MuhammadHassanShah_20P-0025_C_Lab03

February 18, 2023

1 Supervised Learning : K- Nearest Neighbor (KNN)

1.1 Imports

[1]:
```
# Data manipulation imports
import numpy as np
import pandas as pd

# Visualization imports
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Modeling imports
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay,␣
↪confusion_matrix, classification_report
```

1.2 Generating Synthetic Data for a Binary Classification Problem

[2]:
```
np.random.seed(0)

df = pd.DataFrame({'X1': np.random.randint(1, 10, size=50),
                   'X2': np.random.randint(3, 10, size=50),
                   'Y': np.random.choice(['Bad', 'Good'], size=50)})
```

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

[3]:

	X1	X2	Y
0	6	3	Bad
1	1	7	Bad
2	4	4	Good
3	4	7	Bad
4	8	4	Bad

[4]:
```
np.random.seed(0)

good_data = np.random.randint(low=3, high=7, size=(25, 2)) * np.random.
 ↪normal(loc=1, scale=0.5, size=(25, 2))
bad_data = np.random.randint(low=0, high=3, size=(25, 2)) * np.random.
 ↪normal(loc=1, scale=0.5, size=(25, 2))

df = pd.DataFrame({'X1': np.concatenate([good_data[:, 0], bad_data[:, 0]]),
                   'X2': np.concatenate([good_data[:, 1], bad_data[:, 1]]),
                   'Y': np.concatenate([['Good'] * 25, ['Bad'] * 25])})
```

[5]:
```
df.head()
```

[5]:

	X1	X2	Y
0	1.558868	7.130781	Good
1	4.066878	4.020851	Good
2	1.309510	4.299907	Good
3	5.273551	10.543174	Good
4	3.333885	6.142094	Good

[6]:
```
df = df.sample(frac=1).reset_index(drop=True)
```

[7]:
```
df.head()
```

[7]:

	X1	X2	Y
0	5.077906	2.450214	Good
1	1.220203	0.000000	Bad
2	3.333885	6.142094	Good
3	0.000000	0.000000	Bad
4	1.070908	0.000000	Bad

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      50 non-null     float64
 1   X2      50 non-null     float64
 2   Y       50 non-null     object
dtypes: float64(2), object(1)
memory usage: 1.3+ KB
```

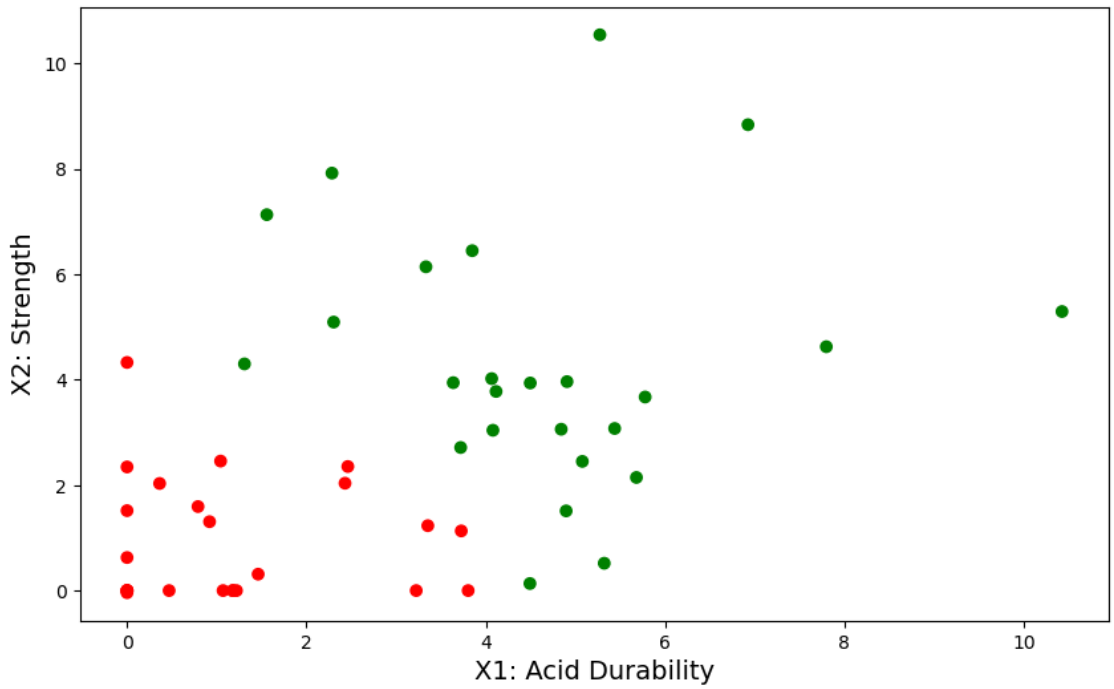
[9]:
```
df.describe()
```

[9]:

	X1	X2
count	50.000000	50.000000
mean	2.886601	2.630512
std	2.391710	2.568298
min	-0.000000	-0.040381
25%	0.951008	0.177889
50%	2.843731	2.245419
75%	4.496188	3.958765
max	10.427969	10.543174

1.3 Visualizing the Data

[10]:
```
plt.figure(figsize=(10,6))
plt.scatter(df['X1'], df['X2'], c=df['Y'].apply(lambda x: 'red' if x == 'Bad'␣
 ↪else 'green'), marker='o')
plt.xlabel('X1: Acid Durability',fontsize=14)
plt.ylabel('X2: Strength',fontsize=14)
plt.show()
```



This plot provides a visual representation of the distribution of the data, which can be useful for understanding the underlying patterns and relationships in the data.

The color of each point is based on the value of 'Y' column

- If the value of 'Y' is 'Bad', the color is set to red

- If the value of 'Y' is 'Good', the color is set to green

1.4 Splitting the Data into Training and Testing Sets

This code is splitting the data into two sets: a training set and a testing set. The features, stored in the dataframe "X", are separated from the labels, stored in the series "y". The train_test_split function from the sklearn library is used to split the data into a training set (80% of the data) and a testing set (20% of the data). The test_size parameter is set to 0.2, indicating that 20% of the data should be set aside for testing. The random_state parameter is set to 0, ensuring that the same data split is used each time the code is run.

[11]:
```
# Split the data into training and testing sets
X = df[['X1', 'X2']]
y = df['Y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

[29]:
```
X.head()
```

[29]:

	X1	X2
0	5.077906	2.450214
1	1.220203	0.000000
2	3.333885	6.142094
3	0.000000	0.000000
4	1.070908	0.000000

[12]:
```
print(X_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40 entries, 33 to 44
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      40 non-null     float64
 1   X2      40 non-null     float64
dtypes: float64(2)
memory usage: 960.0 bytes
None
```

[13]:
```
print(X_test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10 entries, 28 to 4
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      10 non-null     float64
 1   X2      10 non-null     float64
```

```
dtypes: float64(2)
memory usage: 240.0 bytes
None
```

[14]:
```
print(y_train.describe())
```

```
count       40
unique       2
top       Good
freq        21
Name: Y, dtype: object
```

[15]:
```
print(y_test.describe())
```

```
count      10
unique      2
top       Bad
freq        6
Name: Y, dtype: object
```

1.5 Building the model and fitting on training sets

- Create an instance of the KNeighborsClassifier class with the number of neighbors (n_neighbors) to consider for the classification problem set to 1.
- KNN model is implemented with the KNeighborsClassifier imported from sklearn.neighbors and we fit the X_train, y_train data into it and make predictions on X_test data we will get the predictions in the numpy array.

[16]:
```
knn = KNeighborsClassifier(n_neighbors=5)
```

[17]:
```
# Train the model using the fit method
knn.fit(X_train, y_train)
```

[17]:
```
KNeighborsClassifier()
```

[18]:
```
y_pred= knn.predict(X_test)
y_pred
```

```
/home/h/anaconda3/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

[18]:
```
array(['Bad', 'Bad', 'Bad', 'Bad', 'Good', 'Good', 'Good', 'Bad', 'Good',
       'Bad'], dtype=object)
```

1.6 Evaluating Model Performance with Accuracy Score

The accuracy score is calculated by comparing the true labels of the test set (y_test) with the predicted labels generated by the model (y_pred). The accuracy score is a commonly used metric for evaluating classification models, as it measures the proportion of correctly classified samples in the test set.

[19]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

1.7 Predicting Labels for New Data with KNN

This demonstrates how a trained KNN model can be used to make predictions for new, unseen data.

[20]:
```
new_data = pd.DataFrame({'X1': [6,1,3], 'X2': [5,2,3]})

new_label = knn.predict(new_data)
print("Predicted Label for New Data:", new_label)
```

```
Predicted Label for New Data: ['Good' 'Bad' 'Good']
```

```
/home/h/anaconda3/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

[19]:
```
# print(confusion_matrix(y_test,y_pred))
```

2 Question 1

[21]:
```
import numpy as np

class KNN:
    def __init__(self,k=3):
        self.k = k

    def train(self,Input,Label):
        self.X = Input
```

```
        self.Y = Label

    def predict(self,tests):
        predicted = []
        for i in tests:
            distances = []
            for j in self.X:
                distances.append(np.sqrt(np.sum((i-j)**2)))
            results = []
            for j in range(len(distances)):
                results.append([distances[j],self.Y[j]])
            results.sort()
            results = results[:self.k]
            freq = {}
            for lst in results:
                if lst[1] in freq:
                    freq[lst[1]] += 1
                else:
                    freq[lst[1]] = 1
            max_count = max(freq.values())
            most_common = [k for k, v in freq.items() if v == max_count]
            predicted.append(sorted(most_common)[0])
        return predicted

Input = np.array([[7,7],[7,4],[3,4],[1,4]])
Label = np.array([0,0,1,1])

knn = KNN(k=3)
knn.train(Input,Label)

test = np.array([[3,7]])
pred = knn.predict(test)

print("Predicted labels:", pred)
```

```
Predicted labels: [1]
```

3 Question 2

4 Part(a)

[42]:
```
data = pd.read_csv("fruit_data_with_colors _1_.csv", header = 0)
```

5 Part(b)

[43]:
```
data.drop(['fruit_name','fruit_subtype'],axis=1,inplace=True)
```

6 Part(c)

[44]:
```
mean_value = data.mean()

data.fillna(mean_value, inplace=True)
```

7 Part(d)

[59]:
```
X = data[['mass', 'width','height','color_score']]
y = data['fruit_label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.16,␣
 ↪random_state=0)

Accuracy_list = []

for i in range(1,10,2):                              # from 3 to 10, with only odd␣
 ↪numbers because we take odd numbers to determine maximum.

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train, y_train)

    y_pred= knn.predict(X_test)
    print(y_pred)

    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy for k =",i," =", accuracy)
    Accuracy_list.append(accuracy)
```

```
[3 3 4 4 1 1 1 1 3 4]
Accuracy for k = 1  = 0.7
[3 3 4 4 1 1 4 3 3 4]
Accuracy for k = 3  = 0.7
[4 3 4 4 1 1 4 3 1 4]
Accuracy for k = 5  = 0.5
[4 3 4 4 1 1 4 3 3 4]
Accuracy for k = 7  = 0.6
[4 3 4 4 1 1 4 3 3 4]
Accuracy for k = 9  = 0.6
```

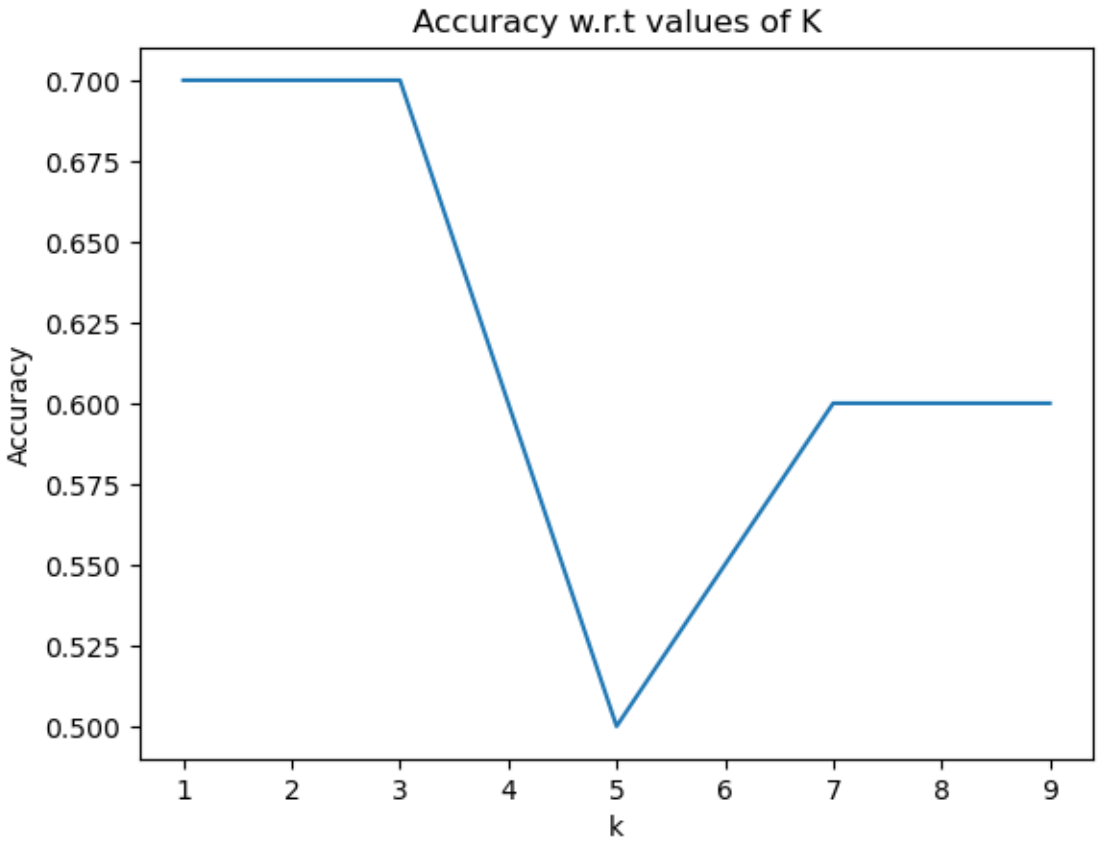
```
/home/h/anaconda3/lib/python3.9/site-
```

```
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/home/h/anaconda3/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/home/h/anaconda3/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/home/h/anaconda3/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
/home/h/anaconda3/lib/python3.9/site-
packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

8 Part(e)

[58]:
```
k = [i for i in range(1,10,2)]
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title("Accuracy w.r.t values of K")
plt.plot(k,Accuracy_list)
```

[58]: [<matplotlib.lines.Line2D at 0x7f5bb97c95b0>]



[ ]: