SPRINGBOARD – DSC

CAPSTONE PROJECT 2

EARLY STAGE DIABETES RISK PREDICTION

By IGOR VEREVKIN

December 2020

1. INTRODUCTION:

Diabetes is a group of metabolic disorders characterized by a high blood sugar level over a prolonged period of time. If left untreated, diabetes can cause many severe health issues or even death. About 422 million people worldwide have diabetes and 1.6 million deaths are directly attributed to diabetes each year.

The goal of this project was to build predictive models that would estimate the probability that a patient is diabetic as a function of available features (symptoms).

Intended stakeholders are:

- M Faniqul Islam
- Rahatara Ferdousi
- Sadikur Rahman
- Yasmin Bushra

From Metropolitan University Sylhet, Bangladesh

During this project we were able to build several models that showed good results in all performance metrics.
We assume the dataset we used was composed in a way that contributed to the models' good performance.

All the implementation details can be found in the notebooks we developed.

All the models presented in this report are built using the same train and test splits.

Link to the notebooks:
https://github.com/igorverevkin1992/Capstone2

2. APPROACH

2.1. Data Acquisition And Wrangling

The dataset we used for the project was created by the staff of Metropolitan University Sylhet, who conducted direct patients questioning in Diabetes Hospital Sylhet, Bangladesh.
The dataset was downloaded from UC Irving machine learning repository:
https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset

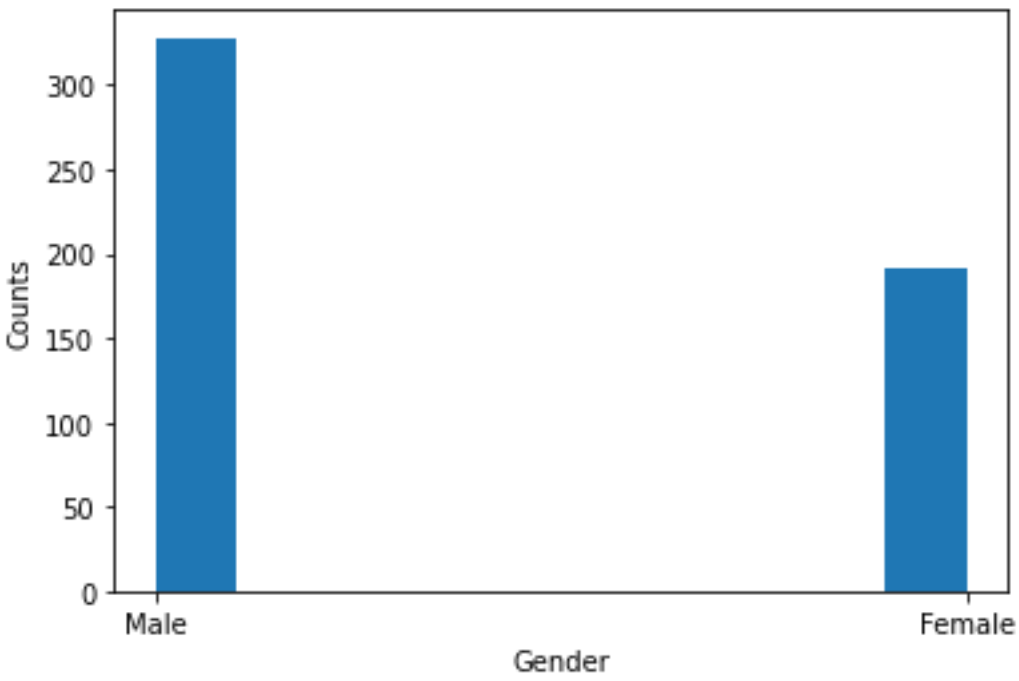
During Data Wrangling step of the Data Science Method, we checked if the data had any missing values, duplicates and if the dataset was balanced.

There were no missing values in the data, but there were many duplicated observations. However, the dataset did not provide us with any primary ID (like patient's name or his ID number). The features that represented age and the symptoms (which were common among most of the diabetic patients) were in a binary form ('Yes' or 'No'). Considering all those, it seemed reasonable to assume that those duplicates were actually different patients of same age who had same symptoms.

We also checked the number of positive and negative cases in the dataset and found out that the dataset was unbalanced – there were 120 more 'Positive' values than 'Negative' ones.

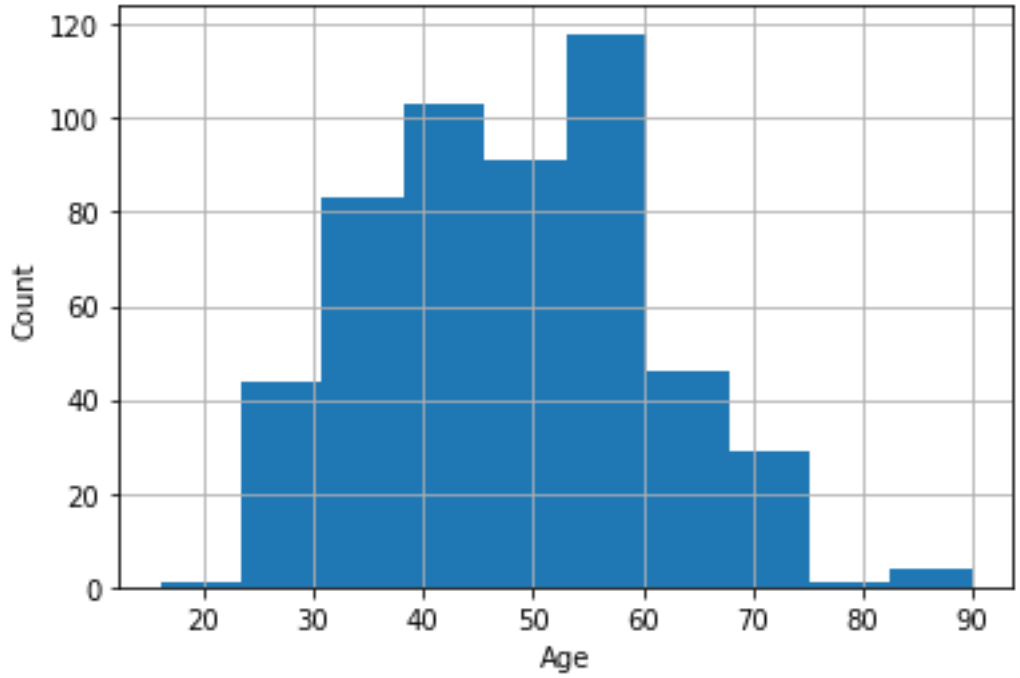
2.2. Storytelling and Inferential Statistics

First, we needed to see a big picture of our data. Let's take a look of the gender distribution of our patients.



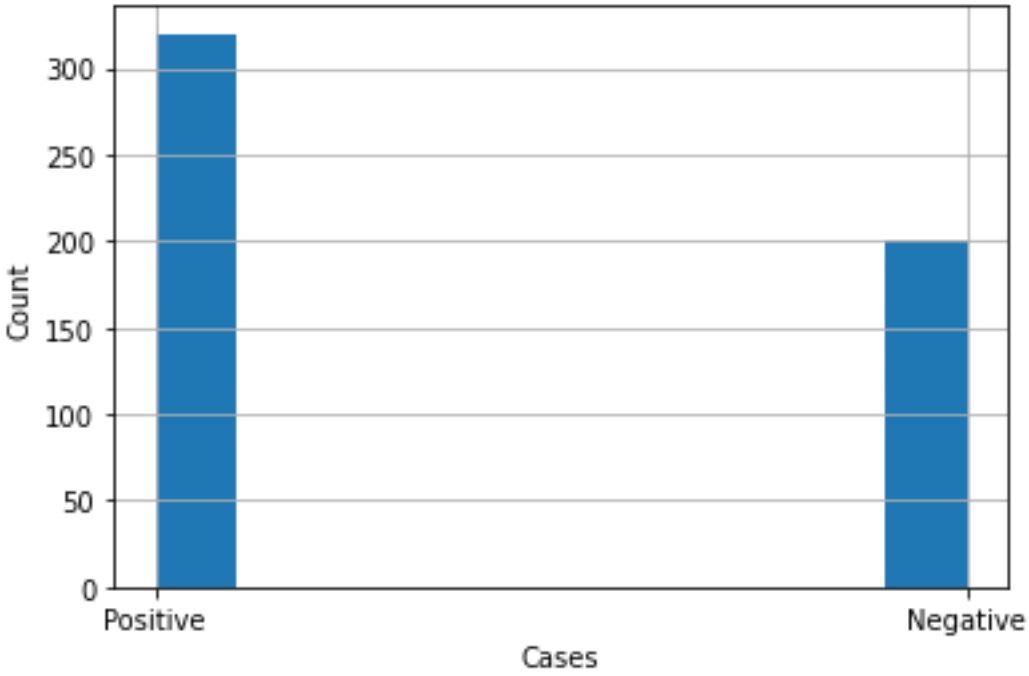
We can clearly see that men comprise the bigger part of the patients. It will be interesting to see if they are more prone to having diabetes. We will find it out later in the notebook.

Here is the age distribution of the patients.



We can see that the age data is normally distributed. The mean age of our patients is 48 years old with the standard deviation of approximately 12 years. The youngest patients is 16 years old and the oldest one is 90 years old.

Now, let us look at our target variable - positive and negative cases.

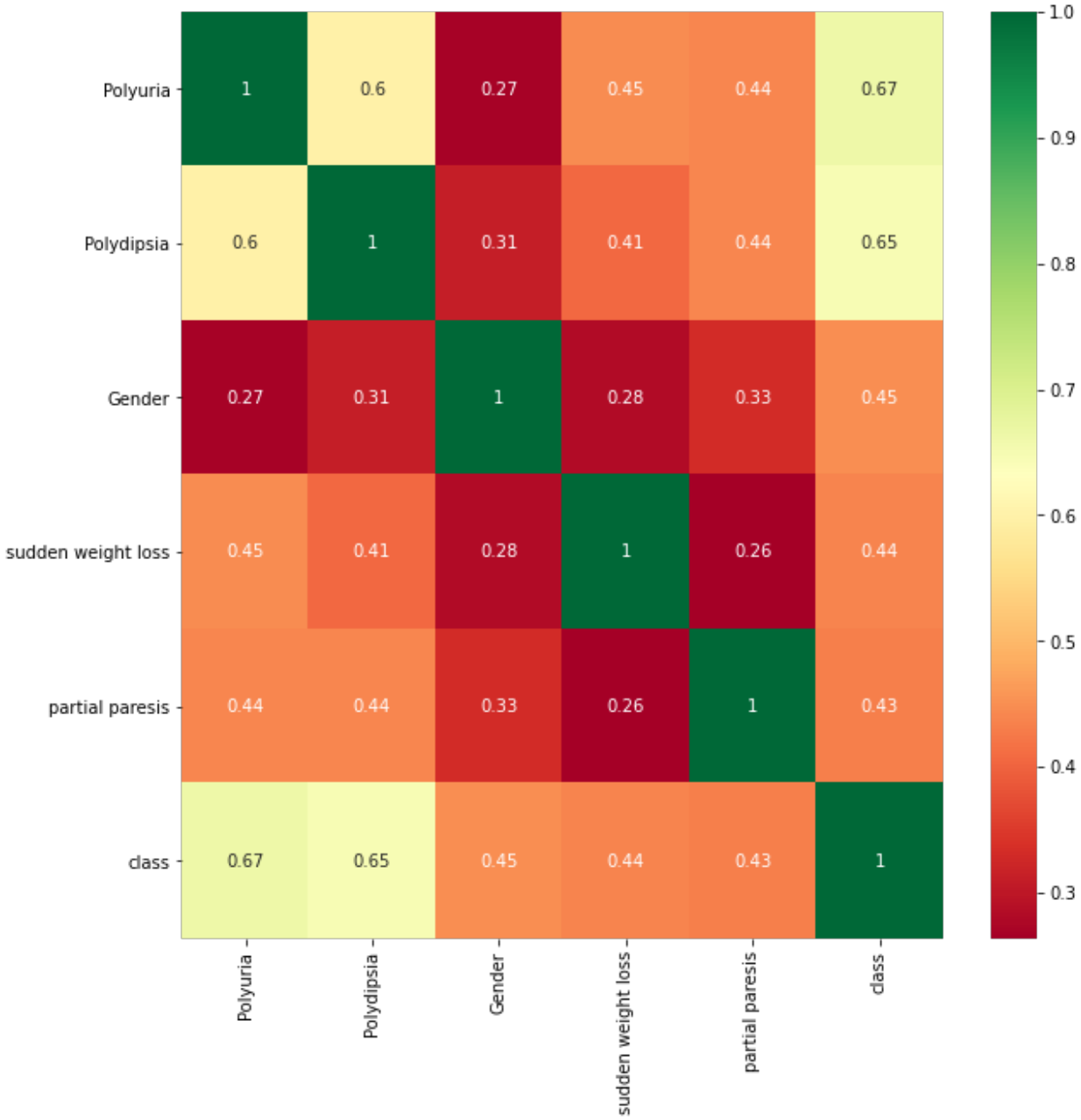


As we mentioned before, the data is unbalanced – we have 320 positive cases and 200 negative. But is it true for both male and female patients?

Only approximately 44% of all men in the dataset had diabetes. What about women?

173 of 192 women in the dataset were diabetic. That is extremely high percentage – 90.1%

Now, the dataset had a decent amount of features – basically, symptoms – which we could use in our models to predict the outcome. However, were they all equally important for the prediction?



This is a heatmap that shows correlation of different features with each other and with our target feature ('class'). As we can infer from the map, 'Polyuria' and 'Polydipsia' have the biggest correlation with 'class'. Later in the report, we will see if they are indeed the most important features for our models.

2.3. Baseline Modeling

Since our problem is a classification problem, we picked **Logistic Regression** as our baseline model. We scaled the data and split it into train and test sets. After that, we trained Logistic Regression classifier, first without hyperparameters tuning and then with it. We also made sure there is no overfitting by comparing model's performance on both train and test splits - there were no significant gaps.

In [24]:
```
print("=== Classification Report ===")
print(classification_report(y_test, clfparams_ypred_test))
```

```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        40
           1       0.98      0.92      0.95        64

    accuracy                           0.94       104
   macro avg       0.93      0.95      0.94       104
weighted avg       0.95      0.94      0.94       104
```

As you can see, our baseline Logistic Regression model showed good results on the test set with regard to all performance metrics.

IMPORTANT NOTE:

Our original problem is classifying patients as diabetic/non-diabetic. Here we want to briefly discuss performance metrics. We assume that our Null Hypothesis is that a patient is **NOT diabetic**. We want to minimize Type II errors (classifying the patient as non-diabetic while he actually IS diabetic), since it will lead to severe health problems for the patient. In order to do it we will focus on improving our models' recall score (particularly, minimizing False Negatives).

2.4. Extended Modeling

Even though our Logistic Regression model showed good results, we would like to check if other models will show similar (or better, hopefully) results. We build three forest-based models (Random Forest, Gradient Boosting and Adaptive Boosting) as well as Support Vector Machine Classifier. The process was the same for all of them – we initialized the model, used GridSearch or RandomizedSearch in order to find the best hyperparameters and printed classification report for each model.

Here are hyperparameters for the models we used:

```
In [46]: #number of trees
n_estimators = [int(i) for i in np.linspace(200, 2000, 10)]

#Learning rates
learning_rate = [0.05, 0.1, 0.25, 0.5, 0.75, 1]

#number of features for each split
max_features = ['auto', 'sqrt']

#maximal depth
max_depth = [int(i) for i in np.linspace(100, 500, 11)]

#parameters grid
param_grid = {'n_estimators':n_estimators, 'learning_rate':learning_rate, 'max_features':max_features, 'max_depth':max_depth}
```

```
In [47]: gbc_rand = RandomizedSearchCV(estimator=gbc, param_distributions=param_grid, n_iter=100, cv=5, random_state=42, n_jobs=-1)
gbc_rand.fit(X_train_norm, y_train)

Out[47]: RandomizedSearchCV(cv=5, estimator=GradientBoostingClassifier(random_state=42),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'learning_rate': [0.05, 0.1, 0.25, 0.5,
                                                          0.75, 1],
                                        'max_depth': [100, 140, 180, 220, 260,
                                                      300, 340, 380, 420, 460,
                                                      500],
                                        'max_features': ['auto', 'sqrt'],
                                        'n_estimators': [200, 400, 600, 800,
```

RandomizedSearchCV for Gradient Boosting

```
In [53]: param_grid = {"base_estimator__criterion" : ["gini", "entropy"],
                "base_estimator__splitter" :   ["best", "random"],
                "learning_rate": [0.05, 0.1, 0.25, 0.5, 0.75, 1],
                "n_estimators": [int(i) for i in np.linspace(200, 2000, 10)]
               }


weak_c = DecisionTreeClassifier(random_state = 42, max_features = "auto", class_weight = "balanced", max_depth = 3)

ABC = AdaBoostClassifier(base_estimator = weak_c)
```

```
In [54]: abc_rand = RandomizedSearchCV(estimator=ABC, param_distributions=param_grid, n_iter=40, cv=5, scoring='roc_auc', random_state=42)
abc_rand.fit(X_train_norm, y_train)
print(abc_rand.best_params_)

{'n_estimators': 200, 'learning_rate': 0.1, 'base_estimator__splitter': 'best', 'base_estimator__criterion': 'gini'}
```

RandomizedSearchCV for Adaptive Boosting

```
In [60]: C_param = np.logspace(-2, 10, 13)
gamma_param = np.logspace(-9, 3, 13)
grid_param = {'C':C_param, 'gamma':gamma_param}
svmclf_grid = GridSearchCV(svmclf, param_grid=grid_param, cv=5)
svmclf_grid.fit(X_train_norm, y_train)

Out[60]: GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': array([1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03, 1.e+04, 1.e+05,
       1.e+06, 1.e+07, 1.e+08, 1.e+09, 1.e+10]),
                         'gamma': array([1.e-09, 1.e-08, 1.e-07, 1.e-06, 1.e-05, 1.e-04, 1.e-03, 1.e-02,
       1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])})
```

```
In [61]: print(svmclf_grid.best_params_)

{'C': 10.0, 'gamma': 0.1}
```

GridSearchCV for Support Vector Machine

We also made sure there were no collinearity in the dataset features as well as no data leakage.

Below you can see the results.

Random Forest Classifier:

In [32]:
```
print("=== RANDOM FOREST TEST SET Classification Report ===")
print(classification_report(y_test, rfcy_pred_params))
```

```
=== RANDOM FOREST TEST SET Classification Report ===
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        40
           1       0.98      0.98      0.98        64

    accuracy                           0.98       104
   macro avg       0.98      0.98      0.98       104
weighted avg       0.98      0.98      0.98       104
```

The results of the Random Forest model is better than the results of our baseline model. 98% of those who actually had diabetes were classified correctly, false negatives – 0.02%

Gradient Boosting Classifier:

In [19]:
```
print('===== GBC_PARAMS TEST SET ACCURACY SCORE =====')
print(accuracy_score(y_test, gbc_params_ypred))
print('===== GBC_PARAMS TEST SET CLASSIFICATION REPORT =====')
print(classification_report(y_test, gbc_params_ypred))
```

```
===== GBC_PARAMS TEST SET ACCURACY SCORE =====
0.9903846153846154
===== GBC_PARAMS TEST SET CLASSIFICATION REPORT =====
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        40
           1       0.98      1.00      0.99        64

    accuracy                           0.99       104
   macro avg       0.99      0.99      0.99       104
weighted avg       0.99      0.99      0.99       104
```

Here the performance is even better. We got perfect score for our recall on '1', false negatives – 0.

Adaptive Boosting Classifier:

In [25]:
```
print('===== ABC_PARAMS ACCURACY SCORE =====')
print(accuracy_score(y_test, abc_params_ypred))
print('===== ABC_PARAMS CLASSIFICATION REPORT =====')
print(classification_report(y_test, abc_params_ypred))
```

```
===== ABC_PARAMS ACCURACY SCORE =====
1.0
===== ABC_PARAMS CLASSIFICATION REPORT =====
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        64

    accuracy                           1.00       104
   macro avg       1.00      1.00      1.00       104
weighted avg       1.00      1.00      1.00       104
```

Perfect results in all the performance metrics.

Support Vector Machine Classifier:

```
print('=====SVMCLF_PARAMS ACCURACY SCORE=====')
print(accuracy_score(y_test, svmclf_params_ypred))
print('=====SVMCLF_PARAMS CLASSIFICATION REPORT=====')
print(classification_report(y_test, svmclf_params_ypred))
```

```
=====SVMCLF_PARAMS ACCURACY SCORE=====
0.9807692307692307
=====SVMCLF_PARAMS CLASSIFICATION REPORT=====
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        40
           1       0.98      0.98      0.98        64

    accuracy                           0.98       104
   macro avg       0.98      0.98      0.98       104
weighted avg       0.98      0.98      0.98       104
```

Again, very good performance. False negatives – 0.02%.

3. FINDINGS

After we finish modeling, we can perform comparative analysis of each model's performance with regard to the others. This will potentially let us identify the best model, which we can then implement for solving our original problem – classifying patients.

Here you can see a table with all the models' performance metrics.

MODEL	ACCURACY	PRECISION (1)	PRECISION (0)	RECALL (1)	RECALL (0)	F1-SCORE (1)	F1-SCORE (0)
LogisticRegression	0.94	0.98	0.89	0.92	0.97	0.95	0.93
RandomForest	0.98	0.98	0.97	0.98	0.97	0.98	0.97
GradientBoosting	0.90	0.98	1.00	1.00	0.97	0.99	0.99
ADABoost	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Support Vector Machine	0.98	0.98	0.97	0.98	0.97	0.98	0.97

Again, we want to emphasize that minimizing False Negatives is a priority for us. This means we will assess our models performance using Recall score with regard to class '1'. But what exactly does "RECALL (1)" mean in the context of our problem? This score shows how many patients, who actually had diabetes, were correctly classified as diabetic. Why is it more important for us than "RECALL (0)"? Because the consequences of saying a patient that he's not diabetic while he actually is will be much more severe for his health than saying a non-diabetic patient that he has diabetes while he actually does not.

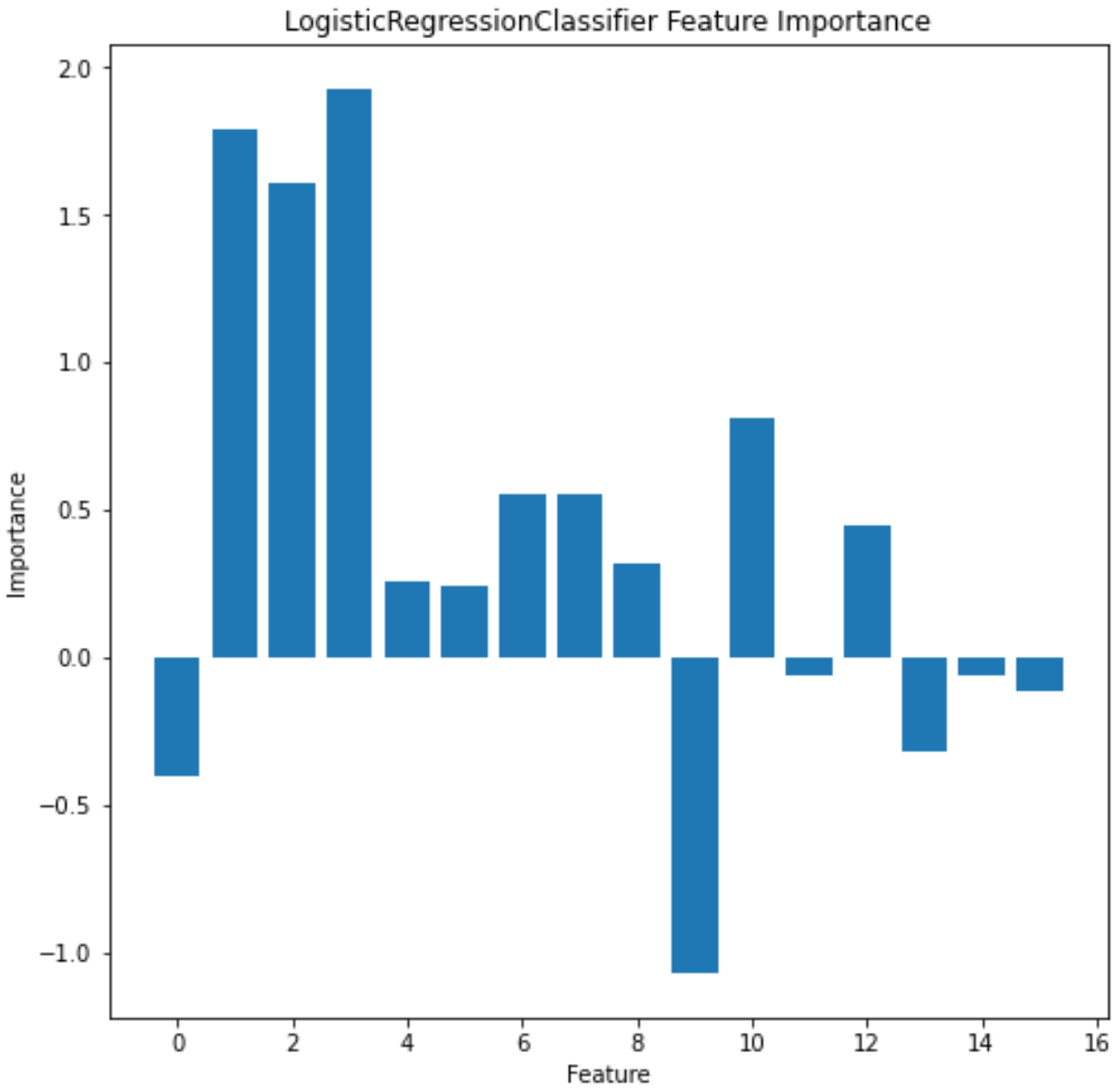
Now let's take a look at the table. We can see that Logistic Regression showed significantly worse results than all the other models. Random Forest and Support Vector Machine (we used Radial Basis Kernel) both showed 0.98 score on RECALL ('1'). However, we got even better – in fact, perfect – results on two tree-based algorithms, Gradient Boosting and Adaptive Boosting. They both showed 100% recall score with regard to class '1'.

Those scores let us assess a model's performance. However, we still don't know what features – in our case, a patient's characteristics and symptoms – have the highest importance in terms of predicting if he is diabetic or not. And this is indeed a very important part of our original problem.

In order to find it out we will extract features importance from two of our best performing tree-models – Gradient Boosting and Adaptive Boosting. We will also look at Logistic Regression Feature Impact.

But what exactly is feature impact? In machine learning, feature impact is a metric that identifies which features in a dataset have the greatest effect on the outcomes of a model. Unlike feature importance for tree-based algorithms, feature impact shows us not only the magnitude of the feature, but also the direction. In other words, we will be able to identify, which features contribute to being classified as class 0 and which do the same for class 1.

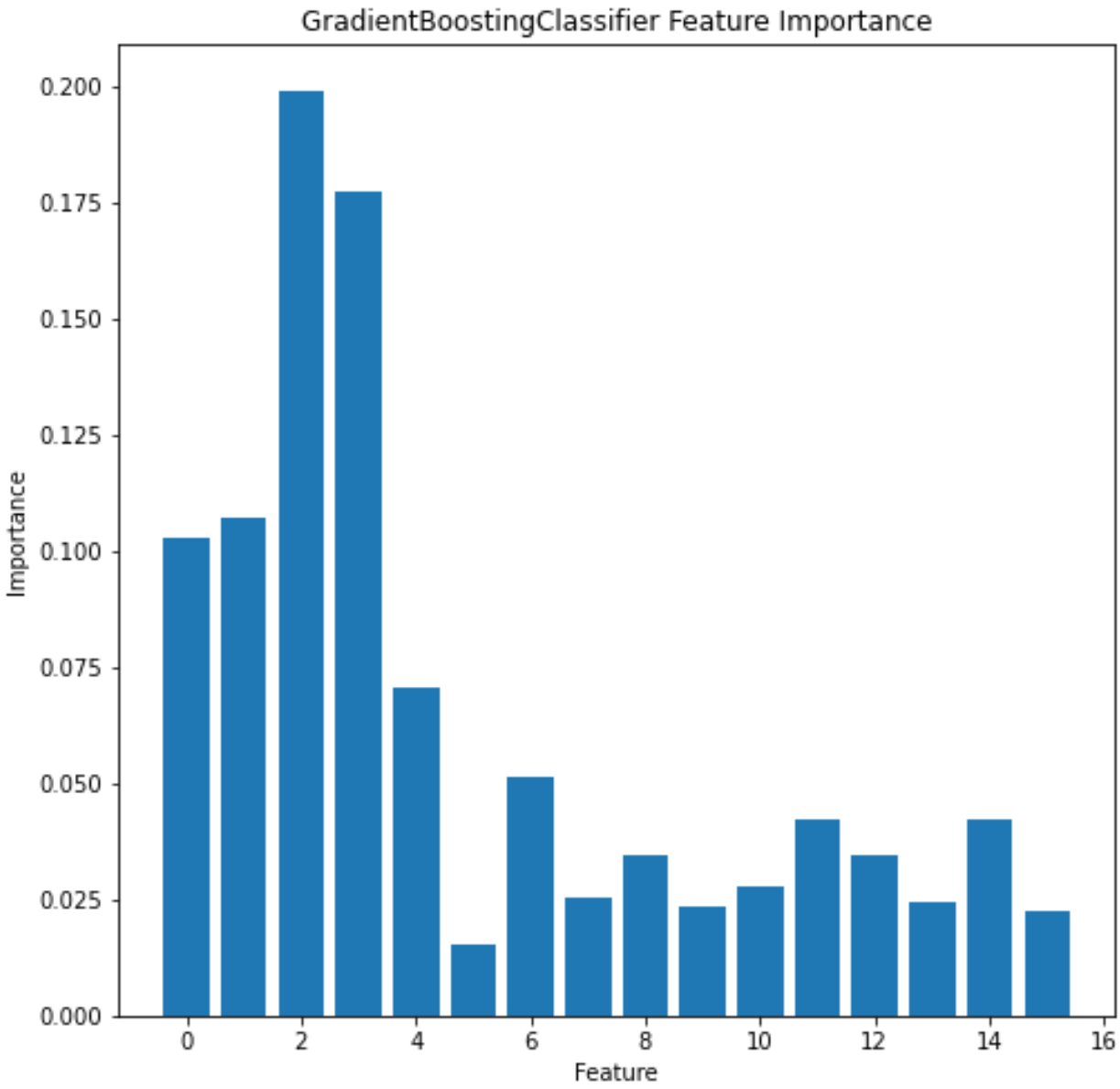
Here is Feature Impact for Logistic Regression.



As you can see, the features with highest impact for being classified as ‘1’ (‘positive’) are feature 3(‘Polydipsia’), feature 1(‘Gender’) and feature 2(‘Polyuria’). The features that are most important for class '0' are feature 9('Itching'), feature 0(‘Age’) and feature 13(‘muscle stiffness).

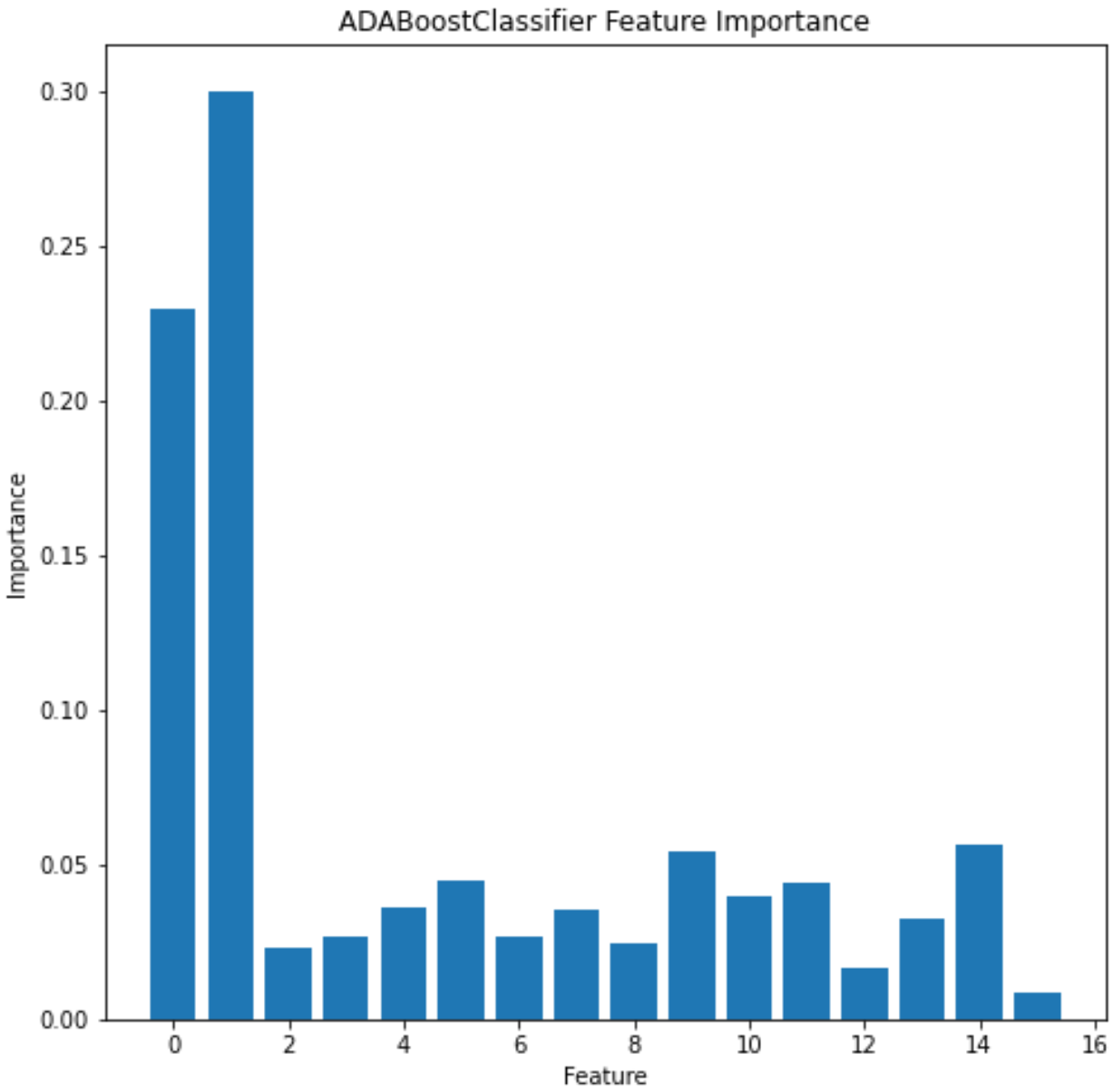
Feature importance for tree-based algorithms, on the other hand, does not show us the direction of the particular feature. Feature importance is a technique that assign a score to input features based on how useful they are at predicting a target variable. Usually, metrics like Gini Impurity or Entropy are used in order to quantify the results of the technique.

Here you can see feature importance for Gradient Boosting



Feature 2('Polyuria') and Feature 3('Polydipsia') have significantly higher importance then the next most important Feature 1 ('Gender').

Now let’s take a look at Adaptive Boosting results



Interestingly enough, ADABoost shows completely different results. The most important features are feature 0 ('Age') and feature 1 ('Gender').

4. Conclusion and Future Work

Now we should answer the question – which features are the most important and which model should we recommend to implement?

Even though out 'best' model – ADABoost – showed completely different results, we would still assume that 'Polyuria' and 'Polydipsia' has the highest importance in terms of predicting a patient's class (diabetic or non-diabetic). In fact, if we go back to the heatmap we saw in the EDA section, we would see that those two features have by far the highest correlation with the

target. Moreover, if we look at Random Forest feature importance and Logistic Regression feature impact we would see that both of them have ‘Polyuria’ and ‘Polydipsia’ as the most important features. Logistic regression also showed that Gender (‘female’) contributing to the patient being classified as ‘1’ (diabetic).

Future work:

- Use different train test splits in order to find out if Adaptive Boosting Classifier performance can be attributed to the particular split we used.

5. Recommendations for the Clients

After performing all the steps described above we can recommend:

- Implement Gradient Boosting Classifier as a production model for predicting patient’s class
- Pay close attention to patients with Polyuria and Polydipsia since those two symptoms are highly correlated with having diabetes
- Take into account that women seem to be more prone to diabetes than men