Capstone Project 1 Final Report:
Predicting clinical genetic variants that will have conflicting classifications by clinicians

Gurdeep Sullan

3/18/2020

Background:

Genetic testing is an important part of disease diagnosis and continues to grow in capacity (in terms of number of patients and detected diseases) as gene sequencing technologies develop. Patients can undergo genetic testing for numerous reasons, among which are preemptive screening for inherited diseases and diagnosis confirmation. In the first case, genetic testing is done when the patient has a family history of a disease and would like to know the likelihood of getting it in the future. In the latter case, a doctor sends a DNA sample to test for the presence of a disease-related mutation after the patient reports symptoms of disease. Thus, genetic testing provides results that have serious implications for guiding treatment and healthcare planning.

Once the decision to perform genetic testing is made, patients submit a sample of their DNA for testing against a large panel of known mutations, also known as variants. After their sample is run and variants are found, an in-house clinical geneticist makes the determination of where on a five-step scale the variant lies:

Benign	Likely Benign	VUS	Likely Pathogenic	Pathogenic

Where benign is an indication of no disease, VUS stands for a variant of uncertain significance, and pathogenic means disease is present. Many genetic variants, or mutations, are benign. Thus it is important to note here that 'mutation' is not synonymous with disease.

This classification is based upon a number of factors such as published literature and studies on that particular variant. There are multiple in silico predictors of deleteriousness, or harmfulness, based upon amino acid changes of a variant. However, these predictors are limited to protein-expressing genes and, by themselves, are not reliable for predicting disease. Some labs do use *in silico* predictors in conjunction with other resources when making their classification while others do not. Consequently, the same mutation can be classified as pathogenic by one lab and benign by another.

Problem:

A large problem in genetic testing is that some mutations are consistently classified, while other mutations receive conflicting classifications when tested at different laboratories. This project aims to address the following question: Can we predict whether or not a mutation will have conflicting classifications by two or more labs based upon the features of the mutation?

Client:
The first group of clients for this project are medical professionals who make the decisions on ordering genetic tests as well as communicate the results of a genetic test to a patient. The second client is the patient themselves, who make the final decision to test their DNA for mutations. The resulting model from this project can help both patients and caregivers decide whether or not to send out DNA samples to different labs when testing for a particular mutant. If they know the mutant of interest is likely to have conflicting classifications, it would be worthwhile to get additional labs' opinions before making major treatment or healthcare decisions.
Finally, a third group of clients is the community of genetic testing professionals. This model can reveal trends in mutants that are likely to be conflictingly classified, and may highlight patterns in classification processes that can be standardized.

Data:
The data used for this project comes from a Kaggle competition dataset (https://www.kaggle.com/kevinarvai/clinvar-conflicting), originally taken from ClinVar (https://www.ncbi.nlm.nih.gov/clinvar/). ClinVar is an online repository of genetic mutations and their features, updated in real time by submitters who submit their data to the database. Submitters to ClinVar include academic institutions, genetic testing laboratories, and hospitals.
The Kaggle dataset includes data up until April 7th, 2018. Each of the 65,188 data entries has 46 features. All of the data points are genetic variants that have two or more classifications on the five-step scale outlined above. The 'CLASS' column contains a binary value for each data entry, either 0 = consistent classifications or 1 = conflicting classifications.

Approach:
First, the data will be uploaded in a pandas dataframe and exploratory data analysis will be completed in a Python Jupyter Notebook. The pandas package will be utilized to understand the structure of the dataframe and the data types in it. To visualize the data and identify interesting trends, the matplotlib and seaborn packages will be used.

Second, I will perform data cleaning. This includes identifying and replacing null and NaN values. I will convert columns to either categorical or numerical, with the exception being the column for disease indication. For this column, I will use NLP and tokenization to extract distinguishing keywords for each disease. Additionally, I will convert the necessary columns that are really bins, such as 'CHROM' (chromosome number), to categorical columns.

Thirdly, I will build a classification model to get a first-pass working model to improve upon. The model will take the necessary features as input and output either 0 or 1 for the 'CLASS' feature. I will be creating a hyperparameter table to track the hyperparameters of my model as I optimize it. I will perform MinMax scaling of features to be able to provide a normalized perspective on weights in the final model.

Deliverables:
A key deliverable will be the hyperparameter table, which will track the building of the predictive model. For clients in the genetic testing professional space, normalized weights of features along with descriptive statistics of the dataset will be presented separately. The final project with all of its code and Jupyter Notebooks will be shared on Github. All relevant reports and data will be included in the Github repository.

Data Cleaning:

Basic Strategy
Jupyter Notebook: Data Cleaning Notebook
First, I will load data into pandas dataframes. Then, using .info(), .describe(), .nunique() and .value_counts() I will get some insight into the amount of missing values and distinguish numerical features from categorical features.

For numerical and categorical data, I will fill in missing data where possible due to feature properties and/or definition.

- ➔ Numerical:
  - ◆ I will fill in missing values for 'ORIGIN'. The feature definition states 'Unknown' is equal to a value of zero - therefore NaNs are treated as "unknown" and can be set to 0 using .fillna().
  - ◆ Fill in missing values for 'INTRON' and 'EXON' as zero, since one entry can either be an intron or an exon - it will not be both.
- ➔ Categorical:
  - ◆ I will fill in missing values for 'SIFT' and 'BIOTYPE' columns as 'unknown' because the classification is unknown. Then use the pandas.get_dummies as outlined below.

For categorical (non-numerical data) follow this rule:

- ➔ If the data is categorical and has reasonably few categories, use pandas get_dummies() to engineer new features for each category. This limitation is because I do not want to end up with a dataset that has more columns than rows
- ➔ If the data is categorical and has hundreds/thousands of categories, do not use get_dummies. Set these aside for future processing during model building and optimization. One example of future processing is using NLP to create numerical representation of strings - see section on NLP below.

Natural Language Processing
Jupyter Notebook: Gene Features NLP Notebook
Use TfidfVectorizer from the scikitlearn package to engineer additional features to represent each of the 4 possible nucleotide symbols in the 2 genome sequence. So far this has been used on the features with gene sequences in them, 'REF' and 'ALT', standing for the reference allele and alternate allele. Future work will use NLP for

additional features in the dataset that are non-numerical and have too many categories to realistically use get_dummies.

Uniprot Dataset

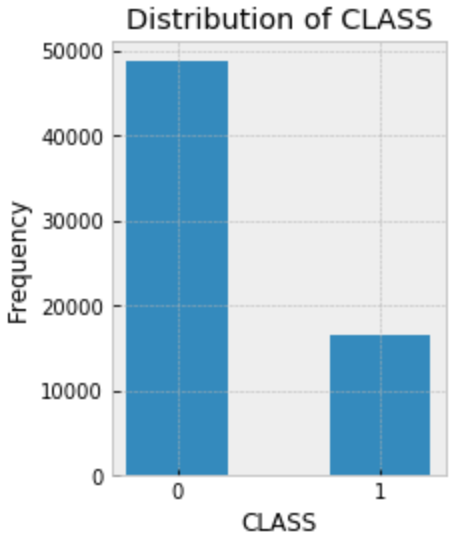
Jupyter Notebook: Uniprot Data Wrangling Notebook

For the Uniprot dataset, I loaded the .csv file into a pandas dataframe, and filtered out unreviewed entries for genes by only retaining 'reviewed' status entries. Then, I grouped by gene symbol and took the mean length in order to have one length per gene symbol.The main dataset was left joined with the "Length" column of the Uniprot dataset using the key of "SYMBOL" (meaning gene symbol). I created a new feature from this column by dividing "Protein_position" by "Length" to get a measure of variant position in the protein, and named it "Relative_position". Wherever length and protein position data was not available, I used .fillna() to fill null values with 0 since 75% of these null values were a result of being intron variants (non-coding regions of the DNA and thus not in the protein).

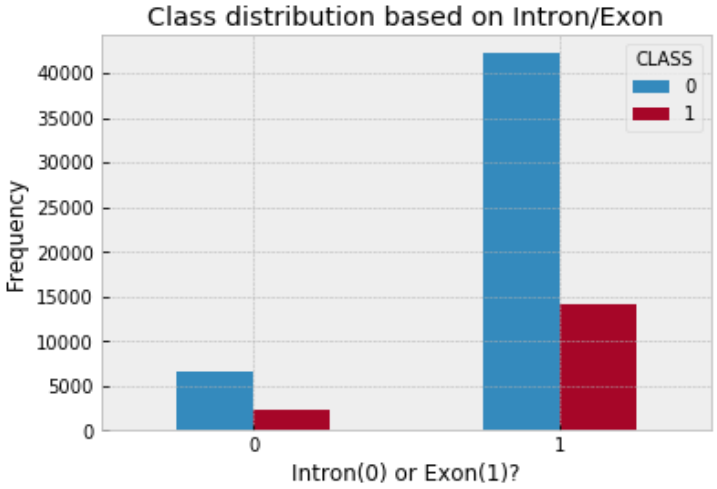
Exploratory Data Analysis:

Jupyter Notebook: Exploratory Data Analysis Notebook

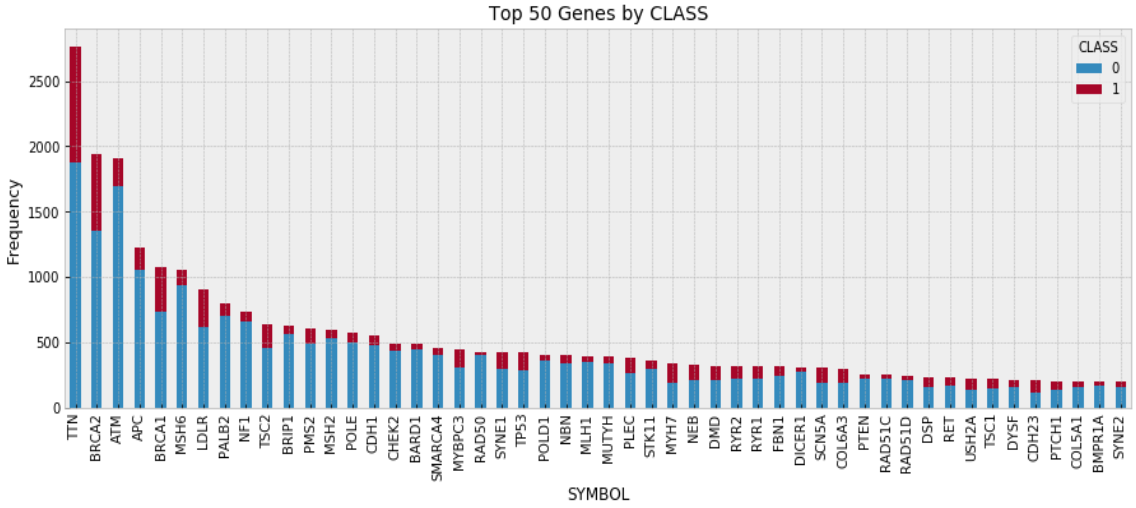
- **Class Imbalance:** It is important to notice that there is a class imbalance in this dataset. Class 1, meaning conflicting classification, is represented at almost ⅓ frequency as Class 0. This follows from intuition of the problem, as it makes sense that professionals tend to be consistent in their classification more often than not. This class imbalance is important to keep in mind for model training and evaluation.



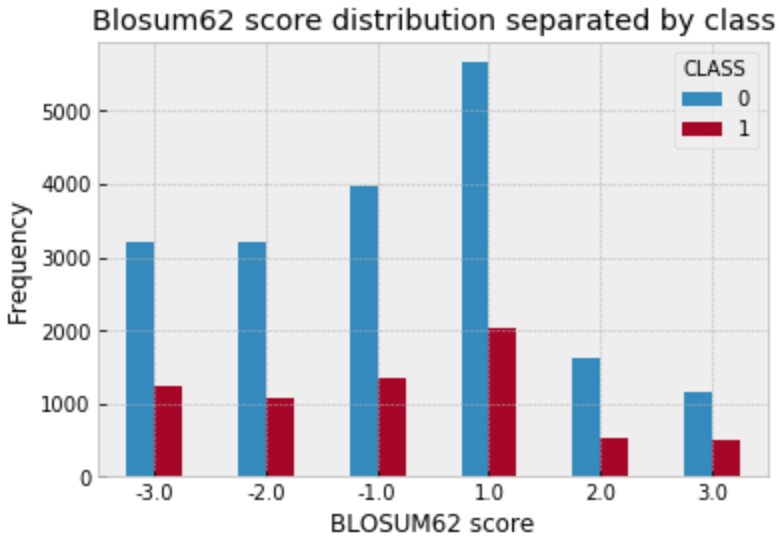
- **Intron vs Exon Variants:** Does it matter if a genetic variant is in the exome or not? In this dataset, most of the entries are exons, which means they occur in expressed regions of the genome. A small percentage are introns, which is the other part of the genome. This makes sense as the exome is more widely studied and understood in literature.



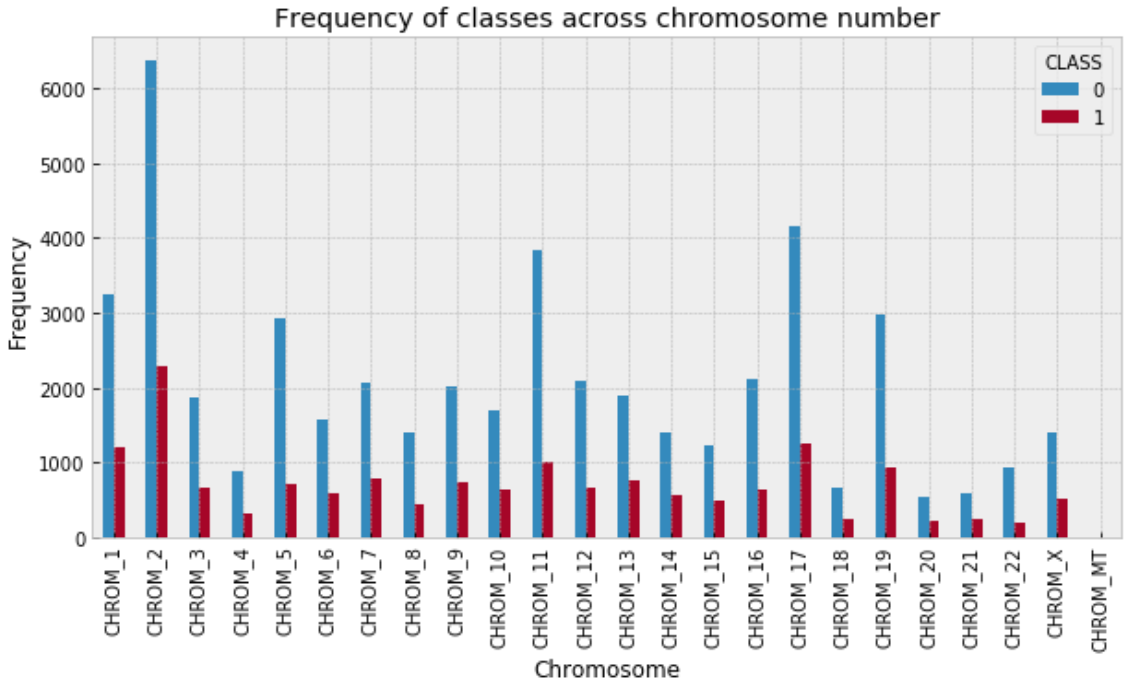
➔ **Gene IDs:** What are the top 50 genes represented in this dataset, and how are different classes represented among those? Not surprisingly, we see widely studied and report genes in the top 50, such as TTN, BRCA1/2, and ATM. This is important to keep in mind because the majority of data for Class 1 is in these first 50 genes. Results of training a model on this data will be skewed towards these top genes that tend to be more well studied and have more literature on them.



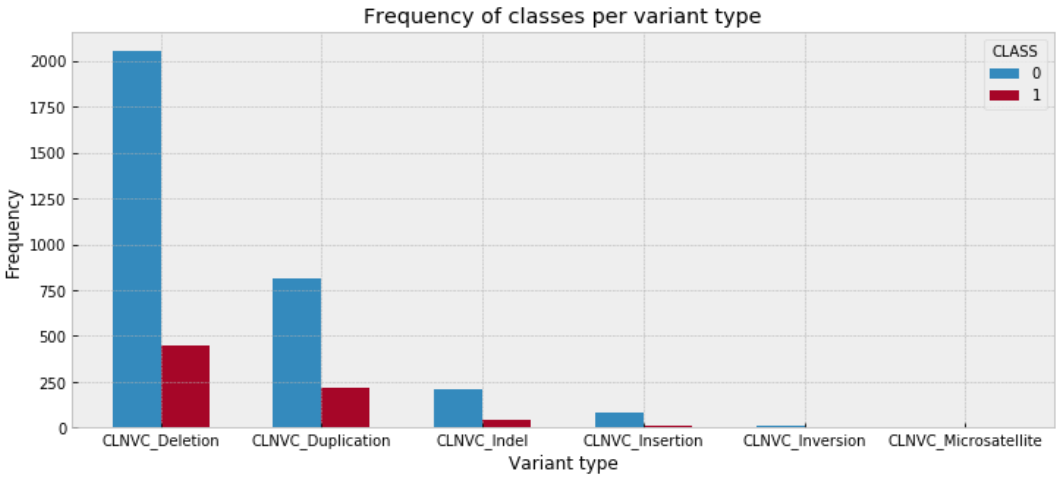
➔ **BLOSUM62 Alignment Score:** The BLOSUM62 score is an alignment score of of the reference and alternate allele transcripts. A higher more positive score indicates a better alignment and thus less of a difference in the resultant protein from this genetic variant. The graph below does not reveal a significant trend between the BLOSUM score and the class.



➔ **Chromosome Number:** Are genetic variants on certain chromosomes more conflictingly classified than others? The plot below shows the frequencies of both classes across all the chromosomes. We can see that Chromosomes 1, 2, 11, 17, and 19 are the most heavily represented in this dataset. They also tend to have relatively more consistent classifications (class 0) compared to conflicting ones (class 1).



➔ **Variant Type:** Do certain types of variants tend to have more conflicting classifications than others? While that question couldn't be clearly answered from EDA, it is addressed in further detail during statistical data analysis. It is evident that Deletions make up a very large portion of our data.



Statistical Data Analysis:

Jupyter Notebook: Statistical Data Analysis Notebook

Background:

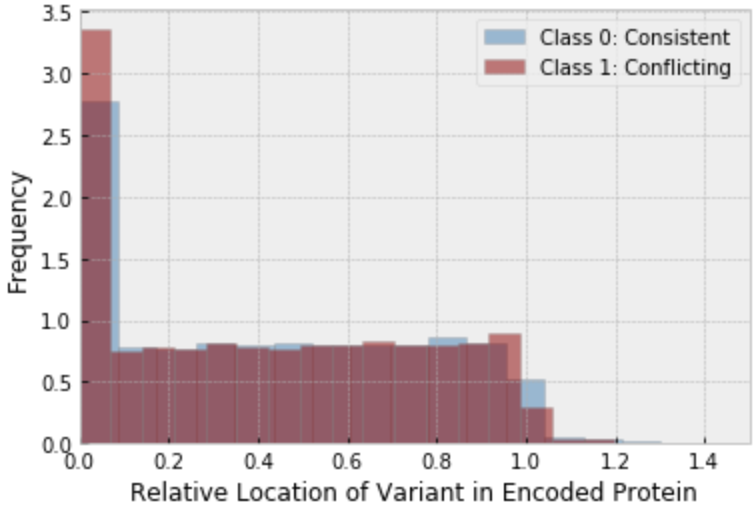
I will use statistical tests from the scipy.stats package in Python to answer two questions about the classification in relation to different variables. One variable is continuous, so I will use the t-test. The second question deals with a categorical variable, so I will use a chi squared test.

Questions:

1. Is the relative location of the mutation in the length of the entire protein related to the classification?

   Approach: Perform a two tailed t-test using scipy.stats t package.

   - Null Hypothesis: There is no difference in mean relative location in protein between class 0 and class 1
   - Alternate Hypothesis: There is a difference in mean relative location in protein between the two classes
   - Alpha = 0.05



   Results: The p-value is 0.057, greater than the alpha = 0.05. I cannot reject the null hypothesis and conclude that there is no significant difference in mean relative location between the two groups.

2. Are Indels or SNPs (or any other type of variant) more prone to conflicting classification compared to other types of variants?
   Approach: Perform a chi squared test on a contingency table that contains the types of variants as column names and classification as row labels. I will use the scipy.stats chi2_contingency function and chi2 package
   - Null Hypothesis: There is no difference in classification between different types of variants (i.e. indels/SNPs)
   - Alternate Hypothesis: There is a difference in classification between different groups of variants
   - Alpha = 0.05

CLASS	CLNVC_ Deletion	CLNVC_ Duplication	CLNVC_ Indel	CLNVC_ Insertion	CLNVC_ Inversion	CLNVC_ Microsatellite	CLNVC_ single_nucleotide _variant
0	2057	816	207	81	13	2	45578
1	452	218	40	14	4	3	15703

Results: The chi2 test shows that the classification and variant type are not independent of each other. The statistic is greater than the critical value, and the p-value is less than alpha = 0.05. Therefore I reject the null hypothesis that there is no difference in classification across different types of variants.

Machine Learning Models

Approach

The goal of this project is to classify genetic variants into two categories, conflictingly classified and consistently classified, making it a binary classification problem. Given some information about a genetic variant, can ML predict which variants are most likely to be conflictingly classified?

Four separate machine learning models will be evaluated as possible models to answer this question. Each of these models needs to be well suited to binary classification, and should be able to handle a large number of features.

These models will be trained and evaluated on 3 sets of data: (1) the raw data, (2) the data after data wrangling, and (3) the data with NLP engineered features. Hyperparameters and model performance will be tracked in a hyperparameter table in order to be able to compare models. These hyperparameter tables are saved each time the notebooks are run in a '.pickle' file.

Logistic Regression

Jupyter Notebook: Logistic Regression Notebook

Logistic regression is a good starting off point as a basic binary classification model. After training the model, evaluation metrics reveal that it did not perform very well. However, some insights I gained from training this model were:

- ➔ My cleaned data, that included one-hot encoded features and imputed numerical features with engineering NLP features for 'REF' and 'ALT' performed better than the raw data
- ➔ Balancing the data was critical in this model due to the class imbalance of my data
- ➔ For this model, MinMax scaling was improved the performance both in terms of accuracy and in f1 score
- ➔ Accuracy is not the best metric of model performance because the data is imbalanced. F1 score is more important in this case because it is the average of precision and recall.
- ➔ Precision and Recall can give a better sense of how good the model is at identifying the positive class (in this case, the minority class)

The best performing model after cross validation and hyperparameter tuning performed a little better than random, with a **test accuracy of 52.8%** and a **f1 score of 0.439**.

Decision Tree

Jupyter Notebook: Decision Tree Notebook

The next model chosen was a Decision Tree Classifier. Decision Tree Classifier works well for learning complex data; however, it is important to be cautious when interpreting the test accuracy score because Decision Trees are notorious for overfitting data. This was evident in the hyperparameter data after training all datasets. Some key conclusions from this classifier were:

- ➔ Decision Tree models gave a higher accuracy than Logistic Regression, however training accuracy was much higher than test accuracy. This was a good indication that it was overfitting to the training data.
- ➔ This overfitting problem could be mitigated by hyperparameter tuning using RandomizedSearchCV and a grid of hyperparameters to test
    - ◆ I made sure to run this CV with scoring argument of 'balanced_accuracy', which doesn't just optimize for validation set accuracy but instead takes into account the imbalanced nature of the data. This results in models that perform better both in terms of accuracy and in terms of f1 score.
    - ◆ The results of this method gave a much better f1 score than before hyperparameter tuning. Additionally, the model did not seem to be as overfit, because training and test accuracy were similar.
    - ◆ The reduction in overfitting follows from the optimal hyperparameters.. For example, the 'max_depth' of the decision tree was limited to 10 instead of being unlimited. This allows more impurity in the final decision boundaries, but makes the model more generalizable and less specific to the training set.

The best performing model using the Decision Tree Classifier provided an improved performance compared to the Logistic Regression Model. After RandomizedSearchCV for hyperparameter tuning, the best model had a **test accuracy of 67.2%** with an **f1 score of 0.516**. Below is a visualization of the Decision tree to a depth of 4.

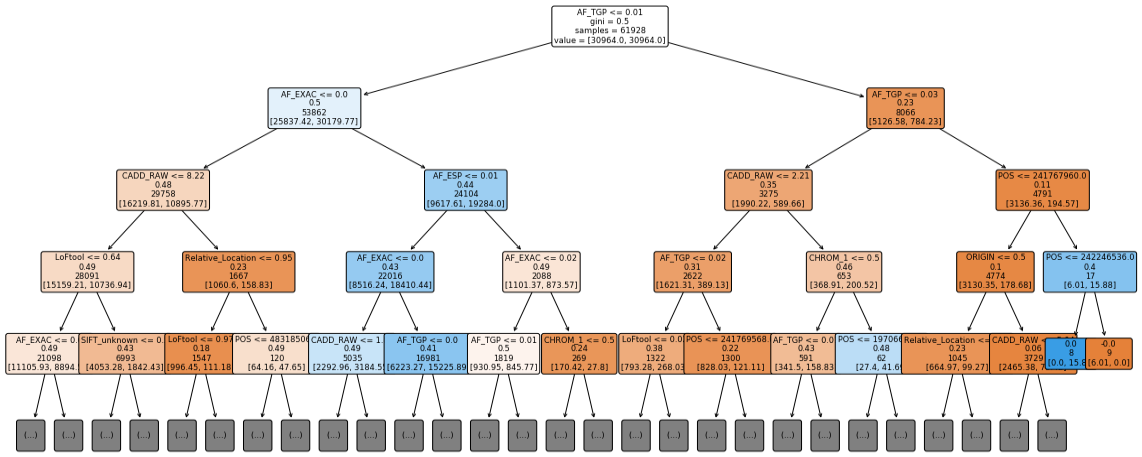


Figure: Decision Tree Visualized (limited depth = 4)

Random Forest
Jupyter Notebook: Random Forest Notebook

Random Forest Classifier is a model based off of the Decision Tree Classifier. It is excellent for binary classification of datasets with complex features. It utilizes bagging of trees, whereby it takes a bootstrapped sample of data (samples taken with replacement) and trains a decision tree on it with only a subset of the features. It does this a number of times until a 'forest' is grown. Some key findings from training and evaluating this model were:

- ➔ Random Forest performed slightly better than Decision Tree Classifier, which is to be expected.
- ➔ The model was overfit to the training data before hyperparameter tuning limited the max_depth of the trees & the n_estimators (number of trees)

Off the shelf, the Random Forest Model outperformed all of the previous models in test accuracy on the raw data, which was to be expected. However, a closer look shows that the training accuracy was 96%, indicating this model was overfit. Training the model on our cleaned dataset as well as performing RandomizedSearchCV for hyperparameter tuning yielded a better model. The best model had a **test accuracy of 71.1%** and an **f1 score of 0.542**. This is the best performing model thus far.

XGBoost
Jupyter Notebook: XGBoost Notebook

XGBoost is a gradient boosting tree-based classifier that is an extension on bagging. This is a widely used algorithm used to classify data with high accuracy. XGBoost works by training models on simpler decision trees (stumps) that evolve over time. This happens through reweighting models as the model learns. Gradient boosting is a sequential learning algorithm, with a number of parameters that can be tuned such as # of trees, # of splits, and how weights evolve. Some key findings from this model:

- ➔ Test accuracy is higher than that of random forest models, however the f1 score is much lower
- ➔ Even after hyperparameter tuning with RandomizedSearchCV using both scoring='balanced_accuracy' and scoring='f1', the f1 score was not improved above that of the Random Forest model.

The best performing model had a **test accuracy of 77%** and an **f1 score of 0.444.**

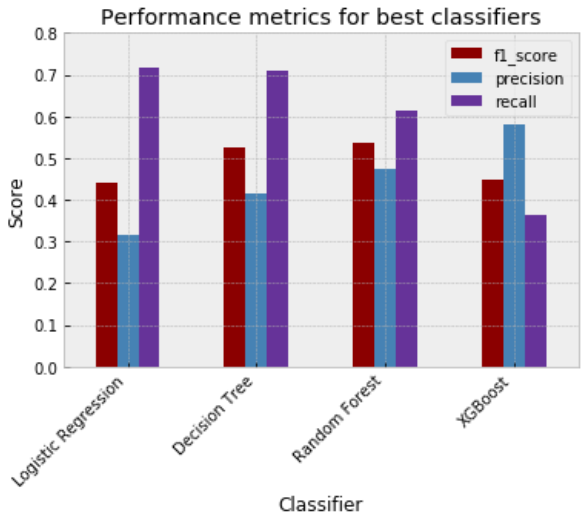
Model Comparison and Analysis:

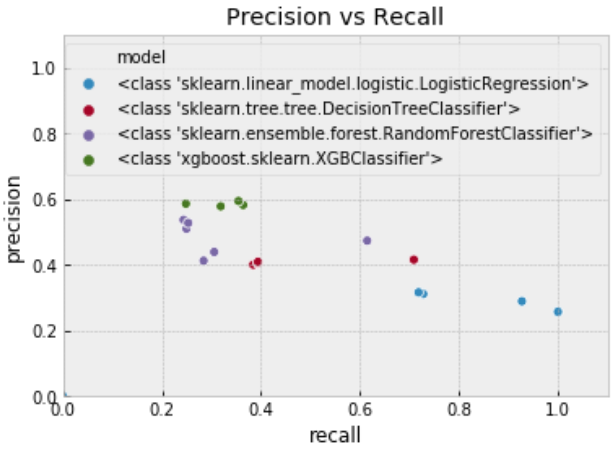
Jupyter Notebook: Final Model Analysis Notebook

The table below shows the final results of the best models from each of the four classifiers. The best performer, Random Forest, is highlighted in red.

Classifier	test accuracy	train accuracy	precision	recall	f1 score
Logistic Regression	0.53	0.54	0.32	0.72	0.44
Decision Tree	0.67	0.69	0.42	0.71	0.52
Random Forest	0.73	0.83	0.47	0.61	0.53
XGBoost	0.77	0.93	0.58	0.36	0.45

The performance metrics are plotted on the bar chart and scatterplot below for easier visualization. The Random Forest model slightly outperforms the Decision Tree based model in terms of f1 score. Interestingly, the XGBoost model performs the best in terms of precision.

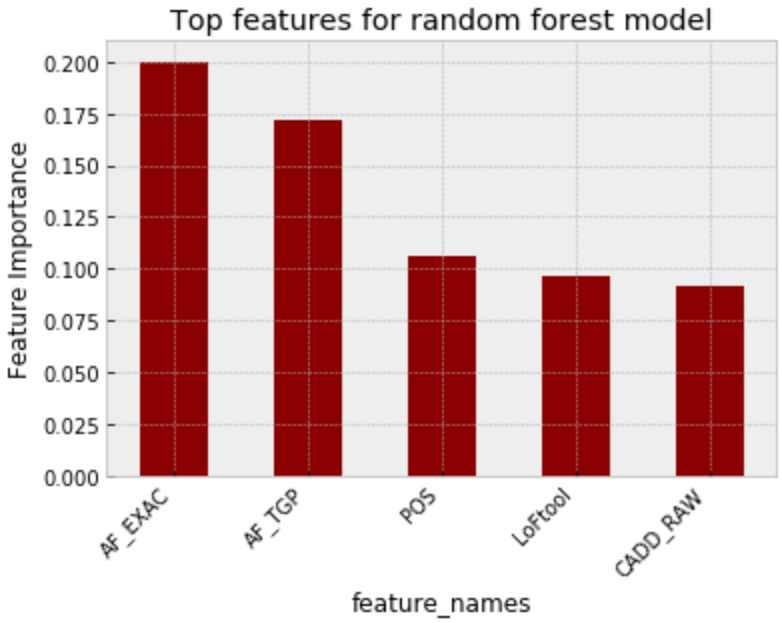




While logistic regression produces an output of coefficient values indicating the weighting of each feature in the model, the rest of the 3 tree-based models generate feature importances. Feature importances are ranked by the drop in gini impurity that a split along that feature provides. Therefore, it is difficult to compare these two types of outputs. Let's take a closer look at the coefficients from the best Logistic Regression model and the feature importances from our best tree-based model, Random Forest:

➔ Below is a plot of the top five most heavily weighted features in our logistic regression model:

➔ Below is a bar plot of the top 5 ranked feature importances in Random Forest:



➔ While it is difficult to compare these two outputs, it is interesting to see that in both rankings, allele frequency shows up, whether as AF_EXAC, AF_TGP, or

AF_ESP. In general it seems like allele frequency is an important feature in both models.

Conclusions & Recommendations

As shown above, the best model in terms of maximizing both precision and recall is the Random Forest Classifier. This model, with its hyperparameters found in the file below, gives a test accuracy of 74% and an f1 score of 0.53: Final Hyperparameters

XGBoost had the best performance in terms of precision, which is important when trying to identify the minority positive class. However, the consequences of a type I error (false positive) are not as severe as most medical testing data problems. This is because the classification from this model will serve as information to decide if additional testing is needed for a particular variant. No clinical diagnoses would be made off of the output of this model. Therefore, we should also focus on having improved recall, which means correctly identifying as many of the true positives as positive. This is the reasoning behind recommending the Random Forest model for predicting whether or not a genetic variant is conflictingly or consistently classified. Future iterations of the XGBoost model should aim to improve recall as well as precision.

Future Directions

There are many features in the raw dataset which were not able to be used due to being non-numeric as well having too many categories for one-hot encoding. Future models can be trained on data that has more NLP engineered features. For example, there are 9,260 unique disease identifiers in the feature named 'CLNDN'; this is a feature where NLP on the words in these disease identifiers might further inform ML models. Additionally, outside of this dataset, the raw genome sequence along with its variants represented in this dataset can be analyzed using byte-pair-encoding (BPE) NLP tools, such as SentencePiece. This additional data on the genome for each variant can provide information on potential variant interactions or rates of co-occurrence.