Machine Learning Engineer Nanodegree

Capstone Project

Amal Alabdulkarim
March 27, 2018

I. Definition

Project Overview

Diabetes can strike anyone, from any walk of life. Diabetes is a serious life-long health condition that occurs when the amount of glucose (sugar) in the blood is too high because the body can't use it properly. If left untreated, high blood glucose levels can cause serious health complications[1]. Diabetes is responsible for 318,036 regional deaths in adults aged 20-79 years in 2017 (13% of all mortality). About 51.8% of all deaths from diabetes in MENA occurred in people under 60 [2]. Approximately 5 million of the 18 million people with diabetes in the U.S. do not know they have it [3]. Early detection and treatment of diabetes is an important step toward keeping people with diabetes healthy. It can help to reduce the risk of serious complications such as premature heart disease and stroke, blindness, limb amputations, and kidney failure [3].

In [4] the authors reviewed several machine learning and data mining approaches applied in the diabetes mellitus research. For the prediction problem, several algorithms and different approaches have been applied, such as traditional machine learning algorithms, ensemble learning approaches and association rule learning in order to achieve the best classification accuracy. Ensemble approaches, which use multiple learning algorithms, have proven to be an effective way of improving classification accuracy. The specific approaches have also been used in DM prediction[5]–[8]. Anderson et al. used a Bayesian scoring algorithm to explore the model space [8]. In [7], authors proposed an ensemble framework with multi-layer classification, using enhanced bagging and optimized weighting, combining seven heterogeneous classifiers. In [6], authors used Rotation Forest (RF), a newly proposed ensemble algorithm, to combine 30 machine learning algorithms. Finally, Han et al. presented an ensemble learning approach, which turns the "black box" of SVM decisions into comprehensible and transparent rules [5].

Early detection and treatment of diabetes is very important to prevent the side effects and to keep patients healthy. Diabetes is not fully curable, but with early detection and good life style it can be freeze and stopped from causing further damage to the human body.

Early diagnosis of diabetes has been an important topic for many researchers worldwide. In this project I will propose another method of predicting if the patient has diabetes or not from a set of given features to a trained model.

The dataset used in this project is the Pima Indians diabetes database, I got this dataset from Kaggle (https://www.kaggle.com/uciml/pima-indians-diabetes-database) It is originally from National Institute of Diabetes and Digestive and Kidney Diseases.

Problem Statement

Late diagnosis of diabetes mellitus can have severe consequences. Therefore, in order to prevent these health issues, we need to expedite the process of diagnosing this condition and even predict it before its first obvious symptoms.

The best solution would be to encourage people to go visit the family doctor more often, especially if they were more likely to get it. However, this solution is not feasible to everyone, knowing that many of those people live in developing countries and may not have access to good and affordable healthcare system. Therefore, I propose using the data we have already about the patients and the disease to try and predict its occurrences and make the diagnosis much faster.

The problem of diagnosing diabetes mellitus is a binary classification problem; therefore, it should be handled using classification techniques and algorithms. The solution would be a model that can predict diabetes given the a of features.

To define this binary classification, we can say that we use the given data to predict whether or not this person has diabetes. So, the outcomes are always 0 for negative and 1 means that the diagnosis is positive.

Metrics

A binary classifier can be evaluated using different metrics and every field has its different metrics that measures its goals. In this project we are testing to predict the occurrences of diabetes. This model will give us four values:

1. The number of positives.
2. The number of negatives.

To evaluate the model, we compare the predicted output to the actual values “the real output” and then yield us four values:

a. If the classifier says correctly that the subjects are negative they are called true negative (TN).
b. If the classifier wrongly says that the subjects are negative, then they are called false negative (FN).
c. If the classifier says correctly that the subjects are positive they are called true positive (TP).
d. If the classifier wrongly says that the subjects are positive, then they are called false positives (FP)

These values will be totaled and calculated into grand total and marginal total and then we will measure the precision (TP/(FP+TP)) and the recall (TP/(TP+FN)).

These values can then be used to evaluate the accuracy of the diabetes prediction. For the evaluation I would use **F1 score,** because this is a binary classification problem and The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal [9]. The formula for the F1 score is:

```
F1 = 2 * (precision * recall) / (precision + recall)
```

My benchmark for this is f1 > 0.70.

F1 score doesn't pay attention the true negatives in the calculation since it depends on the precision and recall, but it it's not as important as the rest of the values in our diabetes prediction case, patient who has been correctly not diagnosed with diabetes are infinite and they outnumber those who are diagnosed.

II. Analysis

Data Exploration

This section provides a detailed explanation of the used dataset and highlights statistical features and outliers. First, the dataset consists of 768 labeled record with 8 features as shown in the general visualization in figure 1. All patients are females at least 21 years of Pima Indian heritage. The features of this dataset are:

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: Diabetes pedigree function
8. Age: Age (years)
9. Outcome: Class variable (0 or 1)

Second, in this dataset 268 raw represent people who has been diagnosed with diabetes and 500 are not. Figure 2, shows the mean, count, standard deviation and minimum and maximum values for all of the features. It is also noticeable that this dataset has wrong zero values in Glucose, BloodPressure, SkinThickness, Insulin and BMI. Which shouldn't be there because it is not medically possible for a living human being to present these values. Therefore, these values are most likely typos.

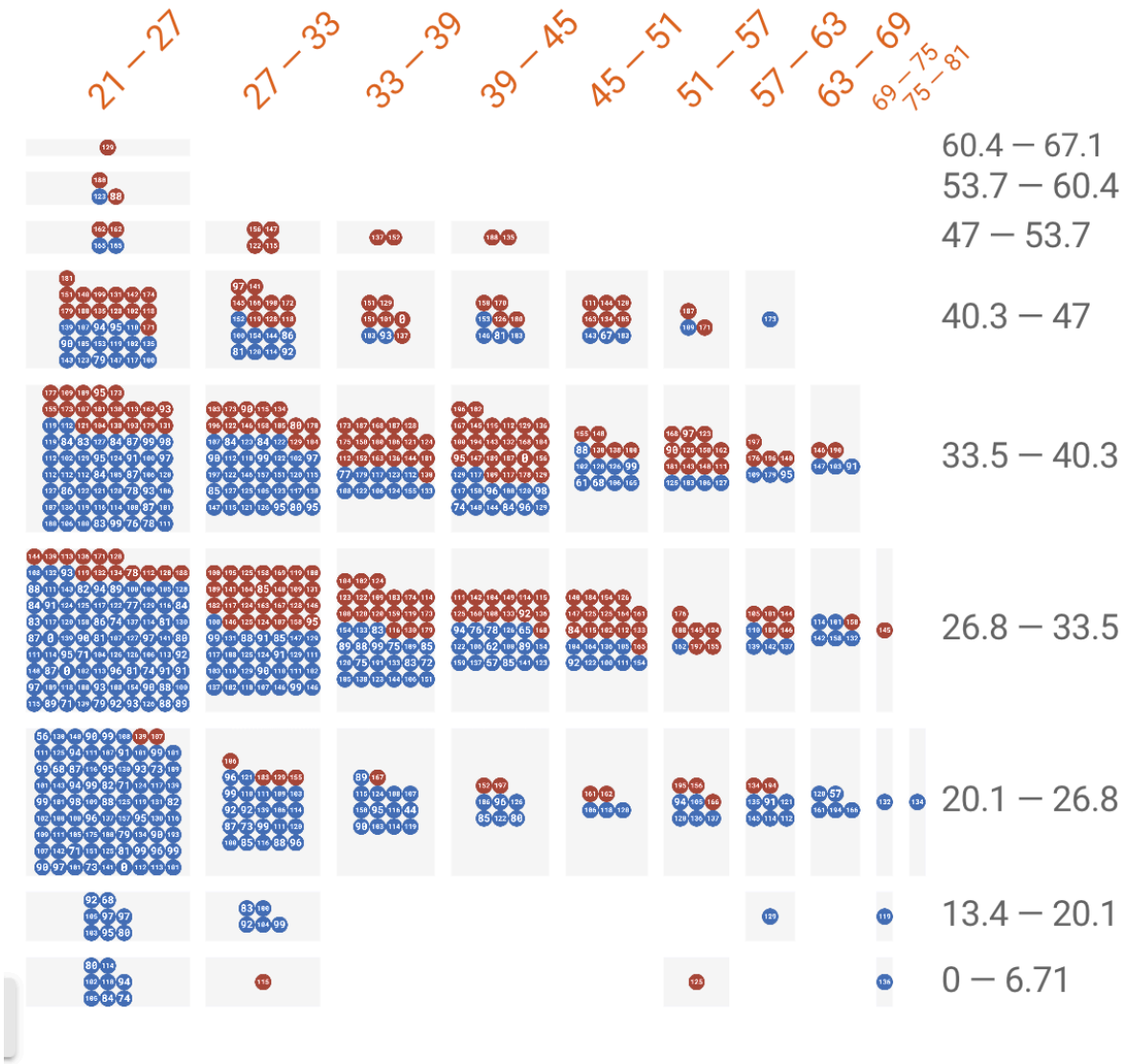

Figure 1: General visualization of the dataset where the rows represent BMI, columns represent Age and color, label on the dots represent glucose, and color represent outcome.

Table 1: Show the dataset description and statistics.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	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

Exploratory Visualization

This section provides several visualizations of the dataset. In order to thoroughly explore the data. The first visualization in figure 2, shows relationships between features using three different plots:

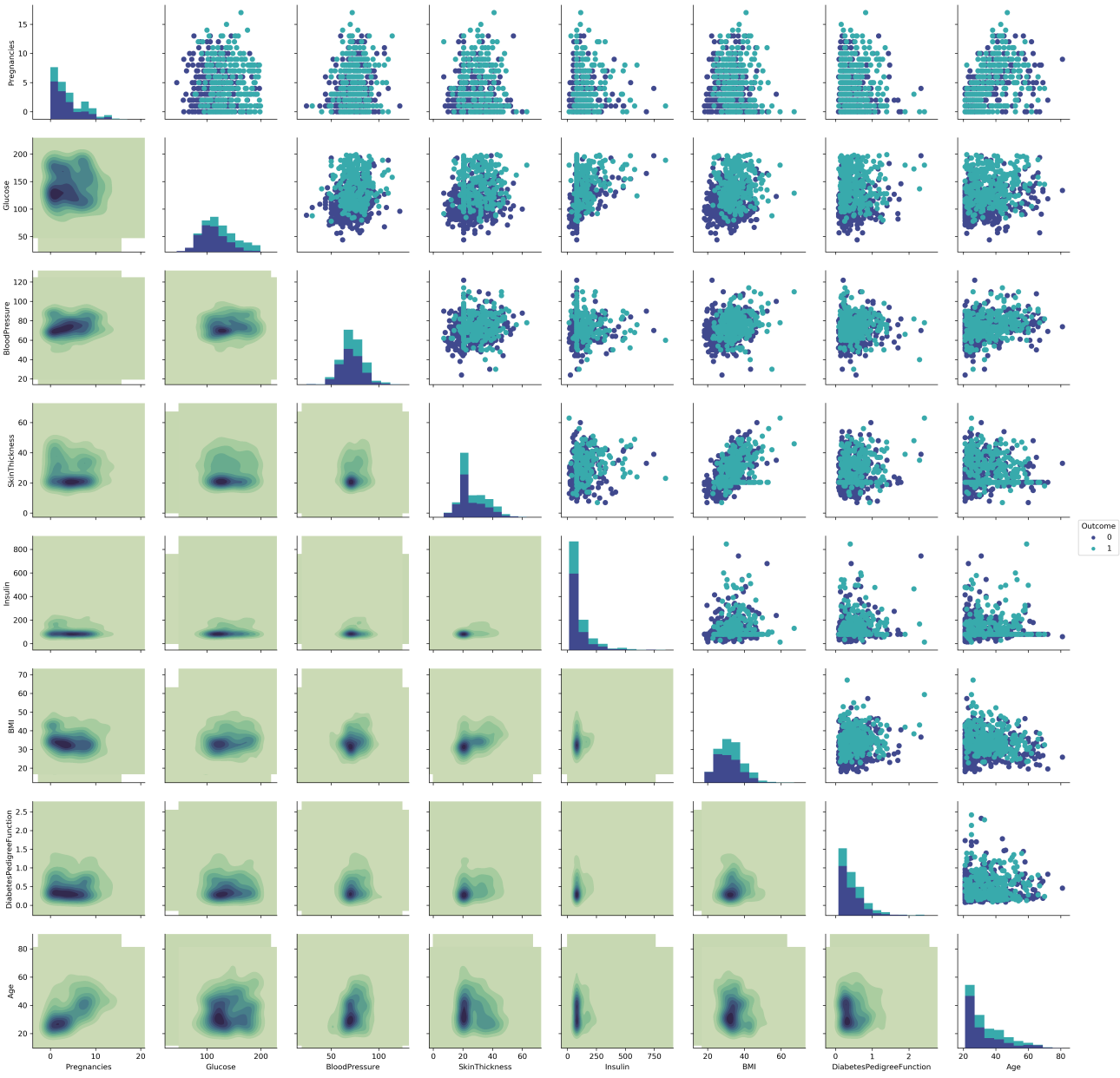

Figure 2: Shows pairwise relationships in the dataset.

- Histograms to show the distribution of each features values over the dataset which helps in recognizing which values of each feature affect the outcome. For example, the BMI histogram shows how a higher BMI value may help in indicating a positive Output.
- Scatterplot on the upper side shows how much each feature is affected by the other. For example, the Skinthickness/BMI scatterplot shows that the two features are proportional.

- Kernel Density plots on the lower side shows the bivariate density of the features. This helps in estimating the distribution of these feature values in relation to each other. For example, we can clearly see that the pregnancies and age are clustered to the lower values of axes which means that most of our dataset is young and have low previous pregnancies.

Figure 3, has several boxplots to show the distribution of the features based on median, minimum and maximum, first quartile and third quartile. It can also show us the outliers in the data. Such as the outlier in Skinthickness which is very high and differs significantly from the rest of the dataset.

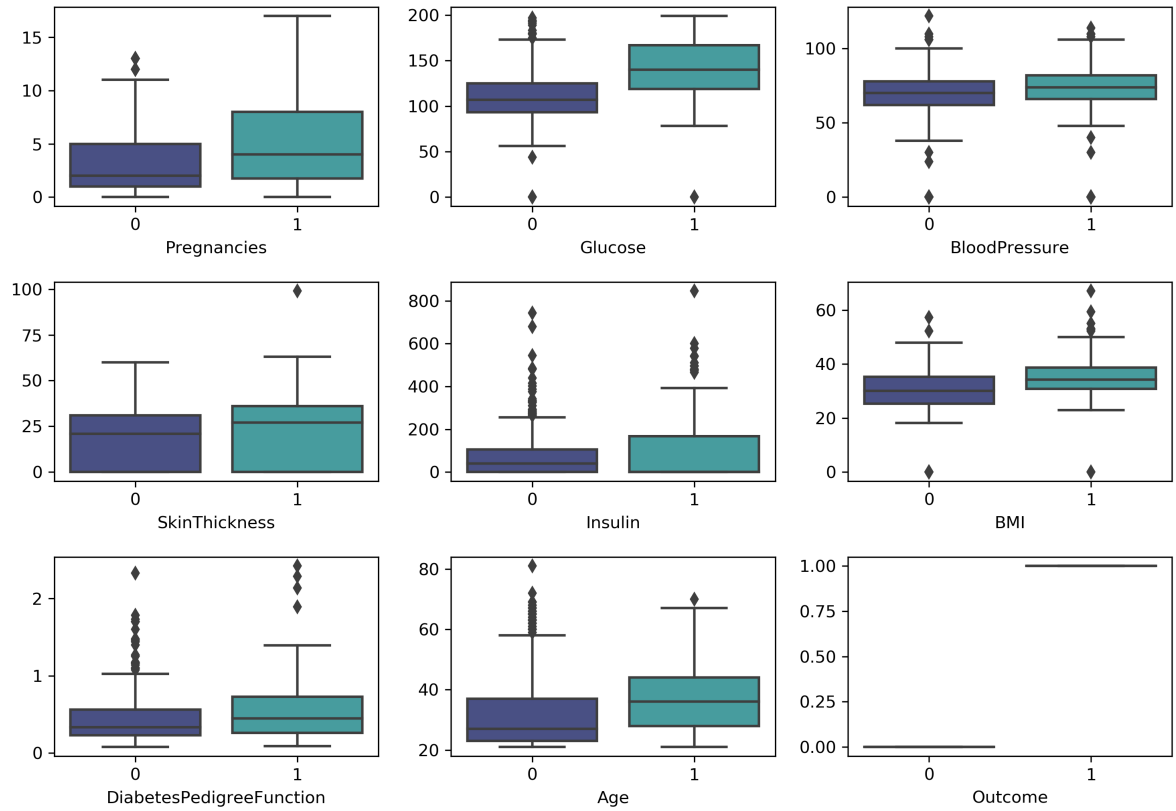

Figure 3: Boxplots shows the distribution of each feature.

Finally, a correlation matrix is used here to show how the features are closely related and to show their relationship with the outcome. This heat map will help us a lot in excluding non-important features in the next steps.

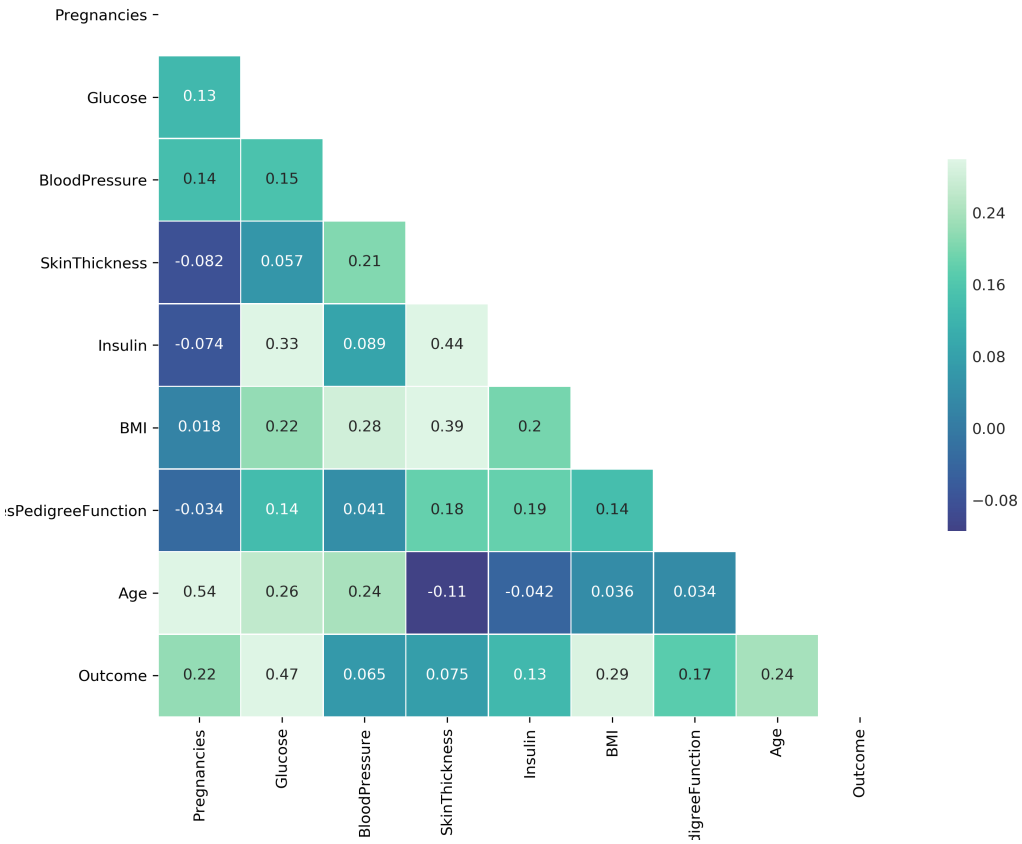

Figure 4: Correlation matrix shows the correlation between features and outcome.

Algorithms and Techniques

For my capstone I plane to use extreme gradient boosting classification. I chose this model because I am trying to predict if the patient has diabetes or not (0 or 1), and my data are less than 100k. Also, XGBoost is extremely fast, accurate and it also performs well in many cases.

XGBoost is a scalable and efficient implementation of Gradient Boosting Machines, which is a multi-purpose supervised learning method. Extreme gradient boosting has proven its high accuracy and speed over many cases. One of the best features of xtreme gradient boosting is that it utilizes the parallelism to push the limit of computational resources which leaves low memory footprint and faster training. Also it handles missing values automatically and it has interactive feature analysis features such as the feature importance analysis which helps you in identifying the most important features for the algorithm.

The XGBoost classifier have several parameters that you can tune to improve the outcome:

- max_depth : int. Maximum tree depth for base learners.
- learning_rate : float. Boosting learning rate (xgb's "eta")
- n_estimators : int. Number of boosted trees to fit.
- silent : Boolean. Whether to print messages while running boosting.
- objective : string or callable. Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).

- booster: string. Specify which booster to use: gbtree, gblinear or dart.
- nthread : int. Number of parallel threads used to run xgboost. (Deprecated, please use n_jobs)
- n_jobs : int. Number of parallel threads used to run xgboost. (replaces nthread)
- gamma : float. Minimum loss reduction required to make a further partition on a leaf node of the tree.
- min_child_weight : int. Minimum sum of instance weight(hessian) needed in a child.
- max_delta_step : int. Maximum delta step we allow each tree's weight estimation to be.
- subsample : float. Subsample ratio of the training instance.
- colsample_bytree : float. Subsample ratio of columns when constructing each tree.
- colsample_bylevel : float. Subsample ratio of columns for each split, in each level.
- reg_alpha : float (xgb's alpha). L1 regularization term on weights
- reg_lambda : float (xgb's lambda). L2 regularization term on weights
- scale_pos_weight : float. Balancing of positive and negative weights.
- base_score: The initial prediction score of all instances, global bias.
- seed : int. Random number seed. (Deprecated, please use random_state)
- random_state : int. Random number seed. (replaces seed)
- missing : float, optional. Value in the data which needs to be present as a missing value. If None, defaults to np.nan.

  These parameters can be toned to improve the prediction accuracy, control overfitting and handle imbalanced data.

For more information about the algorithm, please visit:

1. https://arxiv.org/pdf/1603.02754.pdf
2. http://xgboost.readthedocs.io/en/latest/python/python_api.html
3. http://xgboost.readthedocs.io/en/latest/how_to/param_tuning.html?highlight=parameters

Benchmark

From the Kaggle kernels submitted for this dataset, most submitted kernels have an accuracy value of more than 70%, I will try to achieve this value as my benchmark.

III. Methodology

Data Preprocessing

This section explains the data preprocessing steps done before training the model. XGBoost doesn't require data preprocessing because it is very robust and can handle missing data automatically.

To improve the accuracy of the prediction several data preprocessing steps has been done:

1. Remove outliers and extreme values. There is one outlier in the "Skin thickness" we have to remove it because it is most likely mistyped.
2. "Glucose","BMI", "Insulin", "Blood Pressure" and "Skin thickness" has zero values which is medically not possible. In this case, we have two options to solve this:
    a. Remove the records with zero values.
    b. Replace zero value with the mean.

    I will go with option B, because changing it to the mean will not affect the outcome and will keep the data.

3. Choosing the features that affect the Outcome the most, to identify the features I used the data shown in the correlation matrix in figure 4 and feature importance function in xgboost as shown in figure 5.

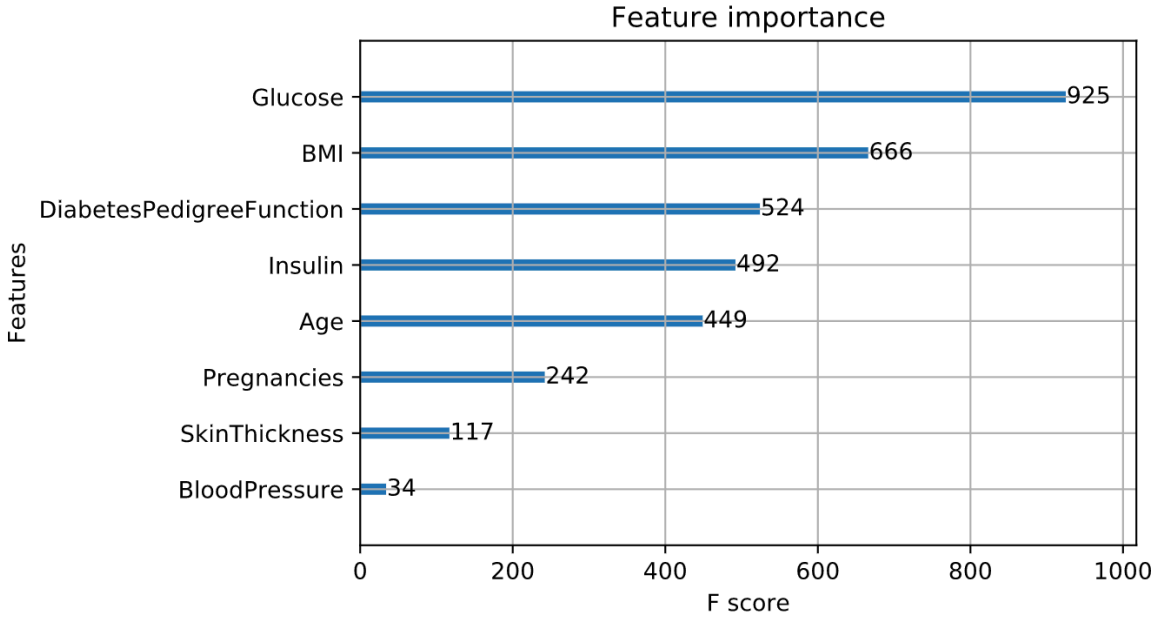

Figure 5: This figure shows how the features affect the outcome.

From the first look I excluded “Pregnancies, SkinThickness and BloodPressure” because they have the least importance and correlation values. And then I started testing and repeating this process with different combinations of the features. The best combination was “Glucose, BMI and Age” as shown in figure 6.

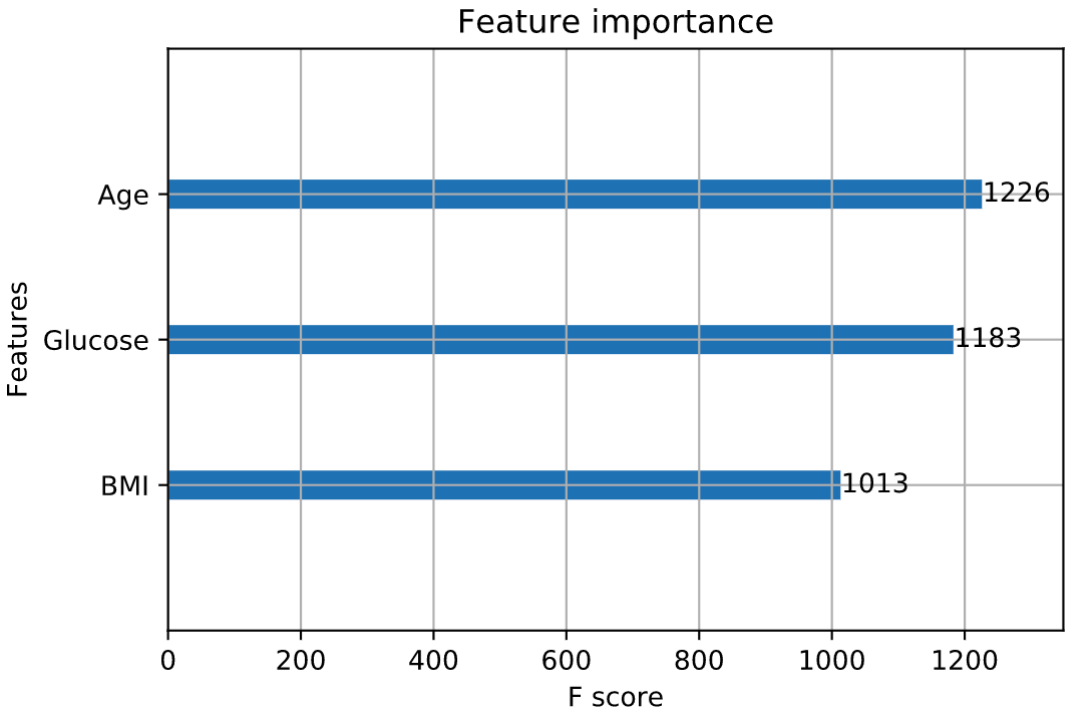

Figure 6: Feature importance of the chosen features "Age, Glucose and BMI"

Implementation

XGBoost is an implementation of gradient boosted decision trees (similar to the one shown in figure 7 [10] designed for speed and performance and XGBoost has its own python programming language library.

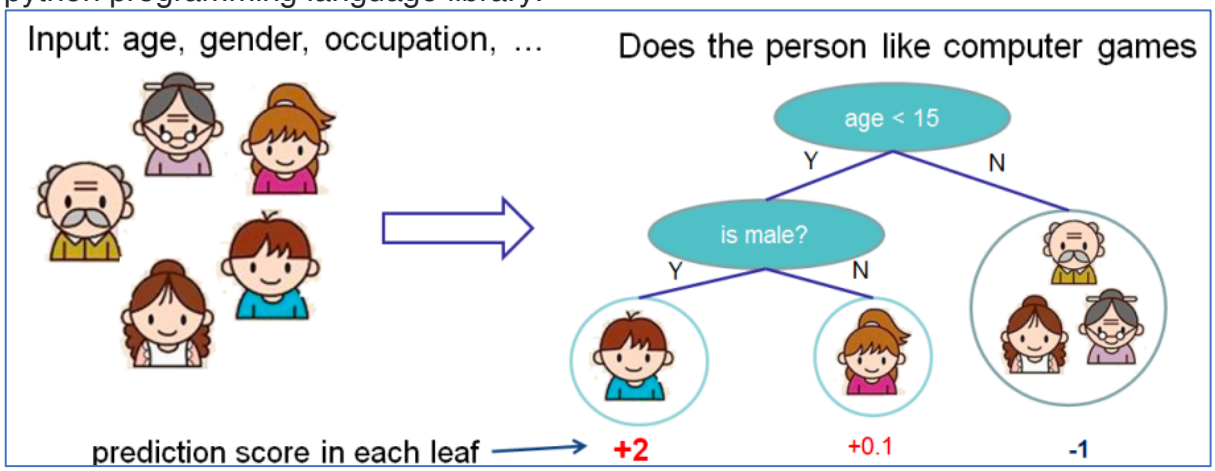

Figure 7: Example of Gradient boosted decision tree

For ease of implementation, I will use scikit-learn which is an open source machine learning python programming language library. Scikit-learn facilitate the use of several machine learning and data mining algorithms. It also has many evaluation methods to test the implemented model.

To use the full scikit-learn library functionality I had to use a wrapper model for XGBoost classification which is XGBClassifier.

In the implementation phase, I trained the XGBclassifier on the preprocessed data with only three features “Age, Glucose and BMI”. The implementation steps are as follows:

1. Take the original dataset and split it into training set and testing set (80% for training and 20% for testing).
   ```
   import xgboost
   # create a train/test split
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=7)
   ```

2. Initialize the XGB classifier model and its parameters.
   ```
   model = xgboost.XGBClassifier(max_depth=3, learning_rate=0.005, n_estimators=500)
   ```

3. fit the model on the training set.
   ```
   model.fit(X_train, y_train)
   ```

4. Predict the values for the testing set.
   ```
   prediction = model.predict(X_test)
   ```
5. Iterate and try steps (2,3,4) until.
6. Evaluate the model by calculating the scores from the classification report.
   ```
   from sklearn.metrics import classification_report
   print(classification_report(y_test, prediction, target_names=["0","1"] ))
   ```

	precision	recall	f1-score	support
0	0.86	0.88	0.87	96
1	0.79	0.76	0.77	58
avg / total	0.83	0.83	0.83	154

I came across several issues when I used the model. One of these issues is, it took me a while to understand the difference between using the wrapper model “XGBClassifier” and using XGBoost immidiatly. The interaction becomes slightly different than the original XGBoost documentation. For example, some methods such as the built-in cross validation are replaced with scikit-learn methods such as gradient search cross validation method. Another issue I faced is when I was choosing the right parameters for the training, XGBoost has a massive parameter set and each of these parameters has its own effect on the model’s performance and the combination of the parameter also has its effects on the result. This part took me a lot of time and many iterations over several days has been done in order to get a decent result.

Refinement

The initial implementation of XGBclassifier with the default parameters resulted in 0. 76 f1 score. Which is already above the benchmark, but it showed me there is a pote ntial for improvement, so I started investigating as shown in table 2.

Table 2 Parameter tuning report.

Learning_rate	Max_depth	n_estimators	F1_score
0.1	3	100	0.76
0.1	5	500	0.75
0.01	5	500	0.78
…	…	…	…
0.005	3	500	0.83

After several iterations, I reached the best result which is : 0.83.

IV. Results

Model Evaluation and Validation

The final model has 83% F1_score and 83% accuracy on 0.05 learning rate, 3 max depth and 500 n estimators. The model has been tested using k-fold to change the inputs of the evaluation and it resulted in 76% accuracy which is higher. The model is robust and can be trusted, because accuracy doesn't change a lot when using different inputs.

Justification

My final result score is 0.83, which is better than then the benchmark score of 0.70. The improvement is about 18.5 %. Since the score is better, the classification is more accurate. This means that the foundation for a diabetes diagnosis system is improved and a more accurate warning can be given.

V. Conclusion

Free-Form Visualization

In this section, I will present 2 visualization of the results, the first one is figure 8 which shows the classification tree of the model. The second visualization is figure 9 which shows a confusion matrix for the results.

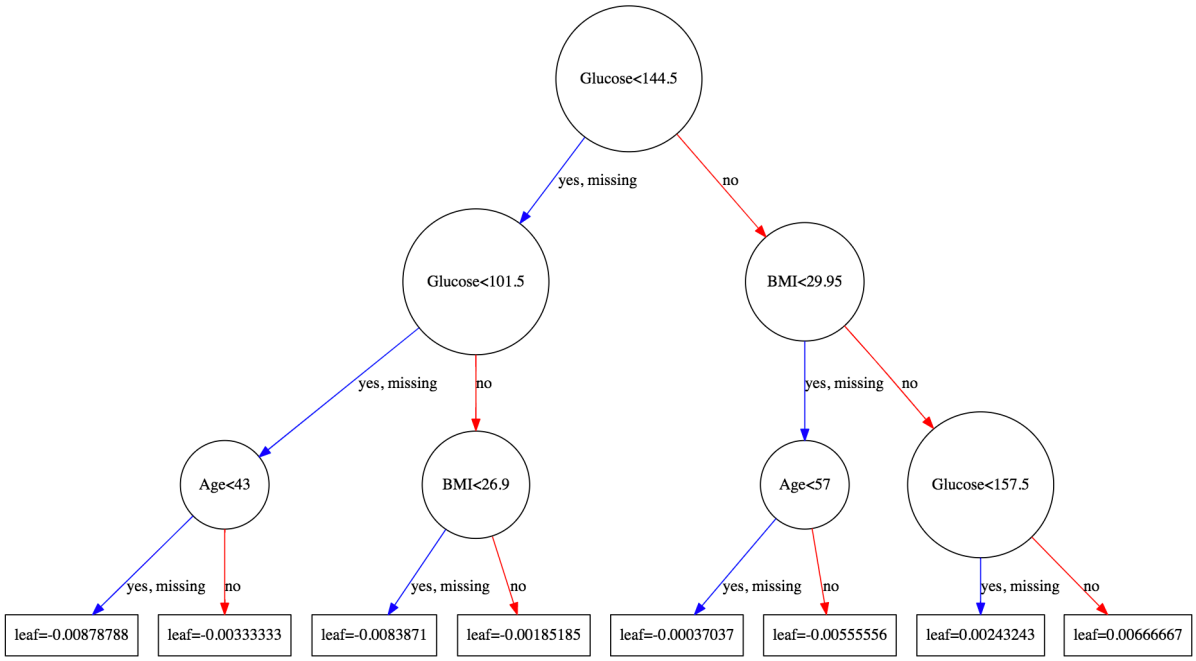

Figure 8: A classification tree that shows how the model works.

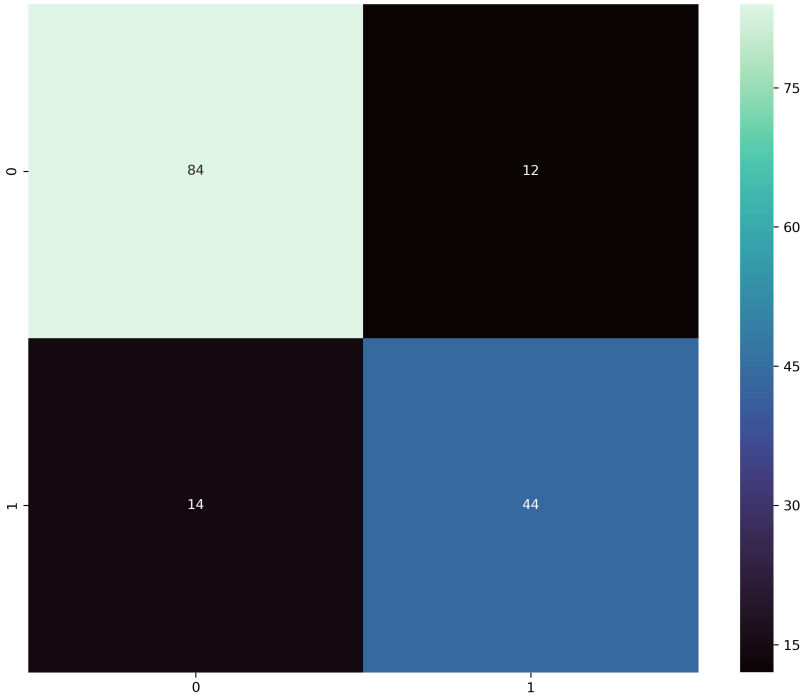

Figure 9: A confusion matrix to show the performance of the model.

Reflection

The process used for the project can be summarized using the following steps:

1. Define the problem.
2. Find a public dataset.
3. Download the dataset.
4. Define the benchmark and evaluation metric.
5. Investigate the dataset.
6. Prepare the data for the classifier.
7. Train the classifier using the data (Iterate, until a good result is achieved).
8. Use the trained model for prediction.
9. Evaluate the model.

The most difficult steps are 6 and 7, as I had to discover the dataset and the parameter variation of the model. These two steps are also the most important steps for the achievement of this model. I learned a lot about XGBoost model, parameter tuning, data preprocessing and feature selection.

Improvement

Even though the results are encouraging. Working on improving this model by investigating and tuning more parameters would result in better outcome. I would try more feature combinations and more parameter tuning for each of these combinations. I also would like to try this model on another dataset. I am sure that this model can be improved and we can probably achieve higher accuracy if the right parameters and features were in place. Trying another algorithm would also help in achieving better results, I would like to try SVM classification for this problem.

Reference

[1] "Diabetes: the basics | Diabetes UK." [Online]. Available: https://www.diabetes.org.uk/diabetes-the-basics. [Accessed: 11-Mar-2018].

[2] International Diabetes Federation, *IDF Diabetes Atlas Eighth Edition 2017*. 2017.

[3] M. O'Grady, "Diabetes, A National Plan for Action," *Natl. Diabetes Action Plan*, no. July 17, 2012, 2004.

[4] I. Kavakiotis, O. Tsave, A. Salifoglou, N. Maglaveras, I. Vlahavas, and I. Chouvarda, "Machine Learning and Data Mining Methods in Diabetes Research," *Comput. Struct. Biotechnol. J.*, vol. 15, pp. 104–116, 2017.

[5] L. Han, S. Luo, J. Yu, L. Pan, and S. Chen, "Rule Extraction From Support Vector Machines Using Ensemble Learning Approach: An Application for Diagnosis of Diabetes," *IEEE J. Biomed. Heal. Informatics*, vol. 19, no. 2, pp. 728–734, Mar. 2015.

[6] A. Ozcift and A. Gulten, "Classifier ensemble construction with rotation forest to improve medical diagnosis performance of machine learning algorithms," *Comput. Methods Programs Biomed.*, vol. 104, no. 3, pp. 443–451, Dec. 2011.

[7] S. Bashir, U. Qamar, and F. H. Khan, "IntelliHealth: A medical decision support

application using a novel weighted multi-layer classifier ensemble framework," *J. Biomed. Inform.*, vol. 59, pp. 185–200, Feb. 2016.

[8] J. P. Anderson *et al.*, "Reverse Engineering and Evaluation of Prediction Models for Progression to Type 2 Diabetes," *J. Diabetes Sci. Technol.*, vol. 10, no. 1, pp. 6–18, Jan. 2016.

[9] "sklearn.metrics.f1_score — scikit-learn 0.19.1 documentation." [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score. [Accessed: 11-Mar-2018].

[10] "Introduction to Boosted Trees — xgboost 0.7 documentation." [Online]. Available: http://xgboost.readthedocs.io/en/latest/model.html. [Accessed: 28-Mar-2018].