# INTRODUCTION

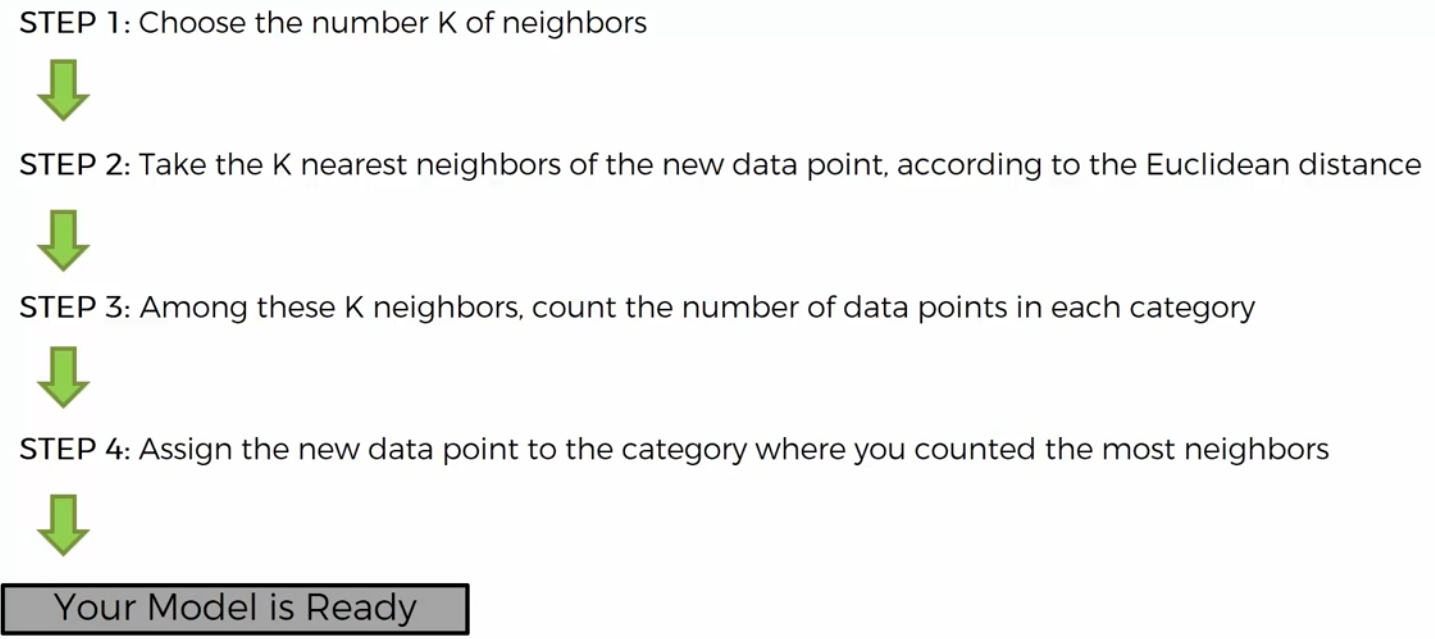
K-Nearest Neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. However, it is more widely used in classification problems in the industry.

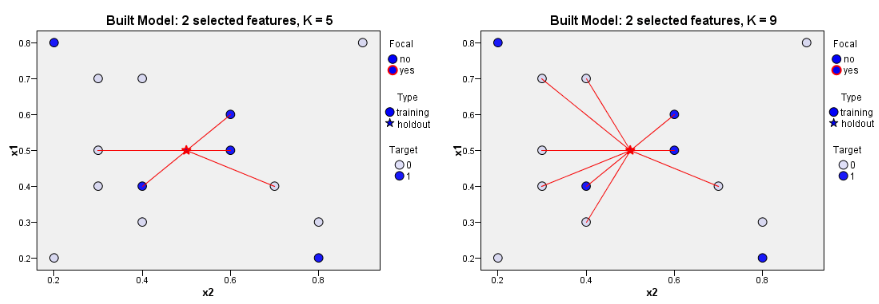
When we say a technique is non-parametric it means that it does not make any assumptions on the underlying data distribution. In other words, the model structure is determined from the data.

* **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.
* **k-NN regression**: the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

In machine learning, lazy learning is a learning method in which generalization of the training data is, in theory, delayed until a query is made to the system, as opposed to in eager learning, where the system tries to generalize the training data before receiving queries. k-NN is a type of instance-based learning, or lazy learning.

Below are the steps for KNN algorithm:





## APPLICATION

A few Applications and Examples of KNN

* Credit ratings
* Should the bank give a loan to an individual?
* classifying a potential voter to a “will vote” or “will not vote”, or to “vote Democrat” or “vote Republican”.
* More advance examples could include handwriting detection (like OCR), image recognition and even video recognition.

## PROS

* No assumptions about data — useful, for example, for nonlinear data
* Simple algorithm — to explain and understand/interpret
* High accuracy (relatively) — it is pretty high but not competitive in comparison to better supervised learning models
* Versatile — useful for classification or regression

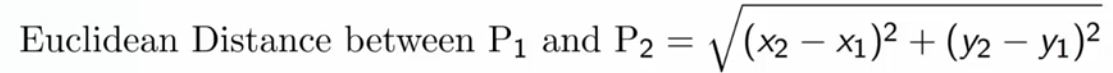
## CONS

* Computationally expensive — because the algorithm stores all of the training data
* High memory requirement
* Stores all (or almost all) of the training data
* Prediction stage might be slow (with big N)
* Sensitive to irrelevant features and the scale of the data

# DISTANCE FUNCTIONS

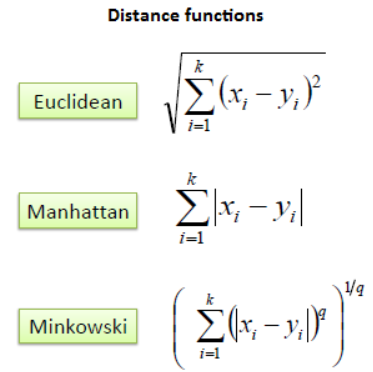
We use below distance functions in KNN for identifying nearest neighbors:

## EUCLIDEAN DISTANCE



## MANHATTAN DISTANCE

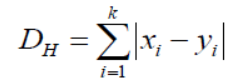
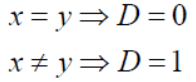
Manhattan distance is given by below formula:



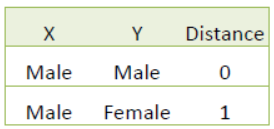
It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables, the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

## HAMMING DISTANCE

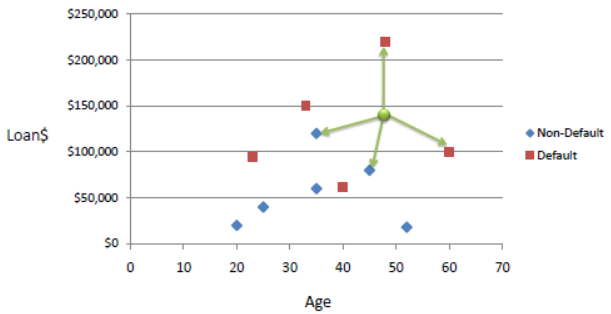
Hamming distance is given by below formula

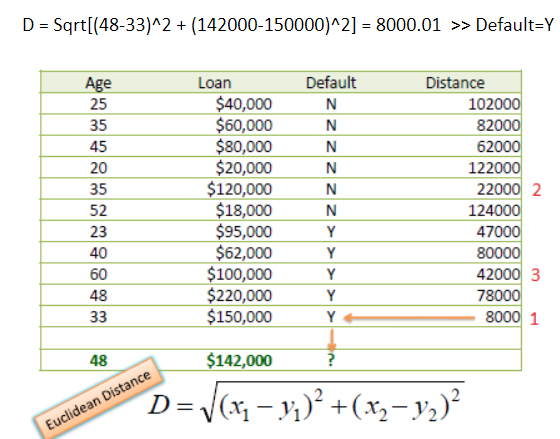
For example:



EXAMPLE: Consider the following data concerning credit default. Age and Loan are two numerical variables (predictors) and Default is the target.



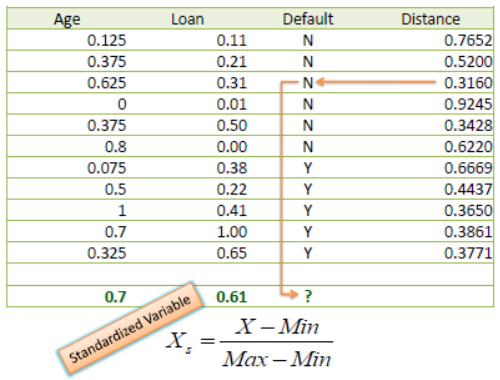
We can now use the training set to classify an unknown case (Age=48 and Loan=$142,000) using Euclidean distance. If K=1 then the nearest neighbor is the last case in the training set with Default=Y.



With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is again Default=Y.

## STANDARDIZED DISTANCE

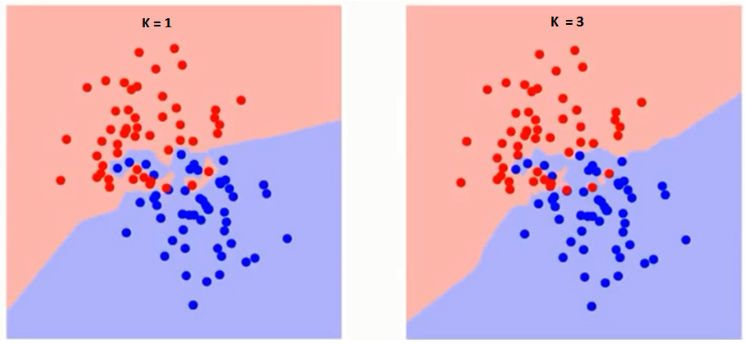
One major drawback in calculating distance measures directly from the training set is in the case where variables have different measurement scales or there is a mixture of numerical and categorical variables. For example, if one variable is based on annual income in dollars, and the other is based on age in years then income will have a much higher influence on the distance calculated. One solution is to standardize the training set as shown below.

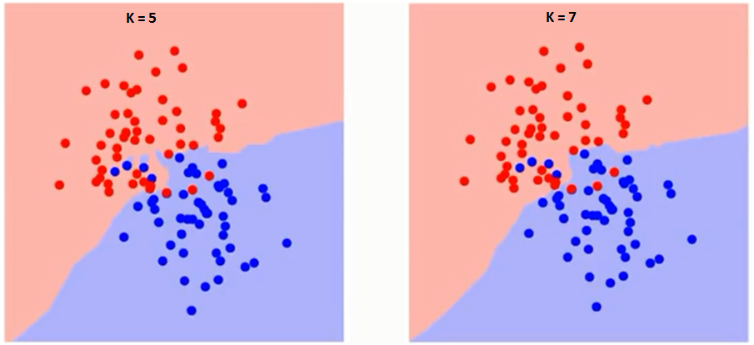


Using the standardized distance on the same training set, the unknown case returned a different neighbor which is not a good sign of robustness.

# CHOOSING K VALUE

First let us try to understand what exactly does K influence in the algorithm. Following are the different boundaries separating the two classes with different values of K.

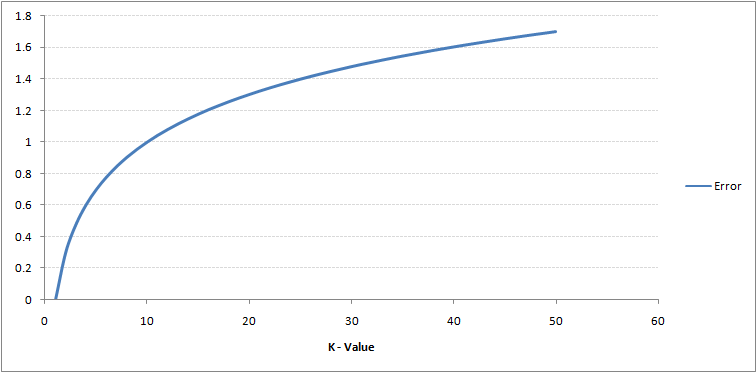




If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority.

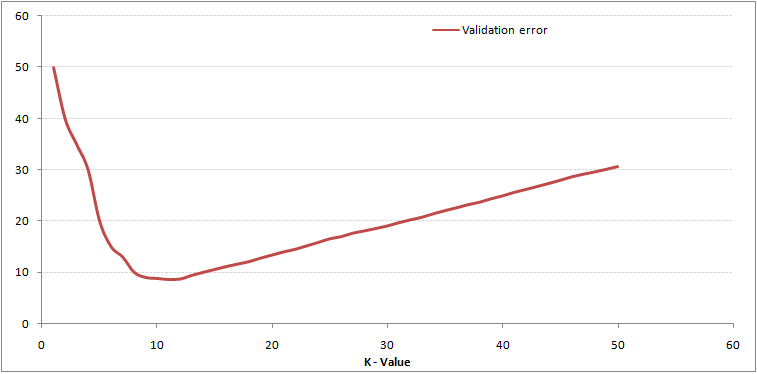
The training error rate and the validation error rate are two parameters we need to access on different K-value.

Following is the curve for the training error rate with varying value of K:



As you can see, the error rate at K=1 is always zero for the training sample. This is because the closest point to any training data point is itself. Hence the prediction is always accurate with K=1. If validation error curve would have been similar, our choice of K would have been 1.

Following is the validation error curve with varying value of K:



This makes the story clearer. At K=1, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minimal. After the minima point, it then increases with increasing K. To get the optimal value of K, you can segregate the training and validation from the initial dataset. Now plot the validation error curve to get the optimal value of K. This value of K should be used for all predictions.

# IMPORTANT POINTS

* Not well with categorical data
* Not good with high dimensional data
* Scale of the features matters a lot so we have to standardize the features. Since we calculate distance between data points so larger values will have an impact on the distance.
* Use elbow method to choose k (plot error rate for k in range 1 to 40).