**I. Description of the design of the prediction system**

Features

Throughout our machine learning pipeline, there are many features that we generated. During our first design, we generated 500 features from each sequence object where each of the first 500 encoded (with a scale from 0-1) amino acids is a feature. If an object in our dataset consisted of a sequence with a length less than 500, we filled the encoded sequence with 0's until we reached a length of 500 for each object. If the sequence had a length of over 500, we removed all of the encoded amino acids after the 499th index.

We implemented that group of features with best feature selection using feature_importances_ in RandomForest, but realized that these features resulted in overfitting to our training data and gave us a very high accuracy as seen in Design 1 in Figure C, so we did not use it. Also, we generated sequence length (total length of each amino acid sequence), unique_val, and unique_val_count. Unique_val generates a list of the unique amino acids in each sequence, and unique_val_count is the count of unique amino acids from each sequence. We chose not to use unique_val, unique_val_count, and length in our final designs, because in all of our design implementations, they tended to make the MCC and performance worse and logically, we did not think they would be valuable features.

Of the features we used in our later designs, aa_count_freq is an array of the frequency (between 0 and 1) of each amino acid within the sequence that is listed in alphabetical order of amino acid name. It generates 20 features for our model. Two-gram and three-gram features were additional valuable features that we implemented and improved our models' performance, but we chose to use two-gram only because the three-gram feature took notably more time and generated way too many features. In two-gram, of our 8000 objects, we look at all the potential pairs for each amino acid in pairs of two (ex. XA, XC, XD, …, YA, YC, YD). For each two letters in each sequence, we mapped it to the corresponding pair (ex. Sequence: XAYAYDXA → {'XA': 2, 'XC': 0, 'XD': 0, …, 'YA': 1, 'YC': 0, 'YD': 1}. Lastly, we implemented the standalone pfeature package, specifically binary and composition. While we thought that binary could give us a very good MCC, it took a lot of time, so we ended up using composition (specifically the PAAC and SER parameters). We tried other composition features, but those either gave us a low MCC or took a long time to run. The pfeature composition is valuable because it is able to calculate more than 70,000 composition features from the primary sequence of protein and is also a popular choice for amino acid sequencing and feature generation in academia. Throughout our progress, we tried different combinations of features (including length, aa_count_freq, two-gram, and different pfeature comp options) and classification methods by trial and error. By design 3 and 4, we settled on aa_count_freq, two-gram, and pfeature composition SER, which generally performed the best with multiple classifiers.

Classification/Other Designs Choices

During the implementation and testing of different features, we used exclusively the Random Forest classifier from sklearn ensemble modules with the default parameters. In order to speed up the computation of our model, we selected a subset of our 63 input features (excluding the 500 encoded features that were removed for overfitting) to remove weaker or noisy features by comparing average MCC values for different feature combinations and using feature_importances_ in RandomForest.

Once we settled on a list of possible features (including aa_count_freq, pfeature, and two-gram) to work within Design 2, we implemented all of the sklearn ensemble classifiers with their default parameters to determine which yielded the best average MCC. This allowed us to then focus on parameterization for the top three performing classifiers, including ExtraTreesClassifers, RandomForestClassifer, HistGradientBoostingClassifier. In addition to the ensemble classifiers, we implemented sklearn's neural network, specifically the multilayer perceptrons classifier (MLPClassifier), due to its flexibility and popularity with tabular datasets and in classification prediction problems.

With four options for classifiers, we implement GridSearchCV to conduct an exhaustive search for the best parameters of each classifier. With each iteration of GridSearchCV, we tested between 2 to 3 values for each of the parameters of each classifier listed in Figure A and used the "best parameter" output of GridSearchCV to then narrow our parameter options and repeat until we settled on a final best parameter value (as seen in Figure A). We ran the newly parameterized classifiers with different combinations of features until we found that HistGradientBoostingClassifier and ExtraTreesClassifer with pfeature, two-gram, and aa_count_freq produced the best performance of all of the classifiers, producing Design 3. In our machine learning pipeline, due to the class imbalance of the training dataset, we also incorporated oversampling with SMOTE to generate synthetic data for our model. While we did not end up incorporating it in our final design, we tested out different options for best feature selection, including the sklearn SelectFromModel meta-transformer that we incorporated in Design 3. The SelectFromModel allowed us to couple multiple classifiers in a single design by using one to pick the best features in our training dataset and a second to train on the subset of features and make predictions, which produced better results than using a single classifier in some of our earlier testing. However, for our final design, we noticed that of all of our implementations, the MCC performed the best with HistGradientBoostingClassifier as a single classifier than any SelectFromModel combination we implemented (with MLPClassifier as a close second).

*Figure A: GridSearchCV Parameterization*

| *MLP Classifer* | *ExtraTrees* | *RandomForest* | *HistGradientBoost* |
|---|---|---|---|
| random_state = 6,<br>activation='relu',<br>alpha=0.05,<br>learning_rate='constant',<br>solver='adam' | random_state=6,<br>n_estimators=165,<br>max_features='sqrt',<br>max_depth=19,<br>criterion='gini' | random_state = 6,<br>n_estimators=650,<br>max_features='sqrt',<br>max_depth=13,<br>criterion='entropy' | random_state = 6,<br>learning_rate=0.1,<br>loss='categorical_crossentropy',<br>max_depth=12,<br>max_iter=300 |

*Figure B: Design Implementation Details*

| | **Design 1** | **Design 2** | **Design 3** | **Design 4** |
|---|---|---|---|---|
| **Features** | Two-gram, composition pfeature with PAAC, aa_count_freq, length | Two-gram, composition pfeature with SER, aa_count_freq, length | Two-gram, composition pfeature with SER, aa_count_freq, length | Two-gram, pfeature with SER, aa_count_freq |

| | | | | |
|---|---|---|---|---|
| **Model** | RandomForest classifier with no parameterization | RandomForest classifier with no parameterization | Feature selection (i.e. SelectFromModel) using ExtraTreesClassifer with best parameters and classification using RandomForestClassifier with best parameters | HistGradientBoostingClassifier with best parameters |

**II. Results**

Cross Validation Results

By modifying our model through the various designs to the most current (as described in Figure B), we were able to achieve slightly higher accuracies and overall a much better average MCC value, as seen in Figure C. Design 1 had very high performance metrics across the board compared to the other designs, which was caused by severe overfitting when we exclusively selected the most important features from the 500 encoded amino acid features to train our model. It is important to note that overfitting was the biggest factor in the process of building the machine learning model in regards to significant redesign choices. As a result, we switched our features entirely from the "encoding the first 500 amino acids of a sequence" method to the "two-gram" method. Originally, we got results that seemed "too good to be true" with the first method, and assumed that our model fit almost perfectly to the training data but could perform worse on a future dataset.

Excluding design 1, we saw several patterns throughout our predictive results for Designs 2, 3, and 4, which used only two-gram, composition pfeature, aa_count_freq, and length (in Design 2 and 3 only). Generally, our three designs performed pretty similar to one another, all performing better than the baseline, which we discuss at more depth in the conclusion. The biggest difference between our different designs was that Design 2 and Design 4 generally performed better than Design 3 for DNA, RNA, and nonDRNA, but Design 2 had a notably lower MCC than both Design 3 and 4 for DRNA. Similarly, the sensitivity performs slightly differently between the three designs for all classes, except for DRNA. That being said, our sensitivity was notably lower for DNA and DRNA (the smallest classes) across the board.

Blind Test Set Results

Our model seemed to have performed well just based on the count of the actual labels in the training dataset and the count of the predicted labels in the blind test dataset (as seen in Figure E). We are coming to this conclusion that it performed well because the proportion of each of the classes is the same in both the training set and the predicted testing set. A minor red flag is that our model leans towards classifying more sequences as nonDRNA than what might be expected. Overall though, we received results that we expected and on par with our training predictions.

*Figure C: Predictive Results*

| **Outcome** | **Quality Measure** | **Baseline Results** | **Design 1** | **Design 2** | **Design 3** | **Design 4** | **Best Design** |
|---|---|---|---|---|---|---|---|
| DNA | *Sensitivity* | 6.9 | 100.0 | 8.7 | 9.5 | 11.8 | 11.8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | *Specificity* | 99.3 | 100.0 | 99.8 | 99.7 | 99.6 | 99.6 |
| | *Accuracy* | 95.2 | 100.0 | 95.8 | 95.7 | 95.7 | 95.7 |
| | ***MCC*** | 0.132 | 1.0 | 0.233 | 0.224 | 0.25 | 0.25 |
| RNA | *Sensitivity* | 39.6 | 100.0 | 42.3 | 43.2 | 45.9 | 45.9 |
| | *Specificity* | 98.9 | 99.9 | 99.3 | 98.9 | 99.0 | 99.0 |
| | *Accuracy* | 95.3 | 99.9 | 95.9 | 95.6 | 95.8 | 95.8 |
| | ***MCC*** | 0.501 | 0.993 | 0.561 | 0.536 | 0.565 | 0.565 |
| DRNA | *Sensitivity* | 4.5 | 71.4 | 9.1 | 9.1 | 9.1 | 9.1 |
| | *Specificity* | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| | *Accuracy* | 99.7 | 99.9 | 99.7 | 99.8 | 99.8 | 99.8 |
| | ***MCC*** | 0.122 | 0.85 | 0.212 | 0.246 | .246 | .246 |
| nonDRNA | *Sensitivity* | 98.6 | 100.0 | 99.3 | 98.8 | 99.9 | 99.9 |
| | *Specificity* | 29.8 | 100.0 | 29.6 | 31.0 | 33.9 | 33.9 |
| | *Accuracy* | 91.3 | 100.0 | 91.9 | 91.6 | 92.0 | 92.0 |
| | ***MCC*** | 0.428 | 1.0 | 0.467 | 0.452 | .483 | .483 |
| ***averageMCC*** | | 0.296 | 0.96 | 0.368 | 0.364 | .386 | .386 |
| accuracy4labels | | 90.8 | 99.9 | 91.6 | 91.3 | 91.7 | 91.7 |

*Figure D: Confusion Matrix for Best Solution*

| | Predicted: RNA | Predicted: DRNA | Predicted: DNA | Predicted: nonDRNA |
|---|---|---|---|---|
| Actual: RNA | 240 | 0 | 5 | 278 |
| Actual: DRNA | 0 | 2 | 2 | 18 |
| Actual: DNA | 21 | 1 | 46 | 323 |
| Actual: nonDRNA | 61 | 0 | 25 | 7773 |

*Figure E: Counts for Actual Labels in Training Dataset Vs. Predicted Labels in Blind Test Dataset*

| ***Class*** | ***Count for Actual Label in Training Dataset*** | ***Count for Predicted Label in Blind Test Dataset*** |
|---|---|---|
| *DNA* | *391* | *33* |

| | | |
|---|---|---|
| *DRNA* | *22* | *1* |
| *RNA* | *523* | *139* |
| *nonDRNA* | *7859* | *8621* |

**III. Conclusion**

The four quality measures being compared between our various designs and the baseline are sensitivity, specificity, accuracy, and MCC. Sensitivity reflects true positive rate whereas specificity reflects the true negative rate. For example, a high sensitivity would result in less false negatives because there are fewer missing positives. Conversely, a high specificity would mean less false positives due to fewer missing negatives. Accuracy is an important metric because it evaluates classification models by determining the number of correct predictions over the total number of predictions. Lastly, MCC (Matthews Correlation Coefficient), is the most important classification metric because it takes in account all four confusion matrix values (true positives, true negatives, false positives, false negatives) and produces a value ranging between -1 and +1 (+1: perfect model -1: poor model).

Overall, looking at all of the performance metrics in Figure C, our model performed generally better (and sometimes the same) as the baseline results across the board. Generally, nonDRNA had the smallest range for MCC across the different designs, ranging from 0.42 to 0.48, which may be attributed to the fact that the training set was largely imbalanced, so both our model and the baseline had a lot of nonDRNA objects to train on. The MCC improved the most from the baseline to Design 2, 3, and 4 for DNA and DRNA, but notable improvements for RNA as well. The accuracies and specificity for all labels in Designs 2, 3, and 4 are nearly the same. Moreover, across the board, our model had higher sensitivity for all classes, with DNA and DRNA experiencing the most significant increases, meaning there were fewer false negative results than the baseline. On the other hand, the specificity was high for the minority classes (i.e. DNA, RNA, DRNA) but not that different to the baseline. This was not surprising since due to the class imbalance, our model was likely to label minority class objects as nonDRNA and less likely to have false positives for the minority classes.

This is especially important because it shows that our model performed notably better than the baseline results with the smaller classes (DNA and DRNA with 0.12 MCC higher for both) where the model had less data to train on, giving it a key advantage over the baseline. Conversely, the baseline, especially with DNA and DRNA, much more often assigns minority objects to the nonDRNA label. Because of oversampling using SMOTE, we were able to generate more synthetic data for the minority classes for our model to build off of, resulting in improved performance. As a result, the quality of our results is better as compared to the baseline results.

A disadvantage in our model is lack of parameterization and possibility of overfitting. In an attempt to combat this, we used GridSearchCV to finalize parameters but selected 4 to 5 relatively random parameters for each classifier to investigate, meaning there is more room for exploration with parameterization. Any possible overfitting may be attributed to weak/noisy features that were not removed from our model during earlier feature selection. For example, pfeature produced 20 features, but it's possible that only 15 were actually valuable for our model and the remaining 5 negatively impacted our predictions.

Once we finalized our design options for algorithms, parameters, and features, we wanted to keep an open mind and figure out ways on how we could improve our performance metrics. This part of the process was done through trial and error, and we eventually found an algorithm and parameters for the algorithm that we were confident in.

Throughout this project, our team was able to experience the entirety of the knowledge discovery process, which allowed us to apply many of the concepts that we learned in class such as oversampling, classifiers, and parameterization. However, one of the biggest lessons we learned in this project was the importance of understanding the data in order to generate appropriate and good features before classification, all of which is trial and error and took us the most time but resulted in the most significant improvement of our model.