CSE 412: Machine Learning Lab
Lab Activity 2: KNN Classification using Scikit-learn

Dr Muhammad Abul Hasan*

February 19, 2024

1 Objective

- To be familiar with K-Nearest Neighbor(KNN) Classification and build KNN classifier using Python Scikit-learn package.

2 Background

K Nearest Neighbor(KNN) is a simple, easy to understand, versatile and one of the machine learning algorithms. KNN used in the variety of applications such as finance, healthcare, political science, handwriting detection, image recognition and video recognition. KNN algorithm based on feature similarity approach.

3 K-Nearest Neighbors

KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real world datasets do not follow mathematical theoretical assumptions. Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier. Costly testing phase means time and memory. In the worst case, KNN needs more time to scan all data points and scanning all data points will require more memory for storing training data.

4 How does the KNN algorithm work?

In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case. Suppose P1 is the point, for which label needs to predict. First, you find the one closest point to P1 and then the label of the nearest point assigned to P1.

Suppose P1 is the point, for which label needs to predict. First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance. KNN has the following basic steps:

*Associate Professor, Deptartment of CSE, GUB ⁘ ✉ muhammad.hasan@cse.green.edu.bd

1. Calculate distance
2. Find closest neighbors
3. Vote for labels

5 Eager Vs. Lazy Learners

Eager learners mean when given training points will construct a generalized model before performing prediction on given new points to classify. You can think of such learners as being ready, active and eager to classify unobserved data points.

Lazy Learning means there is no need for learning or training of the model and all of the data points used at the time of prediction. Lazy learners wait until the last minute before classifying any data point. Lazy learner stores merely the training dataset and waits until classification needs to perform. Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples. Unlike eager learning methods, lazy learners do less work in the training phase and more work in the testing phase to make a classification. Lazy learners are also known as instance-based learners because lazy learners store the training points or instances, and all learning is based on instances.

6 Curse of Dimensionality

KNN performs better with a lower number of features than a large number of features. You can say that when the number of features increases than it requires more data. Increase in dimension also leads to the problem of overfitting. To avoid overfitting, the needed data will need to grow exponentially as you increase the number of dimensions. This problem of higher dimension is known as the Curse of Dimensionality.

To deal with the problem of the curse of dimensionality, you need to perform principal component analysis before applying any machine learning algorithm, or you can also use feature selection approach. Research has shown that in large dimension Euclidean distance is not useful anymore. Therefore, you can prefer other measures such as cosine similarity, which get decidedly less affected by high dimension.

7 How do you decide the number of neighbors in KNN?

Now, you understand the KNN algorithm working mechanism. At this point, the question arises that How to choose the optimal number of neighbors? And what are its effects on the classifier? The number of neighbors(K) in KNN is a hyperparameter that you need choose at the time of model building. You can think of K as a controlling variable for the prediction model.

Research has shown that no optimal number of neighbors suits all kind of data sets. Each dataset has it's own requirements. In the case of a small number of neighbors, the noise will have a higher influence on the result, and a large number of neighbors make it computationally expensive. Research has also shown that a small amount of neighbors are most flexible fit which will have low bias but high variance and a large number of neighbors will have a smoother decision boundary which means lower variance but higher bias.

Generally, Data scientists choose as an odd number if the number of classes is even. You can also check by generating the model on different values of k and check their performance.

8 Implementation Example

In this example, we will be implementing KNN on data set named Iris Flower data set by using scikit-learn KneighborsClassifer.

- This data set has 50 samples for each different species (setosa, versicolor, virginica) of iris flower i.e. total of 150 samples.
- For each sample, we have 4 features named sepal length, sepal width, petal length, petal width)

First, import the dataset and print the features names as follows –

```python
from sklearn.datasets import load_iris
iris = load_iris()
print(iris.feature_names)
```

Output:

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

Now we can print target i.e the integers representing the different species. Here 0 = setos, 1 = versicolor and 2 = virginica.

```python
print(iris.target)
```

Output:

```
[
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
]
```

Following line of code will show the names of the target –

```python
print(iris.target_names)
```

Output:

```
['setosa' 'versicolor' 'virginica']
```

We can check the number of observations and features with the help of following line of code (iris data set has 150 observations and 4 features)

```python
print(iris.data.shape)
```

Output:

```
(150, 4)
```

Now, we need to split the data into training and testing data. We will be using Sklearn train_test_split function to split the data into the ratio of 70 (training data) and 30 (testing data) –

```python
X = iris.data[:, :4]
y = iris.target
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30)
```

Next, we will be doing data scaling with the help of Sklearn preprocessing module as follows –

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Following line of codes will give you the shape of train and test objects –

```python
print(X_train.shape)
print(X_test.shape)
```

Output:

```
(105, 4)
(45, 4)
```

Following line of codes will give you the shape of new y object –

```python
print(y_train.shape)
print(y_test.shape)
```

Output:

```
(105,)
(45,)
```

Next, import the KneighborsClassifier class from Sklearn as follows –

```python
from sklearn.neighbors import KNeighborsClassifier
```

To check accuracy, we need to import Metrics model as follows –

```python
from sklearn import metrics
```

We are going to run it for k = 1 to 15 and will be recording testing accuracy, plotting it, showing confusion matrix and classification report:

```python
range_k = range(1,15)
scores = {}
scores_list = []
for k in range_k:
   classifier = KNeighborsClassifier(n_neighbors=k)
   classifier.fit(X_train, y_train)
   y_pred = classifier.predict(X_test)
   scores[k] = metrics.accuracy_score(y_test,y_pred)
   scores_list.append(metrics.accuracy_score(y_test,y_pred))
result = metrics.confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = metrics.classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
```

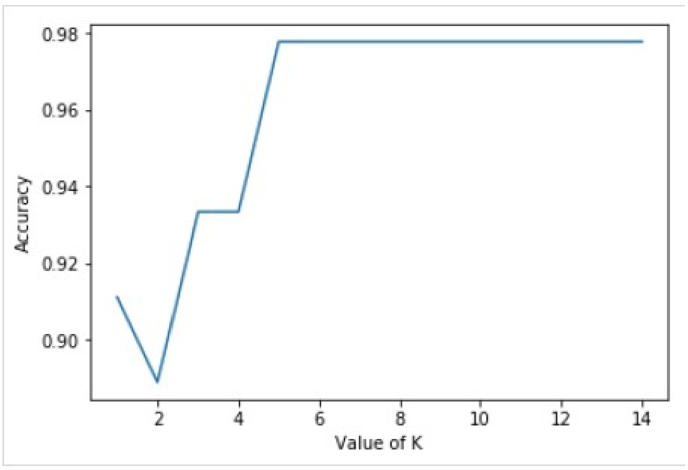
Output:

```
Confusion Matrix:
[
   [15 0 0]
   [ 0 15 0]
   [ 0 1 14]
]
Classification Report:
            precision   recall   f1-score    support
         0     1.00     1.00       1.00        15
         1     0.94     1.00       0.97        15
         2     1.00     0.93       0.97        15

micro avg      0.98     0.98       0.98        45
macro avg      0.98     0.98       0.98        45
weighted avg   0.98     0.98       0.98        45

Text(0, 0.5, 'Accuracy')
```

Now, we will be plotting the relationship between the values of K and the corresponding testing accuracy. It will be done using matplotlib library.

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(range_k,scores_list)
plt.xlabel("Value␣of␣K")
plt.ylabel("Accuracy")
```

Output:



For the above model, we can choose the optimal value of K (any value between 6 to 14, as the accuracy is highest for this range) as 8 and retrain the model as follows –

```
classifier = KNeighborsClassifier(n_neighbors = 8)
classifier.fit(X_train, y_train)
```

Output:

```
KNeighborsClassifier(
   algorithm = 'auto', leaf_size = 30, metric = 'minkowski',
   metric_params = None, n_jobs = None, n_neighbors = 8, p = 2,
   weights = 'uniform'
)
```

Testing the model with two hand made data [1,1,1,1] and [4, 3, 1.3, 0.2].

```python
classes = {0:'setosa',1:'versicolor',2:'virginicia'}
x_new = [[1,1,1,1],[4,3,1.3,0.2]]
y_predict = classifier.predict(x_new)
print(classes[y_predict[0]])
print(classes[y_predict[1]])
```

Output:

```
virginicia
virginicia
```

9 Lab Exercises

Please code yourself and write a report based on your findings:

1. Find the best K based on your experiments. To do this, take 10 accuracy for each K and calculate their average performance for finding the best K.
2. Conduct an experiment to find the best training and test set ratio in classifying Iris flowers.
3. Conduct an experiment to find the best training and test set ratio in classifying classes of your self created dataset.

My Notes Date:____________________