Protein Family Prediction

Tasfiq Ahmed and Josh Kim, (MATH 382)

December 7, 2022

Abstract

Machine learning has rapidly transformed the computational biology space, especially in identifying protein shape. Given DeepMind's recent success in predicting proteins using AlphaFold, replicating some aspects of their workflow is beneficial to better understanding the role of statistical machine learning in modern medicine. In this study, we compare three models and how they perform at classifying the 7 most frequent protein families based on sequence and additional predictors. To apply basic natural language processing (NLP) techniques, features such as amino acid counts or unigrams were developed from amino acid sequence. Then, a Naive Bayes, KNN, and LSTM model were fit. The LSTM had the best performance of all models, with a test accuracy of 60.8%.

Introduction

Misfolded proteins due to genetic mutations frequently malfunction and cause debilitating diseases such as cancer and Alzheimer's Disease. The protein folding problem aims to address these maladaptive proteins via two questions: 1) Can we predict protein shape and family from sequence and 2) what are the mechanisms behind protein formation that minimize energy expenditure? Identifying shape from sequence enables researchers to understand the mechanisms behind malfunctions and make adjustable incisions accordingly in the genetic code. Accurate predictions of protein shape can provide deeper insight into potential patterns in disease-causing genetic code. Thus, we can further develop targeted therapeutics and precision medicine applications.

Despite its significance, generating and predicting proteins is notoriously problematic. Previous methods in experimental laboratory techniques such as X-ray crystallography, cryo-electron microscopy, and nuclear magnetic resonance (NMR) often require millions of dollars worth of reagents and rigid specifications for proper analysis. As a result, these stiff parameters have significantly restricted any rapid drug-discovery process. Although over 200 million proteins exist, experimental techniques have identified shapes only around 170,000, which are stored in the Protein Data Bank (PDB) (Lewis, 2022). Despite this, recent computational advancements in machine learning and algorithmic methods have reshaped the field. During the bi-annual protein folding competition meet, Google's DeepMind developed AlphaFold and successfully predicted 3D protein shapes with extremely high accuracy.

AlphaFold's prediction success is credited to its use of neural networks, a branch of deep learning methods. AlphaFold uses residual neural networks, a form of artificial neural network, that focuses on skipping network layers to optimize efficiency while minimizing training error. In this paper, we explore the use of long

short-term (LSTM) recurrent neural networks, an alternative neural network that also jumps over these connections. We focus on predicting the protein family as a proxy for protein shape. In addition, we implement regression models such as KNN and Naive Bayes to predict for classification. Implementing these algorithmic tools provides insight and first-hand experience into how machine learning is reshaping the computational biology space to answer some of the field's most pressing issues.

Methods

The PDB repository contains protein features including the atomic coordinates of amino acid sequence, protein family, and other critical features added by structural biologists. We use a Kaggle dataset which contains two files derived from the PDB (Shahir, 2017). One file contains protein meta-data and includes protein classification labels and experimental methods used to derive proteins. The other contains over 400,000 protein sequences. Some key features to note include amino acid sequence, protein family classification labels such as "hydrolase", and features collected from experimental methods such as solution density, molecular weight, crystallization temperature, densityMatthews, and pH.

We first preprocessed our class labels and any redundant sequences. We merge only on the proteins that have sequences associated with it. Considering we aim to examine the predictive power of the amino acid sequence, having the sequence is vital. The merged dataset originally contained over 4000 class labels. Much of this count could be attributed to the lack of uniform casing, multiple assigned labels, as well as extremely unique labels or functions. To address multiple labeling, we note the formatting of such proteins and split the label on various forms of punctuation. We then assign the first most label as the final label for the protein. After this pre-processing, there were still 1754 unique classes. For the scope of this project, we have filtered for the seven most frequent labels. To justify this number, we observe that there is a significant decrease in frequency between the 7th most frequent and 8th most frequent label. Furthermore, many of the labels after this point are repeated or are very similar. Thus, we preserve much of the size of our original dataset.

Some labels were assigned to huge proteins with multiple chains. All these chain observations contained repeated amino acid sequences. To minimize class imbalance, we eliminated all but one of these chains for each of these giant proteins. In addition, a lot of our observations contained NA values for some of our predictors. After visualizing our NA values using the MICE package and exploring our dataset, we identified that pH, densityMatthews, and densitySolution contained the most missing values. After generating a histogram for each of these features, most followed a relatively normal distribution. We explored two different approaches to addressing the issue of missing values. One approach was to simply fill in missing values with column means. Since this method is simple and computationally inexpensive, it was a reasonable avenue to explore especially since we planned to standardize our data afterwards. Another approach is to impute the missing values using predicted mean matching, which estimates the likely values of missing data by matching to the observed data. We explore both of these methods in our results.

Given the codified nature of these sequences, we explored basic NLP techniques to potentially improve our prediction power. Specifically, we attempted unigrams or a bag-of-words model approach by counting the number of each amino acid for each amino acid sequence. Bag-of-words essentially captures amino acid counts for protein family classification without considering the order of the sequence. This is a simplistic approach to NLP, as no order in which the amino acids appear is accounted for, but we observed a marked increase in performance in our Naive Bayes and KNN models when including these features.

Despite a plethora of possible models, we limited our selection to three models: a Naive Bayes model, KNN classification model, and a LSTM neural network. We implemented Naive Bayes as our baseline multi-class prediction model, given that our features are relatively independent. We initially implemented a KNN classification model using the caret package; however, given the size of our dataset, R frequently crashed due to memory and RAM issues. As a result, we used the class package to prevent this situation.

The LSTM is a subset of recurrent neural networks that handles text analysis. The model required text-processing of our sequences before model-fitting. First, we encoded all twenty amino acids into integers from 1 to 20. All other amino acids or unidentifiable ones are encoded as 0. Then, we padded sequences to ensure that all of our sequences maintained the same amino acid length. In this case, we set our sequence length to match the longest amino acid sequence and added zeros to all other shorter sequences. Next, we performed one-hot encoding which binarizes our categorical class labels and finished pre-processing.

To build an LSTM, we imported the keras package and Python's tensorflow into our virtual environment using the reticulate R package. We first started with an embedded layer, added an neural network layer, and then flattened the layer to adjust our input dimension for the neural network to properly run. Afterwards, we added a dense layer to better produce an output. Afterwards, we ran the model for 10 epochs.

Results

We initially filled the missing values with column means. Although this improved accuracy of the KNN model by about 2%, it introduced a great deal of noise to Naive Bayes, and performance tanked. As a result, we abandoned this approach. Instead, we first ran our models after dropping all NA values and then re-ran them after imputing our data using the MICE package. For both of these approaches, the accuracies and F1 scores are listed in the Figure 1 below. Given time restrictions and computational considerations, our LSTM model was not run on the MICE pre-processed dataframe. For the LSTM, we split our dataframe into a training, test, and validation set. Our training set is composed of 80% of our observations while the other two sets had 10% respectively. The LSTM had an accuracy of 70.08%, 60.8%, 59.8% for the training, test, and validation set respectively. The LSTM training and loss graphs are listed in Figure 3.

The confusion matrices or heat-maps for these models for each method are listed in Figure 2 below.

Models <chr>	NA_Removed_Accuracy <dbl>	NA_Removed_F1 <dbl>	MICE_Accuracy <dbl>	MICE_F1 <dbl>
Naive Bayes	0.2571383	0.1064709	0.1810389	0.2120429
KNN	0.4781511	0.5177733	0.5779694	0.6475641

Figure 1: Naive Bayes and KNN Accuracy and F1 Scores

Discussion

As can be seen from Figure 1, Naive Bayes performed fairly poorly on this problem in general. When missing values were removed, Naive Bayes had a higher accuracy, but when fitting Naive Bayes on the MICE data, it had a much higher F1-score - in fact, it had a higher F1-score than accuracy. Naive Bayes' poor performance was to be expected. We generally expect a simple model would not predict well for such a complex problem, but it serves as a base-line. Evidently, the class imbalance was a major problem. Naive Bayes' high F1-score

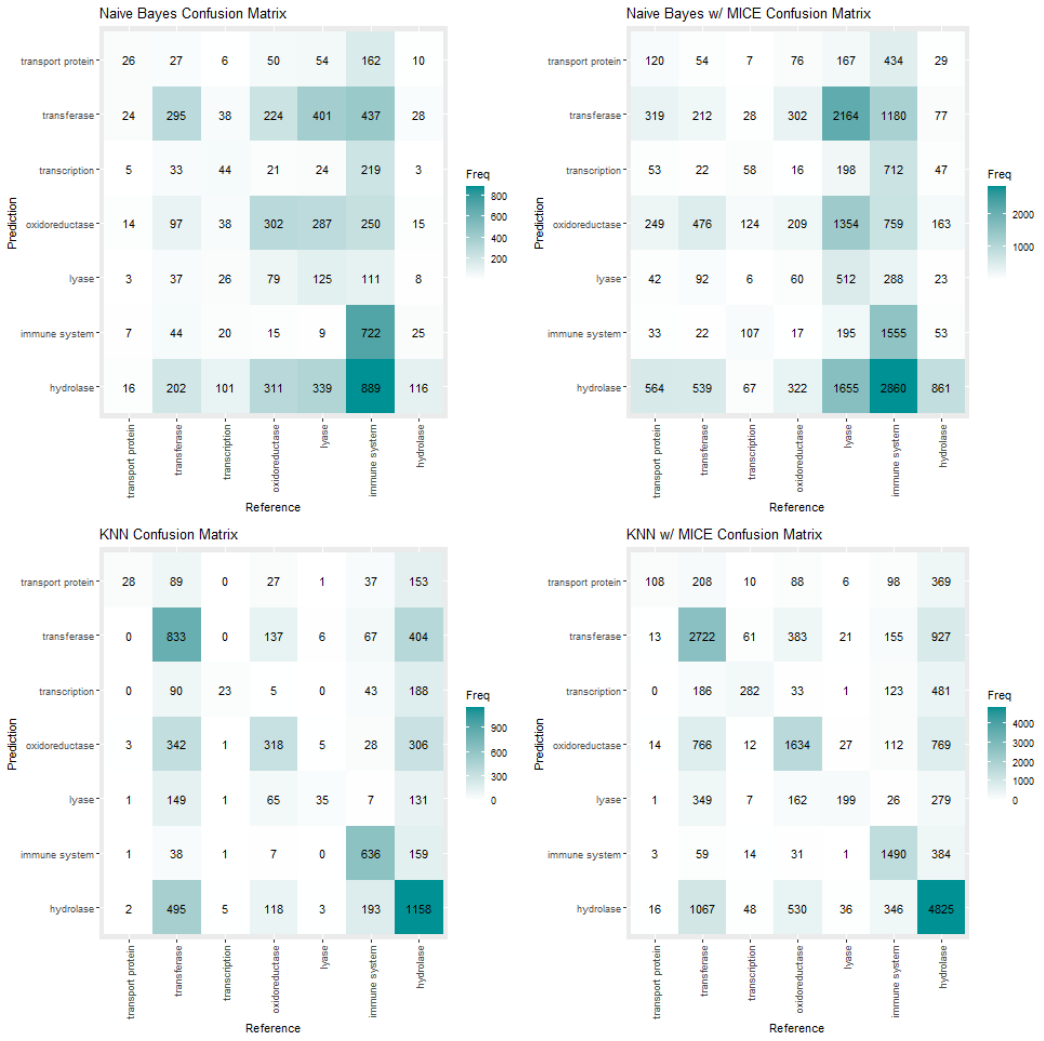

Figure 2: Naive Bayes and KNN Confusion Matrices

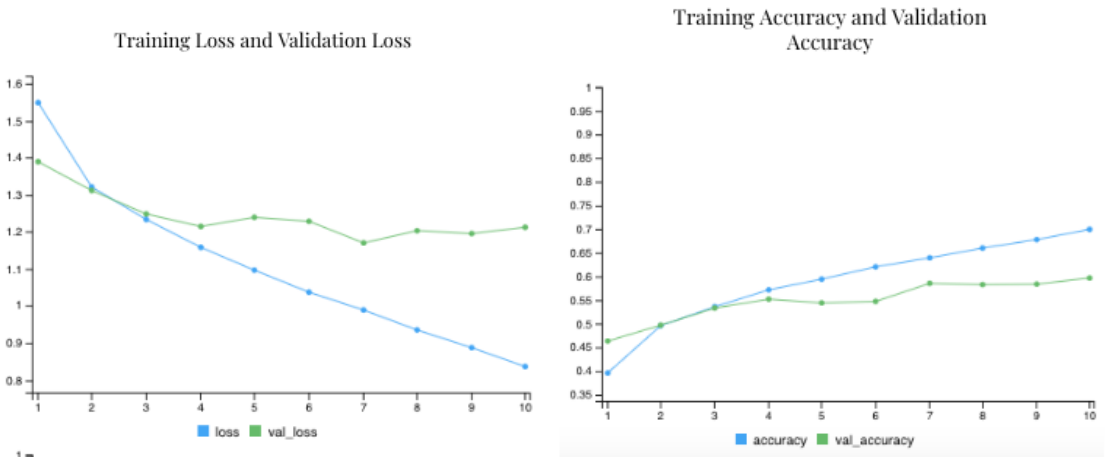

Figure 3: LSTM Training and Loss

relative to accuracy on the MICE data signifies that there was a better balance of precision and recall. Rather than simply predicting the majority class every time, it is actually predicting the class based on the features. However, on the dataset with missing values removed, it simply guessed the most frequent class the majority of the time. Due to the class imbalances, it had a higher accuracy, but the F1-score plummeted.

KNN saw a marked improvement in performance with the MICE imputed data. It saw a 10% increase in accuracy, and a 13% increase in the F1-score. In this case, it seemed that having the extra data improved KNN performance. When building the model, we increased our dimensionality by 21 (one for each amino acid count) and this was initially a cause for concern. However, KNN had far exceeded our expectations.

Based on Figure 3, the LSTM validation loss is significantly above the training loss at the 10th epoch. This suggests that the model is under-fitting due to limited data size for the neural network.

Conclusions

While our models overall had decent prediction power, we can implement future changes to potentially improve performance. To improve our Naive Bayes model, including a way to deal with class imbalances via oversampling or undersampling (SMOTE package) may improve our accuracy and F1 score. In addition, the Naive Bayes model strangely predicted "immune system" as one of the most frequent class predictions. However, why this occurs is yet to be understood. For LSTM, one obvious improvement would be to run the model using the MICE imputed data. To compare with LSTM, we can also implement a simplified residual neural network similar to AlphaFold to compare with our LSTM results.

Furthermore, we can implement more advanced NLP techniques. Our naive approach of approximating a bag-of-words model using amino acid counts evidently helped our models. However, we would like to extend this with bi-grams, and potentially tri-grams. With bi-grams and tri-grams, we can incorporate ordering of the amino acid sequence as well, which is vital to protein function, and consequently, protein classification. Additionally, an even more advanced approach could be taken by mapping sequences to vector embeddings that can encode structural information, as done in (Bepler & Berger, 2019).

In this study, we chose to predict for protein family as a proxy for protein shape. However, running the models to predict atomic coordinates may be a better proxy to explore.

Overall, we're pleased with the results and hope to delve into deeper analysis using more advanced machine learning techniques.

References

Bepler, T & Berger, B. (2019, October 16.) Learning Protein Sequence Embeddings using Information from Structure. International Conference on Learning Representations (ICLR). Retrieved December 6, 2022 from https://paperswithcode.com/paper/learning-protein-sequence-embeddings-using

Lewis, T. (2022, October 31). One of the biggest problems in biology has finally been solved. Scientific American. Retrieved December 6, 2022, from https://www.scientificamerican.com/article/one-of-the-biggest-problems-in-biology-has-finally-been-solved/

Shahir (2017). Structural Protein Sequences. https://www.kaggle.com/datasets/shahir/protein-data-set