Machine Learning Engineer Nanodegree

Capstone Project

Ashwinder Singh Arora
December 7th, 2019

I. Definition

Project Overview

This project addresses a rising modern problem from the medical domain. Diabetes is one of the deadliest diseases which causes an imbalance of blood sugar. As of 2019, an estimated 463 million people have diabetes worldwide.[1] This represents around 10 percent of the adult population, and according to recent studies the trend suggests that rates will continue to rise. Diabetes at least doubles a person's risk of early death. According to the International Diabetes Federation (IDF) 1 in 2 adults with diabetes are undiagnosed. In 2017 alone diabetes resulted in approximately 3.2 to 5.0 million deaths worldwide. Pre-screening of the people with diabetes can help in preventing the disease from worsening and will save many lives.

The goal of this project is to be able to predict if a person has diabetes or not using the medical attributes provided in the UCI PIMA Indian Diabetes Database[2]. This is a binary classification problem. The database consists of 8 medical attributes of 768 females of the PIMA Indians heritage. In this project I have performed exploratory data analysis on the PIMA Indians Diabetes Database and used supervised learning techniques for binary classification to predict if a person is diabetic or not.

Problem Statement

The problem to be solved here is to predict if a person has diabetes with accuracy while avoiding false negatives. Although the research carried in this project will not be a replacement for a lab test, it can surely help in pre-screening people. The problem will be solved with the help of supervised learning algorithms to classify the population into two classes- positive and negative. Here's the outline of what is done in the project-

1. Perform EDA on the data to inspect the statistical features
2. Visualise the data using plots to get more insight into the data
3. Perform data pre-processing

4. Fit different classification models and compare their results
5. Improving performance using hyper-parameter tuning
6. Compare the results with unoptimised models' results

An attempt will be made to reach as close as possible to the scores of the benchmark model which has a maximum F1-score of 0.76 and an accuracy of 76.30 percent. For this project, the data is sourced from Kaggle[3] while the data used in the benchmark model is sourced from UCI Machine Learning Repository[4]. The dataset from Kaggle has been modified to make it dirtier and to make it a bit more challenging.

Metrics

The F1-score is one of the main metrics used to evaluate the performance of the models. The F1 score is the harmonic mean of precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. Mathematically, it can be represented as

$$F_1\ Score = \frac{2 \times precision \times recall}{precision + recall}$$

where,

$$precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

This is a case where detecting if a person has diabetes accurately is more important than detecting if a person does not have diabetes accurately, and hence the F1-score is favoured instead of accuracy as a measure of performance.

II. Analysis

Data Exploration

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases and is available to the public domain on kaggle.com. The dataset consists of several medical predictor (independent) variables and one target (dependent) variable, Outcome.

Below is a brief description of the data.

Format	Comma Separated Values (csv)
Number of instances	768
Number of attributes	8 + 1 (class label)
Number of classes	2
Feature 1	Number of pregnancies
Feature 2	Plasma glucose concentration, 2 hours in an oral glucose tolerance test
Feature 3	Diastolic blood pressure (mm Hg)
Feature 4	Triceps skin fold thickness (mm)
Feature 5	2-Hour serum insulin (mu U/ml)
Feature 6	Body mass index (weight in kg/(height in m)^2)
Feature 7	Diabetes pedigree function
Feature 8	Age (years)
Class Variable	Outcome (0 or 1)

The data consists of 768 females of age 21 years and above. Out of these 500 are non-diabetic (class 0) while 268 are diabetic (class 1). The database does not mention any missing values, however on inspecting the statistical features of the data it becomes clear that some features have 0 values which are not possible in real life, for example there are zero values in the columns glucose, BMI, skin thickness, blood pressure and serum insulin levels. These are anomalies in the data and need to be addressed before fitting any model. On counting these zero values for each column it is observed that there are a total of 5 in Glucose, 35 in Blood Pressure, 227 in Skin Thickness, a whopping 374 in Serum Insulin, and 11 in BMI.

Exploratory Visualization

Histogram and scatter plots are used to observe the distribution and correlation of the data. Some of the interesting results we see are from the below histograms:

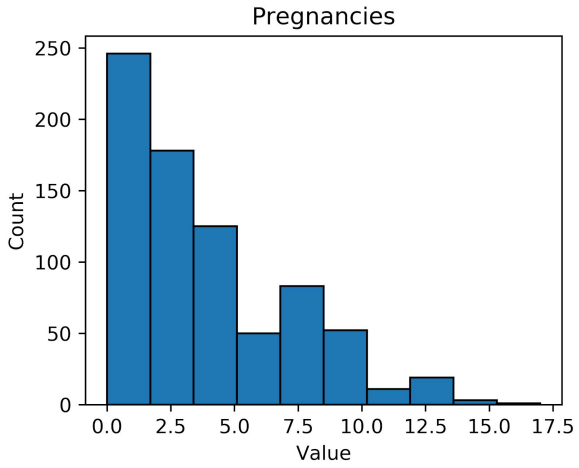

Fig 1. Number of pregnancies distribution

Most women became pregnant 0-3 times which can also be observed when viewing the statistical description of the data as the median value for pregnancies is 3.

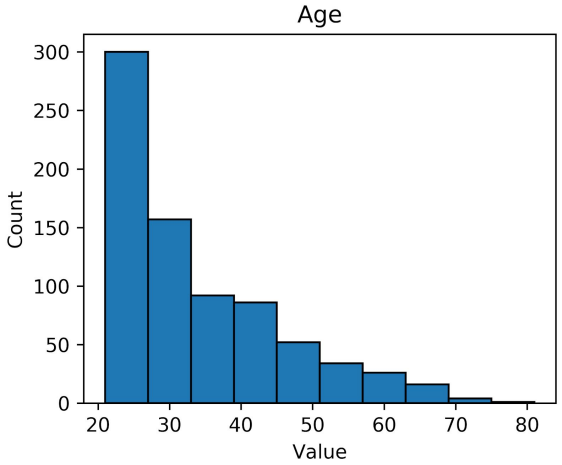

Fig 2. Age distribution

Women aged between 21-30 constitute around 58 percent of the population while around 75 percent of the total women are aged below 45.

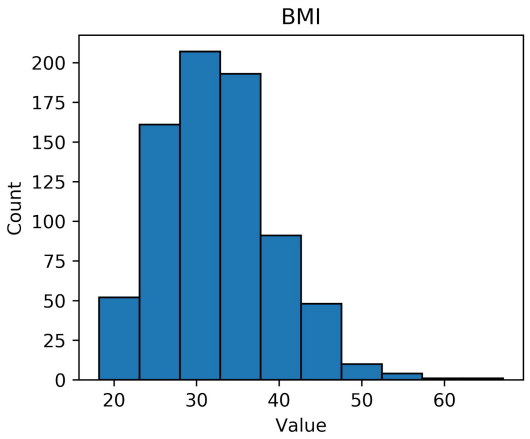

Fig 3. BMI distribution

Considering the fact that healthy BMI ranges from 18.5 to 24.9, we see that most of the women lie in the overweight or obese category with only around 25 percent of the population below a BMI of 30.

A heatmap and scatter matrix are plotted to observe the correlation between various features and some interesting results are observed.

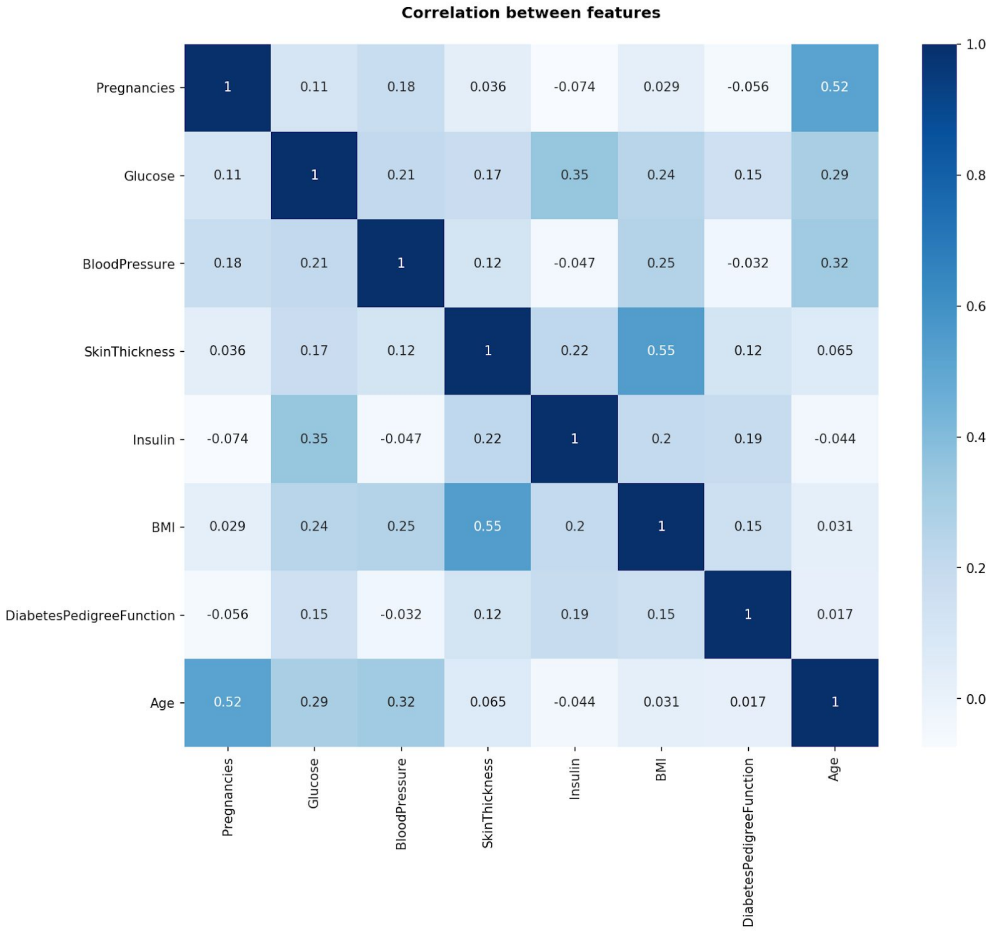

Fig 4. Correlation between features

A strong positive correlation is observed between age and number of pregnancies, and skin thickness and BMI. Age and blood pressure, and serum insulin and glucose levels also show a good correlation. BMI and blood pressure also seem to be moderately correlated.

Algorithms and Techniques

The benchmark model used Gaussian Naive Bayes, SVM, and Decision Trees. This project uses Logistic Regression, Decision Trees and AdaBoost for classification.

Logistic Regression is imported from sklearn's linear_model library, the Decision Tree Classifier is imported from from sklearn's tree library, and AdaBoost is imported from the sklearn's ensemble library.

Logistic Regression is one of the most simple and commonly used Machine Learning algorithms for two-class classification. It is used as the baseline for simple binary classification problems. Logistic regression describes and estimates the relationship between one dependent binary variable (label) and independent variables (features). Logistic Regression is estimated using Maximum Likelihood Estimation (MLE) approach. Maximizing the likelihood function determines the parameters that are most likely to produce the observed data. From a statistical point of view, MLE sets the mean and variance as parameters in determining the specific parametric values for a given model. This set of parameters can be used for predicting the data needed in a normal distribution.

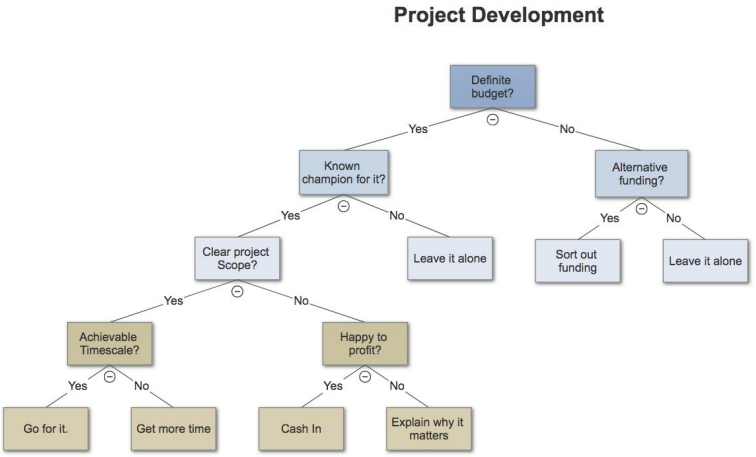

Fig 5. Decision Tree

A **decision tree** is a flowchart-like tree structure where an internal node represents feature, the branch represents a decision rule, and each leaf node represents the

outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in a recursive manner. The time complexity of decision trees is a function of the number of records and number of attributes in the given data. The decision tree is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. Decision trees can handle high dimensional data with good accuracy.

An ensemble is a composite model, combines a series of weak learners with the aim of creating an improved strong learner. Here, individual classifier vote and final prediction label returned that performs majority voting. Ensembles offer more accuracy than individual or base classifier. **AdaBoost**, which stands for Adaptive Boosting is an ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that we get a high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations. Here we use the Decision Tree Classifier as a weak learner to combine and form a strong learner using AdaBoost.

First, the performance of each classifier is evaluated by fitting the model to the test data without and cross-validation or hyper-parameter tuning i.e. with the default values for their parameters. Then, after comparing their respective F1-scores and accuracies, the grid search will be utilised to optimise the parameters to obtain the best model. If the performance improves, the best model will be used to finally predict the classes. The source code presents the best parameters at the end.

Benchmark

The benchmark model is obtained from the research paper titled **Prediction of Diabetes using Classification[5] by Deepti Sisodia and Dilip Singh Sisodia**. The model used the data after preprocessing, however the preprocessing steps are not listed or explained. It used Naive Bayes, SVM, and Decision Trees classifiers and the best F1-scores they obtained were 0.76, 0.513, and 0.736 respectively. The accuracy scores were 76.30, 65.10, and 73.82 percent for each respective classifier.

III. Methodology

Data Preprocessing

The data is not preprocessed from source which means it had many anomalies. Upon inspection it was noticed that some features had zero values which needed to be addressed. The columns Glucose, BloodPressure, SkinThickness, Insulin and BMI

had 5, 35, 227, 374, 11 zero values respectively. Using the pandas.replace() method the zero values were replaced with median values for the respective columns.

The outcome column, which is the class variable was then dropped from the data frame and saved as a separate variable. The data was then split into training and testing sets with a ratio of 70:30. The MinMaxScaler object was fit to the training set and the object was used to transform both training and testing sets.

Implementation

The AdaBoost algorithm was imported from the **sklearn.ensemble.AdaBoost** library and its default parameters **(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=46)** were loaded , the Logistic Regression algorithm was imported from **sklearn.linear_model.LogisticRegression** library and its default parameters **(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=46, solver='warn', tol=0.0001, verbose=0, warm_start=False)** were loaded, and the Decision Tree algorithm was imported from **sklearn.tree.DecisionTreeClassifier** library and its default parameters **(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=46, splitter='best')** were loaded and the objects along with these default parameters were stored in a dictionary. A for-loop was used to iterate through the dictionary and to fit each classifier to the data and predict the outcome on the test set. The F1-scores were calculated using the **f1_score** method from **sklearn.metrics.f1_score** library and accuracy scores were calculated using the accuracy_score method from **sklearn.metrics.accuracy_score** library and rounded to three decimal places and were stored in a dictionary. These unoptimised scores were then plotted against each other to show performance comparison. A dictionary of parameters was made with the same keys as the of the dictionary of classifiers. An F1 scorer was made using the **make_scorer** method from **sklearn.metrics** library. A for-loop was used to iterate through the dictionary of classifiers while fitting each classifier to the grid search object with parameters **(estimator=classifiers[clf], param_grid=parameters[clf], scoring=scorer, cv=10, n_jobs=-1)** created from the **GridSearchCV** library imported from the **sklearn.model_selection.GridSearchCV** which was fit to the training data. There were 10 folds of cross-validation used. The time method from time library was used to track the time taken to fit each model. The best estimator, and best scores (rounded to three decimal places) were stored in different dictionaries.

Everything worked well from the first time except that AdaBoost classifier would take more than a few minutes to train even on such a small dataset, and that is when I checked the documentation and noticed that the n_jobs parameter of the GridSearchCV object was set to None by default which meant it only used 1 of the available CPU cores. I set it to -1 to use all cores and the time taken to fit the classifiers decreased significantly.

Refinement

The initial scores of the untuned models were as follows:

	AdaBoost	**Logistic Regression**	**Decision Tree**
F Score	0.663	0.622	0.599
Accuracy Score	0.762	0.758	0.710

The following hyper-parameter space was used to tune the hyper-parameters using GridSearchCV-

AdaBoost Classifier:

Parameter	**Values**
base_estimator	DecisionTreeClassifier(max_depth=1,random_state=rstate), DecisionTreeClassifier(max_depth=2,random_state=rstate), DecisionTreeClassifier(max_depth=3,random_state=rstate)
n_estimators	25, 50, 75, 100
learning_rate	0.5, 1, 1.5

Logistic Regression:

Parameter	**Values**
penalty	l2
tol	0.0001
C	0.01, 0.1, 1.0

max_iter	50, 100, 250

DecisionTree Classifier:

Parameter	Values
min_samples_leaf	9, 10, 15, 20, 25
min_samples_split	10, 15, 20
max_depth	None, 2, 4, 6, 8, 10

After hyper-parameter tuning the results improved quite a bit. The optimised scores are as follows:

	AdaBoost	Logistic Regression	Decision Tree
F Score	0.667	0.622	0.619
Accuracy Score	0.779	0.758	0.723

Initially I had added several more values to each of the parameters but noticed that they were nowhere near the optimum values and were only making the GridSearch computationally hard while taking more time and giving no improvement in performance, so afterwards those values were dropped from the list of values. The hyper-parameters which gave the optimum results are presented in the Results section.

The scores did not improve for Logistic Regression, whereas for AdaBoost the F1 score and accuracy score improved by around 0.6% and 2.23% respectively. The Decision Tree classifier also showed an improvement with the F1 scores and accuracy scores increasing by 3.34% and 1.8% respectively for the final model.

IV. Results

Model Evaluation and Validation

The number of cross-validation folds are set to 10. The final model for each classifier is as follows:

1. AdaBoostClassifier(algorithm='SAMME.R', base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=1, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=46, splitter='best'), learning_rate=1.5, n_estimators=25, random_state=None)
2. LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=50, multi_class='warn', n_jobs=None, penalty='l2', random_state=46, solver='warn', tol=0.0001, verbose=0, warm_start=False)
3. DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=15, min_samples_split=10, min_weight_fraction_leaf=0.0, presort=False, random_state=46, splitter='best')

Upon testing with different random states it was observed that in general the performance varied by around ±10% for all models, however the trend suggested that AdaBoost performed better in most cases, with Decision Trees following it, and finally Logistic Regression seems to be underperforming in many cases. The final model is reasonable but still has a lot of room for improvement. The results from the model are adequate for such a simple model. More tuning and advanced classification techniques would perform better.

Justification

Looking at the best model we have, with AdaBoost we can say that with an F1 score of 0.667 and an accuracy of 77.9% compared to the F1 score of 0.760 and an accuracy of 76.30% of the benchmark model, the AdaBoost model performed just adequately. The model failed to outperform the benchmark model, but considering the fact that the data we used contained far more zero values we can safely assume that the model could have performed a little better with good data to train. The model is not great by any means. Also, there was no information regarding what preprocessing steps were taken on the benchmark model's data. In case we decided to drop all data with the missing (zero) values we would have lost a good chunk of the data so we had to work with what was available. With 77% accuracy our model still outperformed the benchmark model, but since we were aiming for a high F1 score this is still going to be considered as an inferior model. The problem is still considered to be solved.

V. Conclusion

Free-Form Visualization

The number of zero values in the features are shown below.

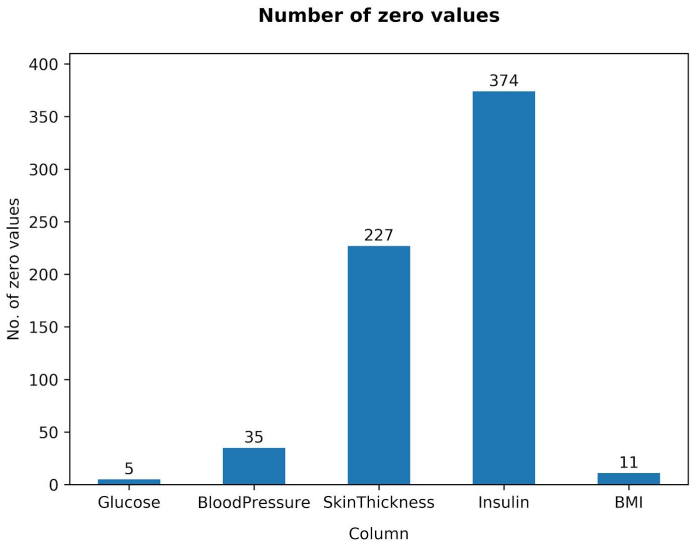

Fig 6. Number of zero values in columns

It can be observed that there are a significant amount of missing values which hinder the learning process of a model. This leads to the model to underperform as the quality of data is deteriorated making it hard for the classifier to learn the actual trends.

The final results can be seen in the bar plot below which shows the performance of the models after hyper-parameter tuning using grid search.

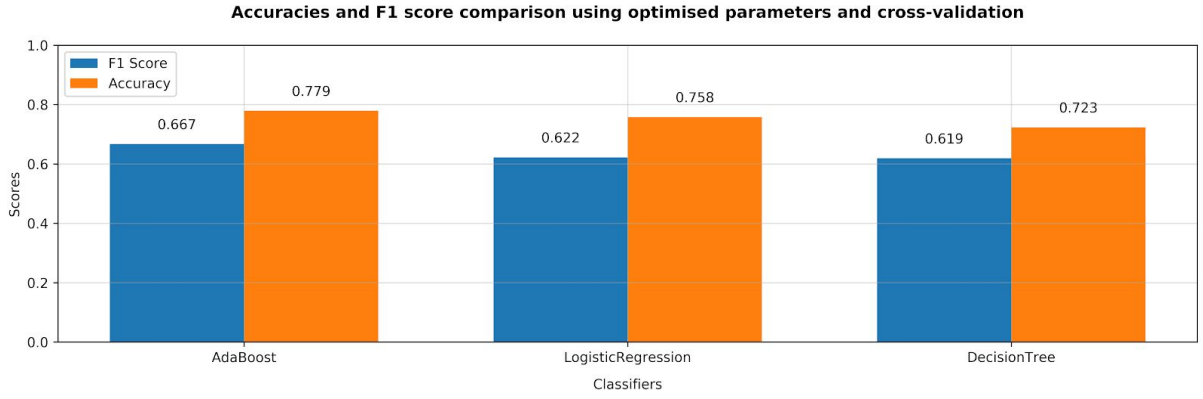

Fig 7. Comparison of scores of all models (optimised)

The percent improvement after hyper-parameter tuning can be seen below.

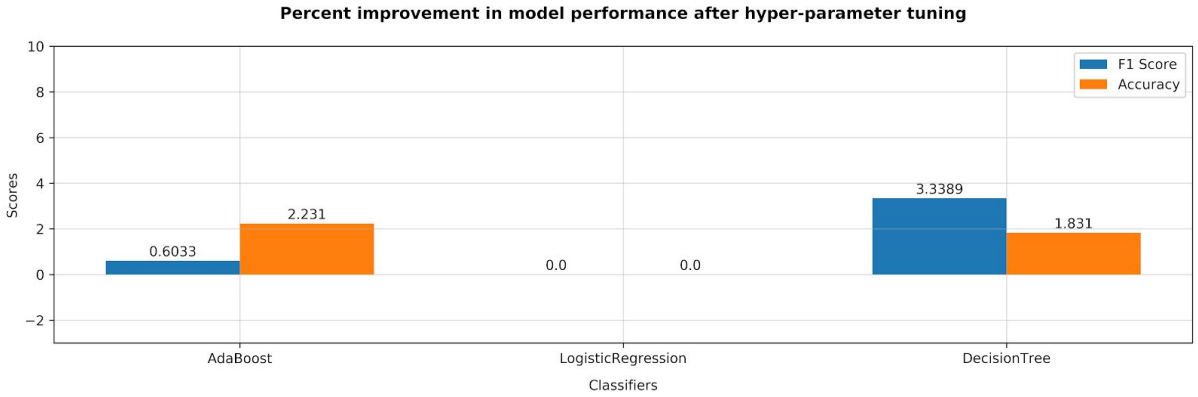

Fig 8. Comparison of percent improvement in model performance after tuning

Reflection

The problem solution is summarized as follows, first we began by importing the necessary libraries, and the dataset. Proceeded to look at the statistics of the dataset. We identified the abnormalities with zero values and replaced them with median values of the respective features. Moving on with separating the feature variables and class labels, then splitting the data into training and test sets. Then scaling the data in both sets. Plotting and observing the correlations between features, creating the classifier objects with default parameters and fitting the data. Storing and plotting the F1 scores and accuracies, optimising the hyper-parameters using grid-search while storing the best models and scores. Plotting the scores to observe the percent improvement over the default parameters, and finally listing the best models and parameters.

The main hit in performance was due to the fact that the data contained many missing values. Choosing the right classifiers, and optimising their parameters was also a tedious task. Looking at the quality of the dataset, the model seems to have performed adequately and the F1-scores and accuracies are indicative of that.

Improvement

There could be many better algorithms that could have been implemented. There are several really sophisticated algorithms that are used on Kaggle which could have performed better. Several ensemble techniques exist that I researched but could not implement which I would consider to use on this dataset. A few of them in particular are XGBoost[6], LightGBM[7], and stacking methods. If using my solution as a new

benchmark, certainly there are better solutions whose performance would surpass the performance of the models used in this project.

References

[1] International Diabetes Federation (2019). IDF Diabetes Atlas, 9th edn. Brussels, Belgium: International Diabetes Federation.
[2] Pima Indian Diabetes Database
[3] Kaggle: Your Home for Data Science, kaggle.com
[4] UCI Machine Learning Repository
[5] Deepti Sisodia, Dilip Singh Sisodia, Prediction of Diabetes using Classification Algorithms, Procedia Computer Science, Volume 132, 2018, Pages 1578-1585, ISSN 1877-0509
[6] XGBoost Library, XGBoost Documentation
[7] LightGBM Library, LightGBM Documentation