Performance analysis of machine learning models to predict diabetes

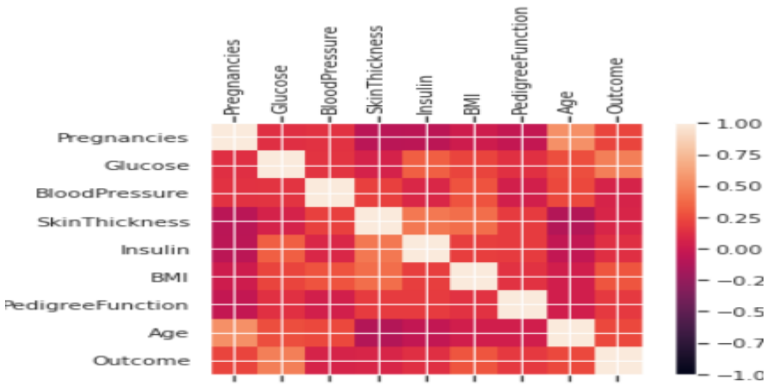
Introduction: Diabetes is one of the world's most common and well-known life-threatening diseases[1]. Diabetes detected early can help to decrease complications and the risk of a wide range of health issues. The purpose of this project is to predict diabetes based on various parameters using machine learning models such as logistic regression, decision tree, naïve bayes, adaboost and to evaluate their accuracy to determine which model performs best in predicting diabetes.

Method:

i) Dataset and preprocessing:

Pima Indian diabetes dataset has been used to classify diabetic and non-diabetic patient based on machine learning models. This dataset has a total of 768 training cases. Each training instance includes eight features and a class variable that serves as the training instance's label. Pregnancies, glucose, blood pressure, BMI, skin thickness, insulin, age, and diabetes pedigree function are the independent variables. The feature that will be predicted is the outcome, where 0 indicates that the patient does not have diabetes and 1 indicates that the patient does have diabetes. There is no null value in the dataset.

ii) Correlation:



A significant correlation can be observed between pregnancy and age in the correlation matrix

iii) Normalization and splitting dataset: The data is normalized from 0 to 1. Following normalization, the dataset is divided into two parts: training and testing. 80% of the data is utilized to train the model, with the remaining 20% used for testing.

iv) Applying machine learning models:

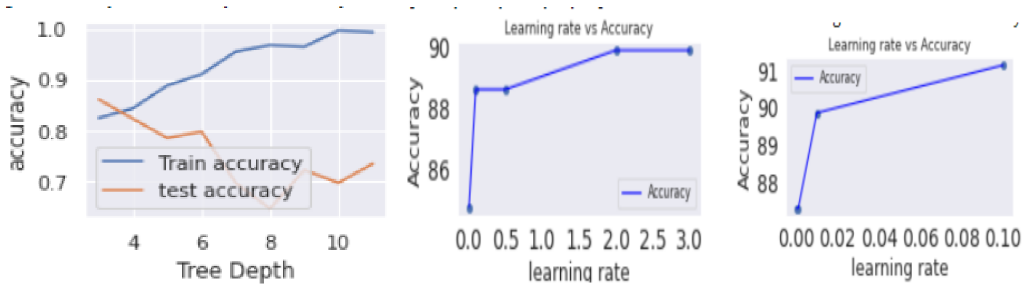
Machine learning models that are applied to predict diabetes are follows:

a) Logistic regression: Logistic regression is a type of supervised machine learning. In this experiment, multiple learning rates were used to assess test accuracy. With a learning rate of 2, the best accuracy is 89.87%.

b) Decision tree: Decision tree is a non-parametric classifier. The accuracy of the tree is measured at various depths. When depth=3, the decision tree has the highest accuracy of 86.08%.

c) Naïve Bayes: The Naive Bayes model is based on the Bayes theorem. In Naïve Bayes data are categorized by computing the probability of the output depending on the attributes. Naive Bayes has an accuracy of 84.81% in predicting diabetes from a dataset.

d) Adaboost: Adaptive Boosting, often known as "AdaBoost," focuses on classification problems and seeks to convert a group of weak classifiers into a powerful one[2].For predicting diabetes, accuracy of adaboost is 91.13% for learning rate 0.1



a)Decision tree b) logistic regression c) Adaboost

Figure 2: HyperTune decision tree, logistic regression, Adaboost for different test accuracy

Result: Adaboost has the highest accuracy of 91.13% among four machine learning models. To predict diabetes from the dataset, logistic regression, naive bayes, and decision tree all perform well. Precision, recall and f1 score of adaboost are 0.91,0.88, 0.89 respectively which are higher than remaining three models. The feature importance of the attributes was analyzed in the AdaBoost model, and it was discovered that glucose has the most significant influence in the model for prediction. Comparison among various models is shown below

	Model Name	Test Accuracy	Precision	Recall	f1-score
0	Logistic Regression	89.873418	0.890678	0.841481	0.860177
1	Naive Bayes	84.810127	0.822293	0.835185	0.828012
2	Decision tree	86.076000	0.836054	0.865926	0.846602
3	Adaboost	91.139241	0.910686	0.881481	0.894000

Table 1: Comparison among four models

Conclusion: The results show that there is no significant difference in accuracy between the four models. On the dataset, all models perform well, with adaboost performing the best. Because of the nature of the data in the dataset, it is not possible to distinguish between type 1 and type 2 databases. The goal for the future is to explore various features and determine the type of diabetes. Furthermore, these four models can be integrated with other machine learning models to improve prediction accuracy.

Bibliography:

1) Butt UM, Letchmunan S, Ali M, Hassan FH, Baqir A, Sherazi HHR. Machine Learning Based Diabetes Classification and Prediction for Healthcare Applications. *J Healthc Eng.* 2021 Sep 29;2021:9930985. doi: 10.1155/2021/9930985. PMID: 34631003; PMCID: PMC8500744.

2)N. H. Taz, A. Islam and I. Mahmud, "A Comparative Analysis of Ensemble Based Machine Learning Techniques for Diabetes Identification," *2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, 2021, pp. 1-6, doi: 10.1109/ICREST51555.2021.9331036.

Performance analysis of machine learning models to predict diabetes

import necessary libraries

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
#for ML model
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.ensemble import AdaBoostClassifier
sns.set(color_codes =True)
%matplotlib inline
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Load dataset

```
data = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
data.head()
```

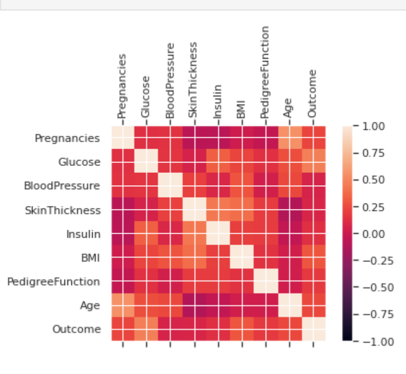
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Find Correlation

```
correlations = data.corr(method = 'pearson') # Correlations between all pairs of attributes
# Print the datatype
type(correlations)
# Show the correlation matrix
correlations
# import required
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

```
import numpy as np

# plot correlation matrix
col=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","PedigreeFunction","Age","Outcome"]
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
#names = data.columns
#names = [for i in col]
ax.set_xticklabels(col,rotation=90) # Rotate x-tick labels by 90 degrees
ax.set_yticklabels(col)
plt.show()
```



Data Normalization

```
data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI']] = data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']].replace(0, np.NaN)
data.dropna(inplace=True)
y = data["Outcome"].values
X = data.iloc[:,0:8].values
#x_data = data.drop(["Outcome"], axis=1)
#x_data = data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI']]
scalerX = StandardScaler().fit(X)
x = scalerX.transform(X)
X1=data.iloc[:,0:8]
X1
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53
13	1	189.0	60.0	23.0	846.0	30.1	0.398	59
...	...	...	...	...	...	...	...	...
753	0	181.0	88.0	44.0	510.0	43.3	0.222	26
755	1	128.0	88.0	39.0	110.0	36.5	1.057	37
760	2	88.0	58.0	26.0	16.0	28.4	0.766	22
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30

392 rows × 8 columns

Split training and test set

```
# train test split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 89)
```

Naive Bayes classifier

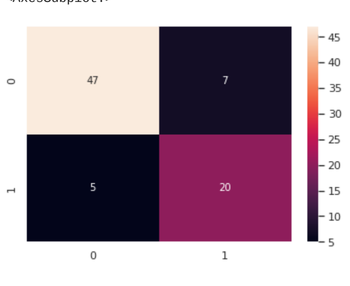
```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(x_train,y_train)
print("Accuracy of Naive Bayes Algorithm: ", nb.score(x_test,y_test)*100)
ac_naive=nb.score(x_test,y_test)*100
```

```
Accuracy of Naive Bayes Algorithm:  84.81012658227847
```

```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import precision_recall_fscore_support as score
# confusion Matrix of Logistic Regression
predictions = nb.predict(x_test)
precision_nb,recall_nb,fscore_nb,support_nb=score(y_test,predictions,average='macro')
print(classification_report(y_test,predictions))
sns.heatmap(confusion_matrix(y_test,predictions),annot=True, fmt="d")
```

```
              precision    recall  f1-score   support

           0       0.90      0.87      0.89        54
           1       0.74      0.80      0.77        25

    accuracy                           0.85        79
   macro avg       0.82      0.84      0.83        79
weighted avg       0.85      0.85      0.85        79
```

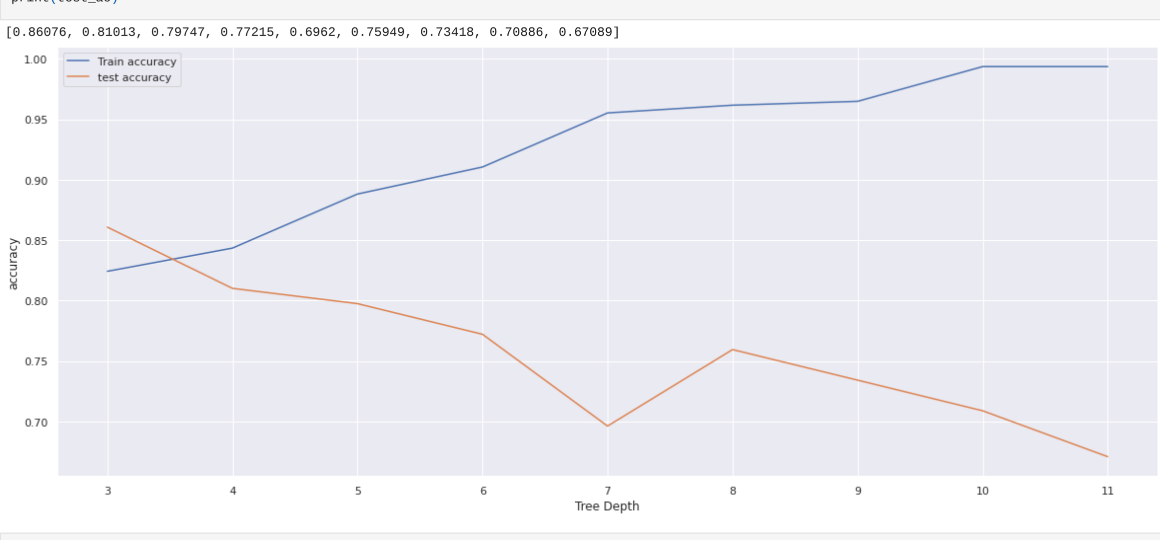
```
<AxesSubplot:>
```



Decision tree classifier

```
from sklearn.tree import DecisionTreeClassifier
def evaluation(p,y):
    matrix=np.zeros((2,2))
    for i in range(len(y)):
        matrix[ round(p[i]) ] [y[i]]+=1
    #True+, True-, False+, False-
    tp,tn,fp,fn=matrix[1,1], matrix[0,0], matrix[1,0], matrix[0,1]
    accuracy=round( (tp+tn)/ (tp+tn+fn+fp),5)
    precision= round(tp/ (tp+fp),5)
    recall= round(tp/(tp+fn),5)
    f_score= round(2*precision*recall/ (precision+recall),5)
    return accuracy,tp,tn,fp,fn, precision, recall, f_score
depth_list=np.arange(3,12)
log = np.zeros((2,len(depth_list)))


def Validation_graph(log, list, x_label):
    plt.figure(figsize=(20,8))
    plt.ylabel("accuracy")
    plt.xlabel(x_label)
    sns.lineplot(y = log[0], x=list, label="Train accuracy")
    sns.lineplot(y = log[1], x=list, label="test accuracy")

test_ac=[]
for i,depth in enumerate(depth_list):
    model_tree= DecisionTreeClassifier(max_depth=depth)
    model_tree.fit(x_train, y_train)

    predict_train = model_tree.predict(x_train)
    predict = model_tree.predict(x_test)
    result_train = evaluation(predict_train, y_train)
    result = evaluation(predict, y_test)
    test_ac.append(result[0])
    log[0][i], log[1][i] = result_train[0], result[0]

#change in accuracy
Validation_graph(log, depth_list,"Tree Depth")
print(test_ac)
```

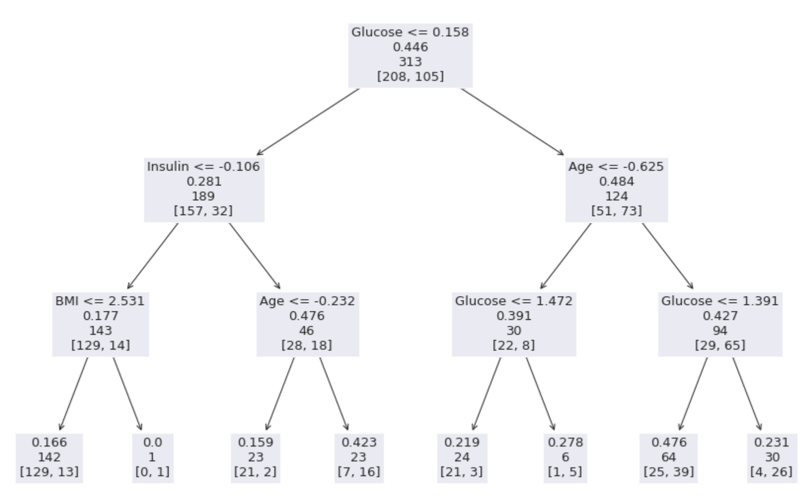
```
[0.86076, 0.81013, 0.79747, 0.77215, 0.6962, 0.75949, 0.73418, 0.70886, 0.67089]
```



Classification report and plotting decision tree

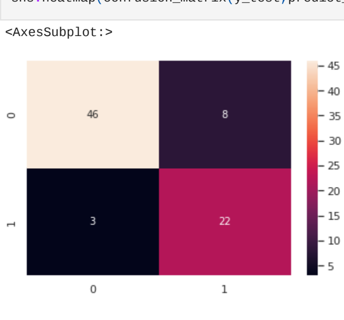
```
#we pick the parameter that perform the best from the above output to predict the test value
def result_output(result):
    print("accuracy:",result[0],"\nprecison:",result[5],"\nrecall:",result[6],"\nf-score:",result[7])

model_tree= DecisionTreeClassifier(max_depth=3)
res=model_tree.fit(x_train, y_train)
model_tree.fit(x_train, y_train)
predict_test = model_tree.predict(x_test)
result = evaluation(predict_test, y_test)
result_output(result)
ac_tree=result[0]*100
#Visualized_matrix(result)
from sklearn.tree import plot_tree
plt.figure(figsize=(15,10))
plot_tree(model_tree,feature_names=data.columns, label='none',fontsize=13)
print(ac_tree)
#print(classification_report(y_test,x_test))
precision_t,recall_t,fscore_t,support_t=score(y_test,predict_test,average='macro')
```

```
accuracy: 0.86076
precison: 0.73333
recall: 0.88
f-score: 0.8
86.076
```




```
sns.heatmap(confusion_matrix(y_test,predict_test),annot=True, fmt="d")
```

```
<AxesSubplot:>
```

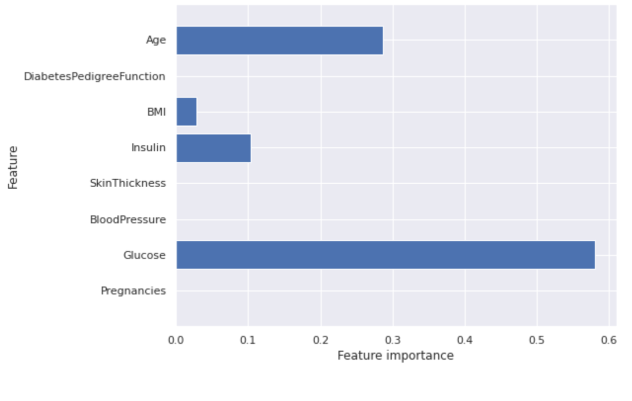


Feature importance of decision tree

```
diabetes_features = [x for i,x in enumerate(X1.columns) if i !=8]
print("Decision Tree Feature importances:\n{}".format(model_tree.feature_importances_))
```

```
Decision Tree Feature importances:
[0.         0.58157956 0.         0.         0.10291882 0.02815081
 0.         0.28735081]
```

```
def plot_feature_importances_diabetes(model):
    plt.figure(figsize=(8,6))
    n_features = 8
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), diabetes_features)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)

plot_feature_importances_diabetes(model_tree)
```



Logistic regression

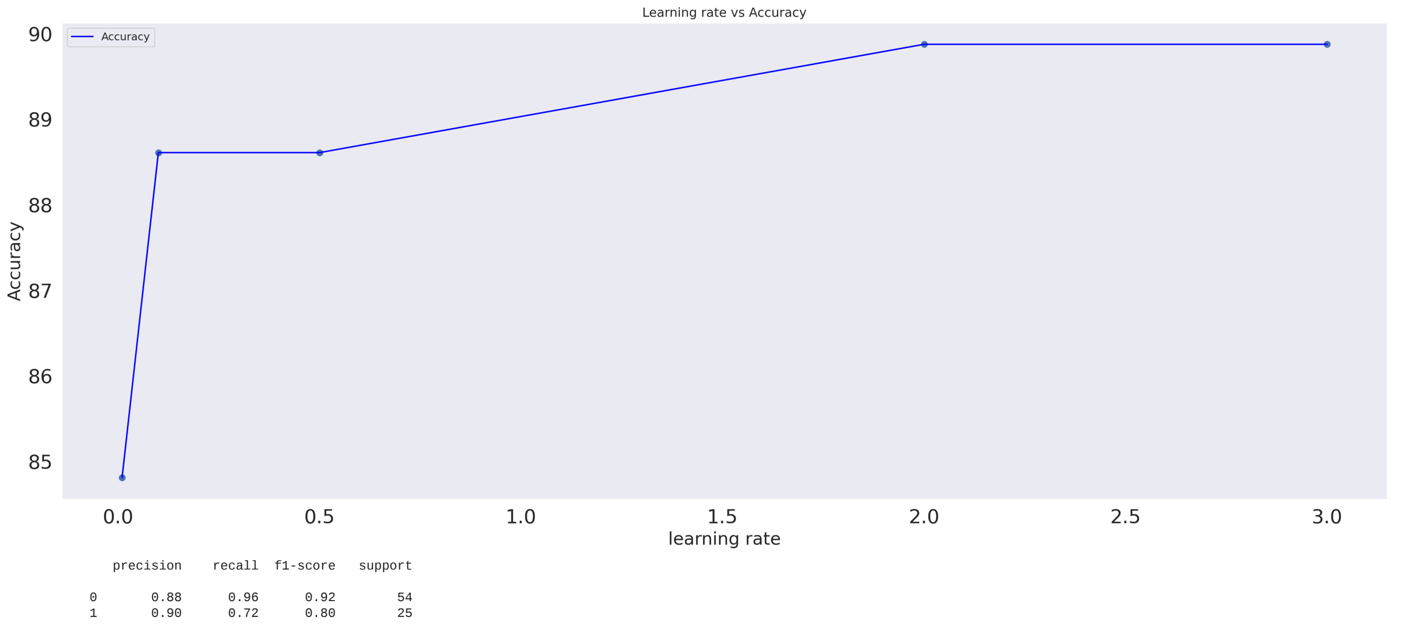
Hypertune logistic regression for different learning rate

```
from sklearn.linear_model import LogisticRegression
learning_rate=[0.01,0.1,0.5,2,3]
ac1=[]
for i in learning_rate :
    lr = LogisticRegression(C=i)
    lr.fit(x_train, y_train.ravel())
    acc=(lr.score(x_test,y_test)) * 100
    ac1.append(acc)
    predictions = lr.predict(x_test)
    print("for learning rate = ",i,"test accuracy: {} %".format((lr.score(x_test,y_test)) * 100))
```

```
for learning rate =  0.01 test accuracy: 84.81012658227847 %
for learning rate =  0.1 test accuracy: 88.60759493670885 %
for learning rate =  0.5 test accuracy: 88.60759493670885 %
for learning rate =  2 test accuracy: 89.87341772151899 %
for learning rate =  3 test accuracy: 89.87341772151899 %
```

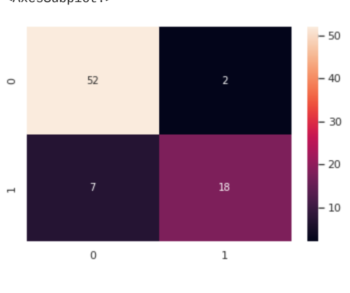
Plotting the accuracy

```
ac_log=max(ac1)
fig = plt.figure(1, figsize=(20,8), dpi=200)
plt.grid()
#plt.subplots(1, figsize=(20,8))
plt.plot(learning_rate, ac1, label="Accuracy", color="blue")
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.scatter(learning_rate, ac1)

plt.xlabel("learning rate",fontsize=18)
plt.ylabel("Accuracy",fontsize=18)
plt.title("Learning rate vs Accuracy",fontsize=13)
plt.tight_layout()
plt.legend(loc="best")
plt.show()
lr = LogisticRegression(C=0.5,max_iter=500)
lr.fit(x_train, y_train)
predictions = lr.predict(x_test)
print(classification_report(y_test,predictions))
precision_log,recall_log,fscore_log,support_log=score(y_test,predictions,average='macro')
sns.heatmap(confusion_matrix(y_test,predictions),annot=True, fmt="d")
```




```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92        54
           1       0.90      0.72      0.80        25

    accuracy                           0.89        79
   macro avg       0.89      0.84      0.86        79
weighted avg       0.89      0.89      0.88        79
```

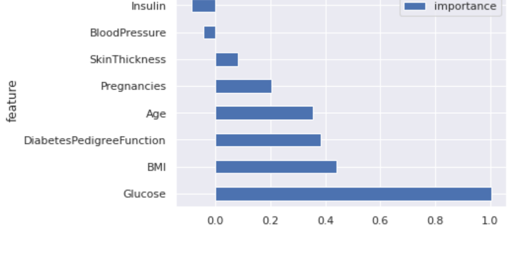
```
<AxesSubplot:>
```



Feature importance for logistic regression

```
lr = LogisticRegression(C=2)
lr.fit(x_train, y_train.ravel())
acc=(lr.score(x_test,y_test)) * 100
predictions = lr.predict(x_test)

feature_names=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction","Age"]
feature_importance = pd.DataFrame(feature_names, columns = ["feature"])

importance=lr.coef_
importance=importance.flatten()
#importance=np.sort(importance).flatten()
feature_importance["importance"] =importance
feature_importance = feature_importance.sort_values(by = ["importance"], ascending=False)
print(importance)
ax = feature_importance.plot.barh(x='feature', y='importance')
plt.show()
```

```
[ 0.20347813  1.00467034 -0.04539638  0.08259775 -0.08866897  0.44111204
  0.38380221  0.35533575]
```

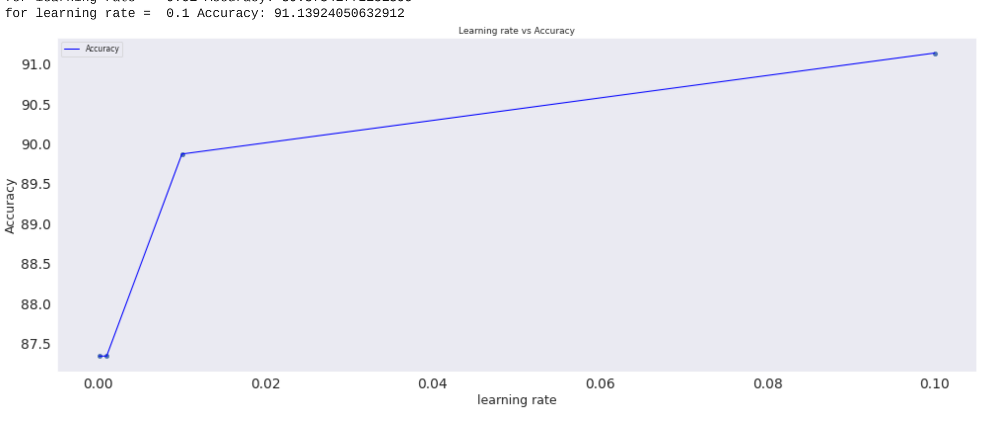


Adaboost Classifier

hypertune Adaboost for different learning rate

```
learning_rate1=[0.0001,0.001,0.01,.1]
ac2=[]
for i in learning_rate1 :
    ada = AdaBoostClassifier(learning_rate=i, n_estimators=30)
    ada.fit(x_train, y_train)
    pred = ada.predict(x_test)
    ac_ada=ada.score(x_test,y_test) * 100
    ac2.append(ac_ada)
    print("for learning rate = ",i,"Accuracy:", ac_ada)

fig = plt.figure(1, figsize=(20,8), dpi=50)
plt.grid()
#plt.subplots(1, figsize=(7,7))
plt.plot(learning_rate1, ac2, label="Accuracy", color="blue")
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.scatter(learning_rate1, ac2)

plt.xlabel("learning rate",fontsize=18)
plt.ylabel("Accuracy",fontsize=18)
plt.title("Learning rate vs Accuracy",fontsize=13)
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

```
for learning rate =  0.0001 Accuracy: 87.34177215189874
for learning rate =  0.001 Accuracy: 87.34177215189874
for learning rate =  0.01 Accuracy: 89.87341772151899
for learning rate =  0.1 Accuracy: 91.13924050632912
```

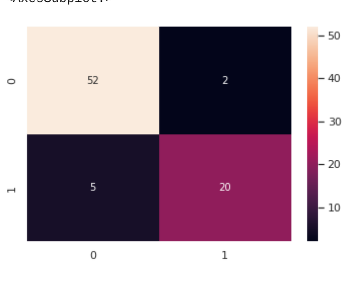


classification report

```
ada = AdaBoostClassifier(learning_rate=0.1, n_estimators=30)
ada.fit(x_train, y_train)
pred = ada.predict(x_test)
ac_ada=ada.score(x_test,y_test) * 100
print("for learning rate = ",i,"Accuracy:", ac_ada)
print(classification_report(y_test,pred))
precision_ad,recall_ad,fscore_ad,support_ad=score(y_test,pred,average='macro')
sns.heatmap(confusion_matrix(y_test,pred),annot=True, fmt="d")
```

```
for learning rate =  0.1 Accuracy: 91.13924050632912
              precision    recall  f1-score   support

           0       0.91      0.96      0.94        54
           1       0.91      0.80      0.85        25

    accuracy                           0.91        79
   macro avg       0.91      0.88      0.89        79
weighted avg       0.91      0.91      0.91        79
```

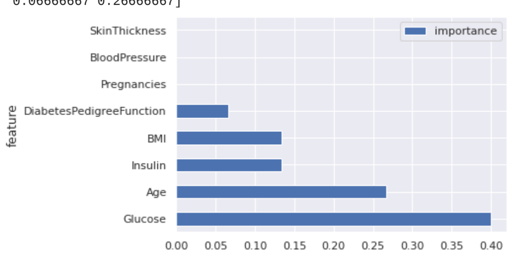
```
<AxesSubplot:>
```



feature importance of Adaboost classifier

```
feature_names=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction","Age"]
feature_importance = pd.DataFrame(feature_names, columns = ["feature"])

importance=ada.feature_importances_
importance=importance.flatten()
#importance=np.sort(importance).flatten()
feature_importance["importance"] =importance
feature_importance = feature_importance.sort_values(by = ["importance"], ascending=False)
print(importance)
ax = feature_importance.plot.barh(x='feature', y='importance')
plt.show()
```

```
[0.         0.4        0.         0.         0.13333333 0.13333333
 0.06666667 0.26666667]
```



Comparing Accuracy of all models

```
all_accuracy = [['Logistic Regression',ac_log,precision_log,recall_log,fscore_log], ['Naive Bayes',ac_naive,precision_nb,recall_nb,fscore_nb], ['Decision tree',ac_tree,precision_t,

accuracy_summary = pd.DataFrame(data = all_accuracy, columns=['Model Name','Test Accuracy','Precision','Recall','f1-score'])
accuracy_summary
```

	Model Name	Test Accuracy	Precision	Recall	f1-score
0	Logistic Regression	89.873418	0.890678	0.841481	0.860177
1	Naive Bayes	84.810127	0.822293	0.835185	0.828012
2	Decision tree	86.076000	0.836054	0.865926	0.846602
3	Adaboost	91.139241	0.910686	0.881481	0.894000

Plotting the accuracy

```
acc=np.array([ac_log, ac_naive, ac_tree,ac_ada])
x=np.array(['Logistic Regression','Naive bayes','Decision Tree Classifier','Adaboost'])
plt.scatter(x,acc,s=200,color=['purple','green','red','blue'])
plt.xticks(rotation=90)
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```