The Use of Pfam Hidden Markov Models for Enzyme Annotation by Machine Learning Methods Trained on Proteins from the Swissprot Database

Author: Michael Kubiak, University of Leicester

Date: 2nd December 2019

Abstract

Protein annotation is a necessary step in the analysis of organisms. It can explain how processes proceed, and how products are produced. Much of the biology community still uses methods, such as simply annotating based on the top BLAST result, which are based on heuristics, and so rather unreliable. More reliable methods of computational prediction are therefore a useful development to aid in the progression of biological understanding.

While there are a number of programs that implement such methods, many of them are focused on structural prediction as either a step on the path, or as their goal, and since structure is a very complex problem to solve, tools which bypass that step are likely to be much faster to run. Described here is an attempt to produce a method that does not rely on structural prediction, instead using protein family similarity.

Pfam families can, in many cases, be linked to a common set of functions that belong to the proteins that comprise them. This relationship can be used as a more useful starting point for annotation than the function of a single protein. Here, those links were discovered by use of HMMER to determine scores of a fully annotated protein database, Swissprot, against the whole Pfam database of families. Those scores and annotations were then used to train a machine learning algorithm, and so enable it to produce annotations for sets of HMMER scores of novel proteins against the same family database.

Three different techniques were trained to produce annotations, a completely random method (as a baseline), a random forest classifier, and a neural network regressor.

The completely random method performed as poorly as expected, with only a 0.6% accuracy on the full dataset.

The random forest was relatively good, with an accuracy of 41.4%, on the test dataset, but unfortunately, due to memory problems, could not be run on the full dataset.

The neural network was less good on the test dataset, 18%, but did run on the full dataset, with a maximum accuracy of 57.5% for the experiments performed. It is believed that this could be improved, and that a classification neural network will be a better way to annotate proteins, when tested.

Contents

1 Introduction

1.1 Functional Annotation

Protein function annotation is a highly important stage of the analysis of genomes. There are a number of annotation schemes that work for different sets of proteins, including Enzyme commission (EC) number, available from Bairoch (2000), which classifies enzymes by the reactions that they catalyse, and Gene Ontology (GO) terms (Ashburner et al. (2000), The Gene Ontology Consortium (2019)), which identify and relate functions of genes and gene products (RNA and proteins) across species.

1.1.1 Uses of Functional Annotation

Functional annotation prediction is used as a basis for further experimentation and manual annotation, particularly of new species. While many mammalian proteins can be identified easily based on a few homologues in closely related species, mammals are only a small section of the life that exists on the planet. Even proteins that are easily annotated to a primary function may have other, less obvious, functions that might be ignored unless discovered by functional prediction. Many newly discovered proteins from other branches of life do not have any close homologues that can be used to determine their function, and so are much more difficult to annotate. These require more in-depth, time consuming methods to be performed in order to discover their functions, and so anything that reduces the time taken by those methods is very useful. Functional prediction can be used to inform the starting points and possibilities explored through these methods, so that, in the case of enzymes, even if a precise function cannot be determined, a particular set of reactions can be investigated as the most likely, based on the predictions made.

1.2 Useful Data for Protein Annotation

1.2.1 Enzyme Commission (EC)

The scheme used for this initial foray into function identification by machine learning will be EC numbers, due to their relatively smaller scope (only classifying enzymes). An EC number has 4 sections, separated by dots, each of which specifies the enzyme function to a greater degree, from general reaction type down to specific substrates. For example, as can be found in Webb (1992) and its supplements, an EC number of 1.2.3.4 means that (1) the enzyme is an oxidoreductase, meaning that it catalyses oxidation/reduction, (1.2) that acts on an aldehyde or oxo group (1.2.3) with oxygen as the electron acceptor for the reaction, (1.2.3.4) and that it the specific molecule that is oxidised is an oxalate molecule. Due to their modularity, EC numbers give levels of annotation, allowing predictions to be useful even when they are not fully accurate, or do not exactly place the specifics of the later levels.

1.2.2 Pfam

Another way that proteins are classified is by homology, providing families that evolved from a common ancestor. Pfam (El-Gebali et al., 2019) is a database that holds profile 'seed' alignments and hidden markov models (HMMs) for these protein families. These can be used through programs such as HMMER (Potter et al., 2018), to determine the likelihood of a new protein being a member of that specific family. Proteins can be comprised of domains related to multiple families which work together to provide function. In the case that only the best HMMER score does not lead to a function prediction, it is possible that the scores against the entire database may allow the function to be discovered.

As shown by Alborzi et al. (2017), which describes the program ECDomainMiner, associations between EC number and Pfam family can be identified. These associations are useful for functional annotation, since Pfam is designed to allow identification to be done against it. Linking Pfam domains to their most likely EC numbers, in effect allows a protein to be searched against EC numbers in order to determine the chances of that protein having a relation to any one that is related to a Pfam domain. Rather than doing this manually, so that the EC numbers related to each family are searchable, it should be possible for a machine learning algorithm to computationally relate the domains to EC numbers, and so spot patterns that may not be initially obvious, as well as increasing the speed at which possible functions can be suggested.

1.2.3 UniProt

Information about known proteins is stored in the "UniProt Knowledgebase" database (UniProtKB) (The Uniprot Consortium, 2019). This information includes annotations, as well as the amino acid sequences of those proteins. UniProt itself consists of two databases, TrEMBL, which contains automatically annotated proteins that have not yet been reviewed and approved, and Swissprot, which contains manually annotated and reviewed proteins. Over time, papers manually annotating TrEMBL proteins are reviewed by a team and moved into Swissprot. This means that a piece of functional prediction software can be trained on the current Swissprot database, and tested by prediction of the whole TrEMBL database, part of which prediction can be confirmed by the next release of Swissprot, allowing a completely blind test.

1.3 Machine Learning

Machine learning allows a computer to perform a task and learn from ”experience” of that task, so that it can perform better on a future attempt. There are a number of distinct types of machine learning, including supervised, unsupervised and reinforcement learning. The choice of general type depends on the problem that must be solved, as well as that data that is available. In a case like the one discussed in this report, supervised learning can be used. This is the use of a large, representative dataset with known outcomes to train a model, which can then be used to predict the outcomes for data outside that initial set. Training of a supervised machine learning algorithm is relatively simple, as there is a required output, so the output can be tested for how accurate it is. Unsupervised learning has unlabelled data, and is used for applications such as clustering and finding unknown patterns. Knowing whether an unsupervised algorithm has produced an answer that is related to the question being asked is difficult due to the difficulty of knowing how the decisions were made. The final type is reinforcement learning is somewhat similar to supervised learning, as there is direction, based on a required outcome (winning a game of chess, etc.). The actions taken are provided a ’score’ based on whether they bring the system closer to or further from a desired goal. This makes it useful in cases where interaction with an outside agent is required.

Machine learning is useful because it reduces the need for hard coded rules that a program must follow, and opens the possibility of finding patterns of relation that are as yet undiscovered. The models built using machine learning provide methods of determining answers to questions that are difficult to answer in any other way.

One feature that increases difficulty of answering questions is the amount of data that may be related to a specific problem. This can be seen in the size of bioinformatics datasets, which has been increasing exponentially due to improved methods of obtaining data. When particular questions are asked this abundance of data can make finding the patterns difficult. In the particular case of training a machine learning algorithm to annotate protein function based on HMMER searches of Swissprot against pfam, the size of Swissprot, along with the number of pfam families means that the relations between those and EC numbers can be difficult to discern. Even when those relations can be found, hard coding them would be an almost impossible task, due to the time it would take. Coding a machine learning algorithm, on the other hand, is a much shorter process.

1.3.1 Libraries

There are various libraries, in various languages, that have been written to make the process of generating a machine learning model easier. Python, which will be used here, has a number machine learning libraries. One of these is Scikit-learn (Pedregosa et al., 2011), which has a large number of pre-written algorithms for many different methods, including various classification, regression and clustering algorithms, as well as data preparation modules for uses such as dimensionality reduction.

Another option, TensorFlow (Abadi et al., 2015) has a toolkit that can be used to produce more specialised algorithms, so that the machine learning solution can be tailored to the task.

1.3.2 Common Techniques

Three of the most common types of supervised machine learning are random forest classification, initially suggested in Tin (1995), and first used in Breiman (2001), support vector machines, whose first appearance in literature is in Boser et al. (1992) and deep neural networks, which is an extension of the idea of neural networks, the mathematics of which were first proposed in McCulloch (1943).

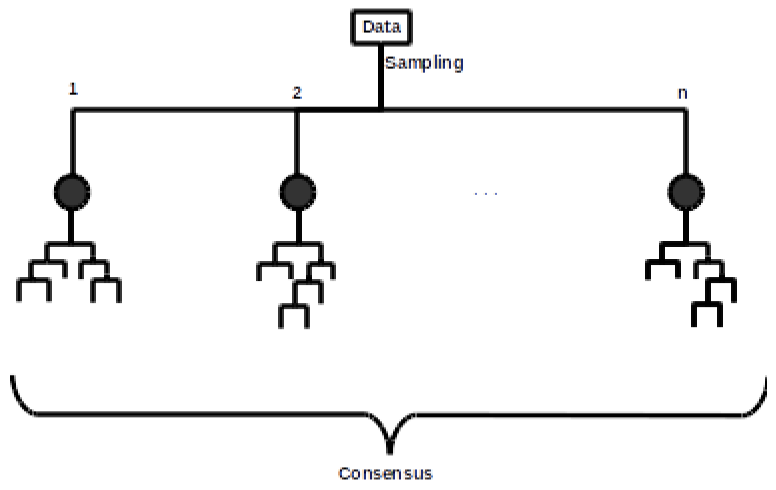


Figure 1: A simple illustration of a random forest, showing how each of a number (n) of samples makes a tree that contributes to the consensus that will be reported

In random forest classification, the data is sampled (with replacement), and each sample is used to grow a decision tree. Each node in the tree splits the data based on a particular question, for example whether an input variable is greater than a threshold value, so that the sampled data is separated back into the known classifications. Once the forest is grown, new data is classified based on the consensus opinion of the trees in the forest. A representation of a random forest is shown by figure 1. Random forests have very few parameters that can be tuned to change their performance, i.e. sample size and tree number/depth. They have multi-class classification built in due to the nature of decision trees, where each branch can lead to a certain classification. This means that they make a good initial benchmark for any machine learning project.

A Support vector machine (SVM) attempts to maximise the 'distance' between the decision boundary, the plane (in a plot of the data points) by which classifications are separated, and the points that it separates. SVMs are less likely to be specific to a training dataset because the boundaries are not being designed only to separate the classes, in which case they might curve around a particular point that is somewhat of an outlier to a group, but instead to ensure that there is the greatest margin between the decision boundary and all of the points that it is separating. In that case of non linearly separable problems, SVMs use the 'kernel trick' to transform the data to a higher dimensional space, in which it becomes linearly separable. The choice of kernel (transformation function), as well as how strict the SVM is about incorrect classification are variable, and can be used to tune for better performance. SVMs usually rely upon One vs All (OvA) classification, meaning that they recognise whether a sample is each possible classification in turn, rather than determining all memberships at the same time.

A neural network (Figure 2) consists of an input layer (the data), a set of hidden layers (the computation) and an output layer (the classification). A deep neural network has many hidden layers, which allows it to model more complex systems. Each layer contains a number of nodes, which are linked to every node in the subsequent layer. The links between nodes each have a weight that determines how the output of the previous node affects the next one. The nodes in the hidden layers each have a bias, which is applied to their total input, and a function that they perform, which determines their output. The network is trained by putting a training value, whose expected output is known, adjusting weights and biases based on the difference between the predicted outcome and the true outcome, by a technique called back-propagation. The use of the network is identical to the training, except without the score calculation and back-propagation. The output layer can be comprised of continuous values, which, in a classification network, provide the probabilities that a certain classification is correct. As with the SVM, a deep neural network is highly tunable, with factors such as number of hidden layers, functions on the nodes, and even learning rate of the system (how much weights and biases are altered during each back-propagation step.

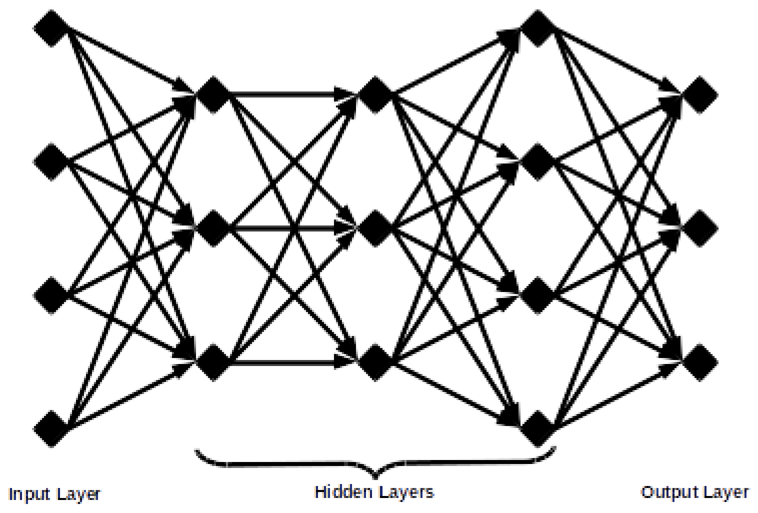


Figure 2: A simple illustration of a neural network with 3 hidden layers. Weights are applied on the connections between nodes, and biases are applied on the nodes themselves, before functions are applied

1.4 Functional Prediction in Literature

In many cases, functional prediction is still performed by taking the first BLAST result of a sequence. This is a relatively arbitrary method of annotation since, due to the heuristic nature of the BLAST algorithm, repeated BLAST searches of the same sequence very often do not produce the same first result. This problem necessitates the development of tools that can more reliably predict protein annotations.

1.4.1 Experimental methods

One type of approach is that which takes information from a specific type of experiment, and attempts an annotation based on that. A method of this variety is described in Espadaler et al. (2008), which details the use of sequence similarity, as well as interaction partners of the proteins to predict its action as an enzyme. This approach needs a set of experiments to determine the interaction partners, which can be difficult to determine, as well as computational methods for each protein. It builds upon previous attempts at prediction, which used only the interaction partners, and no computational methods at all. A second example of protein function prediction using experimental results is Bandyopadhyay et al. (2006), which describes using structural ’packing patterns’ of amino acids that are common in specific families, but much rarer in PDB in general. In this case, a complete structure is required, so that the packing of amino acids can be identified, before annotation can be attempted. Also, if the packing of the specific amino acid is not rare in the PDB, this method is unlikely to determine function with any certainty.

1.4.2 Purely Computational Methods

1.4.2.1 Structure Prediction for Functional Annotation

The ‘DeepMind’ project, funded by Google, has attempted structural prediction, rather than functional prediction, through its offshoot ‘AlphaFold’, described in AlQuraishi (2019), which used deep neural networks to predict protein folding. Wang et al. (2011) also performs structural prediction, these could be thought of as less closely related to this project, but the second example, above, shows that structural information can also be used in functional prediction.

1.4.2.2 Family Based Functional Annotation

Jung et al. (2014) describes the use of three classification techniques, SVM, Sequential Minimal Optimisation (Platt, 1998), which is an extension of SVM which improves learning time, and Adaptive Boosting (Freund (1995) and Schapire (1990)), which combines the output of other machine learning techniques to produce a weighted sum, to relate families and groups found by InterPro (Mitchell et al., 2019) to GO terms. This method is much more similar to the intentions of this project, since it is entirely computational in nature, and uses calculated families to determine likely function. The difference from this project, here, is in what type of annotations are performed. This paper annotates GO terms, rather than EC numbers, as will be attempted here.

1.4.2.3 EC Number Annotation

ECPred (Dalkiran et al., 2018) describes an EC annotator based on an ensemble of machine learning methods. It brings together 3 independent predictors, subsequence-based feature mapping (Sarac et al., 2008), a BLAST score based k-nearest neighbour algorithm, and an SVM based on amino acid ‘physicochemical features’.

1.5 Prediction Competitions

The two structural predictors above (AlphaFold and AlQuraishi (2019)) have been used as part of a group that are judged by competitions such as CASP (Critical Assessment of Structure Prediction), the most recent of which was CASP13 in 2018 (Kinch et al., 2019). CASP provides teams with a set of proteins whose structures are unknown, but will be worked on experimentally once models are submitted. Each team produces predictions based on their modelling software, and those models are graded based on their accuracy. Its offshoot, CAFASP (Critical Assessment of Fully Automated Structure Prediction) was a similar system, except that no human input was allowed, the modelling software was uploaded to a server, and its raw output was graded.

CAFASP has been superseded by another competition, CAFA (Critical Assessment of protein Function Annotation) (Friedberg and Radivojac, 2017) whose most recent competition CAFA took place in 2018-2019. CAFA is more relevant to this project, since the algorithms perform annotation, rather than structure prediction.

The purpose of all of these competitions is to assess how far the field of computational prediction has come in their areas, so that further advances can be made using techniques that iterate on the most successful.

1.6 Aims

The project will attempt to implement some of the functionality of Schmid and Blaxter (2008), but using machine learning, and allowing for multi-domain proteins. The intention is to attempt to use supervised machine learning to produce an algorithm which annotates a protein based on its HMMER scores against the Pfam HMMs. The learning will be done using proteins from the Swissprot database. Since the enzymes among these have EC numbers assigned to them, those classifications can be used as targets for the algorithm to aim towards. It is expected that a number of different supervised machine learning algorithms will be tried, starting with a random forest, and developing from there, based on success or failure of that approach, and any advice that more experienced machine learning users might provide. The variety of attempts will allow the best implementation to be discovered.

2 Methods

The full method can be run using the annotate bash file (Appendix A).

2.1 Data Retrieval

The Swissprot database was downloaded from Uniprot (The Uniprot Consortium, 2019). Due to the rolling updates to the database the version used is now available as a part of the file at `ftp://ftp.uniprot.org/pub/databases/uniprot/previous_releases/release-2019_08/knowledgebase/uniprot_sprot-only2019_08.tar.gz`. The Pfam HMM database version 32.0 was also downloaded from the EBI servers. Enzyme.dat, the file that provided known enzyme commission numbers for the proteins, was downloaded from the expasy servers. Previous versions are not available, but the version used for this project is present on the GitHub page. The bash script is shown in appendix B.1.

2.2 HMMER

HMMER (Potter et al., 2018) version 3.1b2 was used to generate scores for each pfam family against each Swissprot protein. It has two functions for generating scores, 'hmmsearch' compares each HMM to the full database of sequences, while 'hmmscan' runs each sequence against the full HMM database. As described on Eddy (2011), 'hmmsearch' is the faster of the two, for large datasets, due to the much larger input/output load required for 'hmmscan' to function.

The HMMER searches were done using 3 different reporting thresholds for comparison of their effects on models. These thresholds were E-values of 10, 1, and 0.1. All other settings were left at their default values. The results were output as a table to allow easier parsing.

2.3 Preprocessing of HMMER Scores for Use in Training

The tables of scores were converted into a scipy (Jones et al., 2001) sparse matrix, as shown in appendix B.2.1. The targets were also placed into a sparse matrix, this time with boolean values, due to the absolute nature of the classification. The orders of the proteins in these two matrices were kept consistent for ease of comparison, and the orders of each axis of the two matrices was also output to file. The creation of the target matrix was performed using the script in appendix B.2.2.

Other additional preprocessing was performed by the scripts in appendix B.2.3. These differ slightly dependent on the architecture of the algorithm to be used. For the initial attempts, which used classification algorithms, this module simply provides methods with purposes like splitting sparse data into training and test sets (with a 7:3 ratio), removing unnecessary EC numbers (with no proteins),

proteins with no family hits or no EC hits (although these are useful in smaller quantities as a testable true negative) and, later, removing duplicate proteins to improve the running time and memory usage of these algorithms. For later attempts, which used a regression architecture, this module also provided a method to order the EC numbers, so that they could be related to values between 1 and 4940 with 0 denoting no EC number annotation. This function then reduced the matrix to a series of lists, each of which contained the annotations for one protein.

In all cases, the number of non-enzyme proteins was reduced to less than 20% to avoid a fully negative predictor.

2.4 Random Assignment

As a baseline, a naive predictor was produced. This predictor acts by rearranging the rows of the target matrix, to ensure that the same ratios of outcomes were produced. This predictor is created by the module in appendix A.1.1.

2.5 Random Forest

The first attempt at a machine learning classifier was a random forest, using the Scikit-learn (Pedregosa et al., 2011) library (version 0.21.3), since they have very few tunable variables, and so make a good starting point. The variable with the greatest effect on the output of the forest is the number of trees that are built. The random forest training function can be seen in appendix A.1.2.

2.6 Neural Network

A deep neural network was the obvious choice for a third architecture. Some attempts were made using the Multi Layer Perceptron architecture in Scikit-learn, but it was determined that the best way to allow more input and testing flexibility was to use a slightly lower level package, Keras version 1.1.0 (Chollet et al., 2015), with a Tensorflow 2.0.0 (Abadi et al., 2015) back-end. This allowed more granularity in changing variables, such as hidden layer number, size and activation functions, as well as greater control of the training process. Appendix A.1.4 is written to train the network on a batch generator.

2.7 Testing

2.7.1 Test Data Set

To enable easier appraisal of techniques, appendix B.2.4 was used to produce a smaller test dataset that could be used to run an algorithm quickly. This dataset contained only proteins that had scores against an arbitrary group of families, chosen to have exactly 10 protein hits, of which a high proportion were EC annotated. The two files are differentiated only by their ability to process different structures of target data structures.

2.7.2 Metrics

The comparison of models requires them to be tested on a common problem. Appendix A.2 provides methods of testing models against one another. The classifier script is complete, but the regressor version is, unfortunately, still a work in progress, this means that there can be no exact comparison between the two types of method. Regression models were simply evaluated based on their accuracy for a set random seed of 1, as well as classifying the continuous prediction by EC set (what the first 3 numbers of the annotation were, for example).

Fairness was ensured by setting random seeds to standard values across runs, a range between 0 and 10, the values reported (for the classifiers) are the mean of these outputs, and standard deviations are reported where possible. Usefulness of these models was determined using four metrics, accuracy, sensitivity, specificity and precision. Accuracy was calculated for the total prediction, and all four metrics were calculated per first number of the EC (1-7).

2.7.2.1 Accuracy

Accuracy is a measure of how close to correct each prediction is. It is calculated by the formula:

$$Accuracy = \frac{CorrectPredictions}{CorrectPredictions + FalsePredictions}$$

Correctness of a prediction was determined by the complete matrix row being identical between the target and the prediction.

For the regression algorithm, the accuracy was calculated by the keras package (Chollet et al., 2015), from the loss calculated using the loss function (Mean Squared Error).

2.7.2.2 Sensitivity

Sensitivity is a measure of how well classifications are discovered, i.e. what fraction of the actual classifications are labelled correctly. It is calculated using the formula:

$$Sensitivity = \frac{TruePositives}{TruePositives + FalseNegatives}$$

2.7.2.3 Specificity

Specificity is a measure of how well negative classifications are made, i.e. what fraction of the proteins that should not be annotated with a specific classification are not annotated in that way. Unfortunately, this metric is less useful for data with many negatives in the targets, such as the target matrix used in classification, as they will get very high percentage specificities. The formula for specificity is:

$$Specificity = \frac{TrueNegatives}{TrueNegatives + FalsePositives}$$

2.7.2.4 Precision

Precision is a measure of how likely a classification is to be true, i.e. what fraction of the classifications predicted are correct. It is calculated as:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

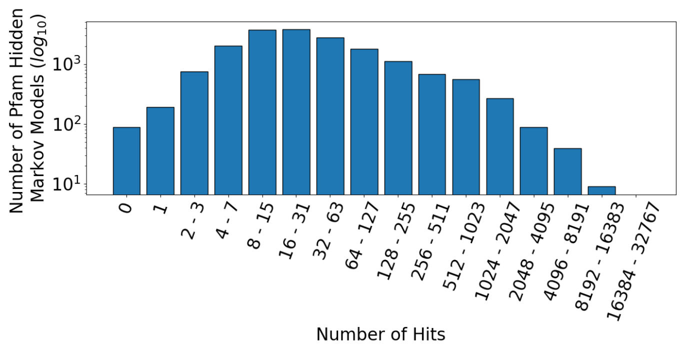
3 Results

3.1 Data Analysis

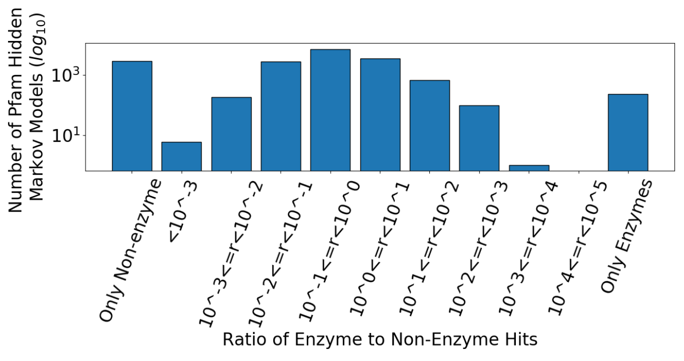
Data analysis is important to improve understanding of the dataset that is being worked with. These results will be discussed in the results section due to their bearing on how the rest of the project proceeded, and on the testing that was performed on some architectures.

3.1.1 Pfam Frequencies

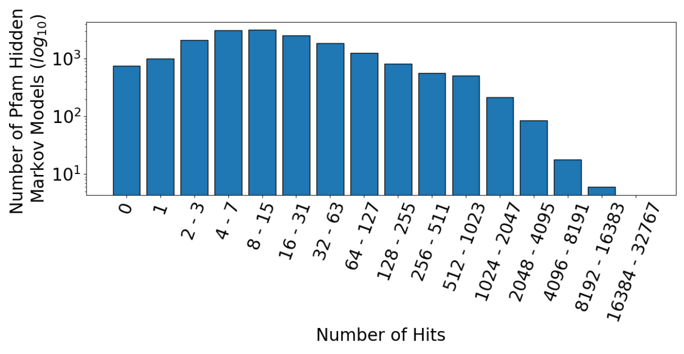
Figure 3 shows how the use of different cut-offs impacts upon the distribution of hits of hidden markov models (a, c and e) and the ratio of enzyme to non-enzyme hits of those HMMs (b, d and f). This figure shows that there is a great difference in the data that is obtained with different cut-off values, and so suggests that attempting machine learning on the different datasets may give different results. How those results differ could give an insight into whether that information is true, or just false negatives



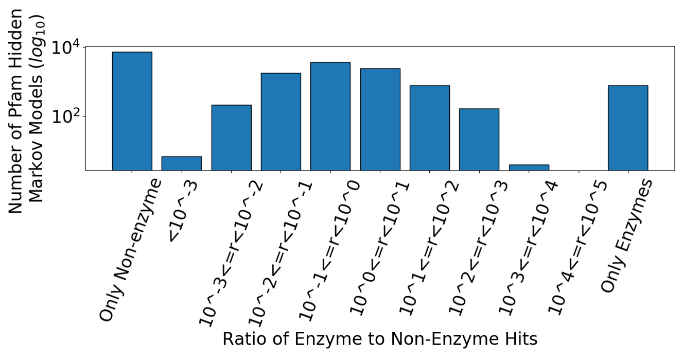
(a)



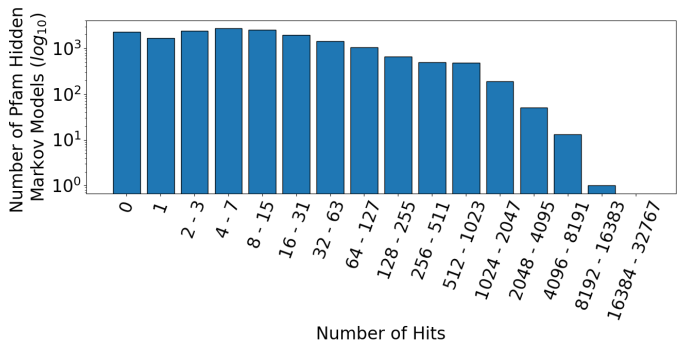
(b)



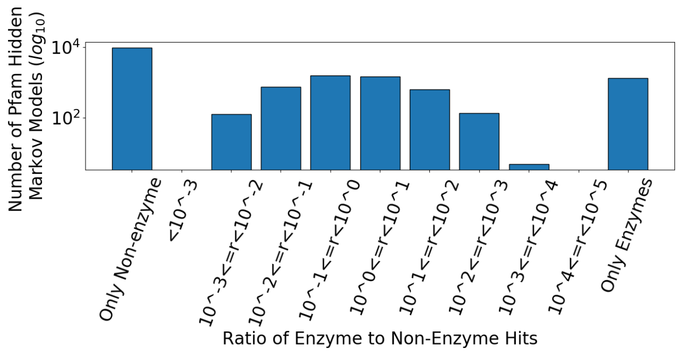
(c)



(d)



(e)



(f)

Figure 3: Histograms showing the effect of changing E-value cut-offs on frequency of numbers of HMM hits, as well as enzyme/non-enzyme ratios of HMM hits for HMMER cut-offs of E-value = a/b) 10, c/d) 1 and e/f) 0.1. As can be seen in a), c) and e), the distribution of hits shifts greatly towards the lower end with increasingly stringent cut-offs, but there are a still number of highly promiscuous HMMs with large numbers of hits. In b), d) and f), it can be seen that the ratios tend to become more extreme for more stringent cut-offs, with the central bars (both enzyme and non-enzyme for the same family) reducing in frequency, while the only enzyme and only non-enzyme bars grow

generated by overgeneralising the family models. It was also possible that a good enough model would implement its own stringent cut-off on the data through the learning process, and so end up producing the same results for two datasets with different cut-offs.

3.1.1.1 Coverage of Swissprot by Pfam HMMs

As can be seen from the histograms in in figure 3 parts a), c) and e), the distribution of hits per HMM are spread across a large range, from 0 to 15772 hits for the least stringent e-value cut-off of 10. This highly hit model is PF13191.6, which is a family of enzymes containing an AAA ATPase domain, as can be discovered from the Pfam website at `http://pfam.xfam.org/family/pf13191.6`. As an ATPase family, it would be expected to be quite common, due to how commonly used ATP is as an energy transporter. The number of hits for this family reduces to 11923 for the e-value 1 cut-off, and to 7389 (less than half of the initial value) for the most stringent cut-off. This large reduction in number of hits means that PF13191.6 is no longer the most hit family. For the e-value 0.1 cut-off, that becomes PF01926.23, with 9097 hits (12181 at e-value 10). This family contains ribosome GTPases, another type of enzyme that is necessary for life, due to the ubiquity of RNA, which the ribosome synthesises using, among other molecules, GTP.

The mean numbers of hits per HMM are 123 ($\sigma = 470$), 96 ($\sigma = 373$) and 77 ($\sigma = 294$) for e-value cut-offs of 10, 1 and 0.1 respectively. The shift towards lower numbers of hits demonstrated by this reduction of the mean is also shown through the change in distribution between the histograms. The distribution changes from a more normal distribution to a more uniform one, at least for lower numbers of hits, such as the 0 bar, where the number of HMMs increases roughly 20-fold between a cut-off of 10 and a cut-off of 0.1.

3.1.1.2 Enzyme Ratios of Pfam HMMs

Again, the differences between the cut-offs is visible in figure 3 parts b), d) and f), especially at the lowest ratio ($< 10^{-3}$), where the value reduces to 0 for the most stringent cut-offs. There is also a marked decrease in the central bars, from the high thousands to roughly 1000, and a corresponding increase in the bars that denote only one type of protein. This suggests that many of the families are segregated between enzyme and non-enzyme, but that the less stringent cut-offs cause the inclusion of some enzymes in non-enzyme families, and *vice versa*.

3.1.1.3 Summary

The differences in numbers of hits, and enzyme/non-enzyme ratios for Pfam domains under different e-value cut-offs (shown in Figure 3) suggested that an algorithm might be affected by which dataset was used. This led to the testing of the random forest with all of the obtained datasets.

3.1.2 Swissprot Statistics

Figure 4 shows that there are many more proteins with 0 or 1 annotations than there are with more (more than 10x), meaning that an architecture that predicts only one annotation per protein can be almost as good as one that predicts more, especially if the almost identical EC numbers that make up many of the higher annotation counts are taken into account.

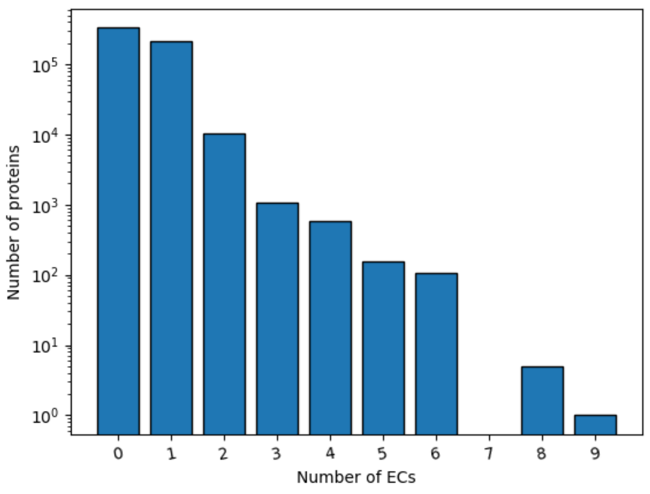


Figure 4: Plot showing the distribution of numbers of EC annotations per protein. Many proteins have a few EC annotations, but it rapidly falls off at higher numbers

The protein with the most EC annotations (a total of 9) is a maize protein, (E)-beta-farnesene synthase, which has a group of functions that are involved in the syntheses of a set of molecules from (2E, 6E)-farnesyl diphosphate, information about which is available from The Uniprot Consortium (2019). As this protein acts upon only one substrate, it is likely that any other proteins that perform this set of functions, which would be very uncommon, would have these other functions discovered upon validation of any prediction by an experimental method.

It was also determined that 59.16% of the dataset was non-enzyme proteins, which are useful as a true negative, but could lead to a fully negative predictor getting a much higher accuracy than it deserved. This caused possible complication led to the decision to reduce the number of non-enzyme proteins to below 20%.

3.1.2.1 Summary

Some proteins were discovered to have a high number of EC annotations. These were, in many cases, related, so predicting only one annotation was determined to be a possible reduction in prediction complexity, with minimal adverse effects and was attempted in the neural network.

3.2 Architecture Outcomes

Different architectures were tested so that they could be compared and contrasted as to their usefulness.

3.2.1 Random Assignment

The random assignment model had an accuracy of 0.10% ($\sigma = 0.05$) on the full dataset and, for comparison with the random forest, 0.56% ($\sigma = 1.18$) with the test dataset. For the other metrics, since they have not been calculated for the neural network, only the test dataset values are reported here, with the full dataset values in appendix C.1.1. The mean sensitivity across sets of EC numbers was 0.65% ($\sigma = 1.45)\%$, the mean specificity was 99.55% ($\sigma = 0.02$) and the mean precision was 0.65% ($\sigma = 0.5$).

3.2.2 Random Forest

The most accurate type of random forest that could be generated had an accuracy of 41.45% ($\sigma = 12.17$) on the test dataset. The sensitivity was 47.62% ($\sigma = 14.17$), specificity was 99.81% ($\sigma = 0.03$), and precision was 56.18% ($\sigma = 13.50$). Unfortunately, random forests do not scale well in terms of time or memory complexity, and running this architecture for the full dataset would have required a very large amount of both.

The tuning that was possible for this architecture was in the number of trees which are built, and so the number of estimators from which the consensus was gathered, (for the case above, a value of 1000 estimators was used) and in the allowed depth of the trees. These are both variables that would have to be tuned for the size of dataset that would be used. In the same way as for random assignment, full results, including results separated by first EC number are shown in appendix C.1.2.

3.2.3 Neural Networks

The best neural network produced had an accuracy of 57.5% when trained on the full dataset (values displayed in C.2), but only 18% on the test dataset. The networks' accuracy values were only slightly different dependent upon the shape of the network, almost all within one percent of one another, so more testing would be needed to determine which, if any, variables could have a greater impact upon the quality of the network.

3.2.4 Summary

The Random Assignment method performed poorly on all datasets, the random forest performed relatively well on the test dataset, but could not be run on the full dataset, and the neural network performed less well on the test dataset than the random forest, but did perform well on the full dataset.

4 Discussion

4.1 Comparisons

4.1.1 Random Assignment Method

As was expected, the completely random method of prediction performed very poorly, with the exception of the specificity, which, as stated in the methods section (specifically section 2.7.2.3) is a less useful metric for this type of prediction, due to the large number of negatives that will always cause artificial inflation of the specificity unless the prediction contains many false positive predictions. The random assignment method performed worse on the full dataset than the test dataset, which, again, is as expected, since there was a smaller probability that the values 'predicted' would be correct by chance.

4.1.2 Random Forest

The random forest performed much better than the random assignment on all useful metrics. Unfortunately, this method was not realistically scalable with the time and technology that was available. Given a large enough amount of memory and enough time, this method could, of course, be tested to a better degree. If an annotation program were to use it, however, the end user would not be likely to have such a cluster to hand, and so could not update the model for a newer version of the databases.

Random forests could be a good way of predicting outcomes from a smaller amount of data, but is unlikely to be useful for any task with as large a dataset as the one used here.

The results for the different e-value cut-offs show that there was no effect of using a different cut-off, which suggests either that the random forest learned around the extra data, as if it was implementing its own cut-offs, or that the test dataset did not contain any of the higher e-value relations.

4.1.3 Neural Network

The neural networks produced were much less accurate on the test dataset than the random forest (although they were still better than the random assignment method). Unfortunately, as stated in the methods (section 2.7) other metrics are not yet available for the neural network, and so it cannot be compared on those.

The neural networks were much faster to run, and had far lower ram costs than the random forest, and so were much easier to train.

4.2 Classification vs Regression

It is not known how well the test dataset models the full dataset, but the fact that the neural network performed better on the full dataset implies that other learning algorithms would perform better there also. This suggests that the random forest would be quite likely to have a higher accuracy on the full dataset than the neural network. One possible reason for the increased accuracy of the random forest was the type of output. The idea to use regression for the problem instead was worth testing, since it greatly reduced the size of output from the model, but it also enabled a slight drift from the correct value to place the output in a completely different category than the true annotation. Classification is expected to provide the best results for this type of problem, but as with all conclusions reached, this is subject to further experimentation.

4.3 Future Work

4.3.1 Testing

For better, more scientific comparison between architectures, methods to obtain all metrics should be implemented. This would be particularly important when comparing between regression and classification methods.

Specificity might be turned into a more useful metric by only including proteins that had no true classifications in the calculation, rather than taking each value in the matrix and making a comparison.

As a final test, once the program is complete, it should be tested in a completely blind way, submission to a competition such as those mentioned in the introduction (in section 1.5), or by training on the current version of Swissprot, and testing on the new annotations in the next version.

4.3.2 Model Improvements and Further Experimentation

As stated above, it is believed that a classification neural network would perform best, and so the development of such a network should be a priority.

It is probable that the regression network built previously is not optimal, and so more experimentation should be performed. It is also possible that a network with multiple regression outputs, such as one per first digit in the EC, would provide a more useful network.

All neural network experimentation should include close scrutiny of all possible variables, which includes finer changes in shape, such as having differing numbers of nodes in different hidden layers, but also activation functions, number of training epochs, size of training batches, as well as others.

4.3.3 Increasing Scope

While EC number is a useful classification system, it only provides annotations for one part of the spectrum of proteins. Including other annotations systems, such as GO, would allow the tool to be more flexible. This could take one of two forms, one large model, which determines all of the annotations for a particular protein, or two smaller models that could be run together or separately, depending on the wishes of the user. It is likely that the second option would be most time and memory efficient, but the single network might produce better results, due to links between different annotation systems.

4.3.4 Improving Accessibility

Completing the program in such a way that it was useful to scientists annotating real proteins would of course be the natural conclusion to the project. This could be achieved by producing a fully functional local version, using a server to run queries, or both, with the local version available if a more specialised model or a quicker result was required by a team with resources to build their own models.

4.3.5 Reduce Family Sizes

Some Pfam families have many proteins with different annotations, as shown by the central bars of figure 3 parts b), d) and f). These families could be reduced into more specific HMMs which only contained either enzymes or non enzymes, rather than both, improving the chances of producing correct annotations.

5 Conclusion

Due to the (likely) better accuracy of the random forest, and the better resource use of the neural network, it is believed that a classification neural network would prove to be the best method for a program of the type being developed (the most efficient for the highest accuracy). More work is required on this algorithm, but, if completed, it could be very useful to any group attempting novel protein annotation.

While all accuracies obtained are much lower than those obtained in the literature, such as Wang et al. (2011), which states an accuracy of between 81% and 98% for the first 3 numbers of the EC (which precludes a direct comparison prior to further metric development), but used a structure prediction followed by function prediction, it is thought that this method should be much faster to apply, and could become much more accurate given enough work and experimentation.

To a large degree, the aims of the project were achieved, a model was built that could, relatively accurately, determine the EC number of a protein. The neural network moved slightly away from the initial aim of including multi-domain proteins, since it only had one output value, but it should be relatively easily adapted to produce more than one output, by changing to classification, rather than regression.

Acknowledgments

My thanks to Dr Ralf Schmid, my supervisor, for guiding my development of techniques and my analysis, and also to Professor Zhang for meeting with us and providing ideas, such as to use a regression algorithm, and to develop the neural network in keras, rather than in sci-kit learn.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015. URL https://www.tensorflow.org/.

Alborzi, S. Z., Devignes, M.-D., and Ritchie, D. W. Ecdomainminer: discovering hidden associations between enzyme commission numbers and pfam domains. *BMC bioinformatics*, 18(1):107, Feb 13, 2017. doi: 10.1186/s12859-017-1519-x. URL https://www.ncbi.nlm.nih.gov/pubmed/28193156.

AlQuraishi, M. Alphafold at casp13. *Bioinformatics (Oxford, England)*, May 22, 2019. doi: 10.1093/bioinformatics/btz422. URL https://www.ncbi.nlm.nih.gov/pubmed/31116374.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., and Sherlock, G. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nature genetics*, 25(1):25–29, May 2000. URL https://www.ncbi.nlm.nih.gov/pubmed/10802651.

Bairoch, A. The enzyme database in 2000. *Nucleic acids research*, 28(1):304–305, Jan 1, 2000. doi: 10.1093/nar/28.1.304. URL https://www.ncbi.nlm.nih.gov/pubmed/10592255.

Bandyopadhyay, D., Huan, J., Liu, J., Prins, J., Snoeyink, J., Wang, W., and Tropsha, A. Structure based function inference using protein family specific fingerprints. *Protein Science*, 15(6):1537–1543, Jun 2006. doi: 10.1110/ps.062189906. URL https://onlinelibrary.wiley.com/doi/abs/10.1110/ps.062189906.

Boser, B., Guyon, I., and Vapnik, V. A training algorithm for optimal margin classifiers. COLT '92, pages 144–152. ACM, 1992. doi: 10.1145/130385.130401. URL http://dl.acm.org/citation.cfm?id=130401.

Breiman, L. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. doi: 1010933404324. URL https://search.proquest.com/docview/757027982.

Chollet, F. et al. Keras, 2015. URL https://keras.io.

Dalkiran, A., Rifaioglu, A. S., Martin, M. J., Cetin-Atalay, R., Atalay, V., and Doan, T. Ecpred: a tool for the prediction of the enzymatic functions of protein sequences based on the ec nomenclature. *BMC bioinformatics*, 19(1):334, Sep 21, 2018. doi: 10.1186/s12859-018-2368-y. URL https://www.ncbi.nlm.nih.gov/pubmed/30241466.

Eddy, S. hmmscan vs. hmmsearch speed: the numerology, , May 27 2011. URL https://cryptogenomicon.org/2011/05/27/hmmscan-vs-hmmsearch-speed-the-numerology/.

El-Gebali, S., Mistry, J., Bateman, A., Eddy, S. R., Luciani, A., Potter, S. C., Qureshi, M., Richardson, L. J., Salazar, G. A., Smart, A., Sonnhammer, E. L., Hirsh, L., Paladin, L., Piovesan, D., Tosatto, S. C., and Finn, R. D. The pfam protein families database in 2019. *Nucleic Acids Research*, 47 (D1):D427–D432, Jan 8, 2019. doi: 10.1093/nar/gky995. URL https://www.ncbi.nlm.nih.gov/pubmed/30357350.

Espadaler, J., Eswar, N., Querol, E., Avils, F. X., Sali, A., Marti-Renom, M. A., and Oliva, B. Prediction of enzyme function by combining sequence similarity and protein interactions. *BMC bioinformatics*, 9(1):249, May 27, 2008. doi: 10.1186/1471-2105-9-249. URL https://www.ncbi.nlm.nih.gov/pubmed/18505562.

Freund, Y. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2): 256–285, 1995. doi: 10.1006/inco.1995.1136. URL https://www.sciencedirect.com/science/article/pii/S0890540185711364.

Friedberg, I. and Radivojac, P. Community-wide evaluation of computational function prediction. *Methods in molecular biology (Clifton, N.J.)*, 1446:133–146, 2017. doi: 10.1007/978-1-4939-3743-1_10. URL https://www.ncbi.nlm.nih.gov/pubmed/27812940.

Jones, E., Oliphant, T., Peterson, P., et al. SciPy: open source scientific tools for python, 2001. URL http://www.scipy.org/. [Online; accessed].

Jung, J., Lee, H. K., and Yi, G. A novel method for functional annotation prediction based on combination of classification methods. *TheScientificWorldJournal*, 2014:542824–9, 2014. doi: 10.1155/2014/542824. URL http://dx.doi.org/10.1155/2014/542824.

Kinch, L. N., Kryshtafovych, A., Monastyrskyy, B., and Grishin, N. V. Casp13 target classification into tertiary structure prediction categories. *Proteins: Structure, Function, and Bioinformatics*, Jul 24, 2019. doi: 10.1002/prot.25775. URL https://search.proquest.com/docview/2256098145.

McCulloch, W. S. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943. URL https://mdc.ulpgc.es/cdm/ref/collection/bibliodes/id/2.

Mitchell, A. L., Attwood, T. K., Babbitt, P. C., Blum, M., Bork, P., Bridge, A., Brown, S. D., Chang, H.-Y., El-Gebali, S., Fraser, M. I., Gough, J., Haft, D. R., Huang, H., Letunic, I., Lopez, R., Luciani, A., Madeira, F., Marchler-Bauer, A., Mi, H., Natale, D. A., Necci, M., Nuka, G., Orengo, C., Pandurangan, A. P., Paysan-Lafosse, T., Pesseat, S., Potter, S. C., Qureshi, M. A., Rawlings, N. D., Redaschi, N., Richardson, L. J., Rivoire, C., Salazar, G. A., Sangrador-Vegas, A., Sigrist, C. J., Sillitoe, I., Sutton, G. G., Thanki, N., Thomas, P. D., Tosatto, S. C., Yong, S.-Y., and Finn,

R. D. Interpro in 2019: improving coverage, classification and access to protein sequence annotations. *Nucleic Acids Research*, 47(D1):D351–D360, Jan 8, 2019. doi: 10.1093/nar/gky1100. URL https://www.ncbi.nlm.nih.gov/pubmed/30398656.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and douard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, Oct 12, 2011. URL https://hal.inria.fr/hal-00650905.

Platt, J. C. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, 1998. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-98-14.pdf.

Potter, S. C., Luciani, A., Eddy, S. R., Park, Y., Lopez, R., and Finn, R. D. Hmmer web server: 2018 update. *Nucleic acids research*, 46(W1):W200–W204, Jul 2, 2018. doi: 10.1093/nar/gky448. URL https://www.ncbi.nlm.nih.gov/pubmed/29905871.

Sarac, O. S., zge Grsoy-Yzgll, Cetin-Atalay, R., and Atalay, V. Subsequence-based feature map for protein function classification. *Computational Biology and Chemistry*, 32(2):122–130, 2008. doi: 10.1016/j.compbiolchem.2007.11.004. URL https://www.sciencedirect.com/science/article/pii/S1476927107001491.

Schapire, R. E. The strength of weak learnability. *Machine Learning*, 5(2):197–227, Jun 1990. doi: 10.1007/BF00116037.

Schmid, R. and Blaxter, M. L. annot8r: Go, ec and kegg annotation of est datasets. *BMC bioinformatics*, 9(1):180, Apr 9, 2008. doi: 10.1186/1471-2105-9-180. URL https://www.ncbi.nlm.nih.gov/pubmed/18400082.

The Gene Ontology Consortium. The gene ontology resource: 20 years and still going strong, Jan 1, 2019. URL https://academic.oup.com/nar/article/47/D1/D330/5160994.

The Uniprot Consortium. Uniprot: a worldwide hub of protein knowledge. *Nucleic Acids Research*, 47(D1):D506–D515, Jan 8, 2019. doi: 10.1093/nar/gky1049. URL https://www.ncbi.nlm.nih.gov/pubmed/30395287.

Tin, K. H. Random decision forests, 1995. ID: TN_ieee_s598994.

Wang, Y.-C., Wang, Y., Yang, Z.-X., and Deng, N.-Y. Support vector machine prediction of enzyme function with conjoint triad feature and hierarchical context. *BMC systems biology*, 5 Suppl 1(Suppl 1):S6, Jun 20, 2011. doi: 10.1186/1752-0509-5-S1-S6. URL https://www.ncbi.nlm.nih.gov/pubmed/21689481.

Webb, E. C. *Enzyme Nomenclature*. Acad. Press, San Diego [u.a.], rev. of the recommendations (1984) of the nomenclature committee of iub edition, 1992. ISBN 9780122271649.

Appendices

A Annotation

`www.github.com/michaelkubiak/annotation/blob/master/annotate`

A.1 Models

A.1.1 Naive Predictor

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Classifiers/random_redistribution.py`

A.1.2 Random Forest

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Classifiers/forest.py`

A.1.3 Support Vector Machine

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Regressors/svm.py`

A.1.4 Neural Network

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Regressors/tf_neural_network.py`

A.2 Testing

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Classifiers/test_harness.py`

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Regressors/test_harness.py`

B Data Preparation

B.1 Download Files

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/get_data`

B.2 Preprocessing

B.2.1 Parsing

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/parse_results.py`

B.2.2 Identification

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/identify.py`

B.2.3 Preparation

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Classifiers/prep.py`

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Regressors/prep.py`

B.2.4 Test Data

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Classifiers/test_data.py`

`www.github.com/michaelkubiak/annotation/tree/master/Scripts/Regressors/test_data.py`

C Results

C.1 Classification

C.1.1 Random Arrangement

Dataset (proteinsxHMMs)	EC Group	Accuracy% (σ)	Sensitivity% (σ)	Specificity% (σ)	Precision% (σ)
Test (996x101)	All	0.56 (1.18)	0.65 (1.45)	99.55 (0.19)	0.65 (1.45)
	1.x.x.x	0.74 (1.14)	0.80 (1.38)	99.60 (0.05)	0.80 (1.38)
	2.x.x.x	1.13 (0.88)	1.50 (1.53)	99.56 (0.03)	1.50 (1.53)
	3.x.x.x	0.73 (0.82)	0.80 (0.89)	99.56 (0.03)	0.80 (0.89)
	4.x.x.x	0.32 (0.97)	0.66 (2.00)	99.71 (0.08)	0.67 (2.00)
	5.x.x.x	0.00 (0.00)	0.00 (0.00)	99.52 (0.14)	0.00 (0.00)
	6.x.x.x	0.26 (0.77)	0.13 (0.40)	99.67 (0.05)	0.13 (0.40)
	7.x.x.x	0.71 (2.14)	0.66 (2.00)	99.19 (0.21)	0.66 (2.00)
Full (560,000x17,929)	All	0.10 (0.05)	0.03 (0.03)	99.98 (0.01)	0.03 (0.03)
	1.x.x.x	0.05 (0.01)	0.01 (0.01)	99.99 (2.20)	0.01 (0.01)
	2.x.x.x	0.12 (0.02)	0.03 (0.01)	99.98 (0.00)	0.03 (0.01)
	3.x.x.x	0.09 (0.01)	0.02 (0.00)	99.98 (0.00)	0.02 (0.00)
	4.x.x.x	0.07 (0.02)	0.02 (0.01)	99.99 (0.00)	0.02 (0.01)
	5.x.x.x	0.08 (0.03)	0.03 (0.02)	99.98 (0.00)	0.03 (0.02)
	6.x.x.x	0.11 (0.03)	0.07 (0.03)	99.95 (0.00)	0.07 (0.03)
	7.x.x.x	0.18 (0.07)	0.04 (0.03)	99.98 (0.00)	0.04 (0.03)

C.1.2 Random Forest (Test Dataset)

E-value cut-off	EC Group	Accuracy% (σ)	Sensitivity% (σ)	Specificity% (σ)	Precision% (σ)
10	All	41.43 (12.30)	47.62 (14.17)	99.81 (0.10)	56.18 (13.50)
	1.x.x.x	49.62 (5.78)	50.13 (8.14)	99.87 (0.03)	62.92 (5.17)
	2.x.x.x	44.21 (4.34)	46.77 (3.97)	99.82 (0.05)	53.89 (7.27)
	3.x.x.x	48.83 (6.28)	52.71 (4.90)	99.85 (0.03)	60.47 (7.72)
	4.x.x.x	26.59 (15.22)	27.52 (10.70)	99.86 (0.07)	40.25 (15.89)
	5.x.x.x	31.37 (10.36)	47.75 (15.56)	99.74 (0.11)	55.09 (15.02)
	6.x.x.x	42.84 (6.55)	5.84 (6.13)	99.86 (0.05)	54.97 (10.04)
	7.x.x.x	46.55 (10.04)	57.64 (18.89)	99.69 (0.12)	65.69 (11.92)
0.1	All	41.43 (12.30)	47.62 (14.17)	99.81 (0.10)	56.18 (13.50)
	1.x.x.x	49.62 (5.78)	50.13 (8.14)	99.87 (0.03)	62.92 (5.17)
	2.x.x.x	44.21 (4.34)	46.77 (3.97)	99.82 (0.05)	53.89 (7.27)
	3.x.x.x	48.83 (6.28)	52.71 (4.90)	99.85 (0.03)	60.47 (7.72)
	4.x.x.x	26.59 (15.22)	27.52 (10.70)	99.86 (0.07)	40.25 (15.89)
	5.x.x.x	31.37 (10.36)	47.75 (15.56)	99.74 (0.11)	55.09 (15.02)
	6.x.x.x	42.84 (6.55)	50.84 (6.13)	99.86 (0.05)	54.97 (10.04)
	7.x.x.x	46.55 (10.04)	57.64 (18.89)	99.69 (0.12)	65.69 (11.92)

C.2 Regressors - Neural Networks (Full Dataset)

Number of Hidden Layers	Number of Nodes per Hidden Layer	Accuracy%
5	1000	57.0
5	4000	57.3
5	8000	57.1
5	12000	57.5
10	8000	53.6