K-Nearest Neighbours

Supervised ML Algorithm

Sahasra Ranjan

April 2020

K-Nearest Neighbours is one of the most basic yet essential algorithms in Machine Learning. It has intense application in pattern recognition, data mining and intrusion detection.

The algorithm

1. Load the data
2. Initialize K
3. For each example in data
    - Calculate the distance between the query example and the current example from the data
    - Add the distance and the index of the example to an ordered collection
4. Sort the collection in ascending order.
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean else if classification, return the mode of the K labels

Chosing the right value for K

To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

Here are some things to keep in mind:

1. As we decrease the value of K to 1, out predictions become less stable.
2. Inversely, as we increase the value of K, our predictions become more stable due to majority voting (up to a certain point). Eventually, increasing K will increase the error.

3. In cases where we are taking majority vote, we usually make K an odd number to have a tiebreaker.

Advantages

- Simple and easy to implement.
- No need to build a model, tune several parameters, or make additional assumptions.
- Versatile, can be used for classification, regresion and search as well.

Disadvantages

- Slow with increase in number of examples and/or predictors.

Applications

KNN's main disadvantage of becoming significantly slower as the volume of data increases makes it an impractical choice in environments where predictions need to be made rapidly. Moreover, there are faster algorithms that can produce more accurate classification and regression results.

However, provided you have sufficient computing resources to speedily handle the data you are using to make predictions, KNN can still be useful in solving problems that have solutions that depend on identifying similar objects. An example of this is using the KNN algorithm in recommender systems, an application of KNN-search.

Implementation

```python
import numpy as np
from matplotlib import pyplot as plt

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

    def fit(self, X, y):
        # Store the original points

        self.X = X
        self.y = y

        return self

    def predict(self, X, y=None):

        # Initialize a zero distance matrix
        dists = np.zeros((X.shape[0], self.X.shape[0]))

        # Loop through all possible pairs and compute their distances
        for i in range(dists.shape[0]):
            for j in range(dists.shape[1]):
                dists[i, j] = self.distance(X[i], self.X[j])


        # Sort the distance array row-wise, and select the top k indexes for each row
        indexes = np.argsort(dists, axis=1)[:,:self.k]

        # Compute the mean of the values
        mean = np.mean(self.y[indexes], axis=1)

        return mean

    def distance(self, x, y):
        return np.sqrt(np.dot(x - y, x - y))

x = np.linspace(0, 5, 20)

m = 1.5
c = 1
y = m * x + c + np.random.normal(size=(20,))

plt.plot(x, y, 'x')

model = KNearestNeighbor(k=3)

model.fit(x,y)

predicted = model.predict(x.reshape(-1, 1))

plt.plot(
    x, y, "x",
    x, model.predict(x), "-o"
)
plt.legend(["actual", "prediction"])

plt.show()
```

Listing 1: Python Implementation for K-Nearest Neighbours Algorithm

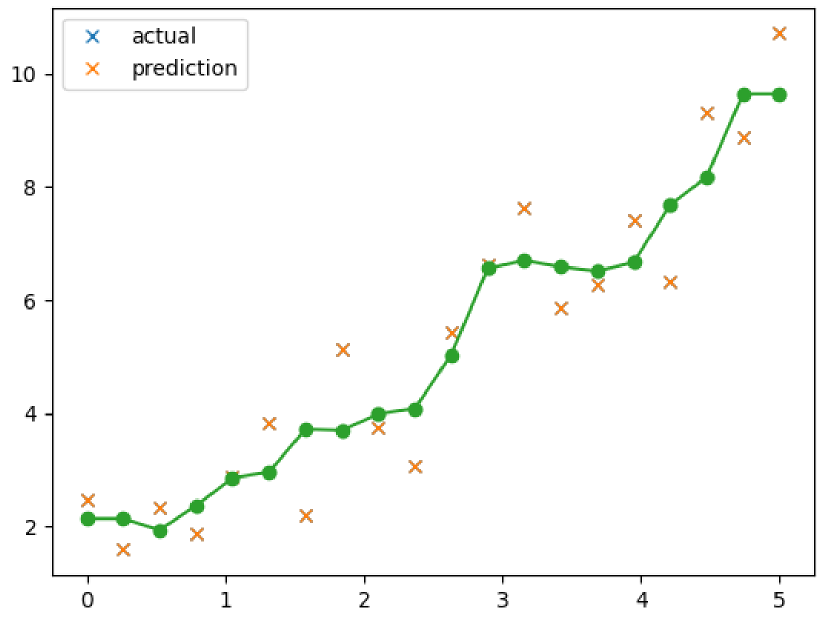

Figure 1: Matplotlib plot for above implementation