RULE-BASED DIABETES DIAGNOSTIC SYSTEM

BY: ALEXIS CASTELLANOS

MAY 31, 2021

UOFM DEARBORN

CIS 479

Overview

This project aims to create a rule-based system which handles diagnosis of diabetes in patients. Diabetes is a chronic condition in which blood glucose levels are consistently elevated. Diabetes is traditionally split into two sub-categories Type 1, and Type 2. For this project, we will be focus on the binary classification of diabetes positive or negative results. This task was originally a Kaggle challenge under the name Pima Indians Diabetes. The winner of this competition reached an accuracy of 90.6% using LightGBM(5 Fold) & KNN. We will be using the same dataset provided by National Institute of Diabetes and Digestive and Kidney Diseases.

Expert Source Described:

We will be basing our rule-based system on a decision tree classifier (base). The decision tree will be created using the machine learning library sklearn. Sklearn's decision tress is a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The result of a non-tuned decision tree on our classification problem is shown in figure 1.0. The result is comparable to a non-tuned KNN model (tuned version currently holds Kaggle's highest accuracy) shown in Figure 1.1.

Decision Tree Figure 1.0

```
from sklearn import tree
from sklearn.metrics import confusion_matrix
tree_model = tree.DecisionTreeClassifier()
tree_model.fit(X_train, y_train.ravel())

y_pred = tree_model.predict(X_test)
print("Accuracy: ",tree_model.score(X_test,y_test))
confusion_matrix(y_test,y_pred)
```

```
Accuracy:  0.70995670995671
```

KNN Figure 1.1

```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_neighbors=6)
knn_clf.fit(X_train,y_train.ravel())
knn_clf.score(X_test,y_test)
```

```
0.6926406926406926
```

Data and Pre-Processing

Inference Strategy Described:

The tools used for this project are python 3 under a Jupyter Notebook environment. The pandas python module will be used to create all data frames used. Machine learning models imported for our task will be form sklearn's library. The features we are provided are the following:

Figure 2.0

Feature	Description	Unit of Measurement
Pregnancies	Number of times pregnant	Numeric
Glucose	Blood Sugar concentration over 2 hours in *Oral Glucose Tolerance Test*	mg/dL
Blood Pressure	Diastolic blood pressure test	mm Hg

Skin Thickness	Bodyfat skin fold test (triceps)	mm
Insulin	Serum insulin levels (2 hours)	mu U/ml
BMI	Body mass index Formula: (kg (weight)/ meters (height)2)	Kg/m(2)
Diabetes Pedigree Function	Likelihood if diabetes based on family history	Percent Probability
Age	In years	Numeric

Using Mayo Clinic as a medica reference, I create a python data frame with traditional medical standards for the following features: Glucose, Blood Pressure, Skin Thickness, BMI.

Figure 2.1

Medical Standards

```
normal = ["less than 140mg/dL","less than 80 mm Hg","less than 25mm","less than 24"]
concerning = ["140-190 mg/dL","80-89 mm Hg","up to 32 mm","25-29"]
problamatic = ["200+ mg/dL","90+ mm Hg","33+ mm","30+"]
```

```
med_df = pd.DataFrame([normal,concerning,problamatic],columns = ['Glucose (mg/dL)', 'Blood Pressure (mm Hg)','Skin T
```

```
med_df.index = ['Normal','Concerning','Problamatic']
```

```
med_df
```

	Glucose (mg/dL)	Blood Pressure (mm Hg)	Skin Thickness (mm)	BMI
Normal	less than 140mg/dL	less than 80 mm Hg	less than 25mm	less than 24
Concerning	140-190 mg/dL	80-89 mm Hg	up to 32 mm	25-29
Problamatic	200+ mg/dL	90+ mm Hg	33+ mm	30+

Using the dataset provided, we can create the data frame and data statistics shown below:

Figure 2.2

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Diabetes
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

Conflict and Resolution:

Further examining the data frame, we can conclude there are 768 total data points, 500 diabetic and 268 non-diabetic. However, it is important to note some features in our data are recorded to be zero. Hence, patients may present up to 8 the features but not required.

Figure 3.0

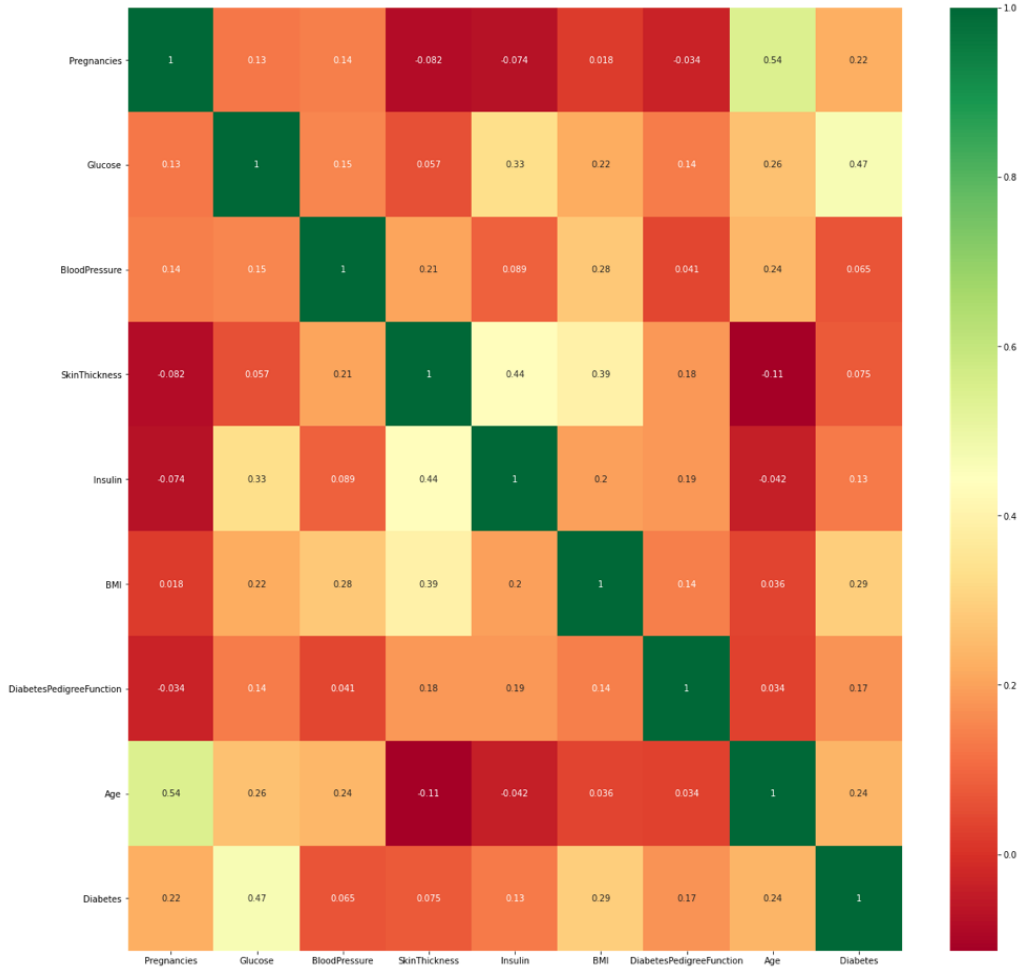
Data Pre-processing- Missing Data

```
print("Total number of rows : {0}".format(len(data)))
print("Total number of Positive Diabetes Patients: {0}".format(len(data.loc[data['Diabetes']==0])))
print("Total number of Negative Diabetes Patients: {0}".format(len(data.loc[data['Diabetes']==1])))
print("Number of rows missing Pregnancies: {0}".format(len(data.loc[data['Pregnancies'] == 0])))
print("Number of rows missing Glucose: {0}".format(len(data.loc[data['Glucose'] == 0])))
print("Number of rows missing BloodPressure: {0}".format(len(data.loc[data['BloodPressure'] == 0])))
print("Number of rows missing Insulin: {0}".format(len(data.loc[data['Insulin'] == 0])))
print("Number of rows missing BMI: {0}".format(len(data.loc[data['BMI'] == 0])))
print("Number of rows missing DiabetesPedigreeFunction: {0}".format(len(data.loc[data['DiabetesPedigreeFunction'] ==
print("Number of rows missing Age: {0}".format(len(data.loc[data['Age'] == 0])))
print("Number of rows missing SkinThickness: {0}".format(len(data.loc[data['SkinThickness'] == 0])))
```

```
Total number of rows : 768
Total number of Positive Diabetes Patients: 500
Total number of Negative Diabetes Patients: 268
Number of rows missing Pregnancies: 111
Number of rows missing Glucose: 5
Number of rows missing BloodPressure: 35
Number of rows missing Insulin: 374
Number of rows missing BMI: 11
Number of rows missing DiabetesPedigreeFunction: 0
Number of rows missing Age: 0
Number of rows missing SkinThickness: 227
```

To overcome missing features in data points we may **impute** the missing values. Imputing is the process of inferring the missing data from the known portion of the data. I used a multivariate imputation algorithm (iterative imputer) to use the entire set of available feature dimensions, estimating the missing values denoted by 0. Correlation between all features and diabetes is represented by the heatmap below:

Figure 3.1



From the figure above we can see the five highest diabetes correlation features are: Glucose, BMI, Age, Diabetes and Diabetes Pedigree Function. The correlation matrix provides us a brief overview for resolution or deciding what features hold higher importance.

Test Cases:

Using sklearn, the data was split into 70% training and 30% testing. The tree model will be trained using the training set. Both rule-based and tree models will be tested on the same test set that was randomly selected. Since the rule-based system was based on the tree model, we can expect similar accuracy results. The figure below shows how the data set was split for training and testing sets.

Figure 4.0

```
# Train Test Split
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

feature_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age',
                   'SkinThickness']
predicted_class = ['Diabetes']

X = data[feature_columns].values
y = data[predicted_class].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state=10)
```

Rule Base Listing Complete:
Shown below is the rule-based system's python script

Figure 5.0

In [37]:
```
#Sample is an array containinig a sequence of features
def rule_based_sys(sample):
    #Feature 1 is Glucose
    if sample[1] <= 143.5:
        #Feature 7 is Age
        if sample[7] <= 30.5:
            #Feature 5 is BMI
            if sample[5] <= 31.40000057220459:
                #Feature is Pregnancies
                if sample[0] <= 7.5:
                    print("node=4 is a leaf node")
                    return 0
                else:
                    print("node=5 is a leaf node")
                    return 0
            else:
                #Feature 2 is Blood Pressure
                if sample[2] <= 37.0:
                    print("node=7 is a leaf node.")
                    return 0
                else:
                    print("node=8 is a leaf node.")
                    return 1
        else:
            #Feature 1 is Glucose
            if sample[1] <= 99.5:
                #Feature 5 is BMI
                if sample[5] <= 34.650001525878906:
                    print("node=11 is a leaf node.")
                    return 0
                else:
                    print("node=12 is a leaf node.")
                    return 1
            else:
                #Feature 5 is BMI
                if sample[5] <= 26.899999618530273:
                    print("node=14 is a leaf node.")
                    return 0
                else:
                    print("node=15 is a leaf node.")
                    return 1
    else:
        #Feature 1 is Glucose
        if sample[1] <= 159.5:
            #Feature 6 is Diabetes Pedigree Function
            if sample[6] <= 0.3149999976158142:
                #Feature is Blood Pressure
                if sample[2] <= 55.0:
                    print("node=19 is a leaf node.")
                    return 1
                else:
                    print("node=20 is a leaf node.")
                    return 0
            else:
                #Feature 7 is Age
                if sample[7] <= 45.0:
                    print("node=22 is a leaf node.")
                    return 1
                else:
                    print("node=23 is a leaf node.")
                    return 0
        else:
            #Feature 7 is Age
            if sample[7] <= 64.0:
                #Feature 4 is Insulin level
                if sample[4] <= 629.5:
                    print("node=26 is a leaf node.")
                    return 1
                else:
                    print("node=27 is a leaf node.")
                    return 0
            else:
                #Feature 6 is Diabetes Pedigree Function
                if sample[6] <= 0.29099999368190765:
                    return 1
                else:
                    return 0
```

In [43]:
```
rule_pred_r =[]
for x in range(len(X_test)):
    rule_pred = rule_based_sys(X_test[x])
    rule_pred_r.append(rule_pred)
```

Expert System Runs and Exercises all Rules:

Decision Tree

Figure 1.0

To the right we can see the results of our expert system. The decision tree created is a binary tree we used to create our rule-based system.

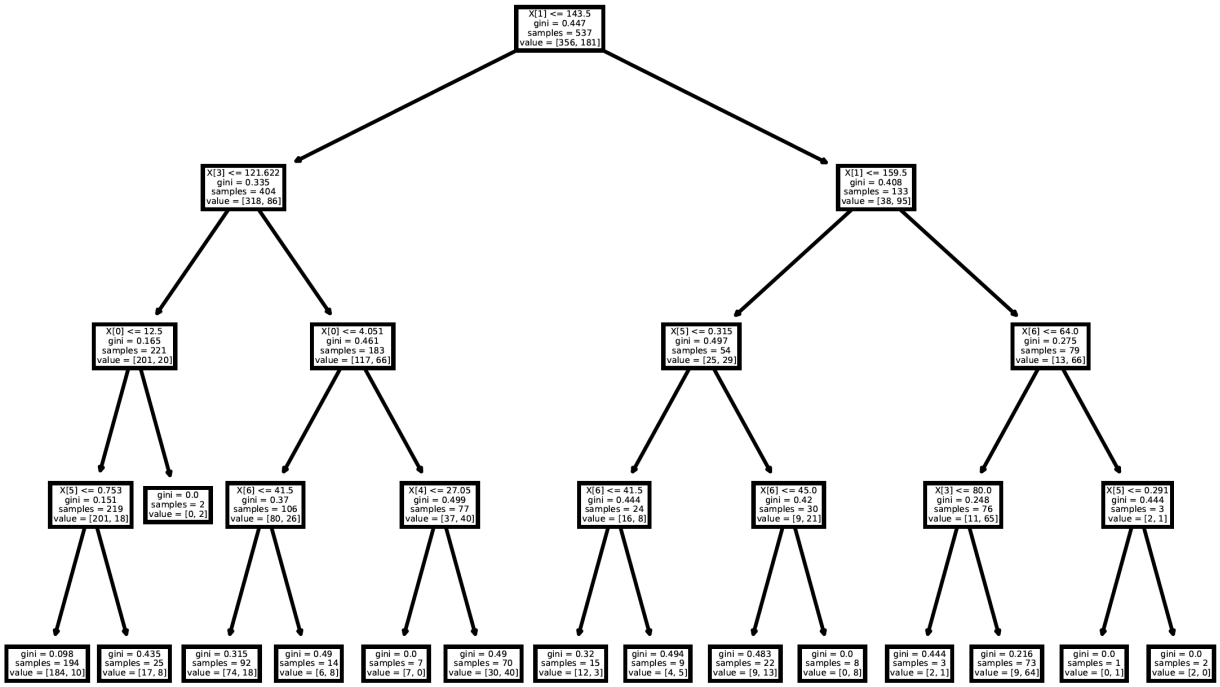
Below we can view the plot of this tree model after training. The feature a tree is currently evaluation in a node is denoted as X[Feature].

```
from sklearn import tree
from sklearn.metrics import confusion_matrix
tree_model = tree.DecisionTreeClassifier()
tree_model.fit(X_train, y_train.ravel())

y_pred = tree_model.predict(X_test)
print("Accuracy: ",tree_model.score(X_test,y_test))
confusion_matrix(y_test,y_pred)
```

```
Accuracy:  0.70995670995671
```

Figure 6.0



Rule Base Comments:

The structure below displays the decision process used to create our rule-based system. This deci sion process from our tree model using encoding regression.

node=0 is a split node: go to node 1 if Glucose <= 143.5 else to node 14.
node=1 is a split node: go to node 2 if Insulin <= 121.62222671508789 else to node 7.
node=2 is a split node: go to node 3 if Pregnancies <= 12.5 else to node 6.
node=3 is a split node: go to node 4 if DiabetesPedigreeFunction <= 0.7525000274181 else to node 5.
node=4 is a leaf node.
node=5 is a leaf node.

node=6 is a leaf node.

node=7 is a split node: go to node 8 if Pregnancies <= 4.051208019256592 else to node 11.
node=8 is a split node: go to node 9 if Age <= 41.5 else to node 10.
node=9 is a leaf node.
node=10 is a leaf node.

node=11 is a split node: go to node 12 if BMI <= 27.050000190734863 else to node 13.
node=12 is a leaf node.
node=13 is a leaf node.

node=14 is a split node: go to node 15 if Glucose <= 159.5 else to node 22.
node=15 is a split node: go to node 16 if DiabetesPedigreeFunction <= 0.3149999976158142 else to node 19.
node=16 is a split node: go to node 17 if Age <= 41.5 else to node 18.
node=17 is a leaf node.
node=18 is a leaf node.

node=19 is a split node: go to node 20 if Age <= 45.0 else to node 21.
node=20 is a leaf node.
node=21 is a leaf node.

node=22 is a split node: go to node 23 if Insulin <= 80.0 else to node 26.
node=23 is a split node: go to node 24 if DiabetesPedigreeFunction <= 1.3805000334 else to node 25.
node=24 is a leaf node.
node=25 is a leaf node.

node=26 is a split node: go to node 27 if Age <= 64.0 else to node 28.
node=27 is a leaf node.
node=28 is a leaf node.

Rules used to predict positive diabetes patient:

decision node 0 : (X_test[0, Glucose] = 154.0) > 143.5)
decision node 14 : (X_test[0, Glucose] = 154.0) <= 159.5)
decision node 15 : (X_test[0, DiabetesPedigreeFunction] = 0.33799999999999997) > 0.3149999976158142)
decision node 19 : (X_test[0, Age] = 37.0) <= 45.0)

Rules used to predict negative diabetes patient:

decision node 0 : (X_test[1, Glucose] = 112.0) <= 143.5)
decision node 1 : (X_test[1, Insulin] = 160.0) > 121.62222671508789)
decision node 7 : (X_test[1, Pregnancies] = 2.0) <= 4.051208019256592)
decision node 8 : (X_test[1, Age] = 28.0) <= 41.5)

Discussion of Performance on Test Cases:

Figure 7.0

To the right we can view the accuracy score of our rule-based system. The rule-based system gives us a score of 70.56%. This is very comparable to our Decision Tree Model's accuracy of 70.99% shown in figure 1.0.

```
def rule_based_score():
    flag = 0

    for f, b in zip(rule_pred_r, y_test):
        if f != b:
            flag = flag+1

    acc = (len(rule_pred_r)-flag)/len(y_test)

    print("Rule Based System Accuracy: ",acc)
```

```
rule_based_score()
```

```
Rule Based System Accuracy:  0.7056277056277056
```

Our rule-based system does not contain nodes, it is based on conditional statements. I believe the slight difference in performance due to the limitation of conditional statements (*if else*). The nodes in the tree-based model are constantly adjusting the thresholds of each feature per node. In contrast

our rule-based system does not have this iterative adjustment. Instead, it was based on the final nodes and feature thresholds.

Attempts to increase accuracy of rule-based system using medical standard (figure 2.1) were not successful. This resulted in a significantly lower accuracy score (64%). After further analyzing, I found about half of all diabetic data samples had a normal blood glucose level (less than 140 mg/dL). I believe this poses a clear obstacle in all diabetic diagnostics since glucose levels are traditionally used the means of diagnosis.

In conclusion, the rule-based system performed relatively close to the decision tree (expert system) it was modeled after. Further improvements the system could benefit from are adjusting depth and pruning. Possible reasons for only a 70% accuracy performance is the limited data et and missing features in some instances. However, Kaggle challenges are based on real life scenarios where noise and unsorted data exist.