İstanbul Bilgi Üniversitesi

CMPE 343 - Spring 2020

PIMA Indian Diabetes Dataset Analysis

Atakan Kaya (115200074)

Volkan Arıslı (116200051)

Selin Yeşilselve (115200041)

Mert İncesu (117200083)

Course Teacher: Savaş YILDIRIM

Faculty of Engineering and Natural Sciences

Istanbul Bilgi University

2020

Aim

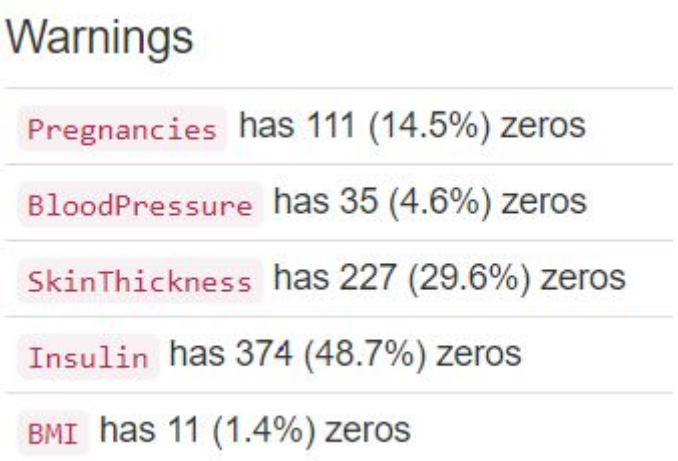
We are going to build a system that estimates whether a patient has diabetes or not depends on related criteria. With this project, we are aiming to reduce the wasted time of diagnosing for a patient. According to this solution, the number of patients that doctors will examine decrease and the time that they will spend per patient will be decreased.

Data Preprocessing

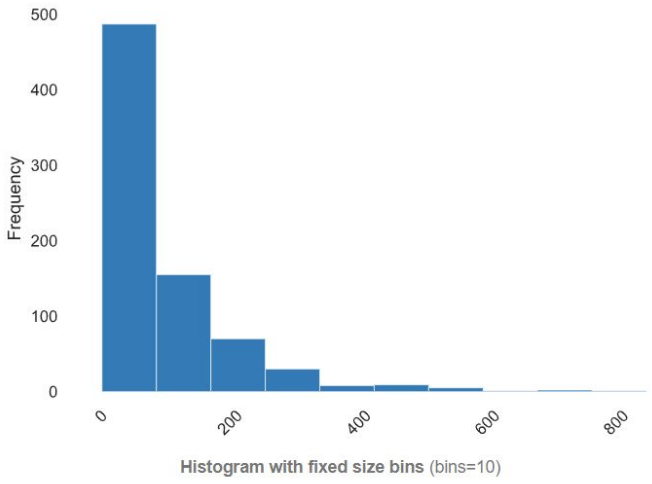
Our analyze will be held on the Pima Indians Diabetes Database is taken from Kaggle. The dataset is to diagnostically predict whether or not a patient has diabetes depends on 8 criteria as the number of times pregnant, plasma glucose concentration a 2 hours in an oral glucose tolerance test, diastolic blood pressure (mm Hg), Triceps skinfold thickness (mm), 2-Hour serum insulin (mu U/ml), Body mass index (weight in kg/(height in m)^2), diabetes pedigree function and age. And categorize person either has diabetes or not.

The characteristics of the dataset that we used are mentioned below. Now we need to examine the characteristics of the variables to observe if there exist abnormal values. To begin with this examination, we will use ‘pandas profiling’ which is a module that direct us to more detailed information about the dataset and variables in it.

Now we will analyze each property separately.



According to the above screenshot from the report, we can observe that there exist some 0 values in each column, which is not efficient. When we discuss more Insulin values, we are able to examine the frequencies of the zero values is really high from the below histogram.



There is 2 different way to handle this situation. We may get rid of from the data equals 0 which acquires loss of data or we may change the zeros with the median of the values. To deal with this situation, we choose to change zeros with the median values.

```
# Calculate the median value for BMI
median_bmi = dataset['BMI'].median()
# Substitute it in the BMI column of the
# dataset where values are 0
dataset['BMI'] = dataset['BMI'].replace(to_replace=0, value=median_bmi)

# Calculate the median value for BloodP
median_bloodp = dataset['BloodPressure'].median()
# Substitute it in the BloodP column of the
# dataset where values are 0
dataset['BloodPressure'] = dataset['BloodPressure'].replace(to_replace=0, value=median_bloodp)

# Calculate the median value for PlGlcConc
median_plglcconc = dataset['Glucose'].median()
# Substitute it in the PlGlcConc column of the
# dataset where values are 0
dataset['Glucose'] = dataset['Glucose'].replace(to_replace=0, value=median_plglcconc)

# Calculate the median value for SkinThick
median_skinthick = dataset['SkinThickness'].median()
# Substitute it in the SkinThick column of the
# dataset where values are 0
dataset['SkinThickness'] = dataset['SkinThickness'].replace(to_replace=0, value=median_skinthick)

# Calculate the median value for TwoHourSerIns
median_twohourserins = dataset['Insulin'].median()
# Substitute it in the TwoHourSerIns column of the
# dataset where values are 0
dataset['Insulin'] = dataset['Insulin'].replace(to_replace=0, value=median_twohourserins)
```

During the training part, the values in different ranges can dominate each other. To prevent this, we will scale our values using feature scaling technique as Standard Scaler. This technique put values in the same range.

```
#Splitting the data into dependent and independent variables
Y = dataset.Outcome
x = dataset.drop('Outcome', axis = 1)
columns = x.columns

#feature scalization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(x)
data_x = pd.DataFrame(X, columns = columns)
```

Model Fitting

We used classification models such as Naive Bayes, Support Vector Machine(kernel, linear), Grid Search, Random Forest, Logistic Regression, K-Nearest Neighbors (K-NN), Decision Tree Classification on our labelled dataset. These algorithms have individual techniques of prediction. Before making a prediction we split the processed dataset into Training and Test data. The Test data size is taken to be 20% of the entire data. Because of the reason that it is a small dataset with fewer columns, we did not choose to apply Feature Selection technique such as PCA.

```
# ------------Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
classifier_RFC = RandomForestClassifier(n_estimators=300, bootstrap = True, max_features = 'sqrt')
classifier_RFC.fit(x_train, y_train)
y_pred_RFC = classifier_RFC.predict(x_test)

# ------------Logistic Regression
from sklearn.linear_model import LogisticRegression
classifier_LR = LogisticRegression(random_state = 0)
classifier_LR.fit(x_train, y_train)
y_pred_LR = classifier_LR.predict(x_test)

# ------------K-Nearest Neighbors (K-NN)
from sklearn.neighbors import KNeighborsClassifier
classifier_KNN = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier_KNN.fit(x_train, y_train)
y_pred_KNN = classifier_KNN.predict(x_test)

# ------------Support Vector Machine (SVM)
from sklearn.svm import SVC
classifier_SVC = SVC(kernel = 'linear', random_state = 0)
classifier_SVC.fit(x_train, y_train)
y_pred_SVC = classifier_SVC.predict(x_test)

# ------------Kernel SVM
from sklearn.svm import SVC
classifier_SVC_rbf = SVC(kernel = 'rbf', random_state = 0)
classifier_SVC_rbf.fit(x_train, y_train)
y_pred_SVC_rbf = classifier_SVC_rbf.predict(x_test)

# ------------Naive Bayes
from sklearn.naive_bayes import GaussianNB
classifier_NB = GaussianNB()
classifier_NB.fit(x_train, y_train)
y_pred_NB = classifier_NB.predict(x_test)

# ------------Decision Tree Classification
from sklearn.tree import DecisionTreeClassifier
classifier_DTC = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier_DTC.fit(x_train, y_train)
y_pred_DTC = classifier_DTC.predict(x_test)
```

Evaluation

	Accuracy	F1 Score	Precision Score	Recall Score
Random Forest Classifier	0.77	0.75	0.75	0.75
Logistic Regression	0.73	0.70	0.71	0.70
K-Nearest Neighbors	0.69	0.66	0.66	0.66
Support Vector Machine (linear)	0.73	0.70	0.71	0.70
Decision Tree Classifier	0.64	0.60	0.60	0.60
Kernel SVM	0.74	0.71	0.72	0.70
Naive Bayes	0.72	0.69	0.69	0.69

As we can observe from the table Random Forest Classifier is the right model because of high accuracy, precision score, f1 score and recall score. The reason to scores are close to each other separately for each model is might be caused by data preprocessing part. There can apply different advanced preprocessing techniques to get more efficient results.
The confusion matrix for the chosen classification model(Random Forest Classifier) is mentioned below.