CoxAssignment07

Task: This assignment will help you understand and to build a machine learning model using the k-Nearest Neighbors algorithm to predict whether the patients in the "Pima Indians Diabetes Dataset" have diabetes or not: https://www.kaggle.com/amolbhivarkar/knn-for-classification-using-scikit-learn

Dataset: Diabetes.csv

The idea for this assignment is not simply to copy and paste some code. There are interesting differences I want you to note in this code compared to what we have done. There are two major learning points here.

1. Scikit-Learn is a FANTASTIC resource and contains many of the things required to complete predictive or classification problems.
2. Kaggle can be a great resource for learning and understanding the models. k-NN is one of the most popular classification models.

To complete this assignment, type in all the code and discuss what each section of code is doing. As opposed to you thinking up the code to use, this assignment will challenge you to discuss the importance of each line and note some of the differences between the code they have written and the code we have written. It also gets you thinking about the evaluation metrics in the context of a project and the importance of them.

In [1]:
```
#Load the necessary python libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

In [3]:
```
#Load the dataset
df = pd.read_csv('diabetes.csv')

#Print the first 5 rows of the dataframe.
df.head()
```

Out[3]:

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

In [4]:
```
df.shape
```

Out[4]: (768, 9)

In [5]:
```
X = df.drop('Outcome',axis=1).values
y = df['Outcome'].values
```

In [6]:
```
from sklearn.model_selection import train_test_split
```

In [7]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4,random_state=42, st
```

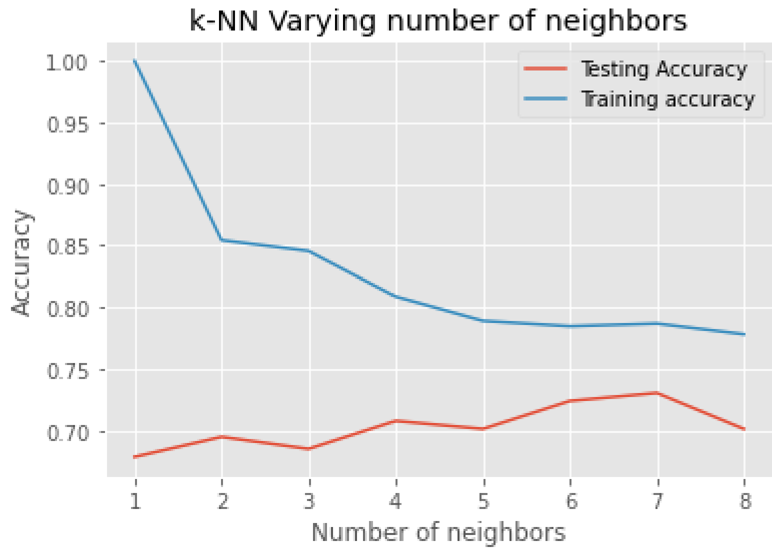
```
- Up until this point we have been doing a similar starting process
to set up the test data from our file import
```

In [29]:
```
from sklearn.neighbors import KNeighborsClassifier

neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):

    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train, y_train)

    train_accuracy[i] = knn.score(X_train, y_train)

    test_accuracy[i] = knn.score(X_test, y_test)
```

```
- This utilizes sklearn to set the KNeighborsClassifier based on 'k'
which loops through and finds how many there are from the for loop.
We then use to fit the model and find the accuracy of the training
set and test set
```

In [9]:
```
#Generate plot
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
```

```
plt.ylabel('Accuracy')
plt.show()
```



- This shows how accurate the testing set and the training set are as the number of neighbors increases. The point where they are they closest is at 7 neighbors where we change knn below.

In [30]:
```
knn = KNeighborsClassifier(n_neighbors=7)
```

In [31]:
```
knn.fit(X_train,y_train)
```

Out[31]:
```
KNeighborsClassifier(n_neighbors=7)
```

-Unlike the above 'knn =' this specifies a specifc amount of neighbors to determine the results from 7 neighbors then fits the new knn to the model

In [12]:
```
knn.score(X_test,y_test)
```

Out[12]:
```
0.7305194805194806
```

In [13]:
```
#import confusion_matrix
from sklearn.metrics import confusion_matrix
```

In [14]:
```
y_pred = knn.predict(X_test)
```

In [15]:
```
confusion_matrix(y_test,y_pred)
```

Out[15]:
```
array([[165,  36],
       [ 47,  60]], dtype=int64)
```

In [16]:
```
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

Out[16]:

Predicted	0	1	All
True			
0	165	36	201
1	47	60	107
All	212	96	308

```
- The confusion matrix is used to help test our models performance
```

In [17]:
```
#import classification_report
from sklearn.metrics import classification_report
```

In [18]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.82      0.80       201
           1       0.62      0.56      0.59       107

    accuracy                           0.73       308
   macro avg       0.70      0.69      0.70       308
weighted avg       0.73      0.73      0.73       308
```

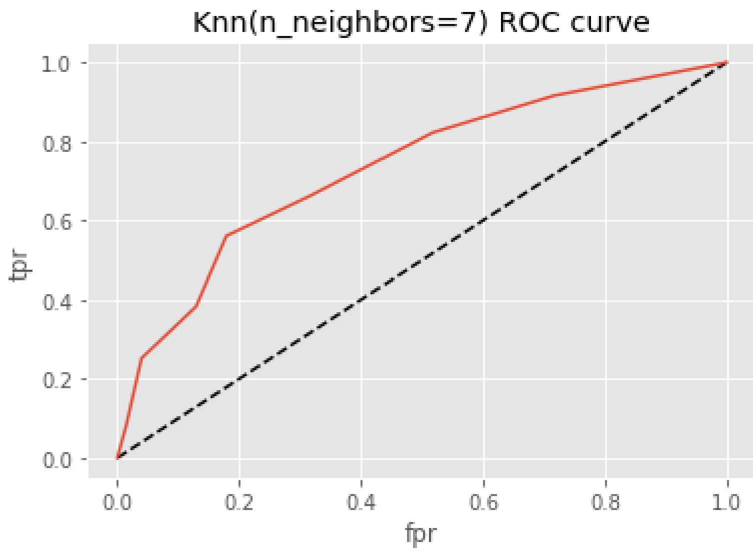
```
- The classification report is then used to determine how good are
our predictions are by showing how accurate they are
```

In [19]:
```
y_pred_proba = knn.predict_proba(X_test)[:,1]
```

In [20]:
```
from sklearn.metrics import roc_curve
```

In [21]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

In [22]:
```
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=7) ROC curve')
plt.show()
```



In [23]:
```
#Area under ROC curve
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

Out[23]: 0.7345050448691124

- This segement where it dives in the ROC Curve is more important then the accuracy from above becuase it balances between precision and recall from the classification report for a more relialbe outcome. The ROC AUC Score tells us how effecient our model is, which by having a .7345 shows that it is effecienct but could be a little better but for what we are trying to determine is high enough to show that it is reliable.

In [24]:
```
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
```

In [25]:
```
#In case of classifier like knn the parameter to be tuned is n_neighbors
param_grid = {'n_neighbors':np.arange(1,50)}
```

In [26]:
```
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X,y)
```

Out[26]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 1
0, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

In [27]:
```
knn_cv.best_score_
```

Out[27]: 0.7578558696205755

In [28]:
```
knn_cv.best_params_
```

Out[28]: {'n_neighbors': 14}

- The GridSearchCV is a function that helps up search through our hyperparameters and fit our model to our training set. The best_score_ is the mean score of the best estimator that our model has. With a best_score_ of .7578 shows that it is fairly good, as the best it could be is 1.0. The best_params_ shows that for the most effecient model we should use n_neighbors equal to 14. When we changed the n_neighbors to 7 above it was still fairly accurate but it did no include enough parameters to be as effecient as possible. When it comes to determining if patients have diabeties you want to be as accurate as possible. For other cases you may be able to get away with slightly less accuracy but most of the time you want to be as accurate as possible.

In [ ]:

```
```