Wine Classification using a K-Nearest Neighbors Algorithm

CSCI 31022 - Machine Learning and Pattern Recognition

Assignment 1

CS/2018/009 – G.W.I. DILSHANI

Abstract

This classification intends to employ modern and effective techniques such as KNN, which groups together datasets and provides a comprehensive and generic approach for recommending wine to customers based on specific features. This will allow the shop owners to anticipate the demand for the wine and adjust their stock accordingly. This will aid shop owners in increasing their profits.

Table of Content

Contents

1. Introduction 3

1.1 K-Nearest Neighbors (KNN) algorithm 3

1.2 A dataset using for wine classification 3

2. Importing the data set and packages to be used 4

3. Normalizing the data 5

3.1 Prepare the dataset 5

3.2 Train the classifiers 5

4. Define the Model 6

4.1Tuning Sensitivity of Model To n_neighbors 6

5. Make Predictions 8

6. Conclusion 10

7. References 10

1. Introduction

For this project, we wanted to practice implementing a K-Nearest Neighbors (KNN) algorithm to classify wines in scikit-learn's wine recognition data set. The basic idea behind the KNN algorithm is that things with similar classifications tend to have similar properties, and so if one looks at the k "nearest neighbors" (in terms of some distance calculation) of a data point one can classify it based on the classifications of its neighbors. Since the wine recognition data set consists of chemical analysis data, the idea of a "distance" between data points is fairly well defined, so this seems an ideal data set to practice implementing a KNN algorithm on. The main idea is to classify wines based on the results of chemical analysis done.

We divide this algorithm in 4 parts.

- Import packages and Dataset
- Normalizing the data
- Define the model
- Make Predictions

1.1 K-Nearest Neighbors (KNN) algorithm

k-NN is a non-parametric classification method. The output of this classification is a class membership. A majority vote of its neighbors classifies an object, with the object assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, the object is simply assigned to the class of the object's single nearest neighbor. Among all machine learning algorithms, it is the most basic.

1.2 A dataset using for wine classification

There are 13 features in this data set and 1 target value i.e. 14 columns and 177 samples. The columns are ‘name’,’alcohol’,’malicAcid’,’ash’,’ashalcalinity’,’magnesium’,’totalPhenols’,’flavanoids’,’nonFlavanoidPhenols’,’proanthocyanins’,’colorIntensity’,’hue’,’od280_od315',’proline’

Here the target label is name. It has 3 classes in it ‘1’, ‘2’ and ‘3’. The wine falls under either of these three classes.

2. Importing the data set and packages to be used

- First, we import the dataset.

```
from sklearn import datasets
```

- Next, import packages which will help me analyze and visualize the data.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- Finally, the packages which will let build a KNN model.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

The whole code which helps to import packages as follows.

```
In [124]: from sklearn import datasets

          import numpy as np
          import matplotlib.pyplot as plt
          import pandas as pd

          from sklearn.model_selection import train_test_split
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import confusion_matrix
          import seaborn as sns

          %matplotlib notebook
```

- Now need to actually load the imported data set.

```
In [98]: wine=datasets.load_wine()
```

- Print(wine_data.csv)

printed out the file description to see what the various columns correspond to, and what the "targets" (i.e., classifications) are.

```
In [99]: df=pd.DataFrame(wine["data"], columns=wine["feature_names"])
         df["target"]=wine["target"]
         df.head()
```

Since loaded the wine recognition data as a DataFrame, don't need to convert all_features and targets to DataFrames; they already are DataFrames/Series. Infer that, in the target Series, 0 is class_0, 1 is class_1, and 2 is class_2. Interestingly, the data seems to be provided in numerical order by class

The output as follows.

Out[99]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_dilu
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	

3. Normalizing the data

3.1 Prepare the dataset

Cleaning of the dataset includes dropping the unnecessary columns and the NaN values with the help of the code mentioned below:

```
In [100]: df.shape
Out[100]: (178, 14)

In [101]: df.isna().sum()
Out[101]: alcohol                        0
          malic_acid                     0
          ash                            0
          alcalinity_of_ash              0
          magnesium                      0
          total_phenols                  0
          flavanoids                     0
          nonflavanoid_phenols           0
          proanthocyanins                0
          color_intensity                0
          hue                            0
          od280/od315_of_diluted_wines   0
          proline                        0
          target                         0
          dtype: int64

In [102]: x=df
          y=x.pop("target")
```

The output as follows:

```
In [103]: x.head()
```

Out[103]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_dilu
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	

```
In [104]: x.shape
Out[104]: (178, 13)

In [105]: y.unique()
Out[105]: array([0, 1, 2])
```

3.2 Train the classifiers

Divide the features and targets into training and test sets. Also, choose a random state so that we can keep the same feature row/target row split each time we run the cells in this Jupiter Notebook..

```
In [106]: #Train Classifier

In [107]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 55)

In [108]: x_train.shape
Out[108]: (142, 13)

In [109]: x_test.shape
Out[109]: (36, 13)
```

4. Define the Model

We now create a KNN model for the data. We don't know how many nearest neighbors to look at to best classify data, so that will be a free parameter in the model. The criterion for selecting k (the number of nearest neighbors used by the algorithm for classification) is maximization of model accuracy when applied to the validation set. In other words, the k choice will be the one that yields the greatest accuracy.

```
In [ ]: #Define the KNN model

In [17]: knn= KNeighborsClassifier(n_neighbors=3)

In [18]: knn.fit(x_train, y_train)

Out[18]: KNeighborsClassifier(n_neighbors=3)

In [19]: knn.score(x_test, y_test)

Out[19]: 0.8333333333333334
```

4.1Tuning Sensitivity of Model To n_neighbors

```
In [55]: k_range = range(1,25)
         scores=[]

         for k in k_range:
             knn= KNeighborsClassifier(n_neighbors=k)
             knn.fit(x_train, y_train)
             scores.append(knn.score(x_test, y_test))

         plt.figure()
         plt.xlabel("K count")
         plt.ylabel("Model Accuracy")
         plt.scatter(k_range, scores)
         plt.grid()
         plt.xticks([0, 5, 10, 15, 20, 30])
         plt.show()
```

- An empty array to hold accuracy scores

```
scores=[]
```

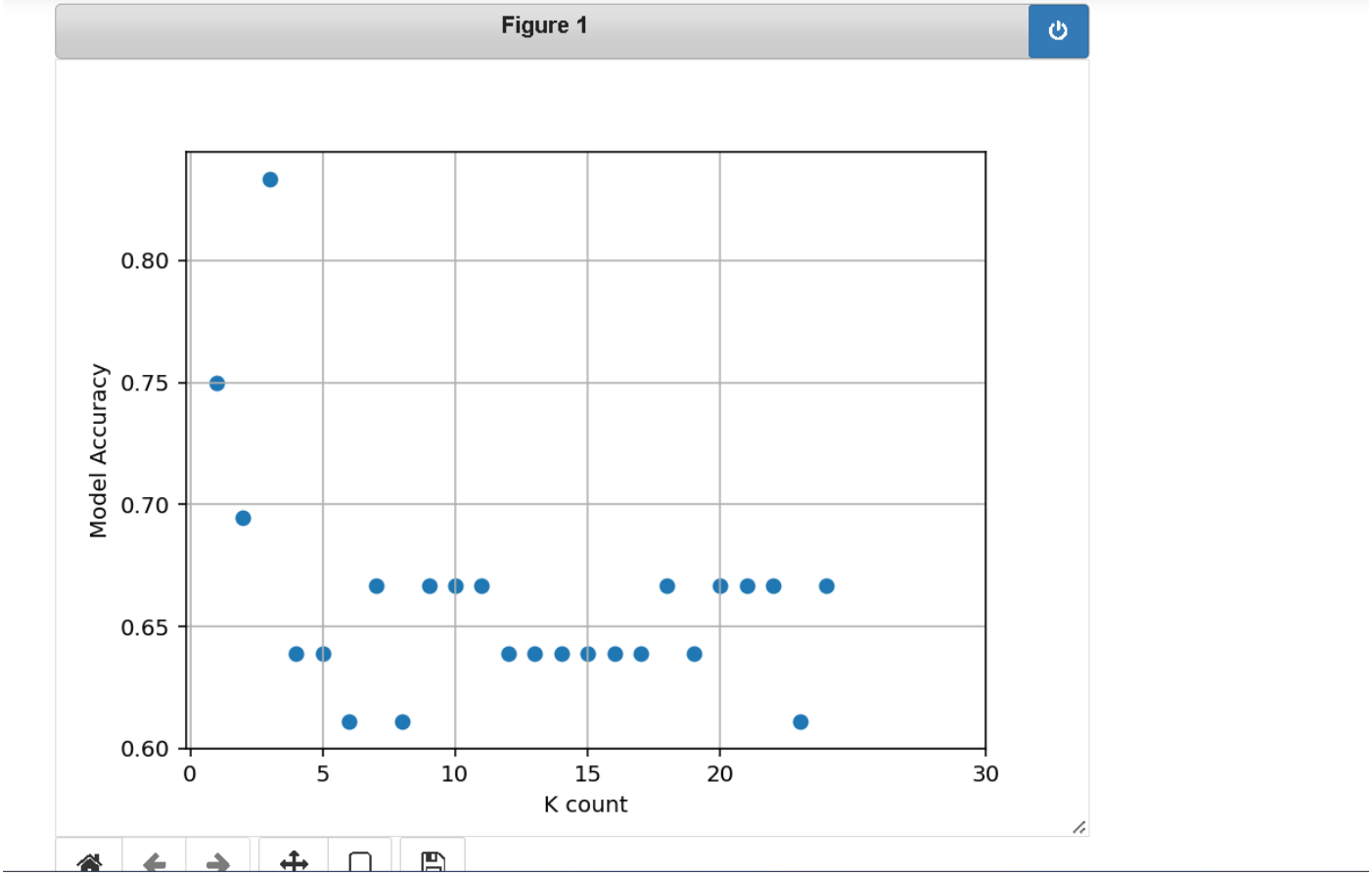
- Loop over a range of k and store the resulting model accuracy score

```
for k in k_range:
    knn= KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    scores.append(knn.score(x_test, y_test))
```

- Show plot chart by using following codes

```
plt.figure()
plt.xlabel("K count")
plt.ylabel("Model Accuracy")
plt.scatter(k_range, scores)
plt.grid()
plt.xticks([0, 5, 10, 15, 20, 30])
plt.show()
```

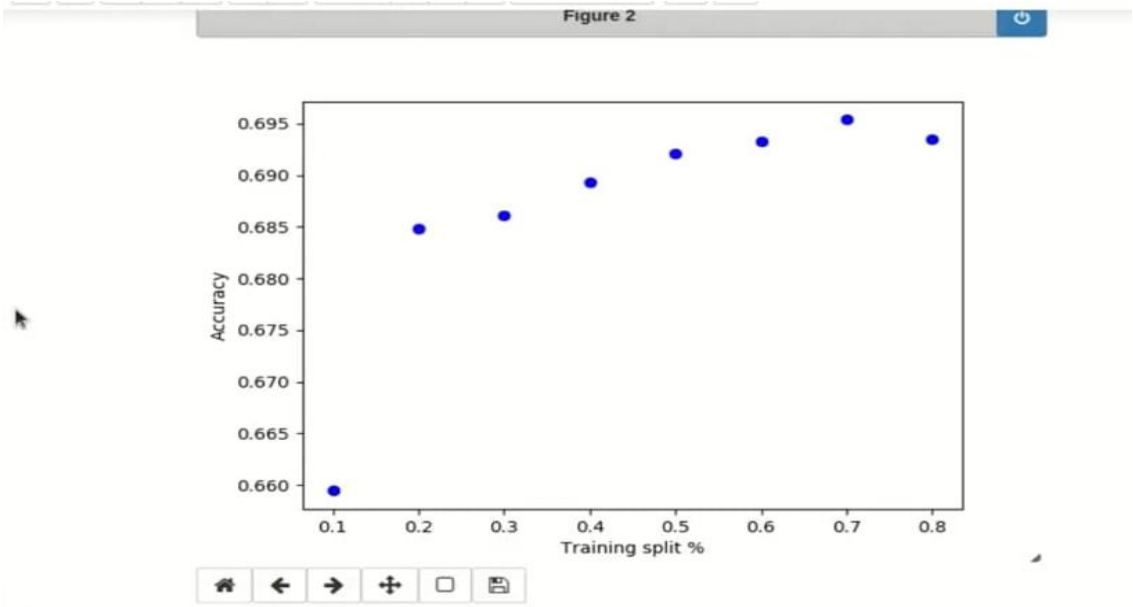
- The output is



Now that we've identified the model with the highest accuracy (when applied to the validation set), we'll use it as "the" model to classify the wines in the test set (the portion of the data that was separated from the rest of the data when train _test_split() was first called). And, because we'll know the true classification of each of those wines, we'll be able to see how well the model performed and where it went wrong.

In [134]:
```
test_sizes= [0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]
knn= KNeighborsClassifier(n_neighbors=5)

plt.figure()

for test_size in test_sizes:
    score=[]

    for i in range(1,1000):
        x_train, x_test, y_train, y_test=train_test_split(x, y, test_size = 1-test_size)
        knn.fit(x_train, y_train)
        scores.append(knn.score(x_test, y_test))

    plt.plot(test_size, np.mean(scores))

plt.xlabel("Training split   %")
plt.ylabel("Accuracy")


plt.xticks([0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1])
plt.show()
```

Output:



At least in general, the accuracy score follows the trend that expect: it increases with increasing k, reaches a maximum, then decreases as k continues to increase.

5. Make Predictions

Following codes are used to make predictions about the classification.

```
In [31]: prediction = knn.predict(x_test)
```

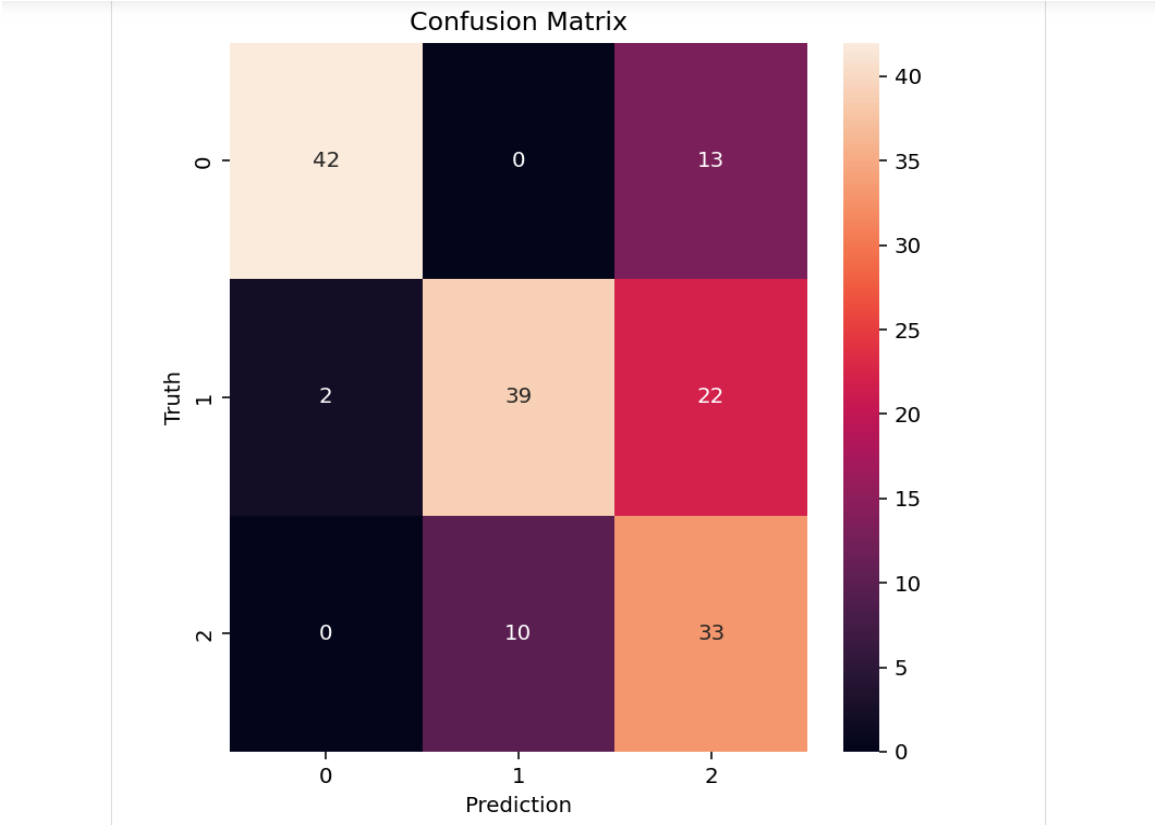
```
In [120]: prediction
Out[120]: array([2, 1, 1, 2, 1, 0, 1, 0, 0, 2, 2, 2, 1, 1, 1, 2, 1, 1, 2, 0, 0, 0,
                 2, 1, 1, 2, 1, 0, 1, 1, 1, 2, 0, 2, 1, 1])
```

```
In [121]: cm=confusion_matrix(y_test, prediction)
```

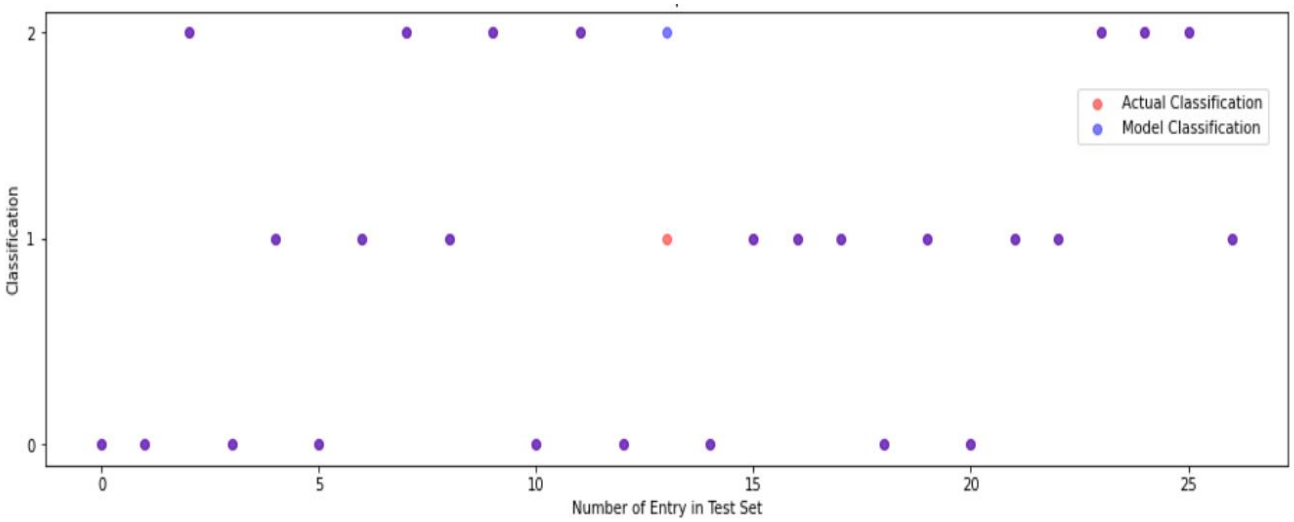
```
In [122]: cm
Out[122]: array([[ 7,  0,  1],
                 [ 0, 12,  5],
                 [ 1,  5,  5]], dtype=int64)
```

We can create confusion Matrix for this classification.

```
In [123]: plt.figure(figsize=(8,7))
          sns.heatmap(cm, annot=True)
          plt.title("Confusion Matrix")
          plt.ylabel("Truth")
          plt.xlabel("Prediction")
```



The following chart shows Actual Wine Classification Compared to Model Classification.



What like about this visualization is that it allows to see precisely which wine samples were misclassified, what the true classification was, and what classification the model assigned to it. It's another great visual way to look at the model's performance.

6. Conclusion

Based on the wine recognition toy data set in the scikit-learn datasets package, this Assignment implemented a K-Nearest Neighbors classification algorithm to classify wines based on their chemical analysis measurements. This model performed admirably, misclassifying only one wine sample out of 27. We plotted the model accuracy score versus k to ensure that the accuracy trend with k was roughly what we expected; then we created two graphs to help me visualize the model's performance. I conclude that a KNN algorithm performs admirably in predicting wine classification..

7. References

- **https://github.com/macaler/wine_knn_classification/blob/main/MAC_wine_knn_classification.ipynb**
- **https://www.askpython.com/python/wine-classification**
- **https://www.youtube.com/watch?v=x74ZnzeLpsA**
- **https://scikit-learn.org/stable/supervised_learning.html#supervised-learning**