# PROJECT REPORT

## 3 STATE SECONDARY STRUCTURE PREDICTOR

VILDE KALDHUSDAL

3D structure prediction is both time consuming and computationally demanding. The quality of the prediction also highly depends on the availability of homologous structures. Recently with the quick evolution of sequencing tools, the growth in sequences has outpaced the rate of new solved structures. Secondary structure and topology is a prerequisite for experimental studies of structure and function, and can be a starting point for attempts to model the 3D structure before molecular dynamics or simulated annealing.

In this report different machine learning models are explored with the help of the scikit-learn python liberary to create and optimize predictor for 3-state secondary structure.

**Methods**

the original dataset named dssp_3state.3line.txt, was split in  two parts 70/30, containing a total of  279 and 120 proteins respectively . All cross validation was performed on the biggest datset and testing on the smaller set.

The two datsets are located:

```
'../project/datsets/train_data0.7.txt
'../project/datsets/test_data0.3.txt
```

Ideally the full dataset should have been split randomly 70/30 several times over to be able to repeat the testing .

**Work flow Summary:**

1. Write a script that can parse the information from FASTA files and encode amino acids to binaries
2. Test different window sizes
3. Modify code to be compatible with position specific matrixes
4. Run grid search to find optimized parametres
5. Run classification report and confusion matrix on SVM, RF and DT
6. Choose and train the best model based on class report

Avilabilety :
The three predictors are to be found on my github, together with training and testing datasets.

https://github.com/vildeka/project

Description of all scripts required to crossvalidate, optimize and train regular and PSSM models:

1. `dataparser.py`
   - First function stores the information from the FASTA in a dictionary. `dictionary = { 'identifier':['sequence', 'topology' }`
   - The second and third function creates the finished X- and y-input vector respectively. The X-vector function creates the window size specified and adds the neccesary padding of 0s to ends of each feature in addition to the encoding
   - This script is imported by every script that uses regular X- and y_input vector
2. `crossval_windowsize.py`
   - Uses the sklearn module `cross_val_score` in a foor?? loop, in range of specified window sizes.
   - Prints a plot of window size against cross val scores.
3. `PSSM_parser.py`
   - Contains modified functions from `dataparser.py` in addition to the function `PSSM`_parser, that parses PSSM files.
   - The right side of the PSSM matrix are used for the X-vector and the values are divided by 100 to get percentage.
   - This script is imported by every script that uses PSSM X- and y_input vector

4. `grid_search.py` and `PSSM_grid_search.py`
    - Uses the sklearn module `cross_val_score` together with `cross_val_score` to attain the scores of the model given different combinations of parameters.
      e.g `parameters = {'C':[ 2, 2.5], 'gamma':[0.01, 0.05,]}`
5. `my_models_compare.py`
    - Only compares PSSM models.
    - Uses the sklearn modules `cross_val_score, classification_report` and `confusion_matrix.`
    - Used the models I had optimixzed from the grid search in addition to training a Decision Tree Classifier model with default parameters.
6. `train_model.py` and `PSSM_train_model.py`
    - trains the models with the optimized parameters.

```
aa_dict = { 'A':[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,],
             'C':[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,],
             'D':[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,],
             'E':[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,],
             'F':[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,],
             'G':[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,],
             'H':[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,],
             'I':[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,],
             'K':[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,],
             'L':[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,],
             'M':[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,],
             'N':[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,],
             'P':[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,],
             'Q':[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,],
             'R':[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,],
             'S':[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,],
             'T':[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,],
             'V':[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,],
             'W':[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,],
             'Y':[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,],
             '0':[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,] }

topologi_dict = {'H':1, 'S':2, 'C':3}
```

**Figure 1:** The features and labels were encoded using the following dictionaries The last key '0' only containing 0's depicts? the encoding for padding.

A *feature* is the same as the encoding of a word.
An *example* is the feature + the topology

```
x = features
y = topology

x = [array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]),
     array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
     array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])]


y = [ 3, 3, 1 ]
```

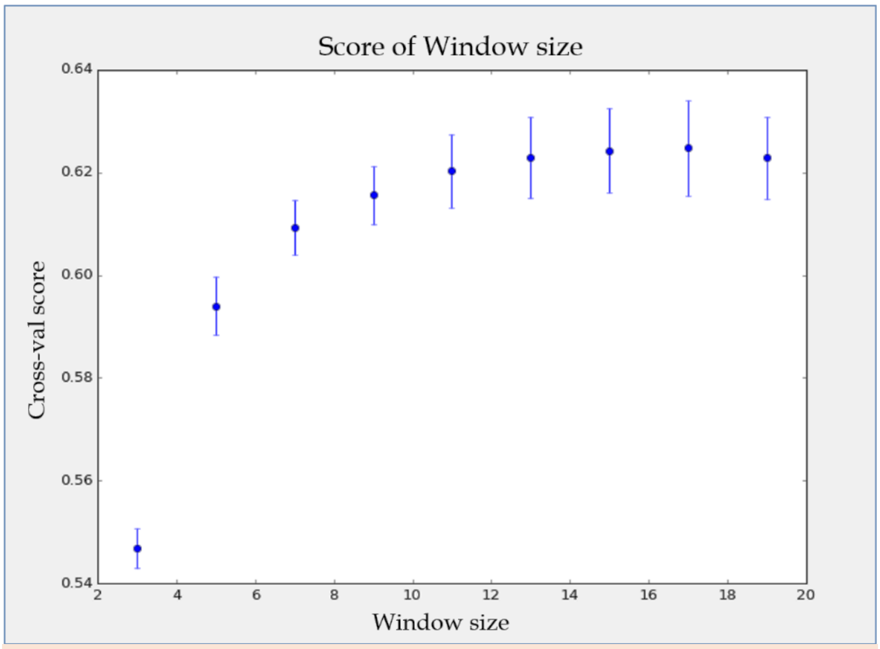
**Figure 2:** Shows 3 examples with a window size of three. Each array is split into three lines to easily visualize each window. One array depicts? a single feature.

## Results

### Window size:

Standard SVM with `rbf` kernel and `OneVsOne` wrapper, was used for testing the window size.

`OneVsOne` is an exhaustive method and is highly time consuming, but yields a more robust result. The cross-validation was performed on the whole dataset containing 399 sequences.



**Graph 1:** Windowsizes of 3, 5, 7, 9, 11, 13, 15, 17 and 19 were 5-fold cross-validated.

Ideally the window size search should have been combined with the testing of different kernels, but in consideration of time the default `rbf` kernel was decided  upon. This kernel was succesfully used by Guo J. et al.

Based on litterature a window size of around 15 is generally found to be optimal (Jones, D.T., 1999). The range of window sizes tested was therfore set between 3-20 with an interval of 2.

As seen in **Graph 1.** the score peaks at a window size of 17, however the differnce in score for window size 15 is minimal and the standard deviation slightly higher for 27. Therfore in agreement with previously mentioned litterature a window size of 15 was choosen for all models created.

### Position specific scoring matrix:

The fundamental priciple behind protein structure prediction is that the order of amino acids determines the protein structure (Anfinsen C.B, 1972). Similar sequences generally attain the same structure. Multiple sequence alignment profiles of homologous proteins are assumed to represent the structural alignment, revealing which residues usally have similar secondary structure (Jiang Q., 2017).

The bash script `PSSM_extractor.sh` was used to generate PSSM frequency profiles running PSI-BLAST on SwissProt. 3 iterations and an e-value of 0.01 was used. Swiss-Prot was chosen primarily in consideration of time, as it is a much smaller database than the UniRefs. SwissProt only has one entry per gene for each organism (UniProt, 2018). However, the reduced number of sequences also means fewer homologues and will result in PSSMs containing less information.

## Grid search optimization of regularization parameters

Based on an initial class report of SVM random forest and decision tree models with default parameters window size 15 and PSSM, SVM and random forest was picked for further optimization. Only one round grid search was run for the SVM as it is very time consuming. The parameters used were picked based on the optimized parameters used by Guo J et al. on their similar SVM model. As seen from the result in the table the change of C from 2-2.3 had no significant impact. However the gamma parameter gave a 6.99% increased improvement of the mean cross-validation score.
`C parameter` – determines the degree of misclassifying allowed. Generally a smaller dataset, as in this case requires higher values of C.
`gamma parameter`- determines the influence of a single training example. Low values mean far- reaching and high values meaning close (Scikit-learn, 2017).

**SVM (PSSM):**

| Mean test score | Parameter: C | Parameter: gamma | Rank test score |
|---|---|---|---|
| 0.6106 | 2 | 0.01 | 4 |
| 0.6810 | 2 | 0.05 | 2 |
| 0.6136 | 2.3 | 0.01 | 3 |
| 0.6835 | 2.3 | 0.05 | 1 |

**Table 1:** Grid search result for SVM model. The parmeter C: 2.3 and gamma: 0.05 was chosen for the final model

For the random forest models several grid searches were performed. The final search is summarized in **Table 2**. There was not much difference in the test score for the last grid search. Changing from balanced to unbalanced gave a 0.5% drop in the test score, which was a surprisingly small change. The real difference came with using PSSM vectors, giving an increase of 9.62% accuracy.

`Balance mode` - Has the similar effect as adjusting the C parameter for SVM, because helixes and coils are more frequent structures than sheets. It is necessary to give more weight to the minority class.
`min_samples_split` –The minimum number of samples required to split an internal node.
`n_estimators` – number of trees created (in the forest).

**Random Forest Classifier (non-PSSM):**

| Mean test score | Parameter: class_weight | Parameter: min_samples_split | Parameter: n_estimators | Rank test score |
|---|---|---|---|---|
| 0.6812 | balanced | 2 | 200 | 9 |
| 0.6831 | balanced | 2 | 300 | 7 |
| 0.6848 | balanced | 2 | 400 | 3 |
| 0.6841 | balanced | 3 | 200 | 5 |
| 0.6855 | balanced | 3 | 300 | 1 |
| 0.6848 | balanced | 3 | 400 | 4 |
| 0.6829 | balanced | 4 | 200 | 8 |
| 0.6836 | balanced | 4 | 300 | 6 |
| 0.6852 | balanced | 4 | 400 | 2 |
| | | | | |
| 0.6805 | unbalanced | 3 | 300 | 10 |
| 0.7817 | balanced | 3 | 300 | (1) PSSM |

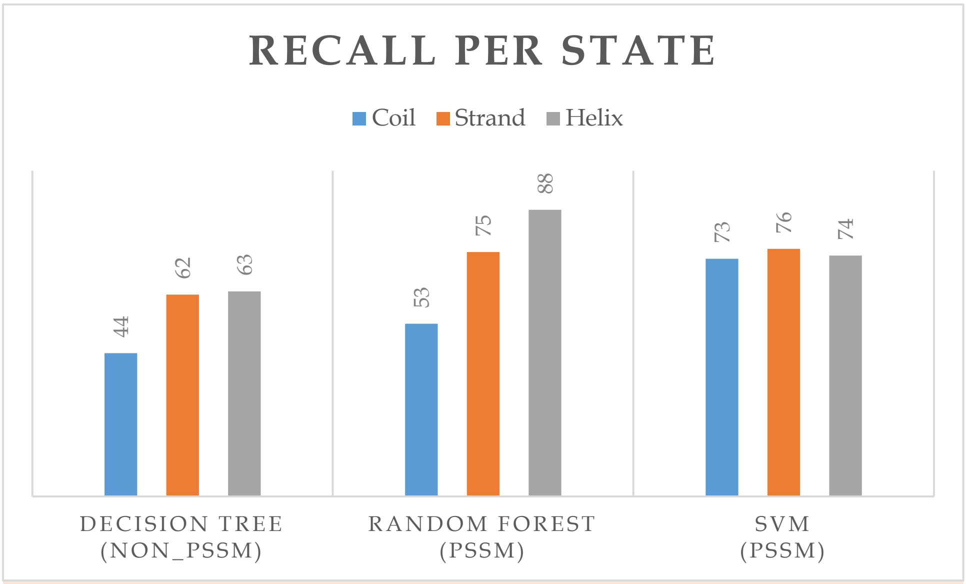
**Table 2:** Grid search result for random forest model. The parmeter C: 2.3 and gamma: 0.05 was chosen for the final model

Class report

The class report was performed on the two PSSM models optimized with the grid search. In addition a Decision Tree Classifier model was trained with PSSM and default parameters for comparision.

$$prcision = \frac{True\ positives}{True\ positives + False\ negatives}$$

$$recall = \frac{True\ positives}{False\ negatives}$$



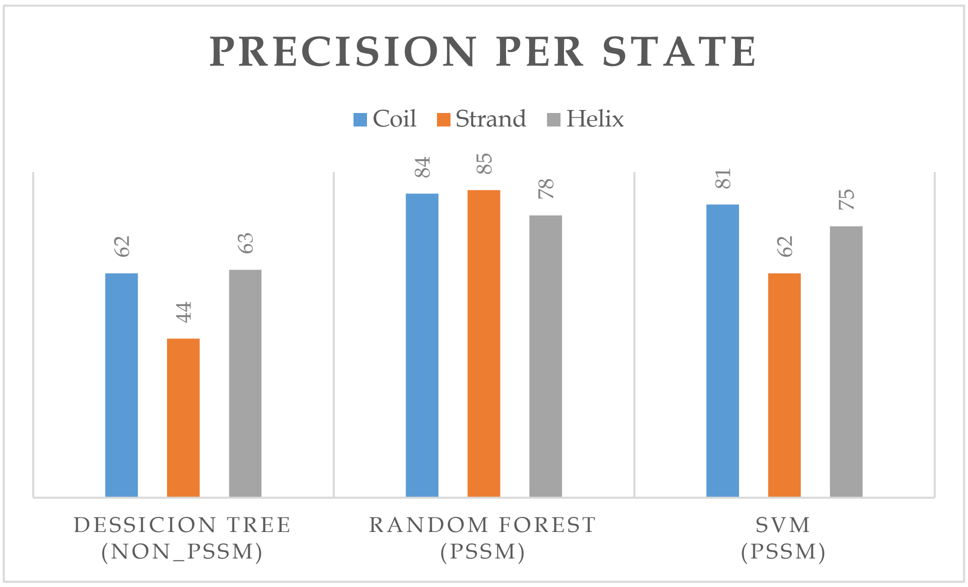
**Graph 2:** Recall is the percentage of each topology that was correctly predicted.

By looking at both **Graph 2** and **Graph 3** we can make several oberations:

- The decision tree model has the worst performance. This is unsurpricing as it was not optimized in any way. Decision trees also have a tendency of overfitting.

- On the other hand the random forest and the SVM is closer in performance. The random forest is superior on precisision. However this is at expence of the prediction of coils and sheets. Coils only has a recall of 53%. When looking at the confusion matrix for the random forest (**supplementary figures**), you see that as much as 39% of the sheets are beeing predicted as coils.

- The SVM model has an overall high recall score, but a lower precision especially for strands. It is over predicting coils , meaning that it is predicting too many strands.



**Graph 3:** Precision is how many of the predicted states that is correctly predicted.

Following models were trained:

1. Parameters: model = RandomForestClassifier(n_estimators=300, min_samples_split=3, class_weight='balanced')
   Location: `'../project/models/Random_balanced_win15_model.sav'`

2. Parameters: model = RandomForestClassifier(n_estimators=300, min_samples_split=3, class_weight='balanced')
   Location: `'../project/models/PSSM_random_balanced_win15_model.sav'`

3. Parameters: model = svm.SVC(C=2.3, decision_function_shape='ovr', gamma=0.05)
   Location: `'../project/models/PSSM_SVM_win15_model.sav'`

| **Predictor** | **Q3** | **Method** | **Database** |
|---|---|---|---|
| PSIPRED | 76.3% | PSSM + Neural networks | CASP3 |
| PHD | 72.0% | PSSM + Neural networks | - |
| PMSVM1 | 75.2% | PSSM + SVM | CB396 |
| **My predictors** | | | |
| SVM_PSSM | 74.4% | PSSM + SVM | train_data0.7.txt |
| RF_PSSM | 76.5% | PSSM + Random Forest | train_data0.7.txt |
| RF | 67.1% | Random Forest | train_data0.7.txt |

**Table 3:** Q3 scores for state of the art predictors, comparable in method for the predictors created.

## 50 novel proteins

50 proteins was taken from the cas3.fasta dataset. The E used to denominate strands was exhanged with  S to enable comparison of real topology and the predicted one.

The classification report gave surpricingly low scores compared to the reports from on the regular test set. This suggsest that the models are over fitting.

Classification report 50 novel proteins

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| H | 0.38 | 0.37 | 0.37 | 2808 |
| S | 0.34 | 0.29 | 0.31 | 2247 |
| C | 0.48 | 0.52 | 0.50 | 3960 |
| avg / total | 0.41 | 0.42 | 0.41 | 9015 |

## Conclusion

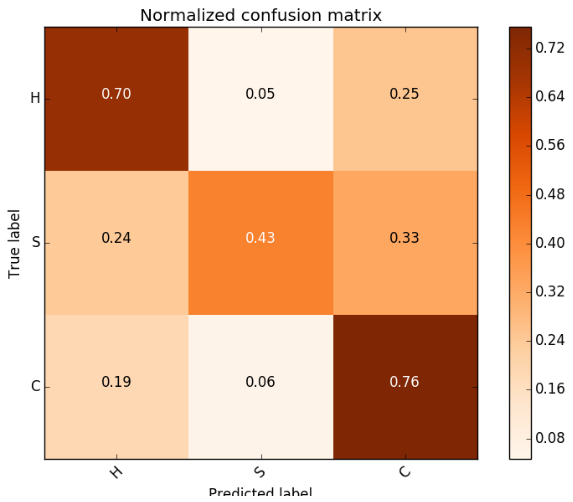
The accuracy scores for the two PSSM models created is within the range of state of the art predictors, of compareable models. As the classification report on the 50 novel proteins suggest, the high scores attained could be due to over fitting. To improve rssult more test sets could have been created.

The huge increase of 9.4% in Q3 score between the random forest with, and without PSSM, shows that use of evolutioary information is a powerfull tool.

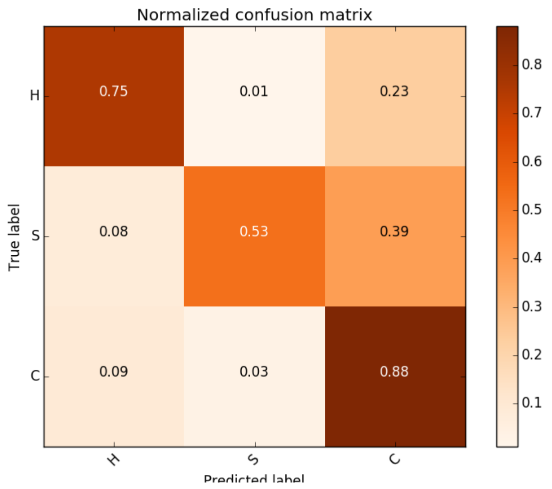
Despite the lower precission and Q3 score, the SVM model has a overall better performance. This can be more easely visulized by looking at the confusion matrix for the SVM (**supplementary figures**). Further deveopment of both the random forest and SVM would involve trying out different kernals functions. The choice of kernals has a high impact on the accuracy of the predictor as it  defines in which way to separet the classes. Further improvement can be achived by combining different models. Singel-method models often have some limitations. Combining models opens up for optimization by asigning tasks for different suitable models This way the unique characteristics of each of model can be taken advantage off (Jiang Q.,2017).

## Supplementary figures:

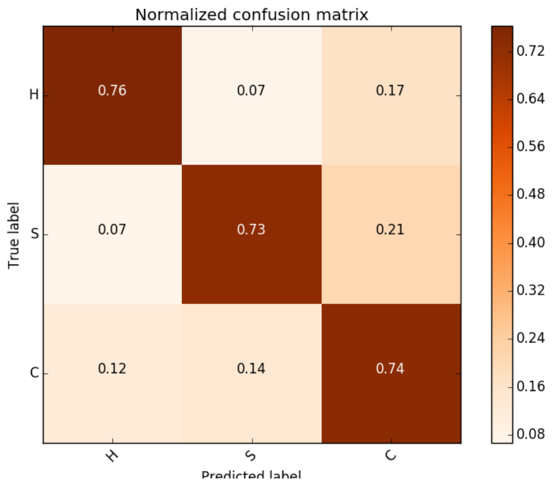
Random Forest



Random Forest (PSSM)



SVM (PSSM)



**References**

Anfinsen, C.B. (1972). The formation and stabilization of protein structure. *Biochemical Journal*, 128(4): 737-749. Available from doi: Available from doi: http://doi.org/10.1038/ [Accessed 18 March 2018]

Guo J., H. Chen, Z. Sun, and Y. Lin (2004) A Novel Method for Protein Secondary Structure Prediction. Using Dual-Layer SVM and Profiles. *PROTEINS: Structure, Function, and Bioinformatics,* volume (54), pp. 738-743. Available from doi: https://doi.org/10.1002/prot.10634 [Accessed 17 March 2018].

Jiang Q., X. Jin, S. Lee, Yao (2017) Protein secondary structure prediction: A survey of the state of the art. *Journal of Molecular Graphics and Modelling,* volume (76), pp. 379-402. Available from doi: https://doi.org/10.1016/j.jmgm.2017.07.015 [Accessed 18 March 2018].

Jones D.T. (1999) Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology,* volume 292(2), pp. 195-202. Available from doi: https://doi.org/10.1006/jmbi.1999.3091 [Accessed 18 March 2018].

Rost B. (1996) PHD: predicting one-dimensional protein structure by profile-based neural networks. *Methods in enzymology* volume (266), pp. 525-39. Available from doi: https://doi.org/10.1016/S0076-6879(96)66033-9 [Accessed 18 March 2018].

Brownlee J. (2016), "How Machine Learning Algorithms Work" [Internet], *Machine Learning Algorithms*. Available from: https://machinelearningmastery.com/how-machine-learning-algorithms-work/ [Accessed 16 March 2017]

Scikit-learn (2011) "Scikit-learn: Machine Learning in Python" [Internet], *Journal of Machine Learning Research* volume (12) pp. 2825-2830 Available from: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html [Accessed 17 March 2017]

Scikit-learn (2017) *RBF SVM parameters* [Internet] Available from: http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html [Accessed 17 March 2017]

UniProt (2018) "How redundant are the UniProt databases?" [Internet], *Live Science*. Available from: https://www.uniprot.org/help/redundancy [Accessed 16 March 2017]