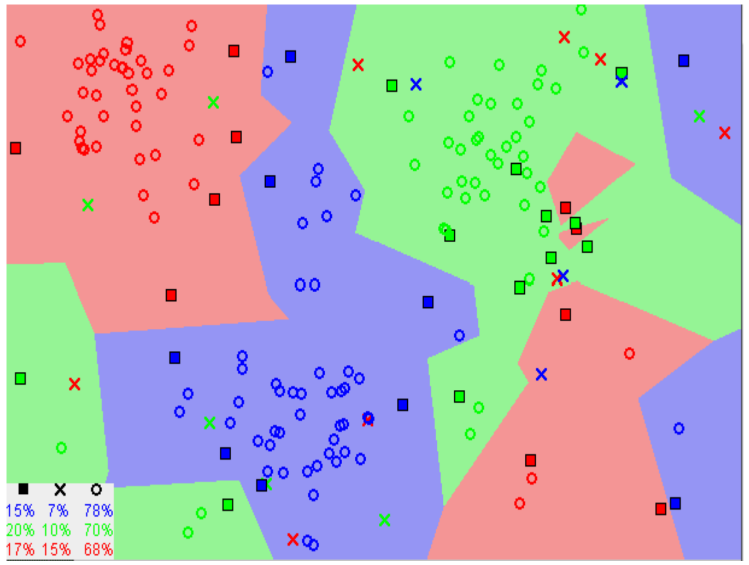
K- Nearest Neighbors (KNN)

K- Nearest Neighbors

KNN stands for K-Nearest Neighbors, and it is a supervised machine learning algorithm used for classification and regression tasks. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. KNN captures the idea of **similarity** (sometimes called **distance, proximity**, or **closeness**) by calculating the distance between points on a graph.

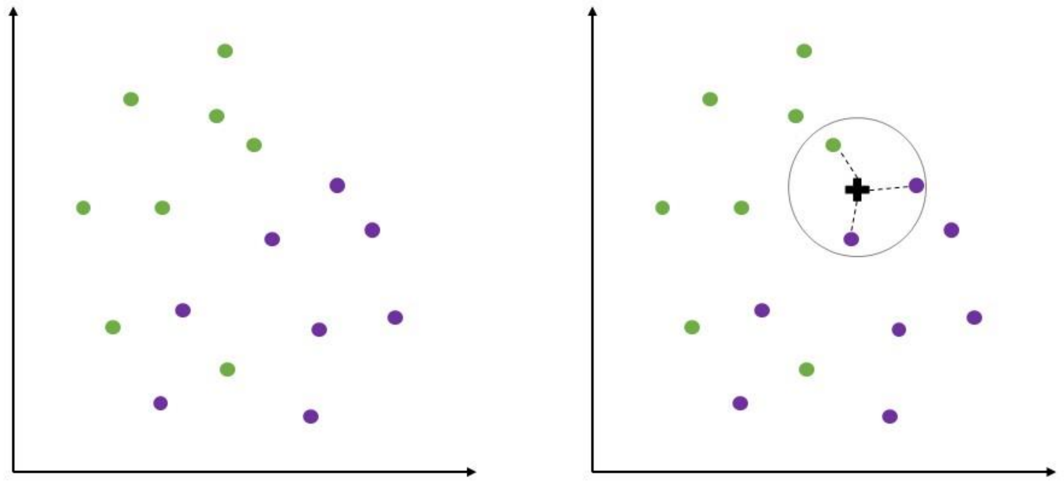
In KNN, the idea is to predict the class of a given data point by looking at the 'K' closest labeled data points in the training set. KNN is non-parametric/instance based learning, which means that the algorithm does not make assumptions about the underlying distributions of the data. In contrast to a technique like linear regression, which is parametric/model based learning, and requires us to find a function that describes the relationship between dependent and independent variables.

Instance-based methods are sometimes called lazy learning methods because they postponed learning until the new instance/data point is encountered for prediction. There is no math or processes before testing the model, all it does is finding closest K points.

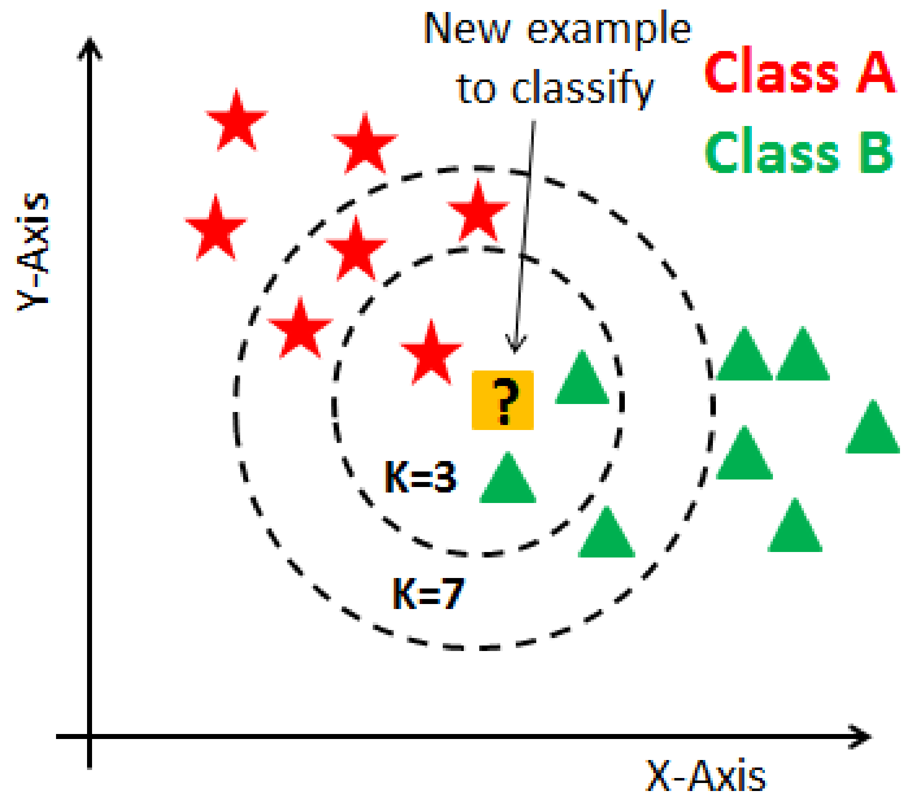


K- Nearest Neighbors

See the following example. The right panel shows how a new point (the black cross) is classified, using KNN when **k**=3.
We find the three closest points, and count how many 'votes' each color has within those points. In this case, two of the three points are purple. So, the black cross will be labeled as purple.



Which class the new data belongs to?



for k = 3?
for k = 7?

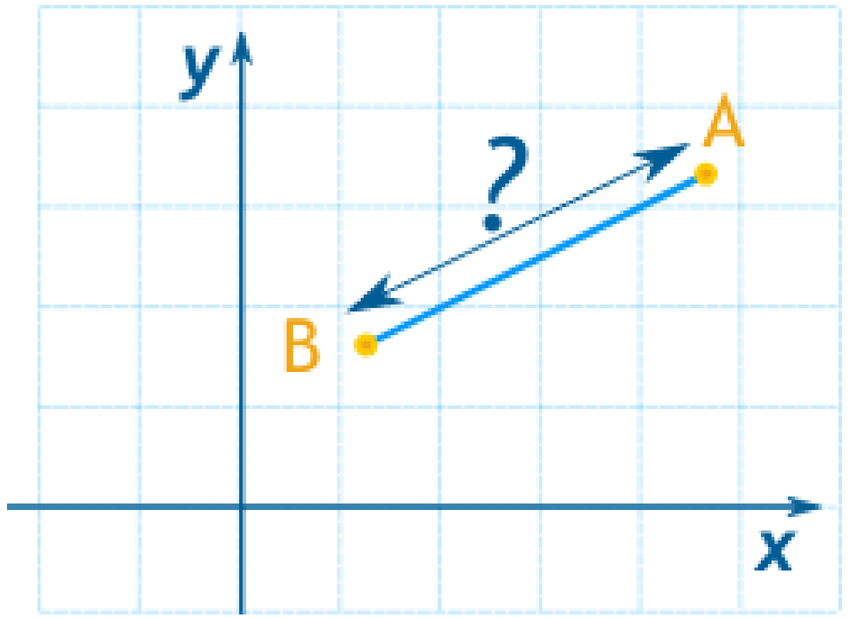
KNN Algorithm

1. Load the given data file into your program
2. Initialize the number of neighbors to be considered i.e. ‘K’ (must be odd to avoid ties in voting).
3. Now for each tuple (entries or data point) in the data file we perform:
    1. Calculate the distance between the data point (tuple) to be classified and each data points in the given data file.
    2. Then add the distances corresponding to data points (data entries) in the given data file (probably by adding a column for distance).
    3. Sort the data in the data file from smallest to largest (in ascending order) by the distances.
4. Pick the first K entries from the sorted collection of data.
5. Observe the labels of the selected K entries.
6. For classification, return the mode of the K labels and for regression, return the mean of K labels.

Distance Metrics

For the algorithm to work best on a particular dataset we need to choose the most appropriate distance metric accordingly. Distance metric is a function that measures the distance between two points in a N-dimensional space. The choice of distance metric is important in KNN since it determines how the algorithm measures the similarity between data points. There are a lot of different distance metrics available, but we are only going to talk about a few widely used ones.

The choice of distance metric depends on the data and the problem at hand. In general, Euclidean distance is a good default choice for continuous data, while Manhattan distance is more suitable for categorical or ordinal data. Chebyshev distance can be useful when working with data that has a lot of outliers or when you want to emphasize differences in only one dimension. Euclidean distance function is the most popular one among all of them as it is set default in the Sklearn KNN classifier library in python.



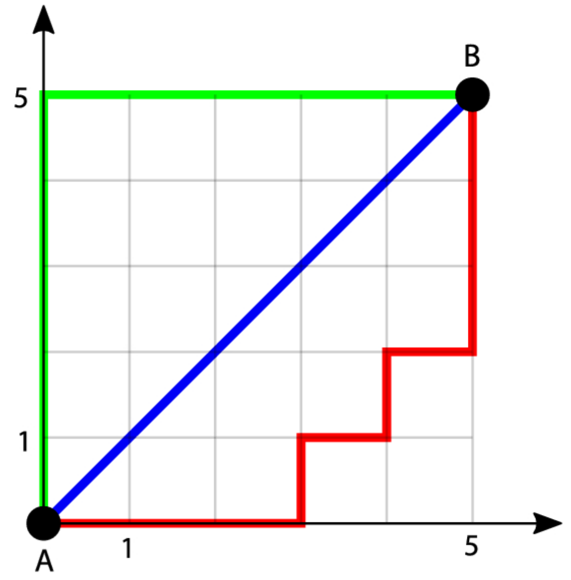
Distance Metrics – Manhattan Distance

Manhattan distance, also known as L1 distance or taxicab distance, is a measure of distance between two points in a N-dimensional space. It is named after the grid-like layout of Manhattan streets, where the shortest distance between two points is the sum of the absolute differences between their coordinates.

To compute the Manhattan distance between two points, you add up the absolute differences between their coordinates along each dimension. In other words, you take the absolute value of the difference in each dimension, add them up, and that gives you the Manhattan distance.

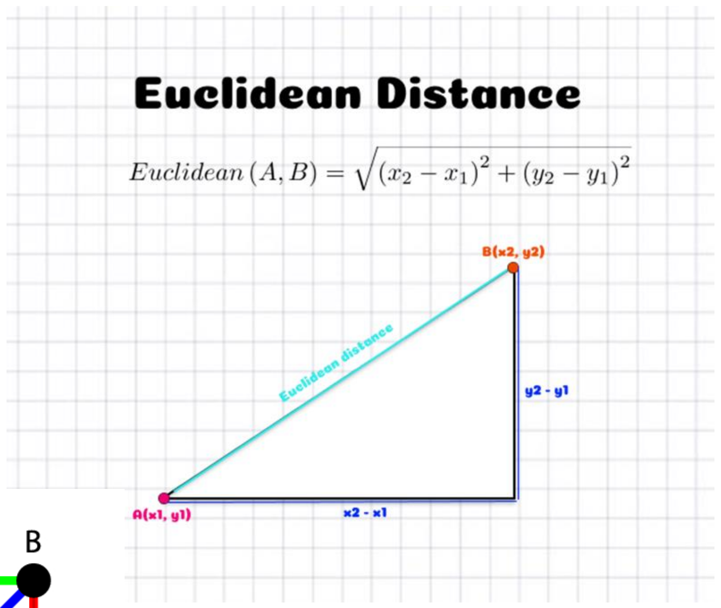
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



Distance Metrics – Euclidean Distance

The Euclidean distance between two points A and B in a N-dimensional space is defined as the straight-line distance between the two points. Mathematically, it is calculated as the square root of the sum of the squared differences between the coordinates of the two points, i.e.,

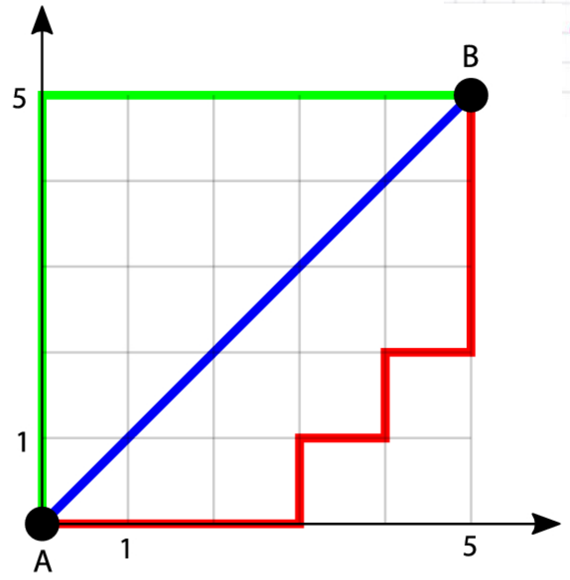


Euclidean Distance

$$Euclidean\,(A,B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



Distance Metrics – Minkowski Distance

Minkowski distance is a generalized distance metric that includes both Manhattan distance and Euclidean distance as special cases. It is named after the Russian mathematician Hermann Minkowski, who first introduced it. The Minkowski distance between two points A and B in a N-dimensional space is defined as:

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

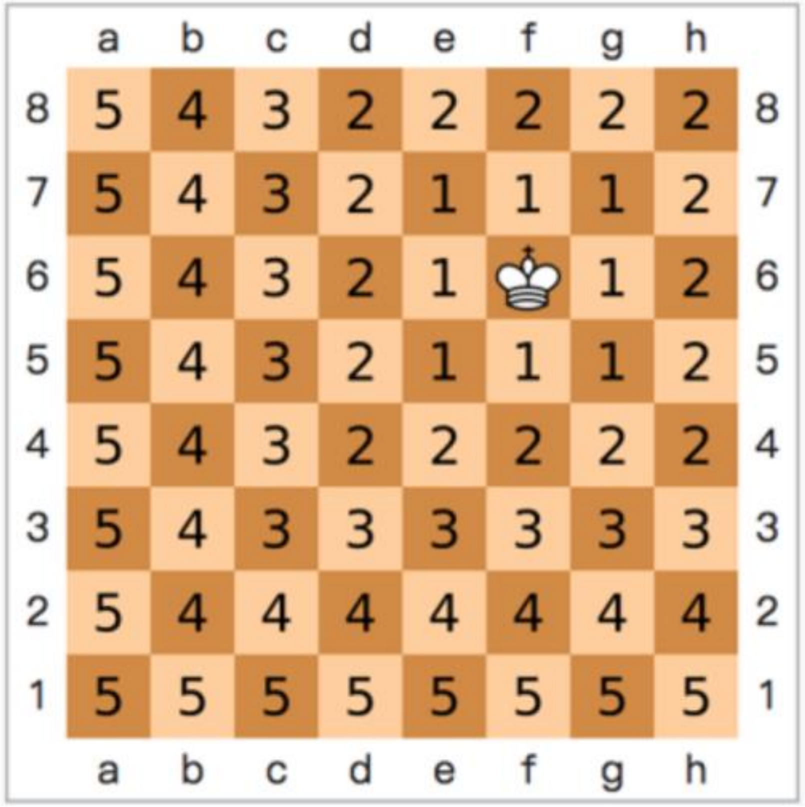
where i ranges from 1 to N, and p is a parameter that determines the "order" of the distance metric. When p=1, the Minkowski distance becomes the Manhattan distance, and when p=2, it becomes the Euclidean distance. In general, the larger the value of p, the more emphasis is placed on large differences in individual coordinates. The choice of p depends on the data and the problem at hand, and a good value of p can often be found through experimentation and validation.

Distance Metrics – Chebyshev Distance

Also known as the chessboard distance because "*in the game of chess the minimum number of moves needed by a king to go from one square on a chessboard to another equals the Chebyshev distance between the centers of the squares.*"

$$d(x, y) = \max_i |x_i - y_i|$$

Chebyshev distance is useful when we want to emphasize differences in only one dimension or when working with data that has a lot of outliers. However, it may not be the best choice for all types of data, such as continuous data with low variance,



	a	b	c	d	e	f	g	h	
8	5	4	3	2	2	2	2	2	8
7	5	4	3	2	1	1	1	2	7
6	5	4	3	2	1	♔	1	2	6
5	5	4	3	2	1	1	1	2	5
4	5	4	3	2	2	2	2	2	4
3	5	4	3	3	3	3	3	3	3
2	5	4	4	4	4	4	4	4	2
1	5	5	5	5	5	5	5	5	1
	a	b	c	d	e	f	g	h	

Choosing the K value

- There is no structured method to find the best value for “K”. We need to find out with various values by **trial and error** and assuming that training data is unknown.
- Choosing smaller values for K can be noisy and will have a higher influence on the result.
- Larger values of K will have smoother decision boundaries which mean lower variance but increased bias. Also, computationally expensive.
- Another way to choose K is through cross-validation. We’ll discuss later
- In general, practice, choosing the value of **k** is **k = sqrt(N)** where **N** stands for the **number of samples in your training dataset**.
- Try and keep the value of k odd in order to avoid confusion between two classes of data

Advantages of KNN

1. No Training Period: KNN is called **Lazy Learner (Instance based learning)**. It does not learn anything in the training period. It does not derive any discriminative function from the training data. In other words, there is no training period for it. It stores the training dataset and learns from it only at the time of predictions.

2. Since the KNN algorithm requires no training before making predictions, **new data can be added seamlessly** which will not impact the accuracy of the algorithm.

3. KNN is very **easy to implement**. There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

Disadvantages of KNN

1. Does not work well with large dataset: In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.

2. Does not work well with high dimensions: The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.

3. Need feature scaling: We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.

4. Sensitive to noisy data, missing values and outliers: KNN is sensitive to noise in the dataset. We need to manually impute missing values and remove outliers.

Usage of KNN

You can use the KNN algorithm for applications that require **high accuracy** but that **do not require a human-readable model**. The quality of the predictions depends on the distance measure. Therefore, the KNN algorithm is suitable for applications for which **sufficient domain knowledge is available**. This knowledge supports the selection of an appropriate measure.

The KNN algorithm is a type of lazy learning, where the computation for the generation of the predictions is deferred until classification. Although this method increases the costs of computation compared to other algorithms, KNN is still the better choice for applications where predictions are not requested frequently but where **accuracy is important**.

Thanks!

This is the end of K- Nearest Neighbors (KNN),

see you in the next topic.