# Nearest Neighbor Classifier – From Theory to Practice

Amirsina Torfi
*Computer Science, Virginia Tech*
*Blacksburg, Virginia, USA*
*amirsina.torfi@gmail.com*

The K-nearest neighbors (KNNs) classifier or simply Nearest Neighbor Classifier is a kind of supervised machine learning algorithm that operates based on spatial distance measurements. In this article, we investigate the theory behind it. Furthermore, a working example of the k-nearest neighbor classifier will be represented. The blog post at: https://www.machinelearningmindset.com/nearest-neighbor-classifier/.

*Keywords*: Machine Learning; Supervised Learning; Nearest Neighbor Algorithm.

## 1. Introduction

The K-nearest neighbors (KNNs) classifier or simply Nearest Neighbor Classifier is a kind of supervised machine learning [1] algorithms used for classification or regression [2]. K-Nearest Neighbor is remarkably simple to implement, and yet performs an excellent job for basic classification tasks such as economic forecasting. It doesn't have a specific training phase as it is classifed as a non-parametric method [3]. Instead, it observes all the data while classifying a query data point. Henceforth, K-Nearest Neighbor does not have any assumption about the underlying data. This characteristic aligns with the underlying pattern of the majority of real-world datasets.

## 2. Description

The presentiment behind the K Nearest Neighbor Classifier algorithm is very simple: The algorithm classifies the new data point based on its proximity to different classes. The algorithm calculates the distance between the query data point (the unlabeled data point that supposed to be classified) and its K nearest labeled data points. Ultimately, it assigns the query data point to the class label that the majority of the K data points have.

Fig.1 demonstrates the algorithm in practice. The test sample (red circle) supposed to be classified as one of the green, blue, or yellow classes. Considering K = 3, the three closest points determine the classification outcome. As the majority vote on the red category, then the algorithm assigns yellow (yellow star) as the test sample class. Considering K = 1, the green circle is the nearest neighbor.

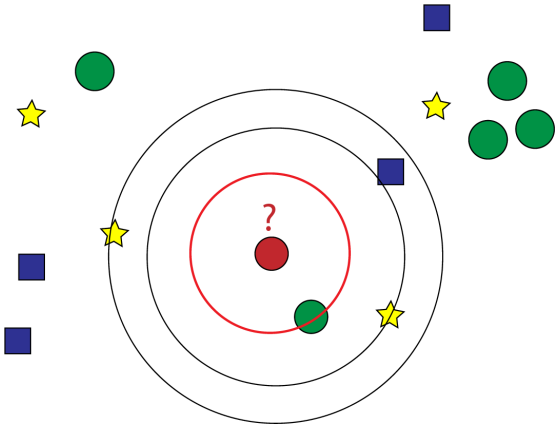


Figure 1. Visualization of an example of k-NN classification. The nearest neighbor to the query sample (red circle) is the green circle.

## 3. Benefits and limitations

Let's focus on the benefits first: **(1)** It is simple to implement as we need only two things: parameter K and the distance metric. Ordinarily used distance metrics are Euclidean distance (continuous variables) and Hamming distance (discrete variables). **(2)** As there is no training phase, the evaluation is simple and fast. However, this is regardless of the computation that leads to identifying the K nearest points. **(3)** The algorithm does not need any prior knowledge regarding the data distribution.

There are drawbacks in the algorithm as follows: **(1)** The basic "majority voting" approach becomes misleading when there is a skewed class distribution. In other words, the proximity criterion cannot adequately identify the data distribution (Fig. 2). To more elaborate on this if the number of one class samples is very frequent in the data, K-Nearest Neighbor fails to observe any meaningful pattern. **(2)** It is computationally expensive to calculate the distance between the unseen sample and all the points to find the K nearest one. Basically, the minimum number of computations for each test sample equals the number of data samples. **(3)** K-Nearest Neighbor fails often in case of encountering high-dimensional data as using the simple distance metric, it is hard to distinguish high-dimensional data.

## 4. In Practice

When we utilize KNN for classification purposes, the prediction is the class associated the highest frequency within the K-nearest instances to the test sample. Basically, each neighboring sample upvotes its associated class, and the class with the most top vote will be selected. Straightforwardly, the probability of a sample belonging to a particular class can be calculated as the fraction of the number of samples belonging to that specific class over the number of all samples as follows:

$$P(C = j) = \frac{Num(C = j)}{\sum_i Num(C = i)}$$

when $P(C = j)$ indicates the probability of class $C$ equals to $j$. It is mistakenly said that try to use an odd number to avoid and even vote! This is simply wrong. Assume we have seven samples including red, green and blue classes. We have 3, 3, and 1 samples for classes red, green and blue, respectively. K-nearest neighbor with $k = 7$ cannot vote on the red or green class! So this is the case for the only $k = 2$ which should be avoided. For parameter $k > 2$, there is no mathematical justification for choosing odd or even numbers.

### *4.1 Reading Data*

Now, let's implement KNN using Python and Scikit-learn [4]. For the first action, we read the data that we desire to use as input. Here, we utilize the iris data set, available here from the UCI Machine Learning [5] and use Pandas [6] to read it.

The data scatter plot is depicted in Fig. 2. As can be observed, the blue class is easily separable from the other classes in the 2D space.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/
    iris.data'

# Determine the column names
col_names = ['sepal_length', 'sepal_width', 'petal_length', '
    petal_width', 'categories']
iris = pd.read_csv(url, header=None, names=col_names)

### PLOT ###
category_1 = iris[iris['categories']=='Iris-setosa']
category_2 = iris[iris['categories']=='Iris-virginica']
category_3 = iris[iris['categories']=='Iris-versicolor']

fig, ax = plt.subplots()
ax.plot(category_1['sepal_length'], category_1['sepal_width'], marker=
    'o', linestyle='', ms=12, label='Iris-setosa')
ax.plot(category_2['sepal_length'], category_2['sepal_width'], marker=
    'o', linestyle='', ms=12, label='Iris-virginica')
ax.plot(category_3['sepal_length'], category_3['sepal_width'], marker=
    'o', linestyle='', ms=12, label='Iris-versicolor')
ax.legend()
plt.show()
```

### *4.2 Data processing*

After reading the data, we should process it in a way that we can represent it to the machine. This is a crucial step for all Machine Learning purposes. Basically, we prepare data in an understandable structure for the machine.

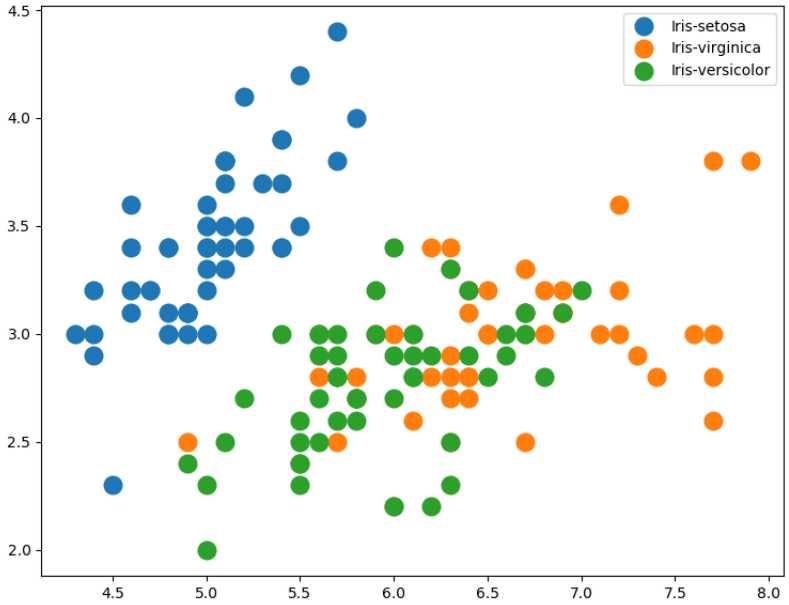


Figure 2. The scatter plot of the data distribution for three categories.

In the preprocessing phase, we perform the following: **(1)** Creating the data and label vectors and **(2)** split the data into train/test. When we split the data into train/test, the data samples are disjoint and all labels in the train set, are also available in the test set.

Both the aforementioned operations will be executed by the following source code:

```
# Creating the dictionary of categories and forming the labels vector.
iris_class = {'Iris-setosa':0, 'Iris-versicolor':1, 'Iris-virginica'
    :2}
iris['labels'] = [iris_class[i] for i in iris.categories]

# Creating the data and label vectors. The iris.drop eliminates the
    irrelevant columns.
X = iris.drop(['categories', 'labels'], axis=1)
Y = iris.labels

# Split the data into train/test.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state
    =1)
```

### *4.3 Train the classifier*

In the next step, we call the classifier by creating and fitting the model and use it to classify the test data.

```
from sklearn.neighbors import KNeighborsClassifier

## Call the model with k=10 neighbors.
knn = KNeighborsClassifier(n_neighbors=10)

## Fit the model using the training data.
knn.fit(X_train, Y_train)

## Test phase.
print(knn.score(X_test, Y_test))
```

The score will be around 97.3%.

## 5. Conclusion

In this post, we have investigated the theory behind the K Nearest Neighbor algorithm for classification. We observed its pros and cons and described how it works in practice. We also demonstrated how to implement it which aimed to bring more insight regarding the algorithm details.

## References

1. Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.
2. Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
3. William Jay Conover and William Jay Conover. Practical nonparametric statistics. 1980.
4. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
5. Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.
6. Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.