ThesisReport

by Srikrishna Datla

FILE	332071_SRIKRISHNA_DATLA_THESISREPORT_2313963_1145264447.PDF		
TIME SUBMITTED	29-APR-2018 10:40PM (UTC+0100)	WORD COUNT	23751
SUBMISSION ID	87820437	CHARACTER COUNT	129789

Development of protein disorder prediction server using machine learning

Srikrishna Datla[1]

MEng Computer Science

Dr. Kevin Bryson

Submission date: 29/04/2018

[1]**Disclaimer:** This report is submitted as part requirement for the MEng Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged

Abstract

Proteins form interesting folds and formations based on a range of factors including their amino acid composition, environmental factors and their role. Proteins tend to conform to the same set of structures, referred to as the 'native fold', although there are some that do not form this stable state and perform different operations to those commonly known. This structural aspect of proteins is called intrinsic disorder and can be predicted through analysis of a protein's secondary structure, however there are some difficulties in this process of prediction and knowledge on intrinsic Disordered Proteins (IDPs) is not complete. Potential prediction of disordered regions in proteins may help in understanding the functionality of these disordered regions and their implications in certain diseases such as cancer or cardiovascular diseases.

Protein secondary structure prediction is a highly competitive and difficult challenge within the Bioinformatics field, with a wide range of prediction services available to the public each with their own niche and accuracy (examples including the popular UCL-founded DISOPRED and PSIPRED). Current methods of prediction use advanced machine learning methods such as Support Vector Machines (SVMs) and Neural Networks to build prediction models and this project aims to utilise these techniques as well as additional environmental information about the protein, in particular organelle location, to learn a model that better fits the optimal predictor for IDP prediction. This project will use a combination of machine learning procedures trained on filtered disordered protein data from the DisProt database to build a model that can accurately predict disorder regions while extracting protein environmental information to attempt to improve accuracy.

The outcome from building several machine learning models, including Support Vector Machines, Bagging Meta-Estimators and Multi-Layered Perceptrons, suggest that organelle prediction does not necessarily improve accuracy scoring for disordered region prediction beyond the expected accuracy from a basic polypeptide structure prediction service. However, by using current day machine learning techniques and optimisations, we are able to predict disordered regions with accuracies relative to leading competitive prediction servers.

Contents

Chapter 1

Project Description

The primary focus of this project is to test and build a variety of machine learning models that can be used to estimate disordered protein regions, given a string of amino acids. Experimentation is also required to determine if there is a possible correlation between disordered region prediction and feature extraction pertaining to the sub-cellular localisation of the protein. For the purposes of this report, there are several key phrases that are used to describe biological concepts and processes, such as:

- Amino Acid: A simple organic chemical compound consisting of at least one amino group (-NH_2) and one carboxyl group (-$COOH$). There are 20 amino acids, with a unique R group for each acid, and due to the nature of amino acids exhibiting stereo-isomerism (variants of amino acids possess the same chemical formula but differ in 3-D orientations of atoms [1]), scientists have discovered that there are two variants of each amino acid - The left-hand variant which is present in most organisms and the right-hand variant.

- Protein: Chemical compounds consisting of one or more amino acids chemically bonded by peptide bonds and stably folded into specific 3-D shapes[3]. A protein's 3-D structure is instrumental to its function and small variations in the orientation of its atoms will greatly affect the protein's ability to perform its function.

- Peptide: Peptides are short chains of amino acids chemically bonded by peptide bonds[4]. The terms 'protein' and 'peptide' are almost interchangeable as the length of peptides is defined arbitrarily, although proteins tend to be much longer in length and can contain more than a single peptide chain. For this reason, proteins are sometimes referred to as 'polypeptides'.

- Organelle: sub-cellular compartments that specialise in cell specific tasks. These locations tend to have particular environmental conditions, such as slight acidity/alkalinity and presence of certain substances (e.g. ribosomes containing high quantities of ribosomal ribonucleic acid (rRNA)).

- Folding: The process of a protein chain forming its native 3-D structure and orientation. Once translated from underlying mRNA to a sequence of amino acids, the protein begins

arranging itself in a specific configuration in order to begin performing its desired function [5].

1.1 Goals

This project focuses on one particularly structural aspect of proteins called intrinsic disorder where proteins do not form the stable native fold that gives the protein its function and instead perform other functions instead, in some cases implicating them in certain diseases. The DisProt database has information about protein disorder where the proteins are categorised by their functional ontology and this project aims to make use of the this data. Delving further into this subject area through this project aims to:

- Learn about disordered protein ontology and its behaviour based on varying environmental effects (e.g. differences in pH or organelles)
- Determine whether disordered region prediction can be aided by factoring the environmental effects into the machine learning models

The following objectives have also been identified as outcomes of this project:

- Review currently available resources / predictions servers for disordered protein regions and organelle prediction.
- Process Raw DisProt and PDB about disordered proteins into a combined dataset for machine learning analysis.
- Generate datasets for organelle prediction and extract features to build a machine learning model for organelle prediction.
- Discover patterns in the combined dataset and build a relevant machine learning model for disorder region prediction.
- Evaluate the success of the model generated with environmental considerations with other respective models which are currently being used (e.g. DISOPRED).

1.2 Deliverables

The project would involve the complete development of a bioinformatics server:

- From processing of raw DisProt and PDB structural data.
- To the application of scikit-learn for machine learning.

During the course of the project the following deliverables were also defined:

- An analysis of currently available datasets related to disordered protein regions and protein secondary structure prediction servers, looking for patterns and distinguishing information about disordered regions and whether environmental attributes affect their prediction.

- Generation of a number of machine learning models and comparison to other current methods.

IMPORTANT NOTE: For organelle prediction, my algorithms and testing were drawn from my submission for COMPGI10/COMPM058 Bioinformatics Coursework Report. Relevant algorithms are based on the processes I followed for the denoted submission, although modifications have been made to repurpose the code to be implemented in this Thesis. A reference to the described material are cited whenever I draw from this source.

Chapter 2

Background

Proteins are chains of amino acids, complexly intertwined and folded, with the structure of the protein denoting the function of the protein. Most proteins tend to have a 'stable' structure, with little variation in the peptide and hydrogen bonds between amino acids, and this 'stable' structure helps the protein in its role - either providing specifically oriented binding sites or positioning to be acceptable for bonds. Despite extended research into the nature of protein folding and potential algorithms for prediction, protein structure prediction from amino acid sequence remains a largely unsolved problem - however reliable techniques have been discovered that greatly improve accuracies for prediction structure.

2.1 Protein Folding

Polypeptide structures are classified into four major categories:

- Primary Structure - This is the initial linear chain of amino acids, arranged in a single plane within a peptide or protein. By convention, primary structure is written from the amino group (N) terminus to the carboxyl group (C) terminus. Most prediction servers infer tertiary structure from sequences gathered from protein primary structures.
- Secondary Structure - Secondary structure is derived from basic folding of the primary structure, whereby localised regions of the chain arrange themselves to form regular shapes, first introduced in 1952 by Linderstrom-Lang K.U. [13] [14]. The formation of these motifs depends on the hydrogen bonding that occurs between amino and carboxyl groups found in the backbone of the peptide, with the 3 most common shapes being:
  - Alpha Helices (α-helix): A peptide tube resembling a corkscrew, where each amino acid within the spiral has two sets of hydrogen bonds with the preceding and following amino acid, separated by approximately four amino acids. Alpha helices are relatively stable due to the large number of bonds in between amino acids within the helix as well as the adequate spacing between peptides in the helix, however heating does break the bonds present. Not all amino acids tend to form the angles and hydrogen bonds required for this shape, with some amino acids such as proline and glycine having a low

affinity for generating alpha helices [15]. Meanwhile amino acids such as Methionine, Alanine, Lysine, Leusine and Glutamic Acid have a much higher affinity for forming helices.

- Beta-Pleated Sheets (β-sheet): The name of this motif comes from the formation of peptide beta sheets bonded laterally by hydrogen bonds. Beta-Pleated sheets can be arranged in parallel, anti-parallel or a mixed arrangement and each configuration affects the number of hydrogen bonds that can form between the sheets (anti-parallel sheets allow for many more hydrogen bonds due to the positioning the amino groups and carboxyl being adjacent in space, while parallel sheets have much wider spacing). Individual strands are typically short in length (roughly 3 to 10 residues long) and the resulting sheets normally contain at most 6 strands [16]. Compared to α-helix, β-sheets are much more stable and heat-resistant and do not melt.
- Coils - Any remaining peptides which are not part of a motif are defined as 'coils'. These coils join together the regular shapes within protein secondary structure and do not tend to conform to a specified pattern, however their arrangement is still vital to the protein's function.

Proteins consist predominantly of α-helices and β-sheets, and identification of these regions is particularly important when predicting protein secondary structure.

- Tertiary Structure - The 3-D shape of the protein, this governs the presence of protein subunits known as 'domains'. Domains are highly conserved regions of proteins that can act independently of the remainder of the protein, and each protein can contain many domains. Domains can be a range of lengths and act as subunits, performing specialised tasks for the protein, and are found to evolve independently of the protein they reside in. Protein tertiary structure is stabilised by the numerous chemical bonds that can form between the members of its chains, with bonds such as hydrogen bonds, ionic bonds and sulphur bridges forming between the R groups of the side-chains [17]. Similar to secondary structure, tertiary structure can be determined from the primary structure of the protein, which highlights the usefulness of primary structure feature extraction in prediction services.
- Quaternary Structure - Quaternary protein structure is similar to tertiary structure, combining multiple protein self-enclosed units into its complex formation. Many proteins consist of multiple polypeptide chains and the use of quaternary structure allows for the description of a protein subunit in reference to another. Nomenclature of these protein complexes uses the '-mer' notation, for example a complex consisting of a single polypeptide chain being referred to as a 'monomer', while combining two 'monomers' results in a 'dimer'.

The expectation for many decades was that protein structures remained fixed, which allowed for their specific 3-D shape, but the discovery of intrinsically disordered proteins in the mid-20^{th} Century challenged this assumption. With the significant advancements in X-ray crystallography and disordered prediction, the significance of disordered region prediction has garnered large amounts of interest within the Molecular Biology and Bioinformatics community.

2.2 Disordered Proteins

Early attempts at verifying protein 3-D structure came from the use of protein crystallography, a high resolution version of microscopy that enabled the visualisation of the proteins atomic composition. Polypeptides crystals would be synthesised in a laboratory and their structure would be determined by refracting X-rays off of the crystal and building an electron density map based on the intensities measured across a diffraction map [28] [29].

Based on these initial processes, biologists theorised that protein structures must be fixed in order for the protein to function properly, and major papers at the time (such as the famous 'Lock-and-Key model' envisioned by Emil Fischer in 1894[30]) supported this idea as it fit the narrative that protein structure could be determined entirely from the amino acid sequence which in itself is translated from base DNA. However, some experts challenged this assumption, such as Fred Karush who suggested that some proteins exhibited the ability to change structure to influence their binding patterns [31] [32]. Karush defined this interaction as 'Configurational Adaptability' and noted that proteins may be able to manipulate their own 3-D structure to accommodate a different binding ion, which occurred with bovine serum albumin that Karush was investigating at the time. This ideology was later supported by large numbers of proteins displaying undetectable regions when undergoing x-ray crystallography, which suggested that the proteins occupied multiple 3-D positions, and from further research this birthed the idea of Intrinsically Disordered Proteins (IDP).

Recent research into the functionality and existence of IDPs have shown that IDPs are present in all organisms (IDPs of length > 30 are present in 33% of eukaryotic cells [10]), and they might be instrumental in understanding some diseases [33]. The disordered regions present within these polypeptides can form regular shapes similar to motifs in protein secondary structure, and examples of the motifs that disordered regions form include Flexible Linkers and Loop Connectors.

2.3 Protein Secondary Structure Prediction

The main issue with protein secondary structure prediction lies with the convoluted folds that proteins undergo, and research has been done in attempt to find mathematical or chemical models that can accurately determine the folds that can be made with a polypeptide sequence[6]. However, such techniques have not adequately accounted for all possible structures and for this reason, protein structure prediction remains one of the top 125 most important problems in science [7]. The Molecular Biology and Bioinformatics community have provided incentive for progress within this field through a variety of competitions, with one of the most popular being Critical Assessment for protein Structure Prediction (CASP).

CASP is one of the most well-known competitions, operating biennially since its inception in 1994, and has gathered interest from organisations across the globe with its 13th iteration set to start in May 2018. Target sets are gathered through purification of new proteins, with competing predictors attempting to deduce the structure of new, unknown protein sequences. The use of new data allows for a platform for completely blind evaluation of any predictors, although the small number of target proteins could be criticised for introducing small amounts of bias when predicting. During its tenure, CASP has grown to accommodate a range of prediction targets

including:

- Disordered Region Prediction
- Domain Boundary Prediction
- Secondary Structure Prediction
- De Novo Prediction
- High Accuracy Template-Based Prediction
- Fold Recognition
- Model Refinement
- Residue-Residue Contact Prediction
- Function Prediction

CASP serves a major role in the development of better protein structure prediction servers, and we will discuss two of the highest performing prediction servers - PSIPRED[8] and DISOPRED [9][10][11].

2.3.1 PSIPRED

PSIPRED is a project developed by David T. Jones, concentrating efforts on predicting protein secondary structure given a sequence of amino acids. PSIPRED made use of several machine learning techniques, building a two-stage neural network which processes Position Specific Scoring Matrices (PSSMs) gathered from running PSI-BLAST (Position-Specific Iterated BLAST) on input data. PSI-BLAST is a revolutionary update to the original BLAST(Basic Local Alignment Search Tool) [40] sequence comparison algorithm, with PSI-BLAST generating position specific profiles based alignments generated against protein sequences in the database, which are sensitive to weak biologically relevant sequence similarities [39].

At the time, the most reliable models were built using the concept of 'comparative modelling' whereby assumptions are made that proteins follow a similar structure to previously labelled proteins and by making comparative relations between new proteins and existing proteins, a model will reliably find a template protein structure with high degrees of sequence similarity. This form of prediction is classified as Template-Based Modelling, which uses pre-existing data about protein structure distributions and similarities to deduce new polypeptide 3-D shapes. The opposite branch of protein structure prediction is Ab Initio prediction (also known as De Novo prediction) which attempts to build classifiers 'from scratch' with no references to a template structure. For PSIPRED the adopted methodology was Ab Initio prediction.

PSI-BLAST results can be prone to errors, with sequencing depending heavily on the data bank used for sequence alignments, as dataset bias is amplified through iterations of PSI-BLAST. Iteration convergence can also be affected detrimentally by the inclusion of protein sequences with regions of relatively low complexity - these regions are somewhat common across proteins and offer little biological relevance in sequence alignment - so careful consideration of the training dataset

must be done to mitigate any erroneous results potentially gained from PSI-BLAST. After gathering protein sequences from a range of public data banks, the collated information is then fed to SEG [41], which filters out low information content residues, and then curated to remove regions likely to form coiled-coil structures and transmembrane segments.

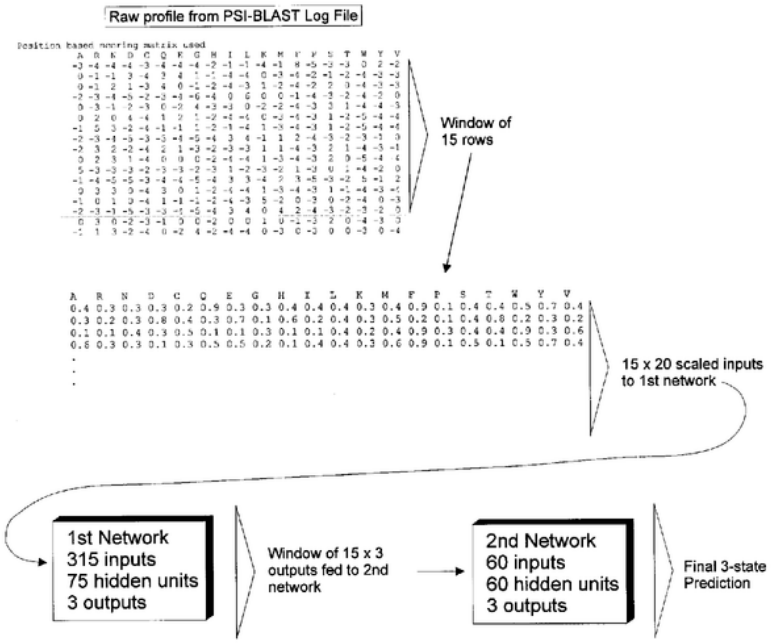

Figure 2.1: Outline of PSIPRED method, describing the processing of PSSM data into a 2-stage neural network

Figure 2.1 displays the methodology for PSIPRED, from PSSM data to final prediction. This filtered data was then fed into a neural network for preliminary prediction, incorporated as a sliding window of length 15 (the sliding window size was determined through extensive testing), and used to predict 3 classes: α-helix, β-sheet or coil. The use of sliding windows for data representation meant that inputs into the neural network needed to account for boundary cases where the sliding window would have to include PSSM data at the start and end terminus. For this reason, an additional amino acid was included to act as a population of amino acid specifically for boundary positions. Final predictions were derived from the second stage of the neural network, which aimed to filter successive outputs from the initial network, and weights were updated using an on-line back-propagation procedure.

Results gathered from testing PSIPRED with CASP3 indicated that PSIPRED's algorithm was quite reliable and accurate, achieving Q3 scores of 76.5% to 78.3% depending the definition of the observed protein secondary structure - the highest published prediction score achieved at the time. Despite the impressive achievements of PSIPRED, David T. Jones did note that there were possibilities for optimisation that may improve scoring, potentially through removing residual biases remaining from training data acquisition. However, the success of PSIPRED does raise

interest in the use of PSSMs as training sets for disordered region prediction as disordered region prediction relies heavily on the underlying secondary structure.

2.3.2 DISOPRED

DISOPRED is one of the highest performing disordered region predictors, built by the same team as the well known PSIPRED. DISOPRED is currently in its 3rd version as of 2014, and has outperformed many competitors in disorder region prediction in CASP10 (DISOPRED3 placed within top two for both Matthew's Correlation Coefficient and Area-Under-Curve metrics in CASP10)[42]. The third iteration of DISOPRED was developed in efforts to improve general accuracy for previous versions through increased sensitivity for intrinsically disordered regions (IDR) longer than 20 amino acids, as well as to annotate protein-binding sites within the IDRs.
At the time of DISOPRED3's inception, protein-binding site prediction had garnered some interest with predictors such as ANCHOR using a linear regression model to model energy deficits and gains from conforming to specific structures for binding [43]. DISOPRED3 intended to introduce novel ideas to this problem with the inclusion of a trained SVM that would distinguish short peptides bound to globular domains from flexible linkers, through extraction of protein sequence features. For disordered region prediction, all entries from the DisProt v5.0[44] database were used as well as a compiled database derived from PISCES[**?**] - these sources provided reliable annotations for both long and short length disordered region. Drawing from previous experience with protein structure prediction with PSIPRED, DISOPRED utilised PSSM information gathered from running all protein sequences 3 times through PSI-BLAST. 3 Models were built to determine the various aspects of protein secondary structure:

- A Neural Network topology based on the original DISOPRED [9] was retrained on data saturated with longer chain disordered regions. This model attempted to improve coverage for long chain disorder region prediction which was lacking in DISOPRED2 [10].
- The main SVM model introduced in DISOPRED2[10] was reintroduced as performed well in predicting shorter chain disordered regions reliably.
- A Nearest Neighbour classifier was built to reduce the latency with training the other models - nearest neighbour classifiers require less training than a neural network or SVM and for this reason can make DISOPRED3 much easier to maintain and update.
- A final Neural Network was included to provide meta-estimation of the 3 main models described earlier. This neural network filtered results and estimated disordered region confidence by comparing prediction probabilities for both classes (Ordered/Disordered). This helped compensate for the imbalanced nature of the original training sets.

The final predictions gathered from DISOPRED3 and its performance at CASP9 and CASP10 show that the optimisations included between DISOPRED2 and DISOPRED3 significantly improve disordered region prediction across a range of disordered region chain lengths. Another important note to acknowledge is the potential accuracy improvements gained from processing pure sequence data as opposed to PSSM data, with PSSM data providing higher sensitivity, MCC and F1-scores at the cost of specificity and some precision. The trade-off between the various

possible data representations raises questions about which representation may be best suited for this Thesis.

2.4 Subcellular Localisation Prediction

By definition, Eukaryotic cells differ from Prokaryotic cells due to the inclusion of highly specialised regions within their membrane, known as organelles, although some bacteria do contain primitive forms of organelles [34]. Examples of organelles include the nucleus - which contains all of the genetic information for the cell in the form of DNA, Mitochondria - which produces the energy for the cell, and chloroplasts - which produce energy in plant cells through the process of photosynthesis. These sub-cellular locations differ largely in their internal environments due to their different functions, and these discrepancies are seen through changes in chemical compositions, acidity/alkalinity levels and presence of specific compounds.
Proteins existing in the cytoplasm may not be present in similar quantities as those in the nucleus or ribosomes, and for this reason sub-cellular localisation may be useful in determining the type of protein present. Especially for disordered region prediction, the environmental conditions of organelles may alter the tertiary structure of the proteins present and knowledge of this may assist disorder predictors in learning a polypeptide structure model.
Organelle prediction has been well researched within the Bioinformatics community, and various predictors are available today that accurately predict intracellular location, but there are limitations on their predicting power. One of the most well known organelle predictors is PSORTb [36], currently in its 3rd revision.

2.4.1 PSORTb

The first iteration for PSORTb was made available in 2003[36], focused on Gram-negative and Gram-positive bacteria sub-cellular localisation and it was widely considered one of the best performers at the time with subsequent versions of PSORT being extended into other protein subcellular localisation predictors (e.g. WoLF PSORT[35]). The third iteration of PSORTb has been developed to make significant improvements to recall scores in addition to allowing for the prediction of all types of Prokaryotes - not only bacteria with atypical cell membranes, but also archaea [37]. Typically sub-cellular localisation predictors attempt to categorise organelles into a small set of organelle classes, and they do not account for all possible types of sub-cellular regions as this is infeasible, either due to limitations in labelled training data or to reduce computational consumption. This inevitably stifles the predictors ability to account for additional classes and may negatively affect prediction due to gaps in annotations, and PSORTb 3.0 attempted to increase coverage of prediction by accounting for the following categories:

- Host-associated proteins - These are proteins that have travelled to the host cell by means of the bacterial secreted systems.
- Type III Secretion: Proteins that compose the Type III secretion apparatus
- Spore: Proteins that compose a spore structure
- Flagellar: Proteins that compose the archaeal or bacterial flagellum

- Fimbrial: Proteins that compose the archaeal or bacterial fimbrium or pilus
- Cytoplasmic: Proteins residing within the cell cytoplasm
- Cytoplasmic Membrane: Proteins residing within the cytoplasmic membrance
- Cell wall: Proteins composing the Cell wall
- Extracellular: Proteins existing in extracellular regions
- Inner/Outer Membrane: Proteins residing in inner and outer membrane
- Periplasmic: Proteins residing in the periplasmic space

The predictions of these classes differ based on the nature of their bacteria of origination, with Gram-positive bacteria proteins having avenues for predicting cystoplasmic membrane and cell wall localisation while Gram-negative bacteria proteins have access to inner/outer membrane and periplasmic region prediction).

In terms of implementation, PSORTb 3.0 follows a similar methodology to its predecessors, training on a culled protein sequence database (PSORTdb[38]) combined with annotations gathered from the SwissProt [46] and protein localisation data gathered from manual literature reviews. After separating the protein sequences by their sorting (Gram-positive/Gram-negative/Archaeal), the training set was populated with 810 archaeal proteins, 8230 Gram-positive proteins and 2652 Gram-negative proteins. The Support Vector Machines built for the preceding PSORTb v2.0 were retrained on the new dataset and an augmented suffix tree was introduced to extract frequent subsequences which was important as proteins often contain repeated sequences of low complexity that may interfere with predictions.

Once the intermediate predictions had been gathered, a Bayesian network collected all results and output a final prediction based on the performance of the SVMS. The final evaluation supported expectations of higher recall and precision scores compared to previous iterations, with PSORTb 3.0 outperforming PSORTb 2.0 with a higher precision rate (96.3 vs 90.3) and higher recall rate (91.8 vs 54.4) when comparing the extracellular and periplasmic fractions for P.aeruginosa PA01. PSORTb 3.0 also achieved its second objective of widening coverage for protein prediction, with an increase of 17.1% in proteomic coverage for Gram-negative bacterial data and an increase of 5.9% in proteomic coverage for Gram-positive bacterial data.

After reviewing methodologies undertaken from PSORTb 3.0, many lessons can be learnt regarding optimal strategies for improving accuracy in sub-cellular localisation prediction. Finding an adequately large training set and determining an appropriate number of classes can greatly mitigate effects of bias and false positive prediction, and care must be taken when dealing with feature extraction for proteins as they can often contain regions of low complexity that interfere with results.

Chapter 3

Methodology

3.1 Training Data

Sub-cellular Localisation and Disordered Region prediction relies heavily on the datasets used for training and testing purposes, with competitive predictor publications outlining the advantages and disadvantages in catering the training set to best represent the distribution of protein sequences for each problem. For this reason, dataset selection and culling has been undergone for both organelle prediction and disordered region prediction, to reduce on any unknown biases when training and validating.

From the initial premise outlined earlier, four major sub-cellular regions have been identified to be used as datasets for organelle prediction:

- Cytosolic: Proteins found within the cytosol of the cell, but not inside any organelles
- Secreted: Proteins which are transported out of a cell
- Nuclear: Proteins found/used within the cell's nucleus
- Mitochondrial: proteins transported to the cell's mitochondria

For organelle prediction, these sequences will have a large number of features extracted and then filtering will be done to remove low relevance features. Recursive Feature Extraction (RFE) will be employed to identify important features and then the culled feature list will be trained on, using a number of machine learning models.

As for disordered region prediction, a variety of disordered region labelled data is provided from open source data banks such as DisProt[44][45], MobiDB[47] and SwissProt[46]. Analysis of these datasets has been done to determine the best source and once the best training source is determined, we then extract features from the protein sequences before experimentally testing different machine learning models.

3.1.1 Organelle Prediction Data[50]

The organelle prediction dataset chosen for implementation is the UCL Bioinformatics Organelle Prediction dataset, provided by David T Jones [27]. The data provided to us is split into the four

organelles described earlier: Cytosolic, Secreted, Nuclear and Mitochondrial. The distribution of the individual labels is as follows:

- Cytosolic - 3004 protein sequences
- Mitochondrial - 1299 protein sequences
- Nuclear - 3314 protein sequences
- Secreted - 1605 protein sequences

As we can see from the distribution, this set is quite imbalanced with the majority of proteins labelled as either Cytosolic proteins or Nuclear proteins. Training on this dataset could result in models being biased towards predicting these classes and therefore appropriate steps need to be taken account for the imbalance. A useful countermeasure to reduce this bias is to balance the datasets through over-sampling or under-sampling techniques, or factor class weights when training the models.

Under-sampling would require creating a subset representation of the majority classes with less data points, until the samples of all labels (including the minority classes) is equal. The benefit of this approach is that the equal distribution of labels would allow for training that has reduced bias, however if we were to employ under-sampling then we would lose a large amount of useful samples (in our case there would be a reduction of of $\frac{2}{3}$ of sequences labelled as Nuclear). a new problem is also introduced as most new sequences provided during testing would retain a distribution similar to our original training set as opposed to the new equalised dataset, and fitting the models to the under-sampled training set may lead to over-predicting of the minority classes.

Another possibility to alleviate the issue is to over-sample minority classes, which makes assumptions about the representation of the minority classes in order to generate new protein sequences representative of the minority classes until we again end up with an equal distribution of labels. There are two possible implementations if we are to over-sample the dataset until an even distribution is found:

- Naive method: Duplicate protein sequences in the minority classes with a stated distribution, until we end with an equal distribution training set.
- Complex method (e.g. Synthetic Minority over-sampling Technique - SMOTE[19]): Build models representing protein sequences belonging to a minority class and generate new proteins using these models.

Over-sampling itself is not necessarily an improvement over under-sampling; the naive method may present biases that can interfere whilst training, leading to the generated model being overfitted on the features that are present in the repeated sequences, which is a particularly difficult problem to address if these proteins contain regions of low complexity. The assumption models built for methods such as SMOTE are also not perfect and can conflict with existing data during training.

After assessing both methods, a combination of both under-sampling and over-sampling may allow for an optimal compromise where we get the induced benefits of a more balanced dataset while not having the drawbacks of highly biased data. For implementation purposes, there are several

Python frameworks available that provide implementations of the methods listed above, one of which is sci-kit learn[48] with its branch library imbalance-learn[49]. Both of these have been included in the source code.

3.1.2 Disorder Region Prediction Data

As stated earlier, disordered region prediction has been improving incredibly over the last few decades, but the gap between the number of unannotated protein sequences found in data banks and the number of manually annotated protein sequences is exceptionally large. For this reason, prediction is required and choosing a reliable, non-redundant dataset is imperative if we are to achieve significant accuracy scores. Disordered region annotated sequences are present in many major databases, with sources such as DisProt, MobiDb and SwissProt all providing sequences annotated with disordered regions, but the information provided is not necessarily reliable or relevant for this project. A comparison of the sources is provided below:

- MobiDB[47] is one of the largest disordered data providers, recently updated to version 3.0 to include more curated annotations for a range of features (disorder, linear interacting peptides and protein secondary structure)[?]. It boasts one of the largest curated datasets (as of 2018, there are 96,102 protein sequences with manually curated disordered regions), but the curated labelling of sequences is derived from collating information from multiple other datasets (including UniProt, DisProt and FuzDB). Having investigated potential issues for overlap in sequence similarity, MobiDB has been determined to contain a lot of redundant sequences which could greatly reduce effectiveness of training on this source, as the models may be affected by little variance in the training set.

- DisProt [44] is a manually curated dataset specifically built to tackle the problem of lack of IDR annotated protein sequences. The most recent iteration [45] has been updated to purge conflicting cases and upgrade functional classification to reflect technological advances made in the last decade. As of 2018, DisProt holds annotations for more than 800 sequences and these annotations have a large amount of the information on their nature, such as chain variants with disordered regions. One of the main problems with DisProt relative to other datasets is the relatively small number of annotated sequences, as this may lead to over-fitting of models due to the small relative bias in data. However, this is mitigated by the fact that the data is non-redundant and accurately annotated.

After analysing the potential datasets for disordered region prediction, all entries (803 sequences) from DisProt were selected for training and validation. Annotations were taken from the PDB consensus and comparison were made to the manually curated labels to see if there were potential conflicts between the labelling. Annotated regions were denoted by 3 identifiers: 's' - structured, 'd' - disordered, 'c'-context dependent and immediately there arose an issue from the ambiguous nature of the context dependent label. During testing this issue was addressed by comparing performances of models when including context dependent labels and removing these labels. From the initial DisProt sequence data, the distribution of labels were as follows:

- Total Amino Acid Count: 406750

- Structured Positions Count: 329237
- Disordered Positions Count: 77431
- Context Dependent Count: 82

The distribution of structured positions compared to disordered positions show that this training set is quite unbalanced, although the distribution appears to follow a similar pattern to the population of disordered regions in many proteins. To accommodate for the imbalance in labels, we implement models that are less sensitive to unbalanced data as well as apply ensemble methods to attempt to counteract the imbalance (Using libraries such as imbalance-learn[49]). We also have to take into consideration the fact that the DisProt dataset only contains protein sequences with disordered regions, meaning that in terms of training models we may not get as many negative training values as we only look at proteins rich in disorder. This is likely mitigated by our methods for training which look at residue windows as opposed to a whole protein sequence.

After gathering the labelled data, sliding windows were built using two methods, One-Hot-Encoding (OHE) and PSSM profiles. OHE historically is a useful representation, particularly in the protein prediction field, and allows for a simplified format to test if there is an strong correlation between presence of amino acids in a sliding window and the likelihood of disordered region. This is undertaken by vectorisation of each position, building a vector of size 21 (20 amino acids[20] plus an additional field for boundary cases) and including a binary value corresponding to the amino acid present at that position. OHE was predominantly used as a naive representation for early testing and provided a basis for determining the best value for the sliding window size. Based on the review of PSIPRED[**?**], PSSM data was also gathered by processing the dataset 3 times through PSI-BLAST (evaluated against the UniRef90 [**?**] database which is a dataset that contains protein sequence clusters with at least 90% sequence identity and 80% overlap), and building corresponding sliding windows.

Comparisons for the performance of these representations can be found in Appendix A. File formats is an especially tedious concept to deal with when building models, and a large amount of time was spent in early development dealing with the formats specific to each stage of the methodology. The initial DisProt file information was provided in JSON format and this data required scrubbing to remove unnecessary fields as well as processing of labels to account for differences in manually curated labels and consensus labels (there were small differences in notation such as using capitalised ASCII characters). Sequences also needed to be exported into FASTA file formats to undergo conversion into PSSM data and relevant files were created to output the FASTA files as well as read the MTX files gathered from running PSI-BLAST on training data. Over time, code has been written to accommodate tab-separated files, csv files, txt files as well as CASP file format '.dr'.

In-sample validation data for testing purposes was split from the initial training set (30& split) and this validation set was used for determining comparative accuracies for models. To make comparisons of the models against well known predictors, CASP10 disorder prediction data and DISOPRED3 testing data was found.

3.2 Model Selection

3.2.1 Organelle Prediction Models [50]

The initial data format provided is in the form of FASTS files, with minimal descriptions and the protein sequence formatted by IUPAC standard [20]. The descriptors provided do contain a set of characteristics of each sequence that could potentially be extracted as features: Species of origination, Accession codes and identifiers, and protein name. There are some discrepancies that must be accounted for however, primarily the inclusion of some additional amino acids:

- Selenocyseine (Amino Acid Code: U)
- Pyrrolysine (Amino Acid Code: O)
- Unknown (Amino Acid Code: X)

These amino acids deviate from the original IUPAC standard amino acid nomenclature as they are normally not encountered in most protein sequences found in data banks (most data banks adopt a strategy of only including the main 20 amino acids), and accounting for this new group requires modification of existing protein analysis algorithm. When calculating features such as Grand Average of Hydropathicity (GRAVY) and other similar measures, we made sure to account for the additional amino acids in the relevant protein scales by using the mean scale value for each of the new amino acids. This method seems naive and somewhat arbitrary as it has no biological basis and makes assumptions about the nature of Selenocysteine and Pyrrolysine, however due to frequencies of these amino acids being minute in comparison to the rest of the amino acids, the biases introduced are somewhat trivial.
Through analysing the original protein sequences with packages such as BioPython, the following features were extracted for organelle prediction:

- Sequence Length
- Amino Acid Global Composition: Distribution of amino acids across the whole sequence.
- Amino Acid Local Composition: Distribution of amino acids across first 50 residues and last 50 residues.
- Molecular Weights (Global and Local): Molecular Weights calculated across the entire sequence and first 50 and last 50 residues.
- Isoelectric Point [22]: Isoelectric point is a good indicator of the pH values that the protein is stable in and early research suggests that using Isoelectric point to determine if a protiein is more acidic or alkaline may be able to assist in localisation, as acidic proteins are more likely to reside in cytosolic areas [23].
- Flexibility: Calculated using the Vihinen[26] flexibility scale for proteins.
- Secondary Structure Proportion: Using the Protein Analysis function, we can find a proportion of secondary structure motifs (Coils/Helices/Sheets) in the protein sequence, which can provide an insight into the role of the protein - potentially useful in determining the sub-cellular location of the protein.

- Aromaticity: Calculated using a modified version of the Lobry scale [25], aromacity may be useful in determining the functionality of a protein (proteins consisting of more aromatic amino acids may function better at absorbing UV light [24]. Due to the relation of protein functional ontology and sub-cellular localisation, analysing aromaticity values may be useful in organelle prediction.

- Grand Average of Hydropathicity (GRAVY): Calculated using the Kyte-Doolittle[21] scale. Using GRAVY values allows us to determine if a protein region is more likely to be hydrophobic or hydrophilic, which could potentially indicate protein function - proteins such as carrier proteins found in regions such as the cell membrane contain more hydrophobic regions in order to perform their role.

After feature extraction, we are left with feature vectors of dimension 82, and culling of this vector must be done to remove irrelevant features that could potentially hinder organelle prediction. A particularly popular method to remove unnecessary is Recursive Feature Elimination (RFE). By modelling the input data with an algorithm that assigns coefficients to each feature, features with little variance can be eliminated and pruned recursively until a small subset of the original feature set that accurately predicts the labels is found. This technique is known as RFE and is an effective method to find the optimal number of features as well as rank the features by their importance. To make use of RFE, a Random Forest classifier is built and trains on the entire training set, weighting features by importance and we extract only the most important features based on this ranking.

Figure 3.1 displays the cross validation scores for Random Forest Classifier when fitting with subsets of features with different length. The highest cross validation score is achieved with a subset of 49 features. A table of the rankings of each feature can be found in Table A.1, which displays not only the ranking but the specific feature type.

Model Selection

After processing and ranking the important features using RFE, we are returned with feature vectors with a dimension of 49, and these are subsequently used for training a set of models (split into training and validation sets using 3-fold split), listed below:

- Multi-Layered Perceptron:

- Support Vector Classifier (SVC): An SVC has been used as it provides the best estimator for unbalanced datasets with the least manipulation of the training algorithm. The library used for the SVC did allow for class weightings to attempt to reduce the effects of the training data being unbalanced.

- Multi-Layered Perceptron (MLP): A basic multi-layered perceptron was used as a benchmark for higher complexity predictors, and the hidden layers would be used to see if there was any additional attributes that could be determined from the initial input list. The MLP had the following node layout ($49 \times 8 \times 4$).

- Logistic Regression: Logistic regression was built in a similar manner to the MLP, as a basic predictor that tried to estimate classes with a linear modelling. If a linear interpolation of the

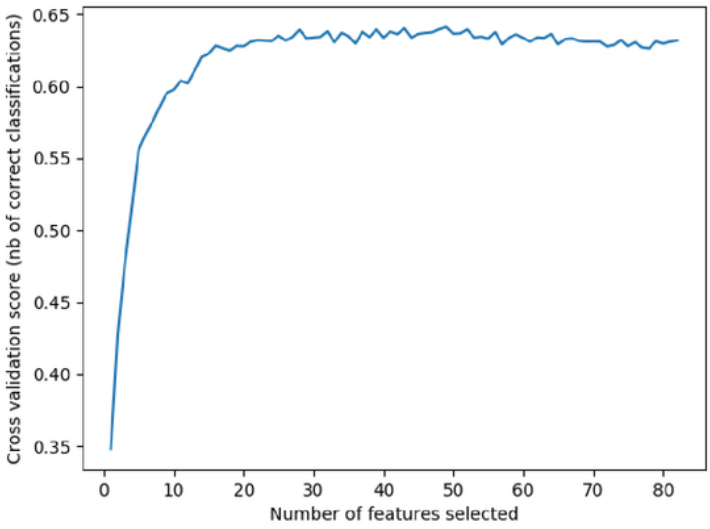

Figure 3.1: Recursive Feature Elimination cross validation scores for different feature subset lengths

training information could be found, then this information would be useful in understanding whether organelle prediction is a linear problem.

- AdaBoost Classifier: AdaBoost is a meta-estimator that uses weak predictors to estimate labels and combining weak estimators may be useful in working with unbalanced datasets, and therefore could be a good predictor using the training set given.
- Balanced Bagging Classifier (BBC): An algorithm derived from a Bagging Meta-estimator, this classifier can be found in the unbalanced learning python library and attempts to perform ensemble under-sampling and over-sampling methods to deal with skewed distributions with training classes.

3.2.2 Disorder Region Prediction

Due to the experimental nature of this research Thesis, a large number of machine learning techniques were built and incorporated in order to understand the effectiveness of different data representations and small optimisations. One of the first predictors build was a basic linear separator, and using the results gathered from tweaking various parameters and viewing evaluation metrics, we would then determine if the basic predictor was sufficient for disordered region prediction. Input data to models was provided in a matrix of form $N \times M$ where N is the observed sliding window size, which was finalised during testing of the Naive estimator and the Complex Estimators, and M is the dimension of the position vector. Much like the organelle predictor, there was a 3 fold split for training and validation set for contrasting the different models.

Naive Estimator

The first model built and tested was a Logistic Regression Model, which was used as a basis for determining disordered region prediction is a linearly separable problem. A logistic model was built and trained using 10-fold cross validation (Table 3.2 contains the results from testing with a range of sliding window sizes and running a Logistic Regression Model). By varying sliding window size and viewing changes in accuracy, we also determined the a good idea of the best sliding window size for future model parameters).

Complex Estimators

After building and testing the naive estimator, the metrics gathered suggested that disordered region prediction could not be simplified to a linear problem, and as such a more complex model would be required to determine important features in the data. Based on experimenting with the sliding window parameters, a correlation shown was larger sliding window sizes resulted in an improvement in accuracy. Provided with this insight, the preliminary complex models used a large window size - 21 - and used this window size when encoding the protein sequences.
Utilising the OHE data gathered from the 803 proteins, we then needed to compare the effectiveness of OHE in predicting disordered protein prediction compared to PSSM values, which was evaluated by training equivalent MLP models with PSSM data and OHE data. The results of this experiment can be found in Appendix E, which displays the training results with MLPs using different hidden node layouts.
Different hidden node layouts were used early on to determine if there was significant issues with modified node layouts affecting the prediction power of MLPs. Based on research material for DISOPRED and PSIPRED as well as minor experimentation, we found that there was little change when modifying the number of hidden layers (e.g. going from 20x20x1 to 20x1) as the secondary layers of nodes did not infer any additional features from the input nodes. Outputting two nodes instead of one did not negatively affect results either, and after testing several layouts, the settled layout for the nodes in later MLP models was ($WindowSize \times 21$, 2). During the testing procedure, results indicated the PSSM values were preferable for prediction and this lead to their implementation in future experimentation. PSSM values returned from PSI-BLAST contained records for the IUPAC standard 20 amino acids but we needed to account for the additional amino acid log probability - the boundary case. Several methods were tested to account for this case:

- Use a basic background model for this boundary case (assuming equal probability of all amino acids) which would be position vectors with log(1/20) in each position of vector. This initiative was quite naive as some amino acids do not naturally occur at the same probabilites as others and boundary cases may be mistaken for actual features.
- Adding an additional amino acid position in a similar manner as that done for OHE. Accounting for an additional case allowed for the distinct separation of boundary cases from original PSSM data and prevented misclassification of boundary cases.

Through testing, it was determined that the best method to continue ahead with was the inclusion of additional row in the PSSM windows.

In addition to building a MLP model for disordered region prediction, several other models were built and trained to find an estimator that distinguished disordered regions best. These models are as detailed below:

- Balanced Bagging Meta-estimator: A modification of a bagging meta-estimator tasked with improving accuracy when dealing with unbalanced datasets, which is particularly important because we want to make sure that we do not want training set bias to affect the returned results. two versions of the Bagging estimator were built and tested - one version with a set of 10 MLP weak estimators (with the same hidden layer layout as the MLP built earlier) and one version with 200 decision tree weak estimators). The different versions of the Bagging Meta-Estimator allowed us to acknowledge the variety of learning algorithms available and if a new model may provide a novel approach for estimation.

- Support Vector Classifier: SVCs were used due to their frequent usage in literature regarding disordered region prediction, for example a SVM is the main component for DISOPRED2, and testing the results from a SVM will allow us to know if disordered regions can be accurately separated from structured positions.

- Multi-Layered Perceptron: Sci-kit learn [48] provides neural network functionality through the use of MLPs and this is used as benchmark for high complexity models. Varying the network topology will allow us to see if there any other hidden features to be extracted from the training data.

One of the main objectives of this thesis is to determine if disordered region prediction can be affected by the inclusion of environmental information for the protein, and to this end we needed to build a model that could be used as a control model to compare against a combined model. The control model built is found by testing the range of models listed above and finding the model that achieves the most consistent and highest scores during testing. Consistency and Accuracy are the cornerstones for a good predictor as consistent results from testing of new out-of-sample data correlates with a less biased predictor and high accuracy is always required.

3.2.3 Combined Estimators

After building the organelle predictor, a combined model incorporating both the organelle predictor and the disordered models was implemented. This combined model contained a base Balanced Bagging Meta-estimator for disordered prediction (this base was chosen by finding the best predictor from preliminary testing of a basic disordered region predictor), and combined the results gathered from running a sequence through the base predictor as well as the organelle predictor. The results from runs were then combined in a matrix with row vectors of size 6 - 4 values corresponding with four class organelle prediction and two last remaining values corresponding to the two class disordered prediction. Figure 3.2 shows the general flow of data formats in the combined model and the transformations that are made.

Figure 3.3 displays the pipeline of models and training data required to build the combined models. Several models were trained to serve as the final combined model: a SVM, two Bagging Meta-estimator (the first was build with 10 MLP weak estimators and the second used 200 weak

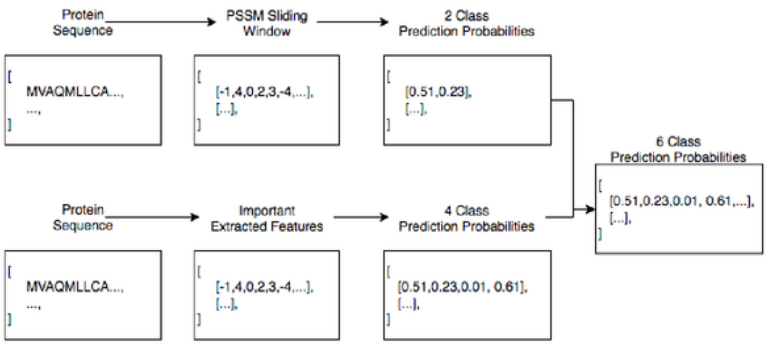

Figure 3.2: Combined model data transformations flowchart

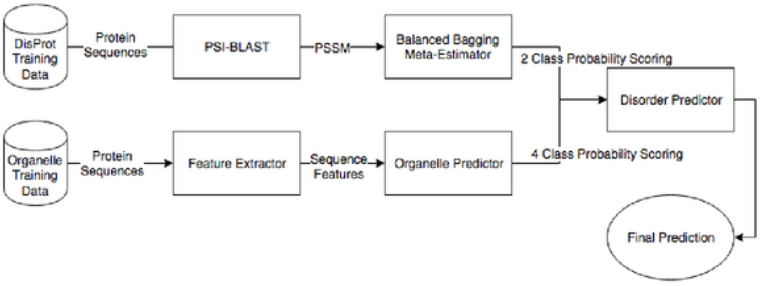

Figure 3.3: Combined model pipeline outline

estimators) as well as MLP. Evaluation of these models against the singular disorder region prediction models would reveal whether organelle prediction benefited disorder prediction and results of these trials can be found in Appendix B.

3.3 Testing

During the design stage of the project, 3 unique testing sets were identified for use:

- Validation Set: The dataset split from training data using 3-fold splitting, the protein sequences found here contained a large number of protein sequences with a similar proportion of disordered regions to the initial training set. There are 108549 positions labelled as structured, 25649 positions labelled as disordered and 30 positions labelled as context-dependent (this label is later designated as structured after initial testing).
- DISOPRED3 Testing Set: The testing set provided with the latest release of DISOPRED3, this testing set is used for validation of the predictor and can help in understanding the comparative fitting of our models versus a competitive solution. This data set is relatively small (only 9 sequences) and this may affect the results gathered from testing.

- CASP10 Disorder Targets: The original CASP10 disorder targets that competitors of CASP attempted to predict, this out-of-sample testing has been tested against to compare accuracies of the models we build in relation to other well-known disorder prediction servers.

When evaluating the predictions of various models, the following metrics were used:

- Confusion Matrix: A direct visualisation of the number of correctly and incorrectly predicted labels, we can use the confusion matrix to see if a model over-predicts a particular label which may be a symptom of failure to correctly separate the individual classes given input data.
- Recall / Specificity / Precision - All 3 of these metrics are a stable in bioinformatics research and allow for a basic insight into the accuracy of predictions as an average across multiple classes.
- Matthew's Correlation Coefficient: Matthews correlation coefficient is a good measure of quality of binary classifiers (although it has been generalised to account for multi-class cases), and can provide a better understanding of accuracy compared to a confusion matrix. This metric is bounded by the range [-1,1] where -1 to 0 indicates a poor performance of the predictor or disagreement between prediction and observation, while MCC values closer to 1 indicate a high beforming predictor across all classes.
- Receiver Operating Characteristic Plot: ROC plots provide an understanding of the true predictive power of a model by visually demonstrating the trade-off between model sensitivity and specificity. For ROC plots, the Area-Under-Curve is another indirect metric that can be used to see the text accuracy of the predictor, with areas closer to 1 denoting an excellent prediction.

3.3.1 Organelle Prediction Testing

Organelle prediction has been conducted using the validation set split from the initial dataset. By selecting the best performing organelle predictor for disordered prediction, we hope to reduce noise in the correlation gathered from subsequent disordered region prediction of the combined model.

Table 3.1: Table displaying the metric evaluation for each model built for subcellular localisation prediction (rounded to 2dp).

Model	Average Recall Score	Average Precision	Average Specificity	Matthew's Correlation Coefficient
SVC	0.67	0.67	0.86	0.54
Multi-Layered Perceptron	0.61	0.61	0.83	0.45
Logistic Regression	0.67	0.67	0.86	0.44
Adaboost Classifer	0.67	0.67	0.86	0.42
Balanced Bagging Classifier	0.65	0.65	0.85	0.51

Table 3.1 displays the individual scores for classification of each model and from the results gathered, we can see that the SVC works best with the unbalanced dataset and produced the highest average recall values, which may be due to its capabilities in finding the largest margins

for each class and a side-effect of the optimisations made within the Python library.
Some interesting observations can be made from Table 3.1 such as, all models providing a relatively high MCC value suggesting the prediction of all classes is relatively accurate (although improvements could definitely be introduced to improve this) and recall scores remain similar between all the models. The little variance in recall and precision/specificity could be understood to mean that the extracted features provide an adequate representation of the protein sequences from which sub-cellular localisation can be inferred.

Figures A.1 to A.5 show the ROC plots for the individual organelle prediction models and the provide an insight into the prediction of the four classes - (the following encoding is done for the four labels: 0 = Cytosolic, 1 = Mitochondrial , 2 = Nuclear, 3 = Secreted). Cytosolic and Nuclear prediction has the lower Area-Under-Curve values across all models despite the distribution of training data being heavily skewed for both classes, which suggests that many members of the other classes are incorrectly classified as either Cytosolic or Nuclear labels), demonstrating the effects of unbalanced training data and therefore an area of improvement for future work.

Due to the SVC providing the highest statistic across the board for all metrics, this model was chosen to be used as the main organelle predictor to be incorporated into the combined model.

3.3.2 Disorder Region Prediction Testing

At the onset of the implementation, testing was done to determine the correlation between sliding window size on prediction as well as the effectiveness of using simple linear models to predict disordered regions. Table 3.2 displays the results gathered from testing the naive Logistic Regression model using 10-fold cross validation using OHE validation set.
An easy observation to make is that the accuracy of the model improves as sliding window size increases, although accuracy plateaus after sliding window sizes of 21. The accuracy measure is not necessarily a good metric for determining if the Logistic model predicted disordered region well as the distribution of disordered positions is only a small factor of the number of structured positions. Therefore overestimating structured positions may still result in a high accuracy value despite the lack of disorder region prediction labels. However, this table does give an indication that disorder feature extraction is possible using the One-Hot-Encoding technique.

Once testing had been complete for naive model, various parameters for the higher complexity models needed to be confirmed and intermediate testing was done at this stage. One of the first key tests to be ran checked whether the accuracy improvement with sliding window increase in the Logistic model is a similar observation that could be made in the higher complexity models.

Tables B.1 to B.7 can be found in Appendix B and provide the results gathered from building disordered region prediction models .
Tables B.1 displays the results from testing a range of sliding window sizes (close to the optimal sliding window size found from earlier testing) and compared the accuracy of predictions using these window sizes. As seen from the table, sliding window size does not greatly improve accuracy

Table 3.2: Table displaying the 10-fold cross validation accuracy scores (rounded to 2dp) for the naive Logistic Regression model.

Sliding Window Size	Average Accuracy	Standard Deviation
7	0.84	0.2
9	0.85	0.2
11	0.87	0.2
13	0.88	0.2
15	0.89	0.2
17	0.90	0.2
19	0.91	0.2
21	0.92	0.1
23	0.92	0.1
25	0.93	0.1

to the same degree as in the Logistic Regression Model, but from this test we can determine a good benchmark value for future development of models - A sliding window size of 21 provides the best metric results across the range of sliding window sizes tested in Table B.1.
We then had to determine the interaction between hidden layer layout for the MLP models and whether the hidden layer actually inferred additional features not visible in the RAW data. Table B.2 displays this experiment and confirms that the hidden layer introduced did not significantly improve prediction of disordered regions beyond an acceptable threshold. When building the MLP models, the single hidden layer was adopted as the prediction accuracy was negligible in comparison to the deeper hidden layer setup while still providing a training time that was much more feasible for testing. The trade-off of training time and prediction is significant as some early tests to build models such as SVMs could often take multiple days and testing became infeasible as building SVMS with different values for hyperparameters could take a ridiculous amount of time.
Table B.3 shows the results from testing different sliding window sizes for PSSM data and comparing if the prediction values followed the same pattern observed for OHE data. As sliding window size increased, it does show an improvement in disordered region prediction but a trade-off occurs a decrease in performance of prediction of structured regions. Compared to the OHE data, PSSM data does favour structured region prediction compared to disordered region prediction, but the average accuracy for both classes in quite similar for both PSSM training data and OHE training data. Based on these results and the review of DISOPRED3 literature, PSSM training data was used for all further model training.
Table B.4 recounts another intermediate test to determine if including the small number of context dependent labels could necessarily improve overall disordered region prediction. As seen from the graph, only the SVM correctly classified the context dependent labels at the costs of falsely predicting structured and disordered labels as context dependent. For this reason, the context dependent label and all corresponding positions were scrubbed from training data.

The three remaining tables in Appendix B (Tables B.5, B.6 & B.7) show the final predicted values for the best performing basic disordered region predictors as well as the combined models. B.5 displays results testing against the validation set, while B.6 and B.7 test against the DISOPRED3 and CASP10 results respectively. Despite the inclusion of additional environmental information, the combined models (last four rows of each table) performed significantly worse

than their basic counterparts. The MLP model with organelle prediction information failed to predict any disordered regions correctly while the other models returned limited accuracy predictions, which contrasts with the basic models having a relatively high prediction accuracy of both classes.By looking at the MCC values in all tables, the combined models fail to achieve an MCC of even 0.5 which suggests that the model does not capture relevant features for disordered region prediction. When testing against the DISOPRED3 testing set, all predictors performed badly with only the basic Balanced Bagging Classifier (with MLP estimator) and the MLP models providing a reasonable recall rate - however this may be due to small sample size as the DISOPRED3 testing set only contained 9 protein sequences.

Figures A.6 to A.9 display the ROC plots for the basic disordered region models while A.10 to A.12 display the ROC plots generated from the combined models. These ROC plots highlight the same issues shown in the tabled results discussed earlier and confirm the same conclusion reached regarding the main objective of this report - that inclusion of sub-cellular localisation for disordered region prediction does not improve the prediction of disordered region prediction.

If we are to compare the accuracy of the final built models against competitive predictors available currently (by looking at the CASP10 testing results), the results suggest that the models built for this thesis are not very competitive and require substantial improvement. The best performing model established in the final results (Balanced Bagging Classifier with MLP weak estimator) achieved an MCC value of 0.27 which outperformed models such as sDisPred and GS metaserver G494 [**?**], although it does not exceed the naive classifier results for CASP10. This is something to note in the conclusion where judgement will be provided on the results achieved from this project. As a result of testing, we can see that including environmental factors for disorder region prediction does not improve the prediction of disordered regions.

Chapter 4

Conclusion

4.1 Achievements

In the Introduction, we highlighted several goals that needed to be achieved for this project, and we address whether we have completed these objectives to the standard expected.

- In the literature review we have demonstrated an understanding of current knowledge of the problem background, with in-depth research to cover the topics of protein folding, protein disorder and environmental effects (particularly organelle information).
- We have also reviewed currently data sources such as MobiDB and DisProt as well as leading prediction servers in the fields of organelle prediction and protein disorder prediction.
- A combined disorder labelled dataset has been developed from the underlying DisProt data and this has been represented in different formats for machine learning analysis. However, there are possibilities for introducing other datasets to gather a larger dataset for training.
- We have not successfully identified and collated a dataset for organelle prediction, instead relying on the use of a pre-existing dataset. This objective was not deemed important during the course of this project, but could be taken into consideration if future work is to be undertaken.
- Several disorder prediction models and organelle prediction models have been built and their performance has been evaluated in comparison with leading models as well as each other. Due to the models not performing at the same level as leading models, there is definitely room for improvement, either through optimisation of the models or building more complex models and testing their effectiveness at disorder prediction.
- A combined model has also been built which takes into consideration environmental factors for disorder prediction and its results have been evaluated compared to previous models we have built.

In terms of the objectives of the project most of the objectives have been suitably met, and some objectives have opportunities to develop and exceed the levels met currently.

4.2 Critical Evaluation

When considering the goals that this project has met, there is plenty of room for improvement and the a different direction could have been achieved to find a contrasting set of results. Firstly, the training sets and testing sets used were selected based on their relevancy to the problem and greater research could be done to find manually curated datasets with a range of short chain and long chain disorder regions. Due to the time limitations of this project, we could not develop our own manually curated dataset with protein sequences based on available literature and this is a method adopted by many other predictors. The training set was also not filtered by other means to remove regions with low complexity, which is an optimisation identified in the literature review, and this was due to lack of complete understanding of how to filter correctly. At the beginning of the project, my knowledge of the subject area was fairly shallow and through research and undertaking of the project my understanding has greatly improved but not enough to necessarily know the optimal choices to make for improving model accuracy. For organelle prediction, several modifications could also be done such as balancing the dataset or including more than four classes.

Despite testing a large number of models with varying hyper-parameters, time restrictions did hinder the number of tests that could have been run, and a better machine learning technique could have been employed to improve overall accuracy for both the organelle prediction and disorder prediction. The subsequent conclusion may therefore differ from the conclusion reached in this report, which is a side-effect of the methodology undertaken. However, the likelihood of this occurring is not particularly high as the methodology followed is reasonable and sound as we have taken into consideration a large number of factors: factors affecting training sets, organelle prediction and machine learning optimisations. From the reasoning followed, we can conclude that the project has succeeded in achieving its primary goal.

4.3 Future Work

Taking into consideration the performance of the final predictions on CASP10 data, we can see that there is significant room for improvement if we are to build a high accuracy disordered region predictor that is relatively competitive. Small modifications could greatly assure this happens - such as balancing the initial dataset or including protein sequences which are rich in longer chain disordered regions. or even using an entirely different model.

Due to the nature of disorder prediction being an unsolved problem, many different approaches could taken to produce a competitive disorder prediction model and therefore this problem remains open-ended. This project intends to verify the effectiveness of considering organelle prediction for disorder prediction and based on the methodology and testing, we can safely confirm that it has completed this goal.

One important feature that could be undertaken in future work is the development a web server that can provide the scientific community with an easy way to use the final predictor from this project and early on, this feature was removed from the project due to limitations in working with technologies such as Django and Azure.

4.4 Final Thoughts

Having undertaken this thesis, I have gained a large amount of respect for the various groups competitive and assisting each other in the development of protein annotation data. At the inception of the project I was largely unaware of the complexities in prediction as well as the large gap in protein sequence annotations which motivated prediction servers, and being exposed to this community has renewed my interest in the Bioinformatics field. I am also excited by the vast number of applications of machine learning in different fields and the potential to apply machine learning techniques into more problems in the Biology discipline.

In conclusion, we can say that the project has been moderately successful in its purpose and some additional work could be done to completely verify and improve on the conclusions drawn today.

Bibliography

[1] IUPAC. Compendium of Chemical Terminology. 2nd edition (the "Gold Book") (1997), Online corrected version: (2006).

[2] Collin's Dictionary. Amino Acid English Definition. collinsdictionary.com/dictionary/english/amino-acid. (2018).

[3] Collin's Dictionary. Protein English Definition. collinsdictionary.com/dictionary/english/protein. (2018).

[4] Nature Education. Peptide English Definition nature.com/scitable/definition/peptide-317. (2018).

[5] B. Alberts, A. Johnson,J. Lewis, M. Raff,K. Roberts & P. Walters The Shape and Structure of Proteins Molecular Biology of the Cell; Fourth Edition. New York and London: Garland Science. ISBN 0-8153-3218-1. (2002).

[6] J.S. Richardson. The anatomy and taxonomy of protein structure- Advances in protein chemistry. 1981 - Elsevier. (1981).

[7] D. Kennedy & C. Norman. Editorial: So much more to know. Science. 309: 78102. (2005).

[8] L.J. McGuffin, K. Bryson & D.T. Jones . The PSIPRED protein structure prediction server. Bioinformatics, Volume 16, Issue 4, 1 April 2000, Pages 404405, https://doi.org/10.1093/bioinformatics/16.4.404 (2000)

[9] J.J. Ward, L.J. McGuffin, K. Bryson, B.F. Buxton & D.T. Jones . The DISOPRED server for the prediction of protein disorder. Bioinformatics. 2004 Sep 1;20(13):2138-9. Epub 2004 Mar 25. (2004)

[10] J.J. Ward, L.J. McGuffin, J.S. Sodhi, B.F. Buxton & D.T. Jones . Prediction and functional analysis of native disorder in proteins from the three kingdoms of life. J Mol Biol. 2004 Mar 26;337(3):635-45.

[11] D.T. Jones & D. Cozzetto. DIOSPRED3: precise disordered region predictions with annotated protein-binding activity. Bioinformatics. 2015 Mar 15;31(6):857-63. doi: 10.1093/bioinformatics/btu744. Epub 2014 Nov 12. (2015)

[12] F. Sanger. The arrangement of amino acids in proteins. Adv Protein Chem. 1952;7:1-67. (1952)

[13] K.U. Linderstrm-Lang. Lane Medical Lectures: Proteins and Enzymes. Stanford University Press. (1952). p. 115. ASIN B0007J31SC. (1952)

[14] Schellman J.A. & Schellman C.G. "Kaj Ulrik Linderstrm-Lang (18961959)". Protein Sci. 6 (5): 1092100. (1997).

[15] C.N. Pace & J.M. Scholtz. A helix propensity scale based on experimental studies of peptides and proteins. Biophys J. 1998 Jul; 75(1): 422427. (1998)

[16] K.D. Berndt. No Title cryst.bbk.ac.uk/PPS2/course/section8/ss-960531_10.html (1996).

[17] J. Clark The Structure Of Proteins chemguide.co.uk/organicprops/aminoacids/proteinstruct.html (2016)

[18] D. Cozzetto*et al.* FFPred 3: Feature-Based Function Prediction for All Gene Ontology Domains. Scientific Reports 6 (2016): 31865. PMC. (2016).

[19] N.V. Chawla *et al.* SMOTE: Synthetic Minority Over-sampling Technique Journal of Artificial Intelligence. Research 16 (2002) 321357. (2002)

[20] IUPAC Commission on the Nomenclature of Organic Chemistry (CNOC) and IUPAC-IUB Commission on Biochemical Nomenclature (CBN). Nomenclature of -Amino Acids Recommendations 1974, Biochem. J. 149, 1-16 (1975); Biochemistry, 14, 449-462 (1975); Eur. J. Biochem. 53, 1-14 (1975).

[21] J. Kyte & R.F. Dolittle. A simple method for displaying the hydropathic character of a protein. J Mol Biol. 1982 May 5;157(1):105-32. (1982)

[22] I. Hunt Isoelectric Point chem.ucalgary.ca/courses/351/Carey5th/Ch27/ch27-1-4.html (2018)

[23] J. Kiraga *et al.* The relationships between the isoelectric point and: length of proteins, taxonomy and ecology of organisms. BMC Genomics. 2007; 8: 163. Published online 2007 Jun 12. doi: 10.1186/1471-2164-8-163 (2007)

[24] The Biology Project Department of Biochemistry and Molecular Biophysics, University of Arizona, August 25, 2003 http://www.biology.arizona.edu/biochemistry/problem_sets/aa/Aromatic.html (2003)

[25] Lobry JR, C. Gautier. Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 Escherichia coli chromosome-encoded genes. Nucleic Acids Res. 1994 Aug 11;22(15):3174-80. (1994)

[26] Vihinen M, Torkkila E, Riikonen P. Accuracy of protein flexibility predictions. Proteins. 1994 Jun;19(2):141-9.

[27] D. Jones. Organelle Prediction Project Dataset cs.ucl.ac.uk/staff/D.Jones/coursework/ (2018)

[28] D. Lawson. A Brief Introduction to Protein Crystallography by Dave Lawson jic.ac.uk/staff/david-lawson/xtallog/summary.htm

[29] M.S. Smyth & J.H.J. Martin. X-Ray Crystallography Mol Pathol. 2000 Feb; 53(1): 814. (2000)

[30] E. Fischer. Einfluss der Configuration auf die Wirkung der Enzyme Ber. Dtsch. Chem. Ges., 27: 2985-2993. doi:10.1002/cber.18940270364 (1894)

[31] F. Karush. Heterogeneity of the Binding Sites of Bovine Serum Albumin-1 J. Am. Chem. Soc., 1950, 72 (6), pp 27052713. DOI: 10.1021/ja01162a099 (1950)

[32] F. Karush. The Interaction of Optically Isomeric Dyes with Bovine Serum Albumin. The Journal of Physical Chemistry. 56. 10.1021/j150493a015. (1952)

[33] V.N. Uversky, C.J. Oldfield & A.K. Dunker. Intrinsically disordered proteins in human diseases: introducing the D2 concept. Annu Rev Biophys. 2008;37:215-46. doi: 10.1146/annurev.biophys.37.032807.125924. (2008).

[34] C.A. Kerfeld, M.R. Sawaya,S. Tanaka,C.V. Nguyen,M. Phillips,M. Beeby & T.O. Yeates . Protein structures forming the shell of primitive bacterial organelles. Science. 2005 Aug 5;309(5736):936-8. (2005).

[35] P. Horton, K-J. Park, T. Obayashi, N. Fujita, H. Harada, C.J. Adams-Collier & K. Nakai. WoLF PSORT: protein localization predictor. Nucleic Acids Research, Volume 35, Issue suppl_2, 1 July 2007, Pages W585W587, https://doi.org/10.1093/nar/gkm259. (2007).

[36] J.L. Gardy, C. Spencer, K. Wang, M. Ester, G.E. Tusndy, I. Simon, S. Hua, K. deFays, C. Lambert, K. Nakai & F.S.L. Brinkman. PSORT-B: improving protein subcellular localization prediction for Gram-negative bacteria. Nucleic Acids Research, Volume 31, Issue 13, 1 July 2003, Pages 36133617, https://doi.org/10.1093/nar/gkg602. (2003).

[37] J.R. Wagner, M.R. Laird, G. Melli, S. Rey, R. Lo, P. Dao, S.C. Sahinalp, M. Ester, L.J. Foster & F.S.L. Brinkman PSORTb 3.0: improved protein subcellular localization prediction with refined localization subcategories and predictive capabilities for all prokaryotes. Bioinformatics, Volume 26, Issue 13, 1 July 2010, Pages 16081615, https://doi.org/10.1093/bioinformatics/btq249. (2010).

[38] S. Rey ,*et al.* Assessing the precision of high-throughput computational and laboratory approaches for the genome-wide identification of protein subcellular localization in bacteria. BMC Genomics. 2005a;6:162.(2005).

[39] S.F. Altschul, T.L. Madden, A.A. Schffer, J. Zhang, Z. Zhang, W. Miller & D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Research, Volume 25, Issue 17, 1 September 1997, Pages 33893402, https://doi.org/10.1093/nar/25.17.3389. (1997).

[40] S.F-1 Altschul ,W. Gish, W. Miller, E.W. Myers & D.J. Lipman. Basic local alignment search tool. J Mol Biol. 1990 Oct 5;215(3):403-10. (1990).

[41] J.C.Wootton & S. Federhen. Statistics of local complexity in amino acid sequences and sequence databases. Computers & Chemistry,Volume 17, Issue 2,1993, Pages 149-163, ISSN 0097-8485, https://doi.org/10.1016/0097-8485(93)85006-X. (1993).

[42] B-1. Monastyrskyy, A. Kryshtafovych, J. Moult, A. Tramontano & K. Fidelis. Assessment of protein disorder region predictions in CASP10. Proteins. 2014 Feb;82 Suppl 2:127-37. doi: 10.1002/prot.24391. Epub 2013 Nov 22. (2014).

[43] B. Mszros, I. Simon & Z. Dosztnyi ANCHOR: web server for predicting protein binding regions in disordered proteins. Bioinformatics, Volume 25, Issue 20, 15 October 2009, Pages 27452746, https://doi.org/10.1093/bioinformatics/btp518. (2009).

[44] M. Sickmeier *et al.* DisProt: the Database of Disordered Proteins. Nucleic Acids Research, Volume 35, Issue suppl_1, 1 January 2007, Pages D786D793, https://doi.org/10.1093/nar/gkl893. (2007).

[45] D. Piovesan *et al.* DisProt 7.0: a major update of the database of disordered proteins. Nucleic Acids Research, Volume 45, Issue D1, 4 January 2017, Pages D219D227, https://doi.org/10.1093/nar/gkw1056. (2017).

[46] B. Boeckmann *et al.* The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. Nucleic Acids Research, Volume 31, Issue 1, 1 January 2003, Pages 365370, https://doi.org/10.1093/nar/gkg095. (2003).

[47] D. Piovesan *et al.* MobiDB 3.0: more annotations for intrinsic disorder, conformational diversity and interactions in proteins. Nucleic Acids Research, Volume 46, Issue D1, 4 January 2018, Pages D471D476, https://doi.org/10.1093/nar/gkx1071 (2018).

[48] F. Pedregosa *et al.* Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12 (2011) 2825-2830. (2011).

[49] G. Lematre, F. Nogueira & C.K. Aridas. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning Journal of Machine Learning Research 18 (2017) 1-5. (2017).

[50] S. Datla. COMPM058 Bioinformatics Coursework Submission 2018

Appendix A

Prediction Tables & Figures

Within this appendix, the figures and tables for organelle prediction and disordered region prediction can be found. These items are not present in the main body of the report to reduce on page clutter, but I have included them in the appendix as proof of the tests ran to gather information about optimal features and hyperparameter testing.

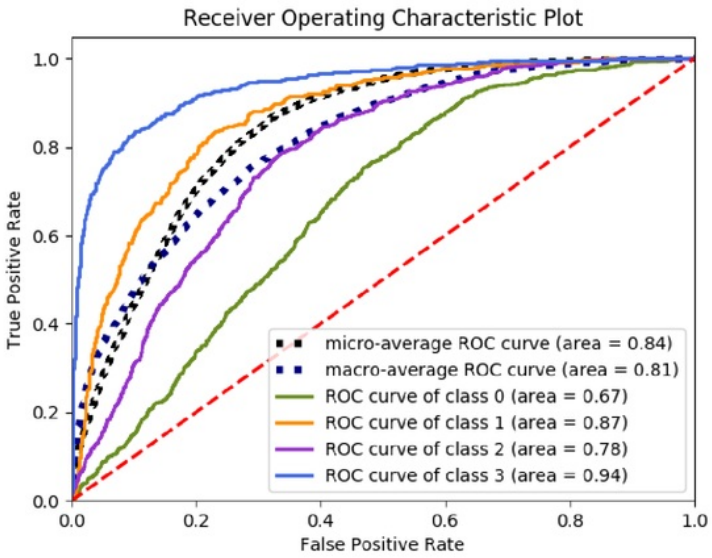

Figure A.1: Receiver Operating Characteristic Curve plot for AdaBoost Classifier organelle predictor

Table A.1: Table displaying the top 50 most important features for subcellular localisation prediction [50]

Feature Index	Feature Label	Rank
1	Sequence Length	1
2	Amino Acid Composition (Global) Pos:R	1
3	Amino Acid Composition (Global) Pos:S	1
4	Amino Acid Composition (Global) Pos:T	1
5	Amino Acid Composition (Global) Pos:V	1
6	Amino Acid Composition (Global) Pos:W	1
7	Amino Acid Composition (Global) Pos:Y	1
8	Amino Acid Composition (First 50) Pos:A	1
9	Amino Acid Composition (First 50) Pos:D	1
10	Amino Acid Composition (First 50) Pos:E	1
11	Amino Acid Composition (Global) Pos:Q	1
12	Secondary Structure Coil	1
13	Amino Acid Composition (First 50) Pos:M	1
14	Amino Acid Composition (First 50) Pos:R	1
15	Amino Acid Composition (First 50) Pos:S	1
16	Amino Acid Composition (First 50) Pos:V	1
17	Amino Acid Composition (Last 50) Pos:C	1
18	Amino Acid Composition (Last 50) Pos:G	1
19	Amino Acid Composition (Last 50) Pos:K	1
20	Amino Acid Composition (Last 50) Pos:L	1
21	Secondary Struct Helix	1
22	Amino Acid Composition (First 50) Pos:L	1
23	Amino Acid Composition (Global) Pos:P	1
24	Secondary Struct Sheet	1
25	Gravy	1
26	Molecular Weight (Last 50)	1
27	Molecular Weight (First 50)	1
28	Amino Acid Composition (Global) Pos:D	1
29	Amino Acid Composition (Global) Pos:E	1
30	Molecular Weight (Global)	1
31	Mean Flexibility (First 50)	1
32	Mean Flexibility (Global)	1
33	Isoelectricity	1
34	Amino Acid Composition (Global) Pos:H	1
35	Amino Acid Composition (Global) Pos:I	1
36	Aromacity	1
37	Amino Acid Composition (Global) Pos:K	1
38	Amino Acid Composition (Global) Pos:L	1
39	Amino Acid Composition (Global) Pos:M	1
40	Amino Acid Composition (Global) Pos:G	1
41	Amino Acid Composition (Global) Pos:C	1
42	Amino Acid Composition (Global) Pos:F	42
43	Amino Acid Composition (Global) Pos:A	43
44	Amino Acid Composition (Global) Pos:N	44
45	Amino Acid Composition (Last 50) Pos:E	45
46	Amino Acid Composition (First 50) Pos:G	46
47	Amino Acid Composition (First 50) Pos:K	47
48	Amino Acid Composition (Last 50) Pos:S	48
49	Amino Acid Composition (Last 50) Pos:V	49
50	Amino Acid Composition (First 50) Pos:F	50

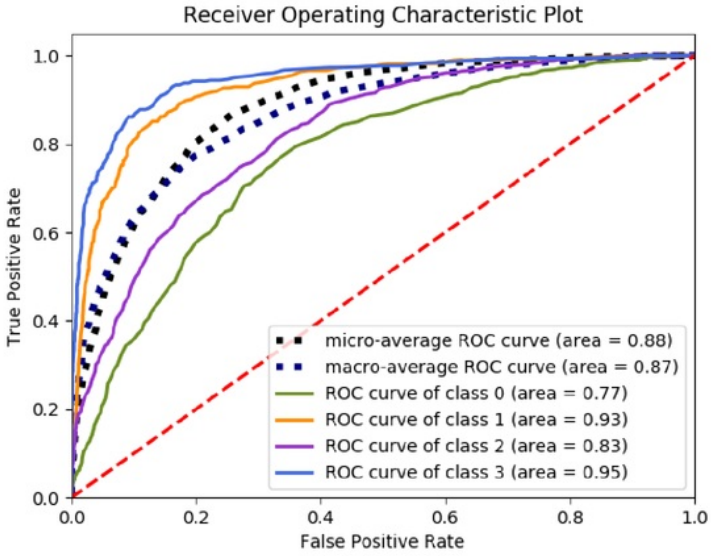

Figure A.2: Receiver Operating Characteristic Curve plot for Balanced Bagging Meta-estimator organelle predictor

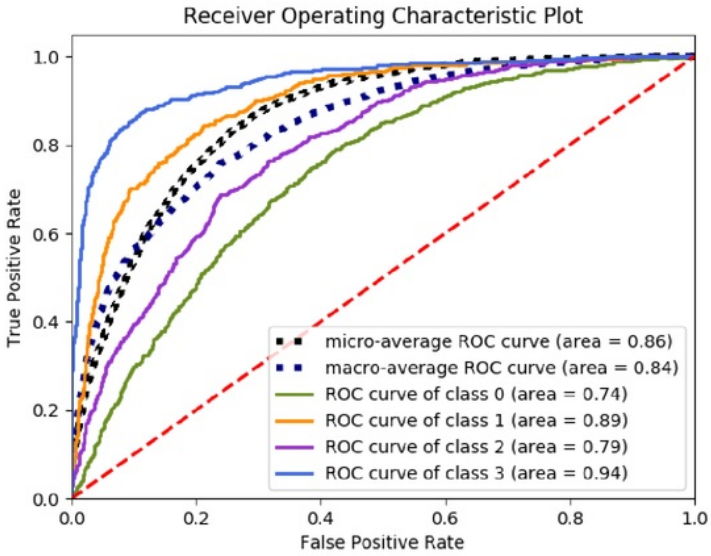

Figure A.3: Receiver Operating Characteristic Curve plot for Logistic Classifier organelle predictor

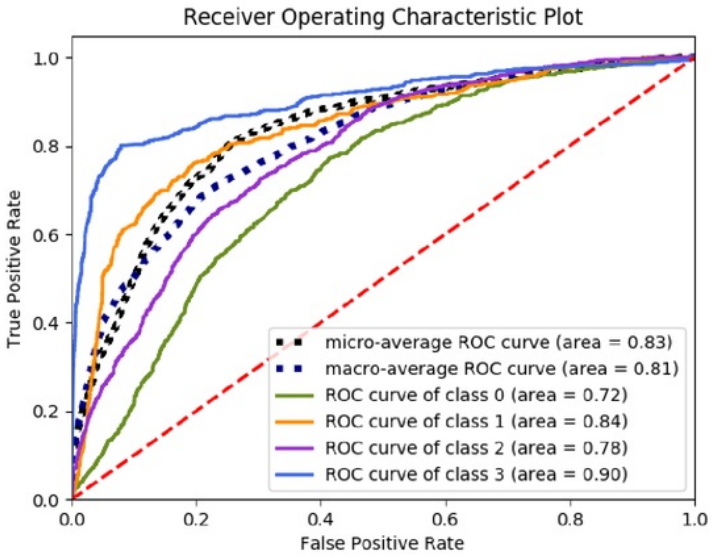

Figure A.4: Receiver Operating Characteristic Curve plot for Multi-Layered Perceptron organelle predictor

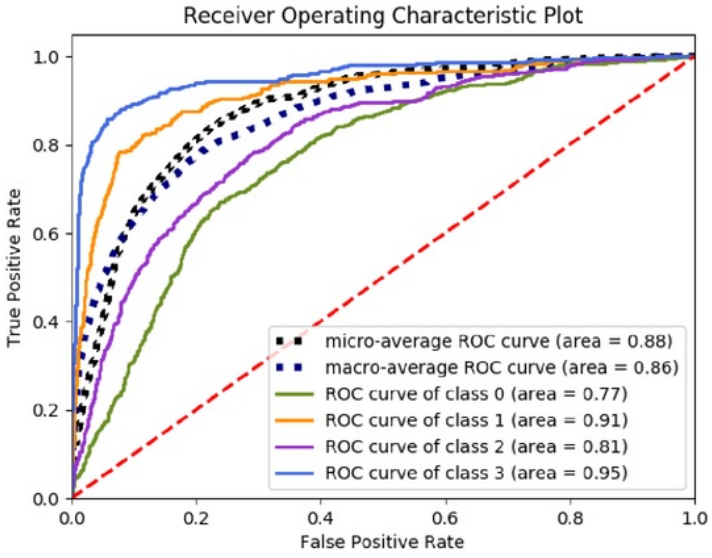

Figure A.5: Receiver Operating Characteristic Curve plot for Support Vector Classifier organelle predictor

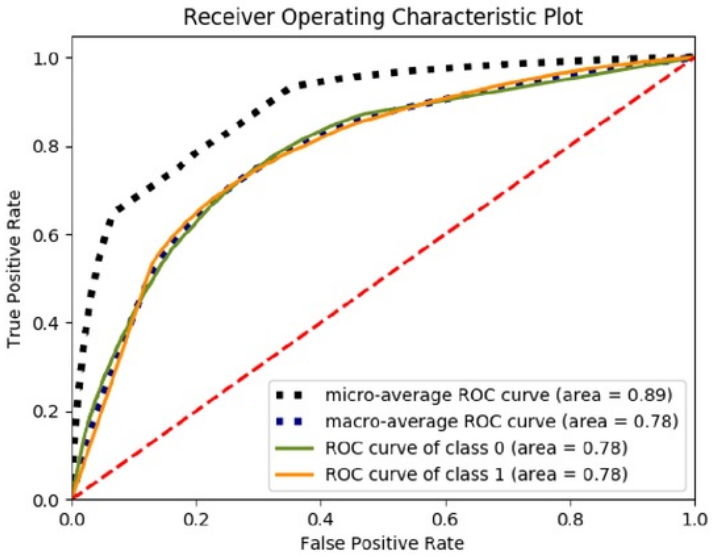

Figure A.6: Receiver Operating Characteristic Curve plot for Multi-Layered Perceptron disordered region predictor

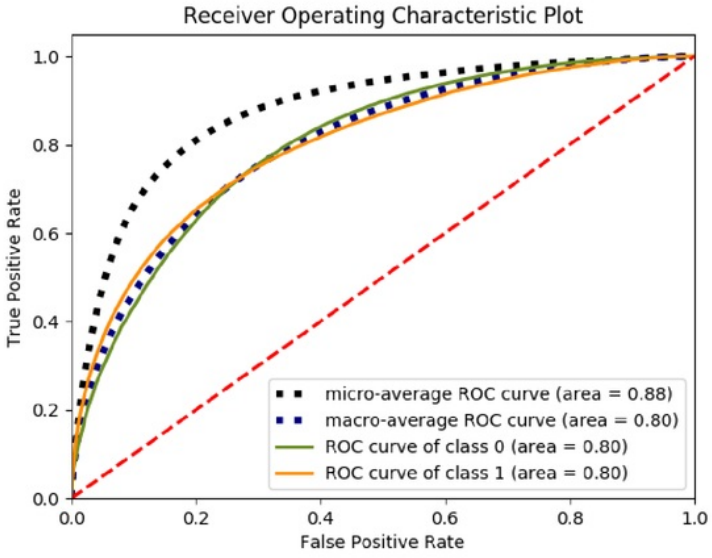

Figure A.7: Receiver Operating Characteristic Curve plot for Bagging Meta-estimator disordered region predictor

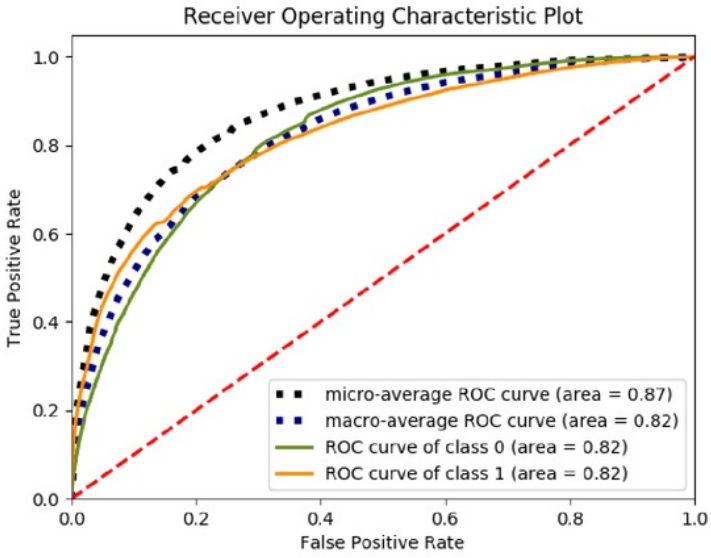

Figure A.8: Receiver Operating Characteristic Curve plot for Balanced Bagging Meta-estimator (with weak MLP estimators) disordered region prediction

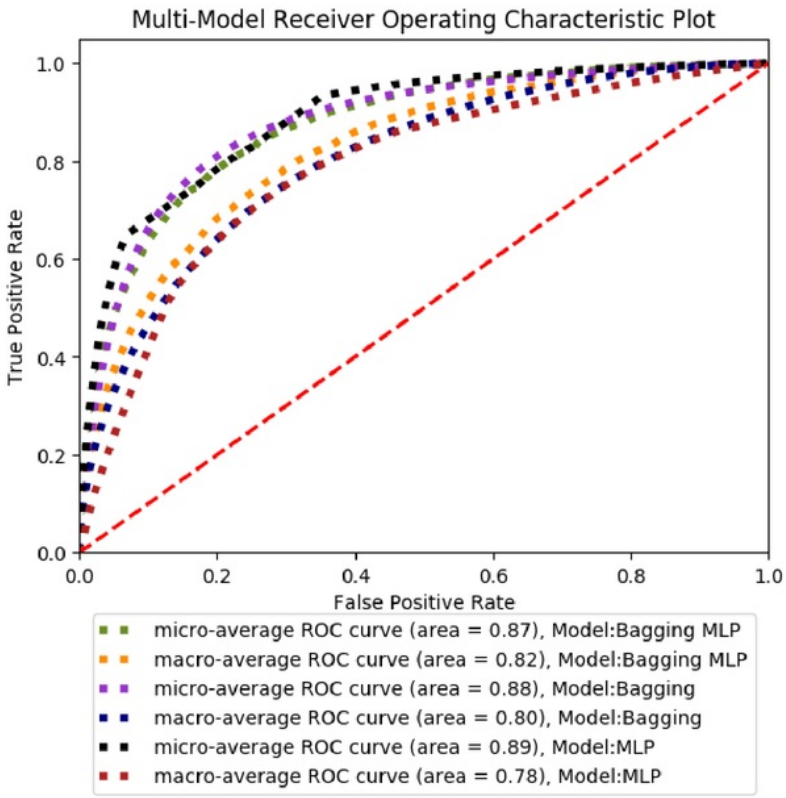

Figure A.9: Receiver Operating Characteristic Curve plot for all three basic disordered region predictors - Multi-Layered Perceptron (MLP), Balanced Bagging Meta-estimator (Bagging), Balanced Bagging Meta-estimator with MLP (Bagging MLP)

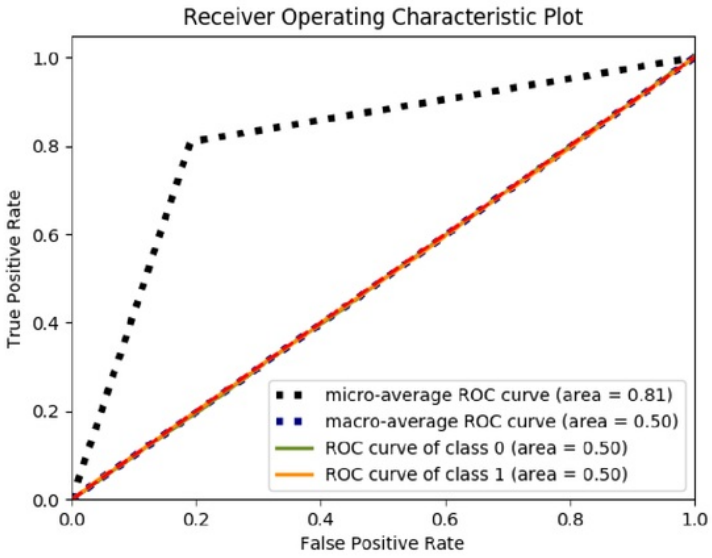

Figure A.10: Receiver Operating Characteristic Curve plot for Multi-Layered Perceptron disordered region predictor with organelle predictor intermediate

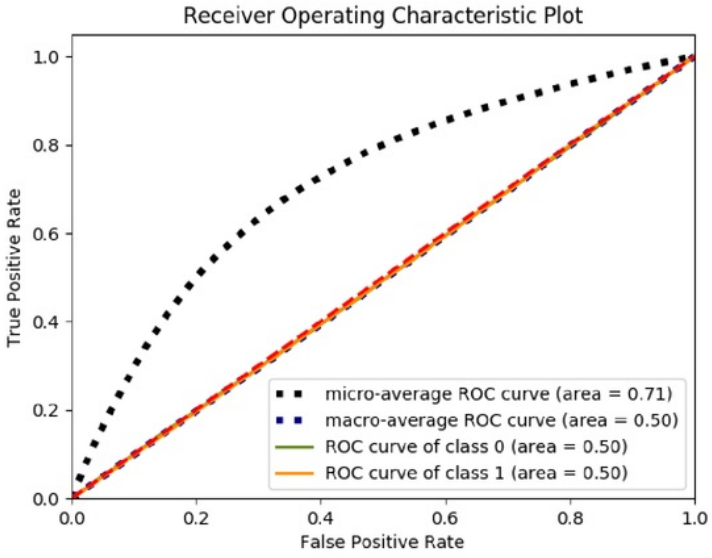

Figure A.11: Receiver Operating Characteristic Curve plot for Balanced Bagging Meta-estimator disordered region predictor with organelle predictor intermediate

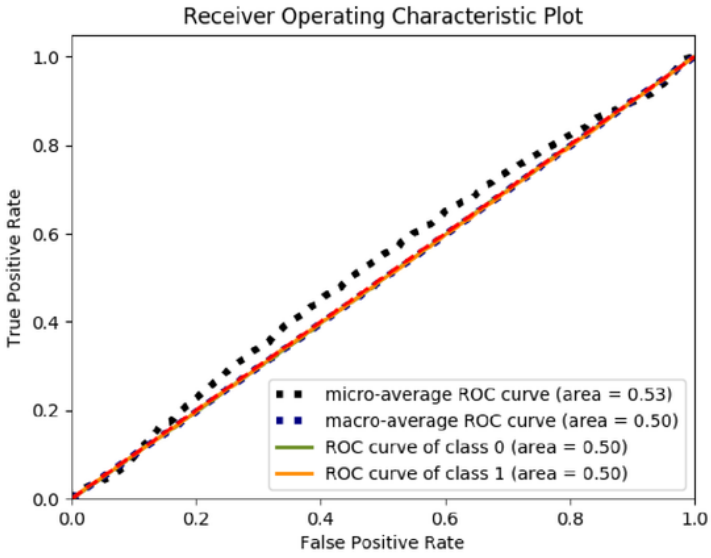

Figure A.12: Receiver Operating Characteristic Curve plot for Balanced Bagging Meta-estimator (with weak MLP estimators) disordered region prediction with organelle predictor intermediate

Appendix B

Experimental Testing Results

Please find attached overleaf the experimental results gathered from testing.

Appendix B:
Disordered Prediction Results

The report below displays the results of running the neural network across a variety of input sizes and hidden layer networks. The results are formatted as such:

- Input Parameters : The size of the sliding window, and whether the input data was one hot encoded (OHE) or PSI-Blast profiles (PSSM)
- Neural Network Layer Setup: Number of nodes at each layer of the network
- Confusion Matrix: Confusion matrix for the results, the first row (0) portraying the prediction of structured residues and the second row (1) portraying disordered residues prediction

Input Parameters	Hidden Layer Setup	Confusion Matrix	Precision	Recall	F1-Score
17 OHE	357x357x1	[[1252 794] [102 61]]	0 - 0.92 1 - 0.07	0 - 0.61 1 - 0.37	0 - 0.74 1 - 0.12
19 OHE	399x399x1	[[2046 0] [163 0]]	0 - 0.93 1 - 0.00	0 - 1.00 1 - 0.00	0 - 0.96 1 - 0.00
21 OHE	441x441x1	[[1263 783] [103 60]]	0 - 0.92 1 - 0.07	0 - 0.62 1 - 0.37	0 - 0.74 1 - 0.12
23 OHE	483x483x1	[[1263 783] [106 57]]	0 - 0.92 1 - 0.07	0 - 0.62 1 - 0.35	0 - 0.74 1 - 0.11
25 OHE	525x525x1	[[1257 789] [102 61]]	0 - 0.92 1 - 0.07	0 - 0.61 1 - 0.37	0 - 0.74 1 - 0.12

Table B.1: Table showing the prediction metrics from One-Hot-Encoded DISOPRED3 testing data using a Multi-Layered Perceptron.

Input Parameters	Hidden Layer Setup	Confusion Matrix	Precision	Recall	F1-Score
21 OHE	441x441x1	[[1263 783] [103 60]]	0 - 0.92 1 - 0.07	0 - 0.62 1 - 0.37	0 - 0.74 1 - 0.12
21 OHE	441x2	[[1261 785] [108 55]]	0 - 0.92 1 - 0.07	0 - 0.62 1 - 0.34	0 - 0.74 1 - 0.11

Table B.2: Table displaying the prediction metrics from different MLP topologies using the same One-Hot-Encoding DISOPRED3 testing data.

Input Parameters	Hidden Layer Setup	Confusion Matrix	Precision	Recall	F1-Score
21 OHE	441x2	[[1261 785] [108 55]]	0 - 0.92 1 - 0.07	0 - 0.62 1 - 0.34	0 - 0.74 1 - 0.11
7 PSSM	140x2	[[1834 212] [160 3]]	0 - 0.92 1 - 0.01	0 - 0.90 1 - 0.02	0 - 0.91 1 - 0.02
9 PSSM	180x2	[[1632 414] [158 5]]	0 - 0.91 1 - 0.01	0 - 0.80 1 - 0.03	0 - 0.85 1 - 0.02
11 PSSM	220x2	[[1661 385] [160 3]]	0 - 0.91 1 - 0.01	0 - 0.81 1 - 0.02	0 - 0.86 1 - 0.01
13 PSSM	260x2	[[1535 511] [144 19]]	0 - 0.91 1 - 0.04	0 - 0.75 1 - 0.12	0 - 0.82 1 - 0.05
15 PSSM	300x2	[[1539 507] [153 10]]	0 - 0.91 1 - 0.04	0 - 0.75 1 - 0.06	0 - 0.82 1 - 0.03
17 PSSM	340x2	[[1540 506] [140 23]]	0 - 0.92 1 - 0.04	0 - 0.75 1 - 0.14	0 - 0.83 1 - 0.07
19 PSSM	380x2	[[1434 612] [133 30]]	0 - 0.92 1 - 0.05	0 - 0.70 1 - 0.18	0 - 0.79 1 - 0.07
21 PSSM	420x2	[[1442 604] [128 35]]	0 - 0.92 1 - 0.05	0 - 0.70 1 - 0.20	0 - 0.80 1 - 0.09

Table B.3: Table displaying the prediction metrics from building and testing against PSSM DISOPRED3 testing data and different sliding window sizes.

Model	Solver	Confusion Matrix	Precision	Recall	F1-Score
Mulit-Layered Perceptron	MLP Weak Estimator	[[98748 9801 0] [13265 12384 0] [28 2 0]]	0 – 0.88 1 – 0.56 2 – 0.00	0 – 0.91 1 – 0.48 2 – 0.0	0 – 0.90 1 – 0.52 2 – 0.0
Mulit-Layered Perceptron	Adam Solver	[[98748 9801 0] [13265 12384 0] [28 2 0]]	0 – 0.88 1 – 0.56 2 – 0.00	0 – 0.91 1 – 0.48 2 – 0.0	0 – 0.90 1 – 0.52 2 – 0.0
Mulit-Layered Perceptron	LBFGS Solver	[[100675 7874 0] [14436 11213 0] [29 1 0]]	0 – 0.87 1 – 0.59 2 – 0.00	0 – 0.93 1 – 0.44 2 – 0.0	0 – 0.90 1 – 0.50 2 – 0.0
Balanced Bagging Classifier	MLP Weak Estimator	[[97014 11535 0] [12675 12974 0] [28 2 0]]	0 – 0.88 1 – 0.53 2 – 0.00	0 – 0.89 1 – 0.51 2 – 0.0	0 – 0.89 1 – 0.52 2 – 0.0
SVM	N/A	[[81522 26497 530] [8909 16687 53] [9 0 21]]	0 – 0.9 1 – 0.39 2 – 0.03	0 – 0.75 1 – 0.65 2 – 0.7	0 – 0.65 1 – 0.76 2 – 1.00

Table B.4: Table displaying the prediction metrics from including context-dependent labels for testing purposes. Comparison of Different MLP solvers was included to see if any higher accuracies could be achieved by using a different solver.

Model	Matthew's Correlation Coefficient	Confusion Matrix	Precision	Recall	F1-Score
Balanced Bagging Classifier (MLP weak estimator)	0.433	[[89116 19463] [8469 17180]]	0 – 0.91 1 – 0.41	0 – 0.82 1 – 0.67	0 – 0.67 1 – 0.82
Balanced Bagging Classifier (200 Weak Estimators)	0.407	[[93694 14885] [11140 14509]]	0 – 0.89 1 – 0.49	0 – 0.86 1 – 0.57	0 – 0.57 1 – 0.86
Mulit-Layered Perceptron	0.395	[[91217 17362] [10507 15142]]	0 – 0.90 1 – 0.47	0 – 0.84 1 – 0.59	0 – 0.59 1 – 0.84
Balanced Bagging Classifier (MLP weak estimator) + Organelle Prediction	-0.004	[[59077 49502] [14070 11579]]	0 – 0.81 1 – 0.19	0 – 0.54 1 – 0.45	0 – 0.45 1 – 0.54
Balanced Bagging Classifier (200 Weak Estimators) + Organelle Prediction	-0.004	[[84463 24116] [20072 5577]]	0 – 0.81 1 – 0.19	0 – 0.78 1 – 0.22	0 – 0.22 1 – 0.78
Mulit-Layered Perceptron + Organelle Prediction	0.0	[[108579 0] [25649 0]]	0 – 0.81 1 – 0.0	0 – 1.00 1 – 0.00	0 – 0.00 1 – 1.00
SVM + Organelle Prediction	-0.003	[[59723 48856] [14190 11459]]	0 – 0.81 1 – 0.19	0 – 0.55 1 – 0.45	0 – 0.45 1 – 0.55

Table B.5: Table displaying complex disordered region predictors (first 3 rows) and combined organelle + disordered region predictor (last 4 rows). The validation set from the original DisProt dataset has been used for evaluation.

Model	Matthew's Correlation Coefficient	Confusion Matrix	Precision	Recall	F1-Score
Balanced Bagging Classifier (MLP weak estimator)	0.099	[[1091 955] [56 107]]	0 – 0.95 1 – 0.10	0 – 0.53 1 – 0.66	0 – 0.66 1 – 0.53
Balanced Bagging Classifier (200 Weak Estimators)	-0.078	[[1274 772] [125 38]]	0 – 0.91 1 – 0.05	0 – 0.62 1 – 0.23	0 – 0.23 1 – 0.62
Mulit-Layered Perceptron	0.142	[[929 1117] [30 133]]	0 – 0.97 1 – 0.11	0 – 0.45 1 – 0.82	0 – 0.82 1 – 0.45
Balanced Bagging Classifier (MLP weak estimator) + Organelle Prediction	-0.206	[[680 1366] [116 47]]	0 – 0.85 1 – 0.03	0 – 0.33 1 – 0.29	0 – 0.29 1 – 0.33
Balanced Bagging Classifier (200 Weak Estimators) + Organelle Prediction	0.025	[[1648 398] [125 38]]	0 – 0.93 1 – 0.09	0 – 0.81 1 – 0.23	0 – 0.23 1 – 0.81
Mulit-Layered Perceptron + Organelle Prediction	0.0	[[2046 0] [163 0]]	0 – 0.93 1 – 0.00	0 – 1.00 1 – 0.00	0 – 0.00 1 – 1.00
SVM + Organelle Prediction	-0.013	[[668 1378] [57 106]]	0 – 0.92 1 – 0.07	0 – 0.33 1 – 0.65	0 – 0.65 1 – 0.33

Table B.6: Table displaying complex disordered region predictors (first 3 rows) and combined organelle + disordered region predictor (last 4 rows). The testing set used here is the DISOPRED3 testing set.

Model	Matthew's Correlation Coefficient	Confusion Matrix	Precision	Recall	F1-Score
Balanced Bagging Classifier (MLP weak estimator)	0.270	[[21799 2069] [839 663]]	0 – 0.96 1 – 0.24	0 – 0.91 1 – 0.44	0 – 0.44 1 – 0.91
Balanced Bagging Classifier (200 Weak Estimators)	0.270	[[22975 893] [1063 439]]	0 – 0.96 1 – 0.33	0 – 0.96 1 – 0.29	0 – 0.29 1 – 0.96
Mulit-Layered Perceptron	0.233	[[20744 3124] [773 729]]	0 – 0.96 1 – 0.19	0 – 0.87 1 – 0.49	0 – 0.49 1 – 0.87
Balanced Bagging Classifier (MLP weak estimator) + Organelle Prediction	-0.091	[[12620 11248] [1083 419]]	0 – 0.92 1 – 0.04	0 – 0.53 1 – 0.28	0 – 0.28 1 – 0.53
Balanced Bagging Classifier (200 Weak Estimators) + Organelle Prediction	-0.024	[[18387 5481] [1222 280]]	0 – 0.94 1 – 0.05	0 – 0.77 1 – 0.19	0 – 0.19 1 – 0.76
Mulit-Layered Perceptron + Organelle Prediction	0.0	[[23868 0] [1502 0]]	0 – 0.94 1 – 0.0	0 – 1.00 1 – 0.00	0 – 0.00 1 – 1.00
SVM + Organelle Prediction	-0.041	[[14052 9816] [1013 489]]	0 – 0.93 1 – 0.05	0 – 0.59 1 – 0.33	0 – 0.33 1 – 0.59

Table B.7: Table displaying complex disordered region predictors (first 3 rows) and combined organelle + disordered region predictor (last 4 rows). The testing set used here is the CASP10 disorder target data.

Appendix C

Source Code

Please find all written source code for this Thesis overleaf. The following Python packages were imported as part of the project: Scipy, Numpy, Scikit-Learn, Imbalance-Learn, BioPython. Instructions for running the code and the original code can be found at the github repository for this project: https://github.com/s-datla/ThesisProject.

Listings

Listing C.1: Disorder Region Prediction Model

```
# Standard imports
import sys
import io
import string
import json
import math

# Utility imports
from collections import Counter
from pprint import pprint
import numpy as np
import scipy.sparse
from scipy import interp
from sklearn.externals import joblib
from sklearn.model_selection import train_test_split
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as pyplot
from itertools import cycle
from organelle import buildPredict

from Bio import SeqIO

# Metric calculation functions
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, matthews_corrcoef, roc_curve, auc
from sklearn.model_selection import cross_val_score
from imblearn.metrics import classification_report_imbalanced

# Classification imports
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import BaggingClassifier
from imblearn.ensemble import BalancedBaggingClassifier


'''
    predict_model.py is the main Python file which generates disorder prediction
    models for testing and runs all tests.
    Models are saved in '.pkl' files and can be reloaded for future prediction
    ROC plots are also generated during training
'''

windowSize = 7

def main():
    global windowSize
    if len(sys.argv) > 4:
        print("Invalid Entry!")
        print("Expect arguments in format <python sieveson.py JSONFILE function>")
    else:
        if (sys.argv[2] == 'mtxBuild' and len(sys.argv) == 4):
            windowSize = int(sys.argv[3])
            mtxBuild(sys.argv[1])
```

```
        elif (sys.argv[2] == 'build' and len(sys.argv) == 4):
            windowSize = int(sys.argv[3])
            buildModel(sys.argv[1])
        elif(sys.argv[2] == 'bagClassify' and len(sys.argv) == 4):
            windowSize = int(sys.argv[3])
            bagClassify(sys.argv[1])
        elif(sys.argv[2] == 'organelle' and len(sys.argv) == 4):
            windowSize = int(sys.argv[3])
            buildOrganelle(sys.argv[1])
        elif(sys.argv[2] == 'predict'):
            predictModel(sys.argv[1])
        elif(sys.argv[2] == 'predictMTX'):
            predictMTX(sys.argv[1])
        elif(sys.argv[2] == 'predictOrganelle' and len(sys.argv) == 4):
            predictOrganelle(sys.argv[1],sys.argv[3])
        else:
            print("Invalid Entry!")
            print("Expect arguments in format <python sieveson.py JSONFILE function> ")


def buildModel(path):
    compressed = np.load(path)
    X = compressed['inputs']
    Y = compressed['labels']
    print X.shape, Y.shape
    '''
    Code for use later (can be diagonostic or reverting back to previous models / parameters)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6, random_state=0)
        model = LogisticRegression()
        model.fit(X,y)
        scores = cross_val_score(model, X, y, cv=10)
        print(scores)
        print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        joblib.dump(model, 'logistic.pkl')
    '''
    scaler = StandardScaler()
    print(scaler.fit(X))
    scaledTrainX = scaler.transform(X)
    model = MLPClassifier(hidden_layer_sizes=(windowSize*21,2), max_iter=500)
    model.fit(scaledTrainX,y)
    # scores = cross_val_score(model, scaledTrainX,y,cv=10)
    # print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
    print("Fitted Model!\n" + "Now saving model and scaler")
    joblib.dump(model, 'model.pkl')
    joblib.dump(scaler, 'scaler.pkl')

def mtxBuild(path):
    mtxCompressed = np.load(path)
    X = mtxCompressed['savedX']
    Y = mtxCompressed['savedY']
    print("Started scaling!")
    X = np.reshape(X,(X.shape[0],X.shape[1] * X.shape[2]))
    scaler = StandardScaler()
    print(scaler.fit(X))
    scaledTrainX = scaler.transform(X)
    print("Splitting into Test and Train")
```

```
    X_train, X_test, Y_train, Y_test = train_test_split(scaledTrainX, Y, test_size=0.33, random_state=19)
    model = MLPClassifier(hidden_layer_sizes=(windowSize*20,2), max_iter=500)

    model.fit(X_train,Y_train)
    print("Fitted_Model_!\n" +  "Now_saving_model_and_scaler")
    joblib.dump(model, 'nn_pssm.pkl')
    joblib.dump(scaler, 'scaler.pkl')
    predictedY = model.predict(X_test)
    print(confusion_matrix(Y_test,predictedY))
    print(classification_report(Y_test,predictedY))

def bagClassify(path):
    mtxCompressed = np.load(path)
    X = mtxCompressed['savedX']
    Y = mtxCompressed['savedY']
    print("Started_scaling!")
    X = np.reshape(X,(X.shape[0],X.shape[1] * X.shape[2]))
    scaler = StandardScaler()
    print(scaler.fit(X))
    scaledTrainX = scaler.transform(X)
    print("Splitting_into_Test_and_Train")
    X_train, X_test, Y_train, Y_test = train_test_split(scaledTrainX, Y, test_size=0.33, random_state=19)
    print("Training_Data_is_distributed_as_follows:_" + str(sorted(Counter(Y_train).items())))
    print("Testing_Data_is_distributed_as_follows:_" + str(sorted(Counter(Y_test).items())))


    '''
        The following models were used:
            Multi Layered Perceptron Classifier (Layout -  441,2)
            Balanced Bagging Meta Estimator (200 Weak Estimators)
            Balanced Baggin Meta Estimator (MLP Layout)
            Support Vector Classifier (Balanced class weights, OneVSRest)
    '''
    # std = MLPClassifier(hidden_layer_sizes=(windowSize*21,2),max_iter=500, solver='adam',random_state=19)
    # svc = SVC(class_weight='balanced',random_state=19,decision_function_shape='ovr')
    # bag = BalancedBaggingClassifier(n_estimators=200,random_state=19)
    # bbc = BalancedBaggingClassifier(base_estimator=MLPClassifier(hidden_layer_sizes=(windowSize*21,2),max_iter=500),ratio='auto',replacement=False,random_state=19)

    # Fitting All Models
    # bbc.fit(X_train,Y_train)
    # svc.fit(X_train,Y_train)
    # bag.fit(X_train,Y_train)
    # std.fit(X_train,Y_train)

    print("Fitted_Model_!\n" +  "Now_saving_model_and_scaler")

    # Saving fitted Models and scaler
    # joblib.dump(svc, 'svc.pkl')
    # joblib.dump(bbc, 'bag_model.pkl')
    # joblib.dump(bag,'bag.pkl')
    # joblib.dump(std,'std.pkl')
    # joblib.dump(scaler, 'bag_scaler.pkl')

    # Loading saved Models and scaler
    bbc = joblib.load('bag_model.pkl')
    bag = joblib.load('bag.pkl')
```

```
    std = joblib.load('std.pkl')
    # svc = joblib.load('svc.pkl')
    # scaler  = joblib.load('bag_scaler.pkl')

    # predY1 = svc.predict(X_test)
    predY1 = bbc.predict(X_test)
    predY2 = bag.predict(X_test)
    predY3 = std.predict(X_test)

    # Classification Metric display
    # print "SVC"
    # print(confusion_matrix(Y_test,predY1))
    # print(classification_report_imbalanced(Y_test,predY1))
    # print(matthews_corrcoef(Y_test, predY1))
    print "Balanced_Bagging_MLP"
    print(confusion_matrix(Y_test,predY1))
    print(classification_report_imbalanced(Y_test,predY1))
    print(matthews_corrcoef(Y_test, predY1))
    print "Balanced_Bagging"
    print(confusion_matrix(Y_test,predY2))
    print(classification_report_imbalanced(Y_test,predY2))
    print(matthews_corrcoef(Y_test, predY2))
    print "Standard_MLP"
    print(confusion_matrix(Y_test,predY3))
    print(classification_report_imbalanced(Y_test,predY3))
    print(matthews_corrcoef(Y_test, predY3))

    probs_bbc = bbc.predict_proba(X_test)
    probs_bag = bag.predict_proba(X_test)
    probs_std = std.predict_proba(X_test)
    # probs_svc = svc.predict_proba(X_test)

    ROCplot(probs_bbc,Y_test,"Plots/ROCplotBBC.png")
    ROCplot(probs_bag,Y_test,"Plots/ROCplotBAG.png")
    ROCplot(probs_std,Y_test,"Plots/ROCplotSTD.png")
    # ROCplot(probs_svc,Y_test,"ROCplotSVC.png")

    multiROCplot([probs_bbc,probs_bag,probs_std],Y_test,"Plots/multiROCplot.png",['Bagging_MLP','Bagging','MLP'])

    # bbc_probs_train = bbc.predict_proba(X_train)
    # bag_probs_train = bag.predict_proba(X_train)
    # std_probs_train = std.predict_proba(X_train)
    # svc_probs_train = std.predict_proba(X_train)
    # print len(std_probs_train), len(bag_probs_train), len(bbc_probs_train)


# Code used for ROC plots is derived a scikit-learn tutorial: http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
def multiROCplot(probs_list, Y_test,save,models):
    Y_test = np.array([[1 if x == 0 else 0,x] for x in Y_test])
    print probs.shape, Y_test.shape
    assert(len(models) == len(probs_list));
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for j in range(0,len(probs_list)):
        for i in range(0,2):
```

```
            fpr[(j,i)], tpr[(j,i)], _ = roc_curve(Y_test[:,i],probs_list[j][:,i])
            # fpr[(j,i)], tpr[(j,i)], _ = roc_curve(Y_test,[probs_list[j][k][i] for k in range(len(probs_list[0]))])
            roc_auc[(j,i)] = auc(fpr[(j,i)],tpr[(j,i)])
        fpr[(j,'micro')], tpr[(j,'micro')], _ = roc_curve(Y_test.ravel(), probs_list[j].ravel())
        roc_auc[(j,'micro')] = auc(fpr[(j,'micro')], tpr[(j,'micro')])
        all_fpr = np.unique(np.concatenate([fpr[(j,k)] for k in range(0,2)]))
        mean_tpr = np.zeros_like(all_fpr)
        for i in range(0,2):
            mean_tpr += interp(all_fpr,fpr[(j,i)],tpr[(j,i)])
        mean_tpr /= 2
        fpr[(j,'macro')] = all_fpr
        tpr[(j,'macro')] = mean_tpr
        roc_auc[(j,'macro')] = auc(fpr[(j,'macro')],tpr[(j,'macro')])
    lw = 2
    pyplot.figure()
    color_list = cycle([['olivedrab', 'darkorange'], ['darkorchid','navy'],['black','firebrick'],['gold','slategrey']])
    for j,colors in zip(range(0,len(probs_list)),color_list):
        pyplot.plot(fpr[(j,'micro')], tpr[(j,'micro')],
             label='micro-average_ROC_curve_(area_=_{0:0.2f}),_Model:{1}'.format(roc_auc[(j,'micro')],models[j]),color=colors[0], linestyle=':', linewidth=4)
        pyplot.plot(fpr[(j,'macro')], tpr[(j,'macro')],
                 label='macro-average_ROC_curve_(area_=_{0:0.2f}),_Model:{1}'.format(roc_auc[(j,'macro')],models[j]),color=colors[1], linestyle=':', linewidth=4)
    pyplot.plot([0, 1], [0, 1], 'r--', lw=lw)
    pyplot.xlim([0.0, 1.0])
    pyplot.ylim([0.0, 1.05])
    pyplot.xlabel('False_Positive_Rate')
    pyplot.ylabel('True_Positive_Rate')
    pyplot.title('Multi-Model_Receiver_Operating_Characteristic_Plot')
    l = pyplot.legend(loc='upper_center', bbox_to_anchor=(0.5,-0.1))
    # pyplot.show()
    pyplot.savefig(save, bbox_extra_artists=(l,), bbox_inches='tight')

# Code used for ROC plots is derived a scikit-learn tutorial: http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
def ROCplot(probs,Y_test,save):
    Y_test = np.array([[1 if x == 0 else 0,x] for x in Y_test])
    # Y_test = label_binarize(Y_test,classes=[0,1])
    print probs.shape, Y_test.shape
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(0,2):
        fpr[i], tpr[i], _ = roc_curve(Y_test[:,i],probs[:,i])
        roc_auc[i] = auc(fpr[i],tpr[i])

    fpr['micro'], tpr['micro'], _ = roc_curve(Y_test.ravel(), probs.ravel())
    roc_auc['micro'] = auc(fpr['micro'], tpr['micro'])

    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(0,2)]))
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(0,2):
        mean_tpr += interp(all_fpr,fpr[i],tpr[i])
    mean_tpr /= 2
    fpr['macro'] = all_fpr
    tpr['macro'] = mean_tpr
    roc_auc['macro'] = auc(fpr['macro'],tpr['macro'])

    lw = 2
```

```
    pyplot.figure()
    pyplot.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'.format(roc_auc["micro"]),color='black', linestyle=':', linewidth=4)
    pyplot.plot(fpr["macro"], tpr["macro"],
             label='macro-average ROC curve (area = {0:0.2f})'.format(roc_auc["macro"]),color='navy', linestyle=':', linewidth=4)

    colors = cycle(['olivedrab', 'darkorange', 'darkorchid'])
    for i, color in zip(range(0,2), colors):
        pyplot.plot(fpr[i], tpr[i], color=color, lw=lw,label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i]))
    pyplot.plot([0, 1], [0, 1], 'r--', lw=lw)
    pyplot.xlim([0.0, 1.0])
    pyplot.ylim([0.0, 1.05])
    pyplot.xlabel('False Positive Rate')
    pyplot.ylabel('True Positive Rate')
    pyplot.title('Receiver Operating Characteristic Plot')
    pyplot.legend(loc="lower right")
    # pyplot.show()
    pyplot.savefig(save)

def buildOrganelle(path):

    seqs = []
    lengths = []
    for seq_record in SeqIO.parse("../Data_Files/train_files/disprot.fa","fasta"):
        seq = str(seq_record.seq)
        seqs += [seq]
        lengths += [len(seq)]
    temp_org = np.array(buildPredict(seqs))
    probs_org = []
    for i in range(0,len(lengths)):
        probs_org += [temp_org[i]] * lengths[i]
    probs_org = np.array(probs_org)
    print "Len Probs Org: {} ".format(len(probs_org))
    print probs_org.shape
    mtxCompressed = np.load(path)
    X = mtxCompressed['savedX']
    Y = mtxCompressed['savedY']
    print("Started scaling!")
    X = np.reshape(X,(X.shape[0],X.shape[1] * X.shape[2]))
    scaler = StandardScaler()
    print(scaler.fit(X))
    scaledTrainX = scaler.transform(X)
    print("Splitting into Test and Train")
    X_train, X_test, Y_train, Y_test = train_test_split(scaledTrainX, Y, test_size=0.33, random_state=19)

    print("Training Data is distributed as follows: " + str(sorted(Counter(Y_train).items())))
    print("Testing Data is distributed as follows: " + str(sorted(Counter(Y_test).items())))

    # base = BalancedBaggingClassifier(base_estimator=MLPClassifier(hidden_layer_sizes=(windowSize*21,2), max_iter=500), ratio='auto', replacement=False, random_state=19)

    # base.fit(X_train, Y_train)
    # Saving fitted Models and scaler
    # joblib.dump(base, 'org_base.pkl')
    # joblib.dump(scaler, 'bag_scaler.pkl')

    # Loading saved Models and scaler
```

```
base = joblib.load('org_base.pkl')
# scaler  = joblib.load('bag_scaler.pkl')

base_probs = base.predict_proba(scaledTrainX)
print base_probs.shape
org_X = np.hstack((base_probs,probs_org))
split_index = len(Y_train)

org_scaler = StandardScaler()
print(org_scaler.fit(org_X))
scaled_org_train = org_scaler.transform(org_X[:split_index])
scaled_org_test = org_scaler.transform(org_X[split_index:])

# std = MLPClassifier(hidden_layer_sizes=(6,2),max_iter=500, solver='adam',random_state=19)
svc = SVC(class_weight='balanced',random_state=19,decision_function_shape='ovo')
# bag = BalancedBaggingClassifier(n_estimators=200,random_state=19)
# bbc = BalancedBaggingClassifier(base_estimator=MLPClassifier(hidden_layer_sizes=(6,2),max_iter=500),ratio='auto',replacement=False,random_state=19)
#
# std.fit(scaled_org_train,Y_train)
svc.fit(scaled_org_train,Y_train)
# bag.fit(scaled_org_train,Y_train)
# bbc.fit(scaled_org_train,Y_train)

# Saving fitted Models and scaler
joblib.dump(svc, 'org_svc.pkl')
# joblib.dump(bbc, 'org_bbc.pkl')
# joblib.dump(bag,'org_bag.pkl')
# joblib.dump(std,'org_std.pkl')
joblib.dump(org_scaler, 'org_scaler.pkl')

# Loading saved Models and scaler
bbc = joblib.load('org_bbc.pkl')
bag = joblib.load('org_bag.pkl')
std = joblib.load('org_std.pkl')
# svc = joblib.load('org_svc.pkl')
# org_scaler  = joblib.load('org_scaler.pkl')

predY1 = bbc.predict(scaled_org_test)
predY2 = bag.predict(scaled_org_test)
predY3 = std.predict(scaled_org_test)
predY4 = svc.predict(scaled_org_test)

# Classification Metric display
print "Balanced_Bagging_MLP"
print(confusion_matrix(Y_test,predY1))
print(classification_report_imbalanced(Y_test,predY1))
print(matthews_corrcoef(Y_test, predY1))
print "Balanced_Bagging"
print(confusion_matrix(Y_test,predY2))
print(classification_report_imbalanced(Y_test,predY2))
print(matthews_corrcoef(Y_test, predY2))
print "Standard_MLP"
print(confusion_matrix(Y_test,predY3))
print(classification_report_imbalanced(Y_test,predY3))
print(matthews_corrcoef(Y_test, predY3))
print "SVC"
```

```
    print(confusion_matrix(Y_test,predY4))
    print(classification_report_imbalanced(Y_test,predY4))
    print(matthews_corrcoef(Y_test, predY4))

    probs_bbc = bbc.predict_proba(scaled_org_test)
    probs_bag = bag.predict_proba(scaled_org_test)
    probs_std = std.predict_proba(scaled_org_test)
    probs_svc = svc.decision_function(scaled_org_test)

    ROCplot(probs_bbc,Y_test,"Plots/ROCplotBBC-org.png")
    ROCplot(probs_bag,Y_test,"Plots/ROCplotBAG-org.png")
    ROCplot(probs_std,Y_test,"Plots/ROCplotSTD-org.png")
    ROCplot(probs_svc,Y_test,"Plots/ROCplotSVC-org.png")

    multiROCplot([probs_bbc,probs_bag,probs_std,probs_svc],Y_test,"Plots/multiROCplot-org.png",['Bagging_MLP','Bagging','MLP','SVC'])

def predictModel(path):
    compressed = np.load(path)
    X = compressed['inputs']
    Y = compressed['labels']
    print X.shape, Y.shape
    print("Loading_model_and_scaler")
    model = joblib.load('model.pkl')
    scaler  = joblib.load('scaler.pkl')
    scaledTestX = scaler.transform(X)

    predictedY = model.predict(scaledTestX)
    print(confusion_matrix(Y,predictedY))
    print(classification_report(Y,predictedY))

def predictMTX(path):
    mtxCompressed = np.load(path)
    X = mtxCompressed['savedX']
    Y = mtxCompressed['savedY']
    print X.shape, Y.shape
    mod_x = np.reshape(X,(X.shape[0],X.shape[1] * X.shape[2]))
    print("Loading_model")
    bbc = joblib.load('bag_model.pkl')
    bag = joblib.load('bag.pkl')
    std = joblib.load('std.pkl')
    # svc = joblib.load('svc.pkl')
    scaler  = joblib.load('bag_scaler.pkl')
    # model = joblib.load('nn_pssm.pkl')
    scaledTestX = scaler.transform(mod_x)

    predY1 = bbc.predict(scaledTestX)
    predY2 = bag.predict(scaledTestX)
    predY3 = std.predict(scaledTestX)

    # Classification Metric display
    print "Balanced_Bagging_MLP"
    print(confusion_matrix(Y,predY1))
    print(classification_report_imbalanced(Y,predY1))
    print(matthews_corrcoef(Y, predY1))
    print "Balanced_Bagging"
    print(confusion_matrix(Y,predY2))
```

```
        print(classification_report_imbalanced(Y,predY2))
        print(matthews_corrcoef(Y, predY2))
        print "Standard_MLP"
        print(confusion_matrix(Y,predY3))
        print(classification_report_imbalanced(Y,predY3))
        print(matthews_corrcoef(Y, predY3))
        # print "SVC"
        # print(confusion_matrix(Y_test,predY4))
        # print(classification_report_imbalanced(Y_test,predY4))
        # print(matthews_corrcoef(Y_test, predY4))

    def predictOrganelle(path,seq_path):

        seqs = []
        lengths = []
        for seq_record in SeqIO.parse(seq_path,"fasta"):
            seq = str(seq_record.seq)
            seqs += [seq]
            lengths += [len(seq)]
        temp_org = np.array(buildPredict(seqs))
        probs_org = []
        for i in range(0,len(lengths)):
            probs_org += [temp_org[i]] * lengths[i]
        probs_org = np.array(probs_org)
        print "Len_Probs_Org:_{}_".format(len(probs_org))
        # print probs_org.shape

        mtxCompressed = np.load(path)
        X = mtxCompressed['savedX']
        Y = mtxCompressed['savedY']
        # print X.shape, Y.shape
        mod_x = np.reshape(X,(X.shape[0],X.shape[1] * X.shape[2]))

        base = joblib.load('org_base.pkl')
        scaler  = joblib.load('bag_scaler.pkl')
        scaledTestX = scaler.transform(mod_x)

        base_probs = base.predict_proba(scaledTestX)
        # print base_probs.shape
        org_X = np.hstack((base_probs,probs_org))
        print org_X.shape
        bbc = joblib.load('org_bbc.pkl')
        bag = joblib.load('org_bag.pkl')
        std = joblib.load('org_std.pkl')
        svc = joblib.load('org_svc.pkl')
        org_scaler = joblib.load('org_scaler.pkl')

        scaled_org_test = org_scaler.transform(org_X)

        predY1 = bbc.predict(scaled_org_test)
        predY2 = bag.predict(scaled_org_test)
        predY3 = std.predict(scaled_org_test)
        predY4 = svc.predict(scaled_org_test)

        # Classification Metric display
        print "Balanced_Bagging_MLP"
```

```
    print(confusion_matrix(Y,predY1))
    print(classification_report_imbalanced(Y,predY1))
    print(matthews_corrcoef(Y, predY1))
    print "Balanced_Bagging"
    print(confusion_matrix(Y,predY2))
    print(classification_report_imbalanced(Y,predY2))
    print(matthews_corrcoef(Y, predY2))
    print "Standard_MLP"
    print(confusion_matrix(Y,predY3))
    print(classification_report_imbalanced(Y,predY3))
    print(matthews_corrcoef(Y, predY3))
    print "SVC"
    print(confusion_matrix(Y,predY4))
    print(classification_report_imbalanced(Y,predY4))
    print(matthews_corrcoef(Y, predY4))

if __name__ == "__main__":
    main()
```

Listing C.2: Organelle Prediction Model

```
import sys, string, io, os, math
import numpy as np
from collections import Counter
import matplotlib.pyplot as plot
from scipy import interp
from itertools import cycle

from Bio import SeqIO
from Bio.Seq import Seq
from Bio.SeqUtils import GC123, lcc
from Bio.SeqUtils.ProtParam import ProteinAnalysis
from Bio.SeqUtils.ProtParamData import kd , Flex


from sklearn.linear_model import LogisticRegression , RandomizedLasso
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier , RandomForestClassifier

from sklearn.feature_selection import RFE, RFECV
from sklearn.model_selection import train_test_split , StratifiedKFold
from sklearn.metrics import matthews_corrcoef , classification_report , confusion_matrix , roc_curve , auc
from sklearn.preprocessing import StandardScaler , label_binarize
from sklearn.externals import joblib

from imblearn.metrics import classification_report_imbalanced
from imblearn.ensemble import BalancedBaggingClassifier


'''
    organelle.py generates the organelle prediction models for disordered prediction
    ROC plots and other graphs are also generated during the training
'''
```

```
windowSize = 9
aminoAcids = 'ACDEFGHIKLMNOPQRSTUVWYX'


proteinWeights = [
    89.047678,121.019749,133.037508,147.053158,165.078979,75.032028,155.069477,131.094629,146.105528,131.094629,149.051049,
    132.053492,255.158292,115.063329,146.069142,174.111676,105.042593,119.058243,168.964203,117.078979,204.089878,181.073893,
    143.656733]

def main():
    if len(sys.argv) > 3:
        print("Invalid inputs, please try again")
        print("Expected Format <FORMAT>")
    else:
        # global windowSize
        # windowSize = int(sys.argv[1])
        sequences,labels = readFasta()
        # num_features = optimalFeatures(np.asarray(sequences),np.asarray(labels))
        new_X = evaluateFeatures(np.asarray(sequences),np.asarray(labels), 49)
        buildModel(np.asarray(new_X),np.asarray(labels))
        # windows, labels = createWindows(s,l)
        # buildModel(np.asarray(windows),np.asarray(labels))

def readFasta():
    # LABELS: 1 = CYTO, 2 = MITO, 3 = NUCLEUS, 4 = SECRETED
    files = []
    for fa in os.listdir("../Data_Files/organelle_files/"):
        filePath = "../Data_Files/organelle_files/" + str(fa)
        files += [filePath]
    files = sorted(files)
    print files
    labels = []
    sequences = []
    # species_list = []
    for i in range(0,len(files)):
        for seq_record in SeqIO.parse(files[i],'fasta'):
            processed = processSeq(str(seq_record.seq))
            sequences += [processed]
            # species_list += [species]
            # sequences += [oneHotEncode(seq_record)]
            labels += [i]
    l = np.asarray(labels)
    s = np.asarray(sequences)
    # labelSpecies(species_list)
    print "Distribution of labels = " + str(sorted(Counter(labels).items()))
    np.savez_compressed('../Data_Files/temp_files/organelle',labels=labels,seqs=sequences)
    return sequences,labels

def oneHotEncode(sequence):
    base = [[0]*21 for _ in range(0,len(sequence))]
    assert(len(base) == len(sequence))
    for i in range(0,len(sequence)):
        pos = aminoAcids.find(sequence[i])
        base[i][pos] = 1
    assert(sum(x.count(1) for x in base) == len(sequence))
    return base
```

```
def createWindows(sequences,labels):
    boundary = [0]*20 + [1]
    new_labels = []
    windows = []
    diff = (windowSize-1)/2
    for i in range(0,len(sequences)):
        curr_seq = sequences[i]
        for j in range(0,len(curr_seq)):
            current = [boundary]*max(diff-j,0) + curr_seq[max(j-diff,0):min(j+diff,len(curr_seq))]
            current += [boundary]*(windowSize - len(current))
            assert(len(current) == windowSize)
            new_labels += [labels[i]]
            windows += [current]
    assert(len(windows) == len(new_labels))
    return windows,new_labels

def processSeq(seq):

    ''' Protein features found:
        - Sequence Length
        - Amino Acid Composition (global)
        - Amino Acid Composition (First 50/Last 50)
        - Isoelectric Point
        - Aromacity
        - Grand Average Hydropathy (Gravy)
        - Molecular Weight (global)
        - Molecular Weight (First 50/Last 50)
        - Secondary Structure Fraction
    '''


    # seq = str(seq_record.seq)
    prot = ProteinAnalysis(seq)
    # desc = str(seq_record.description).split('_')
    # species = desc[1].split(' ')[0]
    seq_length = len(seq)
    isoelectric = prot.isoelectric_point()
    gravy = calculateGravy(seq,0,seq_length)
    aroma = prot.aromaticity()
    ss_frac = prot.secondary_structure_fraction()

    mol_global_weight = calculateMolecularWeight(seq,0,seq_length)
    AA_global_dist = getAAPercent(seq,0,seq_length)
    flex_global = calculateFlexibility(seq,0,seq_length)
    if (seq_length > 50):
        AA_local_head = getAAPercent(seq,0,50)
        AA_local_tail = getAAPercent(seq,seq_length-50,seq_length)
        mol_local_weight_head = calculateMolecularWeight(seq,0,50)
        mol_local_weight_tail = calculateMolecularWeight(seq,seq_length-50,seq_length)
        flex_localh = calculateFlexibility(seq,0,50)
        flex_localt = calculateFlexibility(seq,seq_length-50,seq_length)
    else:
        AA_local_head = AA_global_dist
        AA_local_tail = AA_global_dist
        mol_local_weight_head = mol_global_weight
```

```
            mol_local_weight_tail = mol_global_weight
            flex_localh = flex_global
            flex_localt = flex_global

        return_vector = [seq_length,aroma,
                         isoelectric,
                         mol_global_weight,
                         mol_local_weight_head,
                         mol_local_weight_tail,
                         gravy,flex_global,
                         flex_localh,
                         flex_localt] + \
                         AA_global_dist + AA_local_head + AA_local_tail + list(ss_frac)

        # print seq_length, GC_distribution, mol_weight, aroma, isoelectric
        return return_vector

    def calculateGravy(sequence,start,end):
        total = 0
        for c in sequence[start:end+1]:
            if c in kd:
                total += kd[c]
            else:
                # Average Hydrophobicity score using Doolittle Scale
                total += -0.49
        return total

    def calculateFlexibility(sequence,start,end):
        modified_flex = Flex
        modified_flex['X'] = 0.99065
        modified_flex['U'] = 0.99065
        modified_flex['O'] = 0.99065
        window_size = 9
        weights = [0.25, 0.4375, 0.625, 0.8125, 1]
        flex_list = []
        subsequence = sequence[start:end]
        for i in range(0,len(subsequence) - window_size):
            current_score = 0.0
            current_window = subsequence[i:i+windowSize]
            for j in range(window_size // 2):
                current_score += ((modified_flex[subsequence[j]] + modified_flex[subsequence[windowSize-j+1]]) * weights[j])
            current_score += modified_flex[subsequence[windowSize // 2 + 1]]
            flex_list.append(current_score / 5.25)
        return np.mean(flex_list)

    def labelSpecies(species_list):
        species_unique = sorted(set(species_list))
        # print species_unique

    def calculateMolecularWeight(sequence, start, end):
        mol_weight = 0
        for i in range(start,end):
            position = aminoAcids.find(sequence[i])
            if (position == -1):
                mol_weight += proteinWeights[22]
            else:
```

```
            mol_weight += proteinWeights[position]
    return mol_weight

def getAAPercent(sequence, start, end):
    count = [0.0]*23
    for i in range(start,end):
        position = aminoAcids.find(sequence[i])
        if (position == -1):
            count[22] += 1
        else:
            count[position] += 1
    return [i / len(sequence) for i in count]

def optimalFeatures(X,y):
    scaler = StandardScaler()
    print(scaler.fit(X))
    scaled_train_x = scaler.transform(X)
    trees = RandomForestClassifier(n_estimators=100,random_state=19,class_weight='balanced')
    rfecv = RFECV(estimator=trees, step=1, cv=StratifiedKFold(2),scoring='accuracy')
    rfecv.fit(scaled_train_x,y)
    print("Optimal number of features : %d" % rfecv.n_features_)
    plot.figure()
    plot.xlabel("Number of features selected")
    plot.ylabel("Cross validation score (nb of correct classifications)")
    plot.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
    plot.savefig("Plots/OptimalFeatures.png")

def evaluateFeatures(X,y, num_features):
    scaler = StandardScaler()
    print(scaler.fit(X))
    scaled_train_x = scaler.transform(X)

    trees = RandomForestClassifier(n_estimators=100,random_state=19,class_weight='balanced')
    selector = RFE(trees,step=1,n_features_to_select=num_features)
    selector.fit(scaled_train_x,y)
    AA_global_labels = []
    AA_localh_labels = []
    AA_localt_labels = []

    for i in range(0,23):
        AA_global_labels += ["Amino_Acid_Composition_(Global)_Pos:{}".format(aminoAcids[i])]
        AA_localh_labels += ["Amino_Acid_Composition_(First_50)_Pos:{}".format(aminoAcids[i])]
        AA_localt_labels += ["Amino_Acid_Composition_(Last_50)_Pos:{}".format(aminoAcids[i])]

    labels = ["Sequence_Length","Aromacity","Isoelectricity","Molecular_Weight_(Global)", "Molecular_Weight_(First_50)","Molecular_Weight_(Last_50)","Gravy","Mean_Flexibility_(Global)",
    labels += AA_global_labels + AA_localh_labels + AA_localt_labels
    labels += ["Secondary_Struct_Helix","Secondary_Struct_Coil","Secondary_Struct_Sheet"]

    importances = selector.ranking_
    indices = np.argsort(importances)
    # sorted_labels = []
    # print len(labels),len(indices)
    # for i in range(0,len(indices)):
    #     sorted_labels += labels[indices[i]]
    # print len(importances),len(indices),X.shape[1]
    # print("Feature Ranking:")
```

```
    # for r in range(0,X.shape[1]):
    #     print "{}: {} ({})".format(r+1,labels[indices[r]],importances[indices[r]])
    # f = plot.figure()
    # plot.title("Feature importances")
    # plot.bar(range(X.shape[1]), importances[indices],color="b", align="center")
    # plot.xticks(range(X.shape[1]), sorted_labels,rotation='vertical')
    # plot.xlim([-1, X.shape[1]])
    # plot.savefig("Plots/")

    sorted_features = []
    for i in range(0,len(X)):
        row = X[i]
        current_features = [row[indices[i]] for i in range(0,num_features)]
        # print len(current_features)
        sorted_features += [current_features]
    # print(indices[0:num_features])
    return sorted_features

def buildPredict(predict_seq):
    sequences,labels = readFasta()
    X = np.asarray(sequences)
    y = np.asarray(labels)
    indices = [0,29,30,31,33,35,36,38,80,41,42,27,43,49,52,57,59,61,64,65,72,75,79,48,26,81,24,8,5,25,4,10,11,7,13,14,15,3,12,20,23,1,2,21,16,6,19,18,17]
    sorted_features = []
    for i in range(0,len(X)):
        row = X[i]
        current_features = [row[indices[j]] for j in range(0,len(indices))]
        sorted_features += [current_features]
    new_X = np.asarray(sorted_features)
    scaler = StandardScaler()
    print(scaler.fit(new_X))
    scaled_train_x = scaler.transform(new_X)
    X_train,X_test,y_train,y_test = train_test_split(scaled_train_x,y,random_state=19,test_size=0.3)
    svm = SVC(class_weight='balanced',random_state=19,decision_function_shape='ovr')
    svm.fit(X_train,y_train)

    sorted_predict = []
    for i in range(0,len(predict_seq)):
        temp_predict = processSeq(predict_seq[i])
        temp = [temp_predict[indices[j]] for j in range(0,len(indices))]
        sorted_predict += [temp]
    scaled_predict_x = scaler.transform(sorted_predict)
    probs_svm = svm.decision_function(scaled_predict_x)
    return probs_svm

def buildModel(X,y):
    # X = np.reshape(X,(X.shape[0],X.shape[1] * X.shape[2]))
    print X.shape, y.shape
    scaler = StandardScaler()
    print(scaler.fit(X))
    scaled_train_x = scaler.transform(X)
    X_train,X_test,y_train,y_test = train_test_split(scaled_train_x,y,random_state=19,test_size=0.3)

    bag = BalancedBaggingClassifier(n_estimators=200,random_state=19)
    svm = SVC(class_weight='balanced',random_state=19,decision_function_shape='ovr')
    neural = MLPClassifier(max_iter=500,random_state=19,solver='lbfgs',alpha=1e-5,hidden_layer_sizes=(49,8,4))
```

```
    ada = AdaBoostClassifier(n_estimators=100,random_state=19)
    logistic = LogisticRegression(solver='lbfgs',max_iter=500)


    bag.fit(X_train,y_train)
    svm.fit(X_train,y_train)
    neural.fit(X_train,y_train)
    ada.fit(X_train,y_train)
    logistic.fit(X_train,y_train)

    # joblib.dump(bag,'bag.pkl')
    # joblib.dump(scaler,'scaler.pkl')

    y_pred = bag.predict(X_test)
    y_pred2 = svm.predict(X_test)
    y_pred3 = neural.predict(X_test)
    y_pred4 = ada.predict(X_test)
    y_pred5 = logistic.predict(X_test)

    print matthews_corrcoef(y_test,y_pred)
    print matthews_corrcoef(y_test,y_pred2)
    print matthews_corrcoef(y_test,y_pred3)
    print matthews_corrcoef(y_test,y_pred4)
    print matthews_corrcoef(y_test,y_pred5)

    print confusion_matrix(y_test,y_pred)
    print confusion_matrix(y_test,y_pred2)
    print confusion_matrix(y_test,y_pred3)
    print confusion_matrix(y_test,y_pred4)
    print confusion_matrix(y_test,y_pred5)

    print(classification_report_imbalanced(y_test, y_pred))
    print(classification_report_imbalanced(y_test, y_pred2))
    print(classification_report_imbalanced(y_test, y_pred3))
    print(classification_report_imbalanced(y_test, y_pred4))
    print(classification_report_imbalanced(y_test, y_pred5))

    probs_ada = ada.predict_proba(X_test)
    probs_bag = bag.predict_proba(X_test)
    probs_neural = neural.predict_proba(X_test)
    probs_logistic = logistic.predict_proba(X_test)
    probs_svm = svm.decision_function(X_test)

    ROCplot(probs_ada,y_test,"Plots/ROCplotADA-organelle.png")
    ROCplot(probs_logistic,y_test,"Plots/ROCplotLogistic-organelle.png")
    ROCplot(probs_bag,y_test,"Plots/ROCplotBAG-organelle.png")
    ROCplot(probs_neural,y_test,"Plots/ROCplotNeural-organelle.png")
    ROCplot(probs_svm,y_test,"Plots/ROCplotSVM-organelle.png")

    multiROCplot([probs_ada,probs_logistic,probs_bag,probs_neural,probs_svm],y_test,"Plots/multiROCplot.png",['AdaBoost','Logistic','Bagging_Classifier','MLP','SVM'])

# Code used for ROC plots is derived a scikit-learn tutorial: http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
def multiROCplot(probs_list, Y_test,save,models):
    Y_test = label_binarize(Y_test,classes=[0,1,2,3])
    assert(len(models) == len(probs_list));
    fpr = dict()
```

```
    tpr = dict()
    roc_auc = dict()
    for j in range(0,len(probs_list)):
        for i in range(0,4):
            fpr[(j,i)], tpr[(j,i)], _ = roc_curve(Y_test[:,i],probs_list[j][:,i])
            # fpr[(j,i)], tpr[(j,i)], _ = roc_curve(Y_test,[probs_list[j][k][i] for k in range(len(probs_list[0]))])
            roc_auc[(j,i)] = auc(fpr[(j,i)],tpr[(j,i)])
        fpr[(j,'micro')], tpr[(j,'micro')], _ = roc_curve(Y_test.ravel(), probs_list[j].ravel())
        roc_auc[(j,'micro')] = auc(fpr[(j,'micro')], tpr[(j,'micro')])
        all_fpr = np.unique(np.concatenate([fpr[(j,k)] for k in range(0,4)]))
        mean_tpr = np.zeros_like(all_fpr)
        for i in range(0,4):
            mean_tpr += interp(all_fpr,fpr[(j,i)],tpr[(j,i)])
        mean_tpr /= 4
        fpr[(j,'macro')] = all_fpr
        tpr[(j,'macro')] = mean_tpr
        roc_auc[(j,'macro')] = auc(fpr[(j,'macro')],tpr[(j,'macro')])
    lw = 2
    plot.figure()
    color_list = cycle([['olivedrab', 'darkorange'], ['darkorchid','navy'],['black','firebrick'],['gold','slategrey']])
    for j,colors in zip(range(0,len(probs_list)),color_list):
        plot.plot(fpr[(j,'micro')], tpr[(j,'micro')],
             label='micro-average_ROC_curve_(area_=_{0:0.2f}),_Model:{1}'.format(roc_auc[(j,'micro')],models[j]),color=colors[0], linestyle=':', linewidth=4)
        plot.plot(fpr[(j,'macro')], tpr[(j,'macro')],
                 label='macro-average_ROC_curve_(area_=_{0:0.2f}),_Model:{1}'.format(roc_auc[(j,'macro')],models[j]),color=colors[1], linestyle=':', linewidth=4)
    plot.plot([0, 1], [0, 1], 'r--', lw=lw)
    plot.xlim([0.0, 1.0])
    plot.ylim([0.0, 1.05])
    plot.xlabel('False_Positive_Rate')
    plot.ylabel('True_Positive_Rate')
    plot.title('Multi-Model_Receiver_Operating_Characteristic_Plot')
    l = plot.legend(loc='upper_center', bbox_to_anchor=(0.5,-0.1))
    # plot.show()
    plot.savefig(save, bbox_extra_artists=(l,), bbox_inches='tight')

# Code used for ROC plots is derived a scikit-learn tutorial: http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
def ROCplot(probs,Y_test,save):
    Y_test = label_binarize(Y_test,classes=[0,1,2,3])
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(0,4):
        fpr[i], tpr[i], _ = roc_curve(Y_test[:,i],probs[:,i])
        roc_auc[i] = auc(fpr[i],tpr[i])

    fpr['micro'], tpr['micro'], _ = roc_curve(Y_test.ravel(), probs.ravel())
    roc_auc['micro'] = auc(fpr['micro'], tpr['micro'])

    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(0,4)]))
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(0,4):
        mean_tpr += interp(all_fpr,fpr[i],tpr[i])
    mean_tpr /= 4
    fpr['macro'] = all_fpr
    tpr['macro'] = mean_tpr
    roc_auc['macro'] = auc(fpr['macro'],tpr['macro'])
```

```
    lw = 2
    plot.figure()
    plot.plot(fpr["micro"], tpr["micro"],
          label='micro-average ROC curve (area = {0:0.2f})'.format(roc_auc["micro"]),color='black', linestyle=':', linewidth=4)
    plot.plot(fpr["macro"], tpr["macro"],
              label='macro-average ROC curve (area = {0:0.2f})'.format(roc_auc["macro"]),color='navy', linestyle=':', linewidth=4)

    colors = cycle(['olivedrab', 'darkorange', 'darkorchid','royalblue'])
    for i, color in zip(range(0,4), colors):
        plot.plot(fpr[i], tpr[i], color=color, lw=lw,label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i]))
    plot.plot([0, 1], [0, 1], 'r--', lw=lw)
    plot.xlim([0.0, 1.0])
    plot.ylim([0.0, 1.05])
    plot.xlabel('False Positive Rate')
    plot.ylabel('True Positive Rate')
    plot.title('Receiver Operating Characteristic Plot')
    plot.legend(loc="lower right")
    # plot.show()
    plot.savefig(save)


if __name__ == "__main__":
    main()
```

Listing C.3: File Format Encoding

```
import sys
import io
import string
import json
import csv
from pprint import pprint
import numpy as np
import math
windowSize = 7
aminoAcids = 'ACDEGHILKMNFPQRSTVWY'

'''
    encode.py converts different file formats into the requested filtered JSON formats
    The following file types are processed: FASTA, JSON, Tab-Separated, MTX
    This program also builds the sliding windows for prediction
'''


def main():
    global windowSize
    if len(sys.argv) > 5:
        print("Invalid Entry!")
        print("Expect arguments in format <python encode.py JSONFILE function OUTPUT>")
    else:
        if (sys.argv[2] == 'filter'):
            filter_JSON(sys.argv[1])
        elif(sys.argv[2] == 'fasta1' and len(sys.argv) == 4):
            fasta_format(sys.argv[1], 1, str(sys.argv[3]))
```

```
        elif(sys.argv[2] == 'fasta2' and len(sys.argv) == 4):
            fasta_format(sys.argv[1], 2, str(sys.argv[3]))
        elif(sys.argv[2] == 'tab'):
            filter_TAB(sys.argv[1])
        elif(sys.argv[2] == 'json' and len(sys.argv) == 5):
            windowSize = int(sys.argv[4])
            encode_JSON(sys.argv[1], str(sys.argv[3]))
        elif(sys.argv[2] == 'mtx' and len(sys.argv) == 5):
            windowSize = int(sys.argv[4])
            encode_MTX(sys.argv[1], str(sys.argv[3]))
        else:
            print("Invalid Entry!")
            print("Expect arguments in format <python encode.py JSONFILE function OUTPUT> ")

def filter_JSON(path):
    result = []
    with open(path, 'r') as r:
        raw = json.load(r)
    for i in range(0, len(raw)):
        current = {}
        current['sequence'] = raw[i]['protein']['sequence']
        if not raw[i]['protein']['mobidb']['consensus']['predictors']:
            current['consensus'] = raw[i]['protein']['mobidb']['consensus']['pdb']
        else:
            current['consensus'] = raw[i]['protein']['mobidb']['consensus']['predictors']
        result.append(current)
    print i+1
    with open('../Data_Files/filterJSON.json','w') as w:
        json.dump(result, w)
    print ("Finished, created filtered json file <filterJSON.json>")


def fasta_format(path, state, newfile):
    result = []
    n = 80
    with open(path, 'r') as r:
        raw = json.load(r)
    for i in range(0, len(raw)):
        if (state == 1):
            ident = "> " + str(raw[i]['protein']['disprot_id']) + " | " + str(raw[i]['protein']['protein_name'])
            current = [ident]
            seq = raw[i]['protein']['sequence']
            current += [seq[i:i+n] for i in range(0,len(seq),n)]
            result += current
        elif (state == 2):
            ident = "> " + str(raw[i]['id']) + " | Protein No: " + str(i)
            current = [ident]
            seq = raw[i]['sequence']
            current += [seq[i:i+n] for i in range(0,len(seq),n)]
            result += current
    with open(newfile,'w') as w:
        for line in result:
            w.write(line + "\n")
    print ("Finished, created fasta format file <" + newfile + ">")

def filter_TAB(path):
```

```
result = []
with open(path, 'r') as r:
    current = {}
    sequence = ''
    i = 0
    state = 's'
    consensus = []
    start = 1
    end = 1
    ident = ''
    for line in csv.reader(r, dialect="excel-tab"):
        i += 1
        if (line[1] == '1'):
            current['sequence'] = sequence
            consensus.append({
                'start': start,
                'ann': state,
                'end': end
            })
            current['consensus'] = consensus
            current['id'] = ident
            result.append(current)
            current = {}
            sequence = ''
            i = 1
            start = 1
            state = 's'
            consensus = []
            ident = line[0]
        if (state == 's'):
            if (line[3] == '1'):
                consensus.append({
                    'start': start,
                    'ann': state,
                    'end': end
                })
                state = 'd'
                start = i
        elif (state == 'd'):
            if(line[3] == '0'):
                consensus.append({
                    'start': start,
                    'ann': state,
                    'end': end
                })
                state = 's'
                start = i
        end = i
        sequence += line[2]
current['sequence'] = sequence
current['id'] = ident
consensus.append({
    'start': start,
    'ann': state,
    'end': end
})
```

```
        current['consensus'] = consensus
        current['id'] = ident
        result.append(current)
        result.pop(0)
        with open('../Data_Files/test_files/disopredJSON.json','w') as w:
            json.dump(result, w)
        print ("Finished, created filtered json file <disopredJSON.json>")
        pprint(result)

    def encode_JSON(path,save):
        seqs = []
        anns = []
        numStructured = 0
        numDisordered = 0
        numContext = 0
        with open(path, 'r') as r:
            original = json.load(r)
        print(len(original))
        for i in range(0, len(original)):
            struct, disord, cont, currentSeq, currentAnn = sequence_unravel(original[i]['sequence'],original[i]['consensus'],i)
            seqs += currentSeq
            anns += currentAnn
            numStructured += struct
            numDisordered += disord
            numContext += cont
        result = [seqs, anns]
        seqs1 = np.asarray(seqs)
        anns1 = np.asarray(anns)
        print('Shape of data array: ' + str(seqs1.shape))
        np.savez_compressed(save, inputs=seqs1, labels=anns)
        print('Number of disordered positions: ' + str(numDisordered) + ', Number of structured positions: ' + str(numStructured)
        + ', Number of context dependent positions: ' + str(numContext))

    def sequence_unravel(sequence, consensus,num):
        result = []
        for i in range(0,len(sequence)):
            oneHot = [0] * 21
            for j in range(0,len(aminoAcids)):
                if aminoAcids[j] == sequence[i]:
                    oneHot[j] = 1
            result += [oneHot]
        struct, disord, cont, finalSequence, finalConsensus = split_sequence(result, consensus,num)
        return struct, disord, cont, finalSequence, finalConsensus

    def split_sequence(sequence, consensus,num):
        boundaries = [0]*20 + [1]
        encodedSeq = []
        encodedAnns = []
        struct = 0
        disord = 0
        cont = 0
        total = 0
        for i in range(0,len(consensus)):
            x = range(consensus[i]['start']-1,consensus[i]['end'])
            start = consensus[i]['start']-1
            end = consensus[i]['end']-1
```

```
            total += end - start  + 1
            annotation = str(consensus[i]['ann']).upper()
            if (annotation == 'S'):
                annotation = 0
                struct += (end - start + 1)
            elif (annotation == 'D'):
                annotation = 1
                disord += (end - start + 1)
            else :
                annotation = 0
                struct += (end - start + 1)
            for j in x:
                if (j < ((windowSize-1)/2)):
                    current = [boundaries]*(((windowSize-1)/2)-j) + sequence[0:min(j+((windowSize+1)/2),len(sequence)-1)]
                    current += [boundaries]*(windowSize-len(current))
                elif (j > end-((windowSize-1)/2)):
                    current = sequence[(j-((windowSize-1)/2)):end+1] + [boundaries]*(((windowSize-1)/2)-(end - j))
                else:
                    current = sequence[(j-((windowSize-1)/2)):(j+((windowSize+1)/2))]
                current = np.array(current)
                current = np.ravel(current)
                encodedSeq += [current]
                encodedAnns += [annotation]
        if total < len(sequence) or len(encodedAnns) < len(sequence):
            print "Differences found !"
            print total - len(sequence)
            print total, len(sequence), len(encodedSeq), len(encodedAnns)
            print num
        return struct, disord, cont, encodedSeq , encodedAnns

    def encode_MTX(path,save):
        mtxCompressed = np.load(path)
        labels = mtxCompressed['labels']
        pssm = mtxCompressed['pssm']
        lengths = mtxCompressed['length']
        print len(pssm), len(labels), sum([int(i) for i in lengths])
        X = create_windows(pssm,lengths)
        Y = labels.tolist()
        savedX = np.array(X)
        savedY = np.array(Y)
        print(savedX.shape,savedY.shape)
        np.savez_compressed(save,savedX=savedX,savedY=savedY)


    def create_windows(old_pssm,lengths):
        pssm = old_pssm.tolist()
        background = 10e-07
        boundaries = [math.log(background)]*20 + [math.log(1)]
        windows = []
        pssmPosition = 0
        for i in lengths:
            for j in range(0,int(i)):
                lowerLim = (pssmPosition + j) - ((windowSize-1)/2)
                upperLim = (pssmPosition + j) + ((windowSize+1)/2)
                current = [boundaries]*(((windowSize-1)/2)-j) + pssm[max(lowerLim,pssmPosition):min(pssmPosition+int(i),upperLim)]
                current += [boundaries]*(windowSize-len(current))
```

```
            windows += [current]
        pssmPosition += int(i)
    print "Created Windows!"
    print "Length: " + str(len(windows))
    return windows

if __name__ == "__main__":
    main()
```

Listing C.4: PSI-BLAST MTX File Filtering

```
import sys
import os
import subprocess
import numpy as np
from pprint import pprint
import math

def main():
    if len(sys.argv) > 4:
        print("Invalid Entry!")
        print("Expect arguments in format <python sieveson.py JSONFILE function> ")
    else:
        crawl_files(sys.argv[1],sys.argv[2],sys.argv[3])

def crawl_files(path,anns,save):
    baseFileName = '../Data_Files/DisorderHunter-master/DISOPRED_RUNS/'
    PSSMvalues = []
    seqLengths = []
    with open(path, 'r') as r:
        for lines in r:
            if lines[0] == '>':
                identifiers = lines.split('_')
                accessionID = identifiers[0].strip('>')
                # print accessionID
```

```
                currentPSSM,length = process_mtx(baseFileName + str(accessionID) + '/' + str(accessionID) + '.mtx')
                PSSMvalues += currentPSSM
                seqLengths += [length]
        annotations = load_compressed_file(anns)
        pssm, limits = linear_scaler(PSSMvalues)
        pssmNP = np.array(pssm)
        limitsNP = np.array(limits)
        seqLengthsNP = np.array(seqLengths)
        np.savez_compressed(save,pssm=pssmNP,limits=limitsNP,length=seqLengthsNP, labels=annotations)
        print "Sorted through MTX files (#files: " + str(len(seqLengths)) + ")"
        print "Saved compressed PSSM in " + save

    def process_mtx(file):
        # The amino acid values are encoded in the mtx files as XAXCDEFGHIKLMNPQRSTVWXYXXXX,
        # where X denotes the unknown amino acids / terminating characters
        mtxPositions = [1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,22,23];
        with open(file,'r') as r:
            raw = r.readlines()
        counter = 0
        pssm = []
        for i,line in enumerate(raw):
            if i == 0:
                seqLength = line.strip()
            elif i == 1:
                seq = line.strip()
            elif i >= 14:
                values = line.split()
                currentPSS = []
                for j in range(0,len(mtxPositions)):
                    currentPSS += [int(values[mtxPositions[j]])]
                pssm += [currentPSS]
```

```
            counter += 1
        # print counter, seqLength, len(pssm), len(pssm[0])
        if counter > seqLength:
            print "File: " + str(file) + " has invalid sequence length!"
            print "Expected: " + str(seqLength) + " , Actual: " + str(counter)
        return pssm, seqLength


    def load_compressed_file(path):
        compressed = np.load(path)
        annotations = compressed['labels']
        print str(len(annotations)) + " annotations"
        return annotations

    def linear_scaler(pssm):
        finalLimits = [[0,0]]*21
        for i in range(0,len(finalLimits)):
            finalLimits[i] = [min(el[i] for el in pssm),max(el[i] for el in pssm)]
        for j in range(0, len(pssm)):
            for k in range(0,len(finalLimits)):
                pssm[j][k] = math.log(min(float(pssm[j][k] - finalLimits[k][0]+1)/float(finalLimits[k][1] - finalLimits[k][0]),1))
        return pssm, finalLimits

    if __name__ == "__main__":
        main()
```

Listing C.5: CASP Disorder Target Filtering

```
import sys, os, json, glob
import csv

from Bio import SeqIO
```

```
cwd = os.getcwd()

json_file = []

'''
    filter_casp.py is used to filter CASP '.dr' extension files and convert into a filtered JSON format
'''

total = 0
for drFile in glob.glob("{}/*.dr".format(cwd)):
    print drFile
    with open(drFile,"r") as r:
        counter = 0
        reader = csv.reader(r,delimiter='\t')
        current = {}
        sequence = ""
        state = ''
        consensus = []
        start = 1
        end = 1
        label = ''
        for row in reader:
            if len(row) == 3:
                counter += 1
                sequence += row[0]
                if row[1] == 'D':
                    label = 'd'
                else:
                    label = 's'
                if row[2] == '1':
```

```
                    state = label
                if not (state == label):
                    end = int(row[2]) - 1
                    consensus.append({
                        'start': start,
                        'ann': state,
                        'end': end
                    })
                    state = label
                    start = int(row[2])
                    end = int(row[2])
            consensus.append({
                'start': start,
                'ann': state,
                'end': counter
            })
            json_file.append({
                'consensus': consensus,
                'sequence': sequence
            })
            print counter
            total += counter
    with open('casp10.json','w') as w:
        json.dump(json_file,w)
    print total
```

Listing C.6: PSI-BLAST Output File Cleaning

```
import os, glob, time, sys, shutil
from Bio import SeqIO

'''
```

```
    filter_mtx.py filters the '.mtx' files generated after running PSI-BLAST
    Unnecessary files in the same folder are removed to clean workspace
'''


if not os.path.exists("../mtx_files"):
    os.makedirs("../mtx_files")

counter = 0

for seq_record in SeqIO.parse(sys.argv[1],"fasta"):
    current = seq_record.id
    print current
    file_path = "../DISOPRED_RUNS/{}/*.*".format(current)
    for cleanFile in glob.glob(file_path):
        print cleanFile
        if not cleanFile.endswith('.mtx'):
            os.remove(cleanFile)
        if cleanFile.endswith('.mtx'):
            shutil.move(cleanFile, "../mtx_files/")
            counter += 1
print counter
```

Appendix D

Project Plan

Please find my Project Plan attached overleaf.

Project Plan

NAME: SRIKRISHNA DATLA (ZCABDAT@UCL.AC.UK)
DEGREE PROGRAMME: MENG COMPUTER SCIENCE
SUPERVISOR(S) NAME: DR. KEVIN BRYSON

Project Title:

Development of protein disorder prediction server using machine learning and Django

Aims:

This project focuses on one particularly structural aspect of proteins called intrinsic disorder where proteins don't form the stable 'native' fold that give the protein its function and instead perform other functions instead, in some cases implicating them in certain diseases. The DisProt database has information about protein disorder where the proteins are categorised by their ontology and this project aims to make use of this data. Delving further into this subject area through this project aims to:

- Learn about disordered protein ontology and its behaviour based on varying environmental effects (e.g. differences in pH or organelles)
- Determine whether disordered region prediction can be aided by factoring the environmental effects into the machine learning models

Objectives:

The following objectives have been identified as an outcome of this project:

- Review currently available resources / predictions servers for disordered protein regions
- Process and filter through MobiDB data to find relevant data to train / test our model on
- Discover patterns in the combine dataset and build a relevant SVM model for secondary structure prediction
- Evaluate the success of the model generated with environmental considerations with other respective models which are currently being used (e.g. DISOPRED)
- Build an online server to aid research into disordered protein prediction based on the generated model, for ease of use by academics in the field

Deliverables:

The project would involve the complete development of a bioinformatics web application:

- From processing of raw DisProt and PDB structural data
- To the application of scikit-learn for machine learning
- To development of a Django web application so that medical and bioscience users can employ the newly developed method

During the course of the project I would also submit the following deliverables:

- An analysis of currently available datasets related to disordered protein regions and protein disorder prediction servers, looking for patterns and distinguishing information about disordered regions and whether environmental attributes affect their prediction
- A new model generated based on the dataset and analysis of its effectiveness in prediction compared to other relevant models currently used

Work Plan

Term 1:

- November 15th - Project Plan
- November 22nd - Project Abstract + Project Plan Review
- December 6th - Project Abstract Review + Analysis of Research Papers + List of programming libraries / languages and datasets to be used
- December 13th - Primary analysis of datasets and combining to find patterns
- December 20th - Confirmation of all resources to be used + researching tutorials relevant to the project

Term 2:

- January 10th - First attempt at culling dataset to determine best dataset to train on
- January 17th - Second attempt at culling based on building trained SVM
- January 31st - Built first SVM + perform analysis on testing data
- February 14th - Optimising SVM + dataset based on analysis on initial SVM
- February 21st - Review of algorithm and discussion of analysis
- February 28th - Determination of algorithm / model mechanics and review of system
- March 7th - Building Django application backend
- March 14th - Building Django application frontend
- March 21st - Testing web application + Deployment
- April 4th - Optimisation + analysis of web application
- April 18th - Final review and doing documentation
- April 28th - Final Deadline

Appendix E

Interim Report

Please find my Interim Report attached overleaf.

Interim Project Report

NAME: SRIKRISHNA DATLA (ZCABDAT@UCL.AC.UK)
DEGREE PROGRAMME: MENG COMPUTER SCIENCE
SUPERVISOR(S) NAME: DR. KEVIN BRYSON

Project Title:

Development of protein disorder prediction server using machine learning and Django

Progress to Date:

The current objectives that have been met as of 24th January 2018 are as follows:

- Analysis of Research Papers completed - Write-up to be completed
- Project Abstract completed - Final edits based on supervisor feedback to be done
- Dataset identification completed - I have reviewed the list of all datasets and settled on the Disprot dataset to be incorporated into the training set for the Logistic Regression model
- Primary analysis of datasets and dataset integration completed - Having identified the dataset we used, I culled to make sure that we did not end up with any duplicates, or unnecessary information and that the information gathered was in a relevant schema to be trained on. This was done by using the 'One Hot Encoding' method which converted the list of amino acids into vectors of 21 (1 boundary case included) and then concatenated into matrices of window size (total matrix size = 21 * 21)
- Model selection - We have initialised been using the Logistic Regression model as it provides a good baseline accuracy for the prediction of disordered regions within proteins and this can be a good way to progress to the more advanced models such as Multi-Layered Perceptrons which will require a lot more tweaking to produce optimal accuracy results. After experimenting with the results gained from training the Logistic Regression Model, we will now use the gained information (such as window size) to determine the hyper-parameters we adjust for the MLP model.
- First model built and tested using cross validation and DISOPRED 3 dataset - I have trained the Logistic Regression model on the culled Disprot data and have tested using the testing data for the well known protein disordered region prediction server DISOPRED3.
- Azure Cloud Services Web App initialisation in progress - I have attempted to test Django code on Azure and am in progress of making the initial framework for the Django application that will incorporate the training model that I am creating.

Updated Work Plan

Term 2:

- January 10th - First attempt at culling dataset to determine best dataset to train on
- January 17th - Second attempt at culling based on building trained SVM
- January 31st - Built Django Framework for Azure and optimised testing of model
- February 14th - Analysis of Model Testing and initial integration into Django Application
- February 21st - Django Frontend Building
- February 28th - Django Frontend First Build
- March 7th - Django Frontend + Backend integration and testing
- March 14th - Deployment and Optimisation
- March 21st - Gathering results on tests and Paper Review on testing metrics
- April 4th - Writing report on process and additional documentation
- April 18th - Final review and doing documentation
- April 28th - Final Deadline

Supervisor Signature: _____

Date: _____/_____/_____