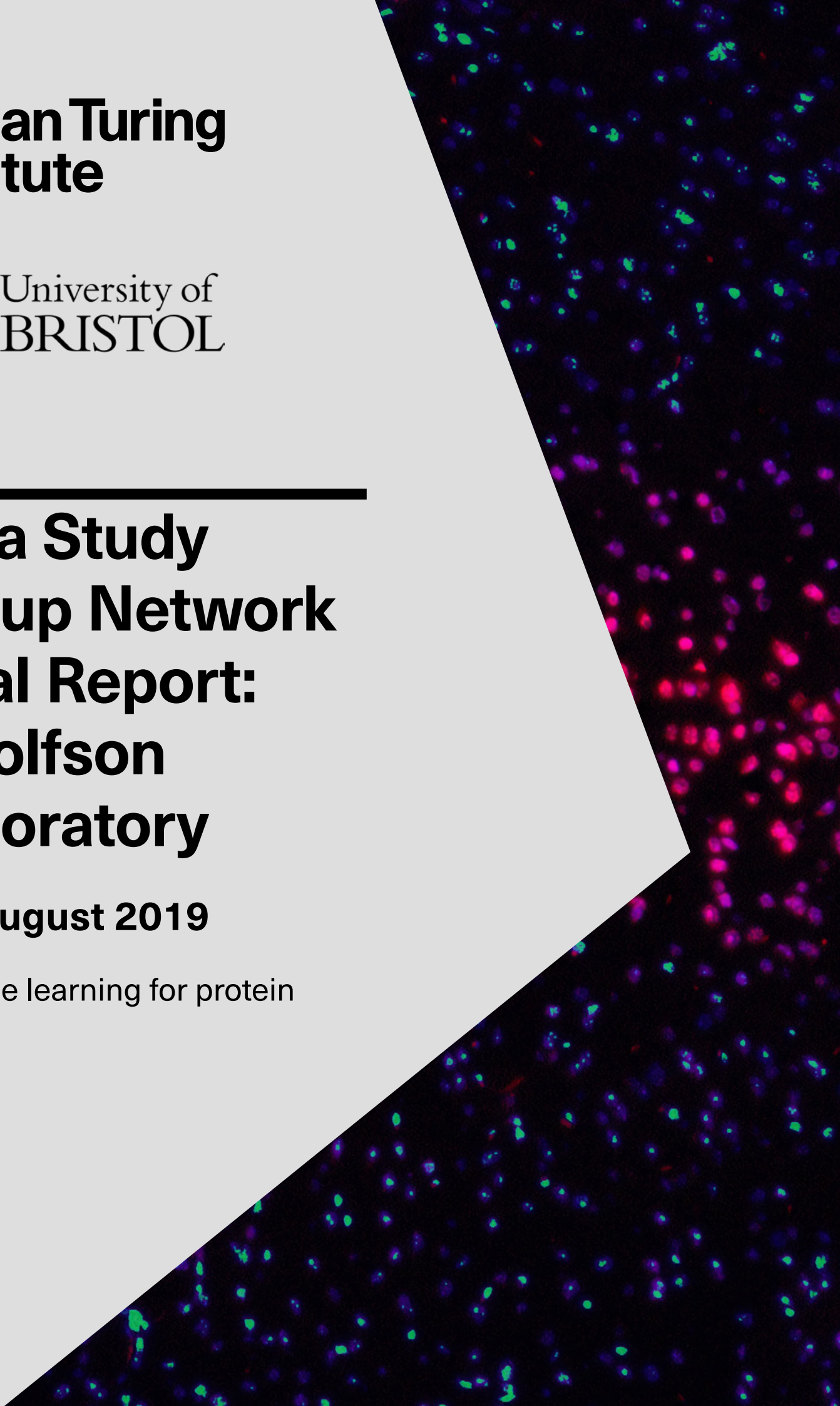
The Alan Turing Institute

University of BRISTOL

Data Study Group Network Final Report: Woolfson Laboratory

5–9 August 2019

Machine learning for protein folding



https://doi.org/10.5281/zenodo.3877119

This work was supported by Wave 1 of The UKRI Strategic Priorities Fund under the EPSRC Grant EP/T001569/1, 'AI for Science and Government' programme at The Alan Turing Institute

DSG 18. Machine learning for protein folding

Stephanie Seiermann, Christopher Doris, Misa Ogura,
Amit Kumar Jaiswal, Sydney Vertigan and Qingfen Yu

November 2019

1 Executive summary

1.1 Challenge overview

This report presents the outputs of a week-long collaboration between the Alan Turing Institute (A.T.I.) and Woolfson laboratory in the School of Chemistry at the University of Bristol, to predict different states of a type of protein fold called coiled coils (CCs) (Woolfson (2017)[1] which are the structural motifs that consist of two or more α-helices winding around each other, from linear sequences of amino acids by machine learning methods.

1.2 Data overview

The team was given a data-set consisting of amino acid sequences known to form coiled coils. The data-set consists of the amino acid sequence information and related structural information such as the number of α helices participating in the fold (the oligomeric state, i.e. dimer, trimer and tetramer for 2, 3 and 4 helices, respectively), the orientation of the alpha-helices with respect to each other (i.e. anti-parallel/parallel), whether the folded protein is formed out of one or more protein chains, and whether it is formed by identical or non-identical sequences (homomer or heteromers). A seven residues repeat found in an alpha helix coiled-coil, called a heptad, enables their assembly.

1.3 Main objectives

The main objective of the data study group was to develop a machine learning model that can, if given a sequence of amino acids, be able to predict whether this sequence will participate in either a) a parallel dimer, b) an anti-parallel dimer or c) any other structure such as a trimer. It was neglected whether this sequence folds with another homomer or heteromer, or whether it binds with a sequence from the same or a different protein chain. Also the orientation for any non-dimer structure was neglected.

[1] Coiled-Coil Design: Updated and Upgraded https://www.ncbi.nlm.nih.gov/pubmed/28101858

The second objective was to predict the above-mentioned classes using the sequence and the information about whether a sequence binds with another homomer or a heteromer.

1.4 Approach

The folding behaviour of coiled coils is mainly driven by the hydrophobic versus hydrophilic behaviour of amino acids and the location of an amino acid in the sequence (as decoded in the so-called registry). The main data scientific approach was to use the string of amino acid codes to predict the protein structures mentioned above as a supervised classification problem. This is similar to the standard data science approaches to deal with classifying strings of text.

In particular five different modeling approaches were used:

- Statistical and string-based feature engineering and classification using Random Forest
- String-Kernels measuring the similarity between two sequences and classification with Support Vector Machines
- Word embedding and text classification using neural nets using a library called FastText
- Pattern recognition within sequences using Recurrent Neural Networks (RNNs)
- Convolutional Neural Nets (CNNs) using adjacency matrices generated from right-padded adjacency matrices between sequences

The performance of all of those models was compared to a baseline derived from LOGICOIL.

For the second problem an ensemble approach was implemented that first uses CNNs to recognise patterns within the sequences and then uses this and other features such as homomer/heteromer in several tree-based models to predict the final class.

1.5 Main Conclusions

It was clear that the ensemble approach which includes extra features in addition to the sequence as inputs (Section 5) was the most successful, gaining 88% accuracy. We believe this is an suitable model for the problem posed as it does not seem to overfit. For the models that learn only based on the sequence, a plateau of 74% was reached, which we believe is the limit of what can be achieved with machine learning models for this approach.

1.6 Limitations

The data-set provided has very imbalanced data in where about 80% of observations were dimers and the rest higher oligomeric states, see table 2.1. Also 68% of all models were anti-parallel dimers, meaning bias could have be created in our predictions. Besides that, only coiled coils that actually fold were present, there was negative examples, i.e that of information about coiled coils that do not fold.

1.7 Recommendations and further work

- In addition to the sequence strings, more information such as the volume and distance of amino acids can be applied to our models.
- Our predictions can be improved by allocating more time and/or computing resources using a more time-consuming method, e.g. the inductive logic programming, which requires a deep understanding of background knowledge of amino acids and proteins.
- The unbiased data-set containing the entire amino-acid sequences and/or sequences from other secondary structures such as β-sheets and loops can be used for further training.
- An important feature, the mutations of one or more amino acids in a sequence, can be processed using the logistic regression method.

2 Problem Formulation and Data Science Approaches

2.1 Problem Formulation

The description of the "ML for protein folding" presented two challenges for the data study group:

(a) Predict protein structures based on sequence data from the CC+ database;

(b) Establish new design rules for protein sequences to enable generation of synthetic coiled coils.

However, after the presentation and the initial catch-up with the Challenge Owner, it was agreed to focus on first challenge in this data study group.

Furthermore, due to the significant imbalance within the data set (see table 2.1), the Challenge Owner suggested that we address the problem in two stages:

1. (a.i) Given one singular sequence (corresponding to one alpha helix), can we predict whether this sequence folds into:
    - Parallel Dimers
    - Anti-parallel Dimers
    - Something else

Oligomeric State	Orientation	Count
2	Anti-parallel	1,752
2	Parallel	391
3	Anti-parallel	120
3	Parallel	171
4	Anti-parallel	146
4	Parallel	61
5	Anti-parallel	1
5	Parallel	10
6	Parallel	9

Table 1: Counts of oligomeric states in the data-set.

2. (a.ii) Can we extend those predictions by adding sub-classes?

2.2 Data Science Approaches

Throughout the data study week, we focused on the first (a.i) problem. The majority of the approaches treat the presented problem as a supervised classification problem and attempt to solve it by extracting features from the actual amino acid sequences (see Section 4). The approaches distinguish each other by (a) the type of features extracted from the sequence and (b) the modeling approach used for classification. Sections 4.1 and 4.2 focus on traditional machine learning approaches such as Random Forests and Support Vector Machines (SVM), sections 4.3 - 4.5 use Deep Learning approaches with section 4.3 focusing on on text classification and sections 4.4 and 4.5 on Recurrent Neural Nets (RNN) and Convolutional Neural Nets (CNN) respectively.

The first approach (section 4.1) uses a combination of statistical features (e.g., length of the actual sequence, number of hydrophobic amino acids in the sequence) as well as the occurrences of different n-grams in a sequence string and trains a random Forest.

Section 4.2 implements custom string-kernel and uses a SVM for classification and two different string-kernels were tried. The first one counts how often a specific amino acid occurs in two sequences. The second one counts how often a specific amino acid motif, i.e., an amino acid sub-sequence made of 1 to n amino acids (with n being the longest sequence in the data-set) is occurring across two sequences.

The third approach (section 4.3) uses FastText for text-classification. Each sequence is treated as one document (to be classified as a specific type) and each amino acid in a sequence is treated as one word within that document. A neural net is trained to do the classification.

In section 4.4 a RNN, a deep learning neural net suitable for textual pattern recognition, was trained on the first 50 one-hot-encoded amino acids of a sequence.

Another deep learning experiment is presented in section 4.5 where a CNN, a

neural net commonly used for image recognition, was trained on directed graph adjacency matrices which were constructed between pairs sequences.

Section 4 ends with a comparison of the above mentioned approaches with one another and with a benchmark model. As benchmark model we were considering the (aggregated) predictions from LOGICOIL (Vincent et al (2013)[2]).

The modelling approach presented in section 5 does not only use the sequence as input, but also the information whether a sequence binds with a homomer or a heteromer.

Apart from the String-Kernel in combinations with SVM approaches, all experiments use a 80/20 train/test split to ensure comparability across modelling results. The SVM had to differ due to publicly available String-Kernels not being able to accommodate different shape of train and test matrices.

3 Data Overview

The following sections describe the provided data-set, any data issues that have been identified, any data cleansing and data enrichment activities that have been performed, results from the exploratory data analysis and their implications for the modelling approaches. Suggestions of how to overcome data limitations will be covered in section 7: Future Work and Research Avenues.

3.1 Dataset Description

The provided data-set is from CC+, a database used by the Woolfson Group at the School of Chemistry from the University of Bristol. It is data collected during biological experiments.

The study group was provided with a CSV file called "CCFold_data_for_Turing.csv" consisting of 3,168 rows. Each row represents one part of a protein chain sequence that forms part of a protein structure (e.g., anti-parallel homomer dimers).

For each row, the following columns where provided:

- **PDBID_ChainID_StartPosition_EndPosition.** A unique identifier consisting of the protein ID (PBDID), the protein chain (ChainID) as well as the start location (StartPosition) and end location (EndPosition) of the sequence in the protein chain.
- **Sequence.** A string consisting of the one-letter amino acid codes represented in the sequence.
- **Register.** A string with the same length as the sequence consisting of the letters 'abcdefg' decoding where an amino acid from the sequence can be found in within the heptad structure.

[2]Vincent, Green, Woolfson (2013): LOGICOIL- multi-state prediction of coiled-coil oligomeric state https://doi.org/10.1093/bioinformatics/bts648

- **Hetro/Homo Oligomer.** A flag indicating whether the protein structure is consisting of homomers (i.e. a copy of the same sequence coming together to form a structure) or heteromers (i.e. two or more different sequences coming together to form a structure).
- **Same/Many Chains.** A flag whether the sequence folds with other sequences from the same or other chains to form a protein structure.
- **Parallel/Anti-parallel.** A flag indicating whether the resulting structure is classified as parallel or anti-parallel. This depends on the orientation of the sequence in the fold.
- **Canonical/Non-Canonical.** A flag indicating whether a sequence is canonical, in that it exactly follows the standard coiled coil heptad structure (starting with a hydrophobic amino acid in position 'a' and ending with a hydrophilic amino acid in position 'g'). Otherwise it is classified as non-canonical.
- **Oligomeric State.** An integer decoding the oligomeric state of the protein fold (2 = dimer, 3 = trimer, 4 = tetramer etc.) The highest oligomeric state in the data-set is 6.

3.2 Data Quality Issues

The data was first checked for duplicates and completeness. There were no duplicated rows detected but we discovered issues with completeness.

3.2.1 Data Issue 1: Some assemblies missing sequences

- For homomer assemblies, the common sequence is listed once in the dataset. The oligomeric state will then indicate how often this sequence is repeated to form the protein structure. Hence, for all homomer assemblies, all participating sequences are known.
- For heteromer assemblies, each of its sequences appears as consecutive rows. We've found that not all heteromer assemblies are fully present in the dataset (e.g., 4pxu).
- Generally we only have sequences that as alpha helices engage in some kind of coiled coil folding, but not any other type of secondary structures that does not. This limits the use of the data as explained in section "Problem Formulation and Data Science Approaches" but opens avenues for future research.

3.2.2 Data Issue 2: Duplicate sequences

There are instances where the sequences appeared in more than one row. This was seen in two cases:

1. For some of duplicate sequences, which had all columns the same apart from the canonical column. The challenge owner confirmed that the canonical/non-canonical form has no impact on the protein folding behaviour. So we agreed to drop this column completely from the data set and dedupe the data afterwards (see section Data Preparation).

2. For the remaining duplicate sequences, the data was the same apart from the ChainID given as part of the PBDID column (2oto_A_158_190 and 2oto_B_158_190). This is caused by duplicate assemblies (see www.rcsb.org/3d-view/2OTO). We identified which sequences belonged to one assembly and afterwards dropped all duplicate assemblies (see section Data Preparation).

After implementing these two fixes, there were still around 1% duplicate sequences in the dataset, but this is valid because in nature the same sequence could occur in several different assemblies.

3.2.3 Data Issue 3: Unknown amino acid X identified in some sequences

The Challenge Owner confirmed that they are used 'X' to represent an unknown or experimentally modified amino acid and that we can either treat this as a placeholder or replace it with 'A', the simplest amino acid depending on which made more sense for the modelling approach. We decided to leave X for now.

3.2.4 Data Preparation

We performed the following steps to prepare the data for the different modelling approaches:

- Converted Same/ManyChain, Antiparallel/Parallel and Homo/Heteromer columns into binary columns
- Split PBDID_ChainID_StartPosition_EndPosition into four columns: pbdid, chainid, start, end
- Dropped canonical/non-canonical column and deduped dataset
- Created an assembly id that groups all rows that only differ in sequence in chain id (or only in chain id) into assemblies
- Dropped duplicate assemblies
- Assigned class labels: 0 = Parallel Dimers, 1 = Anti-parallel Dimers, 2 = Rest

Pre-processed data set is called 'CCPlus_nice_deduped.csv'.

Finally, we split this data set into train ('CCPlus_nice_deduped_train.csv') and test ('CCPlus_nice_deduped_test.csv') datasets by using a random 80:20 split for each class.

Class	No.
0 (parallel dimer)	413
1 (antiparallel dimer)	2162
2 (other)	593

Table 2: Count of structure types in the data-set.

State	No.
2 (dimer)	2575
3 (trimer)	318
4 (tetramer)	246
5 (pentamer)	13
6 (hexamer)	16
Parallel	683
Antiparallel	2485
Homo	476
Hetero	2692
Same chain	2134
Many chain	1034

Table 3: Detailed breakdown of structure types in the data-set.

3.2.5 Exploratory Data Analysis

We counted the number of sequences with various properties, see tables 2, 3 and 4. By far there are more anti-parallel hetero-dimers than any other category. Excluding these, the rest of the sequences are roughly equally balanced between parallel and anti-parallel and between homomers and heteromers.

As shown in table 5, we counted the lengths of the sequences. 33% of sequences have the minimum length of 15. The rest are distributed as in table 5.

4 Classification based on sequence

In this section we evaluate several classification methods on the same problem: using only a sequence of amino acids, predict its assembly class (0, 1 or 2).

All methods (except, for technical reasons, string kernels) are trained on the same 80% of the data and tested on the remaining 20%.

The largest class is 1, accounting for 63% of testing samples, giving a baseline against which we can compare the overall accuracy of our methods.

State	Parallel?	Homomer?	#
2	false	false	2161
2	false	true	1
2	true	false	152
2	true	true	261
3	false	false	138
3	false	true	2
3	true	false	45
3	true	true	133
4	false	false	156
4	false	true	26
4	true	false	28
4	true	true	36
5	false	true	1
5	true	true	12
6	true	false	12
6	true	true	4

Table 4: Count of states in the dataset.

Quantile	Length
33%	15
50% (median)	19
75%	25
90%	39
95%	50
99%	82
100% (max)	148

Table 5: Count of sequence lengths in the data-set.

4.1 Random Forests

Random forests (Ho, 1995)[3] are among a class of decision-tree-ensemble methods for classification. We proceed by converting our input variable-length sequences into a fixed set of features, which are then fed into such a classifier.

4.1.1 Feature Engineering

We begin with a set of predicates on single amino acids:

- Hydrophobic, not hydrophobic, charged, polar and amphipathic. These return true if the amino acid has the corresponding property.
- A predicate for each of the 20 amino acids. These return true if the amino acid is this particular one.

From these, we constructed a set of features, whose input is a sequence of amino acids:

- Length of the sequence;
- For each predicate, the proportion of the sequence for which the predicate is true; and
- For each pair of predicates and a gap size of 1, 3, 4, 7, 10, 11 or 14, the proportion of times the two predicates hold the given gap apart.

As an example of the last kind, the feature '"I E 3"' is the proportion of times the pattern 'I..E' has matches in the sequence. There are $5 + 20 = 25$ predicates and $1 + 25 + 25^2 \cdot 7 = 4401$ features in total.

We also tried various subsets of these features, and considered but did not try the following kinds of features:

- Starting/ending point of the sequence;
- First/last amino acid in the sequence; and
- Counts of amino acids at specific register positions.

Hyperparameters Four classifiers were tried:

- Random forest (julia DecisionTree.jl[4])
- Random forest (scikit-learn[5])
- Extra trees classifier (scikit-learn[6])

[3] Ho (1995): Random Decision Forests https://web.archive.org/web/20160417030218/http://ect.bell-labs.com/who/tkh/publications/papers/odt.pdf

[4] `<https://github.com/bensadeghi/DecisionTree.jl>`

[5] `<https://scikit-learn.org/stable/>`

[6] `<https://scikit-learn.org/stable/>`

- Gradient boosting classifier (scikit-learn[7])

There was no appreciable difference in predictive performance between these classifiers. Increasing the number of trees used in the ensemble from the default 10 to 1000 was necessary for predictive performance, but tweaking other hyperparameters made little difference. For sub-sequence analysis, the 'sklearn' implementation of random forest was used.

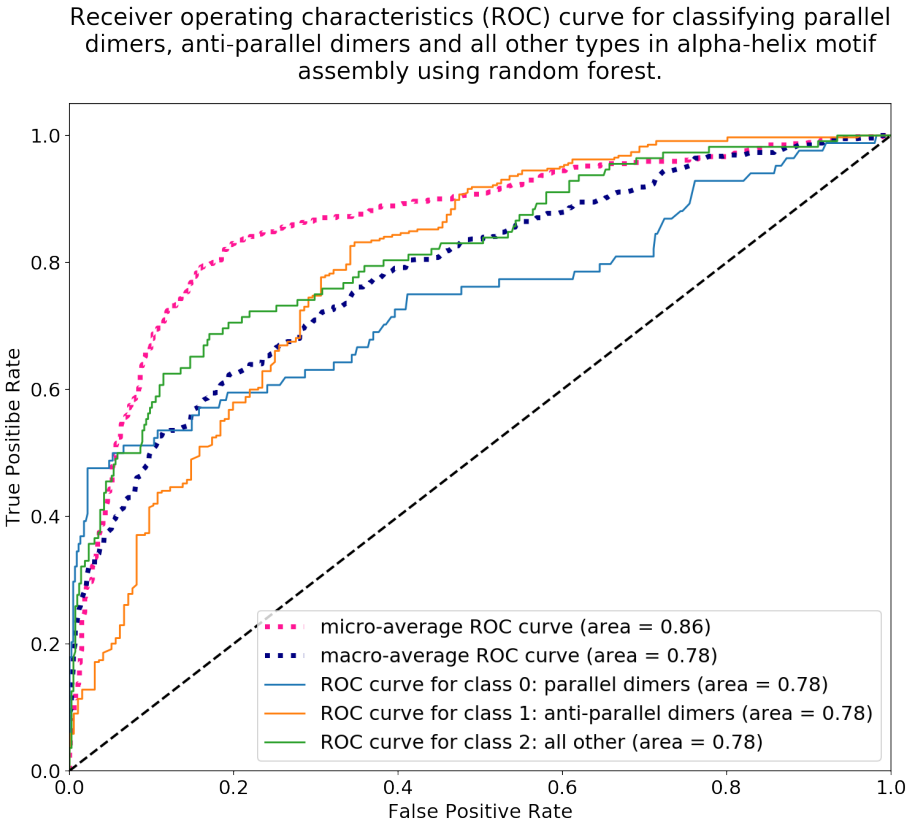
4.1.3 Results The classifier was trained on 80% of the data and tested on the remaining 20%. This method achieves an overall accuracy of 73.9% on the test set. The confusion matrix is below.

Actual	Predicted		
	0	1	2
0	29	51	4
1	4	335	6
2	3	73	36

The performance is fairly consistent among classes. For example, 25% of each class is recalled with precision 90% on the test data. Class-wise and overall ROC (receiver-operator characteristic) curves are in figure 1.

Feature Importance The following are the 25 most important features, according to sklearn:4.4.2

[7] <https://scikit-learn.org/stable/>


Receiver operating characteristics (ROC) curve for classifying parallel dimers, anti-parallel dimers and all other types in alpha-helix motif assembly using random forest.
True Positibe Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (area = 0.86)
macro-average ROC curve (area = 0.78)
ROC curve for class 0: parallel dimers (area = 0.78)
ROC curve for class 1: anti-parallel dimers (area = 0.78)
ROC curve for class 2: all other (area = 0.78)

“#”	0.00746655
“E !hy 14”	0.00365937
“ch !hy 14”	0.00317781
“!hy E 14”	0.00299664
“!hy !hy 14”	0.00297458
“ch po 14”	0.00269947
“ch ch 14”	0.00269464
“!hy Q 14”	0.00254171
“E ch 14”	0.00254025
“!hy po 14”	0.00252651
“K !hy 14”	0.00246237
“!hy ch 14”	0.0024579
“!hy R 14”	0.00228616
“L hy 14”	0.00222497
“Q !hy 10”	0.00196236
“ch Q 14”	0.00194308
“hy hy 4”	0.00193031
“hy hy 7”	0.00192004
“hy hy 3”	0.00190391
“L !hy 3”	0.00188827
“N L 3”	0.00178264
“po !hy 14”	0.00174291
“hy !hy 7”	0.00170973
“ch E 14”	0.00170568
“E E 14”	0.00170499

The most important feature found in this analysis is the length of the sequence (given by # above). The second most important is ‘E !hy 14‘, meaning the fraction of the sequence for which there is an ‘E‘ followed 14 steps later by a non-hydrophobic amino acid.

Using the shap package[8] (Lundberg et al, 2018[9]) the SHAP (SHapley Additive exPlanations) values for each feature for each class was computed, and the 25 features with the greatest average absolute SHAP value on the data are given in table 6.

Again length and ‘E !hy 14‘ are highly predictive. The most important features are generally on a gap of 14, which is notable since most known useful features generally operate on a gap of three or four as Heptad repeats have hydrophobic residues at the first and fourth position that defines the number of helices in the assembly.[10]

The important features with a gap of three or four are ‘L N 4’, ‘N L 3’, ‘L !hy 3’, ‘hy I 3’, ‘hy hy 4’, ‘Q !hy 3’, ‘I hy 4’. Except for ‘Q !hy 3’, these are

[8] <https://github.com/slundberg/shap>

[9] Lundberg, Erion, Lee (2018): Consistent Individualized Feature Attribution for Tree Ensembles https://arxiv.org/abs/1802.03888

[10] DN Woolfson, ”The design of coiled-coil structures and assemblies,” Advances in protein chemistry, 70, 79-112

Class 0		Class 1		Class 2	
“#”	0.00383301	”#”	0.0106915	“#”	0.00687821
“E !hy 14”	0.00265948	“E !hy 14”	0.00585588	“ch !hy 14”	0.00348338
“!hy E 14”	0.00221572	“ch !hy 14”	0.00494725	“E !hy 14”	0.00320706
“E ch 14”	0.00187313	“!hy E 14”	0.00447062	“ch ch 14”	0.00266233
“L N 4”	0.00169594	“ch ch 14”	0.0040577	“ch po 14”	0.00254272
“N L 3”	0.00168336	“ch po 14”	0.00403411	“!hy ch 14”	0.00241962
“!hy Q 14”	0.00164162	“!hy !hy 14”	0.0039521	“!hy po 14”	0.00238629
“!hy !hy 14”	0.00163163	“!hy ch 14”	0.00381551	“!hy !hy 14”	0.00238346
“ch po 14”	0.00149944	“!hy po 14”	0.00380119	“K !hy 14”	0.00237832
“ch !hy 14”	0.00147416	“E ch 14”	0.00357711	“!hy E 14”	0.00227064
“!hy po 14”	0.00143594	“K !hy 14”	0.00352134	“L hy 14”	0.00224802
“ch ch 14”	0.00140146	“!hy Q 14”	0.00340556	“!hy R 14”	0.00198373
“!hy ch 14”	0.0013993	“L hy 14”	0.00313929	“I hy 14”	0.00188525
“E E 14”	0.00131097	“!hy R 14”	0.0031365	“!hy Q 14”	0.00177081
“L E 10”	0.00130598	“Q !hy 10”	0.00255788	“I hy 11”	0.00175556
“Q !hy 10”	0.00123974	“ch Q 14”	0.00249839	“E ch 14”	0.0017182
“ch Q 14”	0.00122363	“po !hy 14”	0.00249594	“K ch 14”	0.00153402
vL !hy 3”	0.00119298	“!hy K 14”	0.00238343	“hy I 3”	0.00146062
“!hy R 14”	0.00115423	“po hy 14”	0.00228368	“po hy 14”	0.00144207
“K !hy 14”	0.00115191	“E E 14”	0.0022122	“!hy K 14”	0.00143164
“L ch 10”	0.0011502	“K ch 14”	0.00217838	“po !hy 14”	0.00141389
“po ch 11”	0.00112304	“ch E 14”	0.00209788	“Q !hy 10”	0.0013297
“L L 7”	0.00109069	“po ch 14”	0.00208171	“hy !hy 7”	0.00131364
“hy hy 4”	0.00108735	“Q !hy 3”	0.00200535	“I hy 4”	0.00129454
“po !hy 14”	0.0010845	“L L 14”	0.00198613	“hy I 10”	0.00128603

Table 6: The 25 features in the SHapley Additive exPlanations approach to the data with the greatest mean vaue.

consistent with known useful features (e.g. Vincent et al, 2013[11]). Curiously, 'Q !hy 3' does not involve any hydrophobics, and could hint towards structure away from registers 'a' and 'd'.

4.2 String-Kernels and SVM

4.2.1 Data Transformation using string-kernels

The string kernel transforms the data based on how many subsequences are the same between two strings. In position i, j of the matrix there is a count of how many subsequences are the same between string i and string j - making a diagonally symmetric matrix.

Three types of kernel were tried: a weighted string-kernel for protein fold recognition (Nojoomi & Koehl (2017)[12] & Zhang et al (2018)[13]), the spectrum kernel, which compares subsequences of length k throughout all strings, and the 'all subsequences' kernel that compares every subsequence, regardless of size, throughout all of the strings.

After testing the weighted string kernel, we found the output was unreliable because the shape of the output matrix did not match that which we expected, so we decided not to use this kernel.

4.2.2 Support vector machine approach.

We used the intuition behind 'Efficient Approximation Algorithms for String Kernel Based Sequence Classification and Mismatch String Kernels for SVM Protein Classification'.

In order to use the kernel with the 'sklearn svm' function, the parameter kernel must be set equal to the kernel function. Then x_train and y_train used in 'clf.fit' will be passed into that kernel. This meant that there were problems as our string kernel needed to take in strings but the 'svm.fit' function would not allow us to input the strings needed to be passed onto the kernel. Therefore, we had to write our own string kernel by modifying the code used for the string kernels we were attempting to use with SVM. We found a very simple way of using a string kernel with sklearn's svm online and adapted this for the kernel we wanted to use. It meant that we would input our string as numbers into the SVM but the kernel would use the original strings. In order to use this method the training and testing split had to be equal, so a 50% was used for training and the remaining 50% for testing. On the original strings from the sequence column we used the spectrum kernel. We tried different variations of k (k is the number of characters that are compared by the string kernel).

[11] Vincent, Green, Woolfson (2013): LOGICOIL- multi-state prediction of coiled-coil oligomeric state https://doi.org/10.1093/bioinformatics/bts648

[12] Nojoomi & Koehl (2017): A weighted string kernel for protein fold recognition https://bmcbioinformatics.biomedcentral.com/articles/10/1186/s12859-017-1795-5

[13] Zhang et al (2018): String-kernel Documentation https://buildmedia.readthedocs.org/media/pdf/string-kernel/latest/string-kernel.pdf

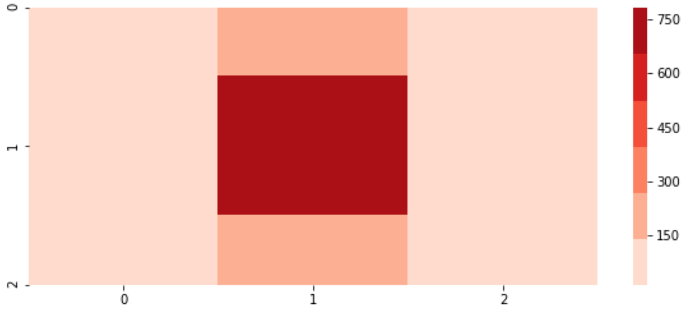


Figure 1: Confusion Matrix for Spectrum Kernel

The original strings didn't represent the relationships we wanted so next we tried the model on just the hydrophobic characters of the string. Hydrophobic residues at first and fourth positions form a hydrophobic line on a helix that allows the helices to assemble together. They also define the number of helices in the assembly and hence, we focused on them. The 'all subsequence' kernel never finished running on the original strings or the hydrophobic strings but we believe this would have given a better result as it will represent more relationships between the strings.

4.2.3 Results The spectrum kernel with $k = 1$ (comparing single characters) gained 61% accuracy when searching through the strings for any singular same character, see figure 1. We believe this 61% is because 63% of the data is class 1 so the model leans towards predicting things as this class (shown in the confusion matrix, true value y-axis, predicted value x-axis) due to the unbalanced data.

This was the best result we obtained with the original strings. The model trained with $k = 2$ also performed quite well. We believed $k = 8$ might perform well as it would have two hydrophobic characters in each substring but this only did just better than random.

With the hydrophobic strings we gained 58% accuracy using the spectrum kernel with $k = 1$.

4.2.3 FastText

FastText[14] (Joulin et al, 2016[15]) is a tool for word embedding and text classification.

We used it as a classifier by converting our amino acid sequence into a list of words, which is presented to FastText as a document labelled with its class.

[14] `<https://fasttext.cc/>`

[15] Joulin et al (2016): Bag of Tricks for Efficient Text Classification https://arxiv.org/abs/1607.01759

4.3.1 Document Preparation To convert a sequence to a "document", we first performed some character substitution, and then split the sequence into multiple words. Our preferred character substitution scheme was to replace each non-hydrophobic with a lowercase '"x"' and leave all hydrophobics as-is. For example, "KLDNLMDLMGELVIA" becomes "xLxxLMxLMGxLVIA". Other schemes tried include leaving the string as-is, or replacing all non-hydrophobics by a letter representing whether they are charged ('c'), polar ('p') or amphipathic ('a').

Our preferred method of converting this string to multiple words was to take all 6-grams from the string on a stride of 2. For example, '"xLxxLMxLMGxLVIA"' becomes '"xLxxLM xxLMxL LMxLMG xLMGxL MGxLVI"'. Other methods tried included using the whole string as a single word, or taking n-grams on a stride of s for various n and s. For example with $n = s = 1$, each letter is a separate word.

4.3.2 Hyperparameters We trained FastText using the one-versus-all ("ova") loss function for 200 epochs with a learning rate of 0.1. We set 'wordNgrams=5' so that the model considers 5-tuples of words at a time. We found that taking our words to be 6-grams on a stride of 2 and setting 'wordNgrams=5' was necessary for the algorithm to learn relationships between amino acids reasonably far apart in the sequence. In particular, a 5-gram of words in this scheme covers 14 amino acids, or 2 heptads. Other parameters were kept at their default.

4.3.3 Results FastText was trained on 80% of the data and tested on the other 20%.

This method achieves an overall accuracy of 73.9% on the test set. The confusion matrix is below.

Actual	Predicted		
	0	1	2
0	28	52	4
1	6	335	4
2	6	67	39

Performance varies widely across classes. When FastText predicts class 1 (anti-parallel dimer) with sufficient confidence, it recalls about 30% of this class with 100% precision. We recall 25% of class 0 (parallel dimer) with around 90% precision. Class 2 (other) is the most difficult class, recalling 25% with precision 70%. Class-wise and overall receiver-operator characteristic (ROC) curves are shown in figure 2.

4.2.4 Recurrent Neural Networks (RNN)

Recurrent neural networks RNN[16] is a type of deep neural network suitable for recognising patterns in sequences. Given the nature of our dataset, we felt that it was natural to choose RNN.

[16] https://en.wikipedia.org/wiki/Recurrent_neural_network

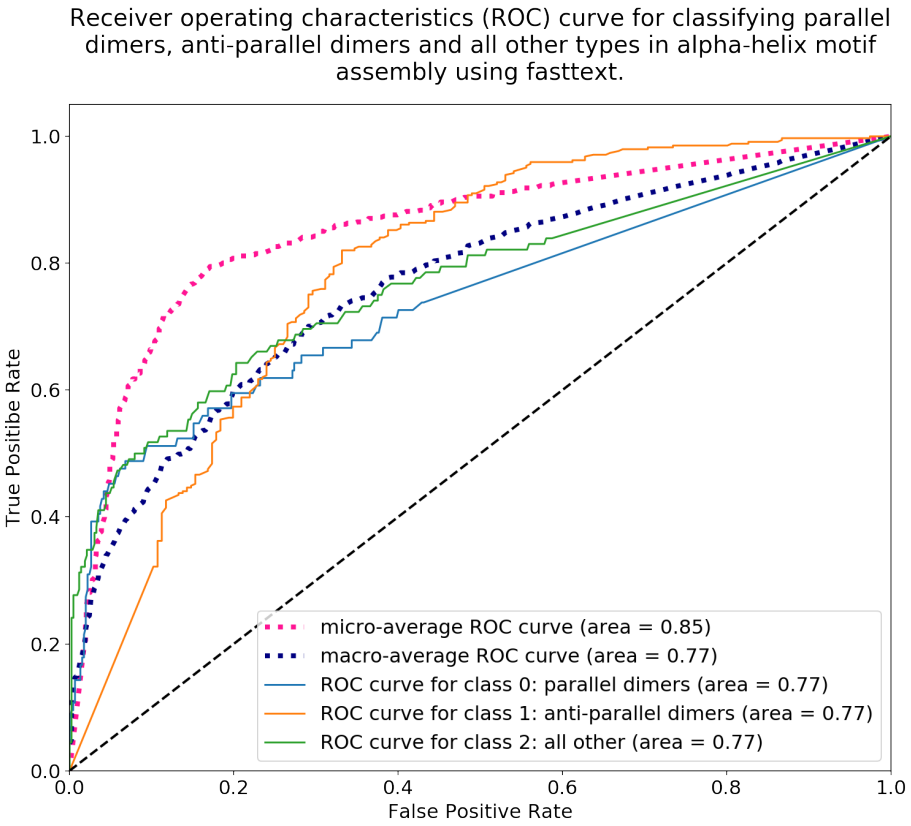


Figure 2: Receiver-operating characteristics (ROC) for FastText.

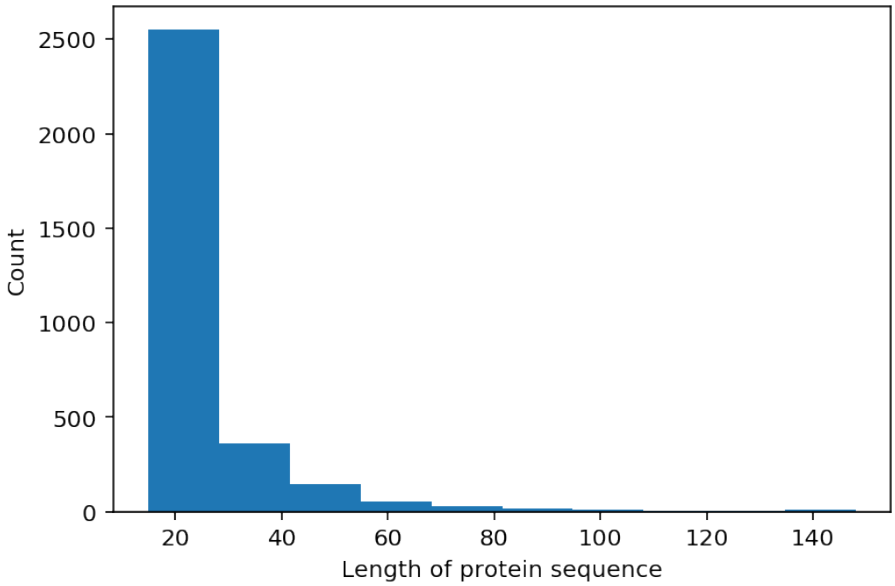


Figure 3: Distribution of protein sequence length in the data-set.

4.4.1 Data Preprocessing Firstly, each letter representing 20 amino acids was mapped to a corresponding integer 1 to 20. A letter 'X' in amino acid sequences represents the presence of a modified amino acid. Because we lacked the information on the original amino acid from which modified amino acids were derived, we replaced all 'X' with the simplest amino acid found in our helices, alanine, denoted by a letter 'A' (glycine is the simplest amino acid, but as it does not have a side chain and is a helix breaker, the probability of it occurring in the helix is very small).

Secondly, the length of the strings were normalised to 50. The distribution of the length of protein sequences was right-skewed as shown in Figure 3. As over 95% of the sequences were less than or equal to 50 characters long, we decided to normalise their lengths to 50, enabling us to train the neural network with mini-batch. We took the first 50 characters for sequences that were longer than 50, and for those that were shorter than 50, we right-padded them with an integer that wasn't part of encoding process. The resulting strings were further processed with one-hot-encoding.

RNN Architecture The RNN model was constructed using PyTorch v1.0.0.

We utilised a long short-term memory (LSTM) cell [17], which has an enhanced ability to retain information across sequences, over an RNN cell.

A LSTM layer with 2 hidden layers and 64 hidden dimension was followed

[17] https://dl.acm.org/citation.cfm?id=1246450

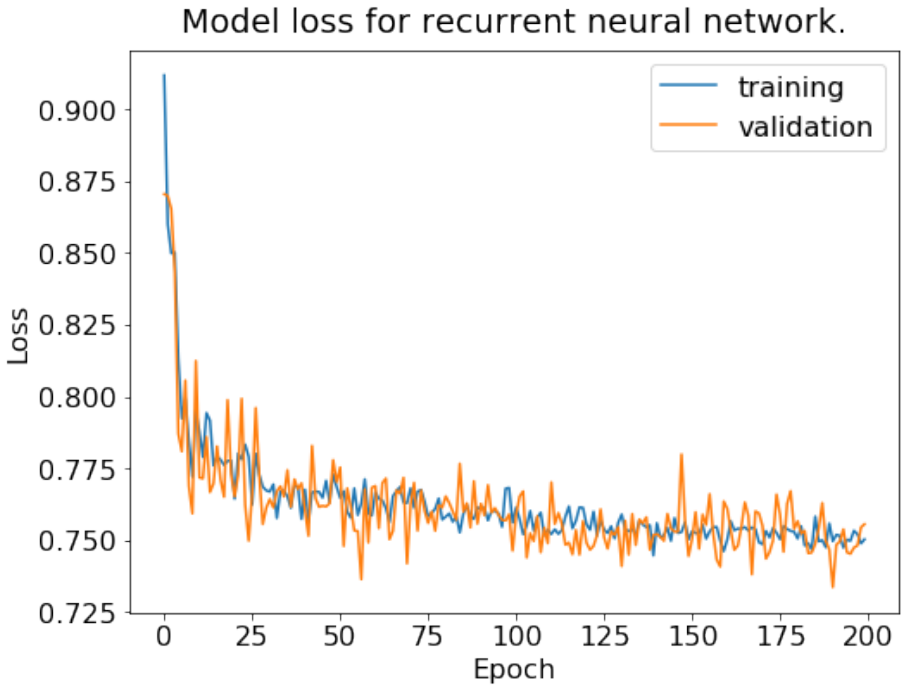


Figure 4: Training and validation loss for the RNN.

by a fully-connected layer with an output dimension of 3, corresponding to the 3 classes of interest: parallel dimers, anti-parallel dimers and all other.

Training The network was trained for 200 epochs with an initial learning rate of 0.001. We optimized the model weights using Adam optimizer[18], with cross entropy loss[19] as a loss function, which combines log-softmax and negative log loss operations.

The training and validation losses (Figure 5), as well as validation accuracy (Figure 6) were tracked throughout the training to monitor the model performance.

Learning rate schedular[20] was used to reduce the learning rate by a factor of 0.9, if the validation loss did not increase after 10 epochs, giving the network a chance to potentially find its way out of local minima.

4.4.4 Evaluation The model achieved an overall accuracy of 68.75% on a held-out test data-set. The ROC curves are given in figure 4.2.4.

[18] <https://arxiv.org/abs/1412.6980>
[19] <https://arxiv.org/abs/1412.6980>
[20] <https://pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.ReduceLROnPlateau>

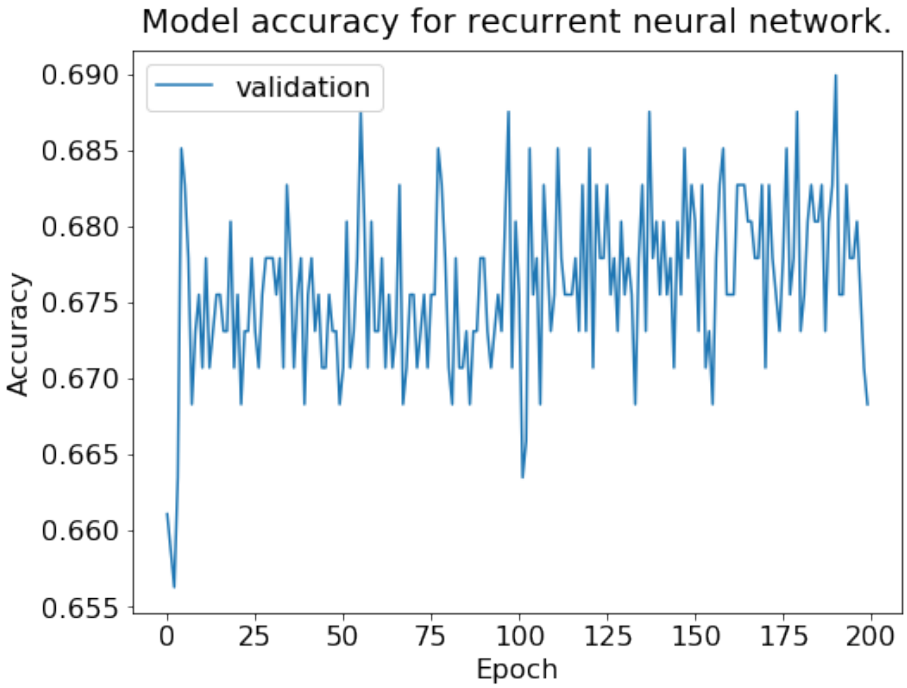
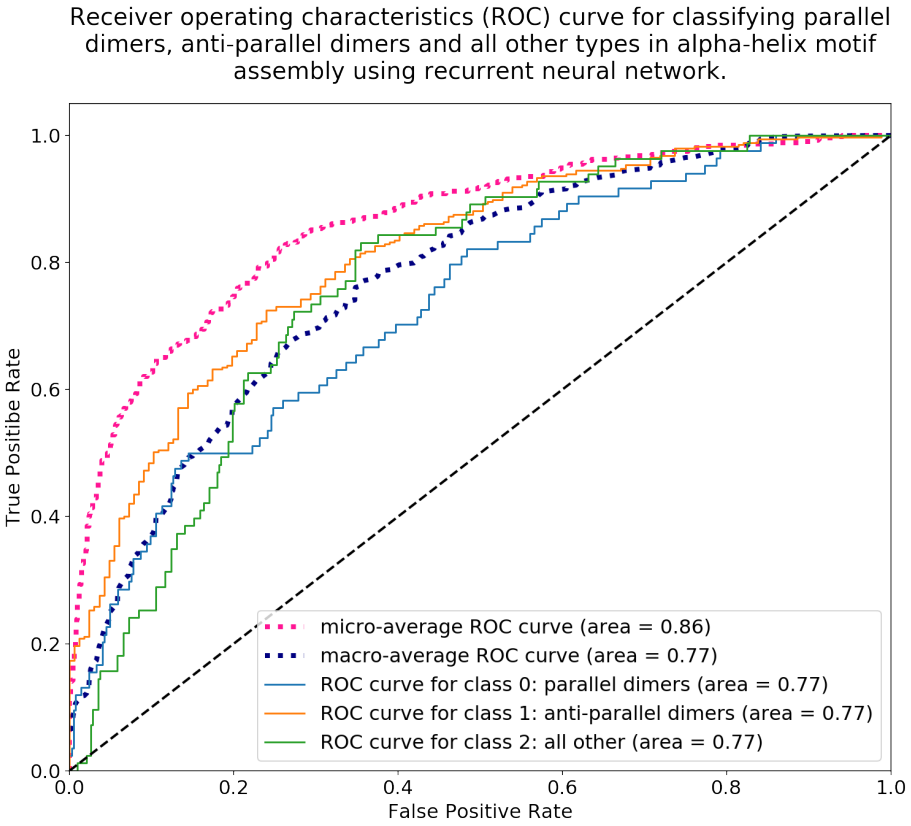


Figure 5: Validation set accuracy for the RNN.


Receiver operating characteristics (ROC) curve for classifying parallel dimers, anti-parallel dimers and all other types in alpha-helix motif assembly using recurrent neural network.
True Positibe Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (area = 0.86)
macro-average ROC curve (area = 0.77)
ROC curve for class 0: parallel dimers (area = 0.77)
ROC curve for class 1: anti-parallel dimers (area = 0.77)
ROC curve for class 2: all other (area = 0.77)

4.5. Convolutional Neural Network (CNN) In searching for other ways to represent our data-set, we rationalised that if we considered a sequence of amino acids as a directed graph, we should be able to use adjacency matrix to represent both the order and frequency of letters.

This was preferable to the RNN approach, as we did not have set a sequence length threshold because each sequence, regardless of its length, produces one matrix.

Treating adjacency matrices as one-channel images, we decided to utilise CNN[21], which is a type of deep neural network suitable for recognising patterns in images.

Data Preprocessing Firstly, each letter representing 20 amino acids was mapped to a corresponding integer 1 to 20. A letter 'X' in amino acid sequences represents the presence of a modified amino acid. Because we lacked the information on the original amino acid from which modified amino acids were derived, we replaced all 'X' with the simplest amino acid, alanine, denoted by a letter'A'.

Secondly, for each sequence, we generated an empty 20x20 matrix filled with zeros to represent all possible combination of neighbouring residues, with rows representing the current letter and columns representing the next letter. Then for each letter in a sequence, we inspected the letter that came after, and incremented the corresponding position in the matrix (i.e. matrix[current][next] += 1).

4.5.2 CNN architecture A single 2-dimensional convolutional layers with an output dimension of 16 was followed by a batch normalisation layer as well as a max pooling layer, consisting a convolutional block. The convolutional block was then followed by a single fully-connected layer via a dropout layer. The CNN model was constructed PyTorch v1.0.0.

4.5.3 Training The network was trained for 1000 epochs with an initial learning rate of 0.0001. The model weights were optimized using the Adam optimizer[22], with cross entropy loss[23] as a loss function, which combines log-softmax and negative log loss operations.

The training and validation losses, as well as validation accuracy were tracked throughout the training to monitor the model performance.

Learning rate schedular[24] was used to reduce the learning rate by a factor of 0.9, if the validation loss did not increase after 10 epochs, giving the network a chance to potentially find its way out of local minima. The model loss on training and validation sets and the validation accuracy during training are given in figures 6 and 7.

[21] <https://en.wikipedia.org/wiki/Convolutional_neural_network>
[22] <https://arxiv.org/abs/1412.6980>
[23] <https://arxiv.org/abs/1412.6980>
[24] <https://pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.

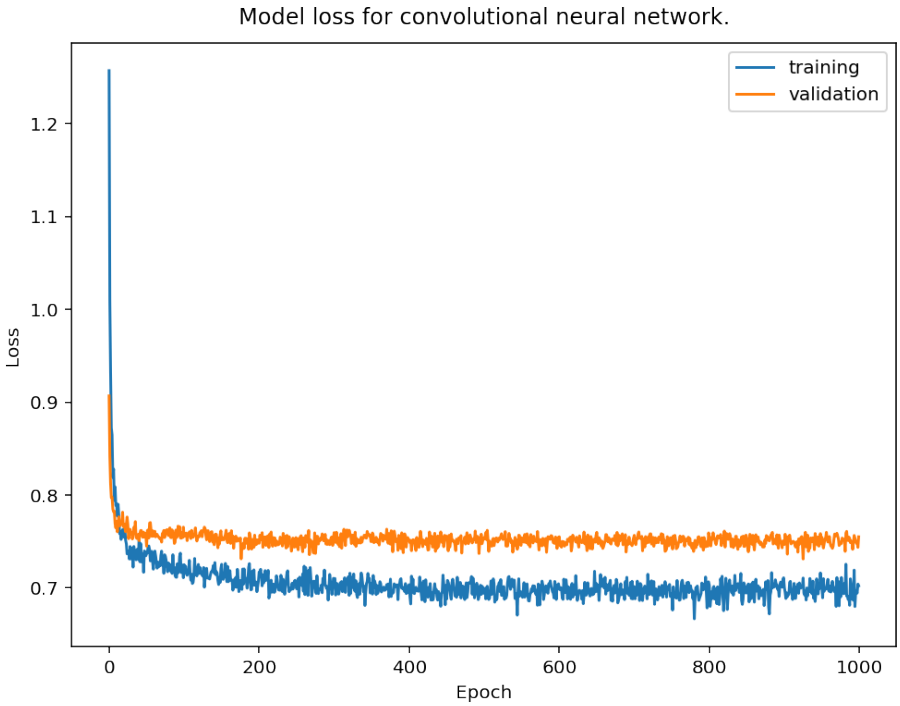


Figure 6: Loss on training and validation sets for the CNN.

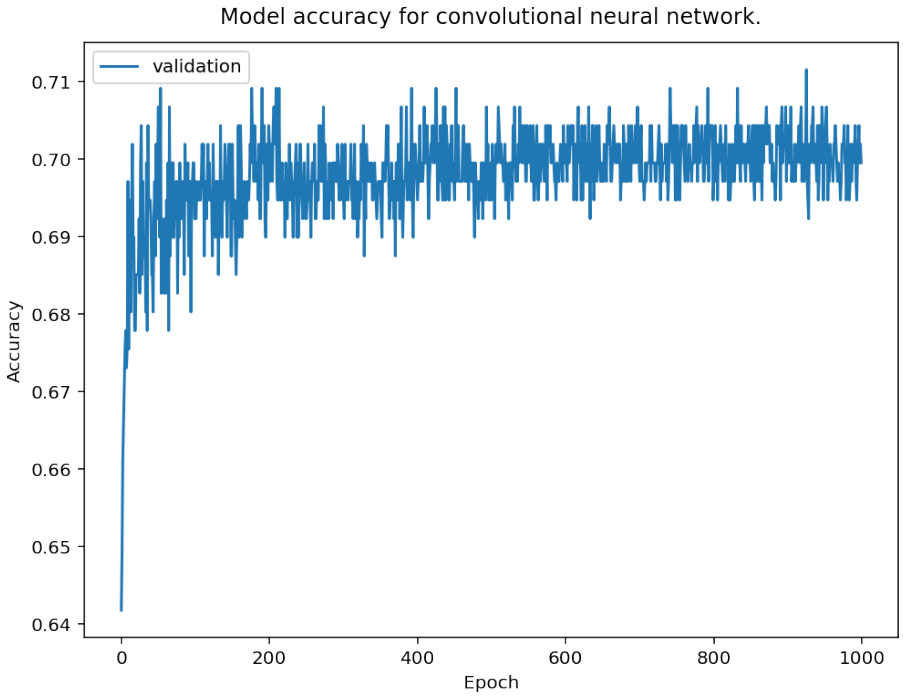


Figure 7: Validation accuracy for the CNN approach.

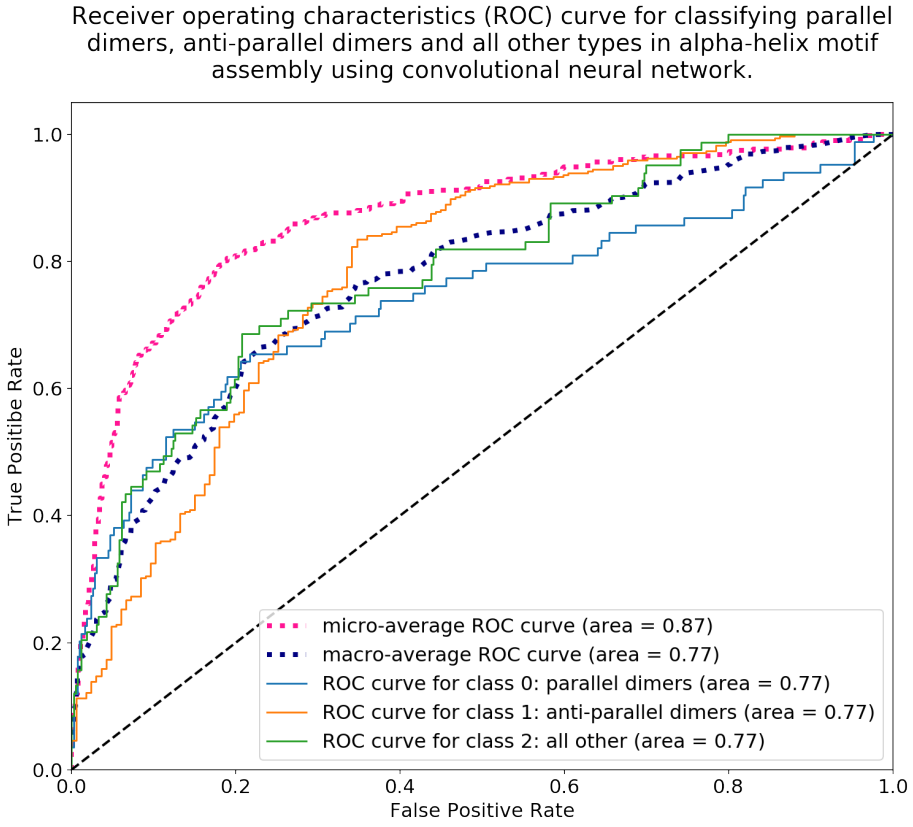


Figure 8: ROC curves for CNN

4.2.5 4.5.4 Evaluation

The model achieved an overall accuracy of 69.21% on a held-out test dataset. The ROC curves are shown in figure 8.

4.2.6 Model Performance Comparison

For comparison, we ran the LOGICOIL algorithm online[25] (Vincent et al, 2013[26]) on the dataset. LOGICOIL classifies sequences as either parallel dimer, anti-parallel dimer, trimer or tetramer. The first two classes correspond to our 0 and 1, and we assume the other classes correspond to our 2.

It achieves an overall accuracy of 52%. This is perhaps an unfair comparison since the online LOGICOIL tool was not trained on our data, and in particular

`ReduceLROnPlateau>`

[25] `<http://coiledcoils.chm.bris.ac.uk/LOGICOIL/>`

[26] Vincent, Green, Woolfson (2013): LOGICOIL- multi-state prediction of coiled-coil oligomeric state https://doi.org/10.1093/bioinformatics/bts648

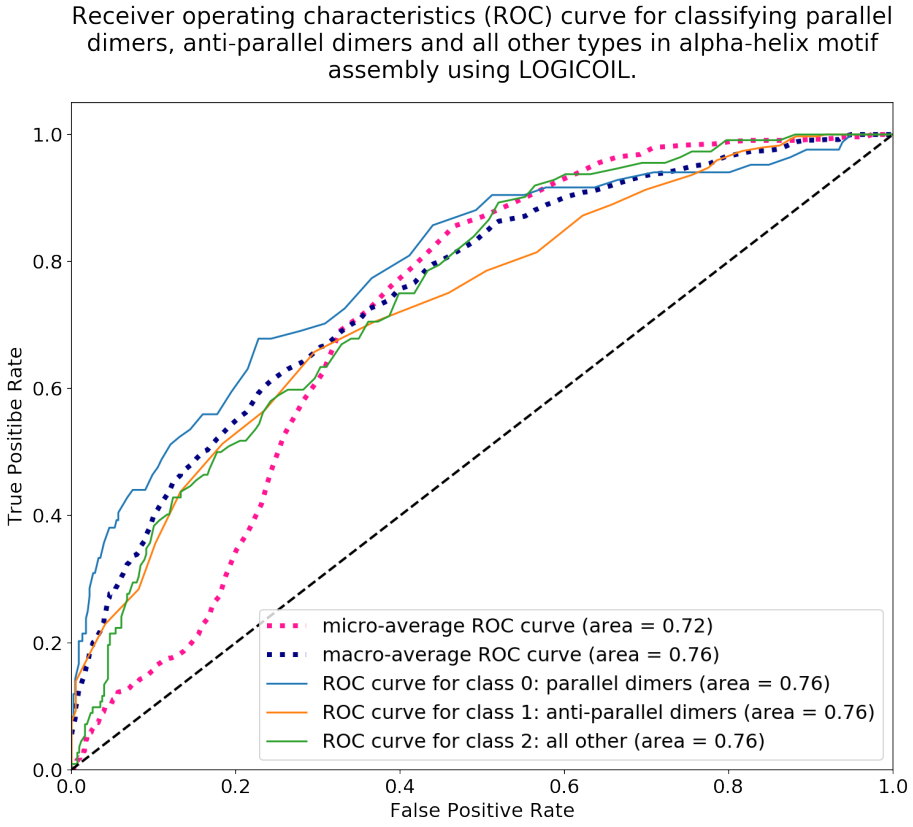


Figure 9: Receiver-operator characteristic on the logicoil data.

is forced to make predictions on pentamers and hexamers for which it was not trained. The confusion matrix is

Actual	Predicted		
	0	1	2
0	210	53	150
1	238	948	976
2	34	74	485

and the ROC curves are shown in figure 10.

The overall accuracy of each method is given in table 7. The 'Guess 1' method always outputs the largest class, 1.

For comparison, the overall ROC curves for each method are given in figure 10.

Method	Accuracy
LOGICOIL	52%
String Kernel + SVM	61%
Guess 1	62%
RNN	69%
CNN	69%
Random Forest	**74%**
FastText	**74%**

Table 7: The accuracy of each approach.

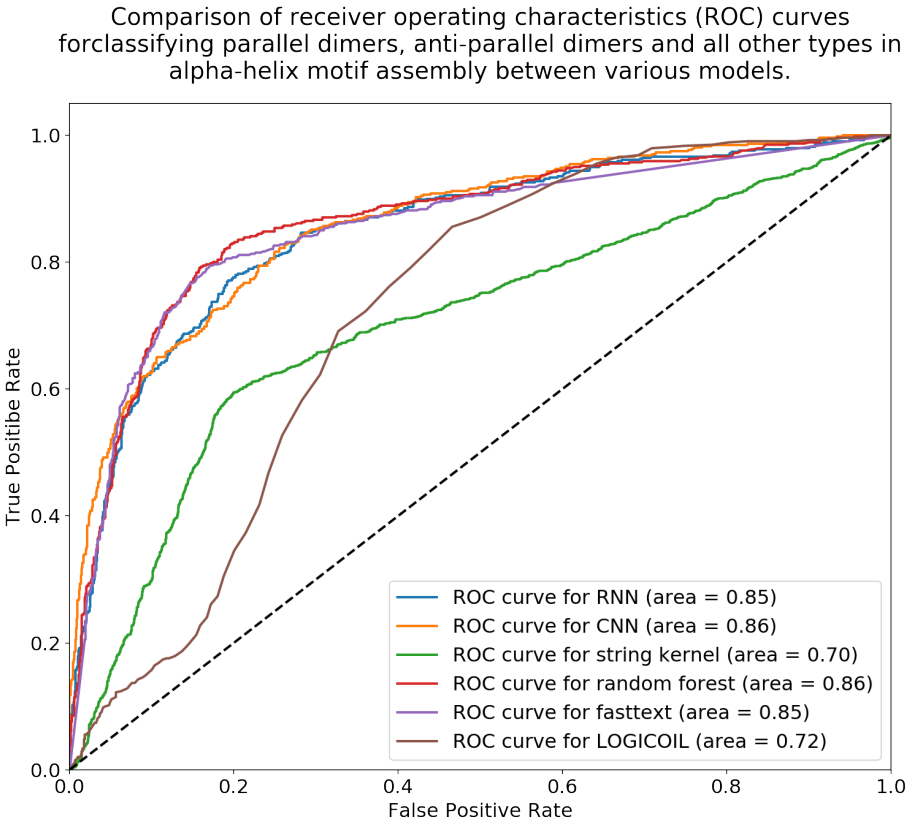


Figure 10: Receiver-operator characteristics for all the methods tried.

5 Classification based on sequence and other features

This section details the various neural network and tree-based models employed in understanding whether the sequences (helix) are actually making assemblies or not based on the two categorical features i.e., sequence arrangement (homomer/heteromer) and orientation (anti-parallel/parallel) respectively.

Firstly, we used text classification approach, which is inspired from language modelling and basic natural language processing (NLP) tasks, performed on amino acid sequence data-sets. In order to understand the formation of amino-acid sequences from perspective of NLP, we have performed some basic NLP operations like tokenizing, concatenating the given sequences by using a list of upper-case letters (excluding X) and lower-case letters ('a','b','c','d','e','f','g') as window-size of varying length provided we assume each given sequence is a helix.

Problem Overview Representing amino acid sequences as biological words is still an open problem. Here, we are characterising sequences by means of text classification approach to understand whether sequences assembles as dimers, trimers etc. based on the available categorical features.

5.1 Convolutional Neural Network (CNN) and Tree-based Models

1- and 2-Dimensional Word-based CNN Convolutional neural network of type 1D model process applies to one-dimensional sequences of data. The models extract features from sequences data and maps the internal feature of sequences provided the sequences have external information about amino acid codon. In our problem scenario, a 1D CNN is effective only for deriving features from fixed length of sequence of the overall dataset, whereas it is not so important where the feature is located in the segment.

The main difference between type 1D and 2D CNNs is the structure of the input data and the convolution kernel which moves across the data.

In both types of CNN, we input sequences as made of n length words (where n varies from 15 to 148 - the length of sequences), which represents a vector. The filters (convolution kernel) covers at least one word, where a height parameter depicting how many words a filter should take at once. In our CNN models, we used multiple filters in corresponding convolutional layers such as 8,16,32 etc. where the filter moves at least 2 times to fully scan the sequences.

5.2 Architecture The 1D and 2D convolutional NN architectures are shown in figures 11 and 12.

5.3 Data Transformation We have updated the data-sets with headers for each column followed by separating the first column by tokenizing in four dif-

```
   OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

       Input   #####          15
   Embedding   emb | -------------------       308     6.2%
               #####     15   14
      Conv1D    \|/  -------------------       464     9.3%
        relu   #####     15   16
     Dropout    | || -------------------         0     0.0%
               #####     15   16
      Conv1D    \|/  -------------------       264     5.3%
        relu   #####     15    8
MaxPooling1D   Y max -------------------         0     0.0%
               #####     15    8
     Flatten   ||||| -------------------         0     0.0%
               #####         120
       Dense   XXXXX -------------------      3872    77.8%
        relu   #####          32
     Dropout    | || -------------------         0     0.0%
               #####          32
       Dense   XXXXX -------------------        66     1.3%
     softmax   #####           2
```

Figure 11: 1-dimensional CNN model used.

```
   OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

       Input   #####     15  132    1
      Conv2D    \|/  -------------------        80     0.0%
        relu   #####     14  131   16
      Conv2D    \|/  -------------------      2080     0.1%
        relu   #####     13  130   32
MaxPooling2D   Y max -------------------         0     0.0%
               #####      7   65   32
     Dropout    | || -------------------         0     0.0%
               #####      7   65   32
      Conv2D    \|/  -------------------      8256     0.5%
        relu   #####      6   64   64
      Conv2D    \|/  -------------------     21074     1.2%
        relu   #####      5   63   82
     Flatten   ||||| -------------------         0     0.0%
               #####       25830
       Dense   XXXXX -------------------   1653184    98.0%
        relu   #####          64
       Dense   XXXXX -------------------      2080     0.1%
        relu   #####          32
     Dropout    | || -------------------         0     0.0%
               #####          32
       Dense   XXXXX -------------------        66     0.0%
     softmax   #####           2
```

Figure 12: 2-dimensional CNN model used.

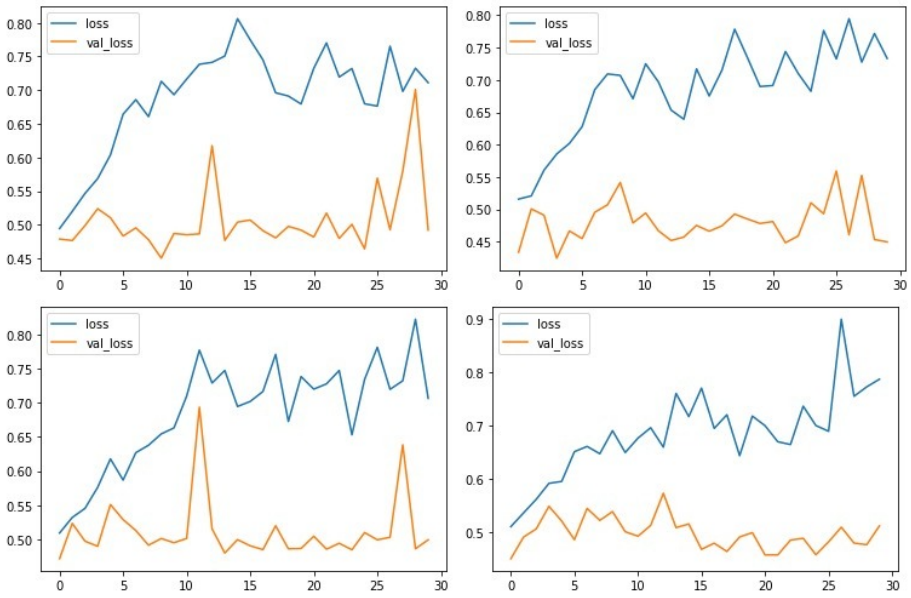


Figure 13: Training (loss) and validation (val_loss) losses for the 1D-CNN model with window sizes of 7 (top left) 14 (top right), 21 (bottom left) and 28 (bottom right).

ferent columns (includes first) having names ‘seq‘, ‘pdb‘, ‘chain_id‘, ‘startr‘ and ‘endr‘ with details given in _cleaned_data.csv_

While modeling, we employ two features by assigning Boolean values:

- Sequence arrangement (Homomer and Heteromer)
- Orientation (Parallel and Anti-parallel)

We split the original dataset using keras’ ‘train_test_split’ library (an automated approach) into 80% samples as train and 20% as test set.

5.4 Training For training, we have parameters such as ‘embedding_dim‘ of ‘7,14,21,28‘ which is basically an embedding window with ‘dropout‘ of 0.5 followed by ‘max_pooling‘ having size ‘1‘and then again using dropout on alternate dense layers. Our loss function uses ‘binary_crossentropy‘. We trained our 1D-CNN over ‘4596‘ parameters where our distribution of training is ‘80% and 20%‘ as per the train and test set. The dynamic loss distribution trained over ‘30 epochs‘ is reported in figure 11.

We performed adaptive learning on 2D-CNN model while training where we found that the training and validation losses converges after 5th epoch due to dynamic training of longer sequences using categorical features (H/HT), see figure 14.

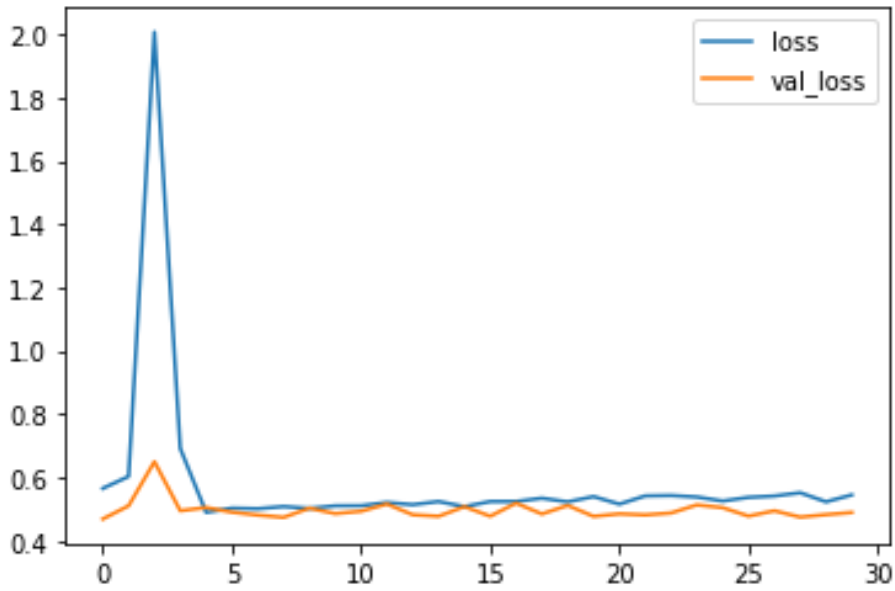


Figure 14: Training (loss) and validation (val_loss) losses for the 2D-CNN model.

Models	Normalized Score (Softmax)	Sequence Window Size	Feature
1D-CNN	82.43%	7	H / HT
1D-CNN	84.42%	14	H / HT
1D-CNN	81.61%	21	H / HT
1D-CNN	83.11%	28	H / HT
2D-CNN	88.95%	7	H / HT

5.5 Results This section details the evaluation of 1D-CNN and 2D-CNN models in which the prediction score is based on the root mean square propagation (RMSprop) learning method. The best model performance is of 2D-CNN and a huge improvement is due to learning the sequences in one after another from length '15' to '132' and rest of the sequences up to length 148 has been used as a test set for validation. From biological perspective, when protein sequences of varying length gets populated and trying to assemble, such interactions follows the sequence properties. In table 8, we have validated with categorical features such as homomer/heteromer (H/HT).

The prediction pattern for 1D-CNN model seems to follow an alternate increase followed by consequent decrease in predicting sequence labels whether the helix assemble to form dimer, trimer etc. To understand this phenomena, we trained the model on different window size as shown in table 8. The prediction score for 1D-CNN shows that when multiple helix try to combine together to form dimer or trimer, it allows the specific helix to combine or move away depends on the contraction between the sequences. So, we have shown that, in

Models	Accuracy	Features
1D-CNN	84.42	H/HT
2D-CNN	**88.95**	**H/HT**
RidgerClassifierCV	**85.18**	**H/HT**
XGBoost Classifier	84.80	H/HT

Table 8: Caption

general, multiple helices combines to form dimers than trimer or tetramer as the lattice formation or other structural properties such as distance between the 'C-alpha' and 'residue', and volume.

5.1.1 II. Tree-based models

We employed the tree-based estimators which has built-in learning algorithm in order to improve robustness/generalisation over a single model. We use various tree-based distinguished models and boosting techniques from seven groups of estimators as well which are listed below:

- Ensembling method - AdaBoost/Bagging/Extra Trees/Gradient Boosting/Random Forest
- Gaussian process Classifier
- Linear Models - Logistic Regression/Passive Aggressive/Ridger Classfier/SGD Classifier/Perceptron (cross validation estimators)
- Naive Bayes - BernoulliNB/GaussianNB
- Scalable Vector Machine (SVM) - SV and Linear Classifier
- Tree - Decision trees and Extra trees classifier
- Discriminant Analysis - Linear and Quadratic
- XGBoost Classifier

Results This section reports the tree-based estimators and cross-validation models. The best models achieved around 80% accuracy.

Our tree-based baseline uses logistic regression and the best tree based model is Ridge classifier validator, see table 8, due to the fact that sequences in combination (which are actually dimer or trimer) improves over training for 30 epochs.

6 Future Work and Research Avenues

This section proposes avenues for future work and research. Those are either focused on extending the existing work for the problem (a) described in section 2.1 or proposed future research.

Additional Data As most modelling approaches plateau at an accuracy of about 73%, we think we might have reached a plateau that we might be able to overcome by training with a larger dataset.

Alternatively, additional data around the bulkiness of different amino acids like volume and distances can be used. The bulkiness of amino acid is a physical property that impacts the bonding behaviour of amino acids and therefore the folding behaviour of proteins. There are different ways to bring this information into the actual models. One idea would be manually creating bulkiness buckets, group amino acids into those buckets and count how often a particular bucket occurs in a sequence and use this as additional features for the Random Forests or the tree-based models presented in section 5.

Alternative Approaches A completely different approach to solve problems proposed would be Inductive Logic Programming, a logic programming based machine learning approach. This method uses facts created from the data we have and background knowledge as inputs. From these inputs it learns and creates rules to classify them. This however is very time-consuming as we would have to code up a lot of background knowledge. With the right expertise, background knowledge could be created about amino acids and the strings we have been provided with could be coded up as facts. This should create a rule that covers each class. The knowledge of how small changes to the amino acid completely change the properties could be something that would work very well with this method.

Unbiased Data and Re-formulation of the problem A described in section 3, the provided data-set only contains those alpha helices of the protein sequence that actually form coiled coils and bind with other coiled coils. So sequences from alpha helices that do not form coiled coils or form other secondary structures like beta sheets or loops are not given.

We suggest to use the entire protein sequence for further work. Especially for deep learning approaches learning patterns from sequences that do not fold might be useful and could increase performance. Apart from that, using the entire protein sequence would allow us to derive additional features like, for example, the length of the sequence between two binding coiled coils which might be an indicator for anti-parallel vs parallel orientation.

Feature importance In section 4.1.4 we began investigating which patterns in our sequences are most informative in classification. Some such patterns are already known, but some appear to be new, and in particular consider amino acids further apart in the sequence. On the other hand, our features are highly correlated which makes feature importance more difficult. More work is required in this regard.

Modelling of protein structure based on a switch of amino acids at a specific location We have found a huge number of sequences that only

distinguish themselves by one amino acid being different at one location in the chain. However, often, this results in a completely different structure. Having a deeper understanding of those patterns and being able to predict how a structure changes based on a switch in amino acids would help targeting the creation of synthetic proteins (see problem b in section 2). One potential modelling approach for this problem could be logistic regression.

Combined Approach In the future it would be desirable to combine the models from section 4. These models predicted different labels correctly e.g. some being more consistent at predicting label 2 than others. Therefore, stacking the methods might produce more accurate results.

Reinforcement Learning based Approach We intend to apply evolutionary algorithm for protein folding which used Q-learning to train the agent based on the given policy which comprises of how protein formation takes place including lattices in 2D/3D structures. Also, recent work found that the reinforcement learning approach is scalable and great potential to predict protein properties using deep reinforcement learning. In Section 5, we try to adopt such sort of learning method in 2D-CNN but it's so preliminary start to tackle protein folding problem.

7 Reading List

- Derevyanko (2018): TorchProteinLibrary: A computationally efficient, differentiable representation of protein structure https://arxiv.org/abs/1812.01108.pdf
- Farhan et al (2017): Efficient Approximation Algorithms for String Kernel Based Sequence Classification https://papers.nips.cc/paper/7269-efficient-approximation-algorithms-for-strings-kernel-based-sequence-classification.pdf
- Leslie et al (2002): Mismatch String Kernels for SVM Proten Classification https://papers.nips.cc/paper/2179-mistmatch-string-kernels-for-svm-protein-classification.pdf
- Li (2018): FoldingZero: Protein Folding from Scratchin Hydrophobic-Polar Model https://arxiv.org/abs/1812.00967.pdf

8 Team Members

Stephanie Seiermann, Data Science Consultant at BJSS, London. *Contributions: Facilitator and 4.2. String-Kernel and SVM*

Christopher Doris, Research Fellow, University of Bristol and Heilbronn Institute for Mathematical Research. *Contributions: 4.1 Random Forests, 4.3 FastText, 4.6 Comparison*

Misa Ogura, Research and Development Software Engineer, BBC, London. *Contributions: 4.4. Recurrent Neural Network (RNN) and 4.5. Convolutional Neural Network (CNN)*

Amit Kumar Jaiswal, PhD Researcher, University of Bedfordshire. *Contributions: 5. Word-based Convolutional Neural Network and Tree-Based Models and Reinforcement Learning section in Section 7*

Sydney Vertigan, MSc Mathematics of Cybersecurity Student, University of Bristol *Contributions: 4.2. String-Kernel and SVM and 4.5. Ideas behind CNN*

Qingfen Yu, Research Associate, Department of Chemistry, University College London. *Contributions: Sequence prediction using the LOGICOIL algorithm*

turing.ac.uk
@turinginst