Project Code: *Please leave this empty*

Prediction of cis-prolines to improve protein modelling

First supervisor
Name: Andrew Martin
Affiliation / Position: UCL / Professor of Bioinformatics and Computational Biology

Telephone: 0207 679 7034
Email: andrew.martin@ucl.ac.uk

Second supervisor (if applicable)
Name:
Affiliation:
Email:

Project Description

For most amino acids, typically 99.9% of peptide bond are in the 'trans' configuration because there is less steric repulsion between the Cα atoms of the two residues. However, in the case of the peptide bond preceding a proline amino acid, as many as 10% are in the 'cis' configuration [1] since both cis and trans result in steric clashes reducing the energy difference. However, the frequency of cis-prolines depends on the preceding amino acid, with glycine [2] and aromatic residues [3] being more common in the cis conformation. Indeed, up to 40% of aromatic-Pro peptide bonds are cis-prolines [4].
When modelling proteins, this difference can make a significant difference to the conformation of loops. In recent work, we have started to develop a method for predicting antibody loop conformation using machine learning; while the method works well, it does not take account of prolines. Thus the aim of this project is to try to develop a machine learning predictor for cis/trans-prolines in variable loops. While there have been some previous attempts to do this [5,6], these rely on position specific scoring matrices (PSSMs) generated from PSI-BLAST searches; these would not be applicable to our problem of looking at prolines in antibody loops since there is much less relationship between the sequences and PSSMs would not be appropriate.

Any particular skills that are required for this project?

Confidence in Python or Perl programming

Is the project suitable for full-time / part-time students?

Both

Please return this form to i.nobeli@bbk.ac.uk by Friday the 26th of February 2021.

Abstract:

Cis/trans proline configuration may cause poor loop predictions in Antibody loops since the current predictor relies on position-specific scoring matrices. They may perform poorly on the Antibody loops, especially CDR3 loops. As a result, machine learning models were built using Random Forest and XGBoost algorithms. Since cis/trans proline datasets are imbalanced, several methods were used to build a predictor that doesn't have any bias towards trans prolines. Out of the three methods; (1) oversampling the minority class (cis prolines), (2) undersampling the majority class (trans prolines) and (3) using XGBoost parameters "max_delta_step" and "scale_pos_weight" to Boost the cis proline prediction in an imbalanced dataset. Amino acid residues that surround the proline and secondary structure information (of the proline and surrounding amino acid residues) were used as the features to build the machine learning models. Amino acid residues were also encoded using six different encoders to convert the one letter amino acid code to numerical data.

Out of all three methods, it was found that the oversampling method performed poorly, with the highest MCC score of 0.09. However, the undersampled method using Random Forest classifiers produced an MCC score of 0.4, close to some of the current predictors (Song et al. 2006).
XGBoost models produced one of the best performing models with an MCC score of 0.41. To boost the performance further, an ensemble model that combines both the XGBoost model and Random Forest models (using the undersampling method). The encoder used for this was the BLOSUM90 which performed the best in both the Random Forest and XGBoost models. This ensemble model produced an MCC score of 0.42 and performed better than the model built using XGBoost alone.

Introduction:

The isomerisation of proline infrequently occurs in about 6% of imide bonds while also having a profound impact on the activity of many other regulatory proteins that play an essential role in cellular processing, such as protein degradation, protein folding, ion channel gating and transcription (Hsin and Manley 2012; Wedemeyer, Welker, and Scheraga 2002; Lummis et al. 2005). (Morgan and Rubenstein 2013)
TAD (The C-terminal transcriptional domain) of BMAL1 (Brain and muscle ARNT-like 1) biological importance of cis/trans proline conformation. TAD regulates the mammalian

circadian clock by being the centre for repressors and transcriptional cofactors involved in this cycle (Gustafson et al. 2017). It does this through a slow exchange between two conformational forms, trans and cis proline isomers. Each isomer state interacts with transcription factors that led to a more controlled, normal caridian timekeeping (Gustafson et al. 2017). Gustafson et al. study also found that locking TAD's Trp-Proline in the trans isomer conformation by mutating the proline residue to alanine led to a more shortened caridian cycle. The study showed no cis configuration was observed via NMR after the point mutation on the proline residue. As a result, it led to a reduced circadian period by 1 hour (Gustafson et al. 2017). However, a mutation in the proline residue (P625A) along with point mutation on the aromatic residue Tryptophan (W624A) showed a shorter circadian cycle of about three hours (Gustafson et al. 2017).

Cis/trans proline isomerisation also plays a role in protein folding since they are one of the factors that affect the rate-determining steps of protein folding (Wedemeyer, Welker, and Scheraga 2002). In natively unfolded proteins, the cis and trans isomer configurations are found as an equilibrium of mixtures between the two states (Wedemeyer, Welker, and Scheraga 2002). However, upon folding, the proteins are locked into either cis or trans proline isomers. By looking at conformational energy analysis performed on peptides, the trans states were dominant in 19 amino acids other than proline, where cis isomers showed better stability than even glycine residues which also had some cis isomer states (Zimmerman and Scheraga 1976). This was demonstrated through the stability of the peptide fragment αcarbon-carboxyl / amine-α carbon ($C^{\alpha 2}C'ONHC^{\alpha 2}$) in trans than cis conformation (Zimmerman and Scheraga 1976). The instability of cis-isomer in the peptide fragments was mainly due to unfavourable nonbonding interactions between $C^{\alpha 1} \bullet\bullet\bullet C^{\alpha 2}$ where the hydrogens that are attached to the $C^{\alpha 1}$ interaction with $C^{\alpha 2}$ and its hydrogen atom, which leads to instability (Zimmerman and Scheraga 1976). However, due to proline having its side chain connected to the C_{α}, both cis and trans have to deal with this issue. Therefore, cis proline is more favourable than cis glycine and, as a result, occurs more frequent (shown in figure 1) (Zimmerman and Scheraga 1976). The cis proline's unfavourability was reduced due to the similarity between $C^{\alpha 2}$ and C^{δ} who are both bonded to the amide, as shown in figure 1.

With this, cis and trans proline observation in native protein structures would be in equal amounts, but the trans proline conformation is more prevalent than the cis conformation (Wedemeyer, Welker, and Scheraga 2002). This is due to favourable interactions between neighbouring atoms. The interchange between the two conformations has a relatively high energy barrier with an activation energy of about 20 kcal/ mol (Wedemeyer, Welker, and Scheraga 2002). Since the peptide bond has a partial double bond attribute, the bond rotation only ever leads to two forms (Wedemeyer, Welker, and

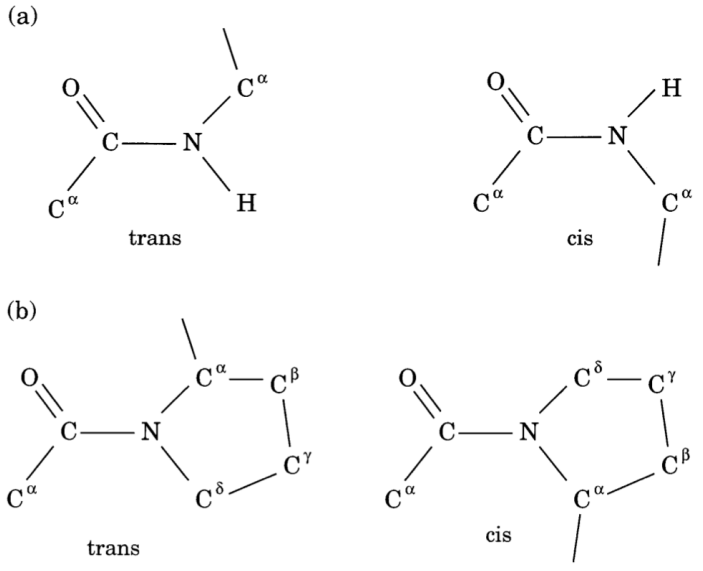
Scheraga 2002). The interchange between the two conformations has a relatively high energy barrier with an activation energy of about 20 kcal/ mol. As a result, enzymes like peptidylprolyl isomerases (PPIases) are needed to overcome the high activation energy and interchange cis /trans isomers by disrupting the partial double bond characteristic of the peptide bond (Wedemeyer, Welker, and Scheraga 2002).

Examples of how important cis /trans proline conformation are in the cellular processes were given above. However, in protein folding, cis/trans proline isomerisation isn't only the rate-limiting step but also plays a crucial role in the type of secondary structure formed (Morgan and Rubenstein 2013). These can be observed through some polyproline helices playing a role in joining alpha helices and beta pleats to form protein motifs (Morgan and Rubenstein 2013). In contrast, some other polyproline helices (polyproline II) seem to play a role in protein-protein interaction with SH3 domains (Morgan and Rubenstein 2013). There are two folds tri-prolines undertake; a left-handed helix made up of trans prolines called polyproline II (PPII) and a right-handed helix made up of cis prolines called polyproline I (PPI). Since PPII is made up of trans prolines, they form a more symmetrical structure with a higher pitch than PPI. However, PPI hasn't yet been Identified in protein structures and has been seen in non-protein molecules such as poly-L-proline. PPII structures in proteins are usually highly conserved above 80% of proteins with sequence identities >=20 (Adzhubei and Sternberg 1994). PPI conservation was found to depend on the conservation of superfamily domains (Adzhubei and Sternberg 1994). With how the stereoisomer form of prolines plays a role in some vital cellular processes, knowing the form of proline confirmation may help predict protein structures with more functional relevance.

Figure. 1. *(a) Shows cis/trans conformation in non-proline amino acid residues while (b)*

shows cis/trans conformation in proline residues (Wedemeyer, Welker, and Scheraga 2002).

As a result, knowledge of proline cis/trans conformation could play an important role in protein structure prediction and may cause errors in predicting the antibody loop structure. Some other cis/trans proline prediction models have been done, which take PSSM matrix score as features in building the machine learning predictor (Song et al. 2006; Exarchos et al. 2009). However, since cis/trans proline conformation prediction is required for a machine learning predictor that predicts Antibody loop conformation, using machine learning models based on PSSMs won't apply to CDR3 loops. This is due to PSSMs working on sequences/chains that have relationships between them, so working on CDR3 loops should reduce the performance of predictors built using PSSMs. Therefore, in this project, machine learning models were built using secondary structure information and neighbouring amino acid residues since the antibody's secondary structure information was already known.



Methods and Database:

Protein dataset:

The non-redundant protein dataset used for this project was obtained from Roland Dunbrack's Pieces PDB culling server (Wang and Dunbrack 2003). Here, PDB chains with 25% sequence identity or less were selected. Additional filters exclude chains with disorder and missing amino acid residues that were not resolved experimentally (referred to as chain breaks). The PDB chains that had been resolved with X-ray crystallography and a resolution of 3Å or less were selected since this PDB should have enough power to label cis/trans proline.
With all these filters, the cullied PDB chains returned from the server were 3,940 chains. Despite this, the number of chains used for the proline dataset was 3,660 since some of the chains were mmCIF files that were not included as only PDB chains were used.

Then using the BiopTools such as PDBtorsion, PDBgetchain and PDBsecstr, all the proline datasets (along with specified window size, e.g. if a proline found in the chain was close to the terminal end of the protein, neighbouring amino acid residues were not extracted, and the proline were skipped) were extracted. To pull the PDB chains listed in the Pisces data, a python script was used to convert the list into a single line with space separating each protein chain as an output. Then a bash script that first saved this list into an array variable was used to extract the correct PDB files from a directory that contains 107, 969 PDB files. After extracting the PDB files, the BiopTools PDBgetchain was used to extract the chains listed in the Pisces database using the chain letter as an input parameter for the program. Then PDBtorrson was run on this chain to extract the torsion angles of each amino acid, where each PDB chain's torsion output was saved as a text file. Once torsion angles were extracted, the secondary structure information was obtained for each amino acid residue of the chain by running the PDBsecstr program with a file that contains the chain information as the parameter. The output of the PDBsecstr program was then saved as a text file named after the PDB name of each chain. The extraction of the torsion angles and secondary structure information led to two directories containing 3,660 torsion files and 3,660 files from PDBsecstr. To build machine learning models, proline datasets in a CSV file format are required. In order to do this, a python script has been used that reads in both the torsion text file and the file that contains the secondary structure information. The python script then searches the torsion file for proline, and once found, the absolute torsion angles greater than 90 degrees were labelled as trans. While the absolute torsion angles less than 90 degrees were labelled as cis using python's abs() method. After each proline's cis/trans conformation was labelled either cis or trans, neighbouring amino acid residues were extracted. The python scripts finally print out strings where each line consists of

comma-separated proline datasets. Each line contained the PDB name where the proline was from alongside its residue number, cis/trans type and flanking amino acid residues. This output was saved as a CSV file.

The proline dataset that consists of only the amino acid residue sequence of the given window was larger than the dataset that included data for secondary structure. Using a window size of five as an example shows the effect of secondary structure data extraction on the size of the dataset. Window size refers to the number of amino acid residues preceding the proline (N-terminal residues) and the same number of amino acids preceding the proline (C-terminal residues).

The dataset size with a window size of five was 37,686 samples when secondary structure data wasn't included. When secondary structure data was included, only 36,680 of the samples had complete secondary structure information for all residues in the window size. As a result, 1,006 samples have partial secondary structure information. This was due to the program PDBsecstr skipping amino acid residues labelled "HETATMs" and only extracting secondary structure information for residues labelled with "ATOM". PDB files label non-modified amino acid residues as "ATOM" and water and other compounds, ions, modified amino acid residues (i.e. p-TYR) or ligands as "HETATMs". As a result of this, the PDBsecstr program skipped entries with the label "HETATMs"; therefore, modified amino acid residues were also skipped. However, since only 41 of the partial samples were cis (which was 2.2% of the cis dataset), they were dropped using pandas "dropna" function before creating the training and testing samples instead of creating a python script that edits the PDB files and converts the modified amino acid residues from "HETATMs" to "ATOM". The issue with the program was notified to the author, so PDBsecstr would be updated to fix this issue.

Secondary structure and Amino Acid residue Encodings:

The secondary structure information extracted using the PDBsecstr program was labelled as shown in appendix tabel1. The one letter (or - sign for coil) code were nominal outputs of PDBsecstr for each amino acid residue. In order to use the PDBsecstr one-letter code for machine learning, they were converted to binary data using one-hot encoding shown in column three of appendix tabel1. To do this, a python program was run that uses a text file containing the one letter secondary structure code along with their associated one-hot encoding. This python program takes command line parameters such as the protein dataset CSV file along with the header for the encoded output CSV file and the type of encoding to use.

Along with encoding the secondary structure information, the python dataset consists of amino acid residues. These residues also need to be converted into numerical datasets. The program uses a command line parameter that selects the encoder type for the amino acid residues in the protein dataset. This encoder was saved as a text file which the program reads in and then converts into a python dictionary where the keys were the one-letter amino acid code, and the values were the types of encoding. Since each line of the proline dataset CSV file contains PDB name, type (type of isomer), proline residue number, sliding window amino acid residues and secondary structure information. The program encoded only the amino acid residues and secondary structure information. The python program can encode the amino acid residues using six different methods:

1. BLOSUM45
2. BLOSUM62
3. BLOSUM90
4. PCA-5 of the 20 BLOSUM scores
5. Abhinandan 4 parameter physical encoding
6. A nine parameter encoding that consists of both PCA-5 and 4 parameters physical encoding

There were three BLOSUM substitution matrices used in this project BLOSUM45, 62 & 90. Each of these encodings was a 20-dimensional matrix that encoded each amino acid with 20 parameters consisting of real numbers (Henikoff and Henikoff 1992). The BLOSUM matrices derived from the BLOCK sequence alignment consists of multiple versions where each version corresponds to the sequence identities (Henikoff and Henikoff 1992). To encode the amino acid residues from the sliding window, each BLOSUM substitution matrix was stored in individual text files. As mentioned earlier, these text files were used to encode amino acid residues where command line parameters such as **b45**,**b62** or **b90** (corresponding to BLOSUM45, BLOSUM63 and BLOSUM90) were used.

The BLOSUM substitution matrix with 20 dimensions was reduced to five parameters using principal component analysis by several studies and shown to have good approximations of the higher dimensional encodings (Maetschke, Towsey, and Bodén 2005, Gu et al. 2007). In these studies, the substitution matrix was first converted to a distance matrix then dimensionally reduced to either five or three using PCA (Maetschke, Towsey, and Bodén 2005, Gu et al. 2007). The encoder used in this project was a PCA 5 encoder that used a substitution matrix (instead of converting it to a distance matrix) produced by Li and Koehl. This PCA-5 encoder also showed no significant difference between different BLOSUM scores (100 to 45) that were dimensionally reduced to five (Li and Koehl 2014). This encoder that consists of five real

numbers parameters used in this project was referred to as **t5** in both this thesis and the python encoder program mentioned above.

The other non 20-dimensional encoder used in this project was Abhinandan 4 parameter physical encoding. This encoding consists of a four-dimensional vector for each amino acid residue instead of the 20 substitution matrix (Abhinandan and Martin 2010). Each parameter represents the physicochemical properties of the amino acid residues, such as the total number of side-chain atoms, the charge, the number of side-chain atoms for the shortest alpha carbon and finally, the Eisenberg consensus hydrophobicity (Abhinandan and Martin 2010, Eisenberg et al. 1982). In the python scripts, the 4 parameters physicochemical encoder was referred to as **a4**.

The final encoding used to encode amino acid residues was a combination of the 5 parameter PCA-5 BLOSUM encoder and the Abhinandan 4 parameter physical encoding to create a nine parameter encoding referred to as **ant** (**A**bhinandan a**n**d **T**5 scale, which was the PCA-5 encoding).

Machine Learning:

The Machine Learning algorithms used in this project were Random Forest and XGBoost. Since the proline dataset was imbalanced (consisting of about 4% cis-proline), three different approaches were carried out:

1. Oversampling the minority class (cis-prolyl samples)
2. Undersampling the majority class (trans-proline samples)
3. Using XGBoost with no over/undersampling

An ensemble model from two machine learning classifiers (Random Forest and XGBoost) was also used to improve the predictor's performance.

1. **Oversampling the Cis dataset**:

The cis dataset comprises 4% of the proline data as previously stated, and to handle this imbalance dataset, the minority sample was oversampled. To carry out oversampling of the cis dataset, they were randomly sampled until the same number of instances as the trans dataset was obtained. To do this, a python script was used that uses imblear.over_sampling module "RandomOverSampler", which with a single line of code can duplicate the minority class (cis class) to be equivalent to the number of instances as majority class (trans). However, first, the test sample was extracted by

taking 20% of the cis dataset and the same number of trans dataset (which leads to an equal number of the cis and trans dataset). Testing sample extraction was done first to prevent having cross-contamination of testing and training samples after duplication.
To see the performance of this method of tackling an imbalanced dataset, a spot test style approach was used where a single set of training and testing samples were used to evaluate the oversampling approach alongside the effect of different window sizes, and encoders have on the performance of the predictor.
The machine learning algorithm used was Random Forest with varying numbers of trees ranging from 20 to 100 in an interval of 20. The Random Forest classifier was built using weka, and this was done by first converting the training and testing samples files from CSV to ARFF via a program called “csv2arff”. This program takes in the CSV files along with the name of the class column (column with cis/trans label) and a text file that contains all the attribute column names. Once ARFF files were obtained, the weka Random Forest classifier was run using a bash script that first builds the model and then runs this model along with a testing sample to output a file containing the model predictions.

2. **Undersampling the trans dataset**:

The trans dataset was undersampled by Randomly selecting the same number as the cis dataset. However, the testing sample was first separated by taking 20% of the cis dataset (about 360 samples) and randomly selected trans dataset of the same amount. The Training set was also built to have a balance between both classes. Here the trans samples used for training were randomly selected using the size of the cis dataset as the amount to extract. This was carried out using a python script that first loads the dataset into pandas series and split testing and training sets using the indexed column of the data frame. By default, creating a pandas data frame also creates an indexed column, and so using this method, the trans dataset can be undersampled in the training set without having the same instance found in the testing set.
A spot test approach was used to select the optimum window size and encoder, where one set of training and testing samples were used.
To see the performance of the predictor, five sets of the testing samples were used (all the cis dataset split into five) to cross-validate the predictor performance. Then taking the mean MCC value from all five sets gives the overall performance of the test. During training, the remaining cis dataset (the remaining 80%) were taken with equivalent size trans dataset through randomly selecting them to create balanced data. Since the cis dataset was about 20 times smaller than the trans dataset, 20 sets of training data were set up for each set of the cis training set. This led to building a predictor on the same cis dataset with 20 different trans datasets (shown in figure 2).

Weka machine learning tool's Random Forest algorithm was used to build the models. Using the training sets, Random Forest models were built first using 500 trees in the forest. In order to do this, CSV files were first converted to ARFF files as mentioned in the Oversamplied method, using the same program (csv2arff). Then a bash script that calls the Weka tool's Random Forest algorithm along with the training ARFF file and the number of trees parameter was used to build the predictors.
After all sets of 20 models (100 overall since they were 5 different cis sets and each cis set has 20 different trans sets) were built, the test set was used to evaluate the models' prediction. Utilising the majority vote method, where each model outputs a prediction, and its probability, the class with the most votes, was taken as the final prediction. Then MCC was used to evaluate the model's performance, and the five MCC values from each set of cis models were averaged to give the model's overall score. This was carried out on multiple encoders to see which encoder provides the best score.

3. **XGBoost with no over/undersampling**:

XGBoost uses a gradient tree boosting machine learning algorithm that is fast and lightweight (Wang, Deng, and Wang 2020). The imbalanced dataset was not handled through oversampling the cis dataset or undersampling the trans dataset but using XGBoost "scale_pos_weight" (weights) parameter that balances the positive and negative classes. To carry out the machine learning task, XGBoost python wrapper was used in a python script with varied parameters such as "max_delta_step" and "scale_pos_weight". To find the optimum parameters for the XGBoost model, 16 different weights ranging from 10 to 26 were used, where each weight had a version with "max_delta_step" at either 0.9 or 1 (which were the only two that showed promising results). XGBoost models were built using different window sizes, and cross-validation was done by first splitting the cis dataset into five different sets using a python script that first loads the whole dataset into a pandas data frame. Since by default, pandas have an indexed column alongside the columns used for the datasets. This index was used to make sure samples in the training set were not in the testing samples. By using this, ⅕ on the cis samples were removed and added to a set of testing samples alongside an equivalent trans sample to create a set of the testing datasets. Once a set of testing samples were removed, the remaining proline dataset was used as the training set since XGBoost can directly handle an imbalanced dataset. As a result, five different pairs of training and testing datasets were used to build and test five different XGBoost models. The training and testing split was repeated for window sizes ranging from 3 to 9, where each window size was encoded using the encoders mentioned above.

4. **Ensemble of XGBoost and Random Forest**:

The ensemble method used in this project combines both XGBoost and Random Forest. The first two types of models were built (an XGBoost and a Random Forest model) using the same encoder and amino acid residue window size from the best performing XGBoost and Random Forest. Then the proline dataset was first split into training and testing set by taking like mention above 20% of the cis dataset and the same number of trans dataset to create the training set. Then the remaining proline dataset was used for XGBoost training since the algorithm can handle imbalanced datasets, and for the Random Forest model, the same training set was split into 20 undersampled cis sets (consisting of the same cis proline dataset and different trans dataset).

Once the XGBoost and Random Forest predictor are built, predictions are made using the same testing set. As a result, the predictions from each machine learning classifier are taken. Then using a python script, predictions from both are weighed against each other, and the one with the higher probability was chosen as the final prediction.

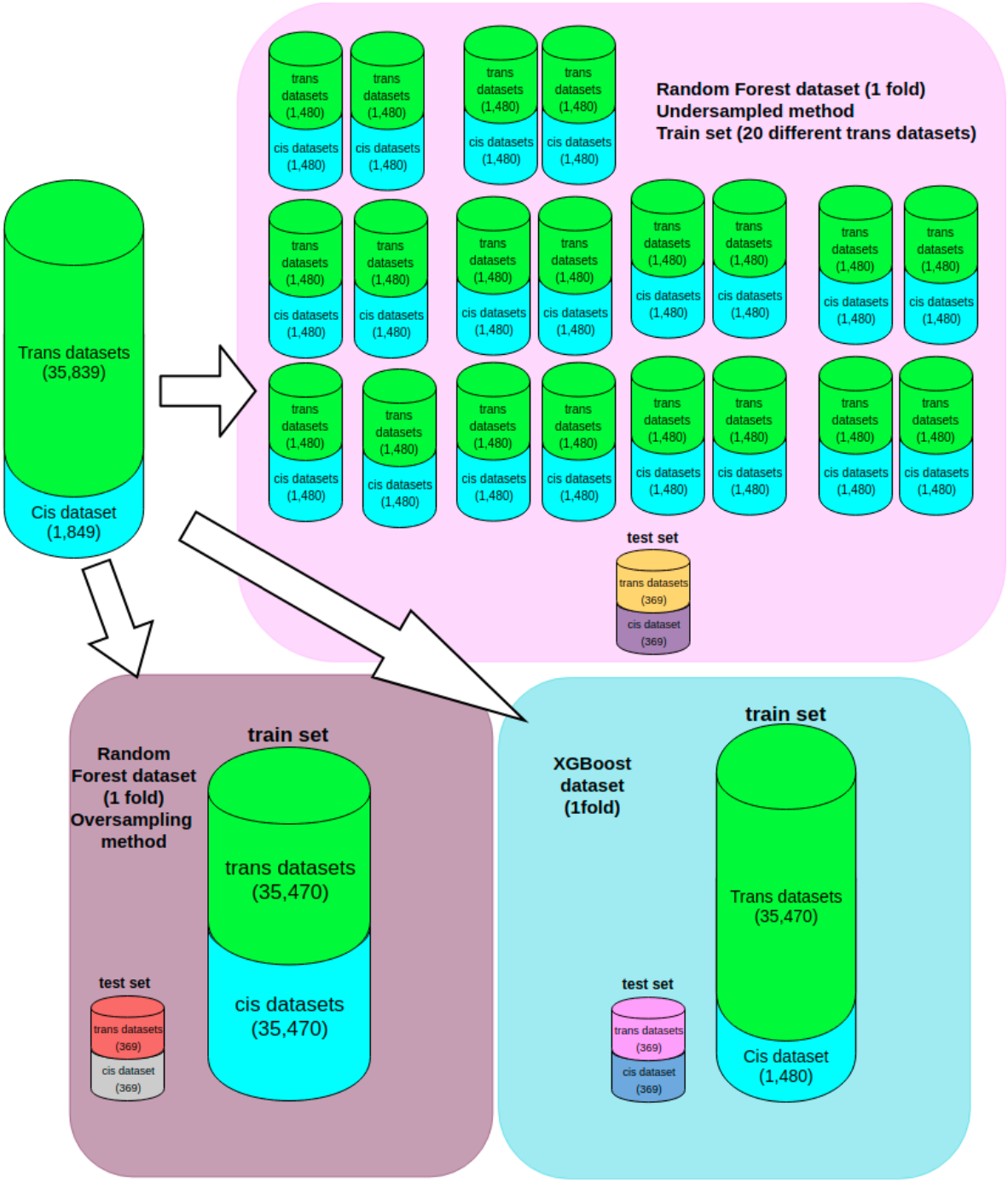


Fig. 2. *Shows how testing and training samples are split, where the undersampling method is shown in the (light)pink bubble, the XGboost samples are shown in (light) blue and finally the brown-red bubble shows the oversampling method.*
Fold (1 fold) refers to one out of the five cross-validation folds samples.

Results:

The machine learning predictors were trained using amino acid residue sequences of a given window size that was encoded. And the surrounding secondary structure information that was encoded using one-hot encoding. To deal with the imbalance in the proline dataset, three methods were used to build a classification predictor. And in order to evaluate the performance of each model, MCC (Matthews correlation coefficient) scores were used instead of accuracy, precision or F1 score based evaluations. In general high scores of MCC are only achieved if all parameters of the confusion matrix are good, e.g. high true positive and true negative, while having low false negative and false positive (Davide and Jurman 2020). In cases where binary classes are imbalanced like this proline dataset, a machine learning model can always predict a trans proline, and since about 95-96% of all prolines are trans, it is theoretically ~95% accurate but would score an MCC close to zero. As a result, MCC is used to evaluate how well each model predicts the two classes.

Initially, amino acid residues for a given window size were tested to see how the models perform compared to the secondary structure. Without the secondary structure, the oversampled Random Forest model had an MCC score of 0.03, while the undersampled Random Forest had a score of 0.21. Due to how low these MCC scores were, both types of features were used to find and build the optimum predictor for cis/trans classification.

The oversampling Random Forest models were built using a training dataset consisting of trans samples of about 34,511 and an oversampled cis class (34,511) to create a dataset of about 69,023 samples. Using the testing sample to evaluate the performance of the model showed that the model predicted only 6 cis samples correctly (MCC score of 0.091) for a tree size of 20, a window size of 5 and a t5 encoding. Varying the trees also led to MCC score reducing as the number of trees were increased. For tree sizes between 40 to 100 had 2 correctly predicted cis prolines out of 362 with an MCC score of 0.053. The highest number of trees tried was 500, and as a result, this produced an MCC score of 0.037 (the lowest score) with only one correctly predicted cis proline. This score was also similar to the MCC score produced by the oversampled model built using no secondary structure features. Therefore the

Using the majority undersampling method to build the Random Forest models, to find the optimum window size and encoder, one out of the five were used to train and evaluate the scores of the models. The results from this are shown in table 1 where the highest score was found in the b90 encoder with a window size of 4 (0.395), and the second-highest score was also found in the b90 encoder (with a window size of 5). The b90 encoder had an average score of 0.373 across all six window sizes, while mean MCC score for the other encoders were; b45= 0.337, b62= 0.354, t5= 0.356, a4= 0.350 and finally ant= 0.348. As a result, the encoder b90 on average performed better than all the other five encoders, with b62, t5 and t5 having almost similar scores.

Window size:	B45 (MCC)	B62 (MCC)	B90 (MCC)	T5 (MCC)	A4 (MCC)	ANT (MCC)
4	0.337	0.351	0.395	0.358	0.349	0.372
5	0.354	0.3568	0.380	0.350	0.361	0.342
6	0.357	0.370	0.369	0.360	0.344	0.301
7	0.335	0.340	0.378	0.357	0.366	0.359
8	0.327	0.339	0.362	0.359	0.348	0.356
9	0.311	0.367	0.353	0.354	0.329	0.358

Table. 1. *Shows MCC scores for each pair of encoder and window size.*

Testing the performance of the window size 5 with b90 encoder on the five cis testing sets produced a mean MCC of 0.395, which was a higher MCC and a more general score for the model's performance overall. The undersampling method, in general, produced a much better score than the model built with oversampling the cis class. As a result, the undersampling technique handled the unbalanced dataset much better than the oversampling technique.

The models that were built using XGBoost had varying "scale_pos_weight" (referred to as weight) and "max_delta_step". According to XGBoost's documentation, the optimum weight used in unbalanced data should be calculated by dividing the majority class amount by the minority class, as shown below. In a window size of 5, 35,839 samples from trans and 1,849 samples from cis led to a weight of 19.38. To evaluate how well it performs alongside max_delta_step, different sets of weight and max_delta_step were used on a window size of 3 with the t5 encoder to see which set of XGBoost parameters were optimum. This led to building and testing 238 XGBoost models that were created from weights varying from 10 to 26 and max_delta_step from 0 to 1.3 (increasing by 0.1). The performance of each model was judged based on the MCC score, and the

model's parameter with the best score was selected. The parameter search results were shown in figure 3.

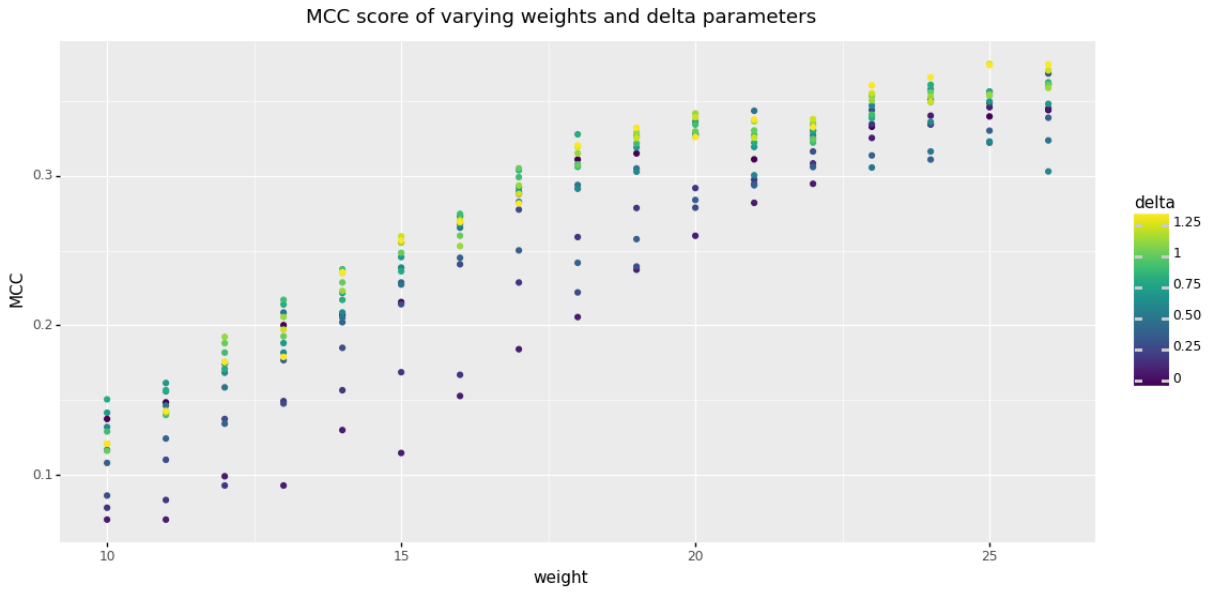


Fig. 3. *Shows MCC scores for XGBoost parameter "scale_pos_weight" (weight). The colour scale indicates a corresponding "max_delta_step" (delta) where the more yellow the colour the higher the delta parameter and the more purple the colour the lower the delta parameter.*

Figure 3 illustrates that as the XGBoost weight parameter increases, the MCC score increases. This was shown by low weights such as ten who had MCC scores ranging from 0.070 to 0.137, while a weight of 26 had MCC scores ranging from 0.303 to 0.374. An increase in max_delta_step (delta) also showed an increase in MCC score wherein the majority of weights a delta of zero had the lowest MCC score. However, in some weights such as 10 and 21, the highest MCC scores were produced by delta parameters of zero or 0.5, respectively. As a result, the right pair of weights and parameters are needed to create the optimum XGBoost models. As a result, a weight of 25 and 1 was selected since this pair produced the highest MCC score.

Using parameters such as: "scale_pos_weight" = 25, "max_delta_step" = 1, "learning rate" = 0.1 and was tested on windows from 3 to 9 with all six encoders. The results from this were shown in figure 4. The MCC score shown in figure 4 was obtained by taking all five sets' average scores. Figure 4 shows that the BLOSUM90 (b90) encoder had a higher MCC on all window sizes except for a window size of 9 than any other encoder. It also had the highest MCC score amongst all XGBoost models (MCC of 0.414). In terms of window size, MCC across all encoders seems to follow a similar

trend where window sizes bigger than 5 showed a decrease in MCC score with an increasing window size except in 8. Comparing the 20 attribute BLOSUM encoders (b45, b62, b90) against the PCA encoder (t5), the physical encoder (a4) and the encoder created from combining these two (ant) showed that three windows out seven windows size the BLOSUM encoders outperformed the other three. While b45 and b90 consistently outperformed all other encoders except for the window size of 9, where the physical encoder (a4) had the best MCC score (0.374). The a4 encoder is also the only encoder with less than 20 attributes that passed the 0.4 MCC score barrier with a score of 0.405. The t5 encoder created from a PCA-5 of the BLOSUM 20 attribute encoder had a worse MCC score in 6 of the seven window sizes than any BLOSUM encoder. T5 encoder only showed a better MCC in window size five, where its MCC (0.394) was slightly better than b62 (0.392). T5 also produced the lowest MCC score in all window sizes except for window sizes 4 and 5. The encoder also had the lowest MCC score in all the XGBoost models, with a score of 0.342 in window size 7.

Looking at the window sizes, window size 5 had three MCC scores above 0.4, with its lowest score being 0.392, higher than four out of the seven window sizes. Window size 9 had the maximum MCC score of 0.374, the lowest in all seven window sizes' maximums. However, it didn't produce the lowest MCC score. As mentioned before, the window size with the lowest MCC score was window size 7 with a score of 0.342.

Window size:	B45 (MCC)	B62 (MCC)	B90 (MCC)	T5 (MCC)	A4 (MCC)	ANT (MCC)
3	0.396	0.394	0.401	0.371	0.387	0.391
4	0.375	0.370	0.382	0.361	0.373	0.356
5	0.407	0.392	0.414	0.394	0.404	0.396
6	0.384	0.370	0.392	0.348	0.378	0.356
7	0.376	0.377	0.381	0.341	0.355	0.346
8	0.377	0.385	0.388	0.353	0.376	0.365
9	0.370	0.367	0.369	0.347	0.374	0.354

Table.2. *Shows MCC scores for XGBoost models and their corresponding MCC scores.*

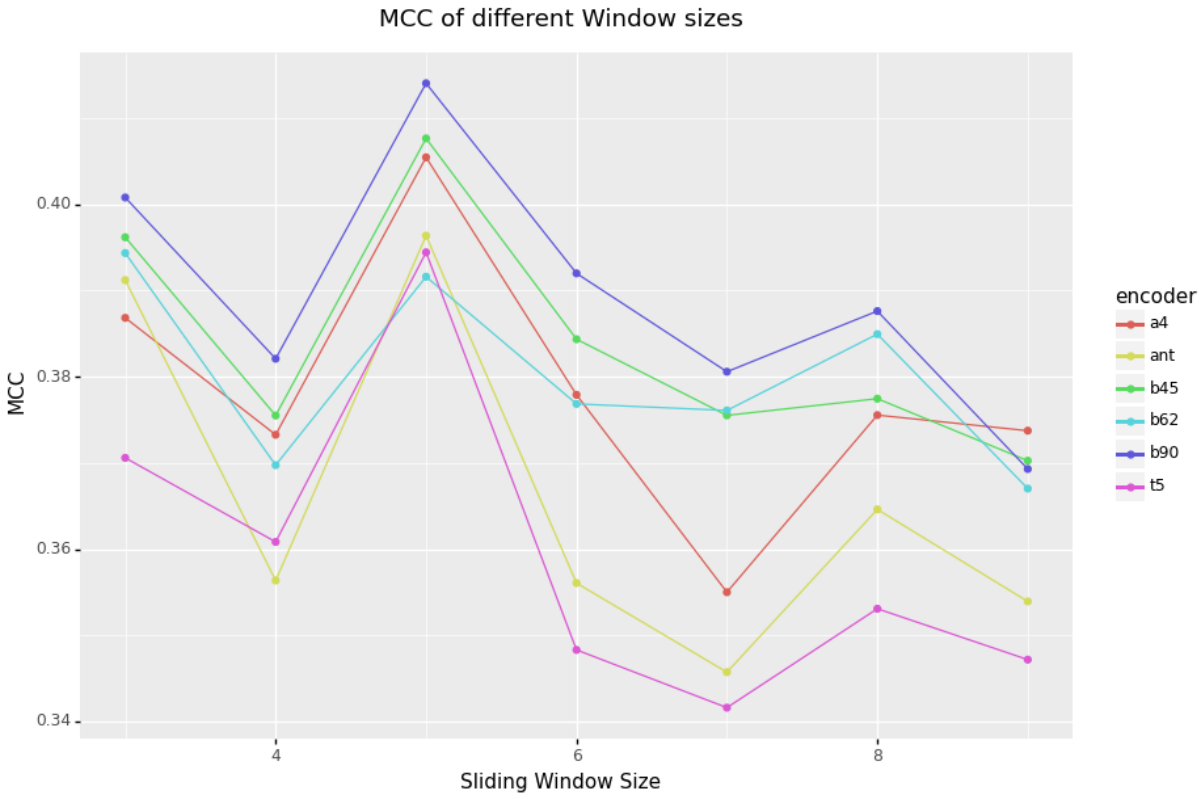


Fig.4. *Shows MCC scores for each window size and its corresponding encoder. The colour scale indicates the encoder each line belongs to*

The model built using XGBoost performed better on most encoders and window sizes than both under/oversampling methods built using Random Forest. In the case of the undersampling method, the b90 encoder for a window size of 5 had an MCC score of 0.395, which was less than the MCC score of 0.414 for encoder b90 and window of 5. As a result, using XGBoost with the correct parameters produces a better predictor for cis/trans proline than the Random Forest models.

The best score for the XGBoost model was obtained using b90 as the amino acid encoder and along with secondary structure for surrounding residues. Using our base model (the main predictor) and the Random Forest model built using the same features (encoder of b90), an ensemble machine learning predictor was made. In this combined model, the predictions are a result of a vote from each individual model. Where a tie occurs (e.g. XGBoost votes for trans and Random Forest model votes for cis), the probabilities are weighted against each other, and so the model with a higher probability was taken as the final prediction. The Random Forest model's prediction outputs are

further adjusted by changing the threshold of the prediction probability. As a result, the predictions that don't make this threshold aren't included in the vote, hence removing less confident predictions. When the threshold is equal to or greater than 0.50, the MCC of the two classifier ensembles were 0.40. Increasing the threshold from 0.50 to 0.65 improved the combined model's performance with an MCC score of 0.42. Further increase in the threshold leads to fewer and fewer predictions from the Random Forest being taken to and so leads to the MCC score plateauing for thresholds greater than 0.70. However, the MCC score was at its highest at 0.65 and dropped with an increasing threshold.

MCC	RF threshold (>=)
0.40	0.50
0.40	0.55
0.41	0.60
0.42	0.65
0.40	0.70
0.39	0.75
0.39	0.80
0.39	0.85
0.39	0.90

Table.3. *Shows MCC scores in different Random Forest probability thresholds. >= indicates thresholds equal to and greater.*

Discussion:

The goal of this project was to build a binary machine learning classifier that predicts the conformational form of proline (e.g. cis or trans). The majority of naturally occurring proline isomers in folded protein is in the trans form (about 90-95%) (Alderson et al. 2018). As a result, the first problem faced when tackling cis/trans classification was the imbalance between the two classes. The Pisces dataset used in this project is a closer distribution of cis/trans classes, with about 4% of the dataset being cis proline.

Therefore, to tackle this imbalance, three approaches were carried out to balance the overwhelming majority of the trans dataset. The first approach was to randomly oversample the cis dataset and so match the number of trans dataset. However, the oversampling was done after the testing samples were removed to prevent a misleading MCC score. Models built using this method produced MCC scores ranging from 0.04 to 0.09, with an increasing number of trees in the Random Forest leading to a decrease in MCC. In contrast to oversampling the minority class leading to a better score than using the imbalanced dataset in other machine learning models. Oversampling the cis dataset led to no significant improvements in MCC; this might be due to overfitting the cis proline datasets. This can be seen as the number of trees increases the performance of the model drops. As a result, the model might have learnt the oversampled cis proline data, and so anything slightly different to is predicted as trans proline. Undersampling the majority class (trans prolines) produced significantly better scores than the oversampling method. This is illustrated by the lowest MCC score being 0.30 using Ant encoder and window size of 6. The undersampling models also produced an equivalent MCC score (0.40) to one of the two classifiers built using PSSMs (Exarchos et al., 2009). The advantage XGBoost has over the Random Forest models is its ability to work with imbalanced data. By scaling the weights and delta parameters, the XBoost model produced better scores than the Random Forest undersampled model and produced a score better than the model built using PSSMs score by Song et al. but didn't perform better than their model with secondary structure information along with PSSMs (MCC of 0.43). However, the ensemble model built using a combination of XGBoost and Random Forest closed this gap by producing an MCC score of 0.42. Since the task was to build a predictor that can work on Antibody loops, a model that doesn't depend on sequence PSSMs was needed. Without using PSSMs score, the machine learning models were built using known secondary structures. However, in Antibody loop modelling, the surrounding secondary structure is known, so this won't cause an issue.

Future Work:

Adding distance between the first and last amino acid showed promising results and so might improve the performance of the models further. Also looking at what the performance of the model would be when the two neighbouring amino acid residues are in a loop. Also finding out what the preforms would be when using secondary structure prediction algorithms to predict the secondary structure information.

GitHub directory:

https://github.com/A-Hareed/cis-proline-pred

References:

Abhinandan, K.R., and Andrew C. Martin. 2010. “Analysis and prediction of VH/VL packing in antibodies. Protein Engineering Design and Selection.” *Protein Engineering, Design & Selection* 23 (May): 689-697. 10.1093/protein/gzq043.

Adzhubei, A.A, and M. J. Sternberg. 1994. “Conservation of polyproline II helices in homologous proteins: implications for structure prediction by model building.” *Protein Science* 3:2395–2410. 10.1002/pro.5560031223.

Alderson, T. Reid, Jung Ho Lee, Cyril Charlier, Jinfa Ying, and Ad Bax. 2018. “Propensity for cis-proline formation in unfolded proteins.” *Chembiochem* 19:37–42.

Davide, Chicco, and Giuseppe Jurman. 2020. “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation.” *BMC Genomics* 21:1-13. https://doi.org/10.1186/s12864-019-6413-7.

Eisenberg, D., R. Weiss, T. Terwilliger, and W. Wilcox. 1982. “Faraday Symposia of the Chemical Society,.” 17:109–120.

Exarchos, Konstantinos P., Costas Papaloukas, Themis P. Exarchos, Anastassios N. Troganis, and Dimitrios I. Fotiadis. 2009. “Prediction of cis/trans isomerization using feature selection and support vector machines.” *Genomics, Proteomics & Bioinformatics* 7:138-142. https://doi.org/10.1016/j.jbi.2008.05.006.

Gu, Shengyin, Olivier Poch, Bernd Hamann, and Patrice Koehl. 2007. “A Geometric Representation of Protein Sequences.” *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 135-142.

Gustafson, Chelsea L., Nicole C. Parsley, Hande Asimgil, Hsiau-Wei Lee, Christopher Ahlbach, Alicia K. Michael, Haiyan Xu, et al. 2017. “A slow conformational switch in the BMAL1 transactivation domain modulates circadian rhythms.” *Molecular Cell* 66:447-457.

Henikoff, S., and J G Henikoff. 1992. “Amino acid substitution matrices from protein blocks.” *Proc. Natl. Acad. Sci. USA* 89:10915-10919.

Hsin, Jing-Ping, and James L. Manley. 2012. “The RNA polymerase II CTD coordinates transcription and RNA processing.” *Genes Dev* 26:2119–2137.

Li, Jie, and Patrice Koehl. 2014. “3D representations of amino acids—applications to protein sequence comparison and classification.” *Computational and Structural Biotechnology Journal* 11 (18): 47-58. https://doi.org/10.1016/j.csbj.2014.09.001.

Lummis, Sarah C. R., Darren L. Beene, Lori W. Lee, Henry A. Lester, R. William Broadhurs, and Dennis A. Dougherty. 2005. “Cis-trans isomerisation at a proline opens the pore of aneurotransmitter–gated ion channel.” *Nature* 438:248-252.

Maetschke, Stefan Renard, Michael Towsey, and Mikael Bodén. 2005. “BLOMAP: AN ENCODING OF AMINO ACIDS WHICH IMPROVES SIGNAL PEPTIDE CLEAVAGE SITE PREDICTION.” *Asia Pacific Bioinformatics Conference*, 141-150. https://doi.org/10.1142/9781860947322_0014.

Morgan, Alexander A., and Edward Rubenstein. 2013. “Proline: The Distribution, Frequency, Positioning, and Common Functional Roles of Proline and Polyproline Sequences in the Human Proteome.” *PLoS One* 8:e53785.

Song, Jiangning, Kevin Burrage, Zheng Yuan, and Thomas Huber. 2006. “Prediction of cis/trans isomerization in proteins using PSI-BLAST profiles and secondary structure information.” *BMC Bioinformatics* 7:124.

Wang, Chen, Chengyuan Deng, and Suzhen Wang. 2020. “Imbalance-XGBoost: leveraging weighted and focal losses for binary lab el-imbalance d classification with XGBoost.” *Pattern Recognition Letters* 136:190-197. https://doi.org/10.1016/j.patrec.2020.05.035.

Wang, Guoli, and Roland L. Dunbrack. 2003. “PISCES: a protein sequence culling server.” *Bioinformatics* 19:1589-1591. 10.1093/bioinformatics/btg224.

Wedemeyer, William J., Ervin Welker, and Harold A. Scheraga. 2002. “Proline Cis-Trans Isomerization and Protein Folding.” *Biochemistry* 41:14637-14644.

Zimmerman, S. Scott, and Harold A. Scheraga. 1976. “Stability of Cis, Trans, and Nonplanar Peptide Groups.” *Macromolecules* 9:408-416.

Appendix:

PDBsecstr output	Type of Secondary Structure	One-hot encoding
-	coil	0 0
B	bridge	0 0
E/e	beta strand	0 1
G/g	3_10 helix	1 0
H/h	alpha helix	1 0
S	bend	0 0
T/t	turn	0 0

Table.1. Shows the secondary structure information output by PDBsecstr and the one-hot encoding used to convert the nominal data into binary data for machine learning.