K-Nearest Neighbors (KNN) for Machine Learning

Harman Bhutani
Jan 6 · 6 min read ★

Introduction

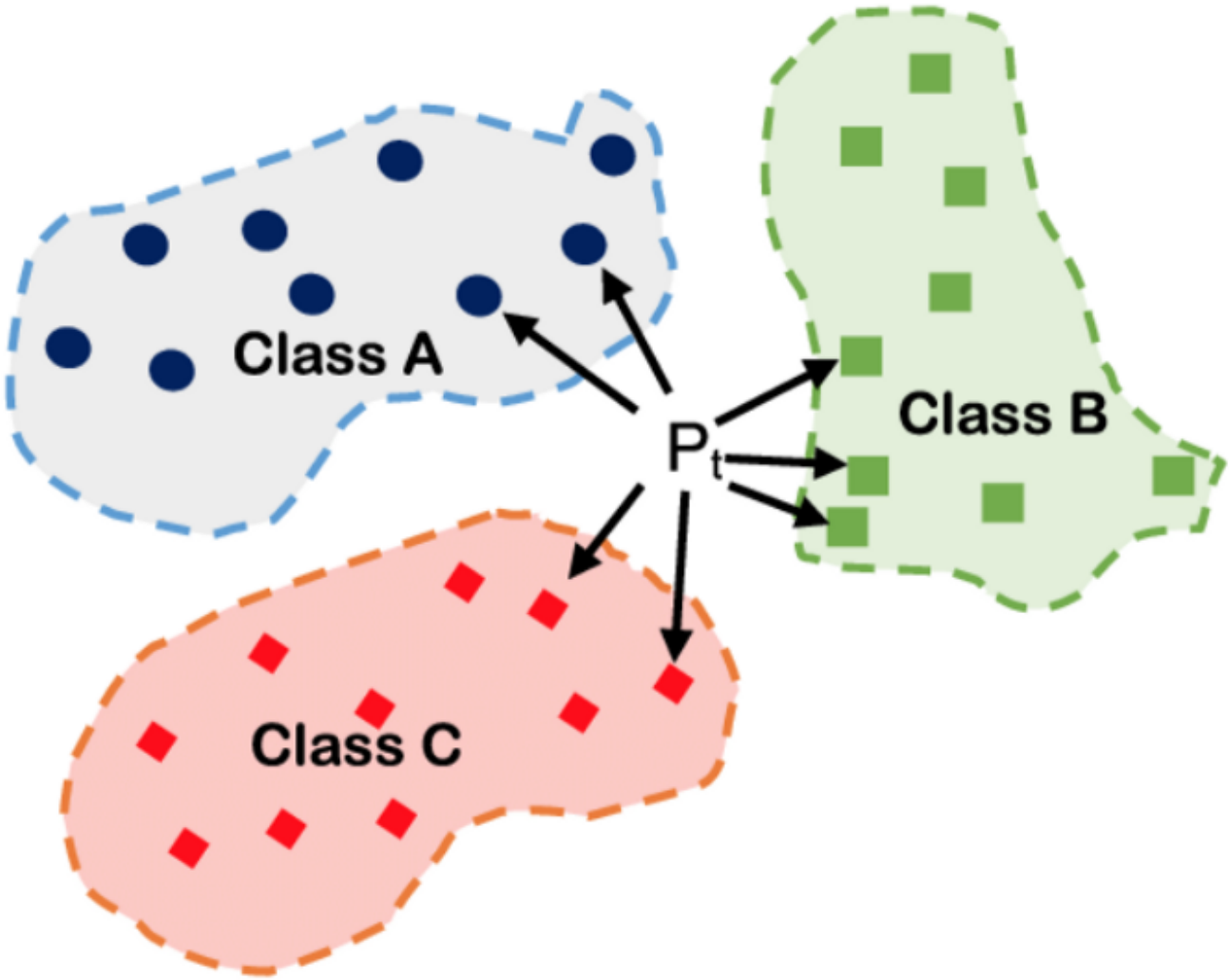
If you are familiar with machine learning and the basic algorithms used in the field, then you may have heard of the algorithm k-nearest neighbors or KNN. One of the more simple techniques used in machine learning is this algorithm. Due to its ease of use and low calculation time, it is a process preferred by many in the industry.

And what's KNN?

A simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems is the k-nearest neighbors (KNN) algorithm.

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms. Both for classification and regression, a useful technique can be used to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones.

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.



How does KNN Algorithm work?

The KNN Algorithm

1. Load the data

2. Initialize K to your chosen number of neighbors

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

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 of the K labels

8. If classification, return the mode of the K labels

To start with KNN, we need to import the CSV data for classification as shown in the below picture.

Before processing, we need to standardize the variables because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

To show the relationship between different parameters of our dataset, we plot the below graph for a better understanding of our data

Splitting the data into Training and Testing dataset

Here we are splitting our dataset into 70–30 ration with a training dataset of 70% and testing data of 30% as shown in the below picture.

Evaluating the best performance of our model

We evaluated the performance of our model when we for instance made k = 1 and the results are shown in the table.

Now we need to find the right value of K which can bring the best results for us, for that we will be making a loop, putting all values of K, starting from 1 and we will make a graph to find out which value of K gives the best result.

Choosing the right value for K

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

Here are some things to keep in mind:

1. Our predictions become less stable as we decrease the value of K from 24.
2. Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.
3. In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

Below is the code of the error rate that is required to check the error on changing the value of K and see which K is working best for our scenario.

After that, we plotted the graph between the error rate and K value, and below is the graph of it. As we can see from the graph, for the value of K = 23 or K =24, we are getting minimum error and hence our best value of K can be one of those.

We evaluated the performance of these values and below are the results:-

For K =1,

For K = 23,

For K = 24,

For K = 25,

So from the above data, we can conclude that K =24 is giving the best result.

Advantages

1. The algorithm is easy and simple to implement.
2. There is no need for a model to be built, several parameters to be tuned, or additional assumptions made.
   There's a versatile algorithm. It can be used to classify, regress, and search for (as we will see in the next section).Disadvantages
3. The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

KNN in Real World!

As the volume of data increases, KNN's main disadvantage of becoming significantly slower makes it an impractical choice in environments where predictions need to be made quickly. There are, moreover, faster algorithms that can produce more precise results for classification and regression.

However, KNN can still be useful in solving problems that have solutions that rely on the identification of similar objects if you have sufficient computing resources to quickly handle the data you are using to make predictions. An example of this is the use of the KNN algorithm, an application of KNN-search, in recommending systems.

Complete Source Code — https://github.com/HarmanBhutani/ML_projects/blob/main/K_Nearest_Neighbours.ipynb

Thank you for reading. You can connect me on Linkedin and you can follow me on Medium for more such articles.