Kaggle Protein Stability Prediction

Before going into strictly my thought process behind my approach to this problem, I think it is important to reflect on a previous experience I had that led me to my approach. In my research lab, I was given a list of chemicals in SMILES format as well as binary hits on these chemicals, and I was told to create a model that could predict binary hits on unalbeled SMILES string. Not only had I never worked with this kind of data, but I also had very little chemistry background. To make it short, it was through this experience that I learned that it is often better to conduct EDA and background readings on the data in order to get a strong idea of how to approach a computing problem, and this is exactly how I approached this thermosatbility predictions problem.

Amongst first seeing the data, I knew I had to do some sorry of feature extraction as creating a model based on the raw day would be extremely difficult. While looking over API and documentation to convert the amino acid sequences into descriptors, I needed to consider ease of use and of course how relevant the features from these APIs were to thermostability. In order to figure out the features relevant to thermostability, I Googled and read many research papers. Some notable features I found relating to thermostabilty were amino acid composition (i.e. % of charged amino acids, % of hydrophobic amino acids, % of polar amino acids), solvency, thermostability index, ionic interactions, and many more. I came to the conclusion that PyBioMed was best to start out using due to its ease of use and useful descriptors, and if I had extra time at the end, I would consider pulling different descriptors from different software to try to find the best combination of dsecriptors. After looking over the API for a bit, I decided to use the CTD function which outputs 20+ dsecriptors. While these descriptors were all useful, I believed there were too many features which could potentially lead to over-fitting. The output features were NormalizedVDWV, Polarizability, Charge, SecondaryStr, SolventAccessibility, Hydrophobicity, and Polarity; all of which were represented 3 time a piece with a trailing C1, C2, and C3; each with a different representation. The C1 stands for a summary of the overall sequence, C2 for transition descriptor, and C3 for distribution descriptor. I used all the C1 values as my only descriptors, giving me a manageable 7 total features.

With feature extraction completed, I next needed to decide which model to start out with. Although there were some simpler regression packages I could have used, I decided to start out with a random forest due to its popularity with regression problems. Random Forests are capable of predicting from non linear data, are not too prone to overfitting, and have outlier detection abilities, all of which were useful to uitilize on the extracted features. Also, my familiarity with random forests played a role in the selection as I first wanted to get my own baseline result.

The next task was to decide upon the hyper parameters to use for the model. While I am experienced with using Random Forest models, hand choosing hyperparameters is both not a smart thing to do and a difficult task. In order to choose my hyperparameters, I used a method called GridSearchCV from the sklearn package. GridSearch essentially works by testing all possible combinations of a given list of hyperparameters. GridSearchCV then evaluates the combinations by using cross-validation where the training data is split into mudltiple subsets where the model is trained on some of the subsets and evaluated with the others. In order to pick the hyperparamters to actually test, I was unsure, so I utilized an LLM (in this case I used Claude) and picked some out myself. Claude gave me 3 different numbers to test for each hyper parameter, consisting of 'n_estimators' (# of trees in the forest), 'max_depth' (max depth of each tree), 'min_samples_split' (minimum number of samples to split an internal node; on a side note a higer value here can help prevent overfitting), and 'min_sample_leaf' (minimum number of samples required to be a leaf node; higher number can also prevent overfitting). Upon running the GridSearch, the optimal results were: n_estimators: 200, max_depth: 10, min_samples_leaf: 2, min_samples_split: 10. I used these, but later on I attempted to further explore with more extreme values of which gave a worse Spearman correlation. In terms of picking the "perfect hyper parameters" I would not say I did so, but I do believe I sampled a wide enough range and variety to have very good hyperparameters.

After fitting the hyperparameters to my tree and training the model, the resulting Spearman correlation was 0.49. This was not terrible, but I still wanted to improve the correlation. To do this, I next tried out a gradient boosting model. My reasoning for attempting gradient boosting is its ability to capture more complex relationships better than a random forest. The biggest difference in the algorithms is that a random forest builds its trees by creating a multitude of them during training time whereas gradient boosting builds its trees sequentially, improving off of the previous tree. One thing to note is that gradient boosting is more prone to overftting due to the process mentioned above, but I still wanted to give it a try given its similarities to random forest. The model resulted in a slightly improved Spearman correlation of 0.5.

I would say my greatest takeaway from this extremely educational and fun learning experience working on the Kaggle assignment is that the most important part of getting good results for a machine learning model is the data pre-processing. In my other machine learning class, there is a saying "garbage in, garbage out" and I witnessed it first hand here. I trust that I used applicable models for the assignment, and although my results were not bad, I fealt they could have been significantly improved if I had been able to perform slightly better feature extraction. This likely would have been through the form of running several different descriptor extracting packages, and then concatenating important results for each data entry. It was also extremely interesting to learn more about protein thrmostability, and I believe this Kaggle assignment gave me the confidence to work with any dataset. Going into the assignment, I had no knowledge on protein thermostability, but I quickly gained enough information to find a software that could extract useful information from the raw data that I found relevant. In the future and in work life, I will spend more time on data pre-processing/feature extraction, and hopefully I can improve my results.