes into account all of the other amino acids in the same sequence.

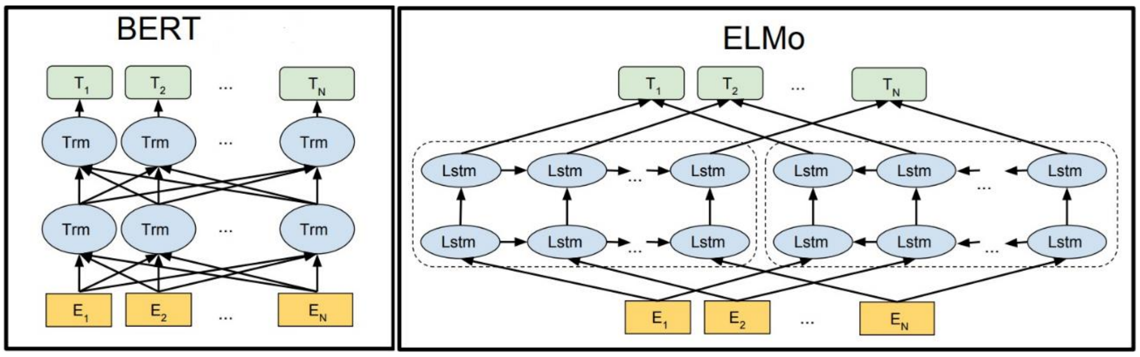


Figure 3.2: ELMo vs BERT architecture [27]

3.4.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) is the current state of the art Natural Language Processing (NLP) framework. BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context [25].

BERT is based on the encoder part of Transformers, left part of Figure 3.3. Transformers are the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned Recurrent Neural Networks (RNNs) or convolution [28].

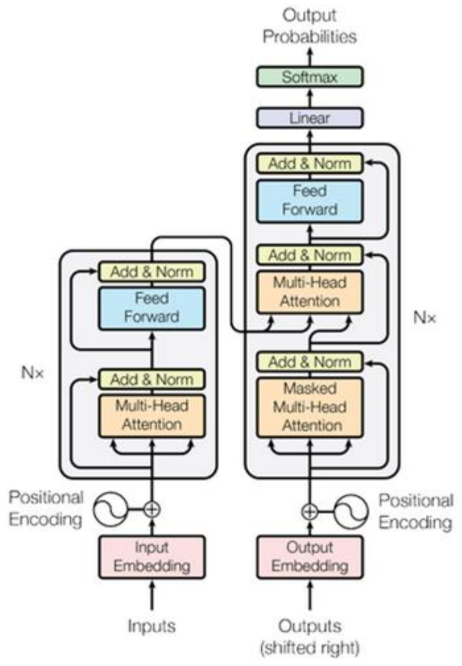


Figure 3.3: Key Components of a Transformer, Encoder (left) and Decoder (right) [28]

Self-attention or so called intra-attention, is a mechanism that relates different positions of a single sequence in order to compute a representation of the sequence. Simpler, attention is a metric that indicates how relevant each word in a sentence is with respect to the other words.

BERT provides the freedom to be pre-train to approach a variety of tasks, one of them is to extract embeddings from protein sequences (ProtBERT), that can be used for the PSSP problem.

3.4.2 ProtBERT

ProtBERT as mentioned before, it is simply the BERT model trained on a large corpus of unlabeled data. There are two different versions of the ProtBERT embedder, the first version was trained with the UniREF100 dataset that contained 216 million protein sequences, and the second with the BFD dataset that has 2.1 billion protein sequences [23]. This dissertation makes use of the version trained with the BFD dataset with the mindset that an embedder that is trained with more data will extract more powerful embeddings. The main concept behind this approach is to interpret protein sequences as sentences and every amino acid in a protein sequence as a word.

During the training phase of the ProtBERT model useful features and constrains are extracted from protein sequences. Finally each amino acid is represented with a vector of size 1024 , which is then transformed in a $32 \times 32 \times 1$ matrix in order to be used as input to the CNN.

3.5 CB513, PISCES and CASP13 Datasets

There are many datasets that can be used to train ANNs to solve the PSSP problem. For the purpose of this dissertation two widely used datasets were utilized, the CB513 [29] and the PISCES [30] datasets. These two datasets have been used for the PSSP problem by many researchers, thus making the results comparison possible.

The CB513 dataset contains 513 unique proteins with maximum similarity per protein pair of 25%. Maximum similarity threshold is really important in order to avoid the selection bias problem, where the data sample is not truly random and there is no even representation of all classes of the problem. In selection bias, the trained model learns some classes better than others, which results in poor classification/prediction on patterns in the testing set, which belong to a poorly represented class on the training dataset.
The PISCES dataset contains 8632 protein sequences with also maximum similarity per protein 25%. Both CB513 and PISCES datasets were used for training and cross validation.

The CASP13 (13th Critical Assessment of Protein Structures) dataset, was used for independent testing. CASP13 contains 40 protein sequences.

3.6 Training/Testing Set and Cross Validation

The training dataset is used for training the model so it can extract useful patterns in order to classify each training example into a specific class. However, it is necessary to have another dataset that will evaluate if the model has the ability to generalize. For this reason, a test dataset is used, which is independent from the training dataset, and its purpose is to measure the effectiveness of the network to classify new data, that has never seen before. In general, the 80-20 rule is used to split a given dataset into training and test set, meaning that 80% of the entire dataset is used as training dataset and 20% is used a test dataset. However different problems may produce better results when the dataset splitting is different.

Sometimes a simple training/testing set split is not enough to test the ability of a network to predict new data, since having a single test set may not give a good indication on how well the model generalizes new data. Another widely used method that solves this issue is the k-fold cross validation (Figure 3.4). This method splits evenly the data into k folds and trains k different models. Each model will have a unique fold selected as the test dataset and the rest k-1 folds will be used as the training dataset. The average test accuracy of all models is the cross validation accuracy.

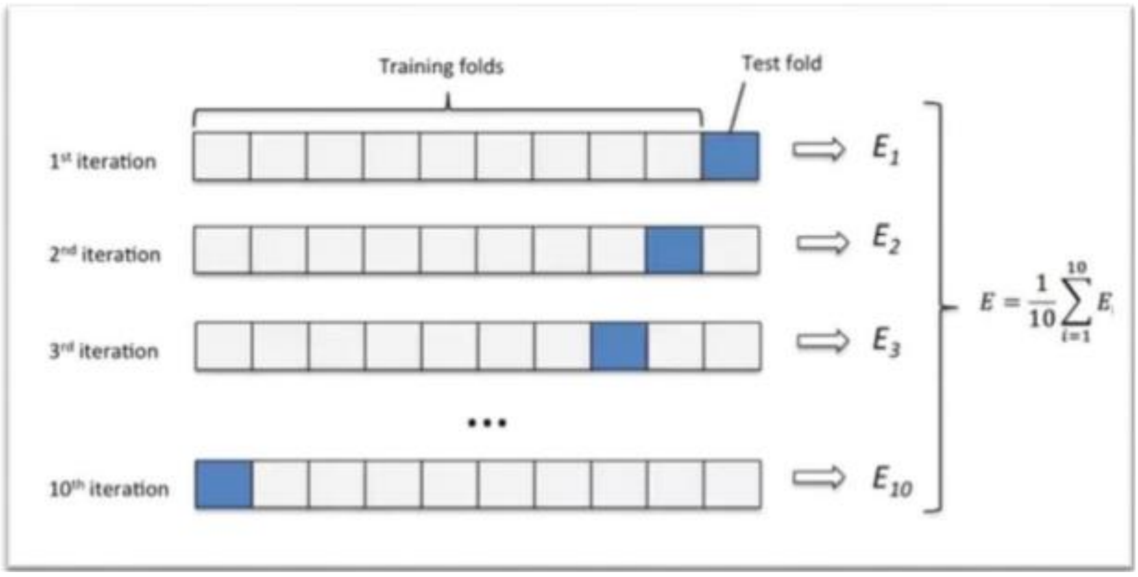


Figure 3.4: 10-fold cross validation

3.7 Ensembles

Ensemble learning, in machine learning, is a method that can be used to enhance the performance of a model. This method works by training multiple models and combining the predictions obtained, instead of just using the predictions of a single model. There are several types of ensemble methods, ranging from simple to advanced, such as the Bayesian Voting and the Bagging method [31]. In this dissertation a fairly simple approach was used.

For example, in the PSSP problem, if we have trained five different models each one of them will produce a prediction/output (H, E, C) for a specific input. The predictions are then compared with the method ‘winner takes all’ and the class with the most appearances is chosen as the final class of the specific input. In case of a tie between some of the classes, an arbitrary class from those tied is selected as the final class. Using the ensemble method it can remove random errors from the models, which may result in better overall predictions.

3.8 Filtering

Post processing filtering is another way of improving the predictions of a model. A filtering technique can either be applying another learning algorithm on the existing predictions [32], or by using some predefined external rules on the predictions. Both types were utilized in this dissertation in order to observe the impact of each filtering technique.

Applying another learning algorithm on the existing predictions demands the creation of a new training and test set. The sets are basically the original sets with the only difference being that instead of having as inputs the amino acids, they have the class of each amino acid. Also to be able to build the training and testing sets, a window variable needs to be determined that indicates the number of neighboring classes that are going to be used to construct the inputs, an example of how the window variable works is illustrated in Figure 3.5. The window variable plays a big role on the accuracy of the predictions. A bigger window variable may be able to capture long range connections/interactions between classes resulting in better Q3 accuracy.

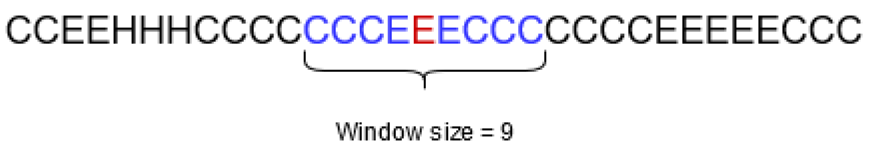


Figure 3.5: Example of how window variable is used

3.8.1 External Rules

External rules are based on empirical observations and are specific for the PSSP problem, targeting on improving the SOV score rather than the Q3 accuracy. They are also computationally cheap. Those external rules were derived from [33].

The external rules for the 3 class prediction (H, E, C) are:

1. Single 'H' or 'E' are replaced with 'C'
2. Sequence 'HEEH' is replaced with 'HHHH'
3. Sequence 'HEH' is replaced with 'HHH'
4. Sequence '!HH!' is replaced with '!CC!'

3.8.2 Decision Trees

Decision Trees (DTs) are a supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

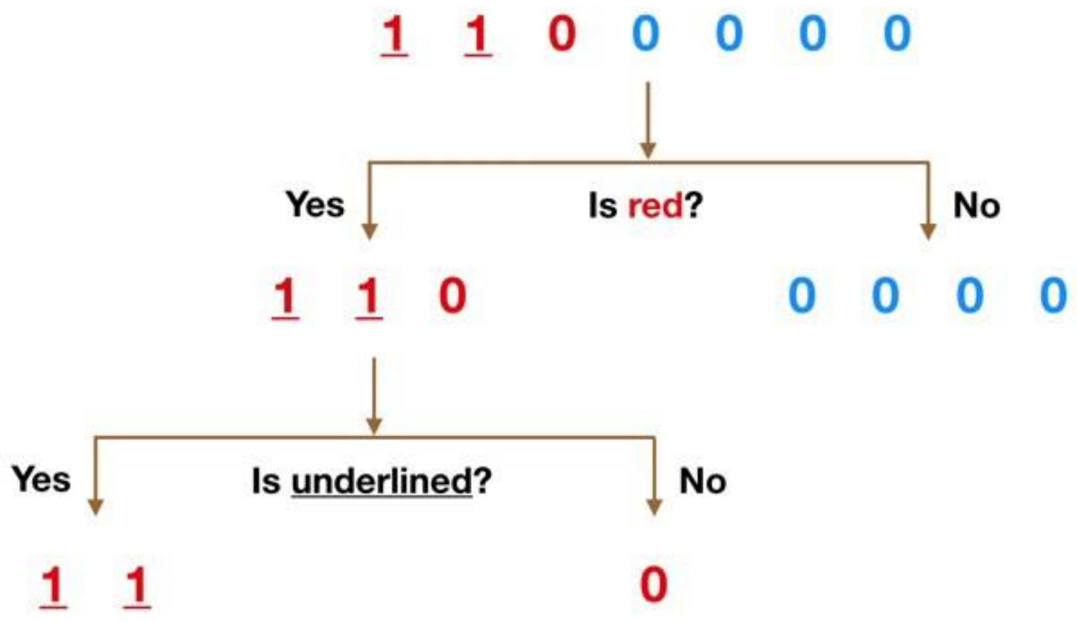


Figure 3.6: Decision tree example

For instance, we have a dataset that consists of numbers with different features as shown in Figure 3.6. We have two different classes (1 and 0), and we want to find a way to separate the classes using their features as decision rules. We need to create decision rules that will split the observations in a way that the resulting groups are as different from each other as possible but observations in the same group are as similar as possible [34].

3.8.3 Random Forest

Random forest is a classification method that combines the predictions produced from multiple decision trees. Each individual tree outputs a class prediction and the class with the most votes becomes the models prediction (Figure 3.7) [34].

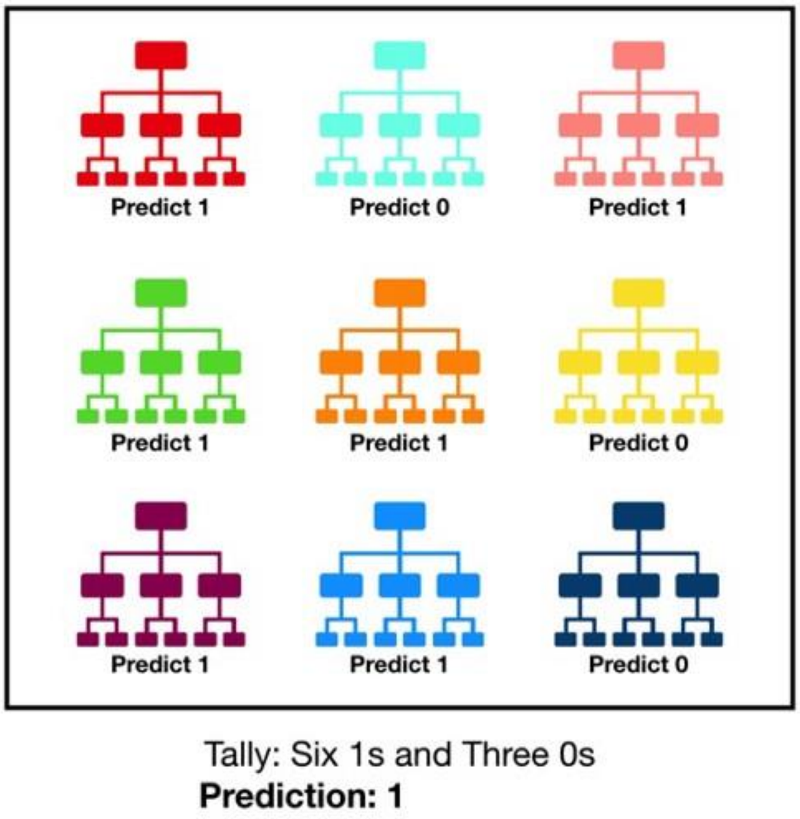


Figure 3.7: Example of random forest

The great performance of Random is based on just a simple concept. Having a large number of relatively uncorrelated models operating as a group, will outperform any of the individual models. In order to ensure that every decision tree inside a random forest is as diverse as possible with respect to the others two methods can be used. The first one is bagging, where each individual tree takes a random sample from the dataset to be trained with instead of the whole dataset. The second method is feature randomness, that restricts the number of features that can be used to split a node in each decision tree, by selecting a random subset of the available features.

3.8.4 Support Vector Machines

Support Vector Machines (SVMs) is a supervised learning algorithm that can be used for classification problems. SVMs trying to find a hyperplane that best separates a dataset into classes. If the data are not linearly separable, SVMs introduce additional features to the data with the hope that they will become separable in a higher dimension. Mapping data in higher dimensions is achieved by using non-linear kernels. Figure 3.8 shows an example of how SVM projects the initial data (left plot) in a higher dimension in order to be separable by a hyperplane (right plot).

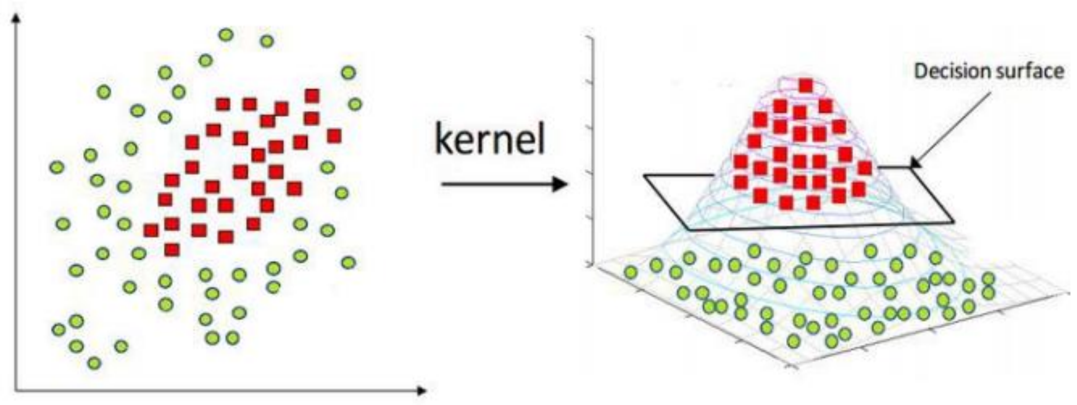


Figure 3.8: How SVM projects the problem in higher dimension in order to be separable

Chapter 4

4 Implementation

4.1 Subsampled Hessian Newton (SHN) Method

There are several studies that used variations of the Newton's method to train deep ANNs but were mostly used on fully connected feed forward neural networks (FFNNs). A new variation of the HFO, called the Subsampled Hessian Newton method, was introduced by Wang et al. [26] and can be applied to CNNs. This optimisation method along with a CNN was used by Leontiou [5] for the PSSP problem and showed very promising results. Due to the complexity of the algorithm it is better to refer to the original paper for a better understanding of the SHN algorithm. An outline of the algorithm is shown at Algorithm 3 below ((35) is Equation 4.1, (36) is Equation 4.2 and (37) is Equation 4.3).

```
Given initial $\theta$. Calculate $f(\theta)$;
while $\nabla f(\theta) \neq 0$ do
    Choose a set $S \subset \{1, \ldots, l\}$;
    Compute $\nabla f(\theta)$ and the needed information for Gauss Newton matrix-vector products;
    Approximately solve the linear system in (36) by CG to obtain a direction $\boldsymbol{d}$;
    $\alpha = 1$;
    while true do
        Compute $f(\theta + \alpha \boldsymbol{d})$;
        if (35) is satisfied then
            break;
        end
        $\alpha \leftarrow \alpha/2$;
    end
    Update $\lambda$ based on (37);
    $\theta \leftarrow \theta + \alpha \boldsymbol{d}$;
end
```

Algorithm 3: Subsampled Hessian Newton method for CNNs

$$f(\theta + \alpha \boldsymbol{d}) \leq f(\theta) + \eta \alpha \nabla f(\theta)^T \boldsymbol{d}$$

Equation 4.1

$$(G + \lambda \mathcal{I})\boldsymbol{d} = -\nabla f(\theta),$$

Equation 4.2

$$\lambda_{\text{next}} = \begin{cases} \lambda \times \text{drop} & \rho > \rho_{\text{upper}}, \\ \lambda & \rho_{\text{lower}} \leq \rho \leq \rho_{\text{upper}}, \\ \lambda \times \text{boost} & \text{otherwise}, \end{cases}$$

Equation 4.3

4.2 Network Implementation

As a means to complete this dissertation , a Convolutional Neural Network (CNN) with the Subsampled Hessian Newton(SHN) optimization method was used, which was implemented in Python by Wang et al. [26] and can be found here [https://github.com/ cjlin1/simpleNN]. The paper also discusses multiple optimisation techniques that were utilized in order to reduce memory consumption and to improve efficiency. The Python implementation used the Tensorflow machine learning framework.

The initial implementation that the paper [26] used for the experiments was implemented in Matlab. Consequently, the input datasets used a matlab format (.mat), which was transferred to the Python version. The input files are made up of a matrix and a vector. The matrix has dimensions N × M where N is the number of amino acids and M is the number of features for each amino acid. In the paper this matrix was called the 'Z' variable whereas in the implementation of this project is referred as the 'x' variable. Moreover the vector, also referred as the 'y' variable, is of size N × 1 and includes the label of every amino acid in the 'x' matrix.

Embeddings are used as input to the CNN, and were derived from here [https://github.com/sacdallago/bio_embeddings]. There are several different bio embeddings but at the current time ProtBERT and SeqVec embeddings are considered the optimal models to use. The necessary code that extracts and converts the embeddings into .mat files can be found at [https://gitlab.com/schatz06/pssp/-/tree/master/data_preprocessing] and as well as in Appendix B.

The implementation of the above network was modified by Leontiou [5] (Appendix A) in order to be adapted to the PSSP problem, and also to be executed in a Jupiter notebook. The matlab files constructed from the CB513 dataset were uploaded to a public Gitlab repository so they can be easily accessible.

Chapter 5

5 Experiments and Results

5.1 Fine tuning of Hyper parameters

In order to find the optimal hyper parameters for the network, various experiments were performed and each time only one hyper parameter was altered while the rest remained the same. For each combination of hyper parameters three different models were trained and the best Q3 accuracy was saved in an excel file. For hyper parameter tuning, all the experiments were performed on fold 8 of CB513 datasets, because models trained with this fold had the worst overall test performance(on CASP13), compared to the rest folds. The motivation behind this was to maximize the performance of the fold with the lowest test accuracy with the hope that this would increase the overall Q3 accuracy and SOV score for the rest of the folds.

Max Iterations were set to 50, a sufficient number of iterations so the model can converge. Since ProtBert embedder extracts 1024 values for each amino acid, the input dimensions of the CNN were 32 × 32 × 1.

The first step was to find the number of samples used in the subsampled Gauss-Newton matrix (GNsize). Seven different values were tested while the rest were set at a constant value.

GNsize	C	CNN Layers	bsize	Max Iterations	Dimensions	Q3 Test Accuracy
50	0.01	4	1024	50	32*32*1	77.94%
100	0.01	4	1024	50	32*32*1	78.34%
200	0.01	4	1024	50	32*32*1	78.45%
512	0.01	4	1024	50	32*32*1	78.17%
1024	0.01	4	1024	50	32*32*1	78.24%
2048	0.01	4	1024	50	32*32*1	78.10%
4096	0.01	4	1024	50	32*32*1	78.26%

Table 5.1: Q3 accuracy results for GNsize for fold 8 of CB513 dataset

For the following experiments GNsize was selected to be equal to 200. After that, the regularization parameter (C value) had to be determined so the same process was repeated but this time the C values were examined.

GNsize	C	CNN Layers	bsize	Max Iterations	Dimensions	Q3 Test Accuracy
200	0.01	4	1024	50	32*32*1	78.40%
200	0.05	4	1024	50	32*32*1	78.46%
200	0.1	4	1024	50	32*32*1	78.30%
200	0.5	4	1024	50	32*32*1	78.24%
200	1	4	1024	50	32*32*1	78.27%

Table 5.2: Tuning the C hyper parameter for fold 8 of CB513 dataset

Table 5.2 shows the Q3 test accuracy results of the models based on the C values. Thus the C value was set to 0.05 because achieved a Q3 accuracy of 78.46%

The next step was to determine the batch size (bsize), but due to memory constrains was set to 1024.

After the optimization hyper parameters were extracted, it was time to alter the CNN parameters to observe whether or not we can achieve higher Q3 accuracies. After various executions we concluded to the parameters shown in Table 5.3, for our CNN. Pooling layers were not utilized in the CNN since they removed sufficient information resulting to poorer results, almost 3% less accuracy on CASP13 dataset (Q3 accuracy with pooling layers ≈ 75.5%, Q3 accuracy without pooling layers ≈ 78.5%) .

Type	Kernel Size	Number of Filters	Activation Function
CNN Layer 0 Convolutional Hidden Layer	5 x 5	64	ReLU
CNN Layer 1 Convolutional Hidden Layer	5 x 5	64	ReLU
CNN Layer 2 Convolutional Hidden Layer	5 x 5	128	ReLU
CNN Layer 3 Fully Connected MLP	-	-	-

Table 5.3: Hyper Parameters for CNN for all experiments

5.2 Experiments with CB513 dataset

To check whether the results of a model are good just for a specific testing dataset or whether the trained network is a good prediction model, additional techniques must be utilized. One such technique is cross-validation. To be precise, a 10-fold cross-validation was used for the CB513 dataset to validate the model's ability to generalize. Table 5.4 shows the hyper parameters for all the trained models, which are used for the cross-validation of CB513.

GNsize	C	CNN Layers	bsize	Max Iterations	Dimensions
200	0.05	4	1024	50	32*32*1

Table 5.4: Hyper parameters for trained models

Table 5.5 shows the cross-validation results for the CB513 dataset. The table shows the overall Q3 accuracy (Appendix F) and SOV score (Appendix E) for the best model trained for each fold. Moreover the Q3 accuracy and SOV scores for each secondary structure are shown. Finally the last row contains the average results for all the folds.

As we can observe from the Table 5.5 the average Q3 accuracy is 79.96% with a standard deviation of 1.23%, and a mean SOV score of 71.76 with a standard deviation of 2.19. Also if we compare the accuracy for every class we can see that models were unable to predict the ‘E’ structure with high success, having an average of just 69% (QE) whereas accuracies for ‘H’ and ‘C’ classes where around 83%.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	78.69	83.98	63.35	81.07	69.27	70.25	63.29	65.89
Fold1	78.01	80	65.88	83.59	69.47	68.09	68.3	69.59
Fold2	78.01	80	65.88	83.59	69.47	68.09	68.3	69.59
Fold3	79.86	84.8	69.5	80.73	69.99	68.4	69.9	68.21
Fold4	81.76	85.46	70.98	83.54	73.45	72.71	64.89	70.94
Fold5	80.89	82.01	74.75	83.95	75.17	72.32	74.14	72.93
Fold6	80.88	86.06	67.79	83.07	73.21	75.93	66.57	71.8
Fold7	80.21	83.83	69.62	83.35	70.44	72.27	69.69	69.05
Fold8	79.77	81.58	70.5	83.19	71.55	69.28	71.67	70.19
Fold9	81.55	84.97	72.07	83.79	75.6	74.85	76.73	72.85
Average	79.96	83.27	69.03	82.99	71.76	71.22	69.35	70.10

Table 5.5: Q3 and SOV results for 10-fold cross validation for the CB513 dataset

In order to utilize the ensembles method, 5 models were trained on each fold. Table 5.6 shows the ensembles method results for the cross validation datasets (Appendix C).

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	78.72	83.63	63.89	81.21	69.4	71.95	63.02	64.99
Fold1	78.1	80.11	66.37	83.42	69.1	66.16	67.99	69.68
Fold2	80.2	81.67	73.54	82.6	72.25	71.18	69.14	69.33
Fold3	80.33	85.57	68.99	81.46	70.63	70.56	69.66	67.85
Fold4	81.92	85.59	71.41	83.6	73.74	72.54	65.7	71.41
Fold5	81.03	82.08	75.07	84.02	75.58	71.69	75.63	73.52
Fold6	81.02	86.06	67.14	83.72	74.43	78.8	68.04	72.23
Fold7	80.38	83.53	69.32	84.12	70.7	72.15	69.92	69.88
Fold8	79.76	81.61	70.05	83.37	71.71	69.61	71.05	70.27
Fold9	81.69	84.56	71.77	84.66	76.85	73.72	77.23	74.82
Average	80.32	83.44	69.76	83.22	72.44	71.84	69.74	70.4

Table 5.6: Q3 and SOV results for ensembles cross validation for the CB513 dataset

If we compare Table 5.5 and Table 5.6, we can see a slight increase in the average Q3 accuracy and SOV score. How the Q3 accuracy and SOV score will fluctuate depends on the variance of the predictions between the trained models, but since all models were trained with the same hyperparameters the variance between these models was small.

5.3 Filtering techniques on CB513 dataset

As mentioned in section 3.8 the use of post-training filtering techniques can help to achieve higher Q3 accuracy and SOV score. Different combinations from the available techniques were used in order to see the impact they have in the final predictions, since the order that the filtering techniques are applied can produce different results. The window size used to extract (Appendix G) the following results was 19 (Appendix H).

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	78.92	81.87	61.35	84.4	73.61	71.4	62.21	69.71
Fold1	77.87	77.17	62.55	87.48	71.29	66.24	69.17	69.39
Fold2	80.03	79.04	71.2	85.38	75.84	73.64	69.64	71.9
Fold3	80.39	83.5	66.55	85.04	74.25	72.69	68.66	71
Fold4	82.27	83.49	68.24	87.69	76.78	73.48	64.04	74.06
Fold5	81.38	80.12	73.78	87.03	78.31	72.41	75.72	74.66
Fold6	81.09	84.08	64.23	87.02	77.38	79.03	67.69	74.47
Fold7	80.61	81.65	67.1	87.22	73.89	74.56	69.98	70.64
Fold8	80.25	79.2	67.77	87.51	77.89	73.52	72.78	76.79
Fold9	81.23	81.96	68.71	87.49	78.53	75.93	77.8	74.56
Average	80.40	81.21	67.15	86.63	75.78	73.29	69.77	72.72

Table 5.7: Q3 accuracy and SOV score for ensembles and external rules for CB513 dataset

As mentioned in section 3.8.1 external rules are usually used in order to improve the SOV score (Appendix D). We can confirm this statement if we compare the Table 5.6 and Table 5.7. The average Q3 accuracy remained almost the same whereas the average SOV score increased from 72.44 to 75.78 when the external rules were applied to the predictions obtained from the ensembles method.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	83.96	86.4	70.05	88.19	77.33	76.03	70.11	74.42
Fold1	83.89	83.51	74.75	89.59	77.29	72.62	77.24	73.24
Fold2	84.92	83.82	77.14	89.81	79.76	76.16	75.73	76.68
Fold3	85.55	88.18	77.15	87.63	78.69	77.29	74.49	74.1
Fold4	86.89	86.72	77.32	91.44	82.7	77.73	77.03	81.62
Fold5	86.44	84.89	81.59	90.59	82.46	76.09	82.47	79.58
Fold6	86.04	88.03	75.74	89.52	81.22	82.34	76.21	77.41
Fold7	86.39	86.78	76.41	91.55	78.53	78.52	75.3	76.01
Fold8	85.53	83.82	76.8	91.36	82.48	77.3	80.36	80.47
Fold9	86.23	86.19	76.67	91.54	82.57	78.87	83.02	79.13
Average	85.58	85.83	76.36	90.12	80.30	77.3	77.2	77.27

Table 5.8: Q3 accuracy and SOV score for ensembles, external rules and SVM filtering for CB513 dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	88.48	90.87	80.01	90.27	83.51	83.38	77.85	81.51
Fold1	91.3	91.88	87.84	92.87	86.97	85.85	85.7	83.96
Fold2	89.68	89.86	85.21	91.89	85.41	83.3	83.07	82.23
Fold3	91.22	94.46	86.9	90.41	85.86	84.92	86.41	82.66
Fold4	91.6	92.65	85.77	93.39	88.54	84.72	85.45	88.09
Fold5	90.25	91.32	86.47	91.85	87.62	84.26	88.98	85.88
Fold6	91.64	93.29	84.82	93.67	89.72	90.7	87.32	88.04
Fold7	90.17	91.54	81.74	93.77	85.09	85.51	83	83.38
Fold8	91.11	91.11	85.97	93.78	86.63	81.25	85.38	86.63
Fold9	91.64	93.17	86.51	93.14	88.23	86.1	88.85	86.35
Average	90.71	92.02	85.12	92.50	86.76	85	85.20	84.87

Table 5.9: Q3 accuracy and SOV score for ensembles and SVM filtering for CB513 dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	87.74	90.15	76.87	90.6	82.25	82.35	70.71	79.57
Fold1	90.27	90.76	84.71	93.16	84.92	84.42	82.2	80.36
Fold2	88.79	89.15	81.88	92.16	84.2	81.63	80.95	80.07
Fold3	90.49	93.98	83.84	90.73	84.29	84.14	80.12	79.69
Fold4	90.7	92.15	81.54	93.67	86.29	83.82	77.69	83.35
Fold5	89.6	90.54	84.13	92.35	86.98	84.57	85.37	83.99
Fold6	90.97	92.73	81.73	94.12	86.9	89.76	76.98	84
Fold7	89.4	91.01	78.47	94.18	83.3	85.29	79.53	80.41
Fold8	90.38	90.47	82.7	94.29	87.19	82.98	81.58	85.55
Fold9	90.88	92.55	83.39	93.55	87.11	85.22	86.79	83.7
Average	89.92	91.35	81.93	92.88	85.34	84.42	80.2	82.07

Table 5.10: Q3 accuracy and SOV score for ensembles, SVM filtering and external rules for CB513 dataset

Tables 5.8, 5.9 and 5.10 show the Q3 accuracies and SOV scores of all the possible combinations that we can apply the SVM filtering method to the results we got from the ensembles method. The impact of SVM filtering was significant for both Q3accuracy and SOV score. According to Tables 5.8 and 5.10 applying the SVM filtering before the external rules the results are substantially better, with almost 4.5% increase in Q3 accuracy and about 5 points growth in the overall SOV score. Lastly Table 5.9 shows that if we apply only the SVM filtering and not the external rules actually improves the Q3 accuracy and SOV score about 0.8% and 1 point respectively.

Tables 5.11, 5.12, 5.13 show the Q3 accuracies and SOV scores for all the possible combinations that we can apply the decision tree filtering method to the predictions we got from the ensembles method (Table 5.6). The best results obtained when just the decision tree method was applied to the ensembles shown in Table 5.12. The Q3 accuracy went form 80.32% to just over 93% and the SOV score increased from 72.44 to 88.59. Decision Trees had a better overall performance than SVM method too. The highest Q3 accuracy concerning the SVM filtering technique was 90.71% almost 2% lower than the highest decision tree accuracy.

	Q3	QH	QE	QC	SOV	SOV	SOV	SOV
Fold0	85.51	86.84	72.04	90.52	78.55	79.85	71.03	76.24
Fold1	86.18	84.96	76.52	92.87	78.92	76.78	76.65	76.83
Fold2	87.5	85.01	78.54	94.04	80.43	77.79	76.63	78.62
Fold3	87.61	88.37	78.8	91.77	79.75	79.35	76.34	76.01
Fold4	88.66	87.22	78.19	94.73	83	79.35	77.49	83.31
Fold5	88.99	85.6	84.29	94.35	84.8	79.5	83.32	83.59
Fold6	87.8	87.5	76.57	93.67	82.36	81.39	78.2	80.45
Fold7	88.36	87.16	78.02	94.87	80.17	80.37	75.75	79.07
Fold8	87.41	83.65	79.18	94.58	82.78	78.26	79.25	82.44
Fold9	88.69	86.85	80.52	94.78	85.48	84.88	84.31	84.45
Average	87.67	86.32	78.27	93.62	81.62	79.75	77.9	80.10

Table 5.11: Q3 accuracy and SOV score for ensembles, external rules and Decision Tree filtering for CB513 dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	90.9	91.03	84.12	93.9	85.25	86.34	81.45	84.75
Fold1	93.9	93.22	91.47	95.88	89.37	90.86	86.72	87.34
Fold2	92.31	90.74	88.44	95.49	86.49	87.06	83.92	84.81
Fold3	93.04	94.3	89.74	93.64	86.27	85.3	87.7	84.18
Fold4	93.75	93.38	89.68	95.93	89.41	87.01	87.48	91.27
Fold5	93.09	92.53	88.97	96.06	90.38	87.27	90.11	90.45
Fold6	93.74	93.4	89.15	96.32	91.28	89.68	91.65	91.89
Fold7	92.41	92.44	85.46	96.19	87.5	86.97	85.85	87.8
Fold8	93.27	91.98	88.7	96.63	88.99	84.81	87.24	89.58
Fold9	94.01	93.42	90.26	96.57	90.94	88.78	89.57	91.53
Average	93.04	92.64	88.6	95.66	88.59	87.41	87.17	88.36

Table 5.12: Q3 accuracy and SOV score for ensembles and Decision Tree filtering for CB513 dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	89.92	90.43	79.35	94.32	83.2	86.32	73.14	81.25
Fold1	92.86	92.14	88.19	96.2	87.85	91.43	84.07	83.51
Fold2	91.16	89.67	84.06	95.98	86.07	86.63	81.65	82.62
Fold3	91.87	93.12	85.71	94.05	84.61	84.88	80.13	79.8
Fold4	92.46	92.48	84.28	96.22	87.7	88.26	79.64	85.61
Fold5	91.78	90.79	85.7	96.29	88.24	86.14	86.01	85.5
Fold6	92.62	92.38	85.17	96.55	88.49	90.2	79.68	86.37
Fold7	91.27	91.61	81.29	96.46	85.61	86.98	83.62	83.43
Fold8	92.19	91.01	84.53	97.07	88.05	83.58	83.41	86.88
Fold9	92.72	92.26	86.01	96.82	89.23	86.58	88.11	86.78
Average	91.89	91.59	84.43	96	86.91	87.1	81.95	84.18

Table 5.13: Q3 accuracy and SOV score for ensembles, Decision Tree filtering and external rules for CB513 dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	85.52	87.91	73.85	88.77	78.4	79.97	71.38	75.69
Fold1	86.27	86.88	78.38	90.43	78.63	78.65	78.02	74.41
Fold2	87.74	86.3	81.46	92.1	80.45	77.36	79.72	78.32
Fold3	87.72	89.65	81.24	89.43	78.71	79.6	75.02	73.97
Fold4	88.73	88.82	80.92	92.24	83.37	81.21	78.08	81.68
Fold5	89.09	86.81	85.86	92.73	84.13	80.3	83.81	81.98
Fold6	87.83	89.45	78.35	91.21	81.94	83.2	77.65	79.03
Fold7	88.38	88.17	79.63	93.31	79.78	81.6	75.86	77.62
Fold8	87.76	85.09	82.45	92.57	84.15	79.52	81.86	83.07
Fold9	88.78	88	82.6	92.85	84.15	85.57	84.07	82.55
Average	87.78	87.71	80.47	91.56	81.37	80.7	78.55	78.83

Table 5.14: Q3 accuracy and SOV score for ensembles, external rules and Random Forest filtering for CB513 dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	91.38	93.26	86.84	91.8	86.48	87.78	82.58	84.43
Fold1	94.41	95.11	92.79	94.81	89.8	92.89	88.15	86.5
Fold2	93.33	92.89	91.61	94.56	87.68	89.83	87.56	83.87
Fold3	93.82	95.99	91.78	92.82	87.99	88.73	89.93	84.83
Fold4	94.19	94.74	91.8	94.82	91.93	90.75	90.24	91.76
Fold5	93.57	93.81	91.62	94.62	90.35	88.56	91.5	89.68
Fold6	93.94	94.74	90.69	94.89	90.57	92.1	90.76	90.47
Fold7	93.04	93.34	88.68	95.2	89.16	89.19	88.39	88.34
Fold8	94.17	93.35	91.87	95.99	90.64	86.74	89.26	90.16
Fold9	94.62	94.71	92.73	95.59	91.67	93.36	90.62	91.03
Average	93.65	94.19	91.04	94.51	89.63	89.99	88.9	88.11

Table 5.15: Q3 accuracy and SOV score for ensembles and Random Forest filtering for CB513 dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	90.44	92.41	82.13	92.52	84.98	88.09	74.6	81.66
Fold1	93.65	94.31	89.61	95.51	89.02	93.18	84.81	84.07
Fold2	92.31	92.11	87.19	95.14	87.33	89.55	83.3	82.67
Fold3	92.99	95.64	87.98	93.17	86.26	87.75	81.9	81.28
Fold4	93.12	94.18	86.58	95.22	89.52	91.89	80.98	86.33
Fold5	92.43	92.53	87.88	95.2	88.92	87.28	86.63	85.89
Fold6	92.89	93.68	86.48	95.43	88.56	90.59	80.4	86.05
Fold7	92.02	92.51	84.66	95.67	86.77	88.31	85.04	84.17
Fold8	93.21	92.64	87.8	96.45	89.35	87.85	84.67	87.37
Fold9	93.57	93.92	88.53	96.03	90.55	88.85	89.51	87.91
Average	92.66	93.40	86.88	95.03	88.13	89.33	83.18	84.74

Table 5.16: Q3 accuracy and SOV score for ensembles, Random Forest filtering and external rules for CB513 dataset

Tables 5.14, 5.15, and 5.16 shows the results of the final machine learning algorithm used as a filtering technique, Random Forest. The Q3 accuracy and SOV score when using Random Forest only outperformed every other combination used on the ensembles. The

Q3 accuracy reached 93.65%, almost a 13% increased and the SOV score rose by almost 17.

5.4 Final results for CB513

The average Q3 accuracy and SOV score for each filtering technique are presented in table 5.17, which makes it easier to compare the different filtering methods. According to table 5.17, the best results for CB513, in terms of overall Q3 accuracy and overall SOV score, came from the ensembles model with the random forest filtering. This model managed to reach 93.65% Q3 (per residue) accuracy and 89.63 SOV score.

Method	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Cross Validation	79.96	83.27	69.03	82.99	71.76	71.22	69.35	70.10
Ensembles	80.32	83.44	69.76	83.22	72.44	71.84	69.74	70.40
Ensembles + External Rules Results	80.40	81.21	67.15	86.63	75.78	73.29	69.77	72.72
Ensembles + External Rules + SVM Results	85.58	85.83	76.36	90.12	80.30	77.30	77.20	77.27
Ensembles + SVM Results	90.71	92.02	85.12	92.50	86.76	85.00	85.20	84.87
Ensembles + SVM + External Rules Results	89.92	91.35	81.93	92.88	85.34	84.42	80.19	82.07
Ensembles + External Rules + Decision Tree Results	87.67	86.32	78.27	93.62	81.62	79.75	77.90	80.10
Ensembles + Decision Tree Results	93.04	92.64	88.60	95.66	88.59	87.41	87.17	88.36
Ensembles + Decision Tree + External Rules Results	91.89	91.59	84.43	96.00	86.91	87.10	81.95	84.18
Ensembles + External Rules + Random Forest Results	87.78	87.71	80.47	91.56	81.37	80.70	78.55	78.83
Ensembles + Random Forest Results	93.65	94.19	91.04	94.51	89.63	89.99	88.90	88.11
Ensembles + Random Forest + External Rules Results	92.66	93.39	86.88	95.03	88.13	89.33	83.18	84.74

Table 5.17: Average Q3 accuracy and SOV score for all methods for CB513 dataset

5.5 CASP13 results with the system trained on CB513

CASP13 was used as an independent testing set on all the models trained with CB513 dataset. Table 5.18 shows the best results obtained for each fold.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	78.66	84.40	68.22	78.97	70.37	71.24	67.12	68.84
Fold1	78.55	84.56	69.10	78.10	70.54	72.44	67.68	68.53
Fold2	78.52	84.42	69.84	77.77	71.33	72.19	69.54	67.89
Fold3	78.69	84.37	69.47	78.43	70.73	71.50	67.43	68.68
Fold4	78.74	84.23	69.01	78.90	71.30	71.33	67.54	69.17
Fold5	78.70	84.83	69.24	78.17	69.96	71.40	67.42	68.06
Fold6	78.60	84.26	68.91	78.59	71.15	71.72	67.25	69.79
Fold7	78.55	84.61	68.59	78.31	71.26	72.79	67.05	69.51
Fold8	78.49	84.29	69.56	77.96	70.66	71.31	67.83	68.62
Fold9	78.50	84.10	68.64	78.62	70.36	73.72	65.89	69.09
Average	78.60	84.41	69.06	78.38	70.77	71.96	67.48	68.82

Table 5.18: Best Q3 and SOV for CASP13 trained on CB513 dataset

Ensembles and filtering techniques were then applied to the results. The window variable for the datasets used for the machine learning techniques was set to 19.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	78.67	84.67	68.04	78.85	71.17	72.38	66.80	68.83
Fold1	78.43	84.29	69.15	78.03	70.82	71.97	67.91	68.36
Fold2	78.49	84.40	69.52	77.89	71.56	72.16	69.21	68.89
Fold3	78.38	84.23	68.41	78.33	70.52	70.66	67.51	68.54
Fold4	78.64	84.34	68.87	78.64	70.95	71.78	66.93	69.12
Fold5	78.76	84.53	68.68	78.85	70.22	71.01	67.77	68.69
Fold6	78.57	84.40	68.96	78.36	70.59	71.11	66.86	69.33
Fold7	78.46	84.42	68.18	78.48	71.16	72.41	67.63	69.42
Fold8	78.52	84.40	69.28	78.07	71.01	71.71	67.71	70.00
Fold9	78.46	84.02	68.59	78.62	69.46	70.82	65.91	68.56
Average	78.54	84.37	68.77	78.41	70.75	71.60	67.42	68.97

Table 5.19: Q3 and SOV for ensembles method of CASP13 dataset trained on CB513

Method	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Cross Validation	78.60	84.41	69.06	78.38	70.77	71.96	67.48	68.82
Ensembles	78.54	84.37	68.77	78.41	70.75	71.60	67.42	68.97
Ensembles + External Rules Results	78.60	83.03	67.16	80.57	73.04	73.11	68.67	70.33
Ensembles + External Rules + SVM Results	82.97	85.95	73.82	85.02	77.17	75.99	75.40	75.52
Ensembles + SVM Results	86.95	90.47	81.16	86.82	82.38	82.53	81.60	81.96
Ensembles + SVM + External Rules Results	86.35	89.61	78.64	87.43	82.83	83.22	79.18	81.28
Ensembles + External Rules + Decision Tree Results	85.67	85.93	75.67	90.57	79.14	76.69	76.83	79.89
Ensembles + Decision Tree Results	90.08	90.75	83.72	92.74	85.08	84.38	84.12	86.97
Ensembles + Decision Tree + External Rules Results	88.99	89.49	79.50	93.40	86.05	86.85	82.62	84.64
Ensembles + External Rules + Random Forest Results	85.72	87.51	78.86	87.66	79.52	79.08	79.02	78.01
Ensembles + Random Forest Results	90.54	92.75	87.35	90.23	86.49	87.00	87.30	85.80
Ensembles + Random Forest + External Rules Results	89.77	91.76	83.45	91.25	87.47	89.88	84.19	85.17

Table 5.20: Average Q3 and SOV for CASP13 trained on CB513 for all filtering techniques

According to Table 5.20 the highest Q3 accuracy is when the ensembles results are filtered with random forest, at 90.54%, whereas the best SOV score is when the ensembles results are filtered with random forest and external rules (87.47).

5.6 Experiments with PISCES dataset

A larger dataset like PISCES can help the model to learn more effectively the patterns of the data, resulting in higher performance. Table 5.21 shows the 5-fold cross validation results, which are about 1.5% on average better than the CB513 cross validation results shown in Table 5.6. Post processing methods were used for the cross validation results of PISCES.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	81.37	86.24	74.03	80.46	75.80	77.20	76.07	70.41
Fold1	81.56	86.06	74.68	81.06	75.75	75.86	77.56	70.70
Fold2	81.49	86.36	74.25	80.88	76.30	76.94	76.89	70.96
Fold3	81.27	85.78	74.21	80.88	75.40	76.22	76.67	70.08
Fold4	81.56	85.98	74.02	81.37	76.28	75.33	77.19	70.16
Average	81.45	86.08	74.24	80.93	75.91	76.31	76.88	70.46

Table 5.21: Q3 and SOV results for 5-fold cross validation for the PSICES dataset

As in CB513 dataset, 5 models were trained on each fold for the PSICES dataset in order to use the ensembles method. Table 5.22 presents the results for Q3 accuracy and SOV score of the ensembles method, for each fold of PISCES.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	81.46	86.21	73.88	80.82	76.23	77.45	76.24	70.88
Fold1	81.70	86.26	74.80	81.15	76.22	76.20	77.70	71.03
Fold2	81.66	86.44	74.46	81.12	76.79	77.43	77.22	71.42
Fold3	81.43	85.79	74.28	81.22	75.65	76.22	76.79	70.43
Fold4	81.70	85.99	74.21	81.60	76.68	75.68	77.38	70.46
Average	81.59	86.14	74.33	81.18	76.31	76.60	77.07	70.84

Table 5.22:Q3 accuracy and SOV score for ensembles method for PISCES dataset

The comparison between table 5.21 and 5.22 reveals that there is a similar issue with the CB513 dataset. There is not enough variance between the trained models, which results in only a small improvement in overall Q3 accuracy (0.14%) and SOV score (0.4).

5.7 Filtering Results for PISCES

The results from the ensembles method were then filtered using external rules, decision trees, and random forest, with a window variable of 19.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	81.55	84.74	72.26	83.49	78.89	78.87	76.69	73.40
Fold1	81.80	84.80	73.19	83.82	78.56	77.20	77.59	73.15
Fold2	81.78	85.06	72.79	83.79	79.29	78.13	77.71	73.96
Fold3	81.44	84.24	72.55	83.84	78.17	77.39	77.00	72.59
Fold4	81.79	84.54	72.62	84.26	79.08	76.93	77.72	72.64
Average	81.67	84.68	72.68	83.84	78.80	77.70	77.34	73.15

Table 5.23: Q3 accuracy and SOV score for ensembles with external rules filtering for PISCES dataset

As expected when external rules are applied to the ensembles results, the Q3 accuracy remains almost the same, but the SOV score increased by over 2 units.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	82.95	85.43	74.50	85.16	79.86	79.59	78.06	74.06
Fold1	83.20	85.49	75.51	85.39	79.54	77.76	78.91	73.86
Fold2	83.17	85.61	75.10	85.48	80.01	78.37	78.86	74.48
Fold3	82.83	85.00	74.88	85.30	78.97	77.67	78.40	73.16
Fold4	83.18	85.30	75.08	85.70	79.98	77.52	79.11	73.32
Average	83.07	85.37	75.01	85.41	79.67	78.18	78.67	73.78

Table 5.24: Q3 accuracy and SOV score for ensembles, external rules and Decision Tree filtering for PISCES dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	85.56	88.06	79.27	86.52	81.78	81.63	80.95	77.54
Fold1	85.85	88.18	80.54	86.63	82.27	80.69	82.82	77.83
Fold2	85.78	88.27	79.88	86.75	82.66	81.55	82.54	78.30
Fold3	85.57	88.01	79.93	86.42	81.45	80.41	82.25	77.15
Fold4	85.80	87.91	80.06	86.96	82.52	80.17	82.71	77.07
Average	85.71	88.09	79.94	86.66	82.14	80.89	82.25	77.58

Table 5.25: Q3 accuracy and SOV score for ensembles and Decision Tree filtering for PISCES dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	85.06	87.11	77.54	87.20	82.37	81.92	80.33	77.06
Fold1	85.37	87.15	78.99	87.33	82.58	80.84	81.97	77.35
Fold2	85.30	87.33	78.19	87.45	83.09	81.88	81.77	77.78
Fold3	85.07	87.03	78.23	87.09	82.22	80.86	81.36	76.81
Fold4	85.32	87.00	78.31	87.65	82.99	80.91	81.71	76.53
Average	85.22	87.12	78.25	87.34	82.65	81.28	81.43	77.11

Table 5.26: Q3 accuracy and SOV score for ensembles, Decision Tree and external rules filtering for PISCES dataset

According to Table 5.24, 5.25 and 5.26 applying the decision tree filtering method produces significantly better results. The best overall Q3 accuracy was achieved when using just the decision tree method on the ensembles results (85.71% Table 5.25 – 81.59% Table 5.22) a 4% increase, whereas the best overall SOV score was when external rules were applied after the decision tree method (82.65 Table 5.26 – 76.31 Table 5.22).

Tables 5.27, 5.28, 5.29 show the results for the different combinations of applying the random forest method on the ensembles results.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	83.12	85.98	75.50	84.47	80.00	80.17	78.60	73.69
Fold1	83.36	86.09	76.49	84.64	79.75	78.39	79.49	73.46
Fold2	83.30	86.16	75.97	84.76	80.23	79.04	79.58	74.14
Fold3	82.99	85.58	75.83	84.58	79.15	78.24	79.06	72.79
Fold4	83.34	85.83	76.15	84.94	80.30	78.25	79.62	73.11
Average	83.22	85.93	75.99	84.68	79.89	78.82	79.27	73.44

Table 5.27: Q3 accuracy and SOV score for ensembles, external rules and Random Forest filtering for PISCES dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	87.04	89.81	82.51	86.71	83.96	84.31	83.77	79.23
Fold1	87.21	89.84	83.41	86.79	84.38	83.37	85.31	79.51
Fold2	87.16	89.88	82.78	87.01	84.55	84.04	84.84	79.76
Fold3	87.02	89.72	83.01	86.64	83.80	83.47	85.09	78.66
Fold4	87.21	89.58	83.16	87.10	84.69	82.97	85.19	78.53
Average	87.13	89.77	82.97	86.85	84.28	83.63	84.84	79.14

Table 5.28: Q3 accuracy and SOV score for ensembles and Random Forest filtering for PISCES dataset

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	86.62	89.16	80.81	87.28	84.06	84.56	82.35	78.56
Fold1	86.78	89.13	81.77	87.36	84.25	83.29	83.78	78.72
Fold2	86.71	89.21	81.03	87.55	84.60	84.07	83.45	79.12
Fold3	86.59	89.06	81.30	87.20	83.83	83.40	83.32	78.16
Fold4	86.77	88.93	81.40	87.65	84.52	82.65	83.68	77.82
Average	86.69	89.10	81.26	87.41	84.25	83.59	83.32	78.48

Table 5.29: Q3 accuracy and SOV score for ensembles, Random Forest and external rules filtering for PISCES dataset

As in CB513 dataset, random forest gave the best overall Q3 accuracy (87.13%) and SOV score (84.28). Again, if we are going to use external rules it is better applying them after the machine learning technique (Table 5.27 and Table 5.29).

5.8 Final Results for PISCES

The average for all filtering methods are shown in table 5.30. The best results according to this table, came from the ensembles model after applying the random forest filtering. This method managed to reach 87.13% overall Q3 (per residue) accuracy and 84.28 overall SOV score.

Method	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Cross Validation	81.45	86.08	74.24	80.93	75.91	76.31	76.88	70.46
Ensembles Results	81.59	86.14	74.33	81.18	76.31	76.60	77.07	70.84
Ensembles + External Rules Results	81.67	84.68	72.68	83.84	78.80	77.70	77.34	73.15
Ensembles + External Rules + Decision Tree Results	83.07	85.37	75.01	85.41	79.67	78.18	78.67	73.78
Ensembles + Decision Tree Results	85.71	88.09	79.94	86.66	82.14	80.89	82.25	77.58
Ensembles + Decision Tree + External Rules Results	85.22	87.12	78.25	87.34	82.65	81.28	81.43	77.11
Ensembles + External Rules + Random Forest Results	83.22	85.93	75.99	84.68	79.89	78.82	79.27	73.44
Ensembles + Random Forest Results	87.13	89.77	82.97	86.85	84.28	83.63	84.84	79.14
Ensembles + Random Forest + External Rules Results	86.69	89.10	81.26	87.41	84.25	83.59	83.32	78.48

Table 5.30: Average Q3 accuracy and SOV score for all method for the PISCES dataset

5.9 CASP13 results with the system trained on PISCES

Independent testing was performed as well for all the models trained with the PISCES datasets. Table 5.31 shows the best results obtained for each fold. In comparison with Table 5.18. The Q3 accuracy slightly increased (~0.2%) and the SOV by almost 1.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	78.78	86.17	71.45	76.04	71.35	73.75	68.99	67.91
Fold1	78.67	86.12	71.59	75.76	70.62	72.77	69.66	66.99
Fold2	78.90	86.01	71.32	76.54	72.17	74.23	68.16	68.88
Fold3	78.89	85.82	71.22	76.73	72.06	74.49	69.12	68.77
Fold4	78.92	86.01	71.32	76.59	71.92	74.19	70.54	67.96
Average	78.83	86.03	71.38	76.33	71.62	73.89	69.29	68.10

Table 5.31: : Best Q3 and SOV for CASP13 trained on PISCES dataset

Table 5.32 shows the ensembles method results and Table 5.33 shows the average results for every filtering technique that was applied to the ensembles. The best overall Q3 accuracy was achieved when using the random forest filtering technique with 90.69%. Training datasets seem to not make a big difference in the accuracy of the predictions on the independent test set (CASP13), since the best overall Q3 accuracy achieved on CASP13 when the system is trained with the CB513 dataset is 90.54% (Table 5.20), just 0.15% lower than the best Q3 accuracy for CASP13 when the system is trained on PISCES dataset.

	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Fold0	78.73	85.88	71.45	76.19	72.58	76.10	69.47	67.86
Fold1	78.76	85.96	71.59	76.12	71.91	73.86	69.54	68.21
Fold2	78.95	86.09	71.45	76.52	72.11	74.47	69.40	68.50
Fold3	78.87	85.71	71.55	76.61	71.92	74.83	69.59	68.00
Fold4	78.89	85.88	71.41	76.59	72.40	74.94	69.66	68.94
Average	78.84	85.90	71.49	76.41	72.18	74.84	69.53	68.30

Table 5.32: Q3 and SOV for ensembles method of CASP13 dataset trained on PISCES dataset

The same applies for the SOV score as well, the best overall SOV score of CASP13 with the system trained with CB513 dataset is 87.47 almost 0.5 lower than the respective SOV score when the system is trained with PISCES dataset.

Method	Q3	QH	QE	QC	SOV	SOVH	SOVE	SOVC
Cross Validation	78.83	86.03	71.38	76.33	71.62	73.89	69.29	68.10
Ensembles Results	78.84	85.90	71.49	76.41	72.18	74.84	69.53	68.30
Ensembles + External Rules Results	79.12	84.87	70.17	78.64	74.27	75.05	70.47	70.82
Ensembles + External Rules + Decision Tree Results	86.29	86.82	76.90	90.63	80.69	80.95	76.29	79.29
Ensembles + Decision Tree Results	90.30	91.10	85.30	92.16	86.07	85.26	85.04	86.40
Ensembles + Decision Tree + External Rules Results	89.24	89.85	81.37	92.72	87.08	88.29	83.79	84.94
Ensembles + External Rules + Random Forest Results	86.32	88.26	80.39	87.64	80.76	81.53	78.39	77.53
Ensembles + Random Forest Results	90.69	92.57	88.03	90.39	87.54	87.61	87.74	85.59
Ensembles + Random Forest + External Rules Results	89.88	91.62	84.40	91.15	87.94	89.52	85.32	84.95

Table 5.33: Average Q3 and SOV for CASP13 trained on PISCES for all filtering techniques

The following confusion matrices show how many times each of the 3 classes were predicted right or wrong, when the system was trained with the Fold 4 of the PISCES dataset. Fold 4 was chosen because it had the highest Q3 accuracy on a single CNN model.

	Predicted H	Predicted E	Predicted C
True H	3197	64	456
True E	82	1544	539
True C	505	487	3245

Table 5.34: Confusion matrix for CASP13 when system was trained on Fold 4 of PISCES dataset on a single CNN model (Q3 accuracy 78.92%)

	Predicted H	Predicted E	Predicted C
True H	3436	25	256
True E	35	1899	231
True C	252	168	3817

Table 5.35: Confusion matrix for CASP13 when the system was trained on Fold 4 of PISCES dataset with Ensembles & Random Forest with window size 19 (Q3 accuracy 90.44%)

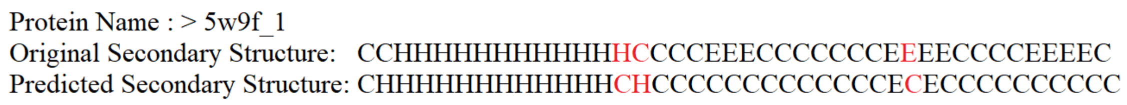
	Predicted H	Predicted E	Predicted C
True H	3645	5	67
True E	10	2134	21
True C	76	7	4154

Table 5.36: Confusion matrix for CASP13 when the system was trained on Fold 4 of PISCES dataset with Ensembles & Random Forest with window size 31 (Q3 accuracy 98.16%)

If we observe table 5.34 we can see that the major misclassifications are when the true structure was H but a C was predicted, the correct structure was E but C was predicted and when in general was a misclassification of the C class. After applying the filtering techniques, as shown in Table 3.35 and Table 3.36, we can clearly see that those misclassifications drop dramatically.

5.10 Observations

After analyzing the predicted secondary structures obtained without any filtering techniques applied to them, we observed that there are misclassified structures that can be easily corrected with the use of post-processing methods, thus increasing the overall Q3 accuracy. An example is illustrated in Figure 5.1. If those 3 misclassified structures with red color are corrected then the Q3 accuracy is going to increase by almost 7% (3/41).



Protein Name : > 5w9f_1
Original Secondary Structure: CCHHHHHHHHHHHHCCCCEEECCCCCCCEEEECCCCEEEEC
Predicted Secondary Structure: CHHHHHHHHHHHHCHCCCCCCCCCCCCCECECCCCCCCCCC

Figure 5.1: Easily corrected misclassified structures

Based on this simple observation we can say that the predicted secondary structure obtained with the NLP embeddings as inputs are in a form which they can be easily corrected with post-processing techniques and maybe this is the reason that the results after the filtering techniques are applied are very high.

5.11 Additional Experiments

5.11.1 Window variable

Additional experiments were performed in order to see the impact the window variable, that is used to create the datasets for the machine learning filtering techniques, has on the predictions. Since the window variable has to be an odd number, we performed experiments from window size 3 to window size 31. A size of 31 can be considered a limit since the smaller protein in the CASP13 dataset is 31 and we want to keep the test dataset intact in order to be able to make comparisons. Furthermore, the best filtering technique is random forest so the following experiments are based only in this technique.

	Q3	QH	QE	QC
WINDOW 3	79.15	85.40	71.09	77.79
WINDOW 5	79.50	85.78	72.08	77.78
WINDOW 7	80.25	86.70	73.17	78.20
WINDOW 9	81.32	87.68	74.81	79.07
WINDOW 11	82.70	88.38	76.85	80.69
WINDOW 13	84.40	89.45	79.42	82.50
WINDOW 15	86.39	90.86	81.02	85.22
WINDOW 17	88.51	91.34	84.33	88.17
WINDOW 19	90.69	92.57	88.03	90.39
WINDOW 21	92.62	94.21	90.71	92.20
WINDOW 23	94.37	95.52	93.78	93.67
WINDOW 25	95.78	96.47	95.83	95.14
WINDOW 27	96.79	97.23	97.04	96.28
WINDOW 29	97.58	97.63	97.97	97.33
WINDOW 31	98.12	98.01	98.46	98.05

Table 5.37: Average Q3, QH, QE and QC accuracies, based on window size, for Random Forest on CASP13 trained with PSICES dataset

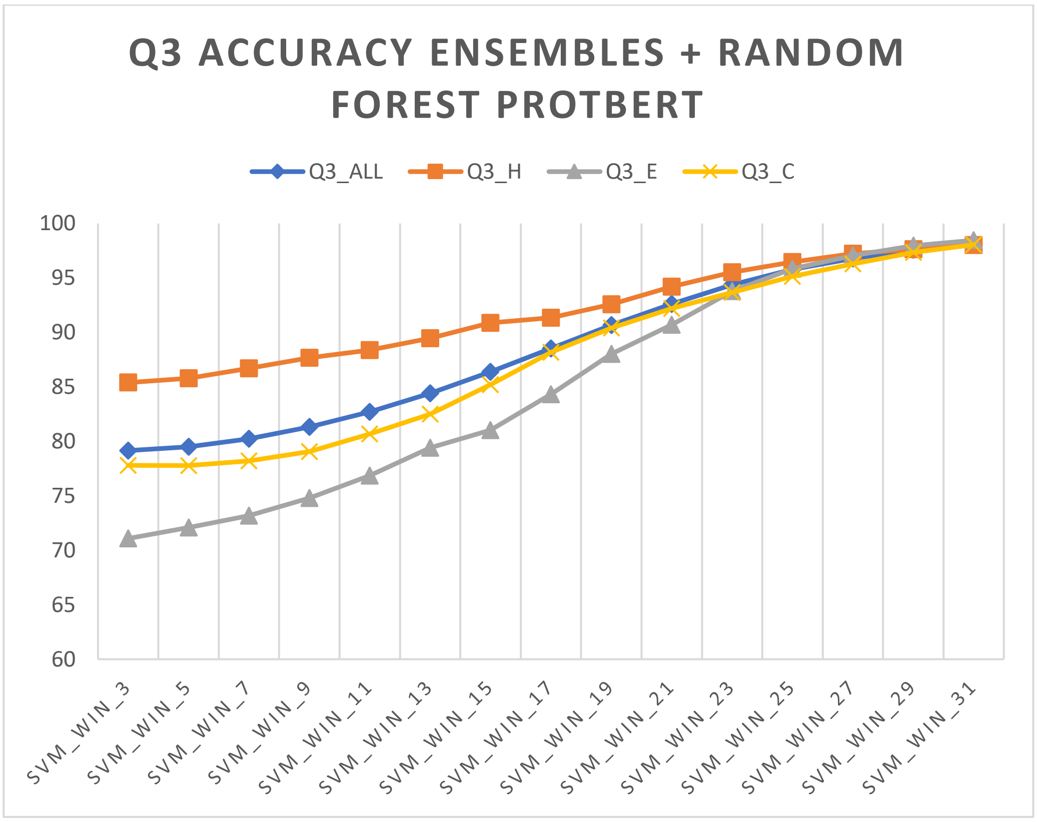


Figure 5.2: Graph illustration of the average Q3, QH, QE and QC accuracies, based on window size, for Random Forest on CASP13 trained with PSICES dataset

According to Figure 5.2 we can actually see that as the window size increases the prediction accuracy increases as well. Moreover we can see that the curves that illustrate the accuracy of each class (QH, QE, QC) and the overall Q3 accuracy converge at around 98%. Finally, we can observe from Table 5.37 that initially the QE accuracy is the worst with just over 71% but at the end QE is the best predicted class with an accuracy of 98.46%. As mentioned in section 3.8, a larger window may capture long range connections/interactions between classes resulting in better Q3 accuracy, and this is proved by the graph of Figure 5.2. The same experiment was used to analyze the SOV score as well. The results are shown in Table 5.38 and Figure 5.3. When using window size from 3 to 17, the SOVE score was the worst but from window size 19 to 31 the SOVE score was the best.

	SOV	SOVH	SOVE	SOVC
WINDOW 3	74.58	75.80	71.65	70.44
WINDOW 5	74.90	76.46	71.82	71.07
WINDOW 7	75.18	76.48	73.43	71.62
WINDOW 9	76.02	77.43	74.97	72.96
WINDOW 11	78.34	79.70	76.16	75.54
WINDOW 13	80.31	82.07	78.87	77.14
WINDOW 15	81.99	83.26	80.56	79.28
WINDOW 17	83.89	84.91	83.29	82.31
WINDOW 19	87.54	87.61	87.74	85.59
WINDOW 21	89.20	88.71	90.72	88.25
WINDOW 23	91.83	90.49	92.89	91.95
WINDOW 25	92.97	92.51	94.93	92.02
WINDOW 27	94.69	93.33	95.53	94.55
WINDOW 29	94.69	92.63	96.77	95.56
WINDOW 31	96.98	95.61	97.35	97.23

Table 5.38: Average SOV, SOVH, SOVE and SOVC scores, based on window size, for Random Forest on CASP13 trained with PSICES dataset

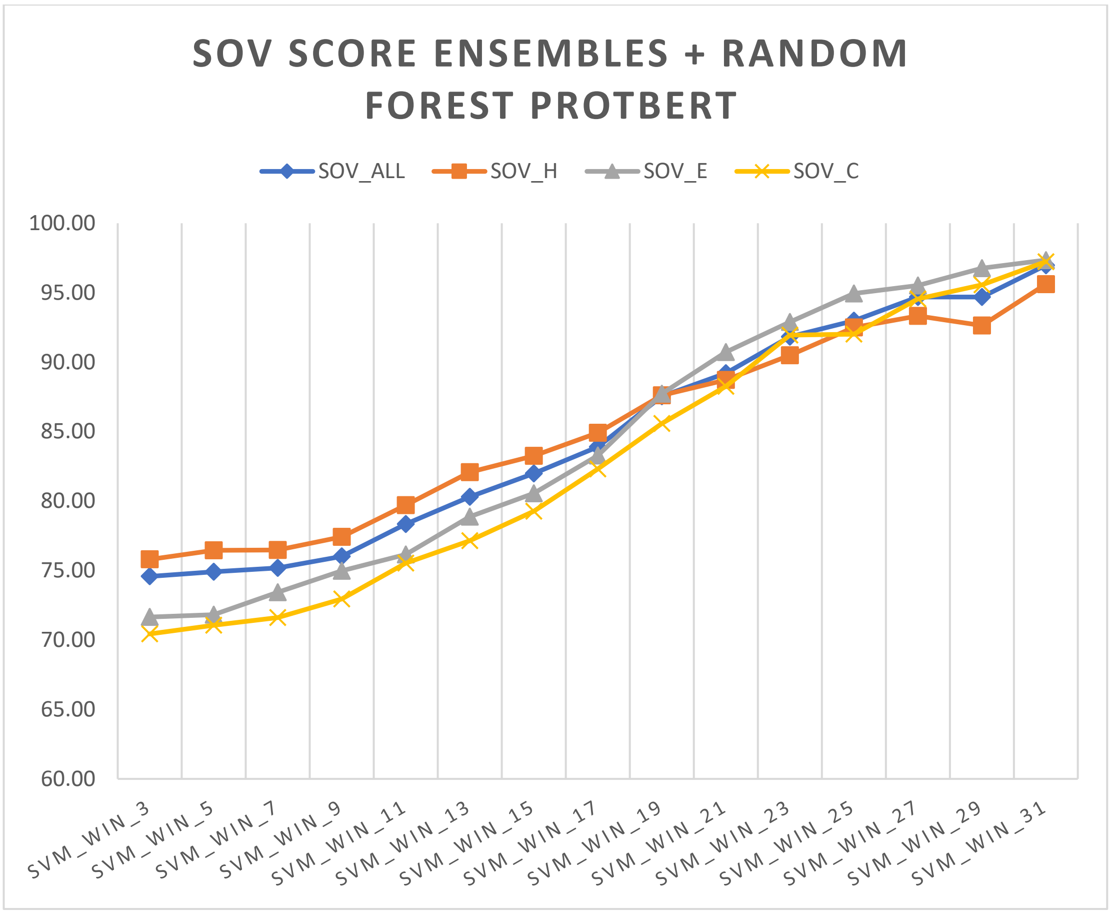


Figure 5.3:Graph illustration of the average SOV, SOVH, SOVE and SOVC scores, based on window size, for Random Forest on CASP13 trained with PSICES dataset

The same experiment was performed on the results when the same CNN with the HFO was trained on a different data representation, implemented by Dionysiou [35] [36]and used by Leontiou [5]. It is a two dimensional (2D) input representation method, where Multiple Sequence Alignment (MSA) profile vectors are placed one under another.

	Q3	QH	QE	QC
WINDOW 3	77.61	84.21	67.03	77.22
WINDOW 5	78.20	84.23	66.71	78.78
WINDOW 7	79.23	85.09	68.29	79.68
WINDOW 9	80.51	86.13	70.46	80.72
WINDOW 11	82.18	87.32	72.29	82.74
WINDOW 13	84.05	88.49	76.21	84.15
WINDOW 15	86.04	89.81	79.50	86.07
WINDOW 17	88.08	91.24	82.27	88.29
WINDOW 19	90.23	92.63	86.67	89.94
WINDOW 21	92.00	93.84	89.91	91.44
WINDOW 23	93.66	95.00	92.44	93.10
WINDOW 25	95.00	96.31	93.92	94.41
WINDOW 27	95.93	97.11	94.96	95.40
WINDOW 29	96.62	97.58	95.95	96.12
WINDOW 31	96.97	97.74	96.43	96.57

Table 5.39: Average Q3, QH, QE and QC accuracies, based on window size, for Random Forest on CASP13 trained with PSICES dataset using Dionysiou data representation

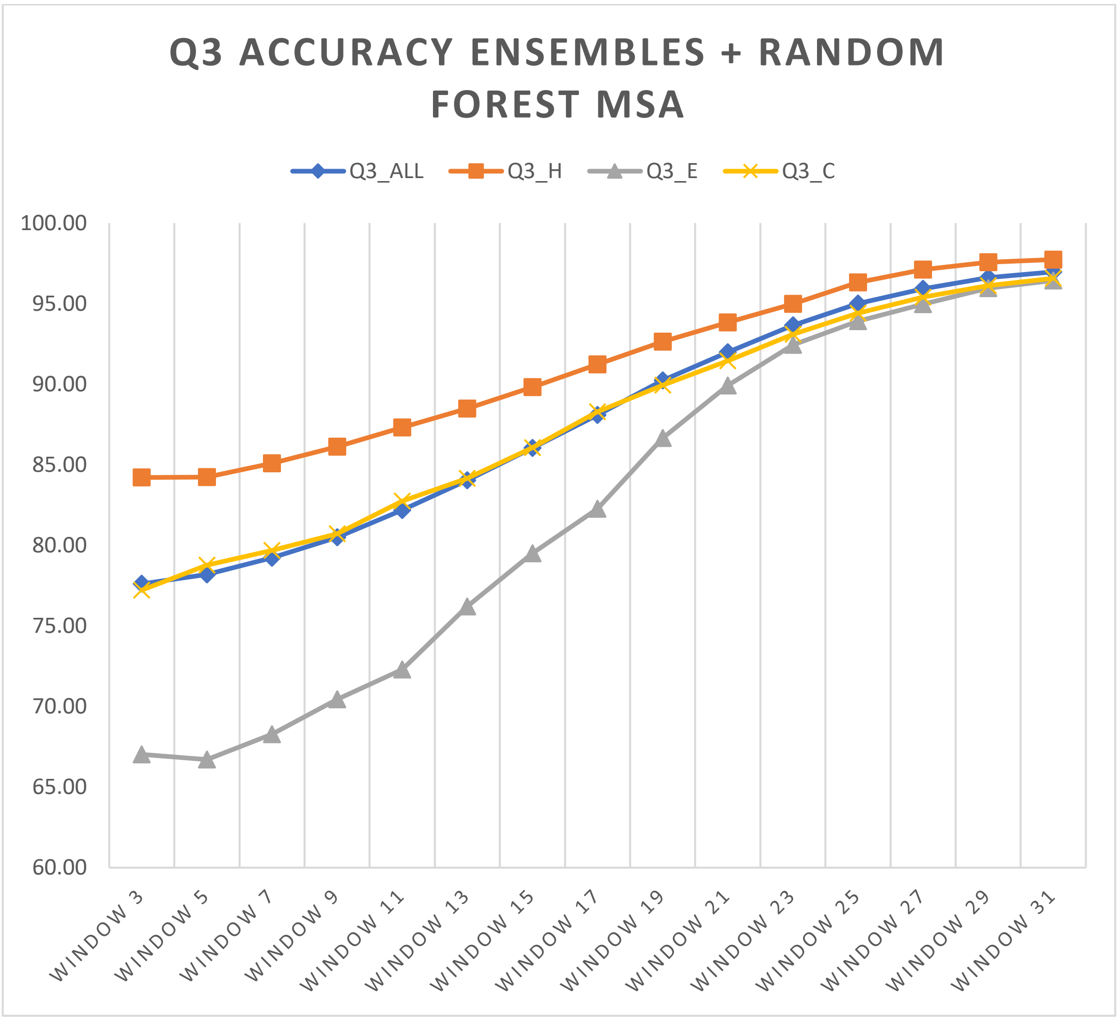


Figure 5.4: Graph illustration of the average Q3, QH, QE and QC accuracies, based on window size, for Random Forest on CASP13 trained with PSICES dataset using Dionysiou data representation

	SOV	SOVH	SOVE	SOVC
WINDOW 3	71.75	76.57	70.93	68.02
WINDOW 5	70.79	76.51	69.98	66.43
WINDOW 7	72.57	77.45	72.18	68.47
WINDOW 9	73.78	78.29	73.48	69.84
WINDOW 11	74.48	78.79	74.46	71.14
WINDOW 13	76.63	80.93	75.76	73.87
WINDOW 15	78.74	82.92	78.33	76.03
WINDOW 17	80.11	84.54	79.93	77.72
WINDOW 19	83.69	86.89	84.28	80.98
WINDOW 21	86.20	88.73	86.76	83.49
WINDOW 23	87.68	88.80	88.70	85.92
WINDOW 25	89.77	90.89	91.03	87.32
WINDOW 27	91.47	92.81	91.75	89.27
WINDOW 29	92.29	93.14	92.88	90.24
WINDOW 31	92.90	93.73	93.55	90.70

Table 5.40: Average SOV, SOVH, SOVE and SOVC scores, based on window size, for Random Forest on CASP13 trained with PSICES dataset using Dionysiou data representation

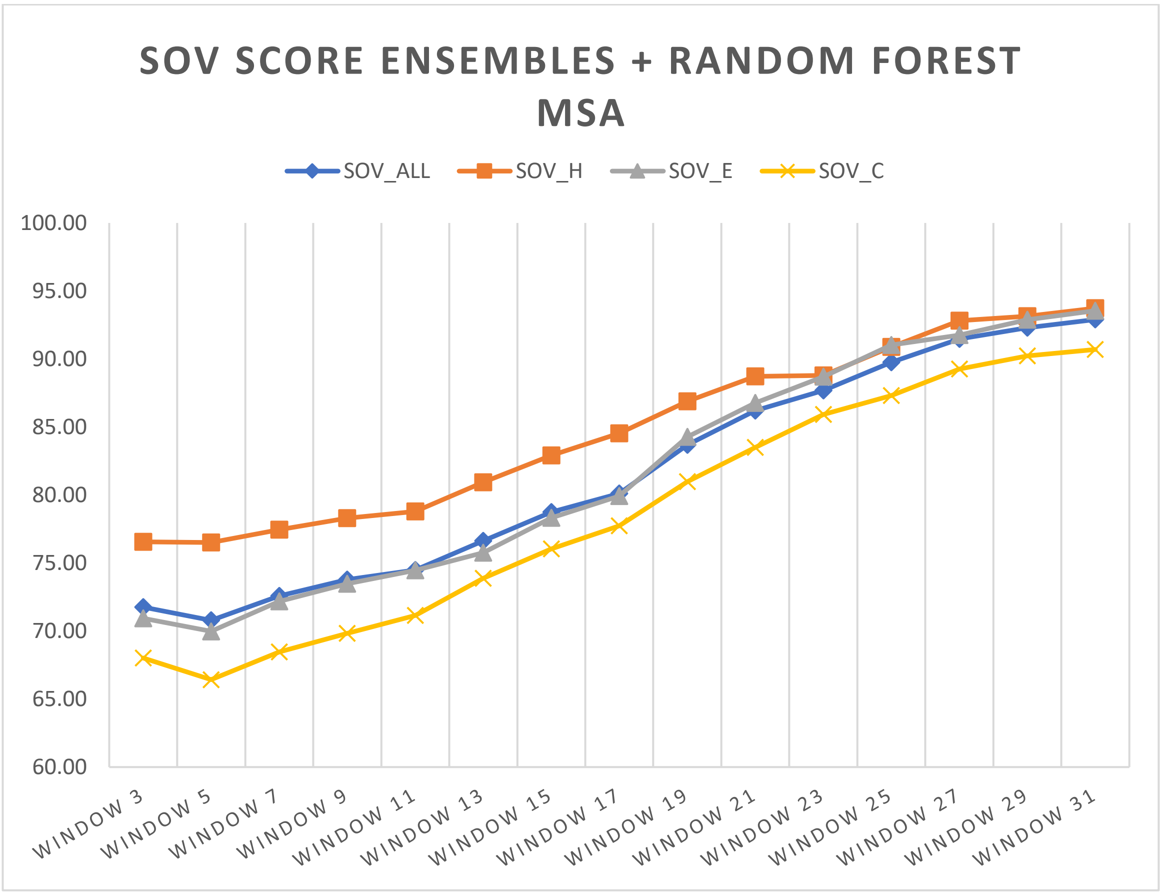


Figure 5.5: Graph illustration of the average SOV, SOVH, SOVE and SOVC scores, based on window size, for Random Forest on CASP13 trained with PSICES dataset using Dionysiou data representation

From Table 5.39 and Figure 5.4 we can observe how the Q3 accuracy varies as we change the window size. For the smallest window we have a Q3 accuracy of 77.61% and for the biggest window we have a Q3 accuracy of 96.97%. Table 5.40 and Figure 5.5 show how the SOV score increases as the window size increases as well. For the smallest window we have an SOV score of 71.75 and for a window size of 31 we have an SOV score of 92.90. Now if we compare the results between the two different data representations we can extract some conclusions. The Q3 accuracy when the smallest window size was used (window size 3), was for the data representation that used embeddings 79.15% and for the other data representation 77.61%. There is a difference of almost 1.5%, this difference fluctuates a bit as the window size increases but even when the biggest window size was used the difference in Q3 accuracy remains around 1.5%. The same applies for the SOV score as well, there is a difference of just under 3, but this difference remains as the window size increases. This leads that the filtering techniques make a proportional increase on the Q3 accuracy and SOV score no matter the data representation that was used to train the CNN. This may contradict the observation made on Section 5.10.

5.11.2 N-terminus vs C-terminus

Inspired from this paper [37], we wanted to determine whether the predictions near the N-terminus (first) amino acid are better than the predictions near the C-terminus (last) amino acid. In order to be able to determine the part of the protein that had the highest prediction accuracy, the protein is theoretically split into quarters as shown in Figure 5.4 (Appendix N). The predictions for CASP13 dataset trained on PISCES were used to extract the conclusions. To be more precise the predictions obtained from the CNN for CASP13 with the system trained on PISCES fold0, and the predictions for CASP13 with ensembles and random forest filtering technique for window sizes 19 and 31.

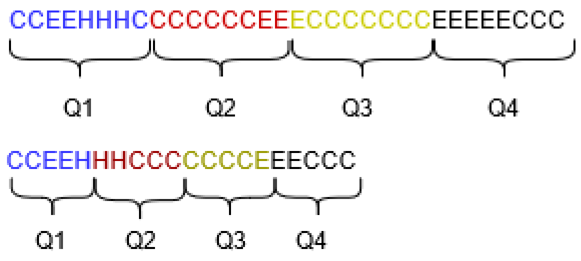


Figure 5.6: Example of how a protein is split in quarters

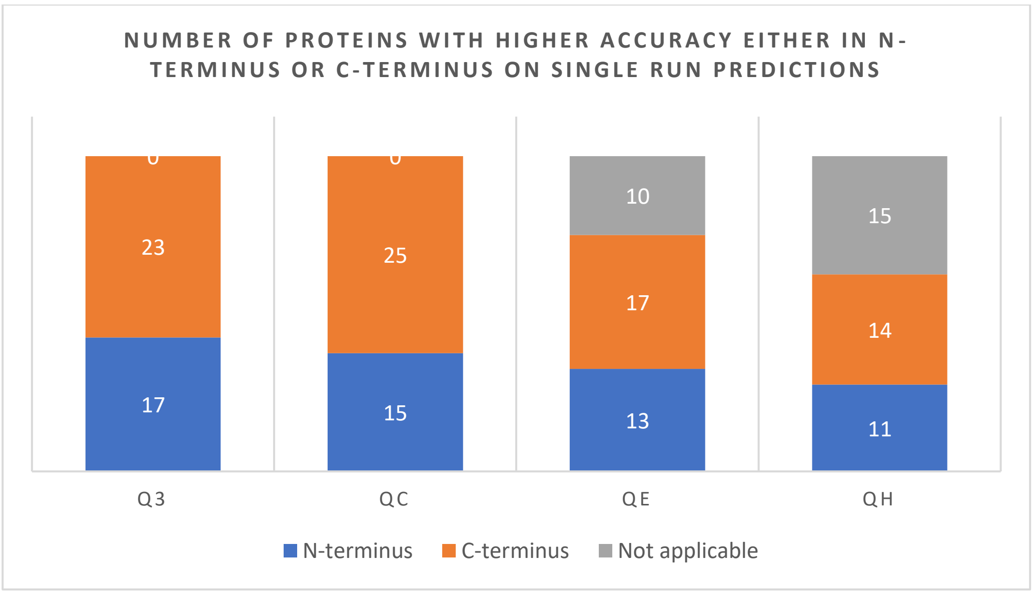


Figure 5.7: Number of proteins with higher accuracy either in n-terminus or c-terminus on single run predictions

Figure 5.5 shows how many proteins have higher overall accuracy as well as accuracy on each class near the N-terminus or C-terminus amino acid. It is obvious that the overall Q3 accuracy as well as QC, QE and QH accuracies were better near the C-terminus (last) amino acid. The non applicable parts represent the proteins that either do not have a specific secondary structure (E, H) near the N or C terminus structures, or they have the same prediction accuracy.

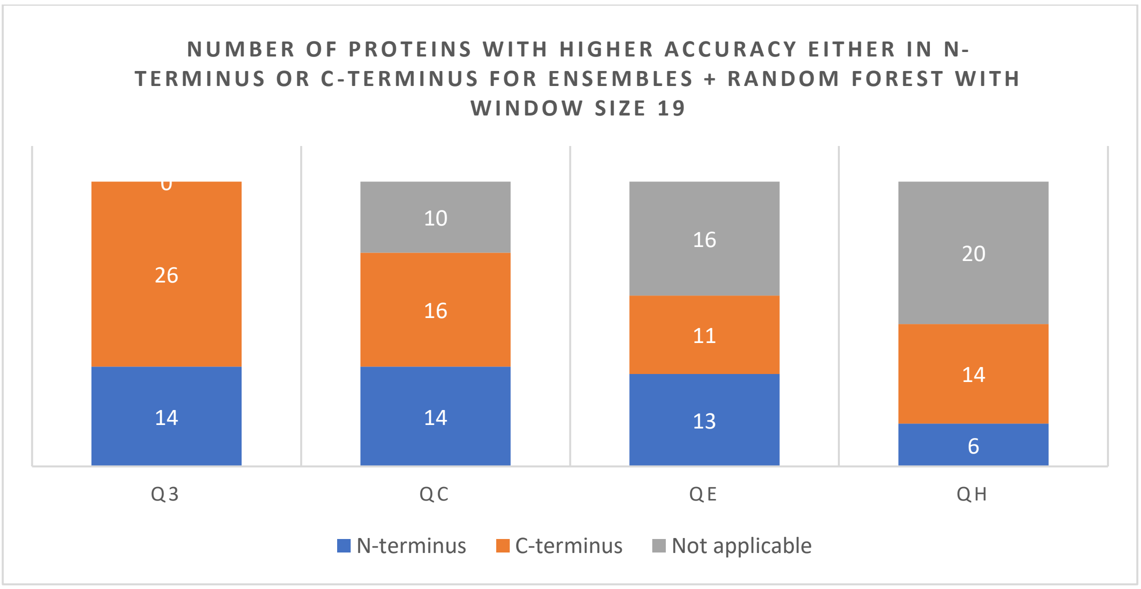


Figure 5.8: Number of proteins with higher accuracy either in n-terminus or c-terminus for ensembles and random forest with window size 19

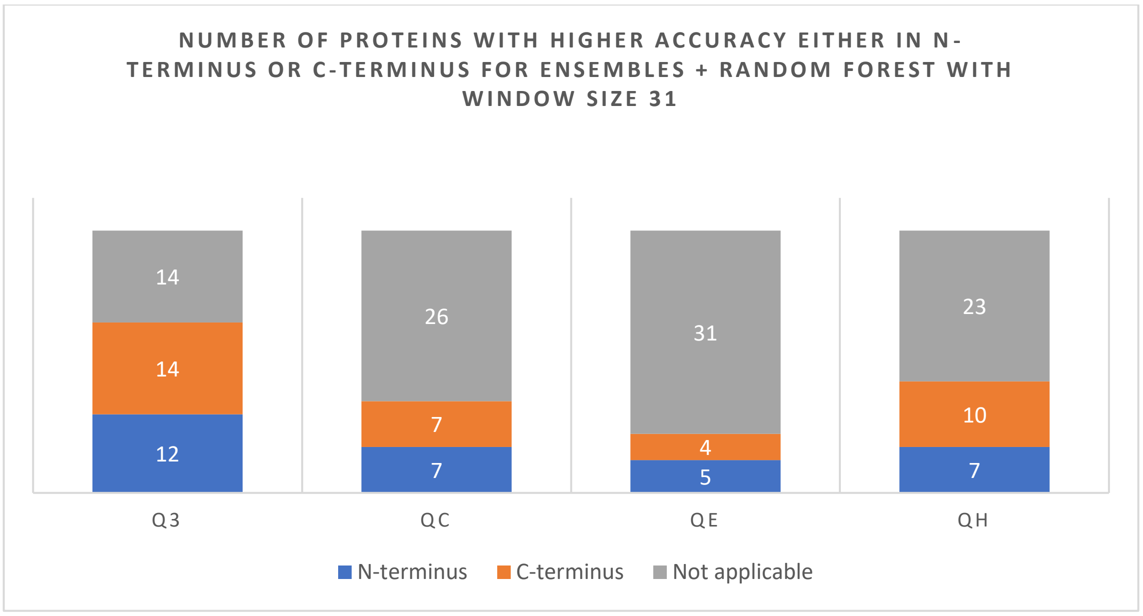


Figure 5.9: Number of proteins with higher accuracy either in n-terminus or c-terminus for ensembles and random forest with window size 31

In addition as we can see from Figure 5.6 and Figure 5.7, as the window size increases less proteins are available for comparison. In almost every occasion the accuracy near the C-terminus (last) amino acid were better.

Chapter 6

6 Conclusions and Future Work

6.1 Conclusions

The main purpose of this dissertation was to use embeddings that were extracted from language models as inputs to a Convolutional Neural Network (CNN) that uses a second order optimization algorithm, the Subsampled Hessian Newton (SHN), in order to train models that predict the Secondary Structure of Proteins (PSSP), given its primary structure.

The initial results, that were obtained from just a model trained on a single fold of the CB513 dataset, without further post processing looked very promising. Specifically when the CNN was trained with fold4 of the CB513 dataset, achieved a Q3 accuracy on the cross validation dataset of 81.76% (Table 5.5), and a test Q3 accuracy (CASP13) of 78.74% (Table 5.18). However the really impressive results came when the various filtering techniques were applied to the initial predictions. As shown in Table 5.17 of section 5.4 and Table 5.20 of section 5.5, the overall Q3 accuracy and SOV score skyrocketed. The overall cross validation Q3 accuracy increased from 79.96% to 93.65%, and the SOV score rose by almost 18 (71.76 to 89.63), with the use of ensembles and random forest filtering techniques and a window of 19. The overall test Q3 accuracy went from 78.60% to 90.54% and the SOV score from 70.77 to 87.47.

For the PISCES dataset, the best overall Q3 accuracy of a single CNN with SHN was 81.56% (Table 5.21), while the overall Q3 accuracy for the 5-fold cross validation was 81.45%. The highest Q3 accuracy for the 5-fold cross validation was 87.13% and was achieved with the ensembles model with the random forest filtering (Table 5.30). The best overall SOV score for a single model was 76.30 and for the 5-fold cross validation was 75.91. The highest overall SOV score achieved was 84.28, with the combination of the ensembles model and random forest filtering (Table 5.30).

The results for the CASP13 dataset when the system was trained on PISCES were slightly better than those when the system was trained on CB513. The best Q3 accuracy of a single CNN with SHN was 78.92% and the overall Q3 accuracy was 78.83% (Table 5.31). The highest Q3 accuracy was 90.69% and was obtained with the ensembles model with random forest filtering (Table 5.33). Finally the best SOV score for a single model was 72.17 and for the 5-fold cross validation was 71.62. The highest overall SOV score was

87.94 and was achieved with the ensembles model , random forest and external rules post processing methods.

Based on the results and the additional experiments we can make some conclusions. The use of embeddings as inputs to the CNN gave results equally good or even slightly better than multiple sequence alignment (MSA) [5]. Thus making the use of embeddings more convenient for multiple reasons. While existing solutions in Protein Bioinformatics usually have to search for evolutionary related proteins in exponentially growing databases, LMs offer a potential alternative to this increasingly time consuming database search as they extract features directly from single protein sequences. Moreover the performance of existing solutions decrease if not a sufficient number of related sequences can be found, e.g. the quality of predicted protein structures correlates strongly with the number of effective sequences found in today's databases. Additionally, some proteins are intrinsically hard to align (e.g. intrinsically disordered proteins or proteins which do not have any related sequences). On the other hand, embeddings require a considerable amount of storage space.

Finally, filtering techniques were the ones that boosted the final results. As we can observe the order that the filtering techniques were applied played a major role in the final outcome. However the most important aspect of the post processing methods was the size of the window variable (section 3.8). As the window variable was getting bigger the results were getting better as well. This concludes that a bigger window is able to capture long range connections between classes resulting in better Q3 accuracy.

6.2 Future Work

Based on section 5.10.1 where various window variables were used we can see that when using a window size of 31, the overall test Q3 accuracy reached 98.12% which is an almost perfect prediction. However since CASP13 is a really small dataset, just the correction of a few secondary structures leads in a big improvement in the accuracy. A different and bigger test set can possibly be used in order to observe if the overall Q3 accuracy remains at the same levels, for instance we can concatenate previous versions of the CASP datasets in order to create a new dataset that will have a couple of hundreds of protein sequences. In addition we can introduce a 3D representation as an input to the CNN, specifically instead of using a $32 \times 32 \times 1$ matrix we can make a $32 \times 32 \times 3$ tensor that consists of the representation of the amino acid we want to classify as well as the left and the right neighbor of this amino acid. This representation may help increase the overall Q3 accuracy and SOV score. Finally the experiment of Section 5.11.1 can be perform on the results of the implementation of Dionysiou [35] [36].

References

[1] F. Haurowitz, "Britannica: Protein Definition," December 2020. [Online]. Available: https://www.britannica.com/science/protein. [Accessed 20 December 2020].

[2] R. Bailey, "ThoughCo: Protein Structrure Types," May 2019. [Online]. Available: https://www.thoughtco.com/protein-structure-373563. [Accessed 20 December 2020].

[3] T. C. Terwilliger, D. Stuart and S. Yokoyama, "Lessons from structural genomics," *Annual review of biophysics,* vol. 38, pp. 371-383 doi:10.1146/annurev.biophys.050708.133740, 2009.

[4] C. N. Magnan and P. Baldi, "SSpro/ACCpro 5: almost perfect prediction of protein secondary structure and relative solvent accessibility using profiles, machine learning and structural similarity," *Bioinformatics,* vol. 30, no. 18, pp. 2592-2597, 2014.

[5] P. Leontiou, *Protein secondary structure prediction using convolutional neural networks and Hessian free optimisation,* BSc Thesis, Computer Science Department,University of Cyprus, 2020.

[6] Y. Yang, J. Gao, J. Wang, R. Heffernan, J. Hanson, K. Pailwal and Y. Zhou, "Sixty-five years of the long march in protein secondary structure prediction: the final stretch?," *Briefings in Bioinformatics,* vol. 19, no. 3, pp. 482-494, 2018.

[7] Science Learning Hub – Pokapū Akoranga Pūtaiao, "Role of proteins in the body," 2011. [Online]. Available: https://www.sciencelearn.org.nz/resources/209-role-of-proteins-in-the-body. [Accessed 21 April 2021].

[8] Learn.Genetics,Genetic Science Learning Center, "Types of Proteins," [Online]. Available: https://learn.genetics.utah.edu/content/basics/proteintypes/. [Accessed 21 April 2021].

[9] Compound Interest, "A Brief Guide to the Twenty Common Amino Acids," [Online]. Available: https://www.compoundchem.com/2014/09/16/aminoacids/. [Accessed 21 April 2021].

[10] "Convolutional neural networks for visual recognition," [Online]. Available: https://cs231n.github.io/neural-networks-1/. [Accessed 23 April 2021].

[11] K. Kawaguchi, "The McCulloch-Pitt Model of Neuron," [Online]. Available: http://osp.mans.edu.eg/rehan/ann/2_3_1%20The%20McCulloch-Pitts%20Model%20of%20Neuron.htm. [Accessed 24 April 2021].

[12] Wikipedia, "Convolutional Neural Networks," [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network. [Accessed 26 April 2021].

[13] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks," [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 26 April 2021].

[14] A. Saxena, "Convolutional Neural Networks: An Illustrated explanation," [Online]. Available: https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/. [Accessed 26 April 2021].

[15] Wikipedia, "Line Search," [Online]. Available: https://en.wikipedia.org/wiki/Line_search. [Accessed 27 April 2021].

[16] J. Hui, "Conjugate Gradient," [Online]. Available: https://jonathan-hui.medium.com/rl-conjugate-gradient-5a644459137a. [Accessed 27 April 2021].

[17] V. Preetham, "How to tame the valley Hessian-free hacks for optimizing large NeuralNetworks," [Online]. Available: https://medium.com/autonomous-agents/how-to-tame-the-valley-hessian-free-hacks-for-optimizing-large-neuralnetworks-5044c50f4b55. [Accessed 28 April 2021].

[18] J. Martens, "Deep learning via Hessian-free optimization," *Proceedings of the 27th International Conferenceon Machine Learning,* pp. 735-742, 2010.

[19] N. Schraudolph, "Fast Curvature Matrx-Vector Products for Second-Order Gradient Descent," *Neural Computation,* vol. 14, no. 7, pp. 1723-1738, 2002.

[20] C. Charalambous, *Protein secondary structure prediction using bidirectional recurrent neural networks and hessian free optimisation,* BSc Thesis, Department of Computer Science, University of Cyprus, 2018.

[21] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers,* vol. 22, pp. 2577-2637, 1983.

[22] M. Heinzinger, A. Elnaggar, Y. Wang, C. Dallago, D. Nechaev, F. Matthes and B. Rost, "Modeling aspects of the language of life through transfer-learning protein sequences," *BMC Bioinformatics,* vol. 20, no. Article 723, 2019.

[23] A. Elnaggar, M. Heinzinger, C. Dallago, G. Rihawi, Y. Wang, L. Jones, T. Gibbs, T. Feher, C. Angerer, D. Bhowmik and B. Rost, "bioRxiv:ProtTrans: Towards Cracking the Language of Life's Code Through Self-Supervised Deep Learning and High Performance Computing," bioRxiv preprint bioRxiv: 2020.07.12.199554, 2020.

[24] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee and L. Zettlemoyer, "Deep contextualized word representations," arXiv preprint arXiv:1802.05365v2, 2018.

[25] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv preprint arXiv: 1810.04805v2, 2019.

[26] C. Wang, K. Tan and C.-J. Lin, "Newton Methods for convolutional neural networks," *ACM Transactions on Intelligent Systems and Technology,* vol. 11, no. 2. Article 19, 2020.

[27] G. Ghati, "Compariston between BERT, GPT-2 and ELMo," [Online]. Available: https://medium.com/@gauravghati/comparison-between-bert-gpt-2-and-elmo-9ad140cd1cda. [Accessed 24 May 2021].

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention is all you need," arXiv preprint arXiv:1706.03762v5, 2017.

[29] J. A. Cuff and G. J. Barton, "Evaluation and improvement of multiple sequence methods for protein secondary structure prediction," *Proteins: Structure, Function, and Bioinformatics,* vol. 34, no. 4, pp. 508-519, 1999.

[30] G.Wang and R. L. Dunbrack Jr, "Pisces: a protein sequence culling server," *Bioinformatics,* vol. 19, no. 12, pp. 1589-1591, 2003.

[31] T. G. Dietterich, "Ensemble Methods in Machine Learning," *MCS '00: Proceedings of the First International Workshop on Multiple Classifier,* pp. 1-15, June 2000.

[32] P. Kountouris, M. Agathocleous, V. J. Promponas, G. Christodoulou, S. Hadjicostas, V. Vassiliades and C. Christodoulou, "A Comparative Study on Filtering protein Secondary Structure Prediction," *IEEE/ACM Transactions on Computational Biology and Bioinformatics,* vol. 9, no. 3, pp. 731-739, 2021.

[33] A. A. Salamov and V. V. Solovyev, "Prediction of Protein Secondary Structure by Combining Nearest-neighbor Algorithms and Multiple Sequence Alignments," *Journal of Molecular Biology* , vol. 247, pp. 11-15, 1995.

[34] T. Yiu, "Understanding Random Forest," Toward Data Science, 2019. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2. [Accessed 29 April 2021].

[35] A. Dionysiou, M. Agathocleous, C. Christodoulou and V. Promponas, "Convolutional Neural Networks in Combination with Support Vector Machines for Complex Sequential Data Classification," *Artificial Neural Networks and Machine Learning - ICANN,* Lecture Notes in Computer Science, ed. by V. Kurkova, Y. Manolopoulos, B. Hammer, L. Iliadis, I. Maglogiannis, Cham: Springer, 11140, Part II, 444-455, 2018.

[36] A. Dionysiou, *Πρόβλεψη δευτεροταγούς δομής πρωτεϊνών με χρήση συνελικτικών νευρωνικών δικτύων σε συνδιασμό με φίλτρα gabor και support vector machines,* BSc Thesis,Department of Computer Science ,University of Cyprus, 2018.

[37] R. Saunders and C. M. Deane, "Protein structure prediction begins well but ends badly," *Proteins,* vol. 78, no. 5, pp. 1282-1290, 2010.

[38] M. Johnson, "The Multilayer Perceptron," [Online]. Available: https://www.massey.ac.nz/~mjjohnso/notes/59302/l10.html. [Accessed 25 Aprl 2021].

A Appendix

Python Implementation

The code below includes the implementation of the Convolutional Neural Network with the Subsampled Hessian Newton method. This program was used to perform all the experiments of this dissertation. Note that commands that begin with '!' should be executed as bash commands. It is recommended to use the notebook version of the implementation which can be found at [https://gitlab.com/schatz06/pssp/-/blob/master/Neural%20Network/CNN_HFO_v2.ipynb]. This implementation was based on the Python implementation from [26], however, several modifications have been made from Leontiou [5] to improve the results of the CNN for the PSSP problem.

```
# -*- coding: utf-8 -*-
"""CNN_HFO_v2.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1ZGe47Gh9ZZ_JgAx8jK8X9Hqx4YGr
3fh5

##Mount drive##
"""

# from google.colab import drive
# drive.mount('/content/drive')

"""##General Variables to use to load data##"""

fold = 0 # which fold to take
embedding = "protbert"
#dataset="PISCES"
dataset="CB513"
USE_HFO=True

"""## Imports ##"""

# Commented out IPython magic to ensure Python compatibility.
# Reload all modules (except those excluded by %aimport) every time
before executing the Python code typed.
# %load_ext autoreload
# %autoreload 2
# matplotlib graphs will be included in your notebook, next to the
code.
# %matplotlib inline

# install hdf5storage package
!pip install hdf5storage
```

```
import pdb # python debugger
import numpy as np # import numpy
import tensorflow as tf # import tensorflow
tf.compat.v1.disable_eager_execution() # disable eager execution
import time # import time
import math # import math
import argparse # import argparse
import os # import os
import scipy.io as sio # import scipy.io
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior() # makes different behaviors betweem tf_v1 &
tf_v2 behave the same
from tensorflow.python.client import device_lib # package to find
available gpus
import pandas as pd # import padas
import hdf5storage # import hdf5storage

"""## Get data ##"""

valid_url="https://gitlab.com/schatz06/pssp/-
/raw/master/Datasets/CB513/matlab_files/{0}_{1}_testSet{2}.mat".forma
t(
    dataset.lower(),embedding,fold)
train_url="https://gitlab.com/schatz06/pssp/-
/raw/master/Datasets/CB513/matlab_files/{0}_{1}_trainSet{2}.mat".form
at(
    dataset.lower(),embedding,fold)
test_url="https://gitlab.com/schatz06/pssp/-
/raw/master/Datasets/CASP13/casp13_protbert.mat"
VALID_FILE="{0}_{1}_testSet{2}.mat".format(dataset.lower(),embedding,
str(fold))
TRAIN_FILE="{0}_{1}_trainSet{2}.mat".format(dataset.lower(),embedding
,str(fold))
TEST_FILE="casp13_protbert.mat"

![ -f "$VALID_FILE" ] && echo "$VALID_FILE exists" || wget
"$valid_url"
![ -f "$TRAIN_FILE" ] && echo "$TRAIN_FILE exists" || wget
"$train_url"
![ -f "$TEST_FILE" ] && echo "$TEST_FILE exists" || wget "$test_url"

# VALID_FILE =
"/content/drive/MyDrive/Datasets/{0}_{1}_testSet{2}.mat".format(datas
et.lower(),embedding,str(fold)) # validation set
# TRAIN_FILE =
"/content/drive/MyDrive/Datasets/{0}_{1}_trainSet{2}.mat".format(data
set.lower(),embedding,str(fold)) # train set
# TEST_FILE =
"/content/drive/MyDrive/Datasets/casp13_{0}.mat".format(embedding) #
test set CASP13
# print(VALID_FILE)
# print(TRAIN_FILE)
# print(TEST_FILE)

HEIGHT = 32
WIDTH = 32
DEPTH = 1
CATEGORIES = 3 # number of different classification categories
```

```
"""## VGG ##"""

"""
Codes are modifeid from PyTorch and Tensorflow Versions of VGG:
https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.
py, and
https://github.com/keras-team/keras-
applications/blob/master/keras_applications/vgg16.py
"""

# import tensorflow.compat.v1 as tf
# tf.disable_v2_behavior()
# import numpy as np
# import pdb
from tensorflow.keras.applications.vgg16 import VGG16 as vgg16 #
import vgg16 convolutional network
from tensorflow.keras.applications.vgg19 import VGG19 as vgg19 #
import vgg19 convolutional network

__all__ = ['VGG11', 'VGG13', 'VGG16','VGG19'] # array holds all vgg
CNN's names

def VGG(feature, num_cls): # define VGG

    with tf.variable_scope('fully_connected') as scope:
        dim =np.prod(feature.shape[1:]) # returns the product of the
given array
        x = tf.reshape(feature, [-1, dim]) # reshape tensor

        x = tf.keras.layers.Dense(units=4096, activation='relu',
name=scope.name)(x) # define layers
        x = tf.keras.layers.Dense(units=4096, activation='relu',
name=scope.name)(x)
        x = tf.keras.layers.Dense(units=num_cls, name=scope.name)(x)

    return x
# make the layers of CNN
def make_layers(x, cfg):
    for v in cfg:
        if v == 'M':
            x = tf.keras.layers.MaxPool2D(pool_size=[2, 2],
strides=2, padding='valid')(x)
        else:
            x = tf.keras.layers.Conv2D(
            filters=v,
            kernel_size=[3, 3],
            padding='SAME',
            activation=tf.nn.relu
            )(x)
    return x

cfg = {
    'A': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512,
'M'],
    'B': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M',
512, 512, 'M'],
    'D': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512,
512, 'M', 512, 512, 512, 'M'],
    'E': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512,
512, 512, 512, 'M',
          512, 512, 512, 512, 'M'],
}

def VGG11(x_images, num_cls):
    feature = make_layers(x_images, cfg['A'])
    return VGG(feature, num_cls)

def VGG13(x_images, num_cls):
    feature = make_layers(x_images, cfg['B'])
    return VGG(feature, num_cls)

def VGG16(x_images, num_cls):
    feature = make_layers(x_images, cfg['D'])
    return VGG(feature, num_cls)

def VGG19(x_images, num_cls):
    feature = make_layers(x_images, cfg['E'])
    return VGG(feature, num_cls)

"""## Net ##"""

# import tensorflow.compat.v1 as tf
# tf.disable_v2_behavior()
# import math
# import pdb
# from tensorflow.python.client import device_lib
# import numpy as np
# from net.vgg import *

def CNN_4layers(x_image, num_cls, reuse=False):
    _NUM_CLASSES = num_cls
    with tf.variable_scope('conv1', reuse=reuse) as scope:
        conv = tf.keras.layers.Conv2D(
            filters=64,
            kernel_size=[5,5],
            padding='SAME',
            activation=tf.nn.relu
        )(x_image)

    with tf.variable_scope('conv2', reuse=reuse) as scope:
        conv = tf.keras.layers.Conv2D(
            filters=64,
            kernel_size=[5,5],
            padding='SAME',
            activation=tf.nn.relu
        )(conv)

    with tf.variable_scope('conv3', reuse=reuse) as scope:
        conv = tf.keras.layers.Conv2D(
            filters=128,
            kernel_size=[5,5],
            padding='SAME',
            activation=tf.nn.relu
        )(conv)

    with tf.variable_scope('fully_connected', reuse=reuse) as scope:
        dim =np.prod(conv.shape[1:])
        flat = tf.reshape(conv, [-1, dim])
        outputs = tf.keras.layers.Dense(units=_NUM_CLASSES,
name=scope.name)(flat)
```

```
    return outputs

    # with tf.variable_scope('conv1', reuse=reuse) as scope:
    #   conv = tf.keras.layers.Conv2D(
    #       filters=32,
    #       kernel_size=[5, 5],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(x_image)
    #   pool = tf.keras.layers.MaxPool2D(pool_size=[2, 2], strides=2,
padding='valid')(conv)
    #   # N x 16 x 16 x 32

    # with tf.variable_scope('conv2', reuse=reuse) as scope:
    #   conv = tf.keras.layers.Conv2D(
    #       filters=64,
    #       kernel_size=[3, 3],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(pool)
    #   pool = tf.keras.layers.MaxPool2D(pool_size=[2, 2], strides=2,
padding='valid')(conv)
    #   # N x 8 x 8 x 64

    # with tf.variable_scope('conv3', reuse=reuse) as scope:
    #   conv = tf.keras.layers.Conv2D(
    #       filters=64,
    #       kernel_size=[3, 3],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(pool)
    #   pool = tf.keras.layers.MaxPool2D(pool_size=[2, 2], strides=2,
padding='valid')(conv)
    #   # N x 4 x 4 x 64

    # with tf.variable_scope('fully_connected', reuse=reuse) as
scope:
    #   dim =np.prod(pool.shape[1:])
    #   flat = tf.reshape(pool, [-1, dim])
    #   outputs = tf.keras.layers.Dense(units=_NUM_CLASSES,
name=scope.name)(flat)

    # return outputs

def CNN_7layers(x_image, num_cls, reuse=False):
    _NUM_CLASSES = num_cls
    with tf.variable_scope('conv1', reuse=reuse) as scope:
        conv = tf.keras.layers.Conv2D(
            filters=64,
            kernel_size=[3, 3],
            padding='SAME',
            activation=tf.nn.relu
        )(x_image)
        conv = tf.keras.layers.Conv2D(
            filters=64,
            kernel_size=[3, 3],
            padding='SAME',
            activation=tf.nn.relu
        )(conv)

    with tf.variable_scope('conv2', reuse=reuse) as scope:
        conv = tf.keras.layers.Conv2D(
            filters=64,
            kernel_size=[3, 3],
            padding='SAME',
            activation=tf.nn.relu
        )(conv)
        conv = tf.keras.layers.Conv2D(
            filters=64,
            kernel_size=[3, 3],
            padding='SAME',
            activation=tf.nn.relu
        )(conv)

    with tf.variable_scope('conv3', reuse=reuse) as scope:
        conv = tf.keras.layers.Conv2D(
            filters=64,
            kernel_size=[3, 3],
            padding='SAME',
            activation=tf.nn.relu
        )(conv)
        conv = tf.keras.layers.Conv2D(
            filters=64,
            kernel_size=[3, 3],
            padding='SAME',
            activation=tf.nn.relu
        )(conv)

    with tf.variable_scope('fully_connected', reuse=reuse) as scope:
        dim = np.prod(conv.shape[1:])
        flat = tf.reshape(conv, [-1, dim])
        outputs = tf.keras.layers.Dense(units=_NUM_CLASSES,
name=scope.name)(flat)

    return outputs

    # with tf.variable_scope('conv1', reuse=reuse) as scope:
    #   conv = tf.keras.layers.Conv2D(
    #       filters=32,
    #       kernel_size=[5, 5],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(x_image)
    #   conv = tf.keras.layers.Conv2D(
    #       filters=32,
    #       kernel_size=[3, 3],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(conv)
    #   pool = tf.keras.layers.MaxPool2D(pool_size=[2, 2], strides=2,
padding='valid')(conv)
    #   # N x 16 x 16 x 32

    # with tf.variable_scope('conv2', reuse=reuse) as scope:
    #   conv = tf.keras.layers.Conv2D(
    #       filters=64,
    #       kernel_size=[3, 3],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(pool)
    #   conv = tf.keras.layers.Conv2D(
    #       filters=64,
    #       kernel_size=[3, 3],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(conv)
    #   pool = tf.keras.layers.MaxPool2D(pool_size=[2, 2], strides=2,
padding='valid')(conv)
    #   # N x 8 x 8 x 64

    # with tf.variable_scope('conv3', reuse=reuse) as scope:
    #   conv = tf.keras.layers.Conv2D(
    #       filters=64,
    #       kernel_size=[3, 3],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(pool)
    #   conv = tf.keras.layers.Conv2D(
    #       filters=128,
    #       kernel_size=[3, 3],
    #       padding='SAME',
    #       activation=tf.nn.relu
    #   )(conv)
    #   pool = tf.keras.layers.MaxPool2D(pool_size=[2, 2], strides=2,
padding='valid')(conv)
    #   # pool = tf.layers.dropout(pool, rate=0.25, name=scope.name)
    #   # N x 4 x 4 x 128

    # with tf.variable_scope('fully_connected', reuse=reuse) as
scope:
    #   dim = np.prod(pool.shape[1:])
    #   flat = tf.reshape(pool, [-1, dim])
    #   outputs = tf.keras.layers.Dense(units=_NUM_CLASSES,
name=scope.name)(flat)

    # return outputs

def CNN(net, num_cls, dim):

    _NUM_CLASSES = num_cls
    _IMAGE_HEIGHT, _IMAGE_WIDTH, _IMAGE_CHANNELS = dim

    with tf.name_scope('main_params'):
        x = tf.placeholder(tf.float32, shape=[None, _IMAGE_HEIGHT,
_IMAGE_WIDTH, _IMAGE_CHANNELS], name='input_of_net')
        y = tf.placeholder(tf.float32, shape=[None, _NUM_CLASSES],
name='labels')

    # call CNN structure according to string net
    outputs = globals()[net](x, _NUM_CLASSES)
    outputs = tf.identity(outputs, name='output_of_net')

    return (x, y, outputs)

"""## Utilities ##"""

# import numpy as np
# import math
# import scipy.io as sio
# import os
# import math
# import pdb

class ConfigClass(object):
    def __init__(self, args, num_data, num_cls):
        super(ConfigClass, self).__init__()
        self.args = args
        self.iter_max = args.iter_max

        # Different notations of regularization term:
        # In SGD, weight decay:
        #     weight_decay <- lr/(C*num_of_training_samples)
        # In Newton method:
        #     C <- C * num_of_training_samples

        self.seed = args.seed

        if self.seed is None:
            print('You choose not to specify a random seed.'+\
                'A different result is produced after each run.')
        elif isinstance(self.seed, int) and self.seed >= 0:
            print('You specify random seed {}.'.format(self.seed))
        else:
            raise ValueError('Only accept None type or nonnegative 
integers for'+\
                    ' random seed argument!')

        self.train_set = args.train_set
        self.val_set = args.val_set
        self.num_cls = num_cls
        self.dim = args.dim

        self.num_data = num_data
        self.GNsize = min(args.GNsize, self.num_data)
        self.C = args.C * self.num_data
        self.net = args.net

        self.xi = 0.1
        self.CGmax = args.CGmax
        self._lambda = args._lambda
        self.drop = args.drop
        self.boost = args.boost
        self.eta = args.eta
        self.lr = args.lr
        self.lr_decay = args.lr_decay

        self.bsize = args.bsize
        if args.momentum < 0:
            raise ValueError('Momentum needs to be larger than 0!')
        self.momentum = args.momentum

        self.loss = args.loss
        if self.loss not in ('MSELoss', 'CrossEntropy'):
            raise ValueError('Unrecognized loss type!')
        self.optim = args.optim
        if self.optim not in ('SGD', 'NewtonCG', 'Adam'):
            raise ValueError('Only support SGD, Adam & NewtonCG 
optimizer!')
```

```
        self.log_file = args.log_file
        self.model_file = args.model_file
        self.screen_log_only = args.screen_log_only

        if self.screen_log_only:
            print('You choose not to store running log. Only store 
model to {}'.format(self.log_file))
        else:
            print('Saving log to: {}'.format(self.log_file))
            dir_name, _ = os.path.split(self.log_file)
            if not os.path.isdir(dir_name):
                os.makedirs(dir_name, exist_ok=True)

        dir_name, _ = os.path.split(self.model_file)
        if not os.path.isdir(dir_name):
            os.makedirs(dir_name, exist_ok=True)

        self.elapsed_time = 0.0

def read_data(filename, dim, label_enum=None):
    """
    args:
        filename: the path where .mat files are stored
        label_enum (default None): the list that stores the original 
labels.
            If label_enum is None, the function will generate a new 
list which stores the
            original labels in a sequence, and map original labels to 
[0, 1, ... number_of_classes-1].
            If label_enum is a list, the function will use it to 
convert
            original labels to [0, 1,..., number_of_classes-1].
    """

    #mat_contents = sio.loadmat(filename)
    mat_contents = hdf5storage.loadmat(filename)
    images, labels = mat_contents['x'], mat_contents['y']

    labels = labels.reshape(-1)
    images = images.reshape(images.shape[0], -1)

    _IMAGE_HEIGHT, _IMAGE_WIDTH, _IMAGE_CHANNELS = dim
    zero_to_append = np.zeros((images.shape[0],
            _IMAGE_CHANNELS*_IMAGE_HEIGHT*_IMAGE_WIDTH-
np.prod(images.shape[1:])))
    images = np.append(images, zero_to_append, axis=1)

    # check data validity
    if label_enum is None:
        label_enum, labels = np.unique(labels, return_inverse=True)
        num_cls = labels.max() + 1

        if len(label_enum) != num_cls:
            raise ValueError('The number of classes is not equal to 
the number of\
                          labels in dataset. Please verify them.')
    else:
        num_cls = len(label_enum)
        forward_map = dict(zip(label_enum, np.arange(num_cls)))
        labels = np.expand_dims(labels, axis=1)
        labels = np.apply_along_axis(lambda x:forward_map[x[0]],
axis=1, arr=labels)


    # convert groundtruth to one-hot encoding
    labels = np.eye(num_cls)[labels]
    labels = labels.astype('float32')

    return [images, labels], num_cls, label_enum

def normalize_and_reshape(images, dim, mean_tr=None):
    _IMAGE_HEIGHT, _IMAGE_WIDTH, _IMAGE_CHANNELS = dim
    images_shape = [images.shape[0], _IMAGE_CHANNELS, _IMAGE_HEIGHT,
_IMAGE_WIDTH]
    # images normalization and zero centering
    images = images.reshape(images_shape[0], -1)
    #images = (images/255.0).astype(np.single)
    if mean_tr is None:
        #print('No mean of data provided! Normalize images by their
own mean.')
        # if no mean_tr is provided, we calculate it according to the
current data
        mean_tr = images.mean(axis=0)
    else:
        #print('Normalize images according to the provided mean.')
        if np.prod(mean_tr.shape) != np.prod(dim):
            raise ValueError('Dimension of provided mean does not
agree with the data! Please verify them!')
    #images = (images - mean_tr).astype(np.single)

    images = images.reshape(images_shape)
    # Tensorflow accepts data shape: B x H x W x C
    images = np.transpose(images, (0, 2, 3, 1))
    return images, mean_tr


def predict(sess, network, test_batch, bsize):
    x, y, loss, outputs = network

    test_inputs, test_labels = test_batch
    batch_size = bsize

    num_data = test_labels.shape[0]
    num_batches = math.ceil(num_data/batch_size)

    results = np.zeros(shape=num_data, dtype=np.int)
    infer_loss = 0.0

    for i in range(num_batches):
        batch_idx = np.arange(i*batch_size, min((i+1)*batch_size,
num_data))

        batch_input = test_inputs[batch_idx]
        batch_labels = test_labels[batch_idx]

        net_outputs, _loss = sess.run(
            [outputs, loss], feed_dict={x: batch_input, y:
batch_labels}
            )
        results[batch_idx] = np.argmax(net_outputs, axis=1)
        # note that _loss was summed over batches
        infer_loss = infer_loss + _loss

    avg_acc = (np.argmax(test_labels, axis=1) == results).mean()
    avg_loss = infer_loss/num_data

    return avg_loss, avg_acc, results

"""## Newton - CG ##"""

# import pdb
# import tensorflow as tf
# import time
# import numpy as np
# import os
# import math
# from utilities import predict

def Rop(f, weights, v):
    """Implementation of R operator
    Args:
        f: any function of weights
        weights: list of tensors.
        v: vector for right multiplication
    Returns:
        Jv: Jaccobian vector product, length same as
            the number of output of f
    """
    if type(f) == list:
        u = [tf.zeros_like(ff) for ff in f]
    else:
        u = tf.zeros_like(f)  # dummy variable
    g = tf.gradients(ys=f, xs=weights, grad_ys=u)
    return tf.gradients(ys=g, xs=u, grad_ys=v)

def Gauss_Newton_vec(outputs, loss, weights, v):
    """Implements Gauss-Newton vector product.
    Args:
        loss: Loss function.
        outputs: outputs of the last layer (pre-softmax).
        weights: Weights, list of tensors.
        v: vector to be multiplied with Gauss Newton matrix
    Returns:
        J'BJv: Guass-Newton vector product.
    """
    # Validate the input
    if type(weights) == list:
        if len(v) != len(weights):
            raise ValueError("weights and v must have the same
length.")

    grads_outputs = tf.gradients(ys=loss, xs=outputs)
    BJv = Rop(grads_outputs, weights, v)
    JBJv = tf.gradients(ys=outputs, xs=weights, grad_ys=BJv)
    return JBJv


class newton_cg(object):
    def __init__(self, config, sess, outputs, loss):
        """
        initialize operations and vairables that will be used in
newton
        args:
            sess: tensorflow session
            outputs: output of the neural network (pre-softmax layer)
            loss: function to calculate loss
        """
        super(newton_cg, self).__init__()
        self.sess = sess
        self.config = config
        self.outputs = outputs
        self.loss = loss
        self.param = tf.compat.v1.trainable_variables()

        self.CGiter = 0
        FLOAT = tf.float32
        model_weight = self.vectorize(self.param)

        # initial variable used in CG
        zeros = tf.zeros(model_weight.get_shape(), dtype=FLOAT)
        self.r = tf.Variable(zeros, dtype=FLOAT, trainable=False)
        self.v = tf.Variable(zeros, dtype=FLOAT, trainable=False)
        self.s = tf.Variable(zeros, dtype=FLOAT, trainable=False)
        self.g = tf.Variable(zeros, dtype=FLOAT, trainable=False)
        # initial Gv, f for method minibatch
        self.Gv = tf.Variable(zeros, dtype=FLOAT, trainable=False)
        self.f = tf.Variable(0., dtype=FLOAT, trainable=False)

        # rTr, cgtol and beta to be used in CG
        self.rTr = tf.Variable(0., dtype=FLOAT, trainable=False)
        self.cgtol = tf.Variable(0., dtype=FLOAT, trainable=False)
        self.beta = tf.Variable(0., dtype=FLOAT, trainable=False)

        # placeholder alpha, old_alpha and lambda
        self.alpha = tf.compat.v1.placeholder(FLOAT, shape=[])
        self.old_alpha = tf.compat.v1.placeholder(FLOAT, shape=[])
        self._lambda = tf.compat.v1.placeholder(FLOAT, shape=[])

        self.num_grad_segment =
math.ceil(self.config.num_data/self.config.bsize)
        self.num_Gv_segment =
math.ceil(self.config.GNsize/self.config.bsize)

        cal_loss, cal_lossgrad, cal_lossGv, \
        add_reg_avg_loss, add_reg_avg_grad, add_reg_avg_Gv, \
        zero_loss, zero_grad, zero_Gv = self._ops_in_minibatch()

        # initial operations that will be used in minibatch and
newton
        self.cal_loss = cal_loss
        self.cal_lossgrad = cal_lossgrad
        self.cal_lossGv = cal_lossGv
        self.add_reg_avg_loss = add_reg_avg_loss
        self.add_reg_avg_grad = add_reg_avg_grad
        self.add_reg_avg_Gv = add_reg_avg_Gv
        self.zero_loss = zero_loss
        self.zero_grad = zero_grad
        self.zero_Gv = zero_Gv
        self.CG, self.update_v = self._CG()
        self.init_cg_vars = self._init_cg_vars()
        self.update_gs = tf.tensordot(self.s, self.g, axes=1)
        self.update_sGs = 0.5*tf.tensordot(self.s, -self.g-self.r-
self._lambda*self.s, axes=1)
        self.update_model = self._update_model()
        self.gnorm = self.calc_norm(self.g)


    def vectorize(self, tensors):
        if isinstance(tensors, list) or isinstance(tensors, tuple):
            vector = [tf.reshape(tensor, [-1]) for tensor in tensors]
            return tf.concat(vector, 0)
        else:
            return tensors

    def inverse_vectorize(self, vector, param):
        if isinstance(vector, list):
            return vector
        else:
            tensors = []
            offset = 0
            num_total_param = np.sum([np.prod(p.shape.as_list()) for
p in param])
            for p in param:
                numel = np.prod(p.shape.as_list())
                tensors.append(tf.reshape(vector[offset:
offset+numel], p.shape))
                offset += numel

            assert offset == num_total_param
            return tensors

    def calc_norm(self, v):
        # default: frobenius norm
        if isinstance(v, list):
            norm = 0.
            for p in v:
                norm = norm + tf.norm(tensor=p)**2
            return norm**0.5
        else:
            return tf.norm(tensor=v)

    def _ops_in_minibatch(self):
        """
        Define operations that will be used in method minibatch
        Vectorization is already a deep copy operation.
        Before using newton method, loss needs to be summed over
training samples
        to make results consistent.
        """

        def cal_loss():
            return tf.compat.v1.assign(self.f, self.f + self.loss)

        def cal_lossgrad():
            update_f = tf.compat.v1.assign(self.f, self.f +
self.loss)
```

```
            grad = tf.gradients(ys=self.loss, xs=self.param)
            grad = self.vectorize(grad)
            update_grad = tf.compat.v1.assign(self.g, self.g + grad)

            return tf.group(*[update_f, update_grad])

        def cal_lossGv():
            v = self.inverse_vectorize(self.v, self.param)
            Gv = Gauss_Newton_vec(self.outputs, self.loss, 
self.param, v)
            Gv = self.vectorize(Gv)
            return tf.compat.v1.assign(self.Gv, self.Gv + Gv)

        # add regularization term to loss, gradient and Gv and 
further average over batches
        def add_reg_avg_loss():
            model_weight = self.vectorize(self.param)
            reg = (self.calc_norm(model_weight))**2
            reg = 1.0/(2*self.config.C) * reg
            return tf.compat.v1.assign(self.f, reg + 
self.f/self.config.num_data)

        def add_reg_avg_lossgrad():
            model_weight = self.vectorize(self.param)
            reg_grad = model_weight/self.config.C
            return tf.compat.v1.assign(self.g, reg_grad + 
self.g/self.config.num_data)

        def add_reg_avg_lossGv():
            return tf.compat.v1.assign(self.Gv, (self._lambda + 
1/self.config.C)*self.v
             + self.Gv/self.config.GNsize)

        # zero out loss, grad and Gv
        def zero_loss():
            return tf.compat.v1.assign(self.f, tf.zeros_like(self.f))
        def zero_grad():
            return tf.compat.v1.assign(self.g, tf.zeros_like(self.g))
        def zero_Gv():
            return tf.compat.v1.assign(self.Gv, 
tf.zeros_like(self.Gv))

        return (cal_loss(), cal_lossgrad(), cal_lossGv(),
                add_reg_avg_loss(), add_reg_avg_lossgrad(), 
add_reg_avg_lossGv(),
                zero_loss(), zero_grad(), zero_Gv())

    def minibatch(self, data_batch, place_holder_x, place_holder_y, 
mode):
        """
        A function to evaluate either function value, global gradient 
or sub-sampled Gv
        """
        if mode not in ('funonly', 'fungrad', 'Gv'):
            raise ValueError('Unknown mode other than funonly & 
fungrad & Gv!')

        inputs, labels = data_batch
        num_data = labels.shape[0]
        num_segment = math.ceil(num_data/self.config.bsize)
        x, y = place_holder_x, place_holder_y

        # before estimation starts, need to zero out f, grad and Gv
according to the mode

        if mode == 'funonly':
            assert num_data == self.config.num_data
            assert num_segment == self.num_grad_segment
            self.sess.run(self.zero_loss)
        elif mode == 'fungrad':
            assert num_data == self.config.num_data
            assert num_segment == self.num_grad_segment
            self.sess.run([self.zero_loss, self.zero_grad])
        else:
            assert num_data == self.config.GNsize
            assert num_segment == self.num_Gv_segment
            self.sess.run(self.zero_Gv)

        for i in range(num_segment):

            load_time = time.time()
            idx = np.arange(i * self.config.bsize, min((i+1) *
self.config.bsize, num_data))
            batch_input = inputs[idx]
            batch_labels = labels[idx]
            batch_input = np.ascontiguousarray(batch_input)
            batch_labels = np.ascontiguousarray(batch_labels)
            self.config.elapsed_time += time.time() - load_time

            if mode == 'funonly':

                self.sess.run(self.cal_loss, feed_dict={
                            x: batch_input,
                            y: batch_labels,})

            elif mode == 'fungrad':

                self.sess.run(self.cal_lossgrad, feed_dict={
                            x: batch_input,
                            y: batch_labels,})

            else:

                self.sess.run(self.cal_lossGv, feed_dict={
                            x: batch_input,
                            y: batch_labels})

        # average over batches
        if mode == 'funonly':
            self.sess.run(self.add_reg_avg_loss)
        elif mode == 'fungrad':
            self.sess.run([self.add_reg_avg_loss,
self.add_reg_avg_grad])
        else:
            self.sess.run(self.add_reg_avg_Gv,
                feed_dict={self._lambda: self.config._lambda})


    def _update_model(self):
        update_model_ops = []
        x = self.inverse_vectorize(self.s, self.param)
        for i, p in enumerate(self.param):
            op = tf.compat.v1.assign(p, p + (self.alpha-
self.old_alpha) * x[i])
            update_model_ops.append(op)
        return tf.group(*update_model_ops)

    def _init_cg_vars(self):
        init_ops = []

        init_r = tf.compat.v1.assign(self.r, -self.g)
        init_v = tf.compat.v1.assign(self.v, -self.g)
        init_s = tf.compat.v1.assign(self.s, tf.zeros_like(self.g))
        gnorm = self.calc_norm(self.g)
        init_rTr = tf.compat.v1.assign(self.rTr, gnorm**2)
        init_cgtol = tf.compat.v1.assign(self.cgtol,
self.config.xi*gnorm)

        init_ops = [init_r, init_v, init_s, init_rTr, init_cgtol]

        return tf.group(*init_ops)

    def _CG(self):
        """
        CG:
            define operations that will be used in method newton
        Same as the previous loss calculation,
        Gv has been summed over batches when samples were fed into
Neural Network.
        """

        def CG_ops():

            vGv = tf.tensordot(self.v, self.Gv, axes=1)

            alpha = self.rTr / vGv
            with tf.control_dependencies([alpha]):
                update_s = tf.compat.v1.assign(self.s, self.s + alpha
* self.v, name='update_s_ops')
                update_r = tf.compat.v1.assign(self.r, self.r - alpha
* self.Gv, name='update_r_ops')

                with tf.control_dependencies([update_s, update_r]):
                    rnewTrnew = self.calc_norm(update_r)**2
                    update_beta = tf.compat.v1.assign(self.beta,
rnewTrnew / self.rTr)
                    with tf.control_dependencies([update_beta]):
                        update_rTr = tf.compat.v1.assign(self.rTr,
rnewTrnew, name='update_rTr_ops')

            return tf.group(*[update_s, update_beta, update_rTr])

        def update_v():
            return tf.compat.v1.assign(self.v, self.r +
self.beta*self.v, name='update_v')

        return (CG_ops(), update_v())
    def newton(self, full_batch, val_batch, saver, network,
test_network=None):
        """
        Conduct newton steps for training
        args:
            full_batch & val_batch: provide training set and
validation set. The function will
                save the best model evaluted on validation set for
future prediction.
            network: a tuple contains (x, y, loss, outputs).
            test_network: a tuple similar to argument network. If you
use layers which behave differently
                in test phase such as batchnorm, a separate
test_network is needed.
        return:
            None
        """
        # check whether data is valid
        full_inputs, full_labels = full_batch
        assert full_inputs.shape[0] == full_labels.shape[0]

        if full_inputs.shape[0] != self.config.num_data:
            raise ValueError('The number of full batch inputs does
not agree with the config argument.\
                            This is important because global loss is
averaged over those inputs')

        x, y, _, outputs = network

        tf.compat.v1.summary.scalar('loss', self.f)
        merged = tf.compat.v1.summary.merge_all()
        train_writer =
tf.compat.v1.summary.FileWriter('./summary/train', self.sess.graph)

        print(self.config.args)
        if not self.config.screen_log_only:
            log_file = open(self.config.log_file, 'w')
            print(self.config.args, file=log_file)

        self.minibatch(full_batch, x, y, mode='fungrad')
        f = self.sess.run(self.f)
        output_str = 'initial f: {:.3f}'.format(f)
        print(output_str)
        if not self.config.screen_log_only:
            print(output_str, file=log_file)

        best_acc = 0.0

        total_running_time = 0.0
        self.config.elapsed_time = 0.0
        total_CG = 0

        for k in range(self.config.iter_max):

            # randomly select the batch for Gv estimation
            idx = np.random.choice(np.arange(0,
full_labels.shape[0]),
                    size=self.config.GNsize, replace=False)

            mini_inputs = full_inputs[idx]
            mini_labels = full_labels[idx]

            start = time.time()

            self.sess.run(self.init_cg_vars)
            cgtol = self.sess.run(self.cgtol)

            avg_cg_time = 0.0
            for CGiter in range(1, self.config.CGmax+1):

                cg_time = time.time()
                self.minibatch((mini_inputs, mini_labels), x, y,
mode='Gv')
                avg_cg_time += time.time() - cg_time

                self.sess.run(self.CG)

                rnewTrnew = self.sess.run(self.rTr)

                if rnewTrnew**0.5 <= cgtol or CGiter ==
self.config.CGmax:
                    break

                self.sess.run(self.update_v)

            print('Avg time per Gv iteration: {:.5f}
s\r\n'.format(avg_cg_time/CGiter))

            gs, sGs = self.sess.run([self.update_gs,
self.update_sGs], feed_dict={
                    self._lambda: self.config._lambda
                })

            # line_search
            f_old = f
            alpha = 1
            while True:

                old_alpha = 0 if alpha == 1 else alpha/0.5

                self.sess.run(self.update_model, feed_dict={
                    self.alpha:alpha, self.old_alpha:old_alpha
                    })

                prered = alpha*gs + (alpha**2)*sGs

                self.minibatch(full_batch, x, y, mode='funonly')
                f = self.sess.run(self.f)

                actred = f - f_old

                if actred <= self.config.eta*alpha*gs:
                    break

                alpha *= 0.5

            # update lambda
            ratio = actred / prered
            if ratio < 0.25:
                self.config._lambda *= self.config.boost
            elif ratio >= 0.75:
                self.config._lambda *= self.config.drop

            self.minibatch(full_batch, x, y, mode='fungrad')
            f = self.sess.run(self.f)

            gnorm = self.sess.run(self.gnorm)

            summary = self.sess.run(merged)
            train_writer.add_summary(summary, k)

            # exclude data loading time for fair comparison
            end = time.time()

            end = end - self.config.elapsed_time
            total_running_time += end-start

            self.config.elapsed_time = 0.0

            total_CG += CGiter

            output_str = '{}-iter f: {:.3f} |g|: {:.5f} alpha: {:.3e}
ratio: {:.3f} lambda: {:.5f} #CG: {} actred: {:.5f} prered: {:.5f}
time: {:.3f}'.\
                            format(k, f, gnorm, alpha, actred/prered,
self.config._lambda, CGiter, actred, prered, end-start)
            print(output_str)
            if not self.config.screen_log_only:
                print(output_str, file=log_file)

            if val_batch is not None:
                # Evaluate the performance after every Newton Step
                if test_network == None:
                    val_loss, val_acc, _ = predict(
                        self.sess,
                        network=(x, y, self.loss, outputs),
                        test_batch=val_batch,
                        bsize=self.config.bsize,
                        )
                else:
                    # A separat test network part has not been
done...
                    val_loss, val_acc, _ = predict(
                        self.sess,
                        network=test_network,
                        test_batch=val_batch,
                        bsize=self.config.bsize
                        )

                output_str = '\r\n {}-iter val_acc: {:.3f}% val_loss
{:.3f}\r\n'.\
                    format(k, val_acc*100, val_loss)
                print(output_str)
                if not self.config.screen_log_only:
                    print(output_str, file=log_file)

                if val_acc > best_acc:
                    best_acc = val_acc
                    checkpoint_path = self.config.model_file
                    save_path = saver.save(self.sess,
checkpoint_path)
                    print('Best model saved in
{}\r\n'.format(save_path))

        if val_batch is None:
            checkpoint_path = self.config.model_file
            save_path = saver.save(self.sess, checkpoint_path)
            print('Model at the last iteration saved in
{}\r\n'.format(save_path))
            output_str = 'total_#CG {} | total running time
{:.3f}s'.format(total_CG, total_running_time)
        else:
            output_str = 'Final acc: {:.3f}% | best acc {:.3f}% |
total_#CG {} | total running time {:.3f}s'.\
                format(val_acc*100, best_acc*100, total_CG,
total_running_time)
        print(output_str)
        if not self.config.screen_log_only:
            print(output_str, file=log_file)
            log_file.close()

"""##Set Train Arguments##"""

if USE_HFO:
    # Arguments for HFO - PSSP dataset
    train_args = ("--optim NewtonCG --GNsize 50 --C 0.05 --net
CNN_4layers --bsize 1024 --iter_max 10 " +
              "--train_set " + TRAIN_FILE + " --val_set " +
VALID_FILE + " --dim " +
              str(HEIGHT) + " " + str(WIDTH) + " " +
str(DEPTH)).split()
else:
    # Arguments for SGD - PSSP dataset
    train_args = ("--optim SGD --lr 0.05 --momentum 0.01 --C 0.01 --
net CNN_4layers --bsize 1024 --epoch_max 1000 " +
              "--train_set " + TRAIN_FILE + " --val_set " +
VALID_FILE + " --dim " +
              str(HEIGHT) + " " + str(WIDTH) + " " +
str(DEPTH)).split()

"""##Declare Train Function##"""

# import pdb
# import numpy as np
# import tensorflow as tf
# tf.compat.v1.disable_eager_execution()
# import time
# import math
# import argparse

# from net.net import CNN
# from newton_cg import newton_cg
# from utilities import read_data, predict, ConfigClass,
normalize_and_reshape

def parse_args():
    parser = argparse.ArgumentParser(description='Newton method on
DNN')
    parser.add_argument('--C', dest='C',
                      help='regularization term, or so-called weight
decay where'+\
                              'weight_decay = lr/(C*num_of_samples)
in this implementation' ,
                      default=0.01, type=float)

    # Newton method arguments
    parser.add_argument('--GNsize', dest='GNsize',
                      help='number of samples for estimating Gauss-
Newton matrix',
                      default=4096, type=int)
    parser.add_argument('--iter_max', dest='iter_max',
                      help='the maximal number of Newton iterations',
                      default=100, type=int)
    parser.add_argument('--xi', dest='xi',
                      help='the tolerance in the relative stopping
condition for CG',
                      default=0.1, type=float)
    parser.add_argument('--drop', dest='drop',
                      help='the drop constants for the LM method',
                      default=2/3, type=float)
    parser.add_argument('--boost', dest='boost',
                      help='the boost constants for the LM method',
                      default=3/2, type=float)
    parser.add_argument('--eta', dest='eta',
                      help='the parameter for the line search
stopping condition',
                      default=0.0001, type=float)
    parser.add_argument('--CGmax', dest='CGmax',
                      help='the maximal number of CG iterations',
                      default=250, type=int)
    parser.add_argument('--lambda', dest='_lambda',
                      help='the initial lambda for the LM method',
                      default=1, type=float)

    # SGD arguments
    parser.add_argument('--epoch_max', dest='epoch',
                      help='number of training epoch',
                      default=500, type=int)
    parser.add_argument('--lr', dest='lr',
                      help='learning rate',
                      default=0.01, type=float)
    parser.add_argument('--decay', dest='lr_decay',
                      help='learning rate decay over each mini-batch
update',
                      default=0, type=float)
    parser.add_argument('--momentum', dest='momentum',
                      help='momentum of learning',
                      default=0, type=float)

    # Model training arguments
    parser.add_argument('--bsize', dest='bsize',
                      help='batch size to evaluate stochastic
gradient, Gv, etc. Since the sampled data \
                      for computing Gauss-Newton matrix and etc.
might not fit into memeory \
                      for one time, we will split the data into
several segements and average\
                      over them.',
                      default=1024, type=int)
    parser.add_argument('--net', dest='net',
                        help='classifier type',
                        default='CNN_4layers', type=str)
    parser.add_argument('--train_set', dest='train_set',
                        help='provide the directory of .mat file for
training',
                        default=None, type=str)
    parser.add_argument('--val_set', dest='val_set',
                        help='provide the directory of .mat file for
validation',
                        default=None, type=str)
    parser.add_argument('--model', dest='model_file',
                        help='model saving address',
                        default='./saved_model/model.ckpt', type=str)
    parser.add_argument('--log', dest='log_file',
                        help='log saving directory',
                        default='./running_log/logger.log', type=str)
    parser.add_argument('--screen_log_only', dest='screen_log_only',
                        help='screen printing running log instead of
storing it',
                        action='store_true')
    parser.add_argument('--optim', '-optim',
                        help='which optimizer to use: SGD, Adam or
NewtonCG',
                        default='NewtonCG', type=str)
    parser.add_argument('--loss', dest='loss',
                        help='which loss function to use: MSELoss or
CrossEntropy',
                        default='MSELoss', type=str)
    parser.add_argument('--dim', dest='dim', nargs='+', help='input
dimension of data,'+\
                          'shape must be:  height width num_channels',
                        default=[32, 32, 3], type=int)
    parser.add_argument('--seed', dest='seed', help='a nonnegative
integer for \
                          reproducibility', type=int)

    args = parser.parse_args(args=train_args)
    return args


args = parse_args()

def init_model(param):
    init_ops = []
    for p in param:
        if 'kernel' in p.name:
            weight = np.random.standard_normal(p.shape)* np.sqrt(2.0
/ ((np.prod(p.get_shape().as_list()[:-1]))))
            opt = tf.compat.v1.assign(p, weight)
        elif 'bias' in p.name:
            zeros = np.zeros(p.shape)
            opt = tf.compat.v1.assign(p, zeros)
        init_ops.append(opt)
    return tf.group(*init_ops)

def gradient_trainer(config, sess, network, full_batch, val_batch,
saver, test_network):
    x, y, loss, outputs,  = network
    global_step = tf.Variable(initial_value=0, trainable=False,
name='global_step')
    learning_rate = tf.compat.v1.placeholder(tf.float32, shape=[],
name='learning_rate')

    # Probably not a good way to add regularization.
    # Just to confirm the implementation is the same as MATLAB.
    reg = 0.0
    param = tf.compat.v1.trainable_variables()
    for p in param:
        reg = reg + tf.reduce_sum(input_tensor=tf.pow(p,2))
    reg_const = 1/(2*config.C)
    batch_size = tf.compat.v1.cast(tf.shape(x)[0], tf.float32)
    loss_with_reg = reg_const*reg + loss/batch_size

    if config.optim == 'SGD':
        optimizer = tf.compat.v1.train.MomentumOptimizer(
                    learning_rate=learning_rate,
                    momentum=config.momentum).minimize(
                    loss_with_reg,
                    global_step=global_step)
    elif config.optim == 'Adam':
        optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=learning_rate,
                                beta1=0.9,
                                beta2=0.999,
                                epsilon=1e-08).minimize(
                                loss_with_reg,
                                global_step=global_step)

    train_inputs, train_labels = full_batch
    num_data = train_labels.shape[0]
    num_iters = math.ceil(num_data/config.bsize)

    print(config.args)
    if not config.screen_log_only:
        log_file = open(config.log_file, 'w')
        print(config.args, file=log_file)
    sess.run(tf.compat.v1.global_variables_initializer())


    print('-------------- initializing network by methods in He et
al. (2015) --------------')
    param = tf.compat.v1.trainable_variables()
    sess.run(init_model(param))

    total_running_time = 0.0
    best_acc = 0.0
    lr = config.lr

    for epoch in range(0, args.epoch):

        loss_avg = 0.0
        start = time.time()

        for i in range(num_iters):

            load_time = time.time()
            # randomly select the batch
            idx = np.random.choice(np.arange(0, num_data),
                    size=config.bsize, replace=False)

            batch_input = train_inputs[idx]
            batch_labels = train_labels[idx]
            batch_input = np.ascontiguousarray(batch_input)
            batch_labels = np.ascontiguousarray(batch_labels)
            config.elapsed_time += time.time() - load_time

            step, _, batch_loss= sess.run(
                [global_step, optimizer, loss_with_reg],
                feed_dict = {x: batch_input, y: batch_labels,
learning_rate: lr}
                )

            # print initial loss
            if epoch == 0 and i == 0:
                output_str = 'initial f (reg + avg. loss of 1st
batch): {:.3f}'.format(batch_loss)
                print(output_str)
                if not config.screen_log_only:
                    print(output_str, file=log_file)

            loss_avg = loss_avg + batch_loss
            # print log every 10% of the iterations
            if i % math.ceil(num_iters/10) == 0:
                end = time.time()
                output_str = 'Epoch {}: {}/{} | loss {:.4f} | lr
{:.6} | elapsed time {:.3f}'\
                    .format(epoch, i, num_iters, batch_loss , lr,
end-start)
                print(output_str)
                if not config.screen_log_only:
                    print(output_str, file=log_file)

            # adjust learning rate for SGD by inverse time decay
            if args.optim != 'Adam':
                lr = config.lr/(1 + args.lr_decay*step)

        # exclude data loading time for fair comparison
        epoch_end = time.time() - config.elapsed_time
        total_running_time += epoch_end - start
        config.elapsed_time = 0.0

        if val_batch is None:
            output_str = 'In epoch {} train loss: {:.3f} | epoch time
{:.3f}'\
                .format(epoch, loss_avg/(i+1), epoch_end-start)
        else:
            if test_network == None:
                val_loss, val_acc, _ = predict(
                    sess,
                    network=(x, y, loss, outputs),
                    test_batch=val_batch,
                    bsize=config.bsize
                    )
            else:
                # A separat test network part have been done...
                val_loss, val_acc, _ = predict(
                    sess,
                    network=test_network,
                    test_batch=val_batch,
                    bsize=config.bsize
                    )

            output_str = 'In epoch {} train loss: {:.3f} | val loss:
{:.3f} | val accuracy: {:.3f}% | epoch time {:.3f}'\
                .format(epoch, loss_avg/(i+1), val_loss, val_acc*100,
epoch_end-start)

            if val_acc > best_acc:
                best_acc = val_acc
                checkpoint_path = config.model_file
                save_path = saver.save(sess, checkpoint_path)
                print('Saved best model in {}'.format(save_path))

        print(output_str)
        if not config.screen_log_only:
            print(output_str, file=log_file)

    if val_batch is None:
        checkpoint_path = config.model_file
        save_path = saver.save(sess, checkpoint_path)
        print('Model at the last iteration saved in
{}\r\n'.format(save_path))
        output_str = 'total running time
{:.3f}s'.format(total_running_time)
    else:
        output_str = 'Final acc: {:.3f}% | best acc {:.3f}% | total
running time {:.3f}s'\
            .format(val_acc*100, best_acc*100, total_running_time)

    print(output_str)
    if not config.screen_log_only:
        print(output_str, file=log_file)
        log_file.close()

def newton_trainer(config, sess, network, full_batch, val_batch,
saver, test_network):
    _, _, loss, outputs = network
    newton_solver = newton_cg(config, sess, outputs, loss)
    sess.run(tf.compat.v1.global_variables_initializer())

    print('-------------- initializing network by methods in He et
al. (2015) --------------')
    param = tf.compat.v1.trainable_variables()
    sess.run(init_model(param))
    newton_solver.newton(full_batch, val_batch, saver, network,
test_network)


def train_model():
    full_batch, num_cls, label_enum =
read_data(filename=args.train_set, dim=args.dim)

    if args.val_set is None:
        print('No validation set is provided. Will output model at
the last iteration.')
        val_batch = None
    else:
        val_batch, _, _ = read_data(filename=args.val_set,
dim=args.dim, label_enum=label_enum)

    num_data = full_batch[0].shape[0]

    config = ConfigClass(args, num_data, num_cls)

    if isinstance(config.seed, int):
        tf.compat.v1.random.set_random_seed(config.seed)
        np.random.seed(config.seed)

    if config.net in ('CNN_4layers', 'CNN_7layers', 'VGG11', 'VGG13',
'VGG16','VGG19'):
        x, y, outputs = CNN(config.net, num_cls, config.dim)
        test_network = None
    else:
        raise ValueError('Unrecognized training model')

    if config.loss == 'MSELoss':
        loss = tf.reduce_sum(input_tensor=tf.pow(outputs-y, 2))
    else:
        loss =
tf.reduce_sum(input_tensor=tf.nn.softmax_cross_entropy_with_logits(lo
gits=outputs, labels=y))

    network = (x, y, loss, outputs)

    sess_config = tf.compat.v1.ConfigProto()
    sess_config.gpu_options.allow_growth = True

    with tf.compat.v1.Session(config=sess_config) as sess:

        full_batch[0], mean_tr = normalize_and_reshape(full_batch[0],
dim=config.dim, mean_tr=None)
        if val_batch is not None:
            val_batch[0], _ = normalize_and_reshape(val_batch[0],
dim=config.dim, mean_tr=mean_tr)

        param = tf.compat.v1.trainable_variables()

        mean_param = tf.compat.v1.get_variable(name='mean_tr',
initializer=mean_tr, trainable=False,
                    validate_shape=True, use_resource=False)
        label_enum_var=tf.compat.v1.get_variable(name='label_enum',
initializer=label_enum, trainable=False,
                    validate_shape=True, use_resource=False)
        saver = tf.compat.v1.train.Saver(var_list=param+[mean_param])

        if config.optim in ('SGD', 'Adam'):
            gradient_trainer(
                config, sess, network, full_batch, val_batch, saver,
test_network)
        elif config.optim == 'NewtonCG':
            newton_trainer(
                config, sess, network, full_batch, val_batch, saver,
test_network=test_network)

"""## Train ##"""

train_model()
```

```
"""## Predict ##"""

# Arguments for prediction PSSP dataset
pred_args = ("--bsize 1024 --valid_set " + VALID_FILE + " --train_set
" + TRAIN_FILE +
                         " --model ./saved_model/model.ckpt --dim " +
             str(HEIGHT) + " " + str(WIDTH) + " " +
str(DEPTH)).split()

# valid_f =
"/content/drive/MyDrive/Datasets/{0}_test_fold{1}.txt".format(dataset
.lower(),str(fold)) # train set
# train_f =
"/content/drive/MyDrive/Datasets/{0}_train_fold{1}.txt".format(datase
t.lower(),str(fold)) # validation set
# test_f = "/content/drive/MyDrive/Datasets/CASP13_3class.txt" # test
set CASP13
# print(valid_f)
# print(train_f)
# print(test_f)

valid_origin = "https://gitlab.com/schatz06/pssp/-
/raw/master/Datasets/{0}/{1}_folds/{2}_test_fold{3}.txt".format(datas
et, dataset.lower(),dataset.lower(),fold )
train_origin = "https://gitlab.com/schatz06/pssp/-
/raw/master/Datasets/{0}/{1}_folds/{2}_train_fold{3}.txt".format(data
set, dataset.lower(),dataset.lower(),fold)
test_origin = "https://gitlab.com/schatz06/pssp/-
/raw/master/Datasets/CASP13/CASP13_3class.txt"
train_origin, valid_origin, test_origin

import requests
valid_f = requests.get(valid_origin)
valid_f = valid_f.text.split('\n')[0:-1]
train_f = requests.get(train_origin)
train_f = train_f.text.split('\n')[0:-1]
test_f = requests.get(test_origin)
test_f = test_f.text.split('\n')[0:-1]

VALID_PRED_FILE="pred_test_fold{0}.txt".format(fold)
TRAIN_PRED_FILE="pred_train_fold{0}.txt".format(fold)
TEST_PRED_FILE="pred_casp13_fold{0}.txt".format(fold)
print(VALID_PRED_FILE)
print(TRAIN_PRED_FILE)
print(TEST_PRED_FILE)

"""##Declare Predict Methods##"""

def create_output_pred(pred, origin_f, outFileName):
    pred = pred.astype(int)
    labels = ['C', 'H', 'E']
    counter = 0
    with open(outFileName, 'w') as out_file:
        for line in range(0, len(origin_f)//3):
            protein_name = origin_f[line*3]
            primary_structure = origin_f[line*3+1].replace('!', '')
            secondary_structure = origin_f[line*3+2].replace('!', '')
            prediction = ""
            for c in secondary_structure:
                if (c != '!'):
                    prediction = prediction + labels[pred[counter]]
                    counter += 1
            out_file.write(protein_name + "\n")
            out_file.write(primary_structure + "\n")
            out_file.write(secondary_structure + "\n")
            out_file.write(prediction + "\n")

# def create_output_pred(pred, origin_f, outFileName):
#     labels = ['C','H','E']
#     read_file = open(origin_f,"r")
#     output_file = open(outFileName,"w")
#     count =1
#     target_name =1
#     target_primary = 2
#     target_secondary = 3
#     counter = 0
#     while True:
#         line = read_file.readline()
#         if not line:
#             break
#         if count == target_name:
#             output_file.write(line)
#             target_name+=3
#         if count == target_primary:
#             output_file.write(line)
#             target_primary+=3
#         if count == target_secondary:
#             output_file.write(line)
#             target_secondary+=3
#             line = line.replace("\n","")
#             prediction = ""
#             for c in line:
#                 if (c!='!'):
#                     prediction = prediction + labels[pred[counter]]
#                     counter +=1
#             output_file.write(prediction + "\n")
#         count+=1

# import tensorflow as tf
# tf.compat.v1.disable_eager_execution()
# from utilities import predict, read_data, normalize_and_reshape
# from net.net import CNN
# import numpy as np
# import argparse
# import pdb

def parse_args():
        parser = argparse.ArgumentParser(description='prediction')
        parser.add_argument('--test_set', dest='test_set',
                            help='provide the directory of .mat file
for testing',
                            default=None, type=str)
        parser.add_argument('--valid_set', dest='valid_set',
                            help='provide the directory of .mat file
for validation',
                            default=None, type=str)
        parser.add_argument('--train_set', dest='train_set',
                            help='provide the directory of .mat file
for training',
                            default=None, type=str)
        parser.add_argument('--model', dest='model_file',
                            help='provide file storing network
parameters, i.e. ./dir/model.ckpt',
                            default='./saved_model/model.ckpt',
type=str)
        parser.add_argument('--bsize', dest='bsize',
                            help='batch size',
                            default=1024, type=int)
        parser.add_argument('--loss', dest='loss',
                            help='which loss function to use: MSELoss
or CrossEntropy',
                            default='MSELoss', type=str)
        parser.add_argument('--dim', dest='dim', nargs='+',
help='input dimension of data,'+\
                            'shape must be:  height width
num_channels',
                            default=[32, 32, 3], type=int)

        args = parser.parse_args(args=pred_args)
        return args

def predict_model():
        args = parse_args()

        sess_config = tf.compat.v1.ConfigProto()
        sess_config.gpu_options.allow_growth = True

        with tf.compat.v1.Session(config=sess_config) as sess:
                graph_address = args.model_file + '.meta'
                imported_graph =
tf.compat.v1.train.import_meta_graph(graph_address)
                imported_graph.restore(sess, args.model_file)
                mean_param = [v for v in
tf.compat.v1.global_variables() if 'mean_tr:0' in v.name][0]
                label_enum_var = [v for v in
tf.compat.v1.global_variables() if 'label_enum:0' in v.name][0]


sess.run(tf.compat.v1.variables_initializer([mean_param,
label_enum_var]))
                mean_tr = sess.run(mean_param)
                label_enum = sess.run(label_enum_var)

                x =
tf.compat.v1.get_default_graph().get_tensor_by_name('main_params/inpu
t_of_net:0')
                y =
tf.compat.v1.get_default_graph().get_tensor_by_name('main_params/labe
ls:0')
                outputs =
tf.compat.v1.get_default_graph().get_tensor_by_name('output_of_net:0'
)

                if args.loss == 'MSELoss':
                        loss =
tf.reduce_sum(input_tensor=tf.pow(outputs-y, 2))
                else:
                        loss = tf.reduce_sum(input_tensor=
tf.nn.softmax_cross_entropy_with_logits(logits=outputs,
labels=tf.stop_gradient(y)))

                network = (x, y, loss, outputs)

                if args.valid_set is not None:
                        valid_batch, num_cls, _ =
read_data(args.valid_set, dim=args.dim, label_enum=label_enum)
                        valid_batch[0], _ =
normalize_and_reshape(valid_batch[0], dim=args.dim, mean_tr=mean_tr)

                        avg_loss_valid, avg_acc_valid, results_valid
= predict(sess, network, valid_batch, args.bsize)

                        # convert results back to the original labels
                        inverse_map = dict(zip(np.arange(num_cls),
label_enum))
                        results_valid = np.expand_dims(results_valid,
axis=1)
                        results_valid = np.apply_along_axis(lambda x:
inverse_map[x[0]], axis=1, arr=results_valid)
                        create_output_pred(results_valid, valid_f,
VALID_PRED_FILE)
                        print('In valid phase, average loss: {:.3f} |
average accuracy: {:.3f}%'.\
                            format(avg_loss_valid,
avg_acc_valid*100))

                if args.train_set is not None:
                        train_batch, num_cls, _ =
read_data(args.train_set, dim=args.dim, label_enum=label_enum)
                        train_batch[0], _ =
normalize_and_reshape(train_batch[0], dim=args.dim, mean_tr=mean_tr)

                        avg_loss_train, avg_acc_train, results_train
= predict(sess, network, train_batch, args.bsize)
                        # convert results back to the original labels
                        inverse_map = dict(zip(np.arange(num_cls),
label_enum))
                        results_train = np.expand_dims(results_train,
axis=1)
                        results_train = np.apply_along_axis(lambda x:
inverse_map[x[0]], axis=1, arr=results_train)
                        # create_output_pred(results, results_train)

                        create_output_pred(results_train, train_f,
TRAIN_PRED_FILE)
                        print('In train phase, average loss: {:.3f} |
average accuracy: {:.3f}%'.\
                            format(avg_loss_train,
avg_acc_train*100))

                if args.test_set is not None:
                        test_batch, num_cls, _ =
read_data(args.test_set, dim=args.dim, label_enum=label_enum)
                        test_batch[0], _ =
normalize_and_reshape(test_batch[0], dim=args.dim, mean_tr=mean_tr)
```

```
                        avg_loss_test, avg_acc_test, results_test =
predict(sess, network, test_batch, args.bsize)
                        # convert results back to the original labels
                        inverse_map = dict(zip(np.arange(num_cls),
label_enum))
                        results_test = np.expand_dims(results_test,
axis=1)
                        results_test = np.apply_along_axis(lambda x:
inverse_map[x[0]], axis=1, arr=results_test)
                        # create_output_pred(results, results_train)

                        create_output_pred(results_test, test_f,
TEST_PRED_FILE)
                        print('In test phase, average loss: {:.3f} |
average accuracy: {:.3f}%'.\
                            format(avg_loss_test, avg_acc_test*100))

"""##Run Predict and Display output##"""

predict_model()

# !head "$VALID_PRED_FILE"

# !head "$TRAIN_PRED_FILE"

"""## Check Test score on CASP13 ##"""

# Arguments for prediction PSSP dataset
pred_args = ("--bsize 1024 --test_set " + TEST_FILE +
                         " --model ./saved_model/model.ckpt --dim " +
             str(HEIGHT) + " " + str(WIDTH) + " " +
str(DEPTH)).split()

predict_model()

# !head "$TEST_PRED_FILE"
```

B Appendix

Data Preprocessing

The following two programs are used in order to extract the ProtBERT embeddings (1st program) and save them into matlab files (2nd program). Both programs can be found at [https://gitlab.com/schatz06/pssp/-/tree/master/data_preprocessing]. Matlab files for the CB513 are already uploaded into the GitLab repository.

```
# -*- coding: utf-8 -*-
"""extract_protbert_embeddings.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1KhN4KVU3uwty5IOn5tOmnfQwrRsE
7sIi

Dataset configuration
"""

dataset = "CASP13_3class.txt"

"""Install and import embedders"""

!pip3 install -U pip > /dev/null
!pip3 install -U "bio-embeddings[all] @
git+https://github.com/sacdallago/bio_embeddings.git" > /dev/null

from bio_embeddings.embed import prottrans_bert_bfd_embedder

embedder_protbert=
prottrans_bert_bfd_embedder.ProtTransBertBFDEmbedder()

"""Import packages"""

import numpy as np
from numpy import asarray
from numpy import save
from tqdm import tqdm
import os

"""Generic method to extract ProtBert Embeddings"""

def extract_protBert(lines,path,embedder_protbert):
  count = 1 # current line
  target_name = 1 # target line that has the name of the protein
  target_primary = 2 # target line that has the primary structure
  for line in tqdm(lines):
    if count == target_name:
      line = line.replace("\n","") # remove the newline character in
the end of the string
```

```
      line = line.replace(" ","")
      line = line.replace(">","")
      path_to_save = path + '/' + line
      target_name +=3
    if count == target_primary:
      line = line.replace("\n","") # remove the newline character in
the end of the string
      embedding_protbert = embedder_protbert.embed(line) # extract
the embeddings
      save(path_to_save,embedding_protbert) # save the embedding in a
new txt in a directory
      target_primary +=3
    count+=1

"""Create new directories to save the embeddings """

filename = dataset.replace(".txt","")
path = os.getcwd() # get the current directory
new_directory = path +'/'+filename +'_ProtBert'
os.mkdir(new_directory)

"""Open files/ Create pointers"""

pointer = open(dataset,"r")

"""Read lines"""

lines = pointer.readlines()

"""Extract ProtBert embeddings"""

extract_protBert(lines,new_directory,embedder_protbert)

"""Zip folders"""

zipped_file = new_directory + '.zip'
!zip -r "$zipped_file" "$new_directory"
```

```
import sys
import numpy as np
import os
import os.path
from os import path
import hdf5storage

####################### FILES AND FOLDERS #######
folder = sys.argv[1] + '/content/' + sys.argv[1] + '/'
fold_file = sys.argv[2]
output_file = sys.argv[3]
# print(folder)
# print(fold_file)
# print(output_file)
train = open(fold_file,"r")
################## MAIN PROGRAM ###########################
target_name = 1 # target line that has the name of the protein
```

```
target_secondary = 3 # target line that has the primary structure
count = 1
new = 0
while True:
    # Get next line from file
    if count % 100 == 0:
        print(count)
    line = train.readline()
    if not line:
        break
    if count == target_name:
        line = line.replace("\n", "")  # remove the newline
character in the end of the string
        line = line.replace(" ", "")
        line = line.replace(">", "")
        name = folder + line + ".npy"
        target_name += 3
        # print(name)
        if path.exists(name):
            embedding = np.load(name)
            a = np.array(embedding)
            #a = a.flatten()
            if count == 1:
                new = a
            else:
                new = np.vstack([new, a])
        else:
            print("Protein not exists: ",name)
    if count == target_secondary:
        line = line.replace("\n", "")  # remove the newline
character in the end of the string
        temp = np.zeros((len(line)), dtype=np.short)
        for j in range(0, len(line)):
            if line[j] == 'C':
                temp[j] = 0
            if line[j] == 'H':
                temp[j] = 1
            if line[j] == 'E':
                temp[j] = 2
        if count == 3:
            y = temp
        else:
            y = np.append(y, temp, axis=0)
        target_secondary += 3
        # scipy.io.savemat(f, {line:arr})
    count += 1
hdf5storage.savemat(output_file, {'x': new})
y = y.reshape(-1,1)
#print(y.shape)
hdf5storage.savemat(output_file, {'y': y})
```

C Appendix

Ensembles Program

This Python program was used to combine the results from multiple trained models using the ensembles method. It was provided by Dionysiou [35].

```
from numpy import *
import string as string
import sys


class Ensembles:
    def run(filenames, windowSize, ensemble, outPred, outSOV,
outWeka):
        f = open(outPred, "w")
        files = open(filenames, "r").readlines()
        files = [w.replace('\n', '') for w in files]
        files = [open(i, "r") for i in files]
        i = 0
        LABELS = ['C', 'E', 'H', '!']
        if ensemble == 1:
            for rows in zip(*files):
                if i == 3:
                    for j in range(0,
len(rows[0].translate(str.maketrans('', '', string.whitespace))),
1):
                        count = [0, 0, 0, 0]
                        for k in range(0, len(rows), 1):
                            if rows[k][j] == 'C':
                                count[0] += 1
                            elif rows[k][j] == 'E':
                                count[1] += 1
                            elif rows[k][j] == 'H':
                                count[2] += 1
                            else:
                                count[3] += 1
                        f.write(LABELS[argmax(count)])
                    f.write('\n')
                    i = 0
                else:
                    f.write(rows[0])
                    i += 1
            f.close()
        else:
            print('ERROR!!! Invalid ensemble option.')

        # count accuracy
        f = open(outPred, "r")
        lines = f.readlines()
        f.close()
        count = 0
        countall = 0
        for i in range(0, len(lines), 4):
            for j in range(0, len(lines[i +
2].translate(str.maketrans('', '', string.whitespace))), 1):
                if lines[i + 2][j] == lines[i + 3][j]:
                    count += 1
                countall += 1

        print('Accuracy: ' + str(float(count) / float(countall) *
100) + '%')

        # Confusion Matrix
        countHH = 0
        countHE = 0
        countHC = 0
        countEH = 0
        countEE = 0
        countEC = 0
        countCH = 0
        countCE = 0
        countCC = 0
        countH = 0
        countE = 0
        countC = 0
        countHp = 0
        countEp = 0
        countCp = 0
        for i in range(0, len(lines), 4):
            for j in range(0, len(lines[i +
2].translate(str.maketrans('', '', string.whitespace))), 1):
                if lines[i + 2][j] == 'H' and lines[i + 3][j] ==
'H':
                    countHH += 1
                elif lines[i + 2][j] == 'H' and lines[i + 3][j] ==
'E':
                    countHE += 1
                elif lines[i + 2][j] == 'H' and lines[i + 3][j] ==
'C':
                    countHC += 1
                elif lines[i + 2][j] == 'E' and lines[i + 3][j] ==
'H':
                    countEH += 1
                elif lines[i + 2][j] == 'E' and lines[i + 3][j] ==
'E':
                    countEE += 1
                elif lines[i + 2][j] == 'E' and lines[i + 3][j] ==
'C':
                    countEC += 1
                elif lines[i + 2][j] == 'C' and lines[i + 3][j] ==
'H':
                    countCH += 1
                elif lines[i + 2][j] == 'C' and lines[i + 3][j] ==
'E':
                    countCE += 1
                elif lines[i + 2][j] == 'C' and lines[i + 3][j] ==
'C':
                    countCC += 1

                '''if lines[i + 2][j] == 'H':
                    countH += 1
                elif lines[i + 2][j] == 'E':
                    countE += 1
```

```
                 elif lines[i + 2][j] == 'C':
                     countC += 1

                 if lines[i + 3][j] == 'H':
                     countHp += 1
                 elif lines[i + 3][j] == 'E':
                     countEp += 1
                 elif lines[i + 3][j] == 'C':
                     countCp += 1'''

         print('\n\t\tCONFUSION MATRIX\n')
         print('{0:10}{1:10}{2:10}{3:10}'.format(' ', 'H', 'E', 'C'))
         print('{0:1}{1:10d}{2:10d}{3:10d}'.format('H', countHH,
countHE, countHC))
         print('{0:1}{1:10d}{2:10d}{3:10d}'.format('E', countEH,
countEE, countEC))
         print('{0:1}{1:10d}{2:10d}{3:10d}'.format('C', countCH,
countCE, countCC))

         # SOV input file
         # f = open(outPred, "r")
         f1 = open(outSOV, "w")
         # lines = f.readlines()
         # f.close()

         for i in range(0, len(lines), 4):
             f1.write('>OSEQ\n')
             f1.write(lines[i + 2])
             f1.write('>PSEQ\n')
             f1.write(lines[i + 3])
             f1.write('>AA\n')
             f1.write(lines[i + 1])
         f1.close()

         # weka input file
         f1 = open(outWeka, "w")
         f1.write('@RELATION secondary_structure\n\n')
         for i in range(0, windowSize * 2 - 1, 1):
             f1.write('@ATTRIBUTE aminoacid' + str(i) + '
{C,E,H,0.0}\n')
         f1.write('@ATTRIBUTE output {C,E,H}\n')
         f1.write('\n@DATA\n')

         leadingzeros = zeros((1, (windowSize - 1)))
         for i in range(3, len(lines), 4):
             line = leadingzeros
             line = append(line, list(lines[i].rstrip()))
             line = append(line, leadingzeros)
             for j in range(0, len(lines[i].rstrip()), 1):
                 for k in range(0, windowSize * 2 - 1, 1):
                     f1.write(str(line[j + k]) + ',')
                 f1.write(lines[i - 1].rstrip()[j] + '\n')

         f1.close()

     files = sys.argv[1].replace(',', '')
     run(files, int(sys.argv[2].replace(',', '')),
int(sys.argv[3].replace(',', '')), sys.argv[4].replace(',', ''),
         sys.argv[5].replace(',', ''), sys.argv[6])
     # print('\nEnd of ensembles script\n')
```

D Appendix

External Rules Program

This Python program was used to apply the external rules filtering. It was provided by Dionysiou [35].

```
import sys

class externalRules:
    def applyRules(filename, outSOV, outPred):
        f = open(filename, "r")
        lines = f.readlines()
        f.close()
        f = open(outSOV, "w")
        f1 = open(outPred, "w")

        for i in range(0, len(lines), 4):
            f1.write(lines[i])
            f1.write(lines[i + 1])
            f1.write(lines[i + 2])
            f.write(">OSEQ\n")
            f.write(lines[i + 2])
            f.write(">PSEQ\n")
            j = 0
            lines[i + 3] = list(lines[i + 3].translate({ord(c):''
for c in ' \n\t\r'}))
            # print(len(lines[i + 3]))
            while j < len(lines[i + 3]):
                if len(lines[i + 3]) - j >= 4:
                    if lines[i + 3][j] == 'H' and lines[i + 3][j +
1] == 'E' and lines[i + 3][j + 2] == 'E' and \
                            lines[i + 3][j + 3] == 'H':
                        lines[i + 3][j] = 'H'
                        lines[i + 3][j + 1] = 'H'
                        lines[i + 3][j + 2] = 'H'
                        lines[i + 3][j + 3] = 'H'
                        j += 4
                        continue
                    if lines[i + 3][j] != 'H' and lines[i + 3][j +
1] == 'H' and lines[i + 3][j + 2] == 'H' and \
                            lines[i + 3][j + 3] != 'H':
                        lines[i + 3][j + 1] = 'C'
                        lines[i + 3][j + 2] = 'C'
                        j += 4
                        continue
                if len(lines[i + 3]) - j >= 3:
                    if lines[i + 3][j] == 'H' and lines[i + 3][j +
1] == 'E' and lines[i + 3][j + 2] == 'H':
                        lines[i + 3][j + 1] = 'H'
                        j += 3
                        continue
                j += 1

            if lines[i + 3][0] == 'E' and lines[i + 3][1] != 'E':
                f.write("C")
                f1.write("C")
            elif lines[i + 3][0] == 'H' and lines[i + 3][1] != 'H':
                f.write("C")
                f1.write("C")
            else:
                f.write(lines[i + 3][0])
                f1.write(lines[i + 3][0])

            for j in range(1, len(lines[i + 3]) - 1):
                if lines[i + 3][j - 1] != 'E' and lines[i + 3][j] ==
'E' and lines[i + 3][j + 1] != 'E':
                    f.write("C")
                    f1.write("C")
                    continue
                elif lines[i + 3][j - 1] != 'H' and lines[i + 3][j]
== 'H' and lines[i + 3][j + 1] != 'H':
                    f.write("C")
                    f1.write("C")
                    continue
                f.write(lines[i + 3][j])
                f1.write(lines[i + 3][j])

            if lines[i + 3][len(lines[i + 3]) - 1] == 'E' and
lines[i + 3][len(lines[i + 3]) - 2] != 'E':
                f.write("C")
                f1.write("C")
            elif lines[i + 3][len(lines[i + 3]) - 1] == 'H' and
lines[i + 3][len(lines[i + 3]) - 2] != 'H':
                f.write("C")
                f1.write("C")
            else:
                f.write(lines[i + 3][len(lines[i + 3]) - 1])
                f1.write(lines[i + 3][len(lines[i + 3]) - 1])

            f.write('\n')
            f1.write('\n')
            f.write(">AA\n")
            f.write(lines[i + 1])

    applyRules(sys.argv[1].replace(',', ''),
sys.argv[2].replace(',', ''), sys.argv[3])
    # print('End of external rules script\n')
```

E Appendix

SOV Calculation

To calculate the SOV score the two following C programs were used. Both were provided by Dionysiou [35].

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* argv[]){
    FILE *fp=fopen(argv[1], "r");
    FILE *out;
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    fclose(fopen("resultSOV.txt","w"));

    if (fp == NULL)
        exit(0);
    system("gcc ./q3_sov_scripts/sov.c -o ./q3_sov_scripts/sov -
lm");
    while ((read = getline(&line, &len, fp)) != -1) {
        out=fopen("SOVinput.txt", "w");
        if (out == NULL)
            exit(0);
        fprintf(out,"%s", line);
        getline(&line, &len, fp);
        fprintf(out,"%s", line);
        getline(&line, &len, fp);
        fprintf(out,"%s", line);
        getline(&line, &len, fp);
        fprintf(out,"%s", line);
        getline(&line, &len, fp);
        fprintf(out,"%s", line);
        getline(&line, &len, fp);
        fprintf(out,"%s", line);
        fclose(out);

        system("./q3_sov_scripts/sov SOVinput.txt >>resultSOV.txt");
    }

    free(line);
    fclose(fp);
    return 0;
}
```

```
/*-----------------------------------------------------------
/
/   Program:      sov.c
/
/   Secondary structure prediction accuracy evaluation
/
/   SOV (Segment OVerlap) measure
/
/   Copyright by Adam Zemla (11/16/1996)
/   Email: adamz@llnl.gov
/
/------------------------------------------------------------
/
/   Compile:      cc sov.c -o sov -lm
/
/------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAXRES            5000

typedef struct {
  int     input;
  int     order;
  int     q3_what;
  int     sov_what;
  int     sov_method;
  float   sov_delta;
  float   sov_delta_s;
  int     sov_out;
  char    fname[100];
} parameters;

char *letter_AA="ARNDCQEGHILKMFPSTWYV-?X";             /* 23 chars */

void default_parameters(parameters *);
int read_aa_osec_psec(char[MAXRES], char[MAXRES], char[MAXRES],
                      parameters *, char*);
float sov(int, char[MAXRES], char[MAXRES], parameters *);
float q3(int, char[MAXRES], char[MAXRES], parameters *);
int check_aa(char, char*, int);

int main(int argc, char *argv[])
{
  int i, n_aa, sov_method;
  char c, aa[MAXRES], osec[MAXRES], psec[MAXRES];
  parameters pdata;
  float out0, out1, out2, out3;

  if(argc<2){
    printf(" Usage: sov <input_data>\n");
    printf(" HELP:  sov -h\n");
    exit(0);
  }
  if(!strncmp(argv[1],"-h\0",2) ||
     !strncmp(argv[1],"help\0",5) ||
     !strncmp(argv[1],"-help\0",6)) {
    system("more ./README.sov");
    printf("\n");
    exit(0);
  }

  default_parameters(&pdata);

  strcpy(pdata.fname,argv[1]);

  n_aa=read_aa_osec_psec(aa,osec,psec,&pdata,letter_AA);

  if(pdata.input==1) {
    n_aa=read_aa_osec_psec(aa,osec,psec,&pdata,letter_AA);
  }

  if(pdata.order==1) {
    for(i=0;i<n_aa;i++) {
      c=osec[i];
      osec[i]=psec[i];
      psec[i]=c;
    }
  }

  if(n_aa<=0) {
    printf("\n ERROR! There is no 'AA OSEC PSEC' data in submited 
prediction.");
    printf("\n        The submission should contain an observed and 
predicted");
    printf("\n        secondary structure in COLUMN format.\n");
    exit(0);
  }

  printf("\n\n SECONDARY STRUCTURE PREDICTION");
  printf("\n NUMBER OF RESIDUES PREDICTED: LENGTH = %d",n_aa);
  printf("\n AA  OSEC  PSEC     NUM");
  for(i=0;i<n_aa;i++) {
    printf("\n  %1c   %1c     %1c     
%4d",aa[i],osec[i],psec[i],i+1);
  }
  printf("\n -----------------------\n");
  printf("\n SECONDARY STRUCTURE PREDICTION ACCURACY EVALUATION. 
N_AA = %4d\n",n_aa);
  if(pdata.sov_out>=1) {
    printf("\n SOV parameters:   DELTA = %5.2f   DELTA-S = %5.2f\n",
              pdata.sov_delta,
              pdata.sov_delta_s);
  }

  printf("\n                                  ALL    HELIX   STRAND   
COIL\n");

  pdata.q3_what=0;
  out0=q3(n_aa,osec,psec,&pdata);
  pdata.q3_what=1;
  out1=q3(n_aa,osec,psec,&pdata);
  pdata.q3_what=2;
  out2=q3(n_aa,osec,psec,&pdata);
  pdata.q3_what=3;
  out3=q3(n_aa,osec,psec,&pdata);
  printf("\n Q3                        :   %6.1f   %6.1f   %6.1f   
%6.1f",
out0*100.0,out1*100.0,out2*100.0,out3*100.0);
  printf("\n");

  sov_method=pdata.sov_method;

  if(sov_method!=0) pdata.sov_method=1;

  if(pdata.sov_method==1) {
    pdata.sov_what=0;
    out0=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=1;
    out1=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=2;
    out2=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=3;
    out3=sov(n_aa,osec,psec,&pdata);
    printf("\n SOV                        :   %6.1f   %6.1f   %6.1f
%6.1f",

out0*100.0,out1*100.0,out2*100.0,out3*100.0);
    printf("\n");
  }

  if(sov_method!=1) pdata.sov_method=0;

  if(pdata.sov_method==0) {
    pdata.sov_delta=1.0;

    pdata.sov_what=0;
    out0=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=1;
    out1=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=2;
    out2=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=3;
    out3=sov(n_aa,osec,psec,&pdata);
    printf("\n SOV (1994 JMB. [delta=50]) :   %6.1f   %6.1f   %6.1f
%6.1f",

out0*100.0,out1*100.0,out2*100.0,out3*100.0);

    pdata.sov_delta=0.0;

    pdata.sov_what=0;
    out0=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=1;
    out1=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=2;
    out2=sov(n_aa,osec,psec,&pdata);
    pdata.sov_what=3;
    out3=sov(n_aa,osec,psec,&pdata);
    printf("\n SOV (1994 JMB. [delta=0])  :   %6.1f   %6.1f   %6.1f
%6.1f",

out0*100.0,out1*100.0,out2*100.0,out3*100.0);

    printf("\n");
  }
  printf("\n ----------------------\n");

  exit(0);
}

/*------------------------------------------------------------
/
/   check_aa - checks an amino acid
/
/------------------------------------------------------------*/
int check_aa(char token, char* letter, int n)
{
  int i;

  for(i=0;i<n;i++) {
    if(letter[i]==token)
      return i;
  }
  return n;
}

/*------------------------------------------------------------
/
/   read_aa_osec_psec - read secondary structure segments file
/
/------------------------------------------------------------*/
int read_aa_osec_psec(char aa[MAXRES], char sss1[MAXRES],
                       char sss2[MAXRES], parameters *pdata, char*
letter)
{
  int i, j, n_aa, n_aa_1, n_aa_2, n_aa_3, f_seq, alt_c, alt_e, alt_h;
  float x;
  char
line[MAXRES],keyword[MAXRES],first[MAXRES],second[MAXRES],third[MAXRE
S],junk[MAXRES];
  FILE *fp;

  alt_c=0;
  alt_e=0;
  alt_h=0;

  if((fp = fopen(pdata->fname,"r"))==NULL) {
    printf("\n# error opening file %s for read\n\n",pdata->fname);
    exit(0);
  }

  f_seq=0;
  pdata->input=0;
  n_aa=0;
  n_aa_1=0;
  n_aa_2=0;
  n_aa_3=0;

  while (fgets(line, MAXRES, fp) != NULL) {
    strcpy(keyword,"   ");
    strcpy(first,"   ");
    strcpy(second,"   ");
    strcpy(third,"   ");
    strcpy(junk,"   ");
    i=0;
    j=0;
    while(line[i] == ' ' && line[i] != '\n' && line[i] != '\0' &&
i<MAXRES) i++;
    if(i<MAXRES) {
      j=i;
      while(line[i] != ' ' && line[i] != '\n' && line[i] != '\0' &&
i<MAXRES) i++;
    }
    j=i-j;
    if(j<MAXRES && j>0) {
      sscanf(line,"%s",keyword);
    }
    if(!strncmp(keyword,"#",1)) {}
    else if(!strncmp(keyword,"-----",5)) {}
    else if(!strncmp(keyword,"NUMBER\0",7)) {}
    else if(!strncmp(keyword,"SECONDARY\0",10)) {}
    else if(!strncmp(keyword,"END\0",4) && f_seq==0) {
      fclose(fp);
      return n_aa;
    }
    else if(!strncmp(keyword,"AA-OSEC-PSEC\0",13)) {
      printf("%s", line);
      sscanf(line,"%s %s",keyword,first);
      strcpy(pdata->fname,first);
      pdata->input=1;
    }
    else if(line[0] == '\n' || !strncmp(keyword,"   \0",4)) {}
    else if(!strncmp(keyword,"AA\0",3) && f_seq==0) {
      sscanf(line,"%s %s %s",keyword,first,second);
      if(!strncmp(keyword,"AA\0",3) &&
         !strncmp(first,"PSEC\0",5) && !strncmp(second,"OSEC\0",5)) {
        pdata->order=1;
      }
    }
    else if(!strncmp(keyword,"SOV-DELTA\0",10)) {
      printf("%s", line);
      sscanf(line,"%s %f",keyword,&x);
      pdata->sov_delta=x;
    }
    else if(!strncmp(keyword,"SOV-DELTA-S\0",12)) {
      printf("%s", line);
      sscanf(line,"%s %f",keyword,&x);
      pdata->sov_delta_s=x;
    }
    else if(!strncmp(keyword,"SOV-METHOD\0",9)) {
      printf("%s", line);
      sscanf(line,"%s %d",keyword,&i);
      pdata->sov_method=i;
    }
    else if(!strncmp(keyword,"SOV-OUTPUT\0",9)) {
      printf("%s", line);
      sscanf(line,"%s %d",keyword,&i);
      pdata->sov_out=i;
    }
    else if(line[0]=='>') {
      printf("%s", line);
      if(f_seq<2) n_aa=0;
      f_seq++;
    }
    else if(f_seq==0) {
      if(j>1) {
        if(!strncmp(keyword,"SSP\0",4)) {
          sscanf(line,"%s %s %s %s
%s",keyword,junk,first,second,third);
        }
        else {
          printf("\n ERROR! (line: %d) Check COLUMN format of your
prediction!\n",n_aa+1);
          fclose(fp);
          exit(0);
        }
      }
      else {
        sscanf(line,"%s %s %s",first,second,third);
      }
      aa[n_aa]=first[0];
      sss1[n_aa]=second[0];
      sss2[n_aa]=third[0];
      if(check_aa(aa[n_aa],letter,23)==23) {
        printf("\n# ERROR!\n%s",line);
        printf("\n# ERROR! (line: %d) Check amino acid code
%c\n",n_aa+1,aa[n_aa]);
        fclose(fp);
        exit(0);
      }
      if(sss1[n_aa]==' ' || sss2[n_aa]==' ') {
        printf("\n# ERROR!\n%s",line);
        printf("\n# ERROR! (line: %d) Check secondary structure
code\n",n_aa+1);
        fclose(fp);
        exit(0);
      }
      if(sss1[n_aa]=='L' || sss1[n_aa]=='T' || sss1[n_aa]=='S') {
        if(alt_c==0) {
          printf("# WARNING! (line: %d) The '%c' characters are
interpreted as 'C' (coil)\n",n_aa+1,sss1[n_aa]);
          alt_c=1;
        }
        sss1[n_aa]='C';
      }
      if(sss1[n_aa]=='B') {
        if(alt_e==0) {
          printf("# WARNING! (line: %d) The '%c' characters are
interpreted as 'E' (strand)\n",n_aa+1,sss1[n_aa]);
          alt_e=1;
        }
        sss1[n_aa]='E';
      }
      if(sss1[n_aa]=='G' || sss1[n_aa]=='I') {
        if(alt_h==0) {
          printf("# WARNING! (line: %d) The '%c' characters are
interpreted as 'H' (helix)\n",n_aa+1,sss1[n_aa]);
          alt_h=1;
        }
        sss1[n_aa]='H';
      }
      if(sss2[n_aa]=='L' || sss2[n_aa]=='T' || sss2[n_aa]=='S') {
        if(alt_c==0) {
          printf("# WARNING! (line: %d) The '%c' characters are
interpreted as 'C' (coil)\n",n_aa+1,sss2[n_aa]);
```

```
          alt_c=1;
        }
        sss2[n_aa]='C';
      }
      if(sss2[n_aa]=='B') {
        if(alt_e==0) {
          printf("# WARNING! (line: %d) The '%c' characters are
interpreted as 'E' (strand)\n",n_aa+1,sss2[n_aa]);
          alt_e=1;
        }
        sss2[n_aa]='E';
      }
      if(sss2[n_aa]=='G' || sss2[n_aa]=='I') {
        if(alt_h==0) {
          printf("# WARNING! (line: %d) The '%c' characters are
interpreted as 'H' (helix)\n",n_aa+1,sss2[n_aa]);
          alt_h=1;
        }
        sss2[n_aa]='H';
      }
      if(sss1[n_aa]!='C' && sss1[n_aa]!='E' && sss1[n_aa]!='H') {
        printf("\n# ERROR!\n%s",line);
        printf("\n# ERROR! (line: %d) Check secondary structure code
%c\n",n_aa+1,sss1[n_aa]);
        fclose(fp);
        exit(0);
      }
      if(sss2[n_aa]!='C' && sss2[n_aa]!='E' && sss2[n_aa]!='H') {
        printf("\n# ERROR!\n%s",line);
        printf("\n# ERROR! (line: %d) Check secondary structure code
%c\n",n_aa+1,sss2[n_aa]);
        fclose(fp);
        exit(0);
      }
      n_aa++;
      if(n_aa>=MAXRES) {
        printf("\n# ERROR! Check number of amino acid lines. (MAX =
%d)\n\n",MAXRES);
        fclose(fp);
        exit(0);
      }
    }
    else if(f_seq==1) {
      i=0;
      while(line[i] != '\n') {
        if(line[i] != ' ' && line[i] != '\t' && line[i] != '\0' &&
           line[i] != '\a' && line[i] != '\b' && line[i] != '\f' &&
           line[i] != '\r' && line[i] != '\v' && i<MAXRES) {
          aa[n_aa]='X';
          sss1[n_aa]=line[i];
          if(sss1[n_aa]=='L' || sss1[n_aa]=='T' || sss1[n_aa]=='S') {
            if(alt_c==0) {
              printf("# WARNING! The '%c' characters are interpreted
as 'C' (coil)\n",sss1[n_aa]);
              alt_c=1;
            }
            sss1[n_aa]='C';
          }
          if(sss1[n_aa]=='B') {
            if(alt_e==0) {
              printf("# WARNING! The '%c' characters are interpreted
as 'E' (strand)\n",sss1[n_aa]);
              alt_e=1;
            }
            sss1[n_aa]='E';
          }
          if(sss1[n_aa]=='G' || sss1[n_aa]=='I') {
            if(alt_h==0) {
              printf("# WARNING! The '%c' characters are interpreted
as 'H' (helix)\n",sss1[n_aa]);
              alt_h=1;
            }
            sss1[n_aa]='H';
          }
          if(sss1[n_aa]!='C' && sss1[n_aa]!='E' && sss1[n_aa]!='H') {
            printf("\n# ERROR!\n%s",line);
            printf("\n# ERROR! Check secondary structure code:
%c\n",sss1[n_aa]);
            fclose(fp);
            exit(0);
          }
          n_aa++;
          if(n_aa>=MAXRES) {
            printf("\n# ERROR! Check number of residues. (MAX =
%d)\n\n",MAXRES);
            fclose(fp);
            exit(0);
          }
        }
        i++;
      }
      n_aa_1=n_aa;
    }
    else if(f_seq==2) {
      i=0;
      while(line[i] != '\n') {
        if(line[i] != ' ' && line[i] != '\t' && line[i] != '\0' &&
           line[i] != '\a' && line[i] != '\b' && line[i] != '\f' &&
           line[i] != '\r' && line[i] != '\v' && i<MAXRES) {
          aa[n_aa]='X';
          sss2[n_aa]=line[i];
          if(sss2[n_aa]=='L' || sss2[n_aa]=='T' || sss2[n_aa]=='S') {
            if(alt_c==0) {
              printf("# WARNING! The '%c' characters are interpreted
as 'C' (coil)\n",sss2[n_aa]);
              alt_c=1;
            }
            sss2[n_aa]='C';
          }
          if(sss2[n_aa]=='B') {
            if(alt_e==0) {
              printf("# WARNING! The '%c' characters are interpreted
as 'E' (strand)\n",sss2[n_aa]);
              alt_e=1;
            }
            sss2[n_aa]='E';
          }
          if(sss2[n_aa]=='G' || sss2[n_aa]=='I') {
            if(alt_h==0) {
              printf("# WARNING! The '%c' characters are interpreted
as 'H' (helix)\n",sss2[n_aa]);
              alt_h=1;
            }
            sss2[n_aa]='H';
          }
          if(sss2[n_aa]!='C' && sss2[n_aa]!='E' && sss2[n_aa]!='H') {
            printf("\n# ERROR!\n%s",line);
            printf("\n# ERROR! Check secondary structure code:
%c\n",sss2[n_aa]);
            fclose(fp);
            exit(0);
          }
          n_aa++;
          if(n_aa>=MAXRES) {
            printf("\n# ERROR! Check number of residues. (MAX =
%d)\n\n",MAXRES);
            fclose(fp);
            exit(0);
          }
        }
        i++;
      }
      n_aa_2=n_aa;
    }
    else if(f_seq==3) {
      i=0;
      while(line[i] != '\n') {
        if(line[i] != ' ' && line[i] != '\t' && line[i] != '\0' &&
           line[i] != '\a' && line[i] != '\b' && line[i] != '\f' &&
           line[i] != '\r' && line[i] != '\v' && i<MAXRES) {
          aa[n_aa_3]=line[i];
          if(check_aa(aa[n_aa_3],letter,23)==23) {
            printf("\n# ERROR!\n%s",line);
            printf("\n# ERROR! (N_res: %d) Check amino acid code
%c\n",n_aa_3+1,aa[n_aa_3]);
            fclose(fp);
            exit(0);
          }
          n_aa_3++;
          if(n_aa_3>=MAXRES) {
            printf("\n# ERROR! Check number of residues. (MAX =
%d)\n\n",MAXRES);
            fclose(fp);
            exit(0);
          }
        }
        i++;
      }
    }
  }
  if(n_aa_1!=n_aa_2) {
    printf("\n# ERROR! Check format of your submission.");
    printf("\n#        Different length of observed and predicted
structures.\n");
    fclose(fp);
    exit(0);
  }
  return n_aa;
}
```

```
/*-------------------------------------------------------------
/
/  default_parameters - default parameters for SOV program
/
/-------------------------------------------------------------*/
void default_parameters(parameters *pdata)
{
  pdata->input=0;
  pdata->order=0;
  pdata->sov_method=1; // 0 - SOV definition (1994 JMB.) , 1 - SOV
definition (1999 Proteins)
  pdata->sov_delta=1.0;
  pdata->sov_delta_s=0.5;
  pdata->sov_out=0;

  return;
}

/*-----------------------------------------------------------
/
/    sov - evaluate SSp by the Segment OVerlap quantity (SOV)
/                Input: secondary structure segments
/
/-------------------------------------------------------------*/
float sov(int n_aa, char sss1[MAXRES], char sss2[MAXRES], parameters
*pdata)
{
  int i, k, length1, length2, beg_s1, end_s1, beg_s2, end_s2;
  int j1, j2, k1, k2, minov, maxov, d, d1, d2, n, multiple;
  char s1, s2, sse[3];
  float out;
  double s, x;

  sse[0]='#';
  sse[1]='#';
  sse[2]='#';

  if(pdata->sov_what==0) {
    sse[0]='H';
    sse[1]='E';
    sse[2]='C';
  }
  if(pdata->sov_what==1) {
    sse[0]='H';
    sse[1]='H';
    sse[2]='H';
  }
  if(pdata->sov_what==2) {
    sse[0]='E';
    sse[1]='E';
    sse[2]='E';
  }
  if(pdata->sov_what==3) {
    sse[0]='C';
    sse[1]='C';
    sse[2]='C';
  }
  n=0;
  for(i=0;i<n_aa;i++) {
    s1=sss1[i];
    if(s1==sse[0] || s1==sse[1] || s1==sse[2]) {
      n++;
    }
  }
  out=0.0;
  s=0.0;
  length1=0;
  length2=0;
  i=0;
  while(i<n_aa) {
    beg_s1=i;
    s1=sss1[i];
    while(sss1[i]==s1 && i<n_aa) {
      i++;
    }
    end_s1=i-1;
    length1=end_s1-beg_s1+1;
    multiple=0;
    k=0;
    while(k<n_aa) {
      beg_s2=k;
      s2=sss2[k];
      while(sss2[k]==s2 && k<n_aa) {
        k++;
      }
      end_s2=k-1;
      length2=end_s2-beg_s2+1;
      if(s1==sse[0] || s1==sse[1] || s1==sse[2]) {
        if(s1==s2 && end_s2>=beg_s1 && beg_s2<=end_s1) {
          if(multiple>0 && pdata->sov_method==1) {
            n=n+length1;
          }
          multiple++;
          if(beg_s1>beg_s2) {
            j1=beg_s1;
            j2=beg_s2;
          }
          else {
            j1=beg_s2;
            j2=beg_s1;
          }
          if(end_s1<end_s2) {
            k1=end_s1;
            k2=end_s2;
          }
          else {
            k1=end_s2;
            k2=end_s1;
          }
          minov=k1-j1+1;
          maxov=k2-j2+1;
          d1=floor(length1*pdata->sov_delta_s);
          d2=floor(length2*pdata->sov_delta_s);
          if(d1>d2) d=d2;
          if(d1<=d2 || pdata->sov_method==0) d=d1;
          if(d>minov) {
            d=minov;
          }
          if(d>maxov-minov) {
            d=maxov-minov;
          }
          x=pdata->sov_delta*d;
          x=(minov+x)*length1;
          if(maxov>0) {
            s=s+x/maxov;
          }
          else {
            printf("\n ERROR! minov = %-4d maxov = %-4d length = %-4d
d = %-4d   %4d %4d  %4d %4d",

minov,maxov,length1,d,beg_s1+1,end_s1+1,beg_s2+1,end_s2+1);
          }
          if(pdata->sov_out==2) {
            printf("\n TEST: minov = %-4d maxov = %-4d length = %-4d
d = %-4d   %4d %4d  %4d %4d",

minov,maxov,length1,d,beg_s1+1,end_s1+1,beg_s2+1,end_s2+1);
          }
        }
      }
    }
  }
  if(pdata->sov_out==2) {
    printf("\n TEST: Number of considered residues = %d",n);
  }
  if(n>0) {
    out=s/n;
  }
  else {
    out=1.0;
  }
  return out;
}

/*-------------------------------------------------------------
/
/    Q3 - evaluate SSp by the residues predicted correctly (Q3)
/                Input: secondary structure segments
/
/-------------------------------------------------------------*/
float q3(int n_aa, char sss1[MAXRES], char sss2[MAXRES], parameters
*pdata)
{
  int i, n;
  float out;
  char s, sse[3];

  sse[0]='#';
  sse[1]='#';
  sse[2]='#';

  if(pdata->q3_what==0) {
    sse[0]='H';
    sse[1]='E';
    sse[2]='C';
  }
  if(pdata->q3_what==1) {
    sse[0]='H';
    sse[1]='H';
    sse[2]='H';
  }
  if(pdata->q3_what==2) {
    sse[0]='E';
    sse[1]='E';
    sse[2]='E';
  }
  if(pdata->q3_what==3) {
    sse[0]='C';
    sse[1]='C';
    sse[2]='C';
  }

  n=0;
  out=0.0;
  for(i=0;i<n_aa;i++) {
    s=sss1[i];
    if(s==sse[0] || s==sse[1] || s==sse[2]) {
      n++;
      if(sss1[i]==sss2[i]) {
        out=out + 1.0;
      }
    }
  }
  if(n>0) {
    out=out/n;
  }
  else {
    out=1.0;
  }

  return out;
}
```

F Appendix

Calculation of Q3 accuracy

The following Python program was used to calculate the Q3 accuracy for each class and the overall Q3 accuracy. It was provided by Leontiou [5].

```
import sys
# Execute: python calc_Q3.py <pred_file>
import string
lines = None
labels = ['H', 'E', 'C']
with open(sys.argv[1]) as file:
    lines = file.readlines()
if lines is None: exit(0)
countCor = [0, 0, 0]
countAll = [0, 0, 0]
for l in range(0, len(lines)//4):
    protein_name = lines[4*l]
    primary = lines[4*l + 1]
    secondary = lines[4*l + 2]
    prediction = lines[4*l + 3]
    for s, p in zip(secondary, prediction):
        if s == '\n': continue
        if s == p:
            countCor[labels.index(s)] += 1
        countAll[labels.index(s)] += 1
total = countAll[0] + countAll[1] + countAll[2]
correct = countCor[0] + countCor[1] + countCor[2]
headers = ['Q3_All', 'Q3_C', 'Q3_E', 'Q3_H']
q3 = [(100*correct/total),
      (100*countCor[0]/countAll[0]),
      (100*countCor[1]/countAll[1]),
      (100*countCor[2]/countAll[2])]
print("\n   {0:11}{1:11}{2:11}{3:11}".format(' Q3_ALL', ' Q3_H', '
Q3_E', ' Q3_C'))
print('{0:11.4f}{1:11.4f}{2:11.4f}{3:11.4f}\n'.format(q3[0], q3[1],
q3[2], q3[3]))
```

G Appendix

Data pre-processing for filtering

This python program was used to prepare the datasets for the filtering methods. Initially was used only for the SVM filtering technique but now the same datasets are used to train the decision trees and random forests. It was provided by Dionysiou [35].

```
# Execute: python prepare_SVM_files.py <test_filename>
<train_filename> <WINDOW> <out_test> <out_train>
import sys
EXCL_MIDDLE = False
#open TEST file to read data
with open(sys.argv[1],"r") as testfile:
    lines_test = testfile.readlines()
#open TRAIN file to read dat
with open(sys.argv[2],"r") as trainfile:
    lines_train = trainfile.readlines()
linenum = 1
window = int(sys.argv[3])
leftwindow = int(window/2)
#create train file
with open(sys.argv[5], "w") as svmtrain:
    for line in lines_train:
        if linenum == 5: linenum = 1
        if linenum == 3:
            target_out = line
            # if linenum == 4:
            for i in range(leftwindow):
                zeros = leftwindow - i
                for zer in range(zeros):
                    svmtrain.write("0,")
                for rem in range(i):
                    if line[rem] == "C": svmtrain.write("0,")
                    if line[rem] == "E": svmtrain.write("1,")
                    if line[rem] == "H": svmtrain.write("2,")
                #place right aminos
                for j in range(leftwindow+1):
                    if (EXCL_MIDDLE and j == 0): continue
                    if line[i+j] == "C": svmtrain.write("0,")
                    if line[i+j] == "E": svmtrain.write("1,")
                    if line[i+j] == "H": svmtrain.write("2,")
                #place label at the end
                if target_out[i] == "C": svmtrain.write("0")
                if target_out[i] == "E": svmtrain.write("1")
                if target_out[i] == "H": svmtrain.write("2")
                svmtrain.write("\n")
            #place aminos with no boundary constraints
            for amino in range(leftwindow,len(line)-leftwindow-1):
                for curr in range(-leftwindow,leftwindow+1):
                    if (EXCL_MIDDLE and curr == 0): continue
                    if line[amino+curr] == "C": svmtrain.write("0,")
                    if line[amino+curr] == "E": svmtrain.write("1,")
                    if line[amino+curr] == "H": svmtrain.write("2,")
                #place label
                if target_out[amino] == "C": svmtrain.write("0")
                if target_out[amino] == "E": svmtrain.write("1")
                if target_out[amino] == "H": svmtrain.write("2")
                svmtrain.write("\n")
            #place last aminos with padding
            for i in range(len(line)-leftwindow-1,len(line)-1):
                printed=0
                for left in range(i-leftwindow-1,i):
                    if (EXCL_MIDDLE and left == i-1): continue
                    if line[left] == "C": svmtrain.write("0,")
                    if line[left] == "E": svmtrain.write("1,")
                    if line[left] == "H": svmtrain.write("2,")
                for j in range(i,len(line)-1):
                    if line[j] == "C": svmtrain.write("0,")
                    if line[j] == "E": svmtrain.write("1,")
                    if line[j] == "H": svmtrain.write("2,")
                    printed=printed+1
                zeros = leftwindow-printed
                for z in range(zeros):
                    svmtrain.write("0,")
                # place label
                if target_out[i] == "C": svmtrain.write("0")
                if target_out[i] == "E": svmtrain.write("1")
                if target_out[i] == "H": svmtrain.write("2")
                svmtrain.write("\n")
        linenum += 1
    svmtrain.flush()
linenum=1
#create TEST file
with open(sys.argv[4], "w") as svmtest:
    for line in lines_test:
        if linenum == 5: linenum = 1
        if linenum == 3: target_out = line
        if linenum == 4:
            for i in range(leftwindow):
                zeros = leftwindow - i
                for zer in range(zeros):
                    svmtest.write("0,")
                for rem in range(i):
                    if line[rem] == "C": svmtest.write("0,")
                    if line[rem] == "E": svmtest.write("1,")
                    if line[rem] == "H": svmtest.write("2,")
                #place right aminos
                for j in range(leftwindow+1):
                    if (EXCL_MIDDLE and j == 0): continue
                    if line[i+j] == "C": svmtest.write("0,")
                    if line[i+j] == "E": svmtest.write("1,")
                    if line[i+j] == "H": svmtest.write("2,")
                #place label at the end
                if target_out[i] == "C": svmtest.write("0")
                if target_out[i] == "E": svmtest.write("1")
                if target_out[i] == "H": svmtest.write("2")
                svmtest.write("\n")
            #place aminos with no boundary constraints
            for amino in range(leftwindow,len(line)-leftwindow-1):
                for curr in range(-leftwindow,leftwindow+1):
                    if (EXCL_MIDDLE and curr == 0): continue
                    if line[amino+curr] == "C": svmtest.write("0,")
                    if line[amino+curr] == "E": svmtest.write("1,")
                    if line[amino+curr] == "H": svmtest.write("2,")
                #place label
                if target_out[amino] == "C": svmtest.write("0")
                if target_out[amino] == "E": svmtest.write("1")
                if target_out[amino] == "H": svmtest.write("2")
                svmtest.write("\n")
            #place last aminos with padding
            for i in range(len(line)-leftwindow-1,len(line)-1):
                printed=0
                for left in range(i-leftwindow-1,i):
                    if (EXCL_MIDDLE and left == i-1): continue
                    if line[left] == "C": svmtest.write("0,")
                    if line[left] == "E": svmtest.write("1,")
                    if line[left] == "H": svmtest.write("2,")
                for j in range(i,len(line)-1):
                    if line[j] == "C": svmtest.write("0,")
                    if line[j] == "E": svmtest.write("1,")
                    if line[j] == "H": svmtest.write("2,")
                    printed+=1
                zeros = leftwindow-printed
                for z in range(zeros):
                    svmtest.write("0,")
                # place label
                if target_out[i] == "C": svmtest.write("0")
                if target_out[i] == "E": svmtest.write("1")
                if target_out[i] == "H": svmtest.write("2")
                svmtest.write("\n")
        linenum += 1
    svmtest.flush()
```

H Appendix

Training Filtering Methods

The following program was implemented to train the filtering models and apply the filtering techniques on the output data of the Convolutional Neural Network. It was provided by Leontiou [5].

```
# Execute: python train_SVM.py <test_filename> <train_filename>
<WINDOW> <pred_file> <out_prediction> <out_sov> <filter_opt>
import sys
import string
import numpy as np
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
SEED = 42
np.random.seed(SEED)

def get_balanced_data(X_train, y_train):
    classH = []
    classE = []
    classC = []
    for i,label in enumerate(y_train):
        if label == 0:
            classH.append(i)
        elif label == 1:
            classE.append(i)
        else:
            classC.append(i)
    rows = min(len(classH), len(classE), len(classC))

    # Create a balanced data set
    X_balanced = np.concatenate((X_train[classH][0:rows],
X_train[classE][0:rows], X_train[classC][0:rows]), axis=0)

    y_balanced = np.concatenate((y_train[classH][0:rows],
y_train[classE][0:rows], y_train[classC][0:rows]), axis=0)

    balanced = np.zeros((X_balanced.shape[0],
X_balanced.shape[1]+1), dtype=int)

    balanced[:,-1] = y_balanced
    balanced[:,:-1] = X_balanced
    np.random.shuffle(balanced)
    return balanced[:,:-1], balanced[:,-1]


def create_output_pred(pred, input_f, out_f, outSOV):
    with open(input_f, "r") as pred_file:
        pred_lines = pred_file.readlines()
    pred = pred.astype(int)
    labels = ['C', 'E', 'H']
    counter = 0
    with open(out_f, 'w') as out_file:
        for line in range(0, len(pred_lines)//4):
            protein_name = pred_lines[line*4][0:-1]
            primary_structure = pred_lines[line*4+1][0:-1]
            secondary_structure = pred_lines[line*4+2][0:-1]
            prediction = ""
            for c in secondary_structure:
                prediction = prediction + labels[pred[counter]]
                counter += 1
            out_file.write(protein_name + "\n")
            out_file.write(primary_structure + "\n")
            out_file.write(secondary_structure + "\n")
            out_file.write(prediction + "\n")

    with open(out_f, "r") as out_file:
        lines = out_file.readlines()
    with open(outSOV, "w") as f1:
        for i in range(0, len(lines), 4):
            f1.write('>OSEQ\n')
            f1.write(lines[i + 2])
            f1.write('>PSEQ\n')
            f1.write(lines[i + 3])
            f1.write('>AA\n')
            f1.write(lines[i + 1])

train_dataset = np.loadtxt(sys.argv[1], delimiter=",")
# Subtract one because we dropped the middle column to prevent data
leaks
win=int(sys.argv[3])
X_train = train_dataset[:, 0:win]
y_train = train_dataset[:, [win]]
test_dataset = np.loadtxt(sys.argv[1], delimiter=",")
X_test = test_dataset[:, 0:win]
y_test = test_dataset[:, [win]]
y_train = np.reshape(y_train,len(y_train))
y_test = np.reshape(y_test,len(y_test))
# X_train, y_train = get_balanced_data(X_train, y_train)

print("Training ...")

if (sys.argv[7] == '1'):
    clf = SVC(C=100, decision_function_shape='ovr', kernel='rbf',
random_state=SEED, gamma='scale')
elif (sys.argv[7] == '2'):
    clf = DecisionTreeClassifier(max_depth=20)
elif (sys.argv[7] == '3'):
    clf = RandomForestClassifier(max_depth=25, random_state=SEED,
n_estimators=100)
elif (sys.argv[7] == '0'):
    kernels = ['Polynomial', 'RBF', 'Sigmoid','Linear']
    #A function which returns the corresponding SVC model
    def getClassifier(ktype):
        if ktype == 0:
            # Polynomial kernal
            return SVC(kernel='poly', degree=8, gamma="auto")
        elif ktype == 1:
            # Radial Basis Function kernel
            return SVC(kernel='rbf', gamma="auto")
        elif ktype == 2:
            # Sigmoid kernel
            return SVC(kernel='sigmoid', gamma="auto")
        elif ktype == 3:
            # Linear kernel
            return SVC(kernel='linear', gamma="auto")

    for i in range(1, 4):
        # Train a SVC model using different kernels
        svclassifier = getClassifier(i)
        svclassifier.fit(X_train, y_train)
        # Make prediction
        y_pred = svclassifier.predict(X_test)
        # Evaluate model
        print("Evaluation:", kernels[i], "kernel")
        print(classification_report(y_test, y_pred))

    from sklearn.model_selection import GridSearchCV
    param_grid = {'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01,
0.001],'kernel': ['rbf']}
    grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
    grid.fit(X_train, y_train)
    print(grid.best_estimator_)

    y_pred = grid.predict(X_test)
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    exit(0)
else:
    print('Error! train_SVM.py currently has no such filtering
option.')
    print('Please try again (availiable options: 0-3)')
    exit(0)

# Predict the response for test dataset
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print("THE SCORE: ", clf.score(X_test, y_test))
print("")

# creating a confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix')
print(cm)
print("")

create_output_pred(y_pred, sys.argv[4], sys.argv[5], sys.argv[6])
```

I Appendix

All filtering methods on CB513

Bash script that is used to apply the ensembles and the various filtering techniques and display the results for each fold of the CB513 dataset. It was provided by Leontiou [5].

```
#!/bin/bash
# Author : Panayiotis Leontiou
# Since  : April 2020
# Version: 1.0
# Bugs   : No known bugs

TEST_FOLDER="./CB513_test_pred"
TRAIN_FOLDER="./CB513_train_pred"
CROSS_VAL_FOLDER="./CB513_cross_validation"
WINDOW="15"
SVM_WIN="11"
filterOpt=( "1" "2" "3" )
SCRIPTS="./q3_sov_scripts"

# Check if required scripts exist
declare -a REQUIRED_SCRIPTS=( "calc_Q3.py" "ensembles.py"
"externalRules.py" "prepare_SVM_files.py" "runSOV.c" "sov.c"
"train_SVM.py")
if [ ! -d "$SCRIPTS" ]; then
    echo "Error! $SCRIPTS directory could not be located."
    exit 1
else
    for s in "${REQUIRED_SCRIPTS[@]}"
    do
        if [ ! -f "$SCRIPTS/$s" ]; then
            echo "Error! $SCRIPTS/$s file is missing."
            exit 1
        fi
    done
    [ -f "$SCRIPTS/runSOV" ] || gcc "$SCRIPTS/runSOV.c" -o
"$SCRIPTS/runSOV"
    [ -f "$SCRIPTS/sov" ] || gcc "$SCRIPTS/sov.c" -o "$SCRIPTS/sov"
fi


echo "


PPPPPPPPPPPPPPPPP      SSSSSSSSSSSSSSS    PPPPPPPPPPPPPPPPP
P::::::::::::::::P   SS:::::::::::::::S
SS:::::::::::::::SP::::::::::::::::P
P::::::PPPPPP:::::P
S:::::SSSSSS::::::SS:::::SSSSSS::::::SP::::::PPPPPP:::::P
PP:::::P     P:::::PS:::::S     SSSSSSSS:::::S     SSSSSSSPP:::::P
P:::::P
  P::::P     P:::::PS:::::S            S:::::S              P::::P
P:::::P
```

```
  P::::P     P:::::PS:::::S            S:::::S              P::::P
P:::::P
  P::::PPPPPP:::::P  S::::SSSS          S::::SSSS
P::::PPPPPP:::::P
  P:::::::::::::PP    SS::::::SSSSS      SS::::::SSSSS
P:::::::::::::PP
  P::::PPPPPPPPP        SSS::::::::SS      SSS::::::::SS
P::::PPPPPPPPP
  P::::P                   SSSSSS::::S        SSSSSS::::S   P::::P
  P::::P                        S:::::S            S:::::S  P::::P
  P::::P                        S:::::S            S:::::S  P::::P
PP:::::PP           SSSSSSS     S:::::SSSSSSSS     S:::::SPP:::::PP
P::::::::P          S:::::SSSSSS::::SS:::::SSSSSS::::SP::::::::P
P::::::::P          S:::::::::::::::SS S:::::::::::::::SS P::::::::P
PPPPPPPPPP           SSSSSSSSSSSSSSS      PPPPPPPPPP


"

print_fold () {
    case $1 in
        fold0)
            cat << 'EOF'
              ___     ___      _        ___                __
    o    | __|   / _ \    | |      |   \              /  \
   o         | _|   | (_) |   | |__    | |) |    ___     | () |
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _\__/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold1)
            cat << "EOF"
              ___     ___      _        ___                _
    o    | __|   / _ \    | |      |   \              / |
   o         | _|   | (_) |   | |__    | |) |    ___      | |
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _|_|_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold2)
cat << "EOF"
              ___     ___      _        ___               ___
    o    | __|   / _ \    | |      |   \             |_  )
   o         | _|   | (_) |   | |__    | |) |    ___      / /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    /___|
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold3)
            cat << "EOF"
              ___     ___      _        ___               ____
    o    | __|   / _ \    | |      |   \             |__ /
   o         | _|   | (_) |   | |__    | |) |    ___      |_ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    |___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
```

```
            ;;
        fold4)
            cat << "EOF"
              ____    ____     _        ____             _ _
    o    | __|   / _ \    | |      |   \            | | |
   o         | _|   | (_) |   | |__    | |) |    ___    |_  _|
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _|_|_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold5)
            cat << "EOF"
              ____    ____     _        ____              ___
    o    | __|   / _ \    | |      |   \             | __|
   o         | _|   | (_) |   | |__    | |) |    ___     |__ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    |___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold6)
            cat << "EOF"
              ____    ____     _        ____               __
    o    | __|   / _ \    | |      |   \              / /
   o         | _|   | (_) |   | |__    | |) |    ___     / _ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    \___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold7)
            cat << "EOF"
              ____    ____     _        ____              ____
    o    | __|   / _ \    | |      |   \             |__  |
   o         | _|   | (_) |   | |__    | |) |    ___       / /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _/_/_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold8)
            cat << "EOF"
              ____    ____     _        ____              ___
    o    | __|   / _ \    | |      |   \             ( _ )
   o         | _|   | (_) |   | |__    | |) |    ___     / _ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    \___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold9)
            cat << "EOF"
              ____    ____     _        ____              ___
    o    | __|   / _ \    | |      |   \             / _ \
   o         | _|   | (_) |   | |__    | |) |    ___     \_, /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _/_/_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
```

```
            ;;
        *)
            ;;
    esac

}

print_SOV_score(){
    cat ./resultSOV.txt | grep -e 'SOV' | awk -F' ' '{sovAll += $3;
sovH += $4; sovE += $5; sovC += $6} END {printf "\n    SOV_ALL
SOV_H      SOV_E      SOV_C\n    %.4f    %.4f    %.4f    %.4f\n",
sovAll/NR, sovH/NR, sovE/NR, sovC/NR}'
}

get_filter_name(){
    case $1 in
        "1")
            filter_name="SVM"
            ;;
        "2")
            filter_name="Decision Tree"
            ;;
        "3")
            filter_name="Random Forest"
            ;;
        *)
            filter_name="Unknown Filter"
            ;;
    esac
}

get_filter_abr(){
    case $1 in
        "1")
            filter_abr="svm"
            ;;
        "2")
            filter_abr="dtree"
            ;;
        "3")
            filter_abr="rforest"
            ;;
        *)
            filter_abr="unknown"
            ;;
    esac
}

TEMP_FOLDER="./temp_runAll_CB513"
RUN_ALL_FOLDER="./CB513_runAll_out_files"
[ -d "$TEMP_FOLDER" ] || mkdir "$TEMP_FOLDER"
[ -d "$RUN_ALL_FOLDER" ] || mkdir "$RUN_ALL_FOLDER"
PRINT_CROSS_VAL=true

if [ "$PRINT_CROSS_VAL" = true ]; then
    echo
"==================================================================
"
    echo " >Cross Validation Results"
    echo "------------------------------------------------------------
---------"
    for i in `ls "$CROSS_VAL_FOLDER"`
    do
        echo "$i"
        new_folder="$RUN_ALL_FOLDER/cross_val_res"
        [ -d "$new_folder" ] || mkdir "$new_folder"
        out_file=("$TEMP_FOLDER/$i""_cross_val.txt")
        for j in `ls "$CROSS_VAL_FOLDER/$i"`
        do
            echo "$CROSS_VAL_FOLDER/$i/$j"
        done > "$out_file"
        python "$SCRIPTS/ensembles.py" "$out_file" "$WINDOW" 1
"$new_folder/ens_pred.txt" "$new_folder/ens_sov.txt"
"$new_folder/ens_weka.txt"
        "$SCRIPTS/runSOV" "$new_folder/ens_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py" "$new_folder/ens_pred.txt"
        echo "--------------------------------------------------------
-------------"
    done
fi
echo
"==================================================================
"
echo ""
for i in `ls "$TEST_FOLDER"`
do
    print_fold $i
    new_folder="$RUN_ALL_FOLDER/$i""_results"
    [ -d "$new_folder" ] || mkdir "$new_folder"
    out_file=("$TEMP_FOLDER/$i""_files.txt")

    for j in `ls "$TEST_FOLDER/$i"`
    do
        echo "$TEST_FOLDER/$i/$j"
    done > "$out_file"
    echo
"==================================================================
"
    echo " >Ensembles Results"
    echo "------------------------------------------------------------
---------"
    python "$SCRIPTS/ensembles.py" "$out_file" "$WINDOW" 1
"$new_folder/ensembles_pred.txt" "$new_folder/ensembles_sov.txt"
"$new_folder/ensembles_weka.txt" > "$new_folder/ensembles_out.txt"
    "$SCRIPTS/runSOV" "$new_folder/ensembles_sov.txt"
    print_SOV_score
    python "$SCRIPTS/calc_Q3.py" "$new_folder/ensembles_pred.txt"
    echo
"==================================================================
"
    echo " >Ensembles + External Rules Results"
    echo "------------------------------------------------------------
---------"
    python "$SCRIPTS/externalRules.py"
"$new_folder/ensembles_pred.txt" "$new_folder/ens_rules_sov.txt"
"$new_folder/ens_rules_pred.txt"
    "$SCRIPTS/runSOV" "$new_folder/ens_rules_sov.txt"
    print_SOV_score
    python "$SCRIPTS/calc_Q3.py" "$new_folder/ens_rules_pred.txt"

    for filter in "${filterOpt[@]}"
    do
        get_filter_name $filter
        get_filter_abr $filter
        train_preds=`ls "$TRAIN_FOLDER" | grep "$i" | head -n 1`
        echo
"======================================================================
"
        echo " >Ensembles + External Rules + $filter_name Results"
        echo "--------------------------------------------------------
-------------"
        python "$SCRIPTS/prepare_SVM_files.py"
"$new_folder/ens_rules_pred.txt" "$TRAIN_FOLDER/$train_preds"
"$SVM_WIN" "$new_folder/temp_svm_test.txt"
"$new_folder/temp_svm_train.txt"
        python "$SCRIPTS/train_SVM.py"
"$new_folder/temp_svm_test.txt" "$new_folder/temp_svm_train.txt"
"$SVM_WIN" "$new_folder/ens_rules_pred.txt"
"$new_folder/ens_rules_$filter_abr""_pred.txt"
"$new_folder/ens_rules_$filter_abr""_sov.txt" "$filter" >
"$new_folder/ens_rules_$filter_abr""_out.txt"
        "$SCRIPTS/runSOV"
"$new_folder/ens_rules_$filter_abr""_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_rules_$filter_abr""_pred.txt"
        echo
"======================================================================
"
        echo " >Ensembles + $filter_name Results"
        echo "--------------------------------------------------------
-------------"
        python "$SCRIPTS/prepare_SVM_files.py"
"$new_folder/ensembles_pred.txt" "$TRAIN_FOLDER/$train_preds"
"$SVM_WIN" "$new_folder/temp_svm_test.txt"
"$new_folder/temp_svm_train.txt"
        python "$SCRIPTS/train_SVM.py"
"$new_folder/temp_svm_test.txt" "$new_folder/temp_svm_train.txt"
"$SVM_WIN" "$new_folder/ensembles_pred.txt"
"$new_folder/ens_$filter_abr""_pred.txt"
"$new_folder/ens_$filter_abr""_sov.txt" "$filter" >
"$new_folder/ens_$filter_abr""_out.txt"
        "$SCRIPTS/runSOV" "$new_folder/ens_$filter_abr""_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_$filter_abr""_pred.txt"
        echo
"======================================================================
"
        echo " >Ensembles + $filter_name + External Rules Results"
        echo "--------------------------------------------------------
-------------"
        python "$SCRIPTS/externalRules.py"
"$new_folder/ens_$filter_abr""_pred.txt"
"$new_folder/ens_$filter_abr""_rules_sov.txt"
"$new_folder/ens_$filter_abr""_rules_pred.txt"
        "$SCRIPTS/runSOV"
"$new_folder/ens_$filter_abr""_rules_sov.txt"
```

```
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_$filter_abr""_rules_pred.txt"
    done
    echo
"==================================================================
"
    echo ""
    # exit 0
done

# Remove temp files
rm -rf "$TEMP_FOLDER"
rm resultSOV.txt
rm SOVinput.txt
```

J Appendix

View filtering results of CB513

The following bash script was implemented and used to view all the ensembles and filtering results in a table format, for the CB513 dataset. It was provided by Leontiou [5].

```
#!/bin/bash

# Path to file with ensembles and filtering results
if [ $# -ne 1 ]; then
    file="./final_results_CB513.txt"
else
    file="$1"
fi

if [ ! -f "$file" ]; then
    echo "This file does not exist: $file"
    exit 1
fi

echo "Cross Validation"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "--------------------------------------------------------------"
sed -n "/fold/,/-----/p" "$file" |  grep -E '[0-9]+' | grep -v '[a-
zA-Z]' | tr -s " " " | sed -e 's/^[ \t]*//' | awk -F' '
'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4;
switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "--------------------------------------------------------------"
sed -n '/Ensembles Results/,/=====/p' "$file" | grep -E '[0-9]+'|
grep -v '[a-zA-Z]' | tr -s " " " | sed -e 's/^[ \t]*//' | awk -F' '
'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4;
switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + External Rules Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "--------------------------------------------------------------"
sed -n '/Ensembles + External Rules Results/,/=====/p' "$file" |
grep -E '[0-9]+'| grep -v '[a-zA-Z]' | tr -s " " " | sed -e 's/^[
\t]*//' | awk -F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2;
v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
```

```
echo ""

echo "Ensembles + External Rules + SVM Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + External Rules + SVM Results/,/=====/p' "$file"
| grep -E '[0-9]+'| grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[
\t]*//' | awk -F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2;
v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + SVM Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + SVM Results/,/=====/p' "$file" | grep -E '[0-
9]+'| grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[ \t]*//' | awk -
F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4;
switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + SVM + External Rules Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + SVM + External Rules Results/,/=====/p' "$file"
| grep -E '[0-9]+'| grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[
\t]*//' | awk -F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2;
v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + External Rules + Decision Tree Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + External Rules + Decision Tree
Results/,/=====/p' "$file" | grep -E '[0-9]+'| grep -v '[a-zA-Z]' |
tr -s " " | sed -e 's/^[ \t]*//' | awk -F' ' 'BEGIN{switch=1}{if
(switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + Decision Tree Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + Decision Tree Results/,/=====/p' "$file" | grep
-E '[0-9]+'| grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[ \t]*//' |
awk -F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3;
v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + Decision Tree + External Rules Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
```

```
sed -n '/Ensembles + Decision Tree + External Rules
Results/,/=====/p' "$file" | grep -E '[0-9]+'| grep -v '[a-zA-Z]' |
tr -s " " | sed -e 's/^[ \t]*//' | awk -F' ' 'BEGIN{switch=1}{if
(switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + External Rules + Random Forest Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "------------------------------------------------------------"
sed -n '/Ensembles + External Rules + Random Forest
Results/,/=====/p' "$file" | grep -E '[0-9]+'| grep -v '[a-zA-Z]' |
tr -s " " | sed -e 's/^[ \t]*//' | awk -F' ' 'BEGIN{switch=1}{if
(switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + Random Forest Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "------------------------------------------------------------"
sed -n '/Ensembles + Random Forest Results/,/=====/p' "$file" | grep
-E '[0-9]+'| grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[ \t]*//' |
awk -F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3;
v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + Random Forest + External Rules Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "------------------------------------------------------------"
sed -n '/Ensembles + Random Forest + External Rules
Results/,/=====/p' "$file" | grep -E '[0-9]+'| grep -v '[a-zA-Z]' |
tr -s " " | sed -e 's/^[ \t]*//' | awk -F' ' 'BEGIN{switch=1}{if
(switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
```

K Appendix

All filtering methods on PISCES

Used to apply the ensembles and the filtering methods in various orders and display the results for each fold of the PISCES dataset. It was provided by Leontiou [5].

```bash
#!/bin/bash
# Author : Panayiotis Leontiou
# Since   : May 2020
# Version: 1.0
# Bugs    : No known bugs

TEST_FOLDER="./PISCES_test_pred"
TRAIN_FOLDER="./PISCES_train_pred"
WINDOW="15"
SVM_WIN="19"
filterOpt=( "2" "3" )
SCRIPTS="./q3_sov_scripts"

# Check if required scripts exist
declare -a REQUIRED_SCRIPTS=( "calc_Q3.py" "ensembles.py"
"externalRules.py" "prepare_SVM_files.py" "runSOV.c" "sov.c"
"train_SVM.py")
if [ ! -d "$SCRIPTS" ]; then
    echo "Error! $SCRIPTS directory could not be located."
    exit 1
else
    for s in "${REQUIRED_SCRIPTS[@]}"
    do
        if [ ! -f "$SCRIPTS/$s" ]; then
            echo "Error! $SCRIPTS/$s file is missing."
            exit 1
        fi
    done
    [ -f "$SCRIPTS/runSOV" ] || gcc "$SCRIPTS/runSOV.c" -o
"$SCRIPTS/runSOV"
    [ -f "$SCRIPTS/sov" ] || gcc "$SCRIPTS/sov.c" -o "$SCRIPTS/sov"
fi


echo "


PPPPPPPPPPPPPPPPP      SSSSSSSSSSSSSSS    PPPPPPPPPPPPPPPPP
P::::::::::::::::P   SS:::::::::::::::S
SS:::::::::::::::SP::::::::::::::::P
P::::::PPPPPP:::::P
S:::::SSSSSS::::::SS:::::SSSSSS::::::SP::::::PPPPPP:::::P
PP:::::P     P:::::PS:::::S     SSSSSSSS:::::S     SSSSSSSPP:::::P
P:::::P
  P::::P     P:::::PS:::::S            S:::::S              P::::P
P:::::P
  P::::P     P:::::PS:::::S            S:::::S              P::::P
P:::::P
```

```
  P::::PPPPPP:::::P  S::::SSSS          S::::SSSS
P::::PPPPPP:::::P
  P:::::::::::::PP    SS::::::SSSSS      SS::::::SSSSS
P:::::::::::::PP
  P::::PPPPPPPPP        SSS::::::::SS      SSS::::::::SS
P::::PPPPPPPPP
  P::::P                   SSSSSS::::S        SSSSSS::::S   P::::P
  P::::P                        S:::::S            S:::::S  P::::P
  P::::P                        S:::::S            S:::::S  P::::P
PP::::::PP          SSSSSSS     S:::::SSSSSSSS     S:::::SPP::::::PP
P::::::::P          S::::::SSSSSS:::::SS::::::SSSSSS:::::SP::::::::P
P::::::::P          S:::::::::::::::SS S:::::::::::::::SS P::::::::P
PPPPPPPPPP           SSSSSSSSSSSSSSS      PPPPPPPPPP


"

print_fold () {
    case $1 in
        fold0)
            cat << 'EOF'
              ___     ___      _        ___                __
    o    | __|   / _ \    | |      |   \              /  \
   o         | _|   | (_) |   | |__    | |) |    ___     | () |
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _\__/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold1)
            cat << "EOF"
              ___     ___      _        ___                _
    o    | __|   / _ \    | |      |   \              / |
   o         | _|   | (_) |   | |__    | |) |    ___      | |
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _|_|_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold2)
            cat << "EOF"
              ___     ___      _        ___               ___
    o    | __|   / _ \    | |      |   \             |_  )
   o         | _|   | (_) |   | |__    | |) |    ___      / /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    /___|
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold3)
            cat << "EOF"
              ___     ___      _        ___               ____
    o    | __|   / _ \    | |      |   \             |__ /
   o         | _|   | (_) |   | |__    | |) |    ___      |_ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    |___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold4)
```

```
            cat << "EOF"
              ___     ___      _        ___              _ _
    o    | __|   / _ \    | |      |   \            | | |
   o         | _|   | (_) |   | |__    | |) |    ___    |_  _|
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _|_|_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold5)
            cat << "EOF"
              ___     ___      _        ___               ___
    o    | __|   / _ \    | |      |   \             | __|
   o         | _|   | (_) |   | |__    | |) |    ___     |__ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    |___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold6)
            cat << "EOF"
              ___     ___      _        ___                __
    o    | __|   / _ \    | |      |   \              / /
   o         | _|   | (_) |   | |__    | |) |    ___     / _ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    \___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold7)
            cat << "EOF"
              ___     ___      _        ___               ____
    o    | __|   / _ \    | |      |   \             |__  |
   o         | _|   | (_) |   | |__    | |) |    ___       / /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _/_/_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold8)
            cat << "EOF"
              ___     ___      _        ___               ___
    o O O    | __|   / _ \    | |      |   \             ( _ )
   o         | _|   | (_) |   | |__    | |) |    ___     / _ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    \___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold9)
            cat << "EOF"
              ___     ___      _        ___               ___
    o O O    | __|   / _ \    | |      |   \             / _ \
   o         | _|   | (_) |   | |__    | |) |    ___     \_, /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _/_/_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        *)
```

```
            ;;
    esac

}

print_SOV_score(){
    cat ./resultSOV.txt | grep -e 'SOV' | awk -F' ' '{sovAll += $3;
sovH += $4; sovE += $5; sovC += $6} END {printf "\n    SOV_ALL
SOV_H      SOV_E      SOV_C\n    %.4f    %.4f    %.4f    %.4f\n",
sovAll/NR, sovH/NR, sovE/NR, sovC/NR}'
}

get_filter_name(){
    case $1 in
        "1")
            filter_name="SVM"
            ;;
        "2")
            filter_name="Decision Tree"
            ;;
        "3")
            filter_name="Random Forest"
            ;;
        *)
            filter_name="Unknown Filter"
            ;;
    esac
}

get_filter_abr(){
    case $1 in
        "1")
            filter_abr="svm"
            ;;
        "2")
            filter_abr="dtree"
            ;;
        "3")
            filter_abr="rforest"
            ;;
        *)
            filter_abr="unknown"
            ;;
    esac
}

TEMP_FOLDER="./temp_runAll_PISCES"
RUN_ALL_FOLDER="./PISCES_runAll_out_files"
CROSS_VAL_FOLDER="./PISCES_cross_validation"
[ -d "$TEMP_FOLDER" ] || mkdir "$TEMP_FOLDER"
[ -d "$RUN_ALL_FOLDER" ] || mkdir "$RUN_ALL_FOLDER"
PRINT_CROSS_VAL=true

if [ "$PRINT_CROSS_VAL" = true ]; then
    echo
"=====================================================================
"
    echo " >Cross Validation Results"
    echo "---------------------------------------------------------------
---------"
```

```
    for i in `ls "$CROSS_VAL_FOLDER"`
    do
        echo "$i"
        new_folder="$RUN_ALL_FOLDER/cross_val_res"
        [ -d "$new_folder" ] || mkdir "$new_folder"
        out_file=("$TEMP_FOLDER/$i""_cross_val.txt")
        for j in `ls "$CROSS_VAL_FOLDER/$i"`
        do
            echo "$CROSS_VAL_FOLDER/$i/$j"
        done > "$out_file"
        python "$SCRIPTS/ensembles.py" "$out_file" "$WINDOW" 1
"$new_folder/ens_pred.txt" "$new_folder/ens_sov.txt"
"$new_folder/ens_weka.txt"
        "$SCRIPTS/runSOV" "$new_folder/ens_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py" "$new_folder/ens_pred.txt"
        echo "-----------------------------------------------------------
-------------"
    done
fi
echo
"======================================================================
"
echo ""
for i in `ls "$TEST_FOLDER"`
do
    print_fold $i
    new_folder="$RUN_ALL_FOLDER/$i""_results"
    [ -d "$new_folder" ] || mkdir "$new_folder"
    out_file=("$TEMP_FOLDER/$i""_files.txt")

    for j in `ls "$TEST_FOLDER/$i"`
    do
        echo "$TEST_FOLDER/$i/$j"
    done > "$out_file"
    echo
"======================================================================
"
    echo " >Ensembles Results"
    echo "---------------------------------------------------------------
---------"
    python "$SCRIPTS/ensembles.py" "$out_file" "$WINDOW" 1
"$new_folder/ensembles_pred.txt" "$new_folder/ensembles_sov.txt"
"$new_folder/ensembles_weka.txt" > "$new_folder/ensembles_out.txt"
    "$SCRIPTS/runSOV" "$new_folder/ensembles_sov.txt"
    print_SOV_score
    python "$SCRIPTS/calc_Q3.py" "$new_folder/ensembles_pred.txt"
    echo
"======================================================================
"
    echo " >Ensembles + External Rules Results"
    echo "---------------------------------------------------------------
---------"
    python "$SCRIPTS/externalRules.py"
"$new_folder/ensembles_pred.txt" "$new_folder/ens_rules_sov.txt"
"$new_folder/ens_rules_pred.txt"
    "$SCRIPTS/runSOV" "$new_folder/ens_rules_sov.txt"
    print_SOV_score
    python "$SCRIPTS/calc_Q3.py" "$new_folder/ens_rules_pred.txt"
    for filter in "${filterOpt[@]}"
    do
        get_filter_name $filter
        get_filter_abr $filter
        # echo "$filter $filter_name"
        train_preds=`ls "$TRAIN_FOLDER" | grep "$i" | head -n 1`
        echo
"======================================================================
"
        echo " >Ensembles + External Rules + $filter_name Results"
        echo "--------------------------------------------------------
-------------"
        python "$SCRIPTS/prepare_SVM_files.py"
"$new_folder/ens_rules_pred.txt" "$TRAIN_FOLDER/$train_preds"
"$SVM_WIN" "$new_folder/temp_svm_test.txt"
"$new_folder/temp_svm_train.txt"
        python "$SCRIPTS/train_SVM.py"
"$new_folder/temp_svm_test.txt" "$new_folder/temp_svm_train.txt"
"$SVM_WIN" "$new_folder/ens_rules_pred.txt"
"$new_folder/ens_rules_$filter_abr""_pred.txt"
"$new_folder/ens_rules_$filter_abr""_sov.txt" "$filter" >
"$new_folder/ens_rules_$filter_abr""_out.txt"
        "$SCRIPTS/runSOV"
"$new_folder/ens_rules_$filter_abr""_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_rules_$filter_abr""_pred.txt"
        echo
"======================================================================
"
        echo " >Ensembles + $filter_name Results"
        echo "--------------------------------------------------------
-------------"
        python "$SCRIPTS/prepare_SVM_files.py"
"$new_folder/ensembles_pred.txt" "$TRAIN_FOLDER/$train_preds"
"$SVM_WIN" "$new_folder/temp_svm_test.txt"
"$new_folder/temp_svm_train.txt"
        python "$SCRIPTS/train_SVM.py"
"$new_folder/temp_svm_test.txt" "$new_folder/temp_svm_train.txt"
"$SVM_WIN" "$new_folder/ensembles_pred.txt"
"$new_folder/ens_$filter_abr""_pred.txt"
"$new_folder/ens_$filter_abr""_sov.txt" "$filter" >
"$new_folder/ens_$filter_abr""_out.txt"
        "$SCRIPTS/runSOV" "$new_folder/ens_$filter_abr""_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_$filter_abr""_pred.txt"
        echo
"======================================================================
"
        echo " >Ensembles + $filter_name + External Rules Results"
        echo "--------------------------------------------------------
-------------"
        python "$SCRIPTS/externalRules.py"
"$new_folder/ens_$filter_abr""_pred.txt"
"$new_folder/ens_$filter_abr""_rules_sov.txt"
"$new_folder/ens_$filter_abr""_rules_pred.txt"
        "$SCRIPTS/runSOV"
"$new_folder/ens_$filter_abr""_rules_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_$filter_abr""_rules_pred.txt"
    done
    echo
"=======================================================================
"
    echo ""
    # exit 0
done

# Remove temp files
rm -rf "$TEMP_FOLDER"
rm resultSOV.txt
rm SOVinput.txt
```

L Appendix

View filtering results of PISCES

The following bash script was used to view all the ensembles and filtering results in a table format, for the PISCES dataset. It was provided by Leontiou [5].

```
#!/bin/bash

# Path to file with ensembles and filtering results
if [ $# -ne 1 ]; then
    file="./final_results_PISCES.txt"
else
    file="$1"
fi

if [ ! -f "$file" ]; then
    echo "This file does not exist: $file"
    exit 1
fi

echo "Cross Validation"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "------------------------------------------------------------"
sed -n "/fold/,/-----/p" "$file" |  grep -E '[0-9]+' | grep -v '[a-
zA-Z]' | tr -s " " | sed -e 's/^[ \t]*//' | awk -F' '
'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4;
switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "------------------------------------------------------------"
sed -n '/Ensembles Results/,/=====/p' "$file" | grep -E '[0-9]+'|
grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[ \t]*//' | awk -F' '
'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4;
switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + External Rules Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "------------------------------------------------------------"
sed -n '/Ensembles + External Rules Results/,/=====/p' "$file" |
grep -E '[0-9]+'| grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[
\t]*//' | awk -F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2;
v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + External Rules + Decision Tree Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + External Rules + Decision Tree
Results/,/=====/p' "$file" | grep -E '[0-9]+'| grep -v '[a-zA-Z]' |
tr -s " " | sed -e 's/^[ \t]*//' | awk -F' ' 'BEGIN{switch=1}{if
(switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + Decision Tree Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + Decision Tree Results/,/=====/p' "$file" | grep
-E '[0-9]+'| grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[ \t]*//' |
awk -F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3;
v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + Decision Tree + External Rules Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + Decision Tree + External Rules
Results/,/=====/p' "$file" | grep -E '[0-9]+'| grep -v '[a-zA-Z]' |
tr -s " " | sed -e 's/^[ \t]*//' | awk -F' ' 'BEGIN{switch=1}{if
(switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + External Rules + Random Forest Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + External Rules + Random Forest
Results/,/=====/p' "$file" | grep -E '[0-9]+'| grep -v '[a-zA-Z]' |
tr -s " " | sed -e 's/^[ \t]*//' | awk -F' ' 'BEGIN{switch=1}{if
(switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + Random Forest Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + Random Forest Results/,/=====/p' "$file" | grep
-E '[0-9]+'| grep -v '[a-zA-Z]' | tr -s " " | sed -e 's/^[ \t]*//' |
awk -F' ' 'BEGIN{switch=1}{if (switch == 1) {v1=$1; v2=$2; v3=$3;
v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
echo ""

echo "Ensembles + Random Forest + External Rules Results"
echo -e "Q3_ALL\tQ3_H\tQ3_E\tQ3_C\tSOV_ALL\tSOV_H\tSOV_E\tSOV_C"
echo "-------------------------------------------------------------"
sed -n '/Ensembles + Random Forest + External Rules
Results/,/=====/p' "$file" | grep -E '[0-9]+'| grep -v '[a-zA-Z]' |
tr -s " " | sed -e 's/^[ \t]*//' | awk -F' ' 'BEGIN{switch=1}{if
(switch == 1) {v1=$1; v2=$2; v3=$3; v4=$4; switch=2;} else {printf
"%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", $1, $2, $3, $4,
v1, v2, v3, v4; switch=1}}'
```

M Appendix

All filtering methods on CASP13

Bash script that is used to apply the ensembles and the various filtering techniques and display the results for the independent test set CASP13. It was provided by Leontiou [5].

```
#!/bin/bash
# Author : Panayiotis Leontiou
# Since  : April 2020
# Version: 1.0
# Bugs   : No known bugs

if [ $# -eq 0 ]; then
    echo "No option provided, using default option: CB513..."
    DS=CB513
    SVM_WIN="9"
elif [ $1 = CB513 ]; then
    DS=CB513
    SVM_WIN="9"
elif [ $1 = PISCES ]; then
    DS=PISCES
    SVM_WIN="11"
else
    echo "This option is not valid: $1"
    echo "Available options: CB513, PISCES"
    exit 1
fi

TEST_FOLDER="./CASP13_pred_for_$DS"
TRAIN_FOLDER="./$DS""_train_pred"
CROSS_VAL_FOLDER="./CASP13_cross_validation_for_$DS"
WINDOW="15"
if [ "$DS" = CB513 ]; then
    filterOpt=( "1" "2" "3" )
else
    filterOpt=( "2" "3" )
fi

SCRIPTS="./q3_sov_scripts"

# Check if required scripts exist
declare -a REQUIRED_SCRIPTS=( "calc_Q3.py" "ensembles.py"
"externalRules.py" "prepare_SVM_files.py" "runSOV.c" "sov.c"
"train_SVM.py")
if [ ! -d "$SCRIPTS" ]; then
    echo "Error! $SCRIPTS directory could not be located."
    exit 1
else
    for s in "${REQUIRED_SCRIPTS[@]}"
    do
        if [ ! -f "$SCRIPTS/$s" ]; then
            echo "Error! $SCRIPTS/$s file is missing."
            exit 1
        fi
    done
    [ -f "$SCRIPTS/runSOV" ] || gcc "$SCRIPTS/runSOV.c" -o
"$SCRIPTS/runSOV"
    [ -f "$SCRIPTS/sov" ] || gcc "$SCRIPTS/sov.c" -o "$SCRIPTS/sov"
fi


echo "


PPPPPPPPPPPPPPPPP      SSSSSSSSSSSSSSS    SSSSSSSSSSSSSSS
PPPPPPPPPPPPPPPPP
P::::::::::::::::P   SS:::::::::::::::S
SS:::::::::::::::SP::::::::::::::::P
P::::::PPPPPP:::::P
S:::::SSSSSS::::::SS:::::SSSSSS::::::SP::::::PPPPPP:::::P
PP:::::P     P:::::PS:::::S     SSSSSSSS:::::S     SSSSSSSPP:::::P
P:::::P
  P::::P     P:::::PS:::::S            S:::::S              P::::P
P:::::P
  P::::P     P:::::PS:::::S            S:::::S              P::::P
P:::::P
  P::::PPPPPP:::::P  S::::SSSS          S::::SSSS
P::::PPPPPP:::::P
  P:::::::::::::PP    SS::::::SSSSS      SS::::::SSSSS
P:::::::::::::PP
  P::::PPPPPPPPP        SSS::::::::SS      SSS::::::::SS
P::::PPPPPPPPP
  P::::P                   SSSSSS::::S        SSSSSS::::S   P::::P
  P::::P                        S:::::S            S:::::S  P::::P
  P::::P                        S:::::S            S:::::S  P::::P
PP::::::PP          SSSSSSS     S:::::SSSSSSSS     S:::::SPP::::::PP
P::::::::P          S::::::SSSSSS:::::SS::::::SSSSSS:::::SP::::::::P
P::::::::P          S:::::::::::::::SS S:::::::::::::::SS P::::::::P
PPPPPPPPPP           SSSSSSSSSSSSSSS    SSSSSSSSSSSSSSS   PPPPPPPPPP


"

print_fold () {
    case $1 in
        fold0)
            cat << 'EOF'
              ___       ___      _        ___                __
    o O O    | __|   / _ \    | |      |   \              /  \
   o         | _|   | (_) |   | |__    | |) |    ___      | () |
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _\__/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold1)
            cat << "EOF"
              ___       ___      _        ___                _
    o O O    | __|   / _ \    | |      |   \              / |
   o         | _|   | (_) |   | |__    | |) |    ___      | |
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _|_|_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
```

```
            ;;
        fold2)
            cat << "EOF"
              ___     ___      _        ___               ___
    o O O    | __|   / _ \    | |      |   \             |_  )
   o         | _|   | (_) |   | |__    | |) |    ___      / /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    /___|
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold3)
            cat << "EOF"
              ___     ___      _        ___               ____
    o O O    | __|   / _ \    | |      |   \             |__ /
   o         | _|   | (_) |   | |__    | |) |    ___      |_ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    |___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold4)
            cat << "EOF"
              ___     ___      _        ___              _ _
    o O O    | __|   / _ \    | |      |   \            | | |
   o         | _|   | (_) |   | |__    | |) |    ___    |_  _|
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _|_|_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold5)
            cat << "EOF"
              ___     ___      _        ___               ___
    o O O    | __|   / _ \    | |      |   \             | __|
   o         | _|   | (_) |   | |__    | |) |    ___     |__ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    |___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold6)
            cat << "EOF"
              ___     ___      _        ___                __
    o O O    | __|   / _ \    | |      |   \              / /
   o         | _|   | (_) |   | |__    | |) |    ___     / _ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    \___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold7)
            cat << "EOF"
              ___     ___      _        ___               ____
    o O O    | __|   / _ \    | |      |   \             |__  |
   o         | _|   | (_) |   | |__    | |) |    ___       / /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _/_/_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
```

```
            ;;
        fold8)
            cat << "EOF"
              ___     ___      _        ___               ___
    o O O    | __|   / _ \    | |      |   \             ( _ )
   o         | _|   | (_) |   | |__    | |) |    ___     / _ \
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    \___/
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        fold9)
            cat << "EOF"
              ___     ___      _        ___               ___
    o O O    | __|   / _ \    | |      |   \             / _ \
   o         | _|   | (_) |   | |__    | |) |    ___     \_, /
  TS__[O]   _|_|_    \___/    |____|   |___/    |___|    _/_/_
 {======| _| """ | _|"""""| _|"""""| _|"""""| _|"""""| _|"""""|
./o--000' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-' "`-0-0-'
EOF
            ;;
        *)
            ;;
    esac

}

print_SOV_score(){
    cat ./resultSOV.txt | grep -e 'SOV' | awk -F' ' '{sovAll += $3;
sovH += $4; sovE += $5; sovC += $6} END {printf "\n    SOV_ALL
SOV_H      SOV_E      SOV_C\n    %.4f    %.4f    %.4f    %.4f\n",
sovAll/NR, sovH/NR, sovE/NR, sovC/NR}'
}

get_filter_name(){
    case $1 in
        "1")
            filter_name="SVM"
            ;;
        "2")
            filter_name="Decision Tree"
            ;;
        "3")
            filter_name="Random Forest"
            ;;
        *)
            filter_name="Unknown Filter"
            ;;
    esac
}

get_filter_abr(){
    case $1 in
        "1")
            filter_abr="svm"
            ;;
        "2")
            filter_abr="dtree"
            ;;
        "3")
```

```
            filter_abr="rforest"
            ;;
        *)
            filter_abr="unknown"
            ;;
    esac
}

TEMP_FOLDER="./temp_runAll_CASP13_for_$DS"
RUN_ALL_FOLDER="./CASP13_runAll_out_files_for_$DS"
[ -d "$TEMP_FOLDER" ] || mkdir "$TEMP_FOLDER"
[ -d "$RUN_ALL_FOLDER" ] || mkdir "$RUN_ALL_FOLDER"
PRINT_CROSS_VAL=true

if [ "$PRINT_CROSS_VAL" = true ]; then
    echo
"======================================================================
"
    echo " >Cross Validation Results"
    echo "------------------------------------------------------------
---------"
    for i in `ls "$CROSS_VAL_FOLDER"`
    do
        echo "$i"
        new_folder="$RUN_ALL_FOLDER/cross_val_res"
        [ -d "$new_folder" ] || mkdir "$new_folder"
        out_file=("$TEMP_FOLDER/$i""_cross_val.txt")
        for j in `ls "$CROSS_VAL_FOLDER/$i"`
        do
            echo "$CROSS_VAL_FOLDER/$i/$j"
        done > "$out_file"
        python "$SCRIPTS/ensembles.py" "$out_file" "$WINDOW" 1
"$new_folder/ens_pred.txt" "$new_folder/ens_sov.txt"
"$new_folder/ens_weka.txt"
        "$SCRIPTS/runSOV" "$new_folder/ens_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py" "$new_folder/ens_pred.txt"
        echo "--------------------------------------------------------
-------------"
    done
fi
echo
"======================================================================
"
echo ""
for i in `ls "$TEST_FOLDER"`
do
    print_fold $i
    new_folder="$RUN_ALL_FOLDER/$i""_results"
    [ -d "$new_folder" ] || mkdir "$new_folder"
    out_file=("$TEMP_FOLDER/$i""_files.txt")

    for j in `ls "$TEST_FOLDER/$i"`
    do
        echo "$TEST_FOLDER/$i/$j"
    done > "$out_file"
    echo
"======================================================================
"
    echo " >Ensembles Results"
    echo "-----------------------------------------------------------
---------"
    python "$SCRIPTS/ensembles.py" "$out_file" "$WINDOW" 1
"$new_folder/ensembles_pred.txt" "$new_folder/ensembles_sov.txt"
"$new_folder/ensembles_weka.txt" > "$new_folder/ensembles_out.txt"
    "$SCRIPTS/runSOV" "$new_folder/ensembles_sov.txt"
    print_SOV_score
    python "$SCRIPTS/calc_Q3.py" "$new_folder/ensembles_pred.txt"
    echo
"=====================================================================
"
    echo " >Ensembles + External Rules Results"
    echo "-----------------------------------------------------------
---------"
    python "$SCRIPTS/externalRules.py"
"$new_folder/ensembles_pred.txt" "$new_folder/ens_rules_sov.txt"
"$new_folder/ens_rules_pred.txt"
    "$SCRIPTS/runSOV" "$new_folder/ens_rules_sov.txt"
    print_SOV_score
    python "$SCRIPTS/calc_Q3.py" "$new_folder/ens_rules_pred.txt"

    for filter in "${filterOpt[@]}"
    do
        get_filter_name $filter
        get_filter_abr $filter
        train_preds=`ls "$TRAIN_FOLDER" | grep "$i" | head -n 1`
        echo
"=====================================================================
"
        echo " >Ensembles + External Rules + $filter_name Results"
        echo "-------------------------------------------------------
-------------"
        python "$SCRIPTS/prepare_SVM_files.py"
"$new_folder/ens_rules_pred.txt" "$TRAIN_FOLDER/$train_preds"
"$SVM_WIN" "$new_folder/temp_svm_test.txt"
"$new_folder/temp_svm_train.txt"
        python "$SCRIPTS/train_SVM.py"
"$new_folder/temp_svm_test.txt" "$new_folder/temp_svm_train.txt"
"$SVM_WIN" "$new_folder/ens_rules_pred.txt"
"$new_folder/ens_rules_$filter_abr""_pred.txt"
"$new_folder/ens_rules_$filter_abr""_sov.txt" "$filter" >
"$new_folder/ens_rules_$filter_abr""_out.txt"
        "$SCRIPTS/runSOV"
"$new_folder/ens_rules_$filter_abr""_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_rules_$filter_abr""_pred.txt"
        echo
"=====================================================================
"
        echo " >Ensembles + $filter_name Results"
        echo "-------------------------------------------------------
-------------"
        python "$SCRIPTS/prepare_SVM_files.py"
"$new_folder/ensembles_pred.txt" "$TRAIN_FOLDER/$train_preds"
"$SVM_WIN" "$new_folder/temp_svm_test.txt"
"$new_folder/temp_svm_train.txt"
        python "$SCRIPTS/train_SVM.py"
"$new_folder/temp_svm_test.txt" "$new_folder/temp_svm_train.txt"
"$SVM_WIN" "$new_folder/ensembles_pred.txt"
```

```
"$new_folder/ens_$filter_abr""_pred.txt"
"$new_folder/ens_$filter_abr""_sov.txt" "$filter" >
"$new_folder/ens_$filter_abr""_out.txt"
        "$SCRIPTS/runSOV" "$new_folder/ens_$filter_abr""_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_$filter_abr""_pred.txt"
        echo
"======================================================================
"
        echo " >Ensembles + $filter_name + External Rules Results"
        echo "-------------------------------------------------------------------
-------------"
        python "$SCRIPTS/externalRules.py"
"$new_folder/ens_$filter_abr""_pred.txt"
"$new_folder/ens_$filter_abr""_rules_sov.txt"
"$new_folder/ens_$filter_abr""_rules_pred.txt"
        "$SCRIPTS/runSOV"
"$new_folder/ens_$filter_abr""_rules_sov.txt"
        print_SOV_score
        python "$SCRIPTS/calc_Q3.py"
"$new_folder/ens_$filter_abr""_rules_pred.txt"
    done
    echo
"======================================================================
"
    echo ""
    # exit 0
done

# Remove temp files
rm -rf "$TEMP_FOLDER"
rm resultSOV.txt
rm SOVinput.txt
```

N Appendix

N-terminus vs C-terminus

The following python program was implemented and used in order to find the part of the protein sequence with the highest Q3 accuracy.

```
import numpy as np
filename = "ENRF_fold0_win19_ProtBERT.txt" # filename of file we
want to find misclassified percentages
f = open(filename,"r")  # open file to read
lines = f.readlines() # read all the lines
r = len(lines) # number of lines in file

Allq = np.zeros((4,)) # matrix to store the part of the sequence we
have a misclassified prediciton in 25s
Eq = np.zeros((4,)) # matrix to store the part of the sequence we
have a misclassified E prediciton in 25s(where it had to be E and we
predicted otherwise)
Hq = np.zeros((4,)) # matrix to store the part of the sequence we
have a misclassified H prediciton in 25s
Cq = np.zeros((4,)) # matrix to store the part of the sequence we
have a misclassified C prediciton in 25s

tAllq = np.zeros((4,)) #
tEq = np.zeros((4,)) # all original E structure in 25s
tHq = np.zeros((4,)) # all original H structure in 25s
tCq = np.zeros((4,)) # all original C structure in 25s
print("Protein name" + "," + "Type"+ ","+"0-24%" + "," + "25-49%" +
"," + "50-74%" + "," + "75-100%")
for i in range(0,r,4):
    ### temp matrices for every protein########
    tempAllq = np.zeros((4,))  # matrix to store the part of the
sequence we have a misclassified prediciton in 25s
    tempEq = np.zeros((4,))  # matrix to store the part of the
sequence we have a misclassified E prediciton in 25s(where it had to
be E and we predicted otherwise)
    tempHq = np.zeros((4,))  # matrix to store the part of the
sequence we have a misclassified H prediciton in 25s
    tempCq = np.zeros((4,))  # matrix to store the part of the
sequence we have a misclassified C prediciton in 25s

    temptAllq = np.zeros((4,))  #
    temptEq = np.zeros((4,))  # all original E structure in 25s
    temptHq = np.zeros((4,))  # all original H structure in 25s
    temptCq = np.zeros((4,))  # all original C structure in 25s

    name = lines[i].replace("\n","")
    aa = lines[i + 1].replace("\n","")  # hold the amino acid
sequence of the current protein(aa)
    orss = lines[i + 2].replace("\n","")  # hold the original
secondary structure(orss)
    prss = lines[i + 3].replace("\n","")  # hold the predicted
structure of the filtering technique(ft)
```

```
    index = 1 # variable to determine the place we are in the 
sequence we are looking
    sequence_len = len(aa) # sequence len
    for i in range(sequence_len):  # start form the 6th char since 
the first five are the title of each line
        placeQuarters = int(np.abs(np.ceil(((index / sequence_len) * 
4) - 1)))
        tAllq[placeQuarters] += 1
        temptAllq[placeQuarters] +=1
        if orss[i] == 'E':
            tEq[placeQuarters] += 1
            temptEq[placeQuarters] += 1
        elif orss[i] == 'C':
            tCq[placeQuarters] += 1
            temptCq[placeQuarters] += 1
        else:
            tHq[placeQuarters] += 1
            temptHq[placeQuarters] += 1
        if orss[i] != prss[i]:
            Allq[placeQuarters] += 1
            tempAllq[placeQuarters] += 1
            if orss[i] == 'E':
                Eq[placeQuarters] += 1
                tempEq[placeQuarters] += 1
            elif orss[i] == 'C':
                Cq[placeQuarters] += 1
                tempCq[placeQuarters] += 1
            else:
                Hq[placeQuarters] += 1
                tempHq[placeQuarters] += 1
        index += 1
    ########################### print results for quarters 
################################################
    print(name,end=',')
    print("Q3",end=',')
    for i in range(0, 4):
        if (temptAllq[i] !=0 ):
            miss =  (tempAllq[i] / temptAllq[i])
            accuracy = (1 -miss) * 100
            accuracy = round(accuracy,2)
            print(accuracy, end=",")
        else:
            print("-", end=",")

    print("QC",end=',')
    for i in range(0, 4):
        if (temptCq[i] !=0 ):
            miss =  (tempCq[i] / temptCq[i])
            accuracy = (1 -miss) * 100
            accuracy = round(accuracy,2)
            print(accuracy, end=",")
        else:
            print("-", end=",")

    print("QE",end=',')
    for i in range(0, 4):
        if (temptEq[i] !=0 ):
            miss =  (tempEq[i] / temptEq[i])
            accuracy = (1 -miss) * 100
            accuracy = round(accuracy,2)
            print(accuracy, end=",")
        else:
            print("-", end=",")

    print("QH",end =',')
    for i in range(0, 4):
        if (temptHq[i] !=0 ):
            miss =  (tempHq[i] / temptHq[i])
            accuracy = (1 -miss) * 100
            accuracy = round(accuracy,2)
            print(accuracy, end=",")
        else:
            print("-", end=",")
    print()

########### PRINT TOTALS##########################
########################### print results for quarters
################################################
print(" " + "," + "Average"+ ","+"0-24%" + "," + "25-49%" + "," +
"50-74%" + "," + "75-100%")
print(" " + "," +"Q3",end= ",")
for i in range(0,4):
    miss = (Allq[i] / tAllq[i])
    accuracy = (1 - miss) * 100
    accuracy = round(accuracy,2)
    print(accuracy, end=",")

print("QC",end=',')
for i in range(0,4):
    miss = (Cq[i] / tCq[i])
    accuracy = (1 - miss) * 100
    accuracy = round(accuracy,2)
    print(accuracy, end=",")

print("QE",end=',')
for i in range(0,4):
    miss = (Eq[i] / tEq[i])
    accuracy = (1 - miss) * 100
    accuracy = round(accuracy,2)
    print(accuracy, end=",")

print("QH",end=',')
for i in range(0,4):
    miss = (Hq[i] / tHq[i])
    accuracy = (1 - miss) * 100
    accuracy = round(accuracy,2)
    print(accuracy, end=",")
print("\n")
```