Master Thesis

PROTEUS

A new predictor for protean segments

LINKÖPING UNIVERSITY

LITH-IFM-A-EX15/3118SE

Author:
Fredrik Söderquist

Examiner:
Björn Wallner

Department of Physics, Chemistry and Biology

September 10, 2015

Abstract

The discovery of intrinsically disordered proteins has led to a paradigm shift in protein science. Many disordered proteins have regions that can transform from a disordered state to an ordered. Those regions are called protean segments.

Many intrinsically disordered proteins are involved in diseases, including Alzheimer's disease, Parkinson's disease and Down's syndrome, which makes them prime targets for medical research. As protean segments often are the functional part of the proteins, it is of great importance to identify those regions.

This report presents Proteus, a new predictor for protean segments. The predictor uses Random Forest (a decision tree ensemble classifier) and is trained on features derived from amino acid sequence and conservation data.

Proteus compares favourably to state of the art predictors and performs better than the competition on all four metrics: precision, recall, F1 and MCC.

The report also looks at the differences between protean and non-protean regions and how they differ between the two datasets that were used to train the predictor.

Acknowledgement

I would like to thank Björn Wallner, Claudio Mirabello and Malin Larsson for their help and untiring support in this project. And Peter Söderquist for proofreading the report.

I also acknowledge with thanks the following software used is this project:

- DISOPRED
- PSI-BLAST
- BLASTClust
- PSIPRED
- scikit-learn
- IPython
- Seaborn

Contents

Terms & Abbreviations

Amino acid: The chemical substances found in plants and animals that combine to make proteins. There are 20 different amino acids. A protein is a single chain of amino acids (although there are proteins consisting of more than one chain).

Bias: Error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs.

Classifier: Machine learning algorithm that classify examples into discrete classes. Proteus is a classifier.

Cross-validation: A way to use all available as both train and test sets. Divides the whole dataset into folds and uses all folds as training and test sets in turn. Does not skew the result as a simple train/test split might do.

DISOPRED: Tool to predict disorder in proteins. Also predicts secondary structure and protein-protein interactions.

Example: A data point in machine learning jargon.

Extremely Randomized Trees: A decision tree based classifier. A more randomized version of Random Forest.

F1: Score metric. Calculated as $2 * \dfrac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} = \dfrac{2 * TP}{2 * TP + FN + FP}$

False negative (FN): A positive example incorrectly classified as negative.

False positive (FP): A negative example incorrectly classified as positive.

Features: The information packages a machine learning algorithm uses to make predictions.

Feature engineering: The process of designing features useful to the machine learning algorithm.

Intrinsically disordered protein (IDP): Proteins that lack a stable 3D structure.

Intrinsically disordered region (IDR): Region in protein that lack a stable 3D structure.

Label: The "correct answer" for a classification example. The classifier is evaluated by comparing predicted labels to true labels.

Machine learning: The construction and study of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from training examples in order to make data-driven predictions rather than following strictly static program instructions.

Matthews correlation coefficient (MCC): Score metric. Calculated as $\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$.

Over-fit: When a machine learning algorithm builds a complex model that fits the training examples almost perfectly, but cannot generalize to new data examples.

Precision: Score metric. Calculated as $\frac{TP}{TP + FP}$.

Predictor: A (trained) machine learning algorithm that can make predictions on new data examples.

Protean segment: A part of a protein that can undergo a disorder-to-order transition when it comes into contact with a partner molecule.

Protein: Large biomolecules consisting of one or more long chains of amino acid residues. Proteins differ from one another primarily in their sequence of amino acids, which is dictated by the DNA of their genes. Proteins usually fold into a specific three-dimensional structure that determines its activity. Intrinsically disordered proteins lack a stable 3D structure.

Proteus: The classifier presented in this report. It takes an amino acid sequence as input and predicts where (if any) protean segments can be found in the sequence.

Random Forest: A machine learning algorithm based on decision trees. Proteus uses a Random Forest algorithm.

Recall: Score metric. Calculated as $\frac{TP}{TP + FN}$.

Regressor: A machine learning algorithm where the output variable takes continuous values.

Residue: In biochemistry, a residue refers to a specific monomer within a protein or nucleic acid. In the case of proteins, residue refers to a specific amino acid.

Test set: The share of data examples that are used to evaluate the machine learning algorithm by predicting the labels for the test set and compare those to the true labels for the test set.

Training set: The share of data examples that are used to train a machine learning algorithm.

True negative (TN): A negative example correctly classified as negative.

True positive (TP): A positive example correctly classified as positive.

Under-fit: The model is too simple to fit even the training examples correctly. If the model under-fit the data, it will always produce bad results, no matter how large training set you use.

Variance: Error from sensitivity to small fluctuations in the training set. High variance can cause over-fitting.

Chapter 1

Project Introduction

1.1 Project Goal

The project's goal is to make a predictor for protean segments in intrinsically disordered proteins.

The predictor would take an amino acid sequence as input and return a prediction of where (if any) protean segments could be found in the sequence.

The predictor will be useful for medical researchers. If a scientist finds that a disordered protein contributes to a disease, he or she can use the predictor to find the functional part of the protein, and then go on to design a cure tailored to the specific region highlighted by the predictor.

1.2 Introduction to Proteins

To understand this project, you first need to understand proteins. Proteins are made up of a single chain of amino acids, like pearls on a string. There are 20 different amino acids, and each amino acid have different properties: some are small, others are large, some have a negative charge, others positive, and so on. These properties interact with each other, and will result in the protein folding itself. Think of it as a string of pearls where the pearls are magnetic; the string will fold up into a stable glob. The stable 3D structure of the protein is what gives its ability to function.

Or so protein scientists thought, until the discovery of intrinsically disordered proteins.[1]

1.2.1 Intrinsically Disordered Proteins

It seems that there are many proteins that lack a stable structure, yet still maintains a function.[2] Such proteins were sporadically discovered over a long period of time before scientists realized that they were part of a coherent group of proteins. Because of that, every scientist came up with their own name to describe disordered proteins (my favourite being *dancing proteins*[3]).

The established term is now *intrinsically disordered proteins*, or IDP for short. In some proteins, only parts of it are disordered. The disordered parts are called intrinsically disordered regions, or IDR for short. For the purposes of this report, intrinsically disordered proteins (where the whole protein is disordered) and intrinsically disordered regions (where only part of the protein is disordered) are both of interest, as long as *some* part of the protein is disordered.

Intrinsically disordered proteins stand in contrast to ordered proteins that fold into a pre-determined structure as soon as they are made, and stay that way until they are destroyed. Of course, this is an over-simplification. Even the most stable proteins have some inherent flexibility (or "wiggle-room"), and there is no clear-cut boundary when a protein ceases to be ordered and lapses into disorder.

The interesting property of disordered proteins (or disordered regions) is that while they lack a stable conformation on its own, *part* of the protein can become ordered when they come into contact with another molecule. The certain parts of intrinsically disordered proteins that can undergo a *disorder-to-order transition* and interact with other molecules are called *Molecular Recognition Features*, or MoRF for short. To complicate matters, they are sometimes known as *Protean Segments*, or ProS (protean means "shifting"). I will call these regions protean segments.

It seems that these intrinsically disordered proteins are more common than previously anticipated. They are especially common in eukaryote cells[4] (cells with a nucleus; not bacteria) and are involved in many different tasks, including signalling, control, regulation and recognition, that are incompatible with stable 3D-structures[5,6,7].

Intrinsically disordered proteins are also involved in different diseases such as Alzheimer's disease, Parkinson's disease and Down's syndrome.[8,9,10] Because the protean segment is the functional part of the disordered protein, it is important to identify that region. If you want to interact with a disordered protein, you need to tailor your molecule to the protean region.

This project aims to better identify the protean segments.

1.3 Process

The time plan for the project was set up at a time where I did not know the first thing about machine learning. And it shows. There are five weeks in the time table that only says "Machine learning", because at the time I had no idea what that process would entail. The time plan is presented in table 1.1.

Project week	Week	Task
1	11	Learn basics in Ubunty and Python
2	12	Parse XML-database
3	13	Visualize protean segments in proteins
4	14	Basic statistics for protean segments
5	15	Generate amino acid concentration features
6	16	Generate disorder prediction features
7	17	Generate secondary structure prediction features
8	18	Compile early results and run additional experiments
9	19	Half-time report
10	20	Learn basics in machine learning
11	21	Machine learning
12	22	Machine learning
13	23	Machine learning
14	24	Machine learning
Vacation	25	Celebrate midsummer!
15	26	Run additional experiments
16	27	Write final report
17	28	Write final report
18	29	Make changes to report after feedback
19	30	Present thesis and do opposition
20	31	Write reflection document

Table 1.1 – The time table as established at the first week of the project.

Chapter 2

Theory

I will introduce the subject of *machine learning* with something more tangible than protean segments in disordered proteins.

Let us instead say we are in the housing business and want to *predict* the selling price of houses.

2.1 Data

The first thing we need is data. We need information about past home sales in order to make predictions about the future. The (made up) data that we will use in this chapter is presented in table 2.1.

2.2 Features

The information a machine learning algorithm uses to make predictions is called *features*. We must extract information from the data above that we think will be useful when predicting the selling price, and feed that to the learning algorithm.

The process of designing good features is called *feature engineering*, and it is the most challenging part in machine learning.

House	Selling price (1,000 SEK)	House size	Number of bedrooms	House colour
1	500	50	1	Red
2	1,200	100	2	Yellow
3	1,600	135	2	Red
4	1,800	175	3	Red
5	1,950	250	4	Yellow

Table 2.1 – House data.

House	House size	Number of bedrooms	House colour
1	50	1	1
2	100	2	2
3	135	2	1
4	175	3	1
5	250	4	2

Table 2.2 – House features.

2.2.1 Feature Intuition

Say we want to predict whether or not a particular house will sell for more than 1,000,000 SEK. What information do we need in order to make an accurate prediction?

Where the house is situated would be essential. How big is it? How many bathrooms? How many bedrooms? Does it have a view or a garden? Those are all good things to know when trying to predict the selling price of a house and are examples of *good features* for this particular example.

You could also find out the colour of the house, or the name of the previous owner, but those things will probably *not* affect the selling price very much, and are therefore *bad features.*

In the made-up house data we only have information about the size, colour and number of bedrooms in the house. Those will be our features. Machine learning algorithms can generally only deal with numerical data (there are exceptions) so in order to use our data from table 2.1 we must "translate" the house colour column into numbers: 1 for "Red" and 2 for "Yellow". The features for our houses are presented in table 2.2.

Note that the selling price is *not* a feature. We are using features to *predict* the selling price.

2.2.2 Training & Test Sets

Before the machine learning algorithm can make a prediction, it must learn from experience. This is done by feeding the learning algorithm data. In machine learning jargon, a data point is called an *example.* In the housing illustration, every house is an example. Every example has a number of *features* and a *label.*

The features contain information about the example (the house). We have three features: House size, number of bedrooms and house colour.

The label contains the *"correct answer"* for the example. The label will be 1 if the house sold for more than 1,000,000 SEK and 0 if it did not. In our case, house 1 will have the label 0 (it did not sell for more than 1,000,000 SEK) while house 2, 3, 4 and 5 will have the label 1 (they did sell for more than 1,000,000 SEK).

In general, the more training examples we feed the learning algorithm, the more accurate it will become. But if we want to measure its performance correctly, we must set aside a part of our examples. The part we set aside is called a *test set*. The learning algorithm is not allowed to see the test set when learning.

Therefore we split our dataset into a training set and a test set. We then feed our learning algorithm the training set together with the labels (the "correct answers") for the training set. After the algorithm has learned from the training set we evaluate the algorithm's performance by feeding it the test set *without the labels*, instead letting it *predict* the labels. We then compare the *true labels* (that we held back) with the *predicted labels* for the test set. The better compliance between the true labels and the predicted labels, the better the learning algorithm.

2.2.3 Cross-Validation

In the train/test split, you cannot train on all examples (because you need to hold back a part to test on). But which data to hold back? The data you choose to hold back could skew the result.

An alternative way to measure the performance of your classifier is to use *cross-validation*. That means you divide your *whole* dataset into equally sized *folds*. I've used 5 folds, but you could use more or less.

When your whole dataset is divided into 5 folds (or parts), you hold the first fold back and train the algorithm on the remaining four. When done, you test the algorithm on the first fold that was held back. Then you hold back the second fold and train on fold 1, 3, 4 & 5, and test on fold 2. And so on, until you have tested on all 5 folds. The overall performance is the mean value of the five test scores.

Cross-validation gives a better indication of your performance than a regular train/test split, as you train and test on your whole dataset.

2.2.4 Clustering

When sorting the data into folds, we must make sure that very similar examples go in the same fold. For instance, two identical houses right next to each other would be two very similar examples.

If we do not do that, the classifier would be able to learn "by heart" from an example in one fold and use that information to correctly classify a very similar example from another fold. Of course we want our algorithm to learn from the test set, but we want it to *generalize to examples not seen before*, not just memorizing the properties for a given set of examples.

2.3 Classifier

2.3.1 Classification & Regression Problems

In the housing example above we wanted to predict whether or not a house would sell for more than 1,000,000 SEK. That is a *classification* problem. Either it will sell for more, or it won't. There are only two well-defined classes.

If we on the other hand wanted to predict the *selling price*, it would become a *regression* problem. The prediction would no longer be into discrete classes but rather a value along a continuous scale. This is called a *regression* problem.

Machine learning algorithms that predict classes are called *classifiers* while algorithms that predict a continuous value are called *regressors*.

The project's goal is to predict whether or not a residue (an amino acid in a protein) is part of a protean segment. It is therefore a *classification problem* and I will refer to the machine learning algorithms used in this project as *classifiers*.

2.3.2 Bias & Variance

When choosing classifier there is one trade-off in particular you need to think about: the *bias-variance dilemma*. A classifier will make a model from the data you feed it, and that model should ideally have both low bias and low variance.

To illustrate this dilemma, let us use the housing data, but as a *regression* problem. Now we want to predict the selling price (not only if it is over 1,000,000 SEK). In figure 2.1 the selling price is predicted by three different regressors using *only* the house size features from table 2.2.

Bias is an error that arises from erroneous assumptions by the model. It is the inability of the model to fit the data properly. In figure 2.1 the blue line is a model too simple to fit the data properly. That model has a *high bias*, and we say that the model *under-fits* the data. On the other hand, the model has *low variance*, because two similar examples will get similar predictions.

The red line in figure 2.1 fits the data points perfectly, so that model has very *low bias*. But the red line also show some strange behaviour between the last two data points (not to mention beyond the last data point!) that probably does not correspond to reality. A new example similar but not identical to an existing example could get a completely different prediction. We say that this model has *low bias* but *hight variance*. The high variance comes from the model *over-fitting* the data. A more complex model is capable of finding more complex patterns in the data, but it is also prone to over-fitting the data, resulting in poor generalisation to new examples.

The green line in figure 2.1 is a model complex enough to describe the data properly, but not so complex that the model over-fits the data. It symbolizes a good trade-off between bias and variance.

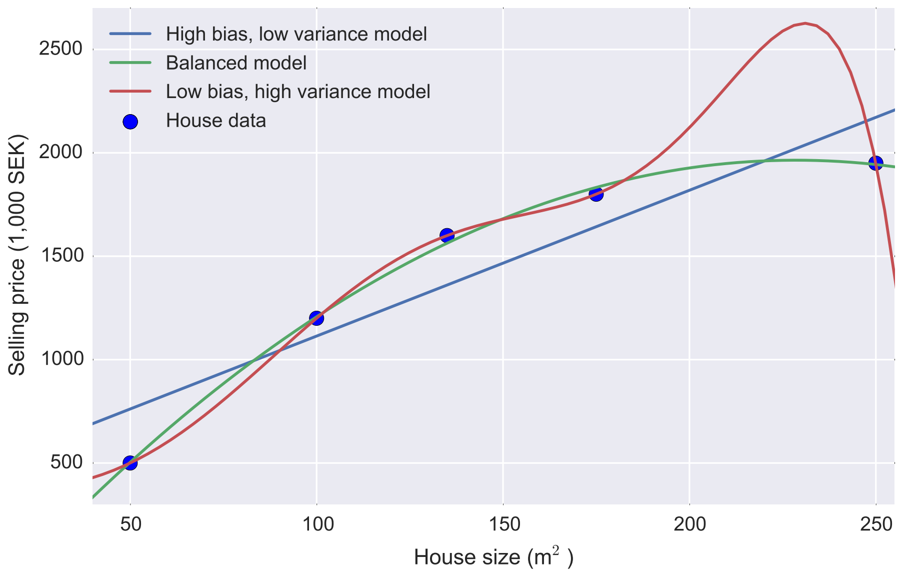

Figure 2.1 – An illustration of bias and variance.

2.3.3 Random Forest

One of the classifiers used in this project is *Random Forest*[11]. Random Forest is a decision tree ensemble classifier.

A ”normal” decision tree will calculate where to best split the data in every step. In Random Forest, the algorithm will randomly leave out features when considering where to make a split, resulting in slightly different trees every time. It is called an ensemble classifier because it makes a lot of decision trees, and a lot of trees is a forest. Every decision tree in the forest then get to vote for the classification of every example, and the example will be classified according to the majority vote.

2.3.4 Extremely Randomized Trees

A variant of the Random Forest classifier is *Extremely Randomized Trees*[12]. In essence, it is an even more randomized version of Random Forest. This makes the classifier more resistant to over-fitting the data (lower variance), but increases the bias slightly. The bias increases because too heavy randomization causes the classifier to miss features with low relevance. Extremely Randomized Trees are also slightly cheaper to compute than Random Forest.

2.3.5 Scoring Methods

In binary classification (two classes), there are only four possible results when classifying an example:

1. The examples was positive and correctly classified as positive, called a *true positive* (TP).
2. The example was negative and correctly classified as negative, called a *true negative* (TN).
3. The example was positive but incorrectly classified as negative, called a *false negative* (FN).
4. The example was negative but incorrectly classified as positive, called a *false positive* (FP).

These four measurements are used when calculating the score. But the score can be calculated in different ways.

Accuracy

The most common for classification problems is *accuracy*: how many percent of the examples did it classify correctly? That score is calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

For unbalanced datasets (the number of positive and negative examples are far from equal) the accuracy method is extremely bad.

In the ProS and MoRF datasets there are around 2% positive examples. That means a naive classifier that classifies *all* examples as negative would achieve 98% accuracy.

This report will instead use four different scoring methods: *Precision*, *Recall*, *F1* and *Matthews correlation coefficient* (MCC).

Precision

Precision measures how many of the examples classified as positive actually were positive. It is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.2}$$

Recall

Recall measures how many of the (true) positive examples we managed to classify correctly. It is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.3}$$

It is trivial to get 100% recall by classifying *all* examples as positive. Similarly, you can get 100% precision by only classifying the one example you are most sure of as positive (and the rest as negative). Precision and recall are therefore not very useful on their own. Enter the F1 score.

F1

The F1 score is a weighted average of the precision and recall scores. It is calculated as:

$$\text{F1} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} = \frac{2 * TP}{2 * TP + FN + FP} \tag{2.4}$$

Basically, the F1 score finds the best trade-off between precision and recall.

Matthews correlation coefficient

Matthews correlation coefficient (MCC) is another balanced measure which can be used regardless of class sizes. The MCC is a correlation coefficient value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 random prediction and -1 a perfect inverse prediction. The MCC is calculated as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.5}$$

Chapter 3

Method

3.1 Tools & Software

This project uses the machine learning package *scikit-learn*[13] together with IPython[14].

BLASTClust[15] is used to cluster the proteins. DISOPRED[16,17,18] is used to predict the disorder score. DISOPRED uses PSIPRED[19] to predict secondary structure and PSI-BLAST[20] to generate a position-specific scoring matrix.

The code written for this report can be viewed and accessed through http://nbviewer.ipython.org/github/Thrinduil/exjobb/blob/master/Proteus.ipynb

3.2 Data

The data used in this project comes from three different sources. A summary can be found in table 3.1.

Dataset	Proteins	Protean residues	Non-protean residues	Total residues
ProS	557	6,245	356,053	362,298
MoRF	840	10,549	494,264	504,813
ProS & MoRF	1,397	16,794	850,317	867,111
Validation	9	163	2046	2209

Table 3.1 – Summary of the two datasets used.

3.2.1 IDEAL

One database, called **IDEAL**[21,22] (short for *Intrinsically Disordered proteins with Extensive Annotations and Literature*), contains 557 proteins with experimentally verified protean segments. However, only 203 of 557 proteins in this database actually contain protean segments. The rest are intrinsically disordered proteins where no protean segments has yet been experimentally verified.

Proteins from this database are referred to as ProS proteins.

3.2.2 MoRF

The other database comes from MoRFpred[23], one of the existing classifiers Proteus try to outperform. This database contains 840 proteins, and all of them has at least one protean segment.

Proteins from this database are referred to as MoRF proteins.

3.2.3 Validation set

The third database is a small validation set of only 9 proteins. The authors of the Disopred3 paper[18] used this validation set to test their new classifier. The same validation set will be used to test Proteus (the classifier presented in this report), which will enable us to make a fair comparison between the methods.

This validation set is *only* used for testing, and only used at the very end. The validation set is not used while tuning the parameters of the classifier, nor is it ever used as training examples.

3.2.4 Data Sample

While the IDEAL database contains a lot of information about the proteins in it, the classifier only use the amino acid sequence and the protean segments labels.

Below is the first 40 residues (amino acid positions) for the protein 1E91_B (also known as Q05195 in the UniProt database). The first row is the one-letter code for the amino acids. M stands for Methionine, A for Alanine, and so on (see table 3.6 for all 1-letter shorthands). The numbers on the second row tells us where we have a protean segment. This particular protein happens to have a protean segment between residue 5 and 28.

```
MAAAVRMNIQMLLEAADYLERREREAEHGYASMLPYNNKD...
0000111111111111111111111111000000000000...
```

The classifier will use the amino acid sequence to calculate the features, and learn by comparing the predicted protean segments labels to the true labels provided by the datasets.

3.3 Data Clustering

As noted in the previous chapter, similar proteins must go *in the same fold.* BLASTClust[15] was used with the following command-line arguments: `-p T -S 30 -L .5 -b T`. To register a pairwise match two sequences will need to be 30% identical (`-S 30`) over an area covering 50% of the length (`-L .5`).

This resulted in 774 clusters, with the largest cluster containing 38 proteins, and 253 clusters containing more than one proteins. 28.2% of all ProS proteins share a cluster with at least 1 MoRF protein and 29.9% of all MoRF proteins share a cluster with at least 1 ProS protein.

The five folds were built while keeping all proteins in the same cluster together. As the proteins vary in length, the folds did not turn out equal in size. A summary of the folds is presented in table 3.2.

The sharp-eyed will notice there are only 1,396 proteins in the folds (and not 1,397 as indicated in table 3.1). One of the ProS proteins clustered with one protein from the validation set. That ProS protein was removed, so no protein in the ProS or MoRF dataset should be similar to the proteins in the validation set.

Fold	Proteins	Examples (residues)	Positive examples	Proportion of positive examples
1	279	218,870	3,100	1.42%
2	279	165,796	3,126	1.89%
3	279	158,651	3,455	2.18%
4	280	163,840	3,545	2.16%
5	279	159,450	3,558	2.23%

Table 3.2 – Summary of the 5 data folds.

3.4 Feature Engineering

The most important part in machine learning is to make *good features.* Unfortunately, this is also the hardest part.

A total of **342** features was engineered in this project. They are generated from 7 different "groups" of features. All feature grops will be explained in turn, but first we need to explain DISOPRED.

3.4.1 DISOPRED

DISOPRED stands for *diso*rder *pred*iction and is a tool that from an amino acid sequence predicts each residue's probability to be disordered.[16,17] Not only does DISOPRED predict disorder, it also predicts secondary structure, and in the latest version, protein-binding.[18]

DISOPRED generates a lot of information about the proteins in order to make predictions, and rather than re-inventing the wheel, some of the information from DISOPRED is used to calculate features for the Proteus classifier.

3.4.2 Feature Group 1: Position-Specific Scoring Matrix

DISOPRED uses PSI-BLAST[20] (Position-Specific Iterated BLAST) to generate a position based scoring matrix.

PSI-BLAST compares the submitted protein sequence against a database of proteins. PSI-BLAST finds proteins similar to the submitted protein and analyses how and where they differ.

Every residue in the submitted protein will be assigned a score for every possible substitution. As there are 20 different amino acids, every residue is assigned 20 scores. A high score indicates that the original amino acid and the substitute are commonly interchanged.

If we submit a 100 residues long protein, the resulting score matrix will have 100 rows (one for each residue in the submitted protein) and 20 columns (one for each amino acid). In table 3.3 the position scores are shown for protein 1E91_B.

The values in that matrix are used as features. Proteus use a "window" 15 residues long, so if it at the moment generate features for residue 10, it will use the values in row 3 to 17. In total **300 features**.

The first feature will correspond to the score of the residue 7 steps down from the current residue being changed to an Alanine. The twenty-first feature corresponds to the residue 6 steps down from the current residue being changed to an Alanine. And so on.

Note that the number of features generated in this group depends on the window size. The number of features returned is window size times 20. If Proteus used a window size of 17 instead of 15, it would generate 340 features instead of 300. Then the first feature will correspond to the score of the residue 8 steps down from the current residue being changed to an Alanine.

3.4.3 Feature Group 2: Amino Acid Concentration

This feature group analyse the concentration of amino acids in a 15 residues long window around the current residue.

Let us again use protein 1E91_B as example. The E in italic is the current residue, and the box is the current window.

MAAAVRMNIQML$\boxed{\texttt{LEAADYL}\mathit{E}\texttt{RREREAE}}$HGYASMLPYNNKD...

In this window Proteus calculates the amino acid concentration. So the first feature would correspond to the amount of Alanine. There are 3 Alanine in the window and $3/15 = 0.2 = 20\%$. The second features corresponds to the amount of Cysteine (0%; there are no C in the window).

Residue	A	C	D	E	F	G	H	I	K	L
1	-222	-275	-451	-337	-121	-409	-289	-1	-268	92
2	215	-270	147	368	-349	148	-169	-176	-81	-291
3	305	-215	-76	-99	-330	120	157	-281	-93	-285
4	229	-239	-96	-183	-333	42	-270	-141	-145	-230
5	-103	-249	-331	-187	-3	-373	-308	103	-147	-31
6	201	74	-256	-157	-218	101	-234	41	-114	12
7	-270	-376	-252	-289	-188	-1	-402	248	-247	-55
8	-248	-423	-225	-273	-512	-241	164	-288	-279	-518
9	-376	-381	-598	-563	-266	-628	-576	582	-521	350
10	77	-500	246	229	-506	-20	-60	-450	-202	-36

Table 3.3 – Position based scoring matrix for the first 10 residues in the protein 1E91_B. Note that only 10 amino acids are shown. The full matrix would contain 20 columns.

As there are 20 different amino acids, this gives us **20 features** in total. Table 3.4 show all non-zero features for residue 20 in protein 1E91_B (the residue shown above).

Residue	A	D	E	L	R	Y
20	0.200	0.067	0.333	0.133	0.200	0.067

Table 3.4 – Amino acid composition for the window around residue 20 in protein 1E91_B. Please note that the amino acids not present in the window are not shown in this table.

3.4.4 Feature Group 3: Order or Disorder Regions

This feature group deals with the disorder prediction from DISOPRED. The first feature in this group is the *disorder prediction score* from DISOPRED averaged over a 15 residues long window. The following features deal with the order or disorder *region*.

If the current residue is predicted to be *disordered* (a score > 0.5), Proteus will find the residues on either side of the current residue where the score dips below 0.5. The start and stop residues become two features (as their placement in comparison to the whole protein rather than their residue number). The final feature is the length of the disordered region (in number of residues it spans). Figure 3.1 shows the disorder region around residue 50 in protein 1E91_B.

If on the other hand the current residue would be predicted to be *ordered*, Proteus calculate the start, stop and length of the *ordered* region instead. The three start, stop and length features for disordered regions will be set to 0.

In total **7 features**, but 3 of them will always be 0 (depending on whether

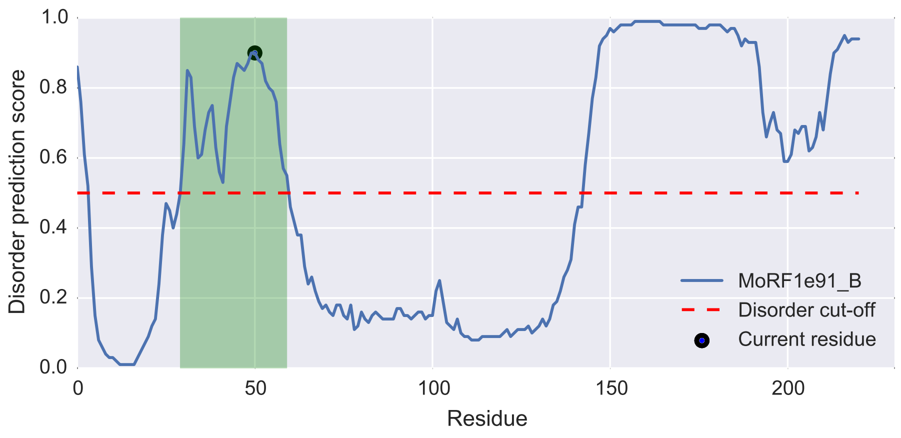

Figure 3.1 – The disordered region around residue 50 in protein 1E91_B. The length of the disordered region is highlighted in green.

the current residue is predicted to be ordered or disordered). Also note that all features except the score will be the same for residues that share the same ordered or disordered region.

3.4.5 Feature Group 4: Secondary Structure

DISOPRED predicts the likelihood of each residue being in one of the three main secondary structures: *helix* (3- and 4-turn alpha helices), *strand* (extended strands in parallel or anti-parallel β-sheet conformation and residues in isolated β-bridges) and *random coil* (residues which are not in any of the above conformations). The three probabilities are averaged over a 15 residues long window and become **3 features**. The probabilities for the first 5 residues in 1E91_B can be viewed in table 3.5.

Residue	Helix	Strand	Coil
1	0.010	0.001	0.992
2	0.334	0.046	0.698
3	0.639	0.050	0.379
4	0.446	0.091	0.452
5	0.240	0.123	0.617

Table 3.5 – Secondary structure prediction for the first 5 residues in protein 1E91_B.

3.4.6 Feature Group 5: Amino Acid Properties

Different amino acids have different properties. This group of features try to capture that in a more explicit way than the amino acid concentration features.

The properties we look at are polarity[24] (polar, nonpolar, acidic polar or basic polar), charge[24] (neutral, positive or negative), hydrophobicity[25] and molecular weight. In total **9 features**.

The new features are calculated much like the amino acid concentration by summing over the current 15 residues window. For instance, if we have 3 positively charged amino acids in the current window, the positive feature will be = 3.

All amino acids' properties can be found in table 3.6.

Amino acid	1-letter shorthand	Side-chain polarity	Side-chain charge	Hydropathy index	Molecular weight
Alanine	A	nonpolar	neutral	1.8	89
Arginine	R	basic polar	positive	−4.5	174
Asparagine	N	polar	neutral	−3.5	132
Aspartic acid	D	acidic polar	negative	−3.5	133
Cysteine	C	nonpolar	neutral	2.5	121
Glutamic acid	E	acidic polar	negative	−3.5	147
Glutamine	Q	polar	neutral	−3.5	146
Glycine	G	nonpolar	neutral	−0.4	75
Histidine	H	basic polar	positive (10%)	−3.2	155
Isoleucine	I	nonpolar	neutral	4.5	131
Leucine	L	nonpolar	neutral	3.8	131
Lysine	K	basic polar	positive	−3.9	146
Methionine	M	nonpolar	neutral	1.9	149
Phenylalanine	F	nonpolar	neutral	2.8	165
Proline	P	nonpolar	neutral	−1.6	115
Serine	S	polar	neutral	−0.8	105
Threonine	T	polar	neutral	−0.7	119
Tryptophan	W	nonpolar	neutral	−0.9	204
Tyrosine	Y	polar	neutral	−1.3	181
Valine	V	nonpolar	neutral	4.2	117

Table 3.6 – Amino acid properties.

3.4.7 Feature Group 6: Topography

If you plot the disorder prediction score for every residue you get an unruly graph with peaks and valleys (see figure 3.1). This feature group captures if the current residue is part of a peak or valley (or neither).

A residue is part of a peak if on both sides there exists another residue with a score at least 10 percentage points *lower* than the current residue.

Likewise, a residue is part of a valley if there are residues with disorder scores at least 10 percentage points *higher* than the current residue.

This is a very crude way to find peaks and valleys, but better than nothing.

In addition to the peak/valley feature (1 for peak, -1 for valley and 0 for neither), Proteus also calculate the *length* of the peak or valley the current residue is part of. In total **2 features**.

3.4.8 Feature Group 7: Conservation

This is not a feature group, as there is only **1 feature**.

This feature describes how well preserved (in an evolutionary sense) the current amino acid is. A well-preserved amino acid indicates that it is important for the function of the protein. As protean regions presumably are very important for the function of the protein, we expect the residues within the protean regions to be more preserved than other residues.

The conservation score is calculated by PSI-BLAST[20] by comparing the protein to similar versions in a protein database. The conservation features are then calculated by averaging over a 15 residues long window.

Chapter 4

Feature Inspection

A rule of thumb in machine learning is that if a human expert can look at the features and classify the examples correctly, a machine should be able to learn to do so as well.

Let us take a closer look at some of the features Proteus designs and see if we as humans can find any patterns in the data.

4.1 Amino Acid Concentration

The percentage of each amino acid in both protean and non-protean regions was calculated for all proteins. The result for protean regions was then subtracted from the result for non-protean regions. The result can be seen in figure 4.1.

The most significant finding is that protean regions contain much less Arginine (R) than non-protean regions. The same is true for Lysine (K).

For some amino acids, the concentration is very different between the two datasets. Look for instance at Leucine (L). Protean regions in MoRF proteins contain much less Leucine than non-protean regions. But in the ProS dataset, there is almost no difference!

Conflicting pattern between the two datasets can be seen in many other amino acids as well, such as Alanine (A), Glutamic Acid (E), Phenylalanine (F) and Tyrosine (Y).

The same amino acid concentrations was calculated, but this time the data was split over ordered and disordered regions instead of protean and non-protean. The result can be seen in figure 4.2.

Disordered regions are rich in Serine and Proline and contain less amino acids with hydrophobic side chains.

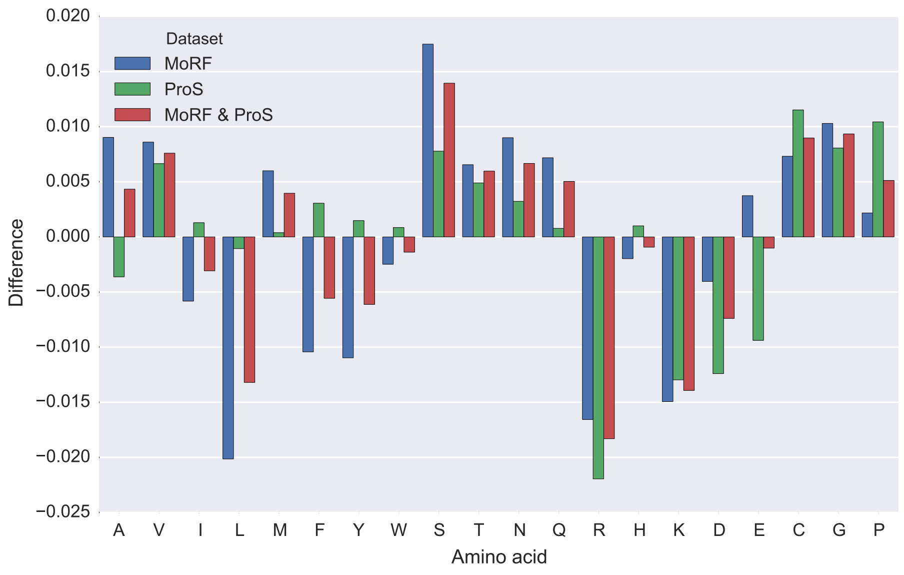

Figure 4.1 – The difference in amino acid concentration between protean and non-protean regions. A positive score means that the amino acid is more common in non-protean regions. The X-axis is sorted by amino acid properties. A, V, I, L, M, F, Y and W are hydrophobic. S, T, N and Q are polar but uncharged. R, H and K are positively charged, D and E are negatively charged. C, G and P are special cases.

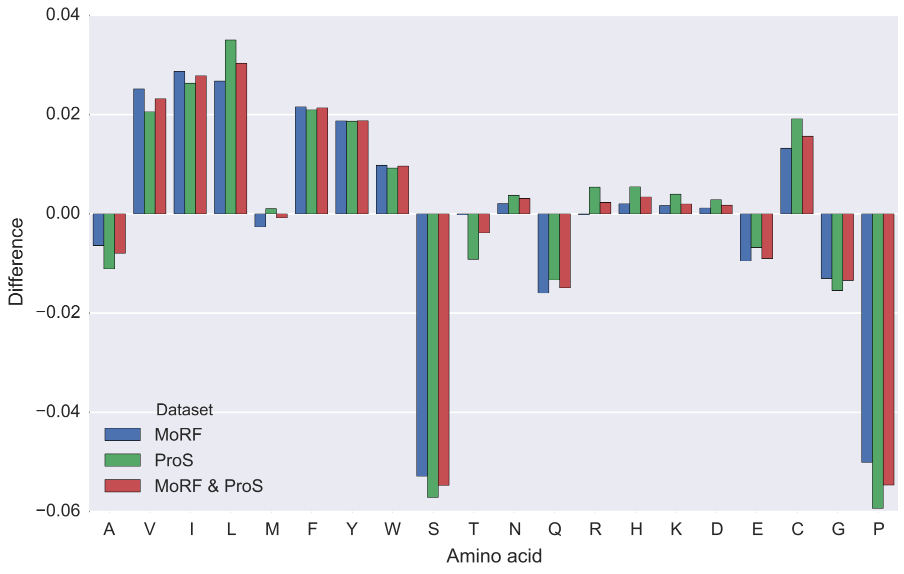

Figure 4.2 – The difference in amino acid concentration between ordered and disordered regions. A positive score means that the amino acid is more common in ordered regions. The X-axis is sorted by amino acid properties. A, V, I, L, M, F, Y and W are hydrophobic. S, T, N and Q are polar but uncharged. R, H and K are positively charged, D and E are negatively charged. C, G and P are special cases.

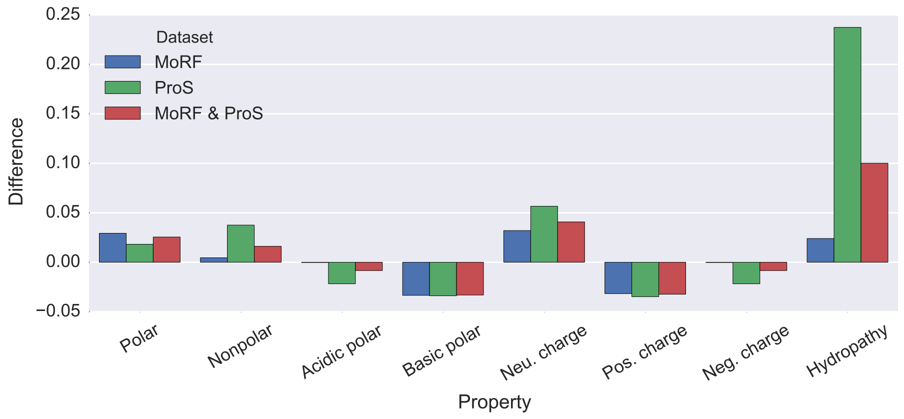

Figure 4.3 – The difference in amino acid properties between protean and non-protean regions. A positive score means that the amino acid property is more common in non-protean regions.

4.2 Amino Acid Properties

The amino acid concentrations were re-used to calculate the amino acid properties. As can be seen in figure 4.3, the most significant different is the hydropathy.

Hydropathy is a measurement of the amino acid's side chain's hydrophobicity.[25] The higher the number the more hydrophobic (”water-hating”) is the side chain. A negative number means the side chain is hydrophilic (”water-loving”).

Again the pattern is conflicting between the datasets. While non-protean regions in the ProS dataset are much more hydrophobic than protean regions, non-protean regions in the MoRF dataset are only slightly more hydrophobic.

4.3 Conservation

A residue in a protean region has on average a 22% higher conservation score than a residue in a non-protean region. Interestingly tough, the difference is much larger for MoRF proteins compared to ProS proteins, 27% compared to 10%.

4.4 Disorder Prediction

As protean segments exists in intrinsically disordered proteins and only become ordered in the proximity of a partner molecule, we would expect protean segments to have a high predicted disorder score. Figure 4.5 show the average predicted disorder score (predicted by DISOPRED) for protean and non-protean regions.

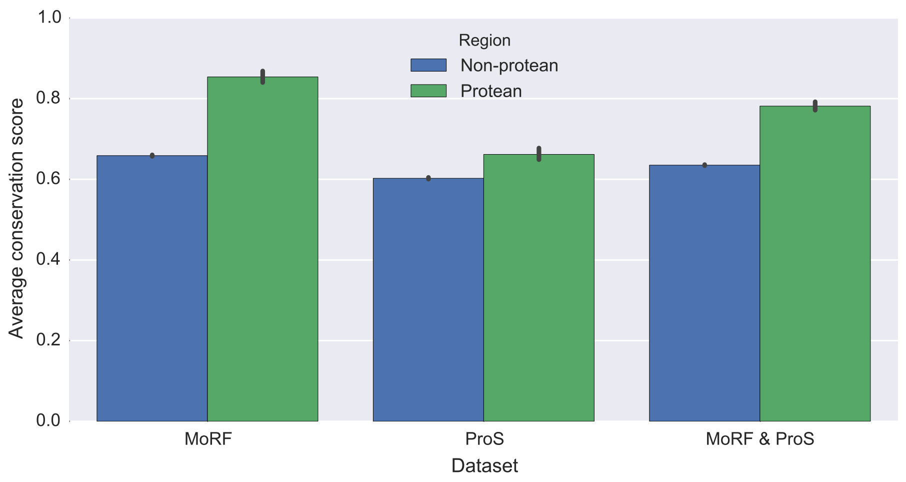

Figure 4.4 – The average conservation score for residues in protean and non-protean regions.

Turns out that protean regions in proteins from the ProS database have on average a higher predicted disorder score, while the reverse is true for proteins from the MoRF database!

4.4.1 Location of Protean Regions

If protean regions are not always predicted to be disordered, where then in the *"disorder landscape"* do we find the protean regions? The disorder prediction was plotted with protean regions highlighted in green to find out (see figures 4.6, 4.7 and 4.8).

The protean regions are found both in regions predicted to be ordered and regions predicted to be disordered. They are often found "valleys" between disordered regions, or in transitions from ordered to disordered regions. Three examples are shown in figures 4.6, 4.7 and 4.8, but they are in no way an extensive representation of all possibilities.

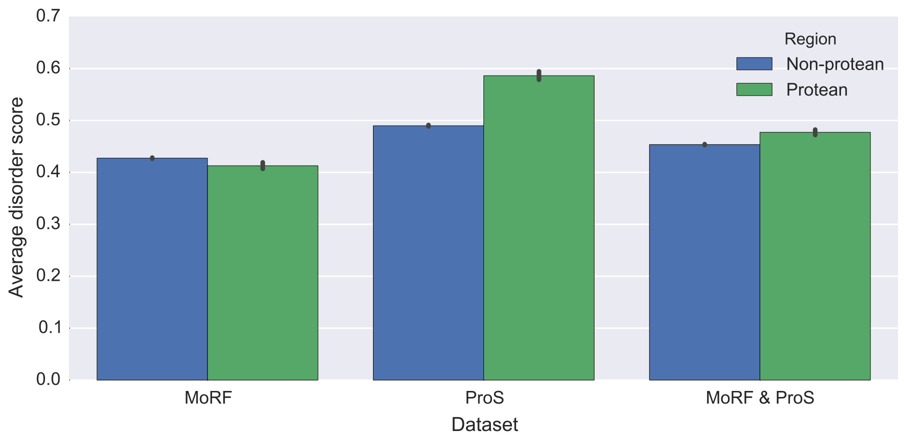

Figure 4.5 – The average disorder prediction score for residues in protean and non-protean regions.

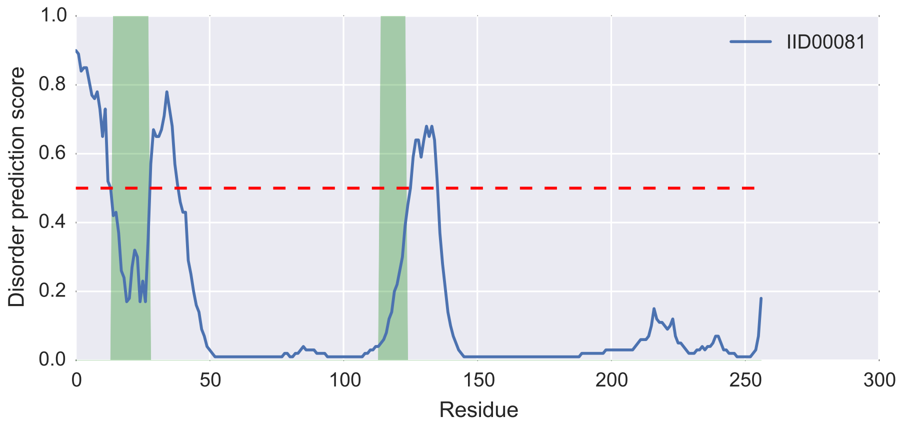

Figure 4.6 – The disorder prediction score for protein IID00081. The protean regions are highlighted in green.

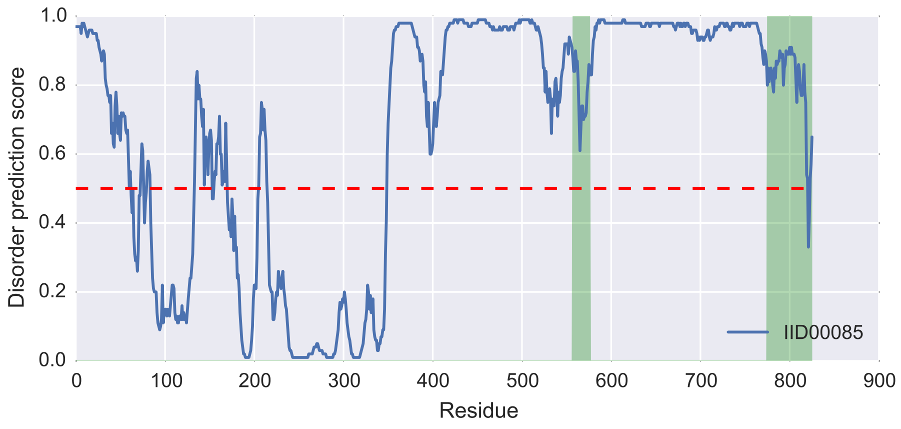

Figure 4.7 – The disorder prediction score for protein IID00085. The protean regions are highlighted in green.

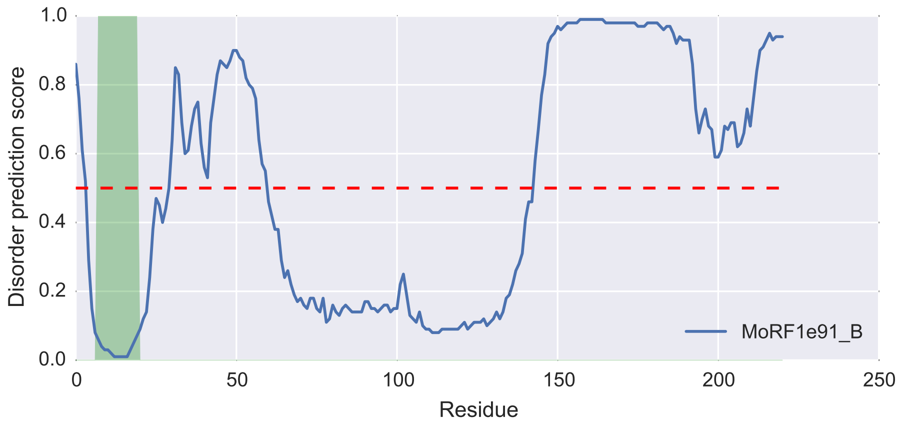

Figure 4.8 – The disorder prediction score for protein 1E91_B. The protean region is highlighted in green.

Chapter 5

Results & Validation

5.1 Tuning the Parameters

Machine learning methods has a lot of knobs, and success often comes from turning those knobs to fit the problem at hand.

5.1.1 Decision Tree Depth

The (arguably) most important parameter is the decision tree depth. The deeper the tree, the more complex patterns it can fit (lowering the bias). This can easily lead to over-fitting (increasing the variance). Finding a good depth is essential.

The number of leafs on a decision tree increases exponentially with the depth. A 3 levels deep tree has $2^3 = 8$ leafs. A 30 levels deep tree has $2^{30} = 1.07 * 10^9$ leafs. The ProS and MoRF datasets contains approximately 850,000 examples. That is roughly equivalent to 2^{20}.

The maximum depth was varied between 1 and 25. The experiment used both the Random Forest and the Extremely Randomized Trees classifiers. The result can be seen in figures 5.1 and 5.2.

The first thing to note is that Random Forest gives us slightly better results than Extremely Randomized Trees (easiest to see if we look at the yellow line).

The second thing to note is the sharp fall in recall. In figure 5.1, the recall falls by almost 50 percentage points as the depths increases from 5 to 15 levels.

A depth of 13 (8192 possible leafs) yields the highest MCC and F1 scores. From now on all experiments will only use the Random Forest classifier with a maximum decision tree depth of 13.

5.1.2 How Many Trees Make a Forest?

Another important parameter is how many decision trees to use. In theory, the more trees the better, but you will get diminishing returns after a critical number of trees. The sole reason to limit the number of trees is to save computation time.

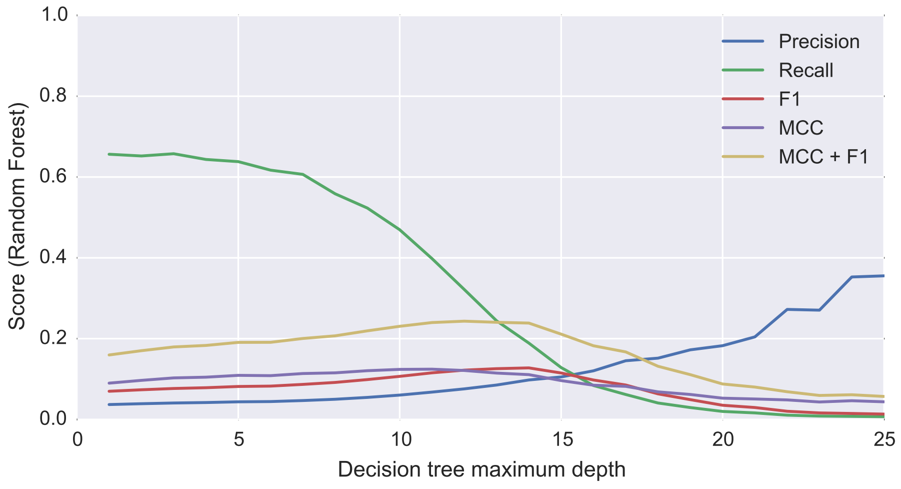

Figure 5.1 – The scores as a function of maximum tree depth for the Random Forest classifier.

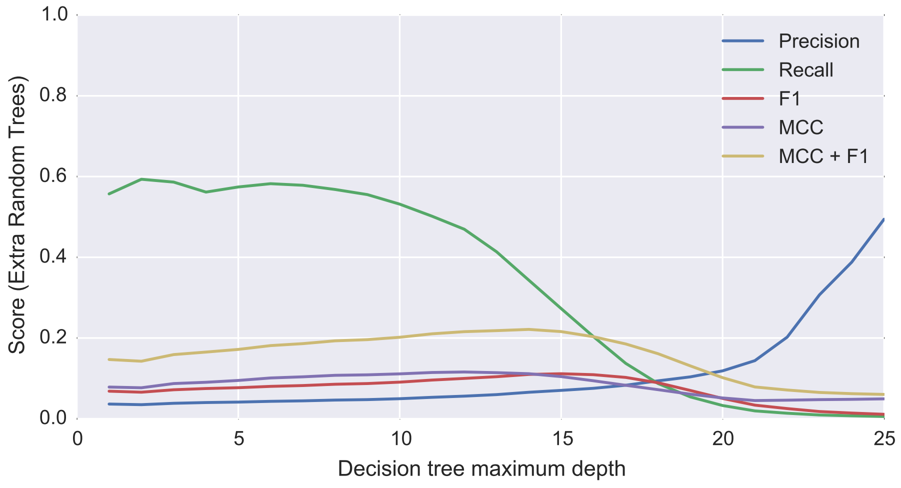

Figure 5.2 – The scores as a function of maximum tree depth for the Extremely Randomized Trees classifier.

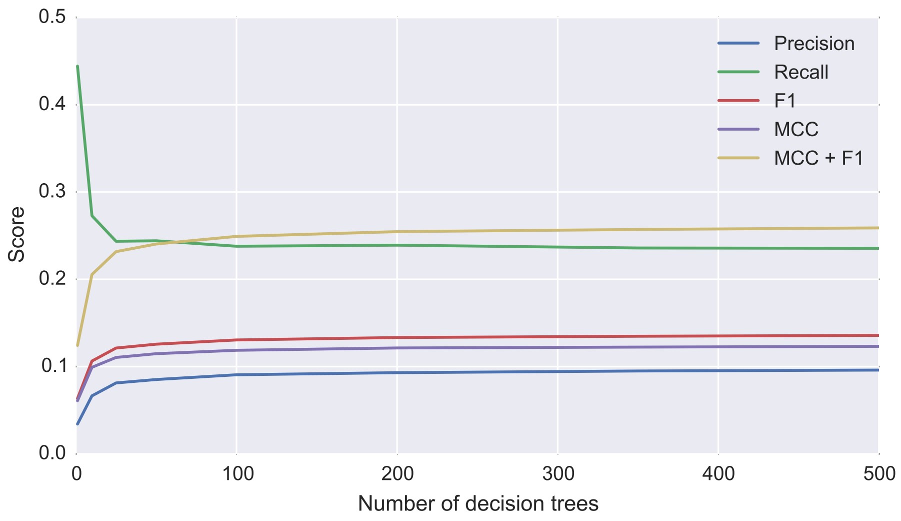

Figure 5.3 – The scores as a function of the number of decision trees.

In the optimal depth experiment above 50 trees were used for both Random Forest and Extremely Randomized Trees. As we can see in figure 5.3, 50 decision trees yields a reasonable score, but increasing the number of trees to 300 push up the score by 5%.

All experiments from now on will use 50 trees, except for the final validation, where 500 trees will be used to push Proteus to the best of its capabilities.

5.1.3 Probability Cut-Off

In the internal workings of the classifier, each example is assigned a probability to be positive. All examples with a probability over 50% is then classified as positive. But other values than 50% could be used as the cut-off.

The cut-off was varied between 0% and 100% to see what would result in the best scores. The result is presented in figure 5.4.

As we can see, 50% seems to be very close to the optimal cut-off for both the MCC and the F1 scores. Proteus will keep the cut-off value at 50%.

5.1.4 Window Size

Most of Proteus' features have been calculated using a "sliding window". That window size can be changed. So far Proteus have used a 15 residues long window, but that might not be optimal.

Note that the for all feature groups except one, the *number of features* will not change with a varying window size, only the *number of residues* that are used to calculate the features. The exception is the first feature group, the Position-Specific Scoring Matrix. There, the number of features equals the window size

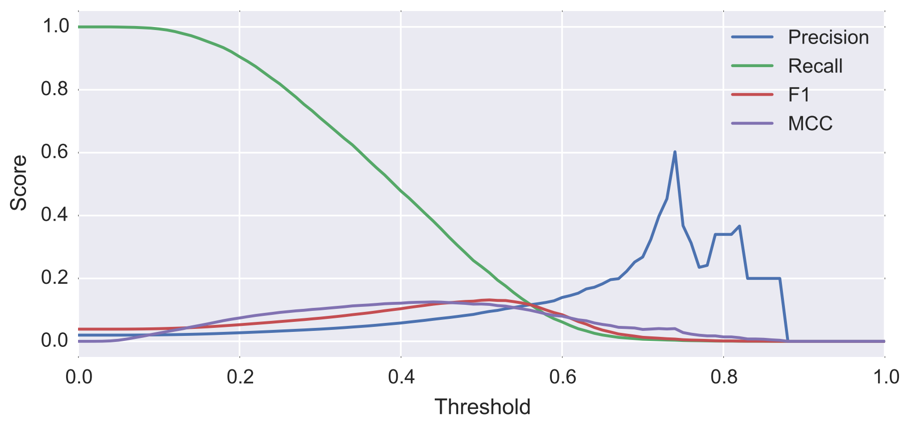

Figure 5.4 – The scores as a function of the probability cut-off.

times 20. So if we extend the window one residue on either side, we get 40 new features.

As shown in figure 5.5, the window size does not impact the score noticeably. Proteus will therefore keep using a 15 residues window.

5.2 Learning Curve

A learning curve can be useful when figuring out what causes miss-classifications: high bias or high variance. The learning curve is made by training and testing the classifier on a subsequently larger subsets of the whole dataset.

You measure both the training score (when you train and test on the same data) and the test score (with the usual cross validation). You then plot the train and test scores as a function of the amount of data used, and from the shape of the curves you can see if you are suffering from high bias, high variance or a bit of both.

As we can see in figure 5.6, gap between the training and test scores indicate that we have a problem with high variance. The training score is decreasing even as we use the full dataset. If we had even more examples, the training score would decrease and the test score would increase until the two curves met. Then we would not have any variance in our model. But looking at the slope of the test scores, getting more training examples would only get us a very slight increase in the test score.

The decline of the training scores as we increase the training set size is due to bias. The model cannot fit the data. Remember that the tree depth is limited to 13 levels. That gives us (at most) 8192 leafs. If the whole dataset would be fitted to that tree, every leaf would on average contain 100 examples.

When we use a small portion of our dataset, the 8192 leafs are enough to differentiate the data. But as the dataset increases, the number of leafs are not

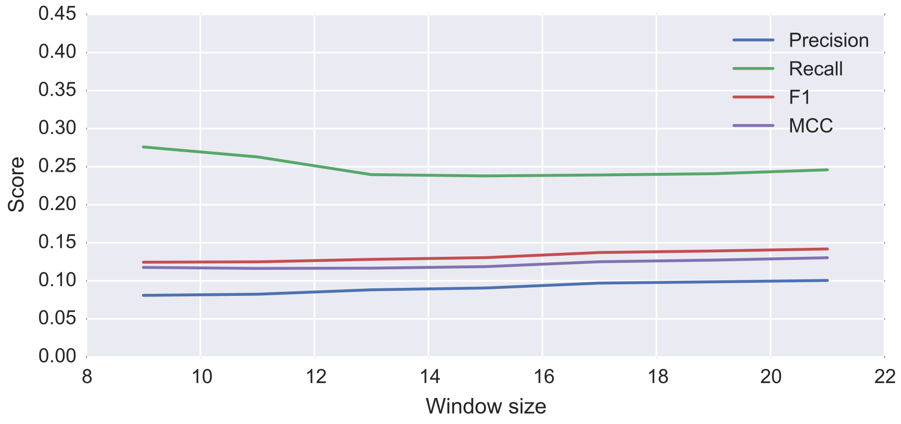

Figure 5.5 – The scores as a function of window size.

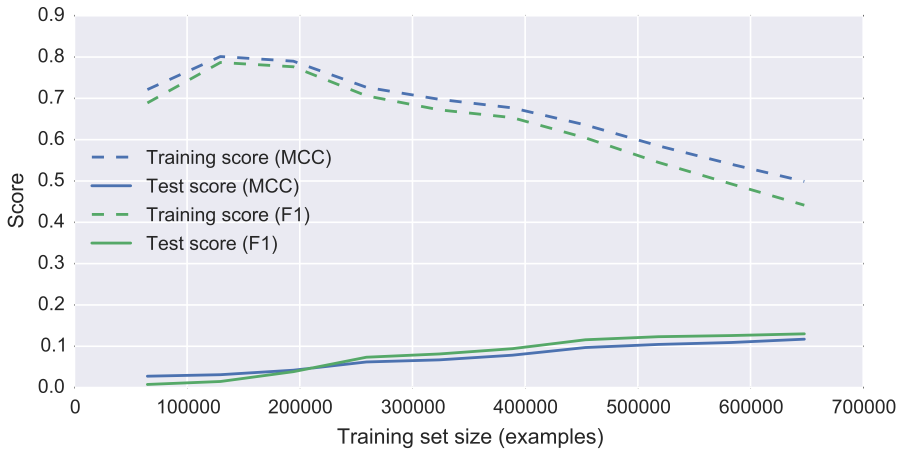

Figure 5.6 – Learning curve. Training and test score as a function of the training set size.

enough for the model to fit the data completely. Or rather, the data pattern is not simple enough to fit unambiguously to the tree.

Remember that we investigated the optimal decision tree depth. A deeper tree would lower the bias, but at the cost of higher variance, and the decrease in bias was not enough to offset the increased variance.

Another way to limit the variance is to use a smaller set of features.

5.3 Feature Group Selection

Now that we have engineered **342 features** (we are still using a window size of 15), it is not certain that all features will be useful. Some features might be uninformative, other might be redundant.

In fact, we know that some features are redundant. All features are calculated directly from the amino acid sequence. But sometimes the classifier do better with explicit rather than implicit features, so redundant features are not always a bad idea.

Uninformative features on the other hand add no information that helps the classifier. At best, those features are just ignored. But uninformative features can confuse the classifier and lead to worse results than if they were not included at all.

All possible combination of the *feature groups* (not all possible combinations of individual features) was examined to find the best combinations. As there are 7 feature groups, there are 127 possible combinations of those feature groups (that contain at least one group). The feature groups were evaluated using both MCC and F1 scores. Figure 5.7 shows the 20 best feature group combinations according to the MCC score and figure 5.8 shows the 20 best feature group combinations according to the F1 score.

There seem to be a very small difference between the top feature group combinations, and the top candidates are not the same for the MCC and the F1 scores. Using all features result in a good score in both evaluations, so Proteus will keep using all 7 feature groups.

5.4 Feature Importance

In the internal workings of the decision tree classifier it calculates which features contributes the most important information. Because there is a random element in our classifier all trees may not present the same feature importance, but the feature importance shown in figure 5.9 is averaged over 500 decision trees. As we can see, there are three features that stand out above the rest.

Feature 341 is by a wide margin the most important feature. Feature 341 describes the length of the topography region the current residue resides in (see section 3.4.8). That feature is part of feature group 6, a group that is completely discarded in the third most optimal feature group combination for the F1 score (see figure 5.8)!

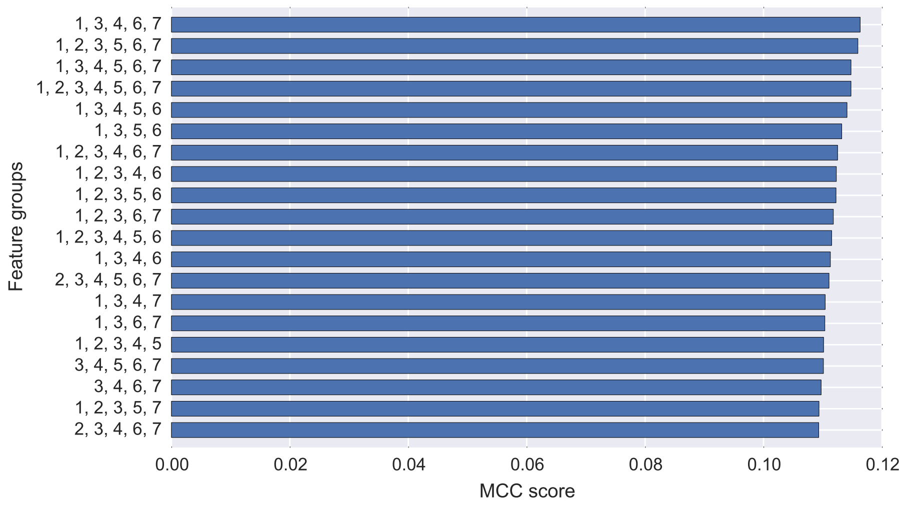

Figure 5.7 – The 20 best combinations of feature groups as measured by the MCC score. The numbers refer to the feature groups presented in section 3.4: 1: Position-specific scoring matrix; 2: Amino acid concentration; 3: Order or disorder region; 4: Secondary structure; 5: Amino acid properties; 6: Topography; 7: Conservation.

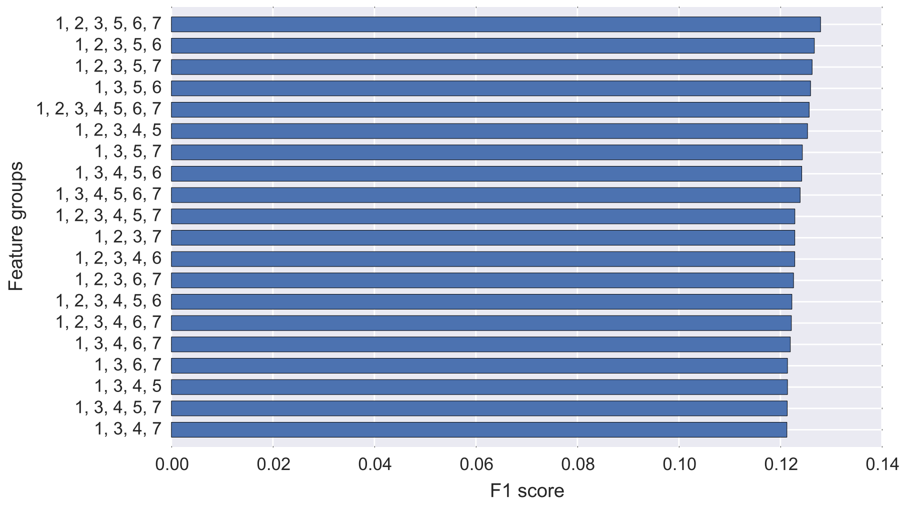

Figure 5.8 – The 20 best combinations of feature groups as measured by the F1 score. The numbers refer to the feature groups presented in section 3.4: 1: Position-specific scoring matrix; 2: Amino acid concentration; 3: Order or disorder region; 4: Secondary structure; 5: Amino acid properties; 6: Topography; 7: Conservation.

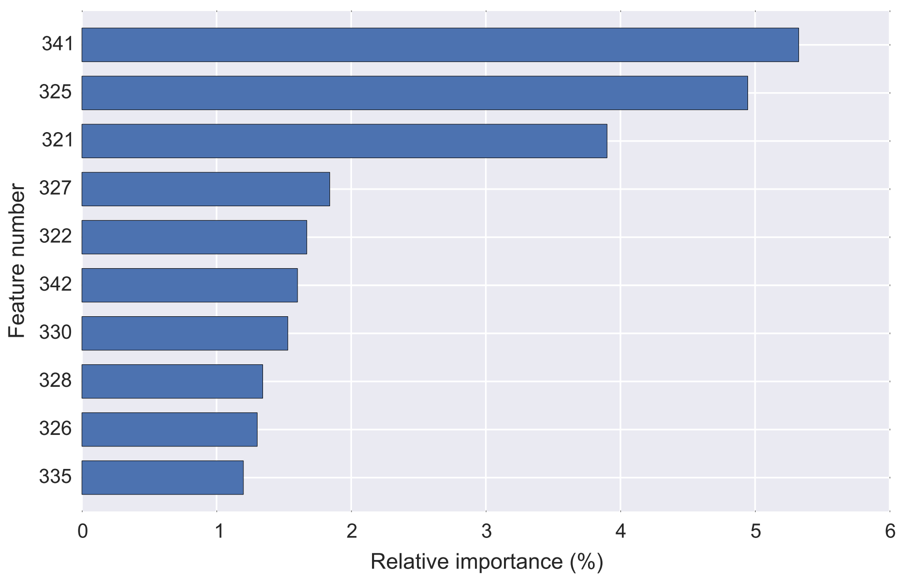

Figure 5.9 – Relative feature importance. Feature 341 describes the length of the topography region, feature 325 describes the length of the ordered region and feature 321 is the predicted disorder score.

The second most important feature (feature 325) is the length of the ordered region the current residue resides in. Note that this feature will be 0 for all residues predicted to be disordered!

The third most important feature (feature 321) is the predicted disorder score averaged over the current window.

5.5 Receiver Operating Characteristic

A *receiver operating characteristic curve* (ROC curve) shows the true positive rate as a function of the false positive rate.

The true and false positive rates are calculated from four metrics: true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). True positives are positive examples correctly classified as such. False positives are negative examples wrongly classified as positive. True negatives are negative examples correctly classified as such, and false negatives are positive examples wrongly classified as negatives.

The true positive rate (also known as *sensitivity* or *recall*, see section 2.3.5) is a measurement of the proportion of all positive examples correctly identified. It is calculated as $TP/(TP + FN)$.

The false positive rate (also known as *fall-out* or *false alarm rate*) is a measurement of the proportion of all negative examples incorrectly identified as

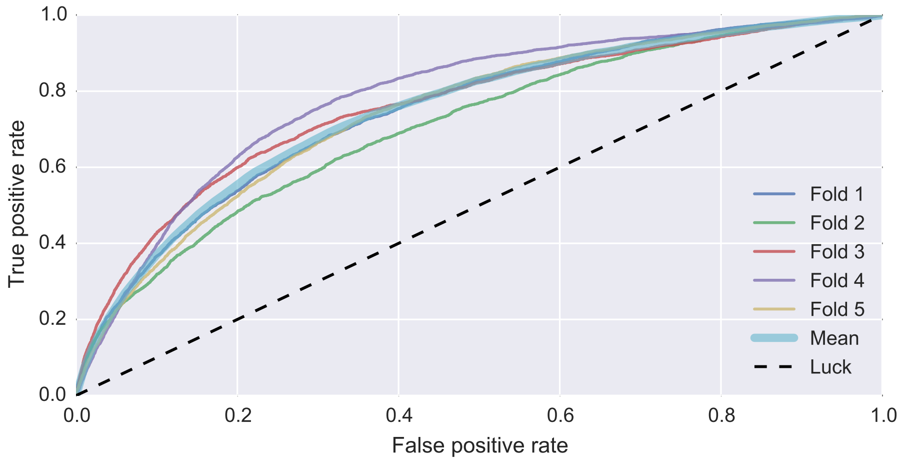

Figure 5.10 – Receiver operating characteristic curve.

positive. It is calculated as $FP/(FP + TN)$.

In the internal workings of the classifier every example is assigned a probability to be positive. In the ”normal” case, all examples with probabilities above 50% would be classified as positive. But maybe 50% is not the optimal cut-off. Maybe it is more important for us to find all true positive examples, and it does not matter that we will also identify a lot of false positive examples in the process. Then we would set our cut-off lower than 50%.

A ROC curve illustrates the correlation between correctly identified positive examples and the number of false alarms as the cut-off is varied. The closer the curve is the top left corner, the better.

As can be seen in figure 5.10, our curves are somewhere halfway between the straight line (the straight black line indicates the result a completely random classifier would get) and the top left corner. We do better than chance, but there is still room for improvement.

There is also a noticeable difference between the folds. This indicates it was a good choice to use cross-validation instead of a simple train/test split. Our result would have varied depending on which part of the data we used as a test set.

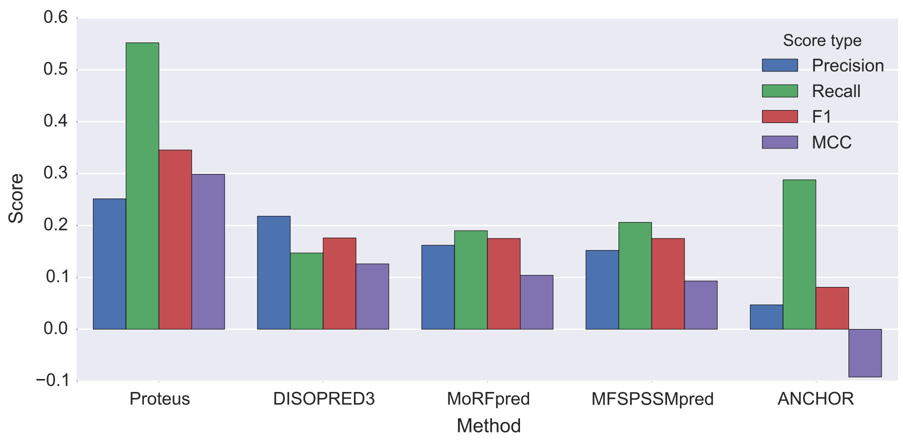

Figure 5.11 – Proteus compared to other classifiers. All methods are tested on the same validation set of nine proteins.

5.6 Comparison to Other Methods

The DISOPRED3 paper[18] list several classifiers for protein-binding interactions. The methods presented in that paper are summarized and compared to Proteus in figure 5.11.

When Proteus' score was calculated, the classifier was trained on all 1396 proteins in the ProS and MoRF datasets, and tested on the validation set. All 342 features calculated with window size 15 was used with a Random Forest classifier max 13 levels deep and 500 decision trees.

Proteus compares very favourably to the existing classifiers. It out-performs the competition on all four metrics: precision, recall, F1 and MCC.

5.7 Process Analysis

As I planned my work in machine learning before I knew anything about the subject, it should come as no surprise that I did not follow the time plan scrupulously. I will quickly narrate the process and point out the largest deviations from the time plan (see table 1.1).

I did not spend the first week learning the basics in Ubuntu and Python, as I figured I would learn along the way. The parsing of the XML database did not take a full week either. The generation of the first set of features went without troubles, and I used those early features as the basis for the statistics.

I felt way ahead of schedule and wanted to try my first machine learning algorithm. I was recommended to start with a linear support vector machine (an algorithm I have not talked about in this report). The calculations took 6 days and the result was dismally poor.

The search for other algorithms led me to *scikit-learn*, a machine learning package for Python. At the same time I started to use IPython. Both tools have been invaluable to me.

Random Forest became my algorithm of choice as it was both fast and easy to understand. But the result was still poor. I tried to improve my features and add new ones. I tried to do PCA analysis on the features to single out the most important ones, and I pre-processed my data in a number of ways. Still, very bad results.

The half-time report became a (belated) collection of graphs from all experiments I had tried so far. I had almost surrendered to the idea that I would never get any good results.

But the breakthrough came when I severely limited the depths of the decision trees (I had tried to limit them before, but not as much). All of a sudden my method performed better than the competition.

With the depth limit in place I had to redo all previous experiments, but that was only a matter of computer power. The final report was finished four weeks later than planned (but I taken two weeks extra vacation during the summer).

Chapter 6

Discussion

6.1 Heterogeneous Data

Proteus was optimized for the ProS and MoRF datasets. Despite that, it gets a much better score on the small validation set. That is not expected.

We found some very strange anomalies between the ProS and the MoRF dataset in chapter 4. Perhaps the two datasets describe slightly different things, and trying to predict both with the same classifier is therefore never going to work very well.

6.2 Feature Selection

In section 5.3 we found that many different combinations of feature groups yield very similar results. That is probably due to large redundancy in Proteus' features. All features stem directly from the amino acid sequence (what else could there be?). For instance, if we remove the amino acid concentration features, almost the same information is captured in the amino acid properties features. So very little information is actually lost and the classifier do almost as well on a smaller feature set.

6.3 The Curse of Dimensionality

I (naively) thought that more information would always be better. But I had not counted on the *curse of dimensionality*.

This expression refers to the fact that learning becomes exponentially harder as the dimensionality increases. That is because a fixed number of training examples cover a rapidly dwindling fraction of the input space as the dimensionality increases.

As an example, say we have a *huge* training set of one trillion (10^{12}) examples and every example have modest 100 binary features (only 1 or 0). Our training set still only cover a fraction of about 10^{-18} of the input space.

Even though the feature groups carry useful information, the information gain by adding them is offset by the increase in dimensionality. Therefore the classifier do almost as well with only feature groups 1, 3, 5 and 6 instead of all seven groups (see figures 5.7 and 5.8).

6.4 The Choice of Classifier

A linear function was used to illustrate a high-bias classifier in the introduction; the straight line could not learn the quadratic data no matter how we tweak it.

Decision trees do not have that problem. They can represent any boolean function. Their problem lies instead in high variance. A slightly different training set can lead to a completely different decision tree.[26]

We had to introduce a depth limit on the decision trees to cull the variance. That also introduced bias in the model. We had to balance a trade-off between bias and variance, the old dilemma (see section 2.3.2).

The fact that Random Forest worked better than Extremely Randomized Trees indicate that we have limited the depth of the decision trees so much that bias now is our main source of error. The extra random element in Extremely Randomized Trees decreases the variance while increasing the bias. That did not seem to be beneficial.

6.5 Only Half the Picture

The only thing that could affect whether or not a region in a protein is protean is the amino acid sequence. Why then is this problem so hard? If protean regions only depends on the amino acid sequence, why is it so hard to identify them even though we have access to the sequence?

Probably because protean regions only undergo a disorder-to-order transitions in close proximity to other molecules. And not any molecule, it must come into contact with its "partner". That means the *other half of the folding code* is in the partner molecule. If we had information about the partner molecule as well, maybe we could do even better.

6.6 Process Follow-Up

I do not think I could have made a better time plan at the beginning. What I could have done is to restructure the time plan with more details halfway through the project, but I honestly do not think that would have benefited me much.

What I do regret though is that I did not make a more thorough research of the available tools at the start of the project. The *scikit-learn* package was a great boon to me, the IPython notebook application streamlined my code base and the Seaborn package allowed me to produce nice graphs without effort. I wish I had had those tools at my disposal from the very beginning.

Chapter 7

Conclusions

7.1 Conclusions

- Proteus do better than the competition on all four metrics.
- The most important feature was the topography length, part of feature group 6 (see section 3.4).
- There were much redundancy in Proteus' features, so many different feature combinations yielded similar results.
- Random Forest with a depth limited to 13 levels was the optimal classifier.
- There were significant differences between the two datasets used to train the classifier. Still, the trained classifier did very well on the validation set.

7.2 Future Work

As more data become available the predictors will become better by the increasing number of training example.

It would be interesting to see an ensemble classifier which incorporate a number of already existing classifiers to see if they combined could produce even better results.

There is also more work to be done in the feature engineering. Proteus' most important feature was the topography length, a very quick-and-dirty feature which could be improved upon.

Bibliography

[1] Vladimir N Uversky, Joel R Gillespie, and Anthony L Fink. Why are natively unfolded proteins unstructured under physiologic conditions? *Proteins: Structure, Function, and Bioinformatics*, 41(3):415–427, 2000.

[2] Vladimir N Uversky. Natively unfolded proteins: a point where biology waits for physics. *Protein science*, 11(4):739–756, 2002.

[3] Dennis R Livesay. Protein dynamics: dancing on an ever-changing free energy stage. *Current opinion in pharmacology*, 10(6):706, 2010.

[4] Jonathan J Ward, Jaspreet S Sodhi, Liam J McGuffin, Bernard F Buxton, and David T Jones. Prediction and functional analysis of native disorder in proteins from the three kingdoms of life. *Journal of molecular biology*, 337(3):635–645, 2004.

[5] H Jane Dyson and Peter E Wright. Intrinsically unstructured proteins and their functions. *Nature reviews Molecular cell biology*, 6(3):197–208, 2005.

[6] Peter E Wright and H Jane Dyson. Intrinsically unstructured proteins: re-assessing the protein structure-function paradigm. *Journal of molecular biology*, 293(2):321–331, 1999.

[7] Peter Tompa. Intrinsically unstructured proteins. *Trends in biochemical sciences*, 27(10):527–533, 2002.

[8] Colin L Masters, G Multhaup, G Simms, J Pottgiesser, RN Martins, and K Beyreuther. Neuronal origin of a cerebral amyloid: neurofibrillary tangles of alzheimer's disease contain the same protein as the amyloid of plaque cores and blood vessels. *The EMBO journal*, 4(11):2757, 1985.

[9] KE Wisniewski, AJ Dalton, DR Crapper McLachlan, GY Wen, and HM Wisniewski. Alzheimer's disease in down's syndrome clinicopathologic studies. *Neurology*, 35(7):957–957, 1985.

[10] Kumlesh K Dev, Katja Hofele, Samuel Barbieri, Vladimir L Buchman, and Herman van der Putten. Part ii: α-synuclein and its molecular pathophysiological role in neurodegenerative disease. *Neuropharmacology*, 45(1):14–44, 2003.

[11] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[12] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[15] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[16] David T Jones. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of molecular biology*, 292(2):195–202, 1999.

[17] David T Jones and Jonathan J Ward. Prediction of disordered regions in proteins from position specific score matrices. *Proteins: Structure, Function, and Bioinformatics*, 53(S6):573–578, 2003.

[18] David T Jones and Domenico Cozzetto. Disopred3: precise disordered region predictions with annotated protein-binding activity. *Bioinformatics*, 31(6):857–863, 2015.

[19] Liam J McGuffin, Kevin Bryson, and David T Jones. The psipred protein structure prediction server. *Bioinformatics*, 16(4):404–405, 2000.

[20] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[21] Satoshi Fukuchi, Shigetaka Sakamoto, Yukiko Nobe, Seiko D Murakami, Takayuki Amemiya, Kazuo Hosoda, Ryotaro Koike, Hidekazu Hiroaki, and Motonori Ota. Ideal: intrinsically disordered proteins with extensive annotations and literature. *Nucleic acids research*, 40(D1):D507–D511, 2012.

[22] Satoshi Fukuchi, Takayuki Amemiya, Shigetaka Sakamoto, Yukiko Nobe, Kazuo Hosoda, Yumiko Kado, Seiko D Murakami, Ryotaro Koike, Hidekazu Hiroaki, and Motonori Ota. Ideal in 2014 illustrates interaction networks composed of intrinsically disordered proteins and their binding partners. *Nucleic acids research*, 42(D1):D320–D325, 2014.

[23] Fatemeh Miri Disfani, Wei-Lun Hsu, Marcin J Mizianty, Christopher J Oldfield, Bin Xue, A Keith Dunker, Vladimir N Uversky, and Lukasz Kurgan. Morfpred, a computational tool for sequence-based prediction and characterization of short disorder-to-order transitioning binding regions in proteins. *Bioinformatics*, 28(12):i75–i83, 2012.

[24] G.M. Cooper and R.E. Hausman. *The Cell: A Molecular Approach.* ASM Press, 2004.

[25] Jack Kyte and Russell F Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of molecular biology*, 157(1):105–132, 1982.

[26] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.