02-K Nearest Neighbors Project

July 16, 2021

---

1 K Nearest Neighbors Project

Welcome to the KNN Project! This will be a simple project very similar to the lecture, except you'll be given another data set. Go ahead and just follow the directions below. ## Import Libraries **Import pandas,seaborn, and the usual libraries.**

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

1.1 Get the Data

** Read the 'KNN_Project_Data csv file into a dataframe **

[2]:
```
prdf = pd.read_csv('KNN_Project_Data')
```

Check the head of the dataframe.

[3]:
```
prdf.head()
```

[3]:
```
          XVPM         GWYH         TRAT        TLLZ         IGGA  \
0  1636.670614   817.988525  2565.995189  358.347163   550.417491
1  1013.402760   577.587332  2644.141273  280.428203  1161.873391
2  1300.035501   820.518697  2025.854469  525.562292   922.206261
3  1059.347542  1066.866418   612.000041  480.827789   419.467495
4  1018.340526  1313.679056   950.622661  724.742174   843.065903

          HYKR         EDFS        GUUB         MGJM         JHZC  \
0  1618.870897  2147.641254  330.727893  1494.878631   845.136088
1  2084.107872   853.404981  447.157619  1193.032521   861.081809
2  2552.355407   818.676686  845.491492  1968.367513  1647.186291
3   685.666983   852.867810  341.664784  1154.391368  1450.935357
```

```
4  1370.554164   905.469453  658.118202   539.459350  1899.850792

   TARGET CLASS
0             0
1             1
2             1
3             0
4             0
```

[ ]:

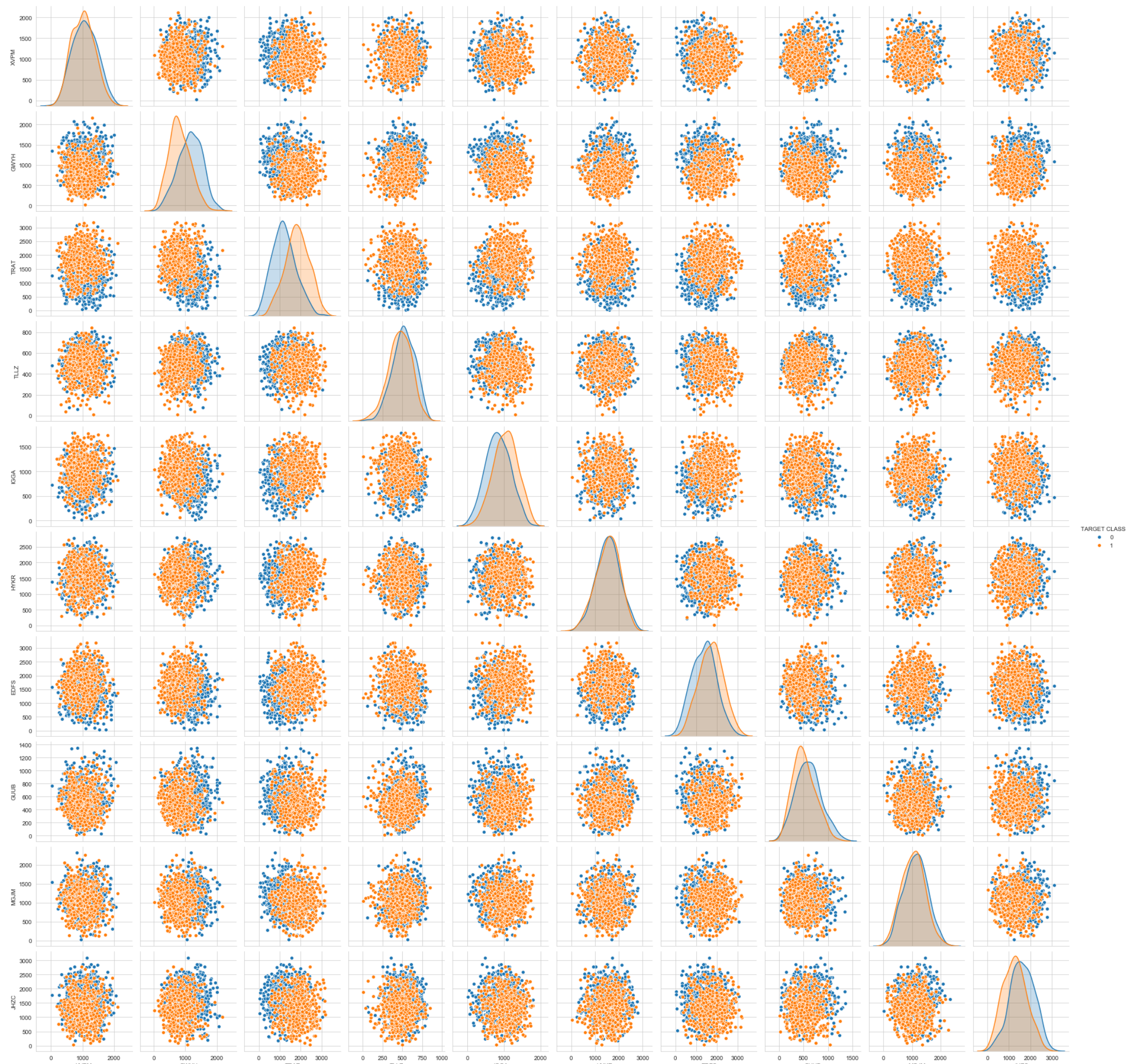
2 EDA

Since this data is artificial, we'll just do a large pairplot with seaborn.

Use seaborn on the dataframe to create a pairplot with the hue indicated by the TARGET CLASS column.

[5]:
```
sns.set_style('whitegrid')
sns.pairplot(prdf,hue='TARGET CLASS')
```

[5]: <seaborn.axisgrid.PairGrid at 0x2882e745248>



```
[ ]:
```

3 Standardize the Variables

Time to standardize the variables.

** Import StandardScaler from Scikit learn.**

```
[4]: from sklearn.neighbors import KNeighborsClassifier
     from sklearn.preprocessing import StandardScaler
```

** Create a StandardScaler() object called scaler.**

```
[5]: scaler = StandardScaler()
```

** Fit scaler to the features.**

[6]:
```python
scaler.fit(prdf.drop('TARGET CLASS',axis=1))
```

[6]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Use the .transform() method to transform the features to a scaled version.

[7]:
```python
scaler.transform(prdf.drop('TARGET CLASS',axis=1))
```

[7]:
```
array([[ 1.56852168, -0.44343461,  1.61980773, ..., -0.93279392,
         1.00831307, -1.06962723],
       [-0.11237594, -1.05657361,  1.7419175 , ..., -0.46186435,
         0.25832069, -1.04154625],
       [ 0.66064691, -0.43698145,  0.77579285, ...,  1.14929806,
         2.1847836 ,  0.34281129],
       ...,
       [-0.35889496, -0.97901454,  0.83771499, ..., -1.51472604,
        -0.27512225,  0.86428656],
       [ 0.27507999, -0.99239881,  0.0303711 , ..., -0.03623294,
         0.43668516, -0.21245586],
       [ 0.62589594,  0.79510909,  1.12180047, ..., -1.25156478,
        -0.60352946, -0.87985868]])
```

[8]:
```python
sc_feat = scaler.transform(prdf.drop('TARGET CLASS',axis=1))
```

Convert the scaled features to a dataframe and check the head of this dataframe to make sure the scaling worked.

[9]:
```python
sc_prdf = pd.DataFrame(sc_feat,columns=prdf.columns[:-1])
sc_prdf
```

[9]:
```
         XVPM      GWYH      TRAT      TLLZ      IGGA      HYKR      EDFS  \
0    1.568522 -0.443435  1.619808 -0.958255 -1.128481  0.138336  0.980493
1   -0.112376 -1.056574  1.741918 -1.504220  0.640009  1.081552 -1.182663
2    0.660647 -0.436981  0.775793  0.213394 -0.053171  2.030872 -1.240707
3    0.011533  0.191324 -1.433473 -0.100053 -1.507223 -1.753632 -1.183561
4   -0.099059  0.820815 -0.904346  1.609015 -0.282065 -0.365099 -1.095644
..        ...       ...       ...       ...       ...       ...       ...
995  0.776682  0.758234 -1.753322  0.507699  0.174588 -1.279354 -1.797957
996 -0.313446  0.385206  0.885502 -0.083136 -1.208486  0.309242  0.746346
997 -0.358895 -0.979015  0.837715  0.014018 -1.397424  0.054473  0.164120
998  0.275080 -0.992399  0.030371  1.062954  1.142871 -0.192872  2.051386
999  0.625896  0.795109  1.121800  1.185944  0.555582 -1.133032  0.746559

         GUUB      MGJM      JHZC
0   -0.932794  1.008313 -1.069627
1   -0.461864  0.258321 -1.041546
```

```
2    1.149298  2.184784  0.342811
3   -0.888557  0.162310 -0.002793
4    0.391419 -1.365603  0.787762
..        ...       ...       ...
995  0.431419  0.088717  1.188886
996 -0.112571 -1.763636 -1.559081
997 -1.514726 -0.275122  0.864287
998 -0.036233  0.436685 -0.212456
999 -1.251565 -0.603529 -0.879859

[1000 rows x 10 columns]
```

[ ]:

4 Train Test Split

Use train_test_split to split your data into a training set and a testing set.

[10]:
```
X = sc_prdf
y = prdf['TARGET CLASS']
```

[11]:
```
from sklearn.model_selection import train_test_split
```

[12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=101)
```

5 Using KNN

Import KNeighborsClassifier from scikit learn.

[13]:
```
from sklearn.neighbors import KNeighborsClassifier
```

Create a KNN model instance with n_neighbors=1

[14]:
```
KNN = KNeighborsClassifier(n_neighbors=1)
```

Fit this KNN model to the training data.

[15]:
```
KNN.fit(X_train,y_train)
```

[15]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

[ ]:

6 Predictions and Evaluations

Let's evaluate our KNN model!

Use the predict method to predict values using your KNN model and X_test.

[16]:
```
predictions = KNN.predict(X_test)
```

** Create a confusion matrix and classification report.**

[17]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[18]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[109  43]
 [ 41 107]]
```

[19]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.73      0.72      0.72       152
           1       0.71      0.72      0.72       148

    accuracy                           0.72       300
   macro avg       0.72      0.72      0.72       300
weighted avg       0.72      0.72      0.72       300
```

[ ]:
```

```

7 Choosing a K Value

Let's go ahead and use the elbow method to pick a good K Value!

** Create a for loop that trains various KNN models with different k values, then keep track of the error_rate for each of these models with a list. Refer to the lecture if you are confused on this step.**

[20]:
```
error_rate=[]

for i in range(1,41):
    KNN = KNeighborsClassifier(n_neighbors=i)
    KNN.fit(X_train,y_train)
    predictions_i = KNN.predict(X_test)
    error_rate.append(np.mean(predictions_i != y_test))
```

Now create the following plot using the information from your for loop.

[21]:
```
plt.figure(figsize=(10,6))
plt.
 ↪plot(range(1,41),error_rate,color='blue',ls='--',marker='o',markersize=10,markerfacecolor='re
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.title('Error Rate vs. K Value')
```

[21]:
```
Text(0.5, 1.0, 'Error Rate vs. K Value')
```

[ ]:
```
```

7.1 Retrain with new K Value

Retrain your model with the best K value (up to you to decide what you want) and re-do the classification report and the confusion matrix.

[22]:
```
#n = 31 from above plot
KNN = KNeighborsClassifier(n_neighbors=31)
KNN.fit(X_train,y_train)
predictions = KNN.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[123  29]
 [ 19 129]]
```

```
              precision    recall  f1-score   support

           0       0.87      0.81      0.84       152
           1       0.82      0.87      0.84       148

    accuracy                           0.84       300
   macro avg       0.84      0.84      0.84       300
weighted avg       0.84      0.84      0.84       300
```

[ ]:

```

```

8 Great Job!