Introduction to K-Nearest Neighbour (KNN)

K-NN is a non-parametric supervised learning technique in which we try to classify the data point to a given category with the help of training set. In simple words, it captures information of all training cases and classifies new cases based on a similarity.

Predictions are made for a new instance (x) by searching through the entire training set for the K most similar cases (neighbours) and summarizing the output variable for those K cases. In classification this is the mode (or most common) class value.

Let's first start by establishing some definitions and notations. We will use x to denote a *feature* (aka. predictor, attribute) and y to denote the *target* (aka. label, class) we are trying to predict.

KNN falls in the **supervised learning** family of algorithms. Informally, this means that we are given a labelled dataset consiting of training observations (x, y) and would like to capture the relationship between x and y. More formally, our goal is to learn a function $h : X \rightarrow Y$ so that given an unseen observation x, $h(x)$ can confidently predict the corresponding output y.

The KNN classifier is also a **non parametric** and **instance-based** learning algorithm.

- **Non-parametric** means it makes no explicit assumptions about the functional form of h, avoiding the dangers of mismodeling the underlying distribution of the data. For example, suppose our data is highly non-Gaussian but the learning model we choose assumes a Gaussian form. In that case, our algorithm would make extremely poor predictions.
- **Instance-based** learning means that our algorithm doesn't explicitly learn a model. Instead, it chooses to memorize the training instances which are subsequently used as "knowledge" for the prediction phase. Concretely, this means that only when a query to our database is made (i.e. when we ask it to predict a label given an input), will the algorithm use the training instances to spit out an answer.

> KNN is non-parametric, instance-based and used in a supervised learning setting.

It is worth noting that the minimal training phase of KNN comes both at a *memory cost*, since we must store a potentially huge data set, as well as a *computational cost* during test time since classifying a given observation requires a run down of the whole data set. Practically speaking, this is undesirable since we usually want fast responses.

> Minimal training but expensive testing.

How KNN algorithm works

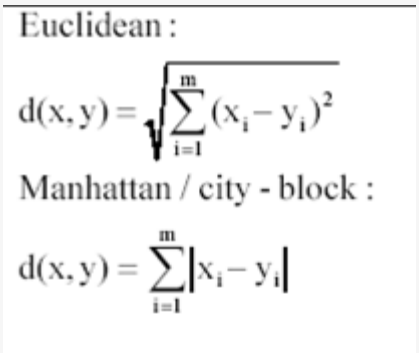
Suppose we have height, weight and T-shirt size of some customers and we need to predict the T-shirt size of a new customer given only height and weight information we have. Data including height, weight and T-shirt size information is shown below -

Height (in cms)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M
158	63	M

160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

Step 1: Calculate Similarity based on distance function

There are many distance functions but Euclidean is the most used measure.



Distance Functions

The idea to use distance measure is to find the distance (similarity) between new sample and training cases and then finds the k-

closest customers to new customer in terms of height and weight.

New customer named 'Monica' has height 161cm and weight 61kg.

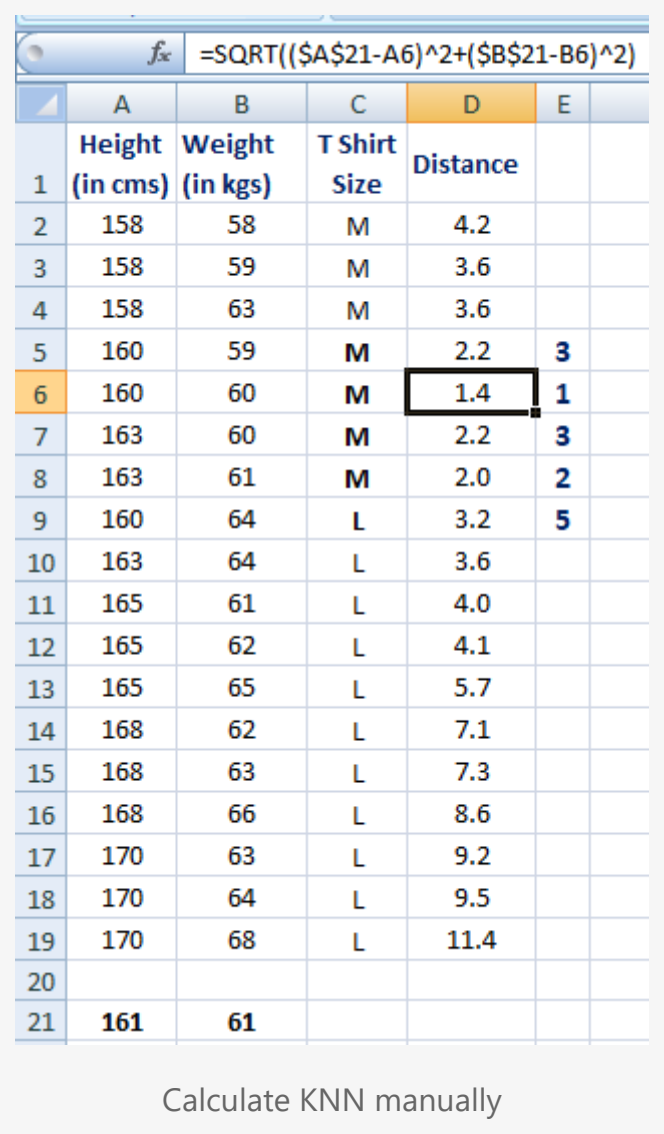
Euclidean distance between first observation and new observation (Monica) is as follows -

```
=SQRT((161-158)^2+(61-58)^2)
```

Similarly, we will calculate distance of all the training cases with new case and calculates the rank in terms of distance. The smallest distance value will be ranked 1 and considered as nearest neighbor.

Step 2: Find K-Nearest Neighbours

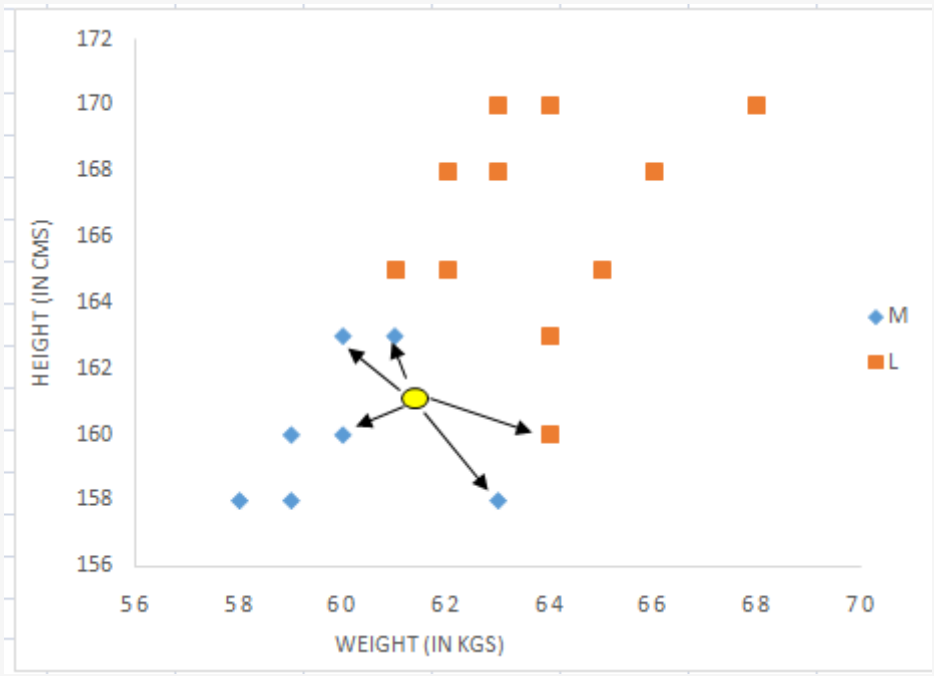
Let k be 5. Then the algorithm searches for the 5 customers closest to Monica, i.e. most similar to Monica in terms of attributes, and see what categories those 5 customers were in. If 4 of them had ‘Medium T shirt sizes’ and 1 had ‘Large T shirt size’ then your best guess for Monica is ‘Medium T shirt. See the calculation shown in the snapshot below -



	A	B	C	D	E
1	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
2	158	58	M	4.2	
3	158	59	M	3.6	
4	158	63	M	3.6	
5	160	59	**M**	2.2	**3**
6	160	60	**M**	1.4	**1**
7	163	60	**M**	2.2	**3**
8	163	61	**M**	2.0	**2**
9	160	64	**L**	3.2	**5**
10	163	64	L	3.6	
11	165	61	L	4.0	
12	165	62	L	4.1	
13	165	65	L	5.7	
14	168	62	L	7.1	
15	168	63	L	7.3	
16	168	66	L	8.6	
17	170	63	L	9.2	
18	170	64	L	9.5	
19	170	68	L	11.4	
20					
21	**161**	**61**			

Calculate KNN manually

In the graph below, binary dependent variable (T-shirt size) is displayed in blue and orange color. 'Medium T-shirt size' is in blue color and 'Large T-shirt size' in orange color. New customer information is exhibited in yellow circle. Four blue highlighted data points and one orange highlighted data point are close to yellow circle. so the prediction for the new case is blue highlighted data point which is Medium T-shirt size.



KNN: Visual Representation

Assumptions of KNN

1. Standardization

When independent variables in training data are measured in different units, it is important to standardize variables before calculating distance. For example, if one variable is based on height in cms, and the other is based on weight in kgs then height will influence more on the distance calculation. In order to make them comparable we need to standardize them which can be done by any of the following methods:

$$Xs = \frac{X - mean}{s.d.}$$

$$Xs = \frac{X - mean}{max - min}$$

$$Xs = \frac{X - min}{\max - min}$$

Standardization

After standardization, 5th closest value got changed as height was dominating earlier before standardization. Hence, it is important to standardize predictors before running K-nearest neighbour algorithm.

	A	B	C	D	E
1	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
2	-1.39	-1.64	M	1.3	
3	-1.39	-1.27	M	1.0	
4	-1.39	0.25	M	1.0	
5	-0.92	-1.27	**M**	0.8	**4**
6	-0.92	-0.89	**M**	0.4	**1**
7	-0.23	-0.89	**M**	0.6	**3**
8	-0.23	-0.51	**M**	0.5	**2**
9	-0.92	0.63	L	1.2	
10	-0.23	0.63	L	1.2	
11	0.23	-0.51	**L**	0.9	**5**
12	0.23	-0.13	L	1.0	
13	0.23	1.01	L	1.8	
14	0.92	-0.13	L	1.7	
15	0.92	0.25	L	1.8	
16	0.92	1.39	L	2.5	
17	1.39	0.25	L	2.2	
18	1.39	0.63	L	2.4	
19	1.39	2.15	L	3.4	
20					
21	**-0.7**	**-0.5**			

2. Outlier

Low k-value is sensitive to outliers and a higher K-value is more resilient to outliers as it considers more voters to decide prediction.

Why KNN is non-parametric?

Non-parametric means not making any assumptions on the underlying data distribution. Non-parametric methods do not have fixed numbers of parameters in the model. Similarly in KNN, model parameters actually grows with the training data set - you can imagine each training case as a "parameter" in the model.

KNN vs. K-means

Many people get confused between these two statistical techniques- K-means and K-nearest neighbour. See some of the difference below -

1. K-means is an unsupervised learning technique (no dependent variable) whereas KNN is a supervised learning algorithm (dependent variable exists)
2. K-means is a clustering technique which tries to split data points into K-clusters such that the points in each cluster tend to be near each other whereas K-nearest neighbour tries to determine the classification of a point, combines the classification of the K nearest points

Pros and Cons of KNN

Pros

1. Easy to understand
2. No assumptions about data
3. Can be applied to both classification and regression
4. Works easily on multi-class problems

Cons

1. Memory Intensive / Computationally expensive
2. Sensitive to scale of data
3. Not work well on rare event (skewed) target variable
4. Struggle when high number of independent variables

For any given problem, a small value of k will lead to a large variance in predictions. Alternatively, setting k to a large value may lead to a large model bias.

How to handle categorical variables in KNN?

Create dummy variables out of a categorical variable and include them instead of original categorical variable. Unlike regression, create k dummies instead of (k-1). For example, a categorical variable named "Department" has 5 unique levels / categories. So we will create 5 dummy variables. Each dummy variable has 1 against its department and else 0.

How to find best K value?

Cross-validation is a smart way to find out the optimal K value. It estimates the validation error rate by holding out a subset of the training set from the model building process.

Cross-validation (let's say 10-fold validation) involves randomly dividing the training set into 10 groups, or folds, of approximately equal size. 90% data is used to train the model and remaining 10% to validate it. The misclassification rate is then computed on the 10% validation data. This procedure repeats 10 times. Different group of observations are treated as a validation set each of the 10 times. It results to 10 estimates of the validation error which are then averaged out.

Parameter Tuning with Cross Validation

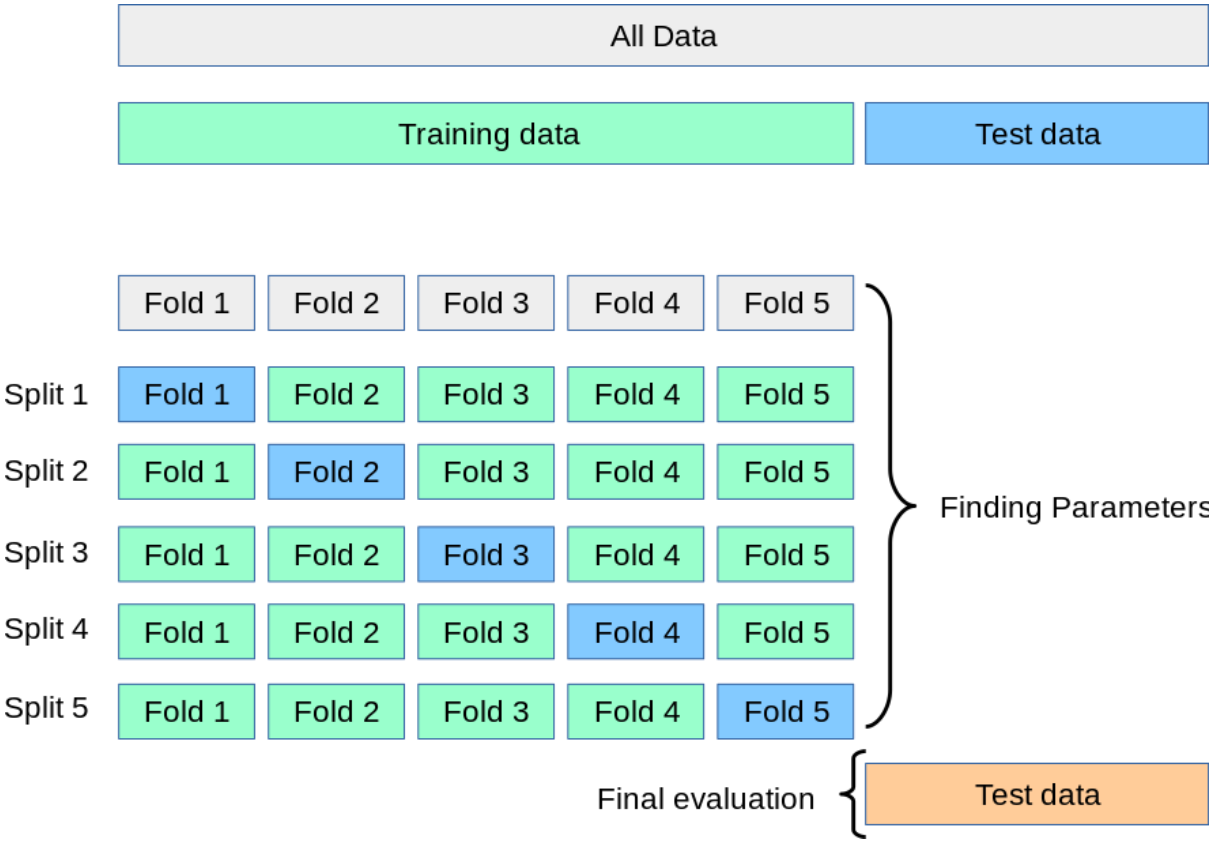
In this section, we'll explore a method that can be used to tune the hyperparameter K.

Obviously, the best K is the one that corresponds to the lowest test error rate, so let's suppose we carry out repeated measurements of the test error for different values of K. Inadvertently, what we are

doing is using the test set as a training set! This means that we are underestimating the true error rate since our model has been forced to fit the test set in the best possible manner.
Our model is then incapable of generalizing to newer observations, a process known as overfitting. Hence, touching the test set is out of the question and must only be done at the very end of our pipeline.

Using the test set for hyperparameter tuning can lead to overfitting.

An alternative and smarter approach involves estimating the test error rate by holding out a subset of the `training set` from the fitting process. This subset, called the `validation set`, can be used to select the appropriate level of flexibility of our algorithm! There are different validation approaches that are used in practice, and we will be exploring one of the more popular ones called **k-fold cross validation**.



K-Fold Cross Validation

As seen in the image, k-fold cross validation (*the k is totally unrelated to K*) involves randomly dividing the training set into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining $k-1$ folds. The misclassification rate is then computed on the observations in the held-out fold. This procedure is repeated k times; each time, a different group of observations is treated as a validation set. This process results in k estimates of the test error which are then averaged out.

Cross-validation can be used to estimate the test error associated with a learning method in order to evaluate its performance, or to select the appropriate level of flexibility.

If that is a bit overwhelming for you, don't worry about it. We're gonna make it clearer by performing a 10-fold cross validation on our dataset using a generated list of odd K's ranging from 1 to 50.

```
# creating odd list of K for KNN
neighbors = list(range(1, 50, 2))

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10,
scoring='accuracy')
    cv_scores.append(scores.mean())
```

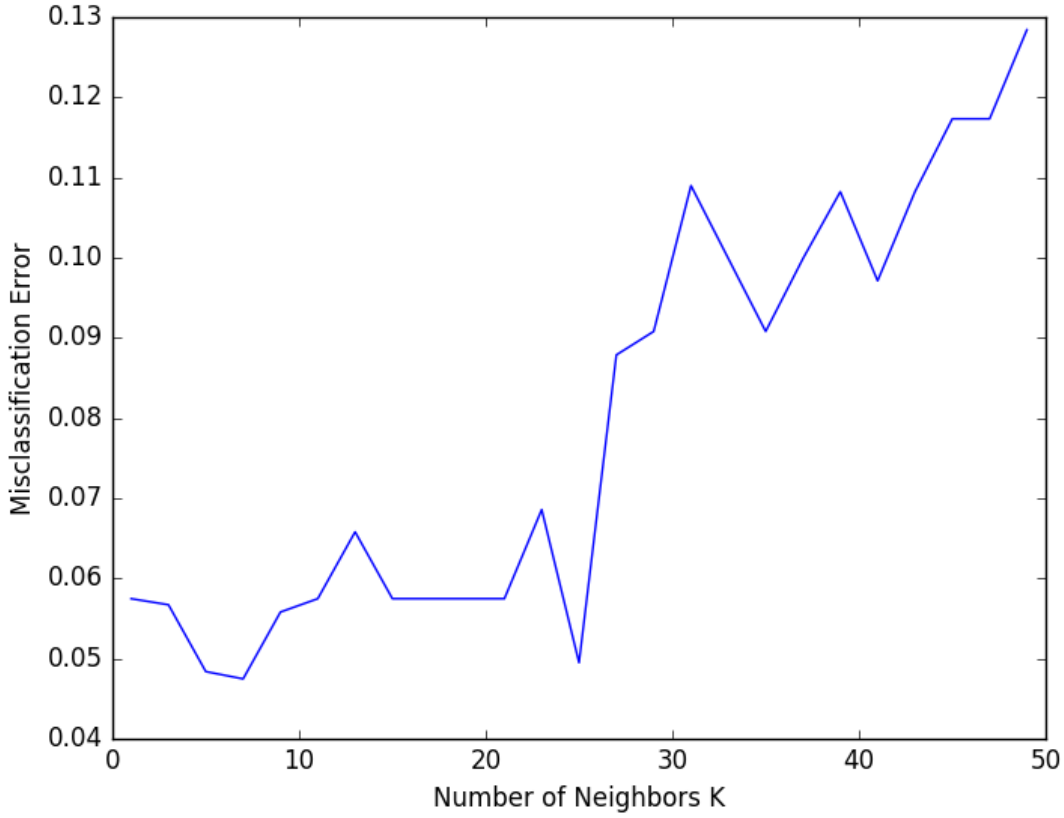
Again, scikit-learn comes in handy with its `cross_val_score` method. We specifiy that we are performing 10 folds with the `cv = 10` parameter and that our scoring metric should be `accuracy` since we are in a classification setting.

Finally, we plot the misclassification error versus K.

```
# changing to misclassification error
mse = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[mse.index(min(mse))]
print("The optimal number of neighbors is {}".format(optimal_k))

# plot misclassification error vs k
plt.plot(neighbors, mse)
plt.xlabel("Number of Neighbors K")
plt.ylabel("Misclassification Error")
plt.show()
```



10-fold cross validation tells us that K=7 results in the lowest validation error.

Why is KNN called a Lazy Learner?

Notes about the nearest neighbour classifier: - It is often called “lazy learning”. It does not build any “model”. Instead, all the work (i.e., finding the nearest neighbours) is done at prediction time.

KNN Code Snippet:

Users > satyam > Desktop > ML_Algos > KNNclassification.py

```python
# KNN Algorithm for Classification | Author: Dhrub Satyam
# Creating our own features
# Assigning features and label variables
# First Feature
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
# Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

# Label or target varible
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)

# converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)

#combinig weather and temp into single listof tuples
features=list(zip(weather_encoded,temp_encoded))

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(features,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print(predicted)
```