Introduction

K Nearest Neighbor algorithm falls under the Supervised Learning category and is used for classification (most commonly) and regression. It is a versatile algorithm also used for imputing missing values and resampling datasets. As the name (K Nearest Neighbor) suggests it considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Datapoint.

The algorithm's learning is

1. Instance-based learning: Here we do not learn weights from training data to predict output (as in model-based algorithms) but use entire training instances to predict output for unseen data.

2. Lazy Learning: Model is not learned using training data prior and the learning process is postponed to a time when prediction is requested on the new instance.

3. Non -Parametric: In KNN, there is no predefined form of the mapping function.

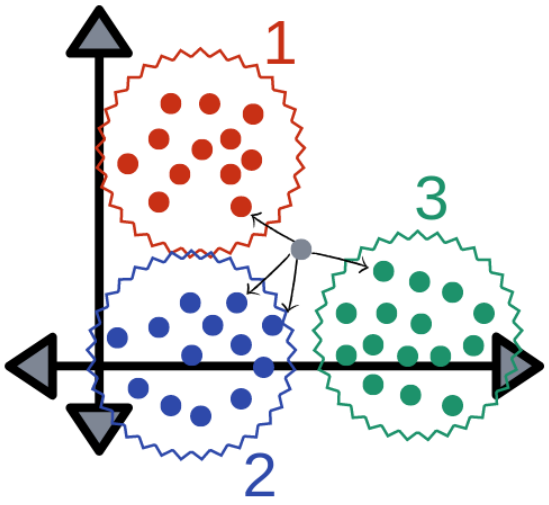
KNN is an algorithm that is considered both non-parametric and an example of lazy learning. What do these two terms mean exactly?

- Non-parametric means that it makes no assumptions. The model is made up entirely from the data given to it rather than assuming its structure is normal.
- Lazy learning means that the algorithm makes no generalizations. This means that there is little training involved when using this method. Because of this, all of the training data is also used in testing when using KNN.

Where to use KNN

KNN is often used in simple recommendation systems, image recognition technology, and decision-making models. It is the algorithm companies like Netflix or Amazon use in order to recommend different movies to watch or books to buy. Netflix even launched the Netflix Prize competition, awarding $1 million to the team that created the most accurate recommendation algorithm!

You might be wondering, “But how do these companies do this?” Well, these companies will apply KNN on a data set gathered about the movies you’ve watched or the books you’ve bought on their website. These companies will then input your available customer data and compare that to other customers who have watched similar movies or bought similar books. This data point will then be classified as a certain profile based on their past using KNN. The movies and books recommended will then depend on how the algorithm classifies that data point.



The image above visualizes how KNN works when trying to classify a data point based a given data set. It is compared to its nearest points and classified based on which points it is closest and most similar to. Here you can see the

point Xj will be classified as either W1 (red) or W3 (green) based on its distance from each group of points.

When do we use KNN algorithm?

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output

2. Calculation time

3. Predictive Power

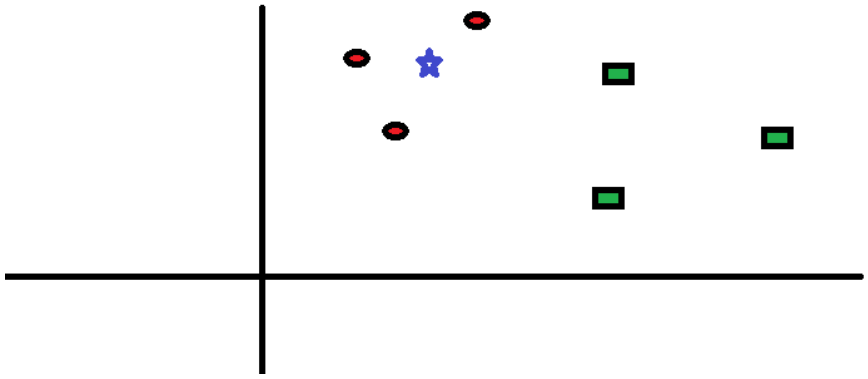
Let us take a few examples to place KNN in the scale :

	Logistic Regression	CART	Random Forest	KNN
1. Ease to interpret output	2	3	1	3
2. Calculation time	3	2	1	3
3. Predictive Power	2	2	3	2

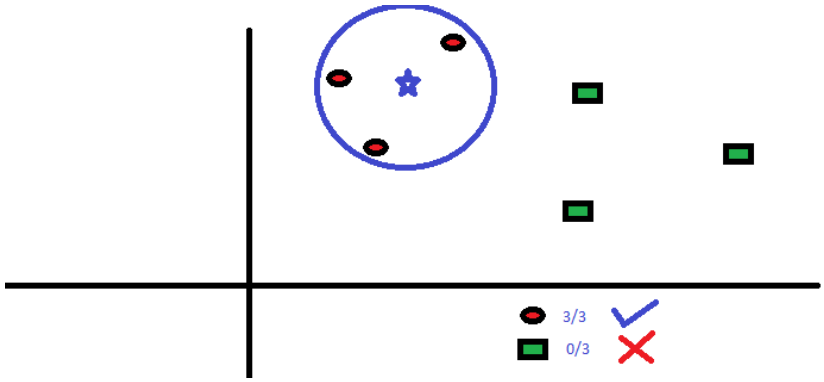
KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time.

How does the KNN algorithm work?

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) :



You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The "K" is KNN algorithm is the nearest neighbor we wish to take the vote from. Let's say K = 3. Hence, we will now make a circle with BS as the center just as big as to enclose only three datapoints on the plane. Refer to the following diagram for more details:



The three closest points to BS is all RC. Hence, with a good confidence level, we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next, we will understand what are the factors to be considered to conclude the best K.

For classification: A class label assigned to the majority of K Nearest Neighbors from the training dataset is considered as a predicted class for the new data point.

For regression: Mean or median of continuous values assigned to K Nearest Neighbors from training dataset is a predicted continuous value for our new data point

The Mathematics Behind KNN

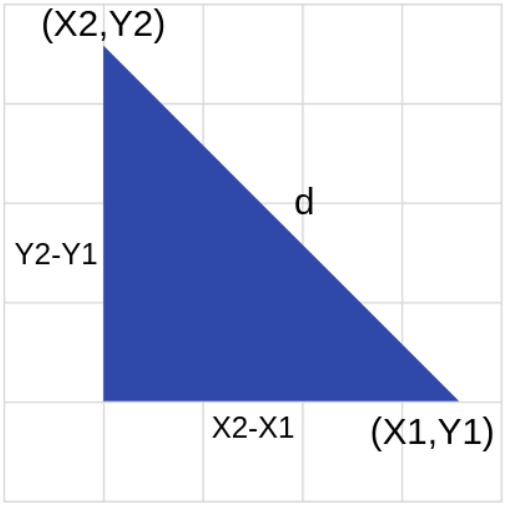
Just like almost everything else, KNN works because of the deeply rooted ..mathematical theories it uses. When implementing KNN, the first step is to transform data points into feature vectors, or their mathematical value. The algorithm then works by finding the distance between the mathematical values of these points. The most common way to find this distance is the Euclidean distance, as shown below.

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

KNN runs this formula to compute the distance between each data point and the test data. It then finds the probability of these points being similar to the test data and classifies it based on which points share the highest probabilities.

To visualize this formula, it would look something like this:



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

Breaking it Down – Pseudo Code of KNN

We can implement a KNN model by following the below steps:

1. Load the data
2. Initialise the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
    1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
    2. Sort the calculated distances in ascending order based on distance values
    3. Get top k rows from the sorted array
    4. Get the most frequent class of these rows
    5. Return the predicted class

Choosing 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:

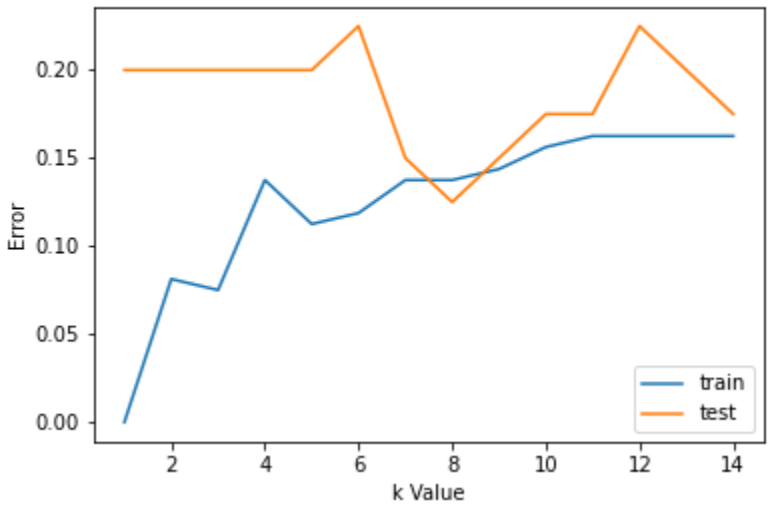
As we decrease the value of K to 1, our predictions become less stable. Just think for a minute, imagine K=1 and we have a query point surrounded by several reds and one green (I'm thinking about the top left corner of the colored plot above), but the green is the single nearest neighbor. Reasonably, we would think the query point is most likely red, but because K=1, KNN incorrectly predicts that the query point is green.

1. 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.

2. 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.

K is a crucial parameter in the KNN algorithm. Some suggestions for choosing K Value are:

1. Using error curves: The figure below shows error curves for different values of K for training and test data.



Choosing a value for K

At low K values, there is overfitting of data/high variance. Therefore test error is high and train error is low. At K=1 in train data, the error is always zero, because the nearest neighbor to that point is that point itself. Therefore though training error is low test error is high at lower K values. This is called overfitting. As we increase the value for K, the test error is reduced.

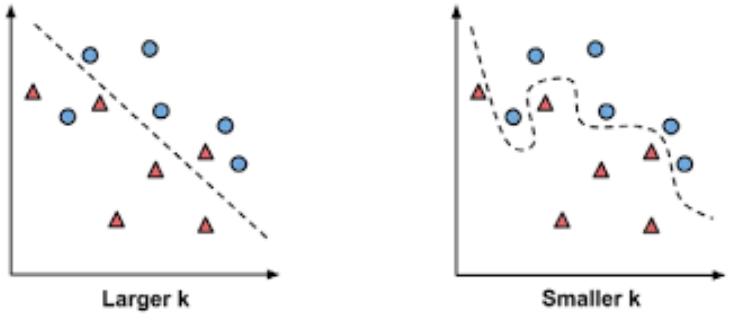
But after a certain K value, bias/ underfitting is introduced and test error goes high. So we can say initially test data error is high(due to variance) then it goes low and stabilizes and with further increase in K value, it again increases(due to bias). The K value when test error stabilizes and is low is considered as optimal value for K. From the above error curve we can choose K=8 for our KNN algorithm implementation.

2. Also, domain knowledge is very useful in choosing the K value.

3. K value should be odd while considering binary(two-class) classification.

The impact of selecting a smaller or larger K value on the model

- **Larger K value:** The case of underfitting occurs when the value of k is increased. In this case, the model would be unable to correctly learn on the training data.
- **Smaller k value:** The condition of overfitting occurs when the value of k is smaller. The model will capture all of the training data, including noise. The model will perform poorly for the test data in this scenario.



Pros:

- Easy to use.
- Quick calculation time.
- Does not make assumptions about the data.

Cons:

- Accuracy depends on the quality of the data.
- Must find an optimal k value (number of nearest neighbors).
- Poor at classifying data points in a boundary where they can be classified one way or another.