BINF 5527 MACHINE LEARNING IN BIOINFORMATICS ASSIGNMENT: GENETIC VARIANT CLASSIFICATION

Mehmet Cihan Sakman
sakmancihan@gmail.com

Friday 10th September, 2021

Abstract

In this project, we'll build Machine Learning algorithms to predict whether a given variant has **conflicting classification** or not based on the dataset obtained from Kaggle. During this project we'll commonly use **scikit-learn** library from Python. While applying Machine Learning Algorithms we'll build four different algorithms such as: Gradient Boost, Logistic Regression, Decision Tree, and Random Forest classifiers.

1 Introduction

The main purpose of this project is predicting given human genetic variant have conflicting classification or not based on Machine Learning classification algorithms. Variants that have conflicting classifications (from laboratory to laboratory) can confuse when clinicians or researchers try to interpret whether the variant impacts the disease of a given patient. If we gain a good classification algorithm at the end of this project, we would like to help clinicians detect the variants classification.

Necessary information and the data itself can be achieved from: `https://www.kaggle.com/kevinarvai/clinvar-conflicting`

2 Features of Data

List of important features for Genetic Variant Classification dataset:

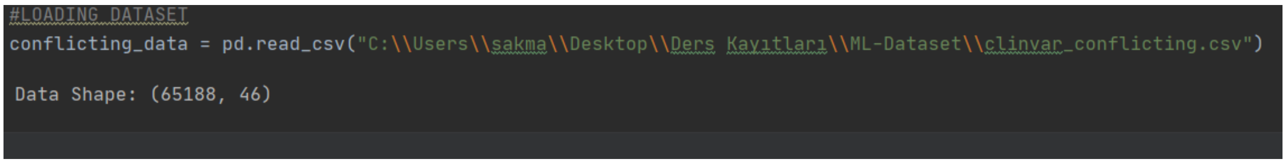
- **CHROM** : Chromosome the variant is located on
- **POS** : Position on the chromosome the variant is located on.
- **REF** : Reference Allele
- **ALT** : Alternate Allele
- **AF_ESP** : Allele frequencies from GO-ESP
- **AF_TGP** : Allele frequencies from the 1000 genomes project
- **AF_EXAC** : Allele frequencies from ExAC
- **CLNDISDB** : Tag-value pairs of disease database name and identifier, e.g. OMIM:NNNNNN

- **CLNDISDBINCL** : For included Variant: Tag-value pairs of disease database name and identifier, e.g. OMIM:NNNNNN
- **CLNDN** : ClinVar's preferred disease name for the concept specified by disease identifiers in CLNDISDB.
- **CLNHGVS** :Top-level (primary assembly, alt, or patch) HGVS expression.
- **CLNVC** : Variant Type
- **CLNVI** : The variant's clinical sources reported as tag-value pairs of database and variant identifier
- **MC** : Comma separated list of molecular consequence in the form of Sequence Ontology ID—molecular_consequence
- **ORIGIN**: Allele origin
- **SSR**: Variant Suspect Reason Codes.
- **CLASS** : The binary representation of the target class. 0 represents no conflicting submissions and 1 represents conflicting.(Target feature)
- **Allele** : The variant allele used to calculate the consequence
- **Consequence** : Type of consequence
- **IMPACT** : The impact modifier for the consequence type
- **SYMBOL** : Gene name
- **Feature_type** : Type of feature. Currently one of Transcript, RegulatoryFeature, MotifFeature.
- **Feature** : Ensemble stable ID of feature
- **BIOTYPE** : Biotype of transcript or regulatory feature
- **EXON**: The exon number (out of total number)
- **INTRON**: The intron number (out of total number)
- **cDNA_position** : Relative position of base pair in cDNA sequence
- **CDS_position**: Relative position of base pair in coding sequence
- **Protein_position** : Relative position of amino acid in protein
- **Amino_acids** : Only given if the variant affects the protein-coding sequence
- **Codons** : The alternative codons with the variant base in upper case.
- **DISTANCE** : Shortest distance from variant to transcript
- **STRAND** : Defined as + (forward) or - (reverse).
- **BAM_EDIT** : Indicates success or failure of edit using BAM file

- **SIFT**: The SIFT prediction and/or score, with both given as prediction(score)
- **PolyPhen** : The PolyPhen prediction and/or score
- **LoFtool** : Loss of Function tolerance score for loss of function variants
- **CADD_PHRED** : Phredscaled CADD score.
- **CADD_RAW** : Score of the deleteriousness of variants:
- **BLOSUM62** : See `http://rosalind.info/glossary/blosum62/`

3 Preprocessing Steps

We load our datasets as seen below as in csv format. We can see that the data has **65188** rows and **48** columns.



```
#LOADING DATASET
conflicting_data = pd.read_csv("C:\\Users\\sakma\\Desktop\\Ders Kayıtları\\ML-Dataset\\clinvar_conflicting.csv")

 Data Shape: (65188, 46)
```

Figure 1: Loading Dataset

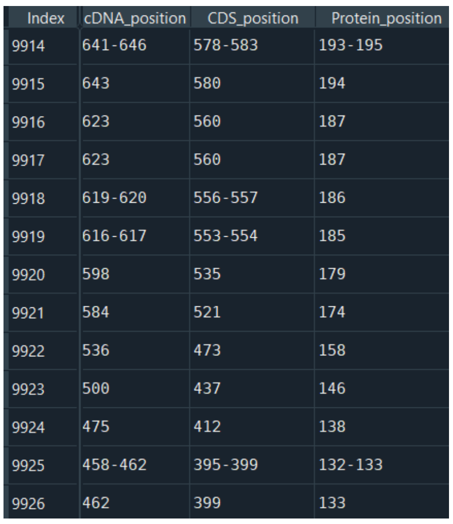
3.1 Clean Fully NaN Features

First of all, there were **10** columns which has %99.9 NaN values. These columns have cleaned from the dataset.

3.2 Convert Into Float

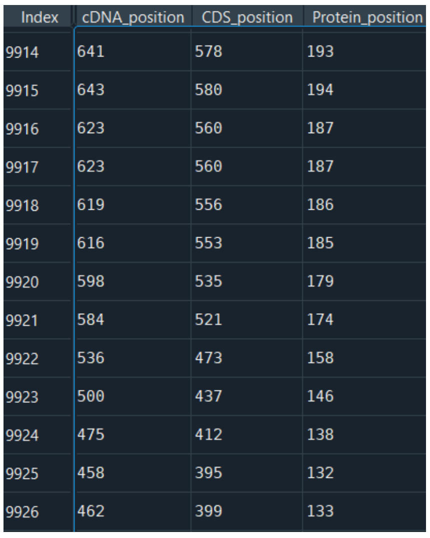
In *cDNA_position*, *CDS_position*, *Protein_position* values are integer but type of features are Object. Because there are some approximation such as 1331-1332 for relative positions. We took the valid(first or second) one because they are already so close to each others.

We can see the changes below.



Index	cDNA_position	CDS_position	Protein_position
9914	641-646	578-583	193-195
9915	643	580	194
9916	623	560	187
9917	623	560	187
9918	619-620	556-557	186
9919	616-617	553-554	185
9920	598	535	179
9921	584	521	174
9922	536	473	158
9923	500	437	146
9924	475	412	138
9925	458-462	395-399	132-133
9926	462	399	133

Figure 2: Before preprocessing



Index	cDNA_position	CDS_position	Protein_position
9914	641	578	193
9915	643	580	194
9916	623	560	187
9917	623	560	187
9918	619	556	186
9919	616	553	185
9920	598	535	179
9921	584	521	174
9922	536	473	158
9923	500	437	146
9924	475	412	138
9925	458	395	132
9926	462	399	133

Figure 3: After preprocessing

3.3 CHROM Feature

CHROM represented for chromosome the variant has located on, and we have 22 + 2 chromosome in the dataset. The first 22 chromosomes are represented from 1 to 22, and the other two chromosomes are Y and Mitochondrial Chromosome. These two are represented as Y and MT. These two chromosomes have converted into 23 and 24.

3.4 Highly Correlated Features

First, correlation matrix has been plotted and the correlation between features analyzed. After

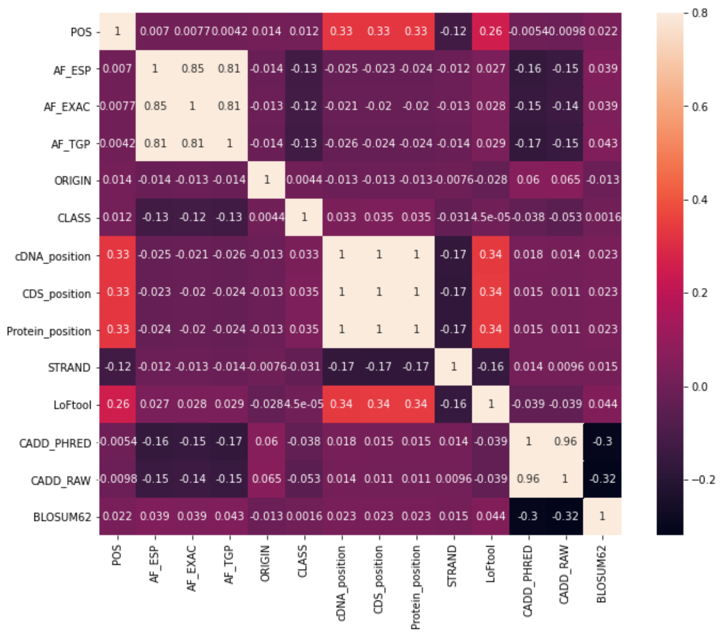


Figure 4: Correlation Matrix

analyzing correlation matrix we clearly see that *'cDNA_position', 'CDS_position','Protein_position'* features has 1 correlation between them. We can drop two of them and use just *cDNA_position* because it has fewer NaN values than others.

3.5 EXON and INTRON

These two columns represent the exon and intron number out of the total number. These values are kept as i.e. *10/12*. These values are converted into float numbers and kept as float.

The other issue for these features were NaN values. For EXON and INTRON, their NaN ratios are parallel. The percentage of NaN values for INTRON is %86.4, and the percentage of NaN values for EXON is %13.6. Therefore the NaN values of EXON with filled with the corresponding values of INTRON.

3.6 Zero Varience

In the dataset, two columns have zero variance(all variables are unique). These two columns are *CLNHGVS*, and *CLNVI* with the Object type. In the CLNVI feature, there are 27659 non-null variables and 27655 unique values. In the CLNHGVS feature, there are 65188 unique values out of 65188. These two columns dropped from the dataset.

3.7 CLNVC

CLNVC feature is stand for variant type. 94% of values are **'single_nucleotide'**, and 6% of the values are **Deletion, Duplication, Inversion**, or **Insertion**. This column converted as a binary column whether it is a **single basepair substitution** or not.

On the other hand, we can conclude if the variant type is a single base-pair or not by checking the length of ALT and REF alleles. If one of the alleles includes more than one basepair, it means that the variant type is multi basepair substitution(not a single basepair). Therefore we can drop the ALT and REF columns and use the CLNVC column.

3.8 Similar Features

ALT(Alternate allele) and **Allele** are referring to the same alleles in the human genetic. Therefore we can drop the Allele column because it has some NaN values, but ALT doesn't.

MC feature is the comma-separated list of molecular Consequence in the form of Sequence Ontology, and **Consequence** feature is the type of molecular Consequence. Therefore, we will keep the type of molecular Sequence and drop the **MC** feature.

CLNDISDB feature is the tag-value pairs of disease database name and identifier, and **CLNDN** feature is the disease name for the identifier. Therefore we will keep the disease name and drop the **CLNDISDB**. At the same time, in the **CLNDN** column, both *not_provided* and *not_specified* values are referring the same meaning. Both values were kept as *not_specified*. For last, the **CLNDN** column has been kept as a binary column. Whether tag-value pairs have a disease or not. If it has a disease, assign 1, else 0.

3.9 Dealing With Missing Values

3.9.1 Missing Because not Exist

cDNA can be described as DNA without all the necessary noncoding regions. That is means if there is no EXON, we cannot talk about the cDNA position. There is an exception. In the lack of EXON and INTRON, we may have a cDNA position. Therefore we will fill the NaN values in **cDNA_position** as -1. That will refer to there is no such position.

CADD is a tool for scoring the deleteriousness of single nucleotide variants and insertion/deletion variants in the human genome. All the NaN values in **CADD_PHRED** and **CADD_RAW** belong to non-single nucleotide variants. That is means there is no CADD information for non-single nucleotide variants. We will fill the values with -1.

In the dataset, **AminoAcids** and **Codons** are only given if the variant affects protein-coding sequence. Therefore, NaN values in AminoAcids and Codons columns filled with *not-affect*.

3.9.2 Categorical Missing Values

For the categorical columns, NaN values have been kept as *unknown*.

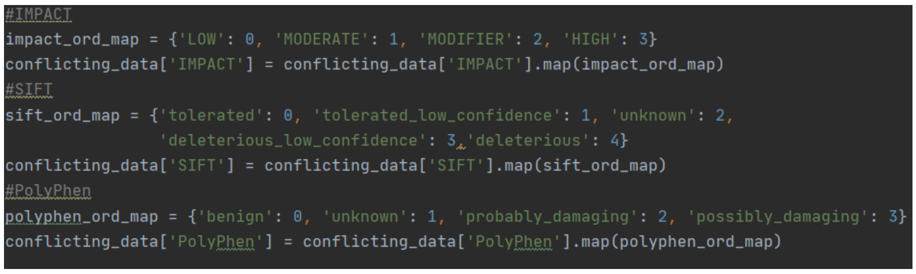
3.9.3 Nominal Missing Values

There is only one case for the Nominal Missing Values and that is for **LoFtool**. Missing values in LoFtool column filled with the mean by using **sklearn** *impute* library.

3.10 Encoding

3.10.1 Ordinal Columns

In dataset, there are three ordinal columns as follow: **IMPACT, SIFT** and **PolyPhen**. These columns mapped as below.



```
#IMPACT
impact_ord_map = {'LOW': 0, 'MODERATE': 1, 'MODIFIER': 2, 'HIGH': 3}
conflicting_data['IMPACT'] = conflicting_data['IMPACT'].map(impact_ord_map)
#SIFT
sift_ord_map = {'tolerated': 0, 'tolerated_low_confidence': 1, 'unknown': 2,
                'deleterious_low_confidence': 3,'deleterious': 4}
conflicting_data['SIFT'] = conflicting_data['SIFT'].map(sift_ord_map)
#PolyPhen
polyphen_ord_map = {'benign': 0, 'unknown': 1, 'probably_damaging': 2, 'possibly_damaging': 3}
conflicting_data['PolyPhen'] = conflicting_data['PolyPhen'].map(polyphen_ord_map)
```

Figure 5: Ordinal Feature Mapping

3.10.2 One Hot Encoding

BAM_EDIT, BIOTYPE, Feature_type columns have three unique values for each. Therefore, One Hot Encoding have been applied to these features by using **pandas'** *get_dummies* feature.

3.10.3 Feature Hashing

Feature Hashing has been applied for the nominal values which have more than 20 unique values. Feature Hashing is the special library in the *sklearn.feature_extraction*. All these nominal columns have been split into **four** new columns.

3.11 Summary of Preprocessing

At the end of the pre-processing we clean up our data from dummy variables, duplicate features, and NaN values, and using *OneHotEncoder* and *FeatureHashing* we transformed the categorical values into unique columns. After all this operations now we have *65188* rows and *48* columns and there is no NaN values in data.

3.12 Splitting Data

We splitted our data such that randomly selected %80 tuples are used for training while %20 tuples are used for testing with **train_test_split** from **sklearn** library.

3.13 Imbalanced Classification

We can clerly see that our target class is imbalanced. Before fit our training dataset we used a sampling algorithm called **Under Sampling**. We can observe the changes in the target class before and after the undersampling.

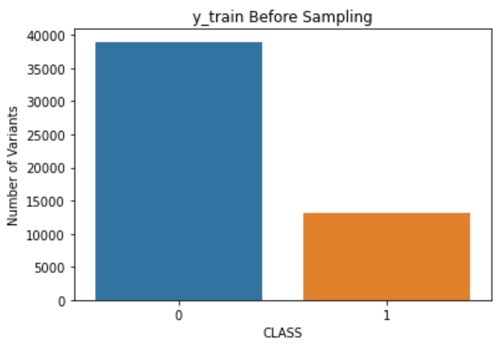


Figure 6: Before Sampling

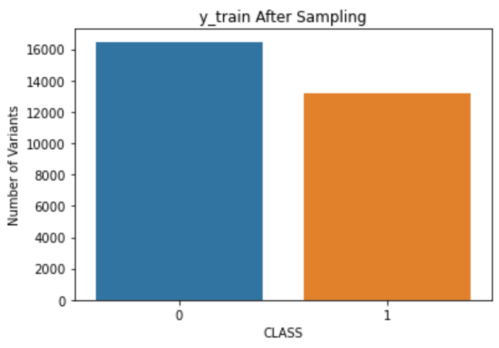


Figure 7: After Sampling

4 Apply Machine Learning Algorithms

After preprocessing we prepared our dataset for applying learning algorithms. We wanted to build four different learning algorithms to find the best for our dataset. We will use the following algorithms: Gradient Boosting Classifier, Logistic Regression, Random Forest, and Decision Tree Classifiers. Our aim to use these algorithms is that predicting the probability of CLASS(conflicting or not) membership for each example. Before these predictions, we can plot the importance of features.

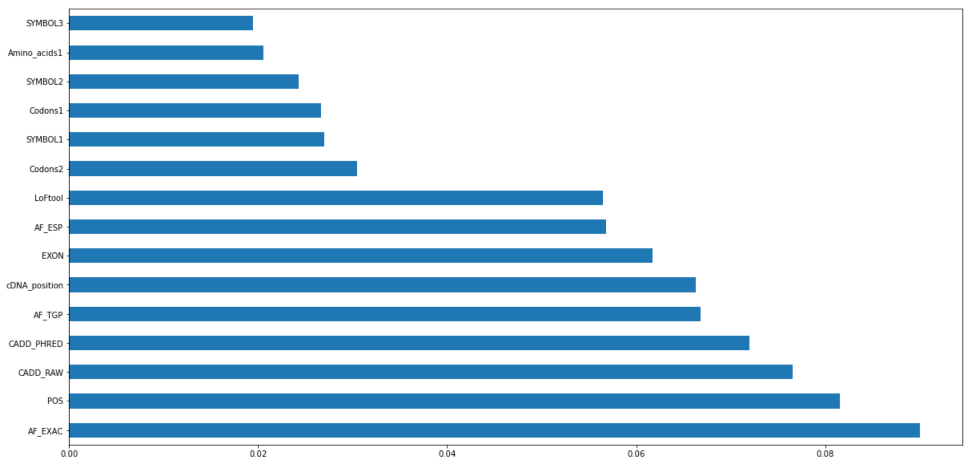


Figure 8: Importance of 15 Features

4.1 Metrics for Performance

Area Under ROC Curve (or ROC AUC for short) is a performance metric for binary classification problems and it is suitable for the predict the probability of the target class. A ROC Curve is a plot of the true positive rate and the false positive rate for a given set of probability predictions at different thresholds used to map the probabilities to class labels. The area under the curve is then the approximate integral under the ROC Curve. Also we'll print the **precision, recall** and **f1-scores** of the algorithms.

4.2 Gradient Boosting Algoirthm

The Gradient Boosting Machine is a powerful ensemble machine learning algorithm that uses decision trees. Boosting is a general ensemble technique that involves sequentially adding models to the ensemble where subsequent models correct the performance of prior models. After fitted our dataset into Gradient Boost Classifier algorithm and we can see the results below.

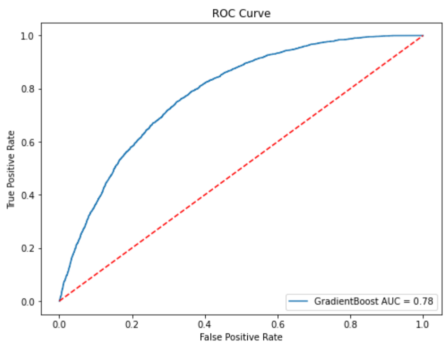
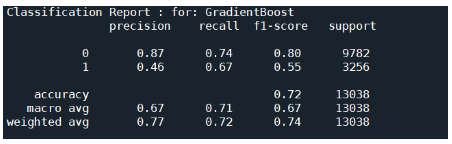


Figure 9: AUC



```
Classification Report : for: GradientBoost
              precision    recall  f1-score   support

           0       0.87      0.74      0.80      9782
           1       0.46      0.67      0.55      3256

    accuracy                           0.72     13038
   macro avg       0.67      0.71      0.67     13038
weighted avg       0.77      0.72      0.74     13038
```

Figure 10: Report

4.3 Logistic Regression Algoirthm

Logistic Regression is the most common learning algorithm for classification tasks. Due to our ground truth vector is binary Logist Regression is on of the best option for our data. After fitted our dataset into Logistic Regression algorithm we can see the results below.

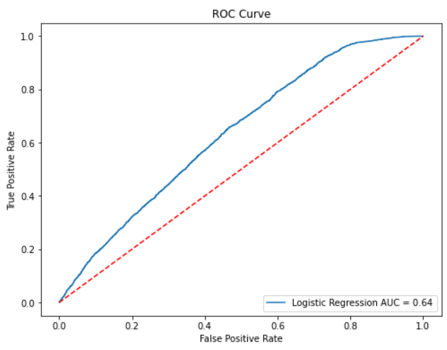
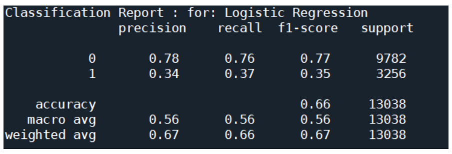


Figure 11: AUC



```
Classification Report : for: Logistic Regression
              precision    recall  f1-score   support

           0       0.78      0.76      0.77      9782
           1       0.34      0.37      0.35      3256

    accuracy                           0.66     13038
   macro avg       0.56      0.56      0.56     13038
weighted avg       0.67      0.66      0.67     13038
```

Figure 12: Report

4.4 Random Forest Algoirthm

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. After fitted our dataset into Random Forest algorithm we can see the results below.

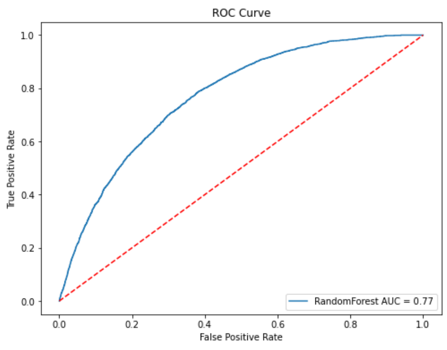
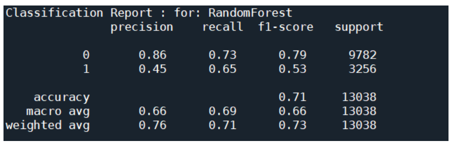


Figure 13: AUC



```
Classification Report : for: RandomForest
              precision    recall  f1-score   support

           0       0.86      0.73      0.79      9782
           1       0.45      0.65      0.53      3256

    accuracy                           0.71     13038
   macro avg       0.66      0.69      0.66     13038
weighted avg       0.76      0.71      0.73     13038
```

Figure 14: Report

4.5 Decision Tree Classifier

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too. The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data). After fitted our dataset into Random Forest algorithm we can see the results below.

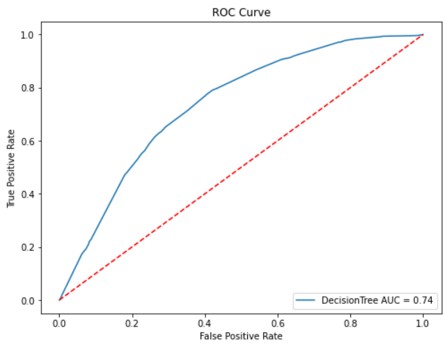
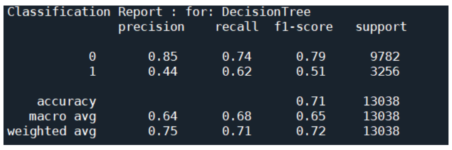


Figure 15: AUC



```
Classification Report : for: DecisionTree
              precision    recall  f1-score   support

           0       0.85      0.74      0.79      9782
           1       0.44      0.62      0.51      3256

    accuracy                           0.71     13038
   macro avg       0.64      0.68      0.65     13038
weighted avg       0.75      0.71      0.72     13038
```

Figure 16: Report

4.6 Curse of Dimensionality

Principal Component Analysis (PCA) is an unsupervised, non-parametric statistical technique primarily used for dimensionality reduction in machine learning. PCA has been applied to the dataset to dealing with high dimensionality with different *n_components* parameters. PCA is supposed to affect it positively, but it has affected it negatively. We can see the results below. After applying PCA, our AUC for Gradient Boost Classifier decreased from 0.78 to 0.59(approximately). Therefore, it is not a good idea to apply PCA to this dataset.

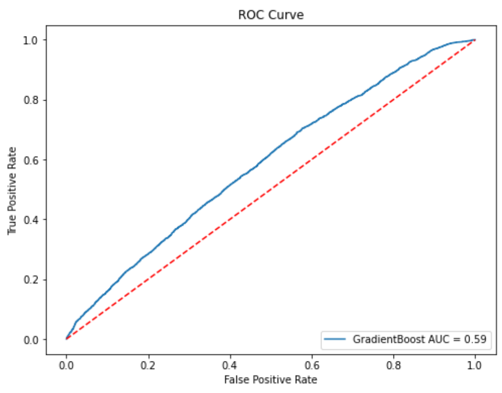
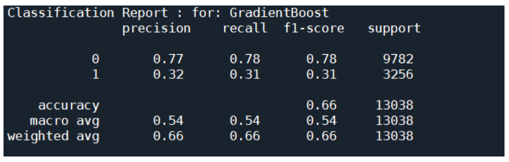


Figure 17: PCA AUC



```
Classification Report : for: GradientBoost
              precision    recall  f1-score   support

           0       0.77      0.78      0.78      9782
           1       0.32      0.31      0.31      3256

    accuracy                           0.66     13038
   macro avg       0.54      0.54      0.54     13038
weighted avg       0.66      0.66      0.66     13038
```

Figure 18: PCA Report

4.7 Comparision of Accuracy Scores

In below figure we can better see which learning algorithm is better for our dataset.

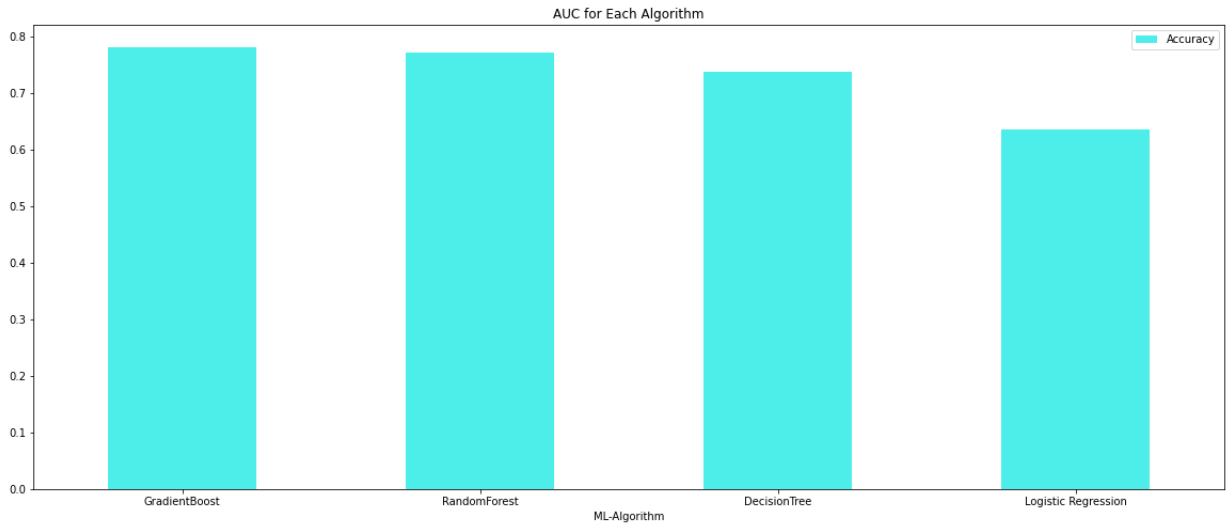


Figure 19: Comparision of Accuracy Scores

According to the above figure, Gradient Boost Classifier is the best option for our algorithm.

We have a 0.78 AUC score. It would be better if we had over 0.85 AUC score. This algorithm still needs to be improved to use for real life problem.

5 Conclusion

In conclusion, we tried to deal with Human Genetic Variant and make a prediction about whether a variant have conflicting. Due to this is a real-world problem our dataset was dirty and we had to clean up the data in preprocessing section. We dropped fully NaN, duplicate, and dummy features from our dataset. After that, we tried to fill NaN values with different approaches. At the end of the preprocessing, we applied OneHotEncoding, Mapping and Feature Hashing to transforming categorical values into numeric values.
After preparing our dataset, we applied four different classification algorithms to find the best algorithm. After applying all algorithms, we could not achieve as good results as we wanted. For improving our accuracy scores, we applied the PCA algorithm to reduce dimension and have better accuracy scores. Nevertheless, PCA would not be beneficial for our algorithms. As a result, we conclude that for this learning algorithm Gradient Boosting Classifier had the best accuracy score with the best parameter settings.