KNN_Classification

May 26, 2025

1 Lab: K-Nearest Neighbors Classifier

Estimated time needed: **25** minutes

1.1 Objectives

After completing this lab you will be able to:

- Use K-Nearest neighbors to classify data
- Apply KNN classifier on a real world data set

In this lab you will load a customer data set, fit the data, and use K-Nearest Neighbors to predict a data point.

1.2 Import the libraries

First, to make sure that all required libraries are available, run the cell below.

[1]:
```
!pip install numpy==2.2.0
!pip install pandas==2.2.3
!pip install scikit-learn==1.6.0
!pip install matplotlib==3.9.3
!pip install seaborn==0.13.2
```

```
Collecting numpy==2.2.0
  Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(62 kB)
Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.1 MB)
                         16.1/16.1 MB
144.5 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-2.2.0
Collecting pandas==2.2.3
  Downloading
pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(89 kB)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-
packages (from pandas==2.2.3) (2.2.0)
```

```
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas==2.2.3) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas==2.2.3) (2024.2)
Collecting tzdata>=2022.7 (from pandas==2.2.3)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas==2.2.3) (1.17.0)
Downloading
pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.7
MB)
                         12.7/12.7 MB
159.7 MB/s eta 0:00:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, pandas
Successfully installed pandas-2.2.3 tzdata-2025.2
Collecting scikit-learn==1.6.0
  Downloading scikit_learn-1.6.0-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Requirement already satisfied: numpy>=1.19.5 in /opt/conda/lib/python3.12/site-
packages (from scikit-learn==1.6.0) (2.2.0)
Collecting scipy>=1.6.0 (from scikit-learn==1.6.0)
  Downloading
scipy-1.15.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(61 kB)
Collecting joblib>=1.2.0 (from scikit-learn==1.6.0)
  Downloading joblib-1.5.1-py3-none-any.whl.metadata (5.6 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn==1.6.0)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading
scikit_learn-1.6.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(13.1 MB)
                         13.1/13.1 MB
98.3 MB/s eta 0:00:00
Downloading joblib-1.5.1-py3-none-any.whl (307 kB)
Downloading
scipy-1.15.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (37.3
MB)
                         37.3/37.3 MB
183.1 MB/s eta 0:00:0000:01
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.5.1 scikit-learn-1.6.0 scipy-1.15.3
threadpoolctl-3.6.0
Collecting matplotlib==3.9.3
  Downloading matplotlib-3.9.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib==3.9.3)
```

```
  Downloading contourpy-1.3.2-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib==3.9.3)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib==3.9.3)
  Downloading fonttools-4.58.0-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (104 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib==3.9.3)
  Downloading kiwisolver-1.4.8-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.2 kB)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-
packages (from matplotlib==3.9.3) (2.2.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (24.2)
Collecting pillow>=8 (from matplotlib==3.9.3)
  Downloading pillow-11.2.1-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (8.9
kB)
Collecting pyparsing>=2.3.1 (from matplotlib==3.9.3)
  Downloading pyparsing-3.2.3-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib==3.9.3) (1.17.0)
Downloading
matplotlib-3.9.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3
MB)
                         8.3/8.3 MB
100.3 MB/s eta 0:00:00
Downloading
contourpy-1.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (323
kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.58.0-cp312-cp312-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (4.9 MB)
                         4.9/4.9 MB
78.5 MB/s eta 0:00:00
Downloading
kiwisolver-1.4.8-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.5
MB)
                         1.5/1.5 MB
48.6 MB/s eta 0:00:00
Downloading pillow-11.2.1-cp312-cp312-manylinux_2_28_x86_64.whl (4.6 MB)
                         4.6/4.6 MB
70.6 MB/s eta 0:00:00
Downloading pyparsing-3.2.3-py3-none-any.whl (111 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.3.2 cycler-0.12.1 fonttools-4.58.0
kiwisolver-1.4.8 matplotlib-3.9.3 pillow-11.2.1 pyparsing-3.2.3
Collecting seaborn==0.13.2
  Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/opt/conda/lib/python3.12/site-packages (from seaborn==0.13.2) (2.2.0)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.12/site-
packages (from seaborn==0.13.2) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/opt/conda/lib/python3.12/site-packages (from seaborn==0.13.2) (3.9.3)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (24.2)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.12/site-
packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas>=1.2->seaborn==0.13.2) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-
packages (from pandas>=1.2->seaborn==0.13.2) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn==0.13.2)
(1.17.0)
Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
Installing collected packages: seaborn
Successfully installed seaborn-0.13.2
```

Now, let's load required libraries.

[2]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
%matplotlib inline
```

```
<h2>About the data set</h2>
```

Imagine a telecommunications provider has segmented its customer base by service usage patterns, categorizing the customers into four groups. If demographic data can be used to predict group membership, the company can customize offers for individual prospective customers. It is a classification problem. That is, given the dataset, with predefined labels, we need to build a model to be used to predict class of a new or unknown case.

The example focuses on using demographic data, such as region, age, and marital, to predict usage patterns.

The target field, called **custcat**, has four possible service categories that correspond to the four customer groups, as follows:

1. Basic Service
2. E-Service
3. Plus Service
4. Total Service

Our objective is to build a classifier to predict the service category for unknown cases. We will use a specific type of classification called K-nearest neighbors.

1.2.1 Load Data

Let's read the data using pandas library and print the first five rows.

```
[3]: df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
      ↪cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/
      ↪teleCust1000t.csv')
     df.head()
```

```
[3]:    region  tenure  age  marital  address  income  ed  employ  retire  gender  \
     0       2      13   44        1        9    64.0   4       5     0.0       0
     1       3      11   33        1        7   136.0   5       5     0.0       0
     2       3      68   52        1       24   116.0   1      29     0.0       1
     3       2      33   33        0       12    33.0   2       0     0.0       1
     4       2      23   30        1        9    30.0   1       2     0.0       0

        reside  custcat
     0       2        1
     1       6        4
```

```
2       2        3
3       1        1
4       4        3
```

<h2>Data Visualization and Analysis</h2>

Let us first look at the class-wise distribution of the data set.

```
[4]: df['custcat'].value_counts()
```

```
[4]: custcat
     3    281
     1    266
     4    236
     2    217
     Name: count, dtype: int64
```

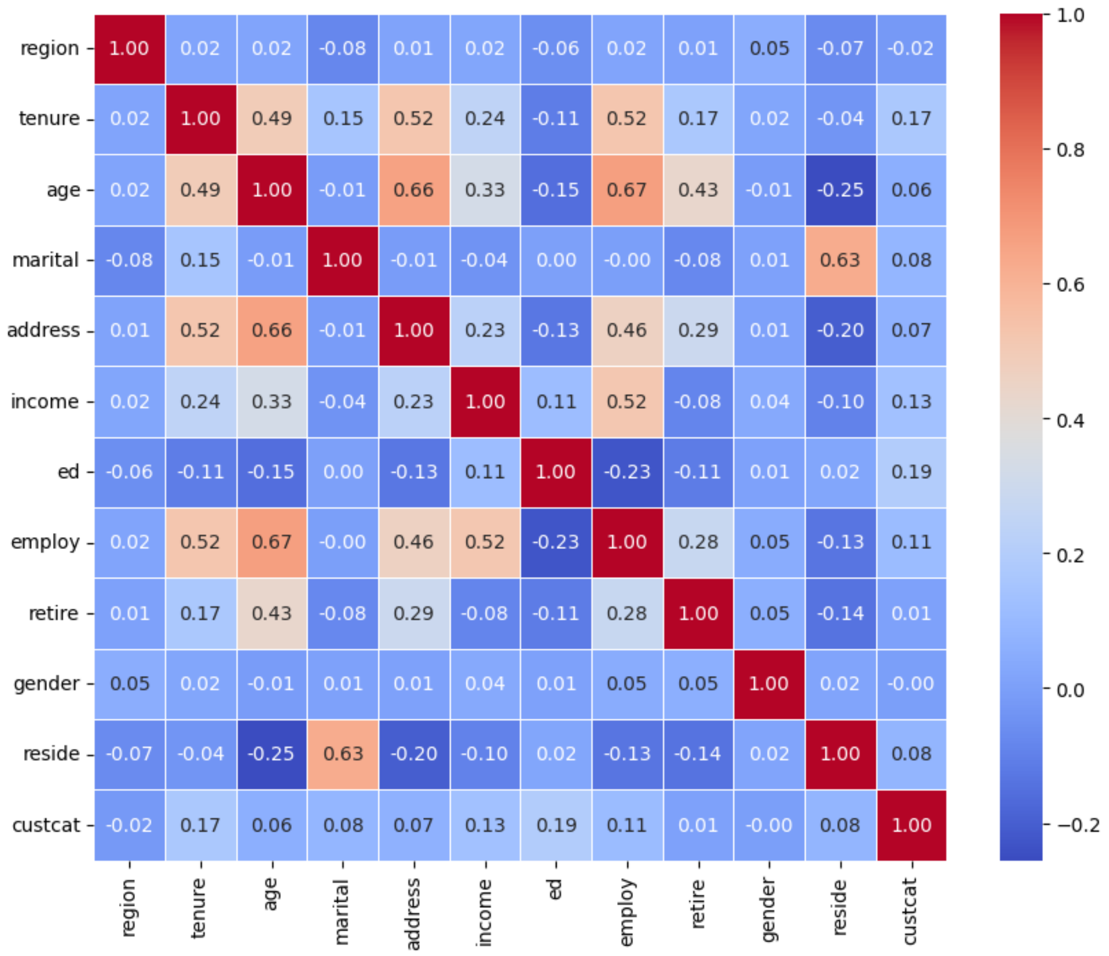
Hence, we can say that we have records of 281 customers who opt for Plus Services, 266 for Basic-services, 236 for Total Services, and 217 for E-Services. It can thus be seen that the data set is mostly balanced between the different classes and requires no special means of accounting for class bias.

We can also visualize the correlation map of the data set to determine how the different features are related to each other.

```
[5]: correlation_matrix = df.corr()

     plt.figure(figsize=(10, 8))
     sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
      ↪linewidths=0.5)
```

```
[5]: <Axes: >
```



As is visible from the correlation map, some features have beeter correlation among them than others, basically indicating the depth of relationship between the two features. What is of interest to us is the correlation of the target feature, i.e. `custcat` with all the other features. This will help us identify which features should be focussed on for modeling and which ones can be ignored.

The following code snippet will give us a list of features sorted in the descending order of their absolute correlation values with respect to the target field.

```
[6]: correlation_values = abs(df.corr()['custcat'].drop('custcat')).
       ↪sort_values(ascending=False)
     correlation_values
```

```
[6]: ed          0.193864
     tenure      0.166691
     income      0.134525
     employ      0.110011
     marital     0.083836
     reside      0.082022
```

```
address    0.067913
age        0.056909
region     0.023771
retire     0.008908
gender     0.004966
Name: custcat, dtype: float64
```

This shows us that the features `retire` and `gender` have the least effect on `custcat` while `ed` and `tenure` have the most effect.

1.2.2 Separate the input and target features

Now, we can separate the data into the input data set and the target data set.

[7]:
```
X = df.drop('custcat',axis=1)
y = df['custcat']
```

1.3 Normalize Data

Data normalization is important for the KNN model.

KNN makes predictions based on the distance between data points (samples), i.e. for a given test point, the algorithm finds the k-nearest neighbors by measuring the distance between the test point and other data points in the dataset. By normalizing / standardizing the data, you ensure that all features contribute equally to the distance calculation. Since normalization scales each feature to have zero mean and unit variance, it puts all features on the same scale (with no feature dominating due to its larger range).

This helps KNN make better decisions based on the actual relationships between features, not just on the magnitude of their values.

[8]:
```
X_norm = StandardScaler().fit_transform(X)
```

1.3.1 Train Test Split

Now, you should separate the training and the testing data. You can retain 20% of the data for testing purposes and use the rest for training. Assigning a random state ensures reproducibility of the results across multiple executions.

[9]:
```
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.2,␣
↪random_state=4)
```

1.4 KNN Classification

Once the data is in place, we can now execute the training of the model.

1.4.1 Training

Initially, you may start by using a small value as the value of k, say k = 4.

```
[10]: k = 3
      #Train Model and Predict
      knn_classifier = KNeighborsClassifier(n_neighbors=k)
      knn_model = knn_classifier.fit(X_train,y_train)
```

1.4.2 Predicting

Once the model is trained, we can now use this model to generate predictions for the test set.

```
[11]: yhat = knn_model.predict(X_test)
```

1.4.3 Accuracy evaluation

In multilabel classification, **accuracy classification score** is a function that computes subset accuracy. This function is equal to the jaccard_score function. Essentially, it calculates how closely the actual labels and predicted labels are matched in the test set.

```
[12]: print("Test set Accuracy: ", accuracy_score(y_test, yhat))
```

```
Test set Accuracy:  0.315
```

1.4.4 Exercise 1

Can you build the model again, but this time with k=6?

```
[13]: # write your code here
      k = 6
      knn_model_6 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
      yhat6 = knn_model_6.predict(X_test)
      print("Test set Accuracy: ", accuracy_score(y_test, yhat6))
```

```
Test set Accuracy:  0.31
```

Click here for the solution

```
k = 6
knn_model_6 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat6 = knn_model_6.predict(X_test)
print("Test set Accuracy: ", accuracy_score(y_test, yhat6))
```

1.4.5 Choosing the correct value of k

K in KNN, is the number of nearest neighbors to examine. However, the choice of the value of 'k' clearly affects the model. Therefore, the appropriate choice of the value of the variable `k` becomes an important task. The general way of doing this is to train the model on a set of different values of k and noting the performance of the trained model on the testing set. The model with the best value of `accuracy_score` is the one with the ideal value of the parameter k.

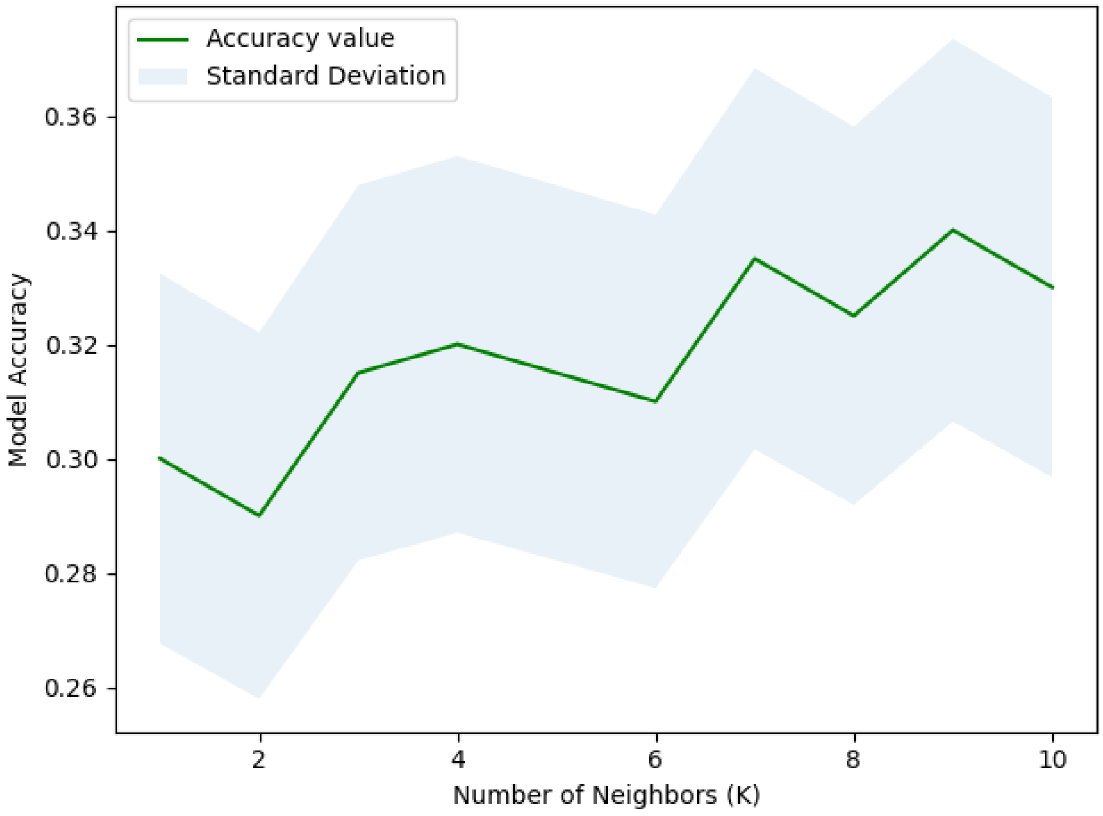
Check the performance of the model for 10 values of k, ranging from 1-9. You can evaluate the accuracy along with the standard deviation of the accuracy as well to get a holistic picture of the model performance.

[14]:
```
Ks = 10
acc = np.zeros((Ks))
std_acc = np.zeros((Ks))
for n in range(1,Ks+1):
    #Train Model and Predict
    knn_model_n = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat = knn_model_n.predict(X_test)
    acc[n-1] = accuracy_score(y_test, yhat)
    std_acc[n-1] = np.std(yhat==y_test)/np.sqrt(yhat.shape[0])
```

1.4.6 Plot the model accuracy for a different number of neighbors.

Now, you can plot the model accuracy and the standard deviation to identify the model with the most suited value of k.

[15]:
```
plt.plot(range(1,Ks+1),acc,'g')
plt.fill_between(range(1,Ks+1),acc - 1 * std_acc,acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy value', 'Standard Deviation'))
plt.ylabel('Model Accuracy')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



[16]:
```
print( "The best accuracy was with", acc.max(), "with k =", acc.argmax()+1)
```

```
The best accuracy was with 0.34 with k = 9
```

However, since this graph is still rising, there can be a chance that the model will give a better performance with an even higher value of k.

1.4.7 Exercise 2

Run the training model for 30 values of k and then again for 100 values of k. Identify the value of k that best suits this data and the accuracy on the test set for this model.

Enter you answer here

Click here for answer

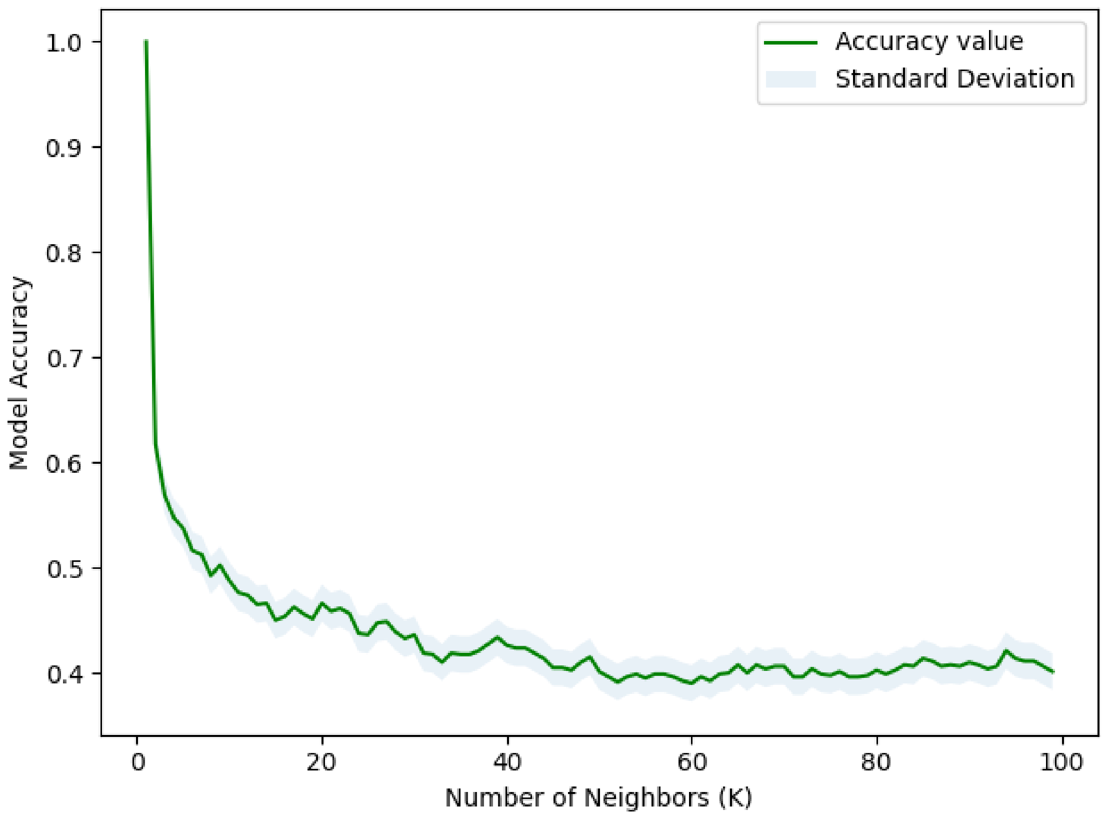
Execute the cells above by changing the value of the variable `Ks` to 30 and then try again by changing it to 100. In case of 30 values, you should find that the best value of accuracy is achieved for k=30, which again indicates that there is a further scope of improvement. In case of 100 values, you will find that the best value of accuracy is achieved for k=38, after which the model performance starts declining. Hence, the best choice of the value for k is 38, yielding 41% accuracy score.

1.4.8 Exercise 3

Plot the variation of the accuracy score for the **training set** for 100 value of Ks.

[18]:
```
# your code here
Ks =100
acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
for n in range(1,Ks):
    #Train Model and Predict
    knn_model_n = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat = knn_model_n.predict(X_train)
    acc[n-1] = accuracy_score(y_train, yhat)
    std_acc[n-1] = np.std(yhat==y_train)/np.sqrt(yhat.shape[0])

plt.plot(range(1,Ks),acc,'g')
plt.fill_between(range(1,Ks),acc - 1 * std_acc, acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy value', 'Standard Deviation'))
plt.ylabel('Model Accuracy')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



Click here for the solution

```
Ks =100
acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
for n in range(1,Ks):
    #Train Model and Predict
    knn_model_n = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat = knn_model_n.predict(X_train)
    acc[n-1] = accuracy_score(y_train, yhat)
    std_acc[n-1] = np.std(yhat==y_train)/np.sqrt(yhat.shape[0])

plt.plot(range(1,Ks),acc,'g')
plt.fill_between(range(1,Ks),acc - 1 * std_acc, acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy value', 'Standard Deviation'))
plt.ylabel('Model Accuracy')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```

1.4.9 Exercise 4

Can you justify why the model performance on training data is deteriorating with increase in the value of k?

Enter you answer here

Click here for the solution

When k is small (e.g., k=1), the model is highly sensitive to the individual points in the dataset. The prediction for each point is based on its closest neighbor, which can lead to highly specific and flexible boundaries. This leads to overfitting on the training data, meaning the model will perform very well on the training set, potentially achieving 100% accuracy. However, it may generalize poorly to unseen data. When k is large, the model starts to take into account more neighbors when making predictions. This has two main consequences: 1. Smoothing of the Decision Boundary: The decision boundary becomes smoother, which means the model is less sensitive to the noise or fluctuations in the training data. 2. Less Specific Predictions: With a larger k, the model considers more neighbors and therefore makes more generalized predictions, which can lead to fewer instances being classified perfectly.

As a result, the model starts to become less flexible, and its ability to memorize the training data (which can lead to perfect accuracy with small k) is reduced.

1.4.10 Exercise 5

We can see that even the with the optimum values, the KNN model is not performing that well on the given data set. Can you think of the possible reasons for this?

Enter you answer here

Click here for the solution

The weak performance on the model can be due to multiple reasons. 1. The KNN model relies entirely on the raw feature space at inference time. If the features do no provide clear boundaries between classes, KNN model cannot compensate through optimization or feature transformation. 2. For a high number of weakly correlated features, the number of dimensions increases, the distance between points tend to become more uniform, reducing the discriminative power of KNN. 3. The algorithm treats all features equally when computing distances. Hence, weakly correalted features can introduce noise or irrelevant variations in the feature space making it harder for KNN to find meaningful neighbours.

1.4.11 Congratulations! You're ready to move on to your next lesson!

1.5 Author

Abhishek Gagneja ### Other Contributors Jeff Grossman

<!– ## Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2024-11-29	3.1	Jeff Grossman	Review and make minor edits
2024-10-31	3.0	Abhishek Gagneja	Rewrite
2020-11-03	2.1	Lakshmi	Made changes in URL
2020-11-03	2.1	Lakshmi	Made changes in URL
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab