KNN

# Introductions to KNN

* Is a classification algorithm.
* The goal is to use existing data points to classify a new data point based off its nearest neighbors of data point.
* The training algorithm simply stores data.
* The prediction algorithm works by: calculating the distance of all your data entries from your new data point, x; sorting those by distance from x, labeling x as the class majority of its k(specified number of neighbors) neighbors belong to.
* Choosing a k affects the accuracy: a very low k will have many noises and a very high k will be cleaner but probably have a higher level of mislabeling.
* KNN’s advantage are its simple, easy to train (just store the training data), works with any number of classes, easy to add more data, has only 2 parameters – k and distance metric being used.
* Distance metric is the distance between your new testing point and your old training points.
* The disadvantages of KNN are: its high cost as data increases, unsuitability for categorical data and poor at high dimensional data (data with a lot of features).
* A common data scientist interview is to be given anonymized data without the full context and being asked to do a classification for it without knowing what the columns represent.

# KNN with python

## Preprocessing

* You can pass on the argument ‘index\_col=o’ to set the first column as the index if there is need.
* A lot of times the scale of some features might be too large which is not good for your model. To deal with this, we can use the sklearn scaler called StandardScaler to scale all the features close to each other by:
* From sklearn.preprocessing import StandardScaler
* Create an instance of the StandardScaler e.g., S = StandardScaler()
* Fit the features into the scaler with s.fit(df.drop([‘Target’], axis=1)). Ensure not to fit the target into the scaler.
* Transform the features into scale by creating a scaled instance of it e.g., scaled = s.transform(df.drop[‘Target’] , axis=1)
* Set the scaled features to a new data frame to be used as the features going forward e.g., df\_features = pd.DataFrame(scaled, columns=df.columns[:-1]) :1 is used if the target is the last column or other suitable slicing is used for getting the columns.
* Note: you can use s.fit\_transform(df.drop(['target’], axis=1)) to do steps 2 and 3 at once instead of over two steps.

## Training

* Split the data into training and testing version
* importing KneighborsClassifier from sklearn.neighbors
* instantiate the KneighboursClassifier e.g., knn = Kneighbours(). Pass in the k (n\_neighbors) argument which is default to 5.
* Train with knn.fit(X\_train, y\_train)
* Predict by naming a prediction variable and calling the knn.predict e.g., predictions = knn.predict(X\_test)
* You can check the predictions afterwards.

## Evaluation

* From sklearn.metrics, import classification report and confusion matrix.

# Elbow method of choosing optimum k value and retraining based on it

* You can use a method called elbow method to choose the most appropriate k value by:
* Create an empty list for error rates that will be used to store the error rate of different values of k for your model.
* Create a function that instantiates the Kneighbors classifier, trains it (fit), predict with it in a predict variable and computes and appends the errors (np.mean(predictions != y\_test)) in the list while testing various k values within given range.
* Plot the error rate on the y axis against the k values
* Visually inspect and pick the optimum error rate’s corresponding k value.
* Use the optimum k value to plot run your model again.