K-NEAREST NEIGHBORS CLASSIFICATION ALGORITHM IMPLEMENTATION IN PYTHON

**What is KNN?**

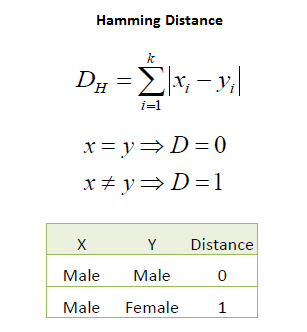
K nearest neighbors is a simple algorithm in supervised machine learning algorithms that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition.

Algorithm

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.



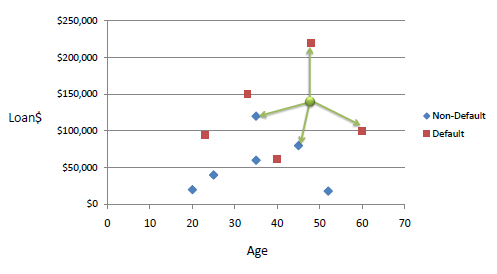
It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical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



Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.

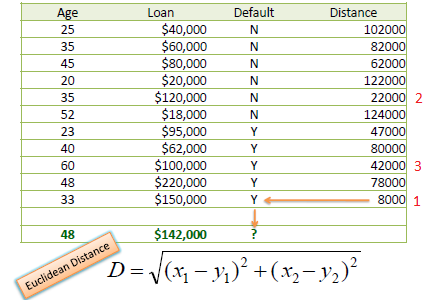
**Example**

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.

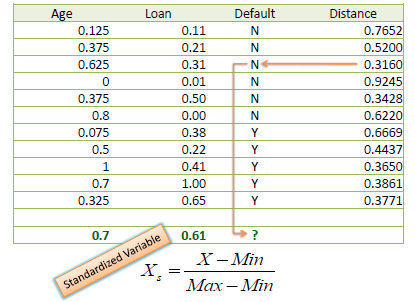
D = Sqrt[(48-33)^2 + (142000-150000)^2] = 8000.01  >> Default=Y



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.

**Pros**

1. It is extremely easy to implement
2. As said earlier, it is [lazy learning](https://en.wikipedia.org/wiki/Lazy_learning) algorithm and therefore requires no training prior to making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g SVM, [linear regression](http://stackabuse.com/linear-regression-in-python-with-scikit-learn/), etc.
3. Since the algorithm requires no training before making predictions, new data can be added seamlessly.
4. 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.)

**Cons**

1. 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 distance in each dimension.
2. The KNN algorithm has a high prediction cost for large datasets. This is because in large datasets the cost of calculating distance between new point and each existing point becomes higher.
3. Finally, the KNN algorithm doesn't work well with categorical features since it is difficult to find the distance between dimensions with categorical features.

Implementation using python – Find accuracy for given dataset

#### Step 1:

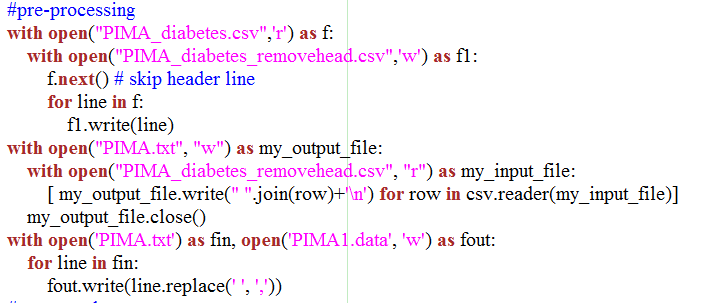
The Dataset - We are going to use the PIMA\_diabetes (csv format) data set for our KNN example. The dataset consists of nine attributes: Pregnancies, Glucose, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age and Outcome. These are the attributes of specific types of diabetes plan.

**Step2:**

Importing the Dataset – Import csv lib to dataset read and write operation.

**Step3:**

Preprocessing – Open and read csv file -> remove header (column name) from file to make all the data as a same format (int) -> change csv file format to .data format to access data easily.



**Step4:**

Train Test Split- To avoid over-fitting, we will divide our dataset into training and test splits, here we use,

Split = 0.67

Result: Train set: 505, Test set: 262

**Step5:**

Feature Scaling - Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated.

Range should base on dataset column count.

Here we use split and range,

range(8)

**Step6:**

Training and Predictions (Load dataset) - Extremely straight forward to train the KNN algorithm and make predictions with it. There is no ideal value for K and it is selected after testing and evaluation, however to start out, 5 seems to be the most commonly used value for KNN algorithm.

Here we use K=3 as default.

# generate predictions

predictions=[]

k = 3

**Step7:**

Get Neighbors – Get neighbors distance from each data in the dataset. It includes 3 parameters,

* Training set
* Test instance
* K value

### def getNeighbors(trainingSet, testInstance, k)

Within getneighbors function we should find out distance from each data

Dist =euclideanDistance(testInstance, trainingSet[x], length)

Finally it returns neighbors distance for whole dataset.

### Step 8:

Get Response of neighbors – It used to display the sortedVotes – Gives predicts and actual value of each data.

getResponse(neighbors)

**Step 9:**

Display both predicted and actual value using Pythonlist- var name as predictions.

#get result and add result to predictions list.

result = getResponse(neighbors)

predictions.append(result)

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))

**Step 10:**

Get Accuracy- Based on predictions and testset it will give come % as a result. If testset data and prediction data is same, then will get Accuracy as %.

**Step 11:**

### Comparing Error Rate with the K Value - In the training and prediction section we said that there is no way to know beforehand which value of K that yields the best results in the first go. We randomly chose 5 as the K value and it just happen to result in 100% accuracy. K value may be 1 to 40.

Result:

If K=3 => Accuracy – 78.678%

If K=4 => Accuracy - 56.8702 %

If K=5 => Accuracy - 58.5551 %

**Conclusion**

KNN is a simple yet powerful classification algorithm. It requires no training for making predictions, which is typically one of the most difficult parts of a machine learning algorithm.

The KNN algorithm has been widely used to find document similarity and pattern recognition. It has also been employed for developing recommender systems and for dimensionality reduction and pre-processing steps for computer vision, particularly face recognition tasks.