K-Nearest Neighbors - KNN

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.

KNN model by following the below steps:

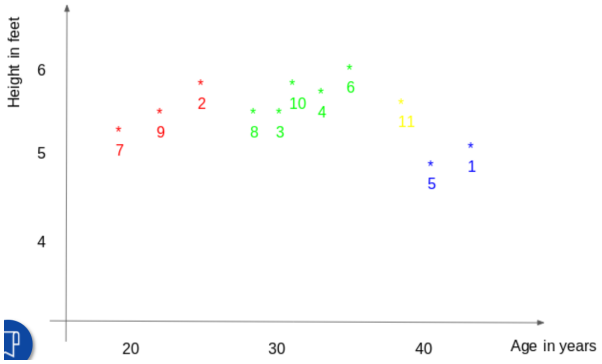
1. Load the data
2. Initialise the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
    - Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular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

Intuition behind KNN -

Let us start with a simple example. Consider the following table – it consists of the height, age and weight (target) value for 10 people. As you can see, the weight value of ID11 is missing. We need to predict the weight of this person based on their height and age.

ID	Height	Age	Weight
1	5	45	77
2	5.11	26	47
3	5.6	30	55
4	5.9	34	59
5	4.8	40	72
6	5.8	36	60
7	5.3	19	40
8	5.8	28	60
9	5.5	23	45
10	5.6	32	58
11	5.5	38	?

For a clearer understanding of this, below is the plot of height versus age from the above table-



In the above graph, the y-axis represents the height of a person (in feet) and the x-axis represents the age (in years). The points are numbered according to the ID values. The yellow point (ID 11) is our test point.

If I ask you to identify the weight of ID11 based on the plot, what would be your answer? You would likely say that since ID11 is closer to points 5 and 1, so it must have a weight similar to these IDs, probably between 72-77 kgs (weights of ID1 and ID5 from the table). That actually makes sense, but how do you think the algorithm predicts the values? We will find that out in this article.

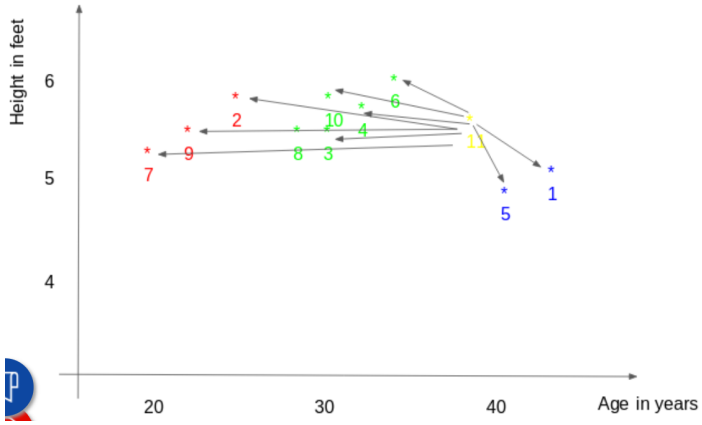
How does the KNN algorithm work?

The KNN algorithm uses **feature similarity** to predict the values of any new data points. This means that the new point is assigned a value **based on how closely it resembles the points** in the training set. From our example, we know that ID11 has height and age similar to ID1 and ID5, so the weight would also approximately be the same.

Had it been a classification problem, we would have taken the mode as the final prediction. In this case, we have two values of weight – 72 and 77. Any guesses on how the final value will be calculated? The average of the values is taken to be the final prediction.

Below is a stepwise explanation of the algorithm-

1. First, the distance between the new point and each training point is calculated.



2. The closest k data points are selected (based on the distance). In this example, points 1, 5, 6 will be selected if the value of k is 3. We will further explore the method to select the right value of k later in this article.
3. The average of these data points is the final prediction for the new point. Here, we have weight of ID11 = (77+72+60)/3 = 69.66 kg.

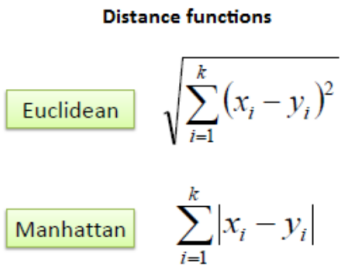
3. Methods of the calculating distance between points

The first step is to calculate the distance between the new point and each training point. Most commonly are-

1. Euclidian (for continuous)
2. Manhattan (for continuous)
3. Hamming distance (for categorical).

1. Euclidean Distance- Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (y).

2. Manhattan Distance- This is the distance between real vectors using the sum of their absolute difference.



3. Hamming Distance- It is used for categorical variables. If the value (x) and the value (y) are the same, the distance D will be equal to 0 . Otherwise D=1.

$$D_H = \sum_{i=1}^{k}|x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

Once the distance of a new observation from the points in our training set has been measured, the next step is to pick the closest points. The number of points to be considered is defined by the value of k.

4. How to choose the k factor?

The **second step is to select the k value**. This determines the number of neighbors we look at when we assign a value to any new observation.

In our example, for a value k = 3, the closest points are ID1, ID5 and ID6. The prediction of weight for ID11 will be:

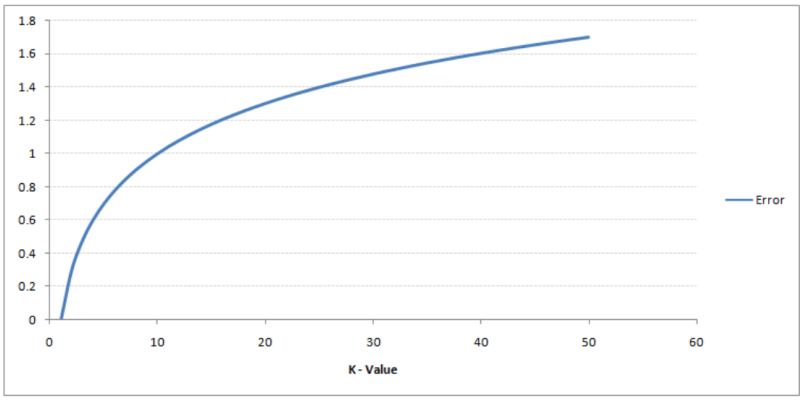
ID11 = (77+72+60)/3
ID11 = 69.66 kg

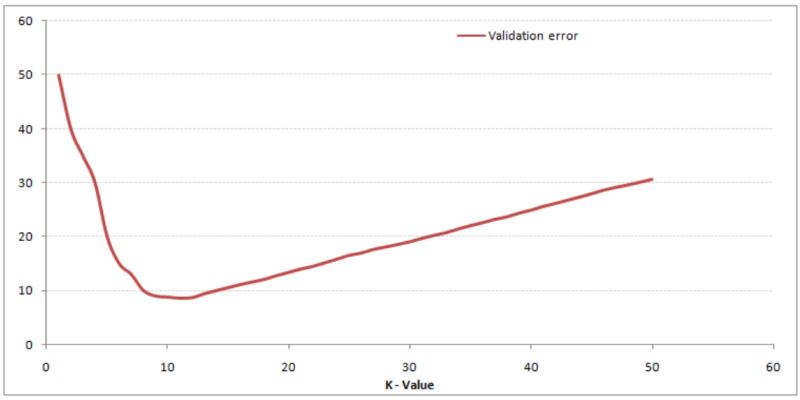
For the value of k=5, the closest point will be ID1, ID4, ID5, ID6, ID10. The prediction for ID11 will be :

ID 11 = (77+59+72+60+58)/5
ID 11 = 65.2 kg

We notice that based on the k value, the final result tends to change. Then how can we figure out the optimum value of k? Let us decide it based on the error calculation for our train and validation set (after all, minimizing the error is our final goal!).

Have a look at the below graphs for training error and validation error for different values of k.





For a very low value of k (suppose k=1), the model overfits on the training data, which leads to a high error rate on the validation set. On the other hand, for a high value of k, the model performs poorly on both train and validation set.

If you observe closely, the validation error curve reaches a minima at a value of k = 9. This value of k is the optimum value of the model (it will vary for different datasets). This curve is known as an **elbow curve** (because it has a shape like an elbow) and is usually used to determine the k value.

You can also use the grid search technique to find the best k value. We will implement this in the next section.

5. Work on a dataset (Python codes)

Refer this link - [https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/ (https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/)](https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/)

Bias means how much on an average are the predicted values different from the actual value.
Variance means how different will the predictions of the model be at the same point if different samples are taken from the same population.

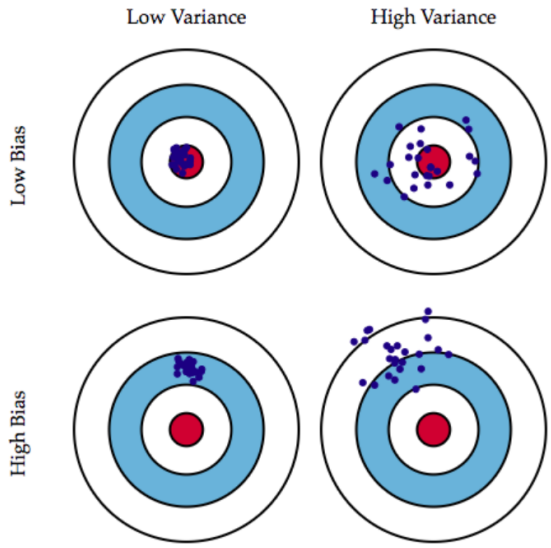
Underfitting vs Overfitting -

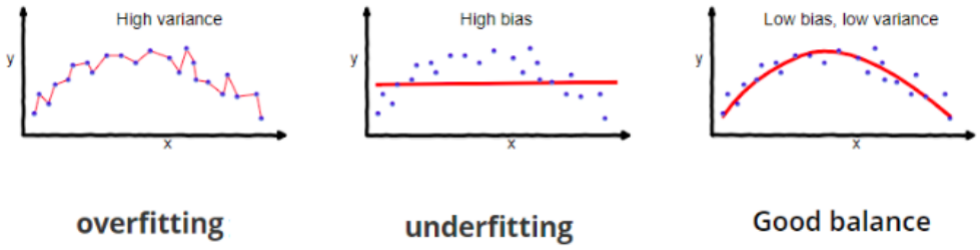
Overfitting means that the model predict the (training) data too well whereas

- It is too good to be true. If the new data point comes in, the prediction may be wrong.
- Overfitting implies low bias but high variance.

Underfitting means the model does not fit, it means it does not predict, the (training) data very well.

- Underfitting implies high bias and low variance.

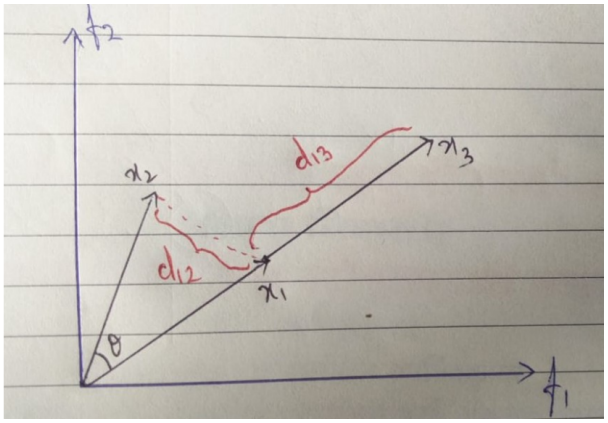




Cosine-similarity and Cosine-distance”

Cosine-similarity considers the angle (θ) between the two points and is equal to the cos(θ). Consider the below example-

A and B are two data points and θ is the angle between them. It is safe to say that if the distance between the two points increases, the angle between the two will increase and as the angle between the two increases the cos(θ) decreases means their cosine-similarity decreases.



We have 3 vectors — x1, x2 and x3.
Cosine-similartiy (x1, x2) = cos(θ)
Cosine-similarity (x1, x3) = 1 (since, θ = 0 and cos(0) = 1)
Cosine-distance (x1, x2) = 1 — cos(θ)
Cosine-distance between (x1, x3) = 1–1 = 0 (since cos(θ) = 1

So the observations are, in terms of Euclidean distance, its clear that distance d13 > d12

But when comparing cosine-distance, cos-dist (x1, x3) < cos-dist (x1, x2)

From the above observations, it is pretty clear that cos-sim or cos-dist uses the angle between the vectors and not the geometric distance.

Is there a relation between Euclidean and cosine-similarity?
Squared Euclidean distance between x1 and x2 = 2*(1 — cos(θ))

Cross Validation -

In cross-validation, instead of splitting the data into two parts, we split it into 3. Training data, cross-validation data, and test data. Here, we use training data for finding nearest neighbors, we use cross-validation data to find the best value of “K” and finally we test our model on totally unseen test data. This test data is equivalent to the future unseen data points.

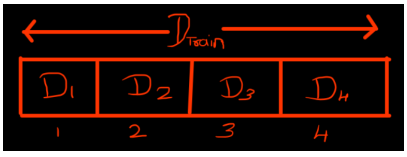
Under the cross-validation part, we use D_Train and D_CV to find KNN but we don’t touch D_Test. Once we find an appropriate value of “K” then we use that K-value on D_Test, which also acts as a future unseen data, to find how accurately the model performs.

Here are the steps involved in cross validation:

1. You reserve a sample data set.
2. Train the model using the remaining part of the dataset
3. Use the reserve sample of the test (validation) set. This will help you in gauging the effectiveness of your model’s performance. If your model delivers a positive result on validation data, go ahead with the current model. It rocks!

K-Fold Cross-Validation -

After splitting the total data set (D_n) into training (D_Train) and test (D_Test) data set, in the ratio of 80:20, we further randomly split the training data into 4 equal parts.



D1, D2, D3, and D4 are the four randomly split equal parts of D_Train. Once done with the splitting, we proceed as follows-

Value of K	Training Data	Cross Validation	Accuracy
K=1	D1, D2, D3	D4	A4
K=1	D1, D2, D4	D3	A3
K=1	D1, D3, D4	D2	A2
K=1	D2, D3, D4	D1	A1
K=2	D1, D2, D3	D4	A4
K=2	D1, D2, D4	D3	A3
K=2	D1, D3, D4	D2	A2
K=2	D2, D3, D4	D1	A1

Step-1- For K=1, I pick D1, D2, and D3 as my training data set and set D4 as my cross-validation data and find the nearest neighbors and calculate its accuracy.

Step-2- Again, for K=1, I pick D1, D2, and D4 as my training data set and set D3 as my cross-validation data, I find the nearest neighbors and calculate its accuracy.

I repeat the above steps with D2 and D1 as my cross-validation data set and calculate the corresponding accuracy. Once done with it, I get 4 accuracies for the same value of K=1. So I consider the mean of these accuracies and assign it as the final value when my K=1.

Now, I repeat the above steps for K=2 and find the mean accuracy for K=2. So on and so forth, I calculate the accuracies for different values of K.

Now, note that for each value of K I had to compute the accuracy 4 times. This is because I randomly split my training data set into 4 equal parts. Suppose I had randomly split my data set into 5 equal parts then I would have to compute 5 different accuracies for each value of K and take their mean.

Capital “K” stands for the K value in KNN and lower “k” stands for k value in k-fold cross-validation.

So, k value in k-fold cross-validation for the above example is 4 (i.e k=4), had we split the training data into 5 equal parts, the value of k=5.

k = number of parts we randomly split our training data set into. Now, we are using the entire 80% of our data to compute the nearest neighbors as well as the K value in KNN.

Types of Cross Validation commonly used are -

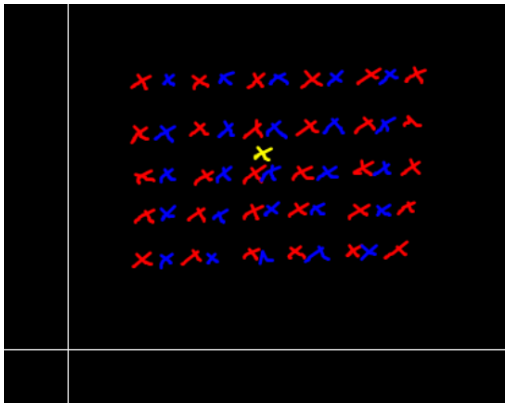
1. The validation set approach
2. Leave one out cross validation (LOOCV)
3. k-fold cross validation
4. Stratified k-fold cross validation
5. Adversarial Validation
6. Cross Validation for time series
7. Custom Cross Validation Techniques

Refer Types of CV -

1. https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/ (https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/)
2. https://towardsdatascience.com/understanding-8-types-of-cross-validation-80c935a4976d (https://towardsdatascience.com/understanding-8-types-of-cross-validation-80c935a4976d)

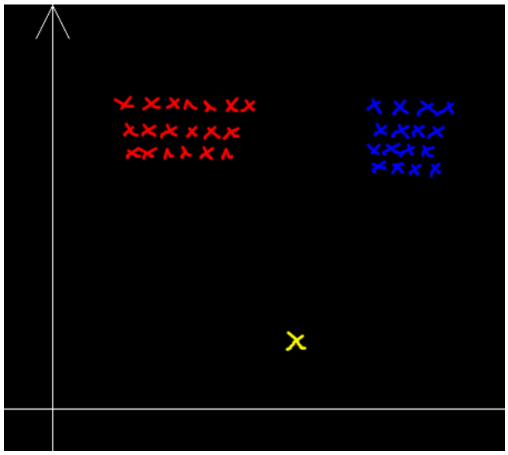
Failure cases of KNN -

1. CASE-1 Consider the below example-



In this case, the **data is randomly spread** and hence no useful information can be obtained from it. Now in such a scenario when we are given a query point (yellow point), the KNN algorithm will try to find the k nearest neighbors but since the data points are jumbled, the accuracy is questionable.

2. CASE-2 Consider the below example-



In this case, the data is grouped in clusters but the **query point seems far away** from the actual grouping. In such a case, we can use K nearest neighbors to identify the class, however, it doesn't make much sense because the query point (yellow point) is really far from the data points and hence we can't be very sure about its classification.

Limitations of KNN:

KNN is a very powerful algorithm. It is also called "lazy learner". However, it has the following set of limitations-

1. **Doesn't work well with a large dataset**
   Since KNN is a distance-based algorithm, the cost of calculating distance between a new point and each existing point is very high which in turn degrades the performance of the algorithm.

2. **Doesn't work well with a high number of dimensions-**
   Again, the same reason as above. In higher dimensional space, the cost to calculate distance

becomes expensive and hence impacts the performance.

3. **Sensitive to outliers and missing values-**
   KNN is sensitive to outliers and missing values and hence we first need to impute the missing values and get rid of the outliers before applying the KNN algorithm.

Time Series Split -

In a Machine Learning algorithm we can split the given dataset into training and test data. We can either split randomly or use time based splitting.

For Time based splitting we need a timestamp as one of the attributes / features.

Like in case of e-commerce website we can have reviews for various products. These reviews can have timestamps also. In such scenarios it's better to use time-based strategy.

To do this we can first sort the reviews using timestamp and then do the split.

1. Sort data by time.
2. Split – Training (80%) and Testing(20%)

This approach can give better accuracy. Since the testing data will be more recent and hence better prediction.

Locality Sensitive Hashing -

Locality sensitive hashing (LSH) is a method to generate hashcodes for data-points with the property that similar data-points will have the same hashcodes.

LSH has many applications, including-
1. Near-duplicate detection- LSH is commonly used to deduplicate large quantities of documents, webpages, and other files.
2. Genome-wide association study- Biologists often use LSH to identify similar gene expressions in genome databases.
3. Large-scale image search- Google used LSH along with PageRank to build their image search technology VisualRank.
4. Audio/video fingerprinting- In multimedia technologies, LSH is widely used as a fingerprinting technique A/V data.

When we say that we store the documents inside Keys present in Hash Table, by Keys we mean Hash Codes. A Hash Code is a K-bit binary code with which we identify a document. The unique fingerprint that we talked about, this Hash Code actually provides that uniqueness. K can be any value. A Hash Code can be 4 bit, 8 bit, 32 bit, etc. A Simple Hash Code may look like this: 11010011

Given below is an example Hash Code generation algorithm- Adler Algorithm

ADLER ALGORITHM -

We are using this algorithm to generate Hash Codes for a string, or a document.

In this algorithm we go character by character of a string. We look at each character and then find its ASCII code. We maintain two values, constantly, during this process. Lets call it A and B.

A will store a normal sum of all the ASCII values of the characters, while B will store the cumulative sum of the ASCII Values. Let us see how it works-

	a		c	a	b
ASCII Value	97	32	99	97	98
B	97	226	454	779	1202
A	97	129	228	325	423

In the table above we can see that we have four ASCII Values, for a, c, b and space. A maintains the normal sum, whose total is 423, while B Maintains the Cumulative Sum and the total is 1202.

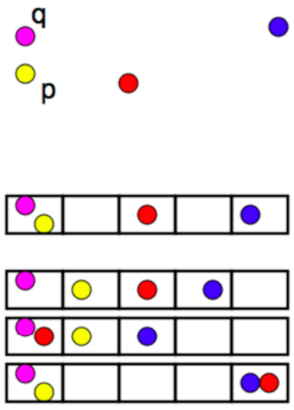
Next step is to find the hash code. Suppose we want an 8 bit hash code. So, for that, we have to choose a maximum prime number of 8 bit. In this case we will select 13, whose binary counterpart is of 8 bit. Once we have decided the number, we will find the modulus of A and B.

For A: 423%13 = 7
For B: 1202%13 = 6

Binary of 7 is 111, while binary of 6 is 110. Now, final task is just to concatenate these two numbers, which leaves us with: 01100111 (Added extra 0 to make it 8 bit). Now, this is our final Hash code for a particular string.

Similarly, we can find hash code for the entire document. Remember, before applying this we have to remove all the things which are not part of the main content and other things like tags, punctuation etc. We use prime number to make the Hash Code as random as possible.

LSH — How it works?



Now that we know basics of Hash Code generation, let us see how LSH works. Process is exactly the same: Put similar documents in one key inside the Hash Table. As I mentioned before, for exact duplicates finding similarity is very easy, but for near duplicates process becomes tough. LSH tries to remove this problem to an extent.

In LSH, we repeat the process of finding Hash Codes and then assigning keys inside Hash Table, multiple times. Therefore, we have multiple Hash Tables with same document assigned to different keys. Because we have repeated this multiple times, there will come a time where two near duplicates will be in the same bucket (or share the same key), hence removing the issue.

For example, look at the table below-

	Table 1	Table 2	Table 3
Doc _2	1001	1011	1101
Doc_5	1100	1011	1011

In the above Table, we are iterating three times. This means that we will be having three Hash Tables. Suppose Document 2 and Document 5 are near duplicates. In table 1, two different keys are assigned. Similarly, in table 3 also two different keys are assigned. But, in table 2 we have same key. Hence we can see that in three iteration, one of the Hash Tables has correctly identified the near duplicate documents.

How the Hash codes are generated - https://medium.com/analytics-vidhya/locality-sensitive-hashing-finding-documents-similarity-6f12d4d83af (https://medium.com/analytics-vidhya/locality-sensitive-hashing-finding-documents-similarity-6f12d4d83af)

Q. Diff betw Classfication and Regression

Ans. The main difference lies in output :-

1. Classfication - y is finite or categorical or discrete.
2. Regression - y is Real no. or numerical or continuous