K-Nearest Neighbors Classification

* KNN is a simple supervised classification algorithm we can use to assign a class to new data point.
* KNN algorithm is **based on feature similarity** approach.
* Used for both classification and regression problems.
* Used in handwriting detection, image/video recognition and political science.
* KNN is **non-parametric and lazy learning** algorithm.
* Non-parametric means there is no assumptions for underlying data distribution.

This means dataset do not follow mathematical theoretical assumptions.

* **Lazy algorithm** means it does not need any training data points for model generation. All training data is used in testing phase. This makes training faster and it will take much time and much memory for testing phase.
* Algorithm does not build a model until the time that a prediction is required.
* It is lazy because it only does work at the last second. It is also called as **Instance based learners** because lazy learners stores the training points or instances, and all learning is based on instances.
* **“Show me who your friends are, I will tell you who you are” - It classifies a data point based on how its neighbors are classified.**

# Eager learners vs Lazy Learners

Eager learners mean when given training points will construct a generalized model before performing prediction on given new points to classify.

# Pros:

* KNN is a faster algorithm, since it doesn’t required training phase. New data can be added seamlessly.
* It required only two parameters to implement KNN, value of K and the distance function.

# Cons:

* KNN won’t work well for high dimension data, because with large number of dimensions, it is difficult for the algorithm to calculate the distance in each column.
* KNN has a high prediction cost for large data sets. As cost to calculate distance for large data set is high.
* KNN won’t work well for categorical features since it is difficult to find the distance between dimensions with categorical features.

# How Does KNN works:

Suppose P1 is the data point for which we need to predict a label,

Then

1. Calculate the distance: Between P1 and all the data points by using

* Euclidean distance
* Hamming distance
* Manhattan distance
* Minkowski distance

**If data is continuous**🡪 Euclidean distance, Manhattan distance

**If data is categorical**🡪 Hamming distance

**Euclidean Distance**:

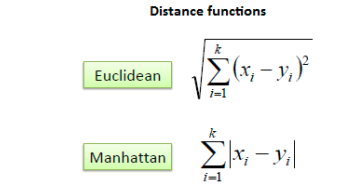
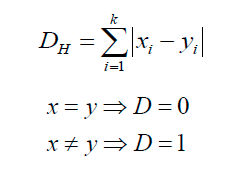
Square root of the sum of the squared differences between a new point (x) and an existing point (y).

**Manhattan Distance:**

Distance between the real vectors using their sum of their absolute difference.

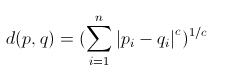
**Hamming Distance (DH):**

It is used for categorical values. If the value of x and y are same, the distance will be equal to 0, otherwise it is 1.

**Minkowski distance:**

Minkowski distance is a metric in a normed vector space which can be considered as a generalization of both the Euclidean and Manhattan distance.



If **c=1** it is equal to **Manhattan distance**

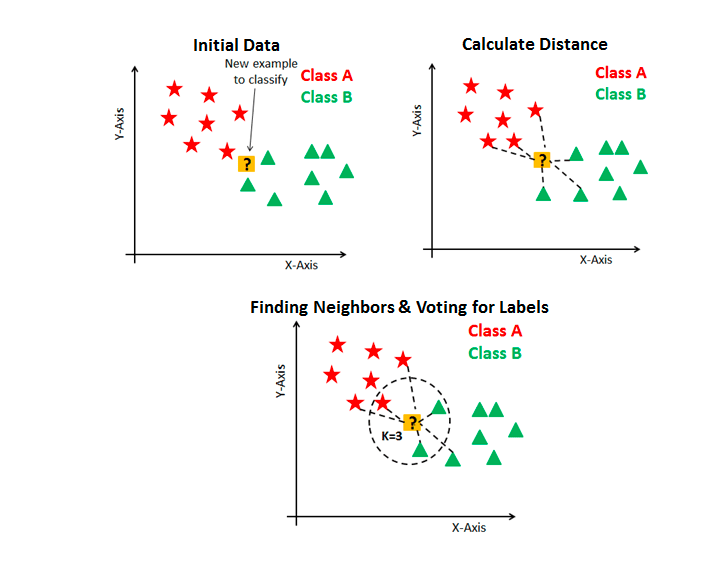
If **c=2** it is equal to **Euclidean distance**

1. Find closet Neighbors (if k=3 it will take 3 data points which are very near to it)

**K factor:**

1. **Sqrt(n),** where nis the total number of data points
2. **Odd value of K** is selected to avoid confusion b/w two classes of data
3. **Elbow curve,** is used to get the optimal value of k. this curve is plotted to calculate the root mean squared error for the model for different k.
4. Vote for labels

* For classification, label is decided for the point based on voting of the results.
* For regression, label is decided based on the average of all the results



Coding Part:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (20.0,10.0)

%matplotlib inline

from sklearn.neighbors import KNeighborsClassifier,KNeighborsRegressor

from sklearn.metrics import accuracy\_score,mean\_squared\_error,classification\_report

#from sklearn.cross\_validation import train\_test\_split

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

data=load\_iris()

print("\nData Type: ", type(data))

print("\nAttributes: ",data.feature\_names)

print("\nlabels: ",data.target\_names)

print("\nType of Data is: {0},\tShape of the X Data{1}".format(type(data.data),data.data.shape))

print("\nType of Target: {0}, \tShape of the Y Data{1}".format(type(data.target),data.data.shape))

X = data.data

Y = data.target

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,random\_state = 0, test\_size=0.3)

print("\nShape of the training, testing Data:{0}, {1}".format(X\_train.shape,X\_test.shape))

Normalization = StandardScaler()

Normalization.fit(X\_train)

X\_train = Normalization.transform(X\_train)

X\_test = Normalization.transform(X\_test)

ERROR = []

Accuracy = []

for i in range(1,40):

KNN = KNeighborsClassifier(n\_neighbors=i)

KNN.fit(X\_train, Y\_train)

Y\_pred = KNN.predict(X\_test)

ERROR.append(mean\_squared\_error(Y\_pred, Y\_test))

Accuracy.append(accuracy\_score(Y\_pred, Y\_test))

curve = pd.DataFrame(ERROR)

plt.plot(curve)

plt.legend("ELBOW CURVE")

plt.xlabel("Value of K for KNN")

plt.ylabel("Mean Squared ERROR")

plt.title("Graph for Optimal Value for K")

# we go with k=7

KNN = KNeighborsClassifier(n\_neighbors=7)

KNN.fit(X\_train, Y\_train)

Y\_pred = KNN.predict(X\_test)

print("\naccuracy of the model: ",accuracy\_score(Y\_pred, Y\_test)\*100)

print("\n\n",classification\_report(Y\_pred,Y\_test))

# Output:

Data Type: <class 'sklearn.utils.Bunch'>

Attributes: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

labels: ['setosa' 'versicolor' 'virginica']

Type of Data is: <class 'numpy.ndarray'>, Shape of the X Data(150, 4)

Type of Target: <class 'numpy.ndarray'>, Shape of the Y Data(150, 4)

Shape of the training, testing Data:(105, 4), (45, 4)

accuracy of the model: 97.77777777777777

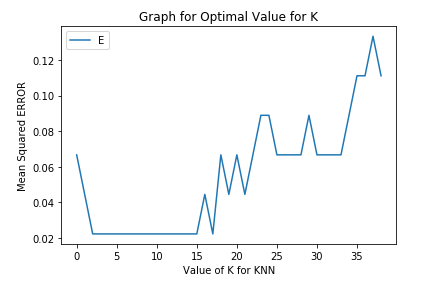
precision recall f1-score support

0 1.00 1.00 1.00 16

1 0.94 1.00 0.97 17

2 1.00 0.92 0.96 12

avg / total 0.98 0.98 0.98 45



#Method to get the optimal value of K

For deciding the value of k, plotting the elbow curve every time is be a difficult process**.**You can simply use GridSearch to find the best value.

from sklearn.model\_selection import GridSearchCV

params = {'n\_neighbors': [2,3,4,5,6,7,8,9]}

knn = KNeighborsRegressor()

model = GridSearchCV(knn, params, cv=5)

model.fit(X\_train,Y\_train)

model.best\_params\_

output: {'n\_neighbors': 5}