Diabetes Prediction using Data Mining

Smit Patel (hg4807)

A. Overview of the Problem

Diabetes is the 7th major cause of deaths in the United States[1] and across the world has left major medical issues and outcomes. There are three major types of diabetes including Type 1, Type 2 and Gestational diabetes. There have been errors in misclassifying the type of diabetes a patient might have in large numbers but even before that there have been non-diabetic patients having classified as diabetic and vice-versa. When a diabetic patient is classified as non-diabetic, it can lead to severe consequences due to no treatment taking place. Similarly for the case in which a non-diabetic patient is classified as diabetic, then it can lead to several complications especially if the patient is subjected to insulin injections with any need.

The reason behind the misclassification of diabetes or for that matter any other disease as such is vastly due to the overworked, overbooked doctors and at times lack of their understanding, communication (especially pre-test instructions) which is devastating to the patient. It is of paramount importance to make sure that correct classification takes place for diabetes as heart problems, visual impairment, foot issues, liver complications, amputations are few of the severe results of diabetes. In one of the findings by GP in England it found out that about 80 out of 1600 diabetic patients were misclassified and about 65-70 out of the 500 labelled[2] as diabetic would need to be re-checked for some sort of diagnostic error. Hence, making the use of Machine Learning & Data Mining Techniques can be a huge plus even if it outperforms the human performance in diagnosis by a bit. Diabetes can be misclassified as Coeliac Disease, Thyroid Disease, Mastopathy, Pancreatitis etc. by doctors.[3] There is a medical harm to the patients for the mistake and also lawsuits are filed against the medical professionals for the errors. Our aim is to try and help this problem by using Machine Learning and Data Mining Techniques.

A.1 Literature Overview

A lot of work has been done in the field of accurately predicting diabetic patients so as to reduce the errors and human efforts. They have used datasets from local hospitals to the online machine learning repositories like UCI. Different data preprocessing techniques such as imputation, feature scaling, feature reduction have been performed according to the necessity. And, various algorithms have been used such as Logistic Regression, Naïve Bayes, K-Nearest Neighbors as per intuition and performance. Work in relation with the Diabetes Prediction, that has been referred includes :

Hang Lai, Huaxiong Huang, Karim Keshavjee, Aziz Guergachi, Xin Gao on 15th October 2019 published "**Predictive Models for diabetes Mellitus using machine learning techniques**" and used Gradient Boosting Machine, Logistic Regression, Random Forest & Decision Trees for Classification.

"**A Novel Diabetes Healthcare Disease Prediction Framework Using Machine Learning Techniques**" by Raja Krishnamoorthi, Shubham Joshi, Hatim Z. Almarzouki, Piyush Kumar Shukla, Ali Rizwan, C. Kalpana and Basant Tiwari using Logistic Regression, K-Nearest Neighbors, Support Vector Machine, Random Forest and Proposed Logistic with hyperparameters finetuned.

"**Research on Diabetes Prediction Method Based on Machine Learning**" by Jingyu Xue, Fanchao Min, Fengying Ma using the techniques Support Vector Machine, Naïve Bayes Classifier and Light GBM.

"**Prediction and diagnosis of future diabetes risk : a machine learning approach**" by Roshan Birjais, Ashish Kumar Mourya, Ritu Chauhan & Harleen Kaur using Logistic Regression, Naïve Bayes and Gradient Boosting.

B. Dataset Overview

The dataset that has been used for building an accurate model is the PIMA INDIANS Diabetes Dataset[4]. Following are the facts about the dataset that provides its overview :-

- All the information in the dataset is of females over the age of 21 from PIMA INDIANS heritage.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Fig.1 Representation of Dataset

- There are 9 columns in the dataset. The 9th out of all is the ‘Outcome’, whose values are 0 and 1, representing non-diabetic and diabetic records respectively.

- There are 8 deterministic features which determine the output or ‘Outcome’ namely ‘Pregnancies’, ‘Glucose’, ‘Blood Pressure’, ‘Skin Thickness’, ‘Insulin’, ‘BMI’, ‘Diabetes Pedigree Function’ and ‘Age’.

- In total there are 768 records out of which 500 are of non-diabetic people and 268 rest are of the diabetic patients.

C. Methods

After importing the dataset the first job was to know the dataset, and apply any data-preprocessing techniques in case there was a need for so. Below is a flow of the whole procedure and we’d look into detail on each and every step.

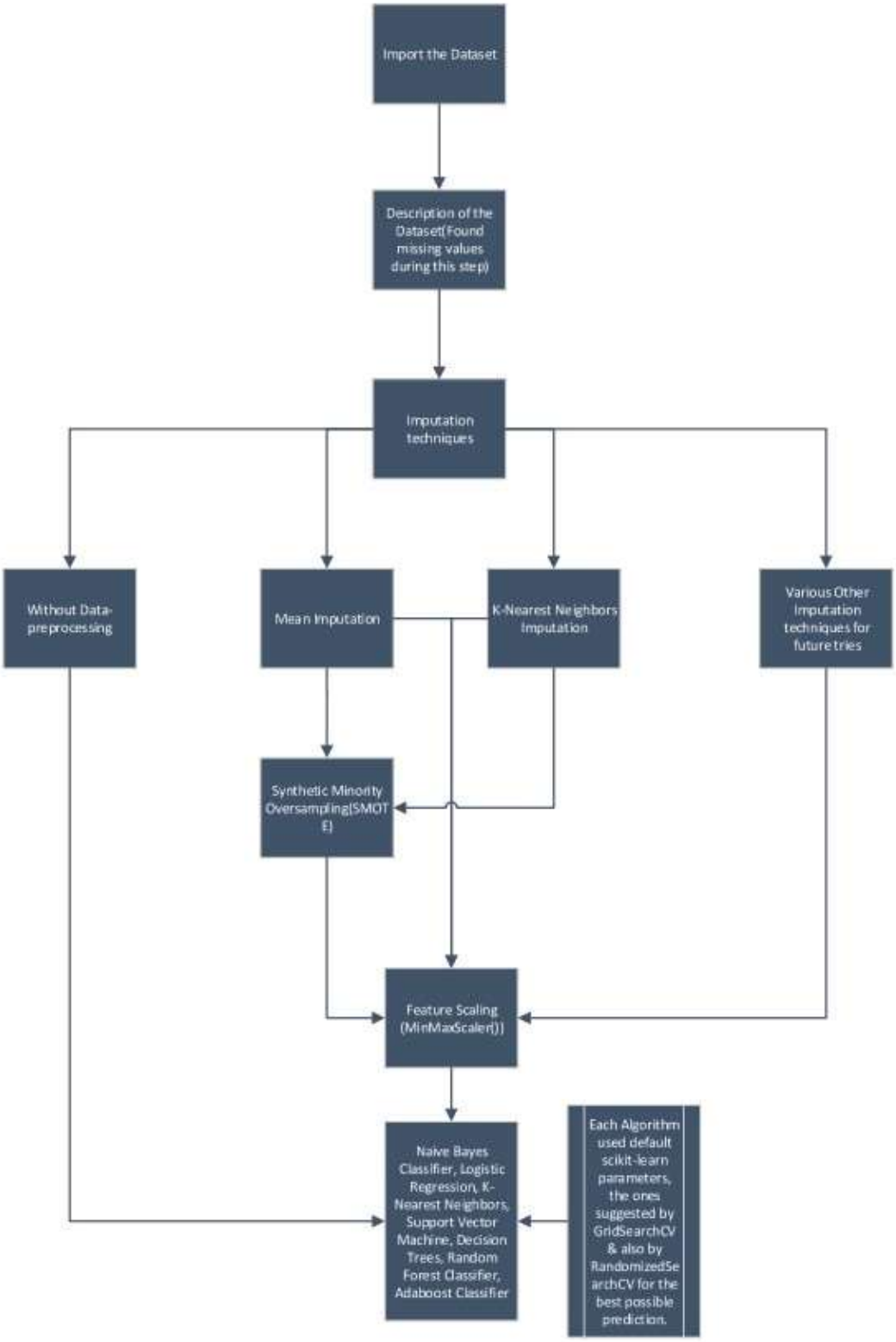


Fig.2 Flow-Diagram for Building the most accurate model.

While looking at the description of dataset using ‘describe()’ of pandas’ dataframe, the minimum values of columns such as ‘Pregnancy’, ‘Glucose’, ‘Blood Pressure’, ‘Skin Thickness’, ‘Insulin’, and ‘BMI’ were observed as zero.

Here’s the code snippet that explains the above statement;

	preg	glu	bp	skinThickness	insulin	bmi	dpf	age	outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Fig.3 Description of Dataset (‘min’ showed the missing values in some columns)

Now, for the ‘Pregnancy’ column it is valid for the minimum value to be zero as there can be no pregnancies for certain females. But, the other 5 columns including ‘Glucose’, ‘Blood Pressure’, ‘Skin Thickness’, ‘Insulin’ and ‘BMI’ being zero as minimum are indicators that those columns have missing values.

Now, to find out the number of missing values in those 5 columns a function was defined and used which can be seen below :

```
# Let's have a look at the number of  missing values in the dataframe

columns_missing_values = ['glu', 'bp', 'skinThickness', 'insulin', 'bmi', 'dpf']

def zeroes_in_columns(columns, dataframe):
    for missing_values_column in columns:
        print(f'Number of missing values in {missing_values_column} are : \
        {len(dataframe[dataframe[missing_values_column]==0])}')


zeroes_in_columns(columns_missing_values, df)
```

```
Number of missing values in glu are : 5
Number of missing values in bp are : 35
Number of missing values in skinThickness are : 227
Number of missing values in insulin are : 374
Number of missing values in bmi are : 11
Number of missing values in dpf are : 0
```

Fig.4 Number of missing values

Hence, 'Glucose', 'Blood Pressure', 'Skin Thickness', 'Insulin' and 'BMI' had 5, 35, 227, 374 & 11 values as missing respectively.

The task at hand was that of imputation after finding missing values. The data distributions using histogram plots for each column to see how the data was distributed in each column, were also plotted to see how the data was distributed in each column.

In order to see if any of the features could be dropped though there isn't a very high number of columns in the dataset, correlation between each pair of dataset was looked at using the 'corr()' function of pandas' dataframe. But, the highest correlation was 0.544341 between 'age' and 'preg'. It didn't seem enough to drop either of the columns.

Here's the table of correlation :

	preg	glu	bp	skinThickness	insulin	bmi	dpf	age	outcome
preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
glu	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
bp	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
skinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
bmi	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
dpf	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

Fig. 5 Correlation Table and Highest value of it.

C.1 Performance Metrics :-

Performance of the Model was determined using 'Accuracy' and 'Area Under Receiver Operating Characteristic Curve'(i.e. Auroc score). 'Auroc' score was chosen as one of the metrics as it indicates the discriminative capability of a model.

And, in this particular problem it is important that a diabetic patient gets classified as diabetic and a non-diabetic as it is. Hence, 'Auroc' was also one of the model performance determining metrics.

C.2 Different Approaches :-

Now, there were 5 different approaches that were used from this point. One of the 5, was out of interest to see how the prediction was even without any sort of data pre-processing. Nevertheless, the 5 approaches were :-

- Applying ML Algorithms without Data Pre-processing.
- Mean Imputation -> Feature Scaling -> ML Algorithms.
- KNN Imputation -> Feature Scaling -> ML Algorithms.
- Oversampling Mean Imputed Data -> Feature Scaling -> ML Algorithms.
- Oversampling KNN Imputed Data -> Feature Scaling -> ML Algorithms.

All of the above mentioned approaches are also evident in the Flow Diagram in Fig.2 . Let's go through each approach in detail and then we'll look at the results and comparison of performance of each of them in the next section of 'Analysis Results and Performance'.

→ Applying ML Algorithms without Data Pre-processing :-

Initially, for looking at how the prediction was without imputation; various algorithms such as Naive Bayes, K-Nearest Neighbors, Logistic Regression, Support Vector Machine, Decision Trees, Random Forest Classifier, AdaBoost were used.

Now, one important point to note here is :- **For each of the above mentioned Algorithms,** the default parameters suggested by scikit-learn, the best set of parameters suggested by Grid Search Cross Validation & the best set of parameters suggested by Randomized Search Cross Validation were used.

One of the examples that demonstrates what is stated above is :- (Example with DecisionTrees is shown)

Training the model **using default parameters** that scikit-learn suggests :-

```
decision_tree = DecisionTreeClassifier(random_state=42)

decision_tree.fit(X_train, y_train)

y_pred = decision_tree.predict(X_test)

accuracy_dt_normal_1 = accuracy_score(y_test, y_pred)*100

auroc_dt_normal_1 = roc_auc_score(y_test, y_pred)*100

print(f"{accuracy_dt_normal_1}% is the accuracy score in percentage form with DecisionTreesTechnique.")
```

```
70.995670995671% is the accuracy score in percentage form with DecisionTreesTechnique.
```

```
decision_tree.get_params()
```

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 42,
 'splitter': 'best'}
```

Fig.6 Default parameters of scikit-learn used for DecisionTrees.

Now, for **Hyperparameter tuning** using **GridSearchCV**, training the model with best suggested parameters from GridSearchCV and testing the 'X_test' **:-**

GridSearchCV (DecisionTreesClassifier)

```
decision_tree = DecisionTreeClassifier()

params_grid_dict = {
    'max_depth' : np.array([i for i in range(2,20)]),
    'criterion' : np.array(['gini', 'entropy']),
    'min_impurity_decrease': np.arange(0.01, 0.06, 0.01),
    'min_samples_split': np.arange(10,100,10)
}

accuracy_dt_grid_1, auroc_dt_grid_1  = grid_search_cv(decision_tree, params_grid_dict,
               X_train=X_train,
               y_train = y_train,
               X_test = X_test,
               y_test = y_test)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=9,
                       min_impurity_decrease=0.01, min_samples_split=50) is the Best Estimator according to Exhaustive Hyperparam

77.92207792207793% is the accuracy score for the DecisionTreeClassifier() using the best hyparameters suggested by GridSearchCV.
74.6696212731668% is the auroc score for the DecisionTreeClassifier() using the best hyparameters suggested by GridSearchCV.
73.61111111111111% is the precision score for the DecisionTreeClassifier() using the best hyparameters suggested by GridSearchCV.
62.35294117647059% is the recall score for the DecisionTreeClassifier() using the best hyparameters suggested by GridSearchCV.
67.51592356687898% is the f1_score for the DecisionTreeClassifier() using the best hyparameters suggested by GridSearchCV.
```

Fig.7 Hyperparameters Suggested By GridSearchCV used for training.

Training the model with best suggested parameters from **RandomizedSearchCV**, and testing the 'X_test' :-

RandomizedSearchCV (DecisionTrees):-

```
decision_tree = DecisionTreeClassifier()

params_grid_dict = {
    'max_depth' : np.array([i for i in range(2,40)]),
    'criterion' : np.array(['gini', 'entropy']),
    'min_impurity_decrease': np.arange(0.01, 1.05, 0.01),
    'min_samples_split': np.arange(10,100,10)
}

accuracy_dt_random_1, auroc_dt_random_1 = randomized_search_cv(decision_tree,params_grid_dict,X_train=X_train,
                  y_train=y_train, X_test=X_test, y_test=y_test)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=15,
                       min_impurity_decrease=0.01, min_samples_split=50) is the Best Estimator according to Randomized Hyperparamet

77.92207792207793% is the accuracy score for DecisionTreeClassifier() using the best hyparameters suggested by RandomizedSearchCV.
74.6696212731668% is the auroc score for DecisionTreeClassifier() using the best hyparameters suggested by RandomizedSearchCV.
73.61111111111111% is the precision score for DecisionTreeClassifier() using the best hyparameters suggested by RandomizedSearchCV.
62.35294117647059% is the recall score for DecisionTreeClassifier() using the best hyparameters suggested by RandomizedSearchCV.
67.51592356687898% is the f1_score for DecisionTreeClassifier() using the best hyparameters suggested by RandomizedSearchCV.
```

Below values are to be used for comparison purpose in the bar plot :-

```
decisionTreeAcc1 = max(accuracy_dt_normal_1, accuracy_dt_grid_1, accuracy_dt_random_1)

decisionTreeAuroc1 = max(auroc_dt_normal_1, auroc_dt_grid_1, auroc_dt_random_1)
```

Fig.8 Hyperparametrs suggested by RandomizedSearchCV used for training.

Note :-

- The splitting of dataset was in the ratio 7:3 for training and testing dataset. And, the testing dataset had 85 diabetic datapoints & 146 non-diabetic datapoints.
- In some of the cases have used random parameters in place of by-default parameters' strategy.
- The average of 'auroc' score would be taken into account for finding the best set of parameters in both the cases (i.e. GridSearchCV and RandomizedSearchCV).
- '5-fold' cross-validation has been performed. The reason for not increasing folds is not having a large number of records in the dataset.

Functions 'randomized_search_cv()' & 'grid_search_cv()' have been used for hyperparameter tuning as we can see in the Fig.7 & Fig.8 . Now, there are no such functions in the in-built scikit-learn package.

Those two functions were defined by me, in order to not repeat the code.

The **job** of **'**grid_search_cv()**' :**

- To take 'estimator', 'param_grid' (i.e. dictionary of parameters and its values), X_train, y_train, X_test and y_test as input parameters.

- Use X_train and y_train for performing Grid Search Cross Validation using 'GridSearchCV()' class from scikit-learn.

- Use the best set of parameters suggested by GridSearchCV(). And, then train the model using those parameters.

- Finally test the X_test and check the performance metrics 'Accuracy' and 'Auroc' scores. (though all the performance metrics' values were printed to notice.)

The **job** of 'randomized_search_cv()' is similar to 'grid_search_cv()' with the difference being, RandomizedSearchCV() class was used instead.

The best of all the three results (i.e. using default parameters, Grid Search CV suggested parameters, Randomized Search CV suggested parameters) is eventually used for comparing with other algorithms' best performance. The bar plot of performance will be shown in the "Analysis Results and Performance" section.

→ Mean Imputation -> Feature Scaling -> ML Algorithms :-

Now, having already tried applying Machine Learning without Mean Imputation, now was the turn for imputing and then predicting. Hence, Mean Imputation Technique was used for imputing the missing values in all 5 columns using the SimpleImputer() class in sklearn.impute.

Mean Imputation : In a particular column of the dataset all the missing values would be imputed/substituted with the average of non-missing values in the column. This is indeed introducing misinformation in the column; but it works often.

Thereafter, since Support Vector Machine, K-Nearest Neighbors, Logistic Regression would require feature scaling; the features were scaled in the mean imputed data using MinMaxScaler() class in scikit-learn.

Having performed imputation and feature scaling all the 7 algorithms were again used for prediction. And, as in the case before hyperparameter tuning was done for all ML techniques

using GridSearchCV and RandomizedSearchCV in each 7 cases. Eventually, the barplot with performance of the best of all algorithms was plotted to catch the best performing model.

→ KNN Imputation -> Feature Scaling -> ML Algorithms :-

After Mean Imputation, now was the turn to use one more imputation technique to see if prediction was improving than before. Hence, KNN Imputation was tried.

KNN Imputation :- It is a multivariate imputation technique as all the variables/features/columns are involved in imputing the missing value. Herein, the missing value would be imputed as K-Nearest neighbors' that column's average.

Then, features were once again scaled using the MinMaxScaler() class. And, all the 7 ML algorithms were applied as before; with hyperparameter tuning in each algorithm using GridSearchCV and RandomizedSearchCV, also using default parameters.

–> Oversampling Mean Imputed Data -> Feature Scaling -> ML Algorithms :-

In this dataset, 65.10 % of the data is of non-diabetic females and the rest 34.90 % belongs to diabetic female patients. There isn't a high imbalance in the dataset between both the classes as some ratio of 8:2 or 9:1 .

Now, oversampling the minority samples isn't a seeming requirement. And, also it would introduce unknown information in the data. Yet, oversampling of minority class was done with an idea that the model would be better trained with minority samples and its ability to discriminate as well as to predict correctly would significantly enhance.

The technique that was used for oversampling was SMOTE (Synthetic Minority Oversampling Technique). The way this technique works is :

Step 1 : Pick one of the samples in the minority class randomly.
Step 2 : Pick one of the other datapoint from the minority class randomly.
Step 3 : On the Line Segment connecting both the datapoints mentioned in Step 1 and Step 2, generate samples synthetically.
Step 4 : Repeat the above three steps as long as the number of samples in minority samples does not become equal to them in majority class.

After, oversampling the minority class in training data, feature scaling was done with the help of MinMaxScaler() class in scikit-learn.

Now, the data was ready to be used to train the models. Hence, it was used to train the Machine Learning models and then X_test was tested to notice the performance difference.

The important point to note here is, Mean Imputed data was oversampled here using SMOTE. Because, mere Mean Imputation performed better than mere KNN Imputation. However, KNN Imputation was also oversampled to see if it'd still after oversampling outperform SMOTE Mean Imputation, as a rarity.

–> Oversampling KNN Imputed Data -> Feature Scaling -> ML Algorithms :-

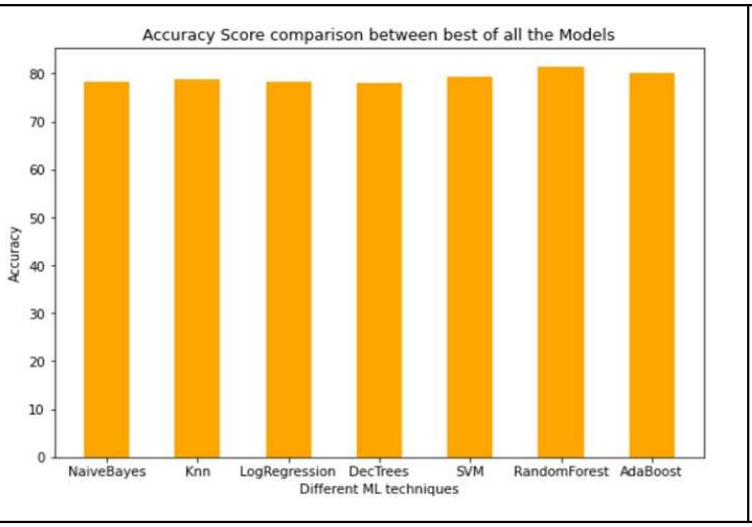
Herein, the minority class of training dataset of KNN imputed data was oversampled using SMOTE technique. And, then the feature scaling was done with MinMaxScaler() scaler.

As before, all the algorithms were then used to train the models on the Oversampled KNN Imputed data. And, while using each of the algorithms; for once default parameters, Grid Search CV suggested parameters and Randomized Search CV suggested parameters were used.

D. Analysis Results and Comparison :-

→ Applying ML Algorithms without Data Pre-processing :-

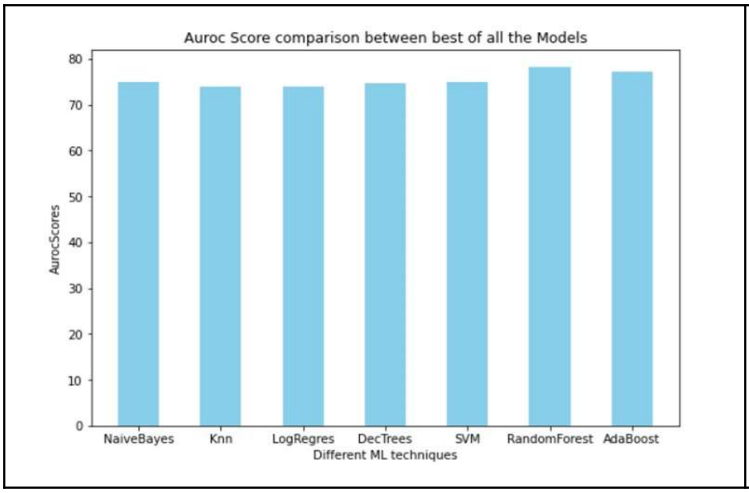
The performance of all supervised algorithms on un-imputed and un-preprocessed data in terms of accuracies was :-



Accuracy scores w/o Data-Preprocessing :	
Naïve Bayes	78.36%
Knearest Neighbors	78.79%
Logistic Regression	78.35%
Decision Trees	77.92%
Support Vector Machine	79.22%
Random Forest Classifier	81.39%
Adaboost Classifier	80.09%

Fig.9 Accuracies without any sort of Data-preprocessing

Performance in terms of Auroc scores of all Algorithms without Data Preprocessing is :-

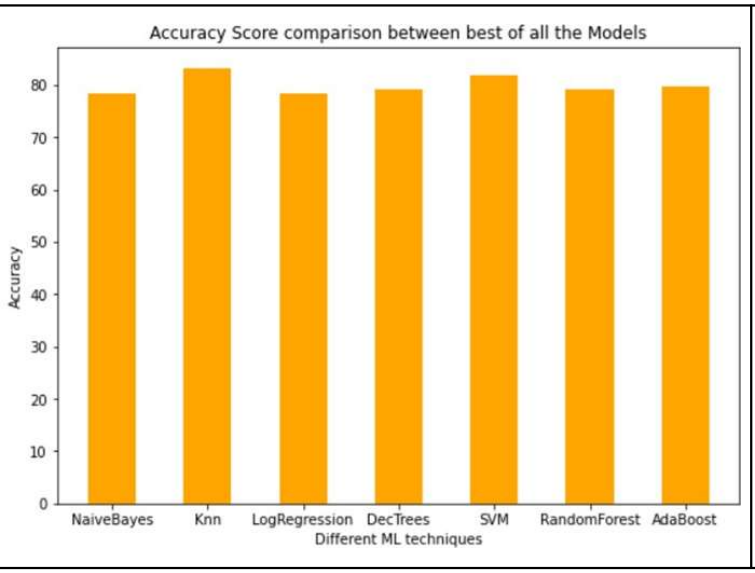


Auroc scores w/o Data-Preprocessing :	
Naïve Bayes	75.01%
Knearest Neighbors	73.88%
Logistic Regression	74.03%
Decision Trees	74.67%
Support Vector Machine	74.96%
Random Forest Classifier	78.15%
Adaboost Classifier	77.12%

Fig.10 AUROC scores without any sort of Data-preprocessing

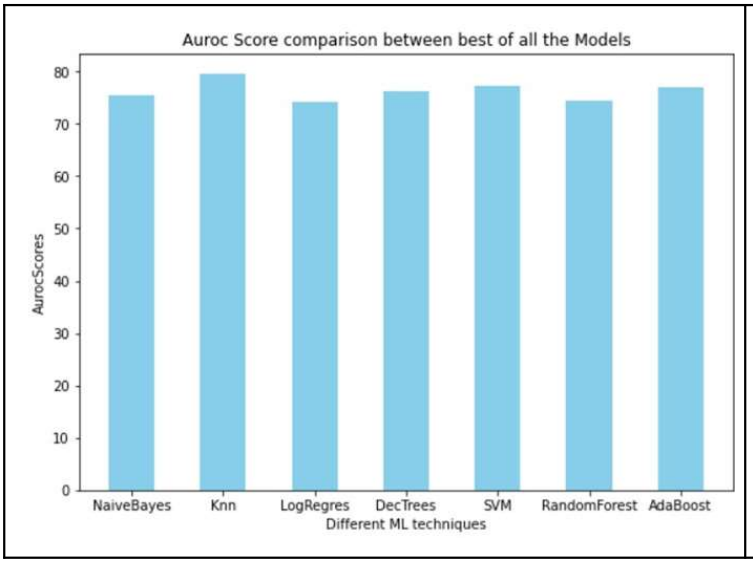
Result & Comparison :- As we can see, RandomForestClassifier is the best performer with the Accuracy of 81.39% and Auroc score of 78.15%. This result was obtained with default parameters from scikit-learn.

→ Mean Imputation -> Feature Scaling -> ML Algorithms :-



Accuracy scores after Mean Imputation :	
Naïve Bayes	78.35%
Knearest Neighbors	83.12%
Logistic Regression	78.35%
Decision Trees	79.22%
Support Vector Machine	81.81%
Random Forest Classifier	79.22%
Adaboost Classifier	79.65%

Fig.11 Accuracy scores after Mean Imputation.



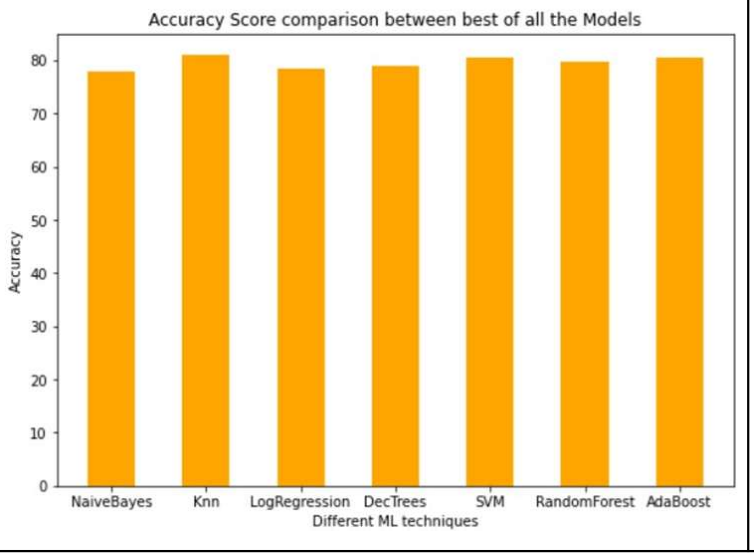
Auroc scores after Mean Imputation :	
Naïve Bayes	75.50%
Knearest Neighbors	79.52%
Logistic Regression	74.27%
Decision Trees	76.18%
Support Vector Machine	77.26%
Random Forest Classifier	74.47%
Adaboost Classifier	77.02%

Fig.12 AUROC scores after Mean Imputation.

Result & Comparison :- As we can see, K-Nearest Neighbors is the best performer with the Accuracy of 83.12% and Auroc score of 79.52%. This result was obtained with parameters suggested by Grid Search CV and they were 'n_neighbors'=23, 'weights'=distance.

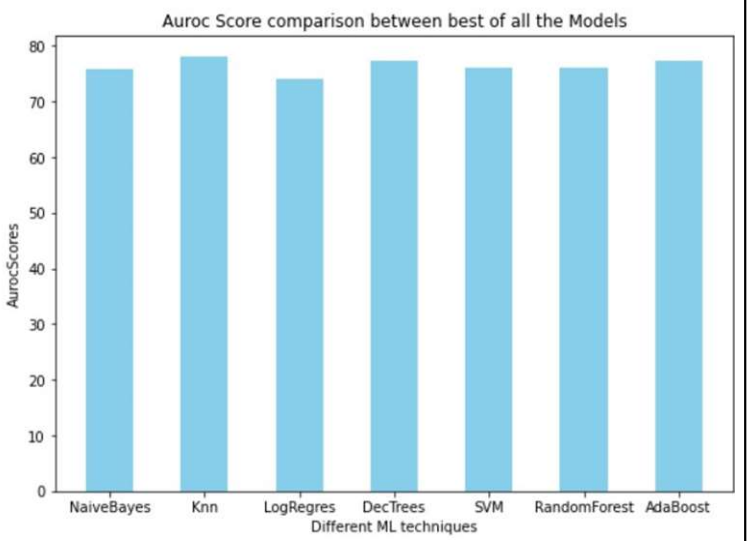
Before :-	Random Forest Classifier	Accuracy = 81.39%	Auroc = 78.15%
After :-	K-Nearest Neighbors	Accuracy = 83.12%	Auroc = 79.52%

→ KNN Imputation -> Feature Scaling -> ML Algorithms :-



Accuracy scores after KNN Imputation :	
Naïve Bayes	77.92%
Knearest Neighbors	80.95%
Logistic Regression	78.35%
Decision Trees	78.79%
Support Vector Mach	80.52%
Random Forest Class	79.65%
Adaboost Classifier	80.52%

Fig.13 Accuracy scores after KNN Imputation.



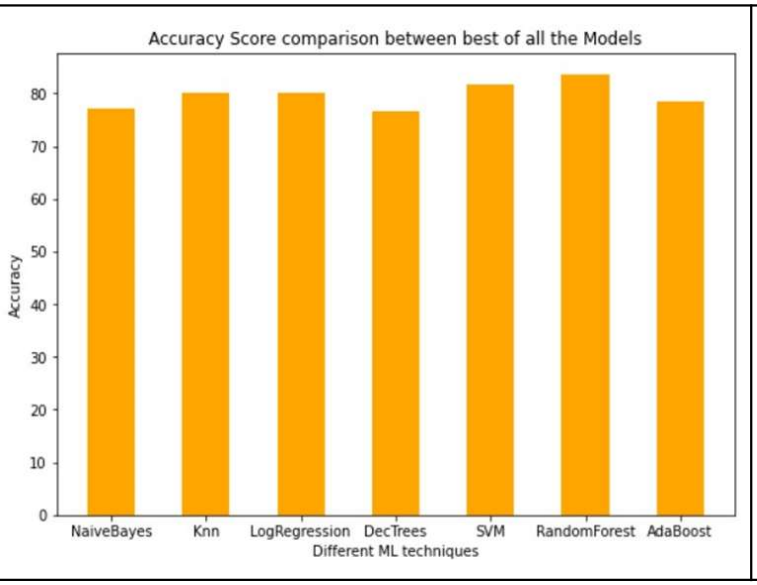
Auroc scores after KNN Imputation :	
Naïve Bayes	75.90%
Knearest Neighbors	78.04%
Logistic Regression	74.02%
Decision Trees	77.32%
Support Vector Mach	75.99%
Random Forest Class	76.03%
Adaboost Classifier	77.46%

Fig.14 AUROC scores after KNN Imputation.

Result & Comparison :- As we can see, K-Nearest Neighbors is the best performer with the Accuracy of 80.95% and Auroc score of 78.04%. This result was obtained with default params.

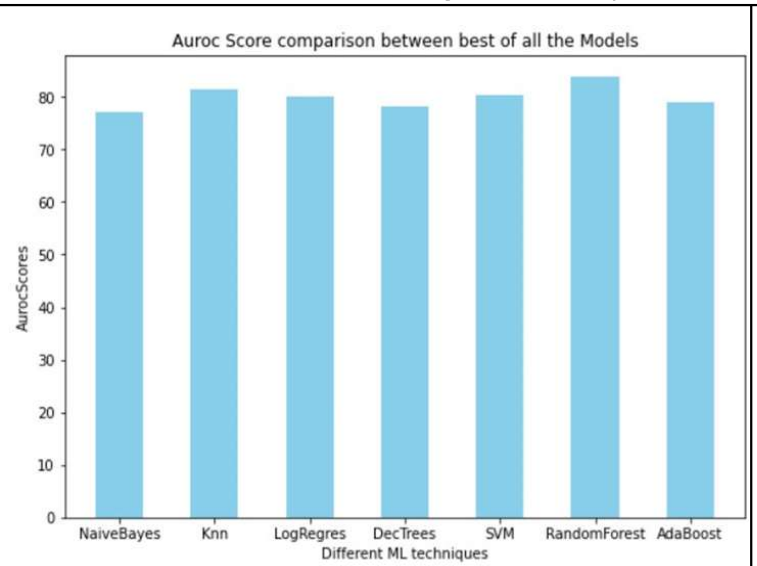
w/o Data Preprocessing :	Random Forest Classifier	Accuracy = 81.39%	Auroc = 78.15%
After Mean Imputation :	K-Nearest Neighbors	Accuracy = 83.12%	Auroc = 79.52%
After KNN Imputation :	K-Nearest Neighbors	Accuracy = 80.95%	Auroc = 78.04%

–> Oversampling Mean Imputed Data -> Feature Scaling -> ML Algorithms :-



Accuracy scores after SMOTE Mean Imputation :	
Naïve Bayes	77.06%
Knearest Neighbors	80.09%
Logistic Regression	80.09%
Decision Trees	76.62%
Support Vector Machine	81.82%
Random Forest Classifier	83.55%
Adaboost Classifier	78.35%

Fig.14 Accuracy scores after SMOTE of Mean Imputation.



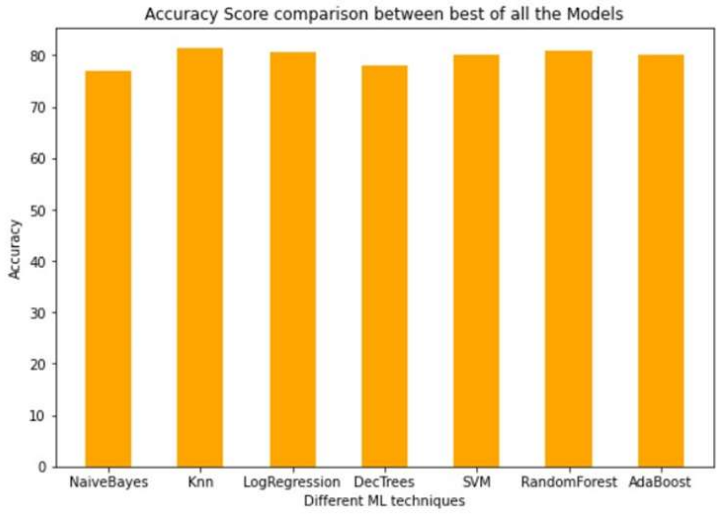
Auroc scores after SMOTE Mean Imputation :	
Naïve Bayes	77.17%
Knearest Neighbors	81.29%
Logistic Regression	80.07%
Decision Trees	78.07%
Support Vector Machine	80.26%
Random Forest Classifier	83.79%
Adaboost Classifier	78.94%

Fig.15 AUROC scores after SMOTE of Mean Imputation.

Result & Comparison :- As we can see, RandomForestClassifier is the best performer with the Accuracy of 83.55% and Auroc score of 83.79%. This result was obtained with RandomizedSearchCV's suggested parameters: 'max_depth'=13, 'min_impurity_decrease'=0.01, 'min_samples_split'=40, 'n_estimators'=90.

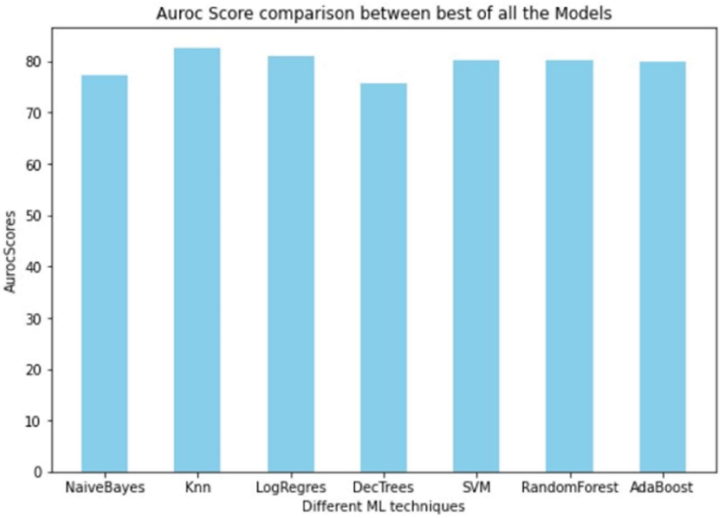
w/o Data Preprocessing :	Random Forest Classifier	Accuracy = 81.39%	Auroc = 78.15%
After Mean Imputation :	K-Nearest Neighbors	Accuracy = 83.12%	Auroc = 79.52%
After KNN Imputation :	K-Nearest Neighbors	Accuracy = 80.95%	Auroc = 78.04%
Oversampling Mean Imp :	Random Forest Classifier	Accuracy = 83.55%	Auroc = 83.79%

–> Oversampling KNN Imputed Data -> Feature Scaling -> ML Algorithms :-



Accuracy scores after SMOTE KNN Imputation :	
Naïve Bayes	77.06%
Knearest Neighbors	81.38%
Logistic Regression	80.52%
Decision Trees	77.92%
Support Vector Machine	80.09%
Random Forest Classifier	80.95%
Adaboost Classifier	80.09%

Fig.16 Accuracy scores after SMOTE of KNN Imputation.



Auroc scores after SMOTE KNN Imputation :	
Naïve Bayes	77.42%
Knearest Neighbors	82.57%
Logistic Regression	80.90%
Decision Trees	75.65%
Support Vector Machine	80.31%
Random Forest Classifier	80.31%
Adaboost Classifier	80.06%

Fig.17 AUROC scores after SMOTE of KNN Imputation.

Result & Comparison :- As we can see, K-Nearest Neighbors is the best performer with the Accuracy of 81.38% and Auroc score of 82.57%. This result was obtained with RandomizedSearchCV's suggested parameters: 'metric'=euclidean , 'n_neighbors'=7, 'weights'=distance..

w/o Data Preprocessing :	Random Forest Classifier	Accuracy = 81.39%	Auroc = 78.15%
After Mean Imputation :	K-Nearest Neighbors	Accuracy = 83.12%	Auroc = 79.52%
After KNN Imputation :	K-Nearest Neighbors	Accuracy = 80.95%	Auroc = 78.04%
Oversampling Mean Imp :	Random Forest Classifier	Accuracy = 83.55%	Auroc = 83.79%
Oversampling KNN Imp :	K-Nearest Neighbors	Accuracy = 81.38%	Auroc = 82.57%

Since, Mean Imputation of data worked better than KNN Imputation, Oversampling of Mean Imputation was done with an idea to better train the model with minority samples. And, as expected; it gave the best results with an Accuracy of 83.55% and AUROC score of 83.79% with Random Forest Classifier.

E. Conclusion:-

- After performing Mean Imputation for dataset & then oversampling the minority class of training data using SMOTE technique, scaling down the features using MinMaxScaler(), and using Random Forest Classifier with the parameters as :

- 'max_depth'=13, 'min_impurity_decrease'=0.01, 'min_samples_split'=40, 'n_estimators'=90, would give the best results for prediction with the highest value of Accuracy and AUROC scores as 83.55% and 83.79% respectively.

- This is post performing five different preprocessing approaches, and using seven Machine Learning Algorithms each time, with each Algorithm using default, suggested by Grid Search CV and Randomized Search CV parameters to know which one combination of parameters works the best.

F. Repository:-

GitHub link [5] to the code and results of the project.

G. Future Work:-

- Try some out of the box data preprocessing techniques to better predictive perfection. Also, see if some different imputation methods such as 'Median Imputation' apart from the tried can work better.

- The current work focuses on predicting correctly if a patient is diabetic or otherwise. Future work can be on predicting if it is Type-1, Type-2 or Gestational Diabetes. The reason for so, is that Type-1 and Type-2 are commonly mistaken and it creates a medical severity for patients as they both have different approaches of treatment altogether.

- Try to use the same model on a different diabetes dataset to see if it is able to perform well there and hence generalizable.
- Make a Web App with some frontend and with python in backend, to make the use of work.

H. **References:-**

[1] From the Centers for Disease Control and Prevention https://www.cdc.gov/diabetes/data/statistics-report/index.html - National Diabetes Statistics Report.

[2] Review reveals thousands ‘wrongly diagnosed diabetic’ by BBC UK - https://www.bbc.com/news/health-12612344

[3] By an Advocacy Firm that helps medical patients sue professionals for loss due to wrong diagnosis - https://www.hamptonking.com/blog/how-to-avoid-a-diabetes-misdiagnosis-diabetes-awareness-month/

[4] Dataset downloaded from Kaggle - https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database?page=2

[5] GitHub link to the project code - https://github.com/Patel27-S/Diabetes-Prediction---Data-Mining