Human Genetic Variants Classification

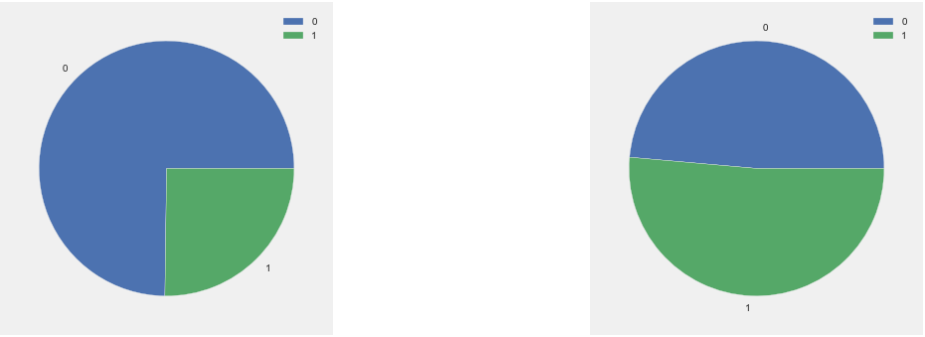
ORCA 4500 Foundation of Data Science, Project Report

Chenyu Huang
Team #5

- **Contents**
  - About the Dataset
  - Problem Statement
  - Q1: Classification
    - Logistic Regression
    - Decision Tree
    - Random Forest
    - Adaboost
  - Q2: Feature Engineering
    - Random Forest
  - Q3: Clustering
    - K-means
  - Summary

https://github.com/frostace/BinaryClassification

- **About the Dataset (ClinVar)**
  - **Interpretation**
    Each row represents a human genetic variant, each column represents a feature of the variant
  - **Label**
    Each variant has a label.
    There will be several clinics making diagnosis, telling us whether the variant is benign or neutral, or pathogenic.
    If all clinics have the same diagnosis, this variant is classified as label 0, which stands for non-conflicting, otherwise, it will be classified as label 1 – conflicting.
  - **Feature**
    After omitting the columns with too many empty entries, there are 37 features in all in this dataset, 24 of them are categorical, and 13 of them are numerical.
  - **Size**
    There are 65,188 genetic variants contained in the dataset.
  - **Normalize**
    Normalizing the numerical values can reduce computational cost to some extent.
  - **Balance**
    At the beginning, the dataset is extremely imbalanced, over 75% variants are labeled as non-conflicting. So I resampled the dataset to make the new dataset balanced, and it contains 20,000 variants.



a) Original Dataset b) Resampled Dataset

Figure 1. Dataset Balancing

https://github.com/frostace/BinaryClassification

- # Problem Statement

Due to the rocketing development of Artificial Intelligence, some regular problems which are relatively difficult for traditional techniques can be easily solved with advanced machine learning algorithms. Among them, 2 most frequently discussed problems are classification and regression.

This project mainly focused on classification. Various approaches can be applied to deal with classification problems, such as: regression approach, decision tree, bagging approach, boosting approach, etc. Since labels are used during the training process in these learning algorithms, they are called supervised learning. If the there's no label in the dataset or the labels are not used, this is called unsupervised learning, e.g. K-means Clustering, Hierarchical Clustering.

The whole project can be divided into 3 parts, or 3 questions as listed below.

1. **Classification**
   Estimate how many variants will have conflicting classifications, why are they considered to have conflicting classifications?
   - Methodology:
     - Logistic Regression
     - Decision Tree
     - Random Forest
     - Adaboost
2. **Feature Engineering**
   Evaluate the importance of all these features, find the most important one and analyze why is it so important.
   - Methodology:
     - Random Forest
3. **Clustering**
   Redo Question 1 with classification label unknown, compare the 2 results.
   - Methodology
     - K-means Clustering

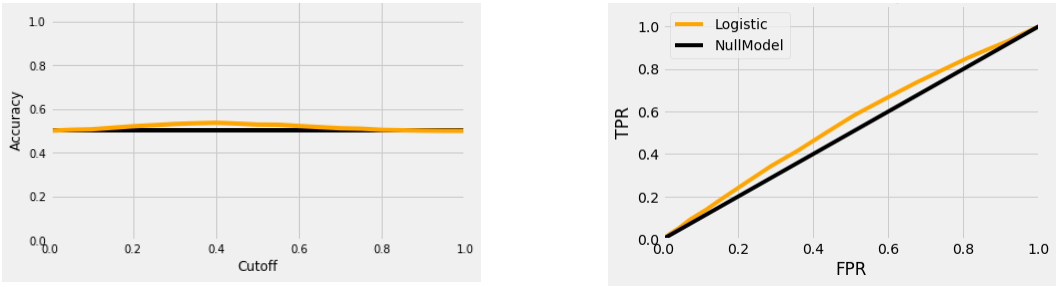
Q1: Classification

- ## Method 1: Logistic Regression
  - ### Main Idea

    Logistic Regression is actually a regression algorithm. And the output domain is $(-\infty, \infty)$. So in order to solve a binary classification problem, we have to map the output domain to a smaller domain $(0, 1)$ with a link function. And then decide a cutoff point to turn the continuous result to a discrete one.
  - ### How to Deal with Categorical Data?

    At the beginning, I though logistic regression cannot take in categorical data because I always got some errors or warnings during the training process, so I mapped categorical data to numerical values according to the indices of the categories, and used this mapped dataset to train my logistic regression model. However, it turns out that the performance is just slightly better than the null model.

    The highest accuracy is only 53.6% while the AUC is only 0.544, it's basically the same as blindly guessing.

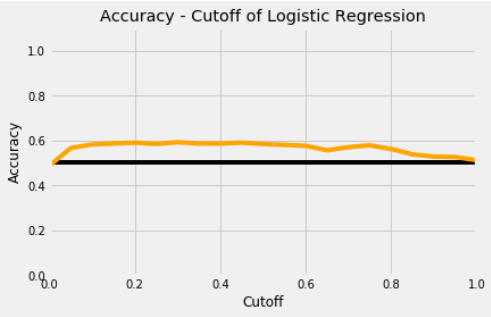


a) Acc - Cutoff b) ROC Curve

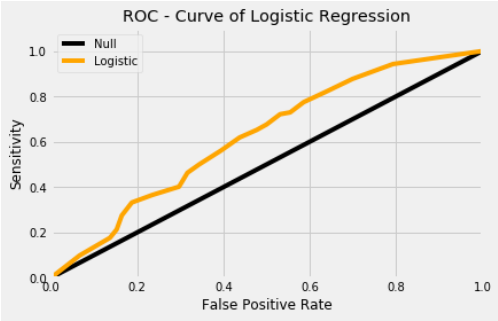
Figure 2. Logistic Regression Performance if mapped categorical data to numerical data

Later I tried another technique called one-hot encoding to before feeding categorical data into the model. It works much better because this method makes the categorical data sparse while my mapping method is just randomly giving value to the categorical data.

https://github.com/frostace/BinaryClassification



a) Acc - Cutoff



b) ROC Curve

Figure 3. Logistic Regression Performance with one-hot encoding

The accuracy reaches 61% this time and the ROC curve looks much better as well, the AUC is 0.625.

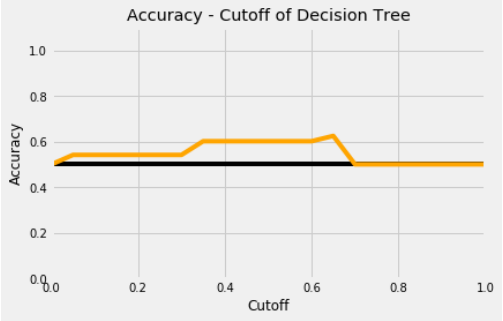
- # Method 2: Decision Tree
  - ## Main Idea

    The main idea of decision tree for binary classification is to ask yes / no questions and evaluate whether or not a question is a good question with cross entropy or gini index. If a question does a perfect job in splitting the data and marking them as one of the 2 labels, the branch data should appear to have a great information attenuation. So we can recursively ask questions until the decision tree converge.

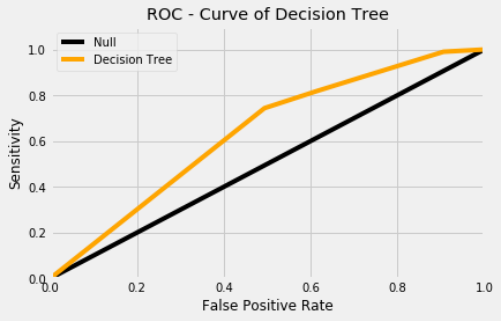
    What to mention here is, in order to avoid an overfitting problem, we cannot set the convergence condition to be too strict.
  - ## Performance

    The performance of decision tree is better than logistic regression. As shown in Figure 4, the accuracy is 64.1%, AUC is 0.669.

https://github.com/frostace/BinaryClassification



a) Acc - Cutoff



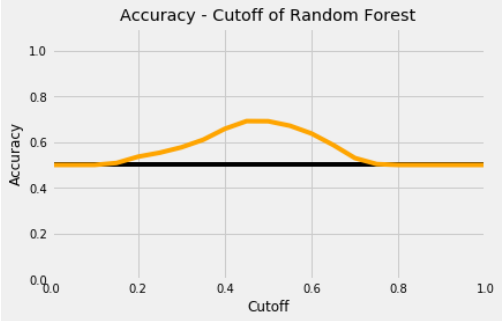
b) ROC Curve

Figure 4. Decision Tree Performance

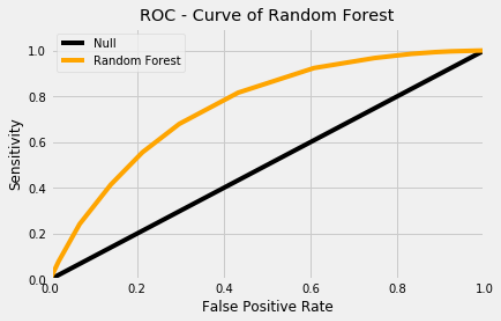
- # Method 3: Random Forest
  - ## Main Idea

    Random Forest is the ensemble of several decision trees trained with a bootstrapping method. I iteratively take the original dataset and generate a bootstrapped dataset, train a decision tree with a random subset of features. Finally gather all the trees together to form a random forest. The final prediction of the random forest is the average voting of all decision trees.
  - ## Performance

    The accuracy achieves 68.6%, and the AUC achieves 0.747. We can see clearly from Figure 5 that Random Forest outperformed the previous 2 algorithms.



a) Acc - Cutoff



b) ROC Curve

Figure 5. Random Forest Performance

https://github.com/frostace/BinaryClassification

- **Method 4: Adaboost**
  - **Main Idea**

Adaboost algorithm is pretty similar with random forest. The main difference is every tree in Adaboost is not considered equally, instead, it has an amount of say according to its misclassification rate. Also, the dataset recursively updates to tend to pay more attention on training the misclassified data.

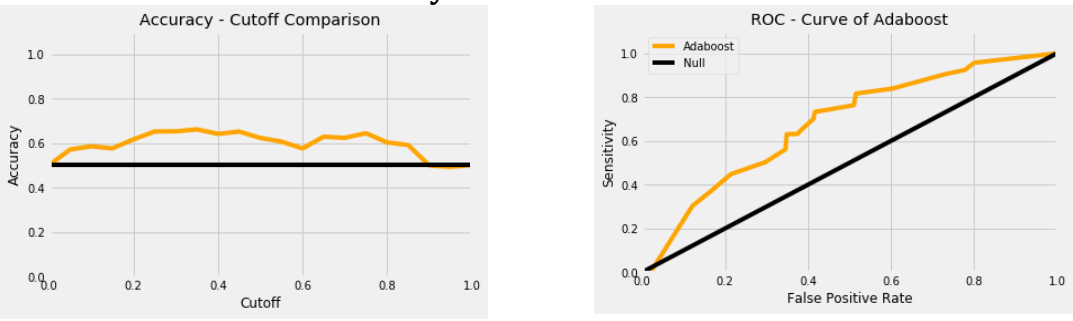
Here I noticed that if I feed in a small training set to Adaboost, after a few rounds, the training set will gradually lose diversity until it only contains several rows of the same data, which leads to an overfitting problem.

So I have to feed in a relatively big training set to train Adaboost for each cutoff separately. And I used Kaggle Kernels to help me do the training job online. The concurrence helps a lot in reducing computational cost, but it also leads to a small problem that the result turns out to be fluctuating, and the curves are not that smooth.

  - **Performance**

The accuracy is 66.2% while the AUC is 0.679.

I expected Adaboost to perform better than Random Forest because it's a boosting algorithm and it makes more sense than a bagging algorithm. However, due to the overfitting problem and the time consumption, I haven't tried using a even larger training set to train Adaboost. I think theoretically it will work better than Random Forest.



a) Acc - Cutoff b) ROC Curve

Figure 6. Adaboost Performance

https://github.com/frostace/BinaryClassification

Q2: Feature Engineering

- ## Method: Random Forest
  - ### Main Idea

    In Random Forest, I use a random subset of features to train it. If I fix a specific feature in this subset, I should see the importance of this fixed feature according to the accuracy in the end.

    And I ran several times to get the expected accuracy because I included random sampling here and this should be a stochastic process.

    Here I used subset size = 10, and iteratively train Random Forest for all available features.
  - ### Rank of Feature

    After sorting by accuracy, I listed the top features below.

Table 1. Weight of Features

Feature	Accuracy
AF_EXAC	0.641
CADD_PHRED	0.632
AF_ESP	0.631
CLNDISDB	0.620
AF_TGP	0.614

Maybe I should have did this question in the beginning to give me a basic sense on which features are important and which features are trivial, and this can be considered as Feature Engineering to give us guide on selecting features.

https://github.com/frostace/BinaryClassification

Q3: Clustering

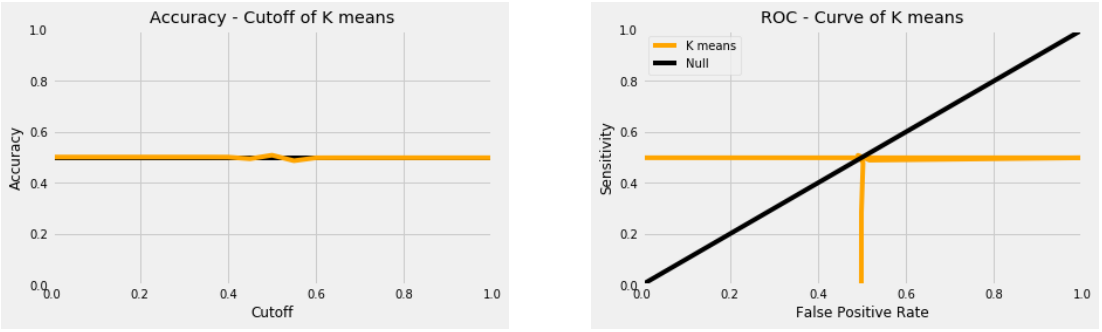
- **Method: K-means**
  - **Main Idea**

Step	Procedure
1	Select the number of clusters you want to identify in your data
2	Randomly select k distinct data points
3	Measure the distance between the 1st point and the k initial clusters
4	Assign the 1st point to the nearest cluster
5	Iterate through all points and do step 3 & 4
6	Calculate the mean of each cluster
7	Use the calculated mean of each cluster as k new initial data points and restart from 3
8	Loop until the mean converge
9	Do Step 1 - 8 for n times, select the best one

Before I started to implement K-means algorithm, I mapped the categorical data to numerical data because categorical data don't have Euclidean distance.

  - **Performance**

It turns out that the K-means completely messed up and it's not working at all. This is because my mapping issue changed the original categorical similarity to be a random distributed numerical similarity, and 65% of my features are categorical, so the final prediction is basically a random result.



a) Acc - Cutoff b) ROC Curve

Figure 7. K-means Performance

https://github.com/frostace/BinaryClassification

Summary

- **Comparison of all methods**

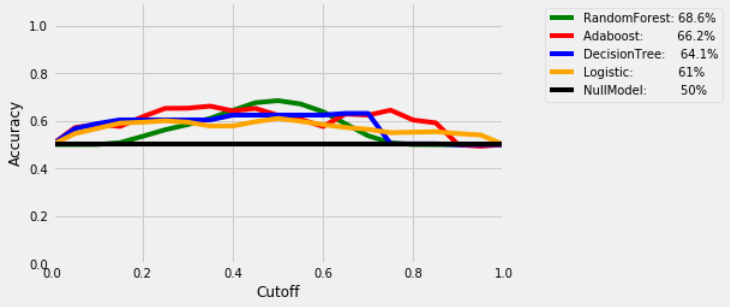


Figure 8. Accuracy – Cutoff Comparison

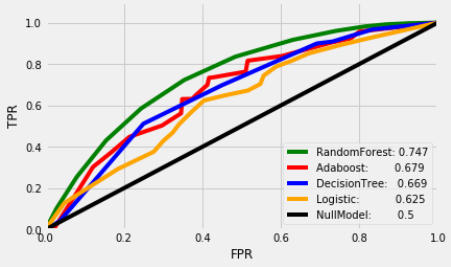


Figure 9. ROC Curve Comparison

In conclusion:

I really enjoyed playing with these machine learning algorithms. I see how they predict according to the feedbacks and observe the performance and modify my code again as if me myself is an agent as well.

I have understood the basis of tree-based algorithms and saw the power of ensemble learning through this project.

At the meantime, I retrospect the process of my project, I found that this project is actually lack of well-organized data preprocessing and feature engineering. This may be a reason why the acc never hits 70%. Also, the low accuracy may result from the original data quality.

https://github.com/frostace/BinaryClassification