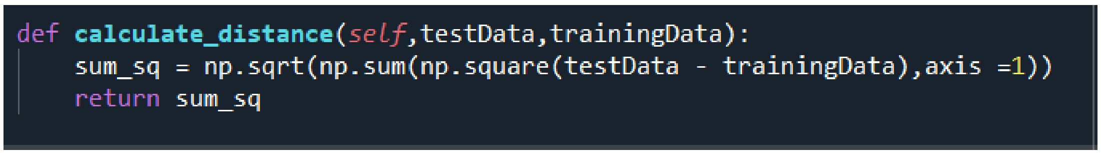
Practical Machine Learning

Part 1 A– Distance Weighted Nearest Neighbour Algorithm for Regression

The following is the implementation of the KNN Algorithm

1.Calculate_distances : The distance is calculated using standard Euclidean distance. It accepts the each row of the test data(single query instance) and subtracted from the all feature training data.and calculate the distance by using following formula

sum_sq = np.sqrt(np.sum(np.square(testData - trainingData),axis =1))



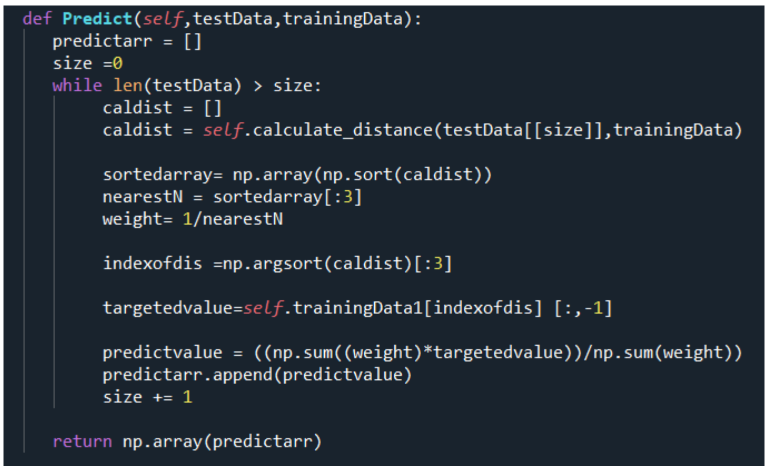
```
def calculate_distance(self,testData,trainingData):
    sum_sq = np.sqrt(np.sum(np.square(testData - trainingData),axis =1))
    return sum_sq
```

2.Predict :

1. This method accepts the each row of the test data(single query instance) and all the feature data.
2. First it calculate the distance between single query instance and all feature data by calling calculate_distance function
3. the output of distance sorted in ascending order and select 3 mini distance and calculate the inverse distance weight by dividing 1 from all 3 distances.
4. Then select the label from feature training data of that 3 distances
5. calculate the value by using below formula and then it is stored in the list and this process will run for all the test data

   predictvalue = ((np.sum((weight)*targetedvalue))/np.sum(weight))

6. return the predicted regression value for each instances.



```
def Predict(self,testData,trainingData):
    predictarr = []
    size =0
    while len(testData) > size:
        caldist = []
        caldist = self.calculate_distance(testData[[size]],trainingData)

        sortedarray= np.array(np.sort(caldist))
        nearestN = sortedarray[:3]
        weight= 1/nearestN

        indexofdis =np.argsort(caldist)[:3]

        targetedvalue=self.trainingData1[indexofdis] [:,-1]

        predictvalue = ((np.sum((weight)*targetedvalue))/np.sum(weight))
        predictarr.append(predictvalue)
        size += 1

    return np.array(predictarr)
```

3. calculate_r2 :

1.This function will accepts the targeted regression value for all instances and predicted values.

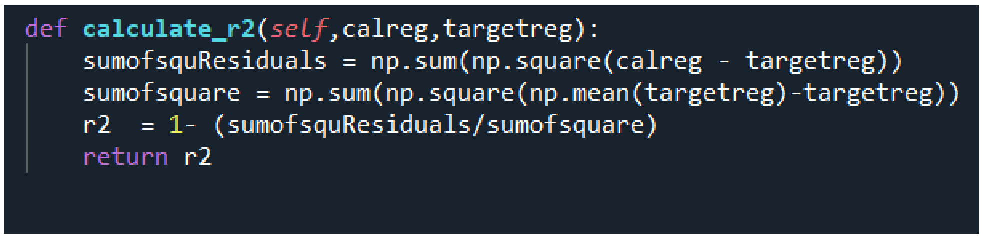
2.calculate sum of squared residuals by using following formula.

sumofsquResiduals = np.sum(np.square(calreg - targetreg))

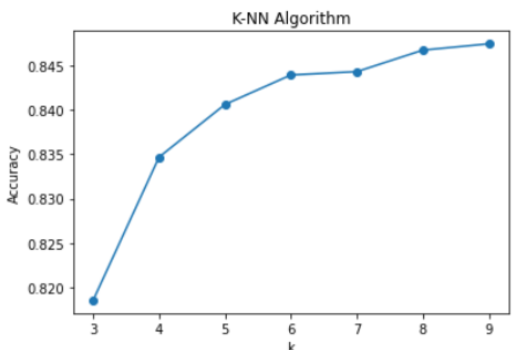
3.calculate sum of square using following formula.

sumofsquare = np.sum(np.square(np.mean(targetreg)-targetreg))

4. then divide 1 by subtraction of sumofsquResiduals and sumofsquare and calculate r2



```python
def calculate_r2(self,calreg,targetreg):
    sumofsquResiduals = np.sum(np.square(calreg - targetreg))
    sumofsquare = np.sum(np.square(np.mean(targetreg)-targetreg))
    r2  = 1- (sumofsquResiduals/sumofsquare)
    return r2
```



r2 = 0.8185732982178427

Part 1(B) – Parameters/Techniques Impact Model Performance

Knn Algoritham : This algorithm is widely used for classification and predictive problem. It is measuring the distance between training data and test data to predict the output

"By default, a k-NN algorithm will weigh the contribution of each feature equally when using standard Euclidean distance"

The Above statement states that when we use standard Euclidean Distance then it treats all dimensions equally.

It gives importance of the all the feature data but it is sensitive to extreme differences in single attribute.

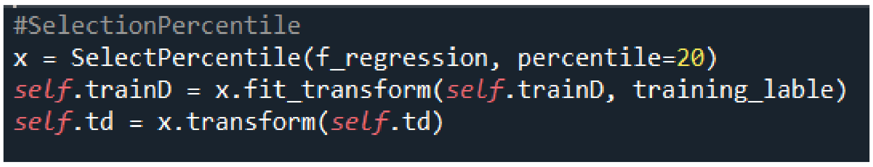
For example, If we have 1000 attributes and all are range into training and testing points but one attribute which is gives us one big number compare to all the other points, so because we are considering all the feature data, that **one irrelevant data** decrease the performance of KNN algorithm.

To overcome this issue we can implement following feature selection techniques.

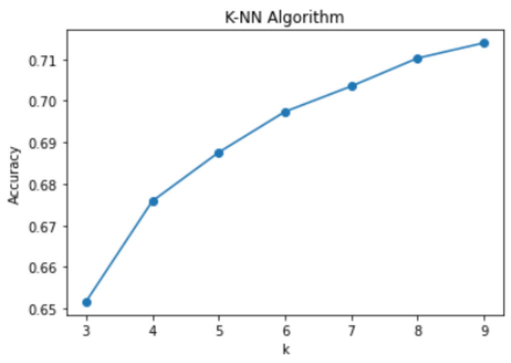
- **VarianceThreshold** (Removing features with low variance)- This method will remove all the zero variance features it means that have same values.
- **Univariate feature selection** : selecting best feature based on univariate statistical best
  - **Selectkbest** : removes k highest scoring feature
  - **Selectpercentile** : removes all but only user specified percentage of the feature
  - **GenericUnivariateSelect** : select the features with configurable strategy.Hyper-parameter search estimator select best univariate selection strategy.

Below method is implemented in my KNN algorithm

Selectpercentile :



```
#SelectionPercentile
x = SelectPercentile(f_regression, percentile=20)
self.trainD = x.fit_transform(self.trainD, training_lable)
self.td = x.transform(self.td)
```

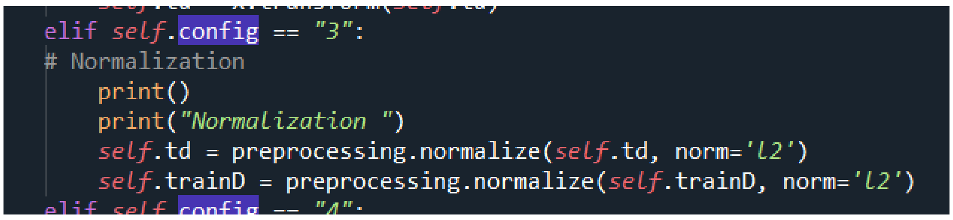


r2 = 0.713923568066267

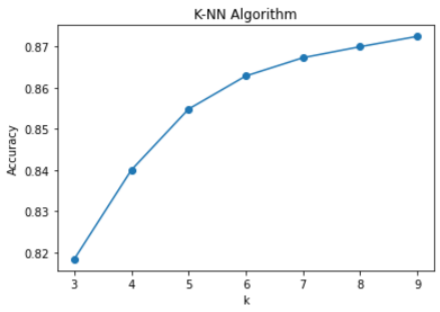
The techniques/parameters that can potentially impact the performance of a kNN.

1. **Normalization : we can use normalization to normalize data and that will gives you the value form that range of 0 – 1,using the sklearn use the following method to normalize the data**

Normalize_testData = preprocessing.normalize(testData, norm='l2')



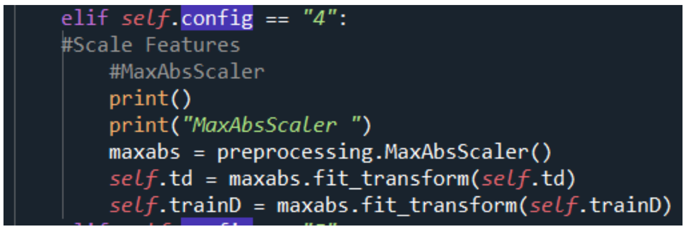
```
elif self.config == "3":
# Normalization
    print()
    print("Normalization ")
    self.td = preprocessing.normalize(self.td, norm='l2')
    self.trainD = preprocessing.normalize(self.trainD, norm='l2')
```



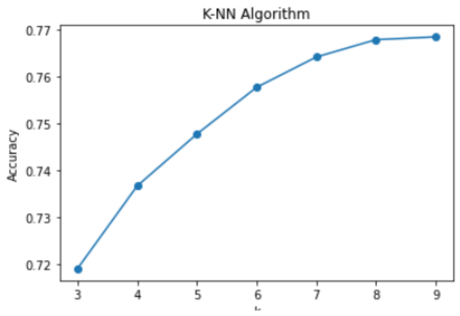
r2 = 0.8724952254511451

2.Scaling features to range: This features lie between minimum and maximum value between 0 and 1 .It can use using MaxAbsScaler,StandardScaler, MinMaxScaler

MaxAbsScaler: training data lies within the range [-1, 1] by dividing through the largest maximum value in each feature

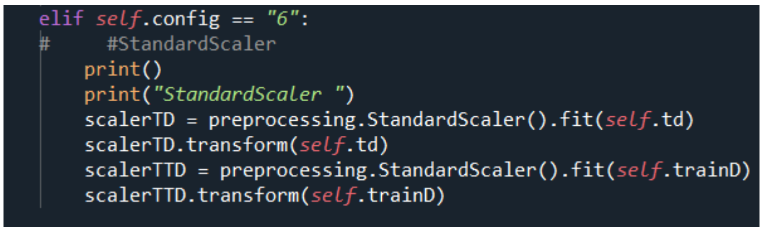


```
elif self.config == "4":
#Scale Features
    #MaxAbsScaler
    print()
    print("MaxAbsScaler ")
    maxabs = preprocessing.MaxAbsScaler()
    self.td = maxabs.fit_transform(self.td)
    self.trainD = maxabs.fit_transform(self.trainD)
```

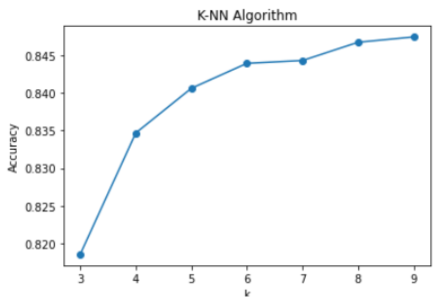


r2 = 0.8679634065862826

StandardScaler : Use following formula

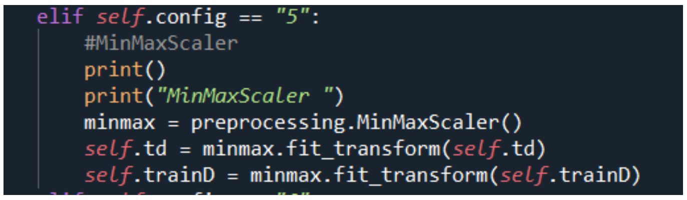


```
elif self.config == "6":
#     #StandardScaler
    print()
    print("StandardScaler ")
    scalerTD = preprocessing.StandardScaler().fit(self.td)
    scalerTD.transform(self.td)
    scalerTTD = preprocessing.StandardScaler().fit(self.trainD)
    scalerTTD.transform(self.trainD)
```

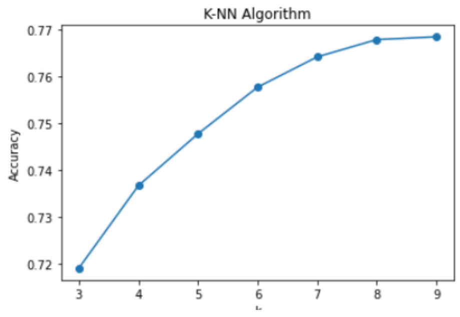


r2 = 0.8474151167266731

MinMaxScaler: data matrix to the [0, 1] range. In this case the r2 value is 0.7 and when I compared with the other techniques its decreased the performance.



```
elif self.config == "5":
    #MinMaxScaler
    print()
    print("MinMaxScaler ")
    minmax = preprocessing.MinMaxScaler()
    self.td = minmax.fit_transform(self.td)
    self.trainD = minmax.fit_transform(self.trainD)
```



r2 = 0.7684430125646045

Note : to run the Part1B.py it will ask the input

Please Enter function number (Standard KNN --1 , SelectionPercentile -- 2,Normalization -- 3 , MaxAbsScaler --4 ,MinMaxScaler --5,StandardScaler --6)

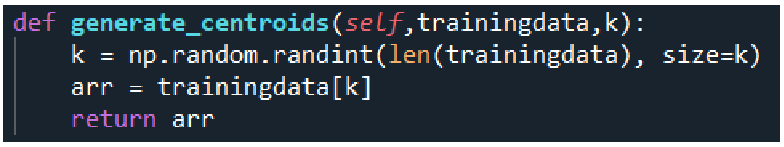
Part 2(A) – Development of a KMeans Clustering Algorithm

KMeans Clustering algorithm –

- Widely used for unsupervised learning
- Specify the number of cluster k,allocates each data point to the nearest cluster while we get the centroids as small as possible.

Following are the steps of KMeans Clustering Algorithm

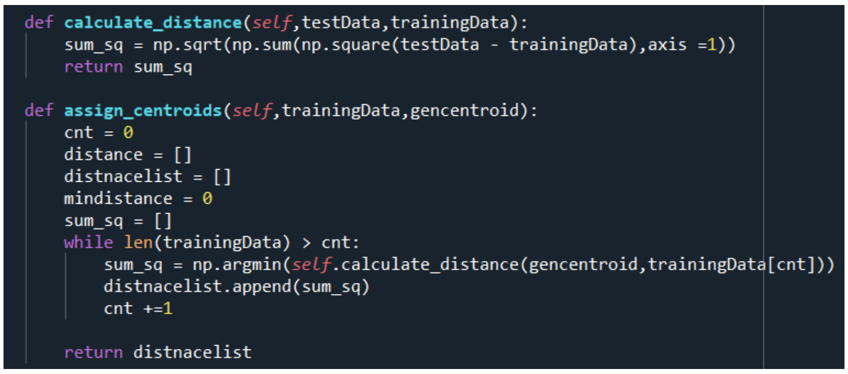
1.generate_centroids : This fuction randomly selected the centroids and return the selected the centroids



```
def generate_centroids(self,trainingdata,k):
    k = np.random.randint(len(trainingdata), size=k)
    arr = trainingdata[k]
    return arr
```

2.assign_centroids :

1. accepts the all feature data and generated centroids
2. calculate the distance by calling distance function between each centroids and all feature data and there we return the index of the feature data which is close to centroid
3. return the list of array of distance.

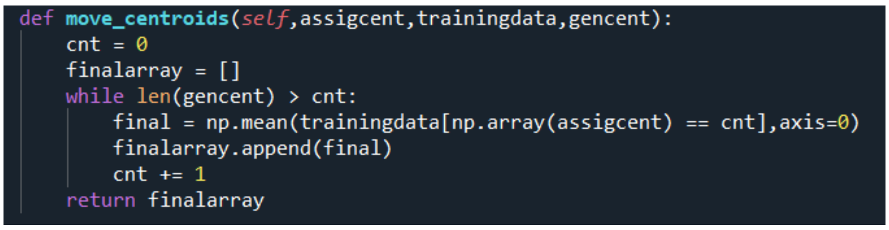


```
def calculate_distance(self,testData,trainingData):
    sum_sq = np.sqrt(np.sum(np.square(testData - trainingData),axis =1))
    return sum_sq

def assign_centroids(self,trainingData,gencentroid):
    cnt = 0
    distance = []
    distnacelist = []
    mindistance = 0
    sum_sq = []
    while len(trainingData) > cnt:
        sum_sq = np.argmin(self.calculate_distance(gencentroid,trainingData[cnt]))
        distnacelist.append(sum_sq)
        cnt +=1

    return distnacelist
```

3.move_centroids :

1.accepts the all feature data ,centroid indices and current centroids.

2.calculate the mean of the data points and assign to each specific centroids and return the new centroids.

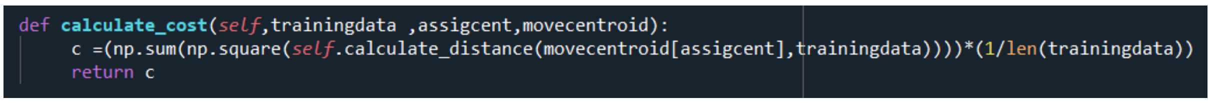


```
def move_centroids(self,assigcent,trainingdata,gencent):
    cnt = 0
    finalarray = []
    while len(gencent) > cnt:
        final = np.mean(trainingdata[np.array(assigcent) == cnt],axis=0)
        finalarray.append(final)
        cnt += 1
    return finalarray
```

4.calculate_cost :

1.accepts the all feature data, centroid indices and current centroids.

2.calculate the distortion cost fun using the formula

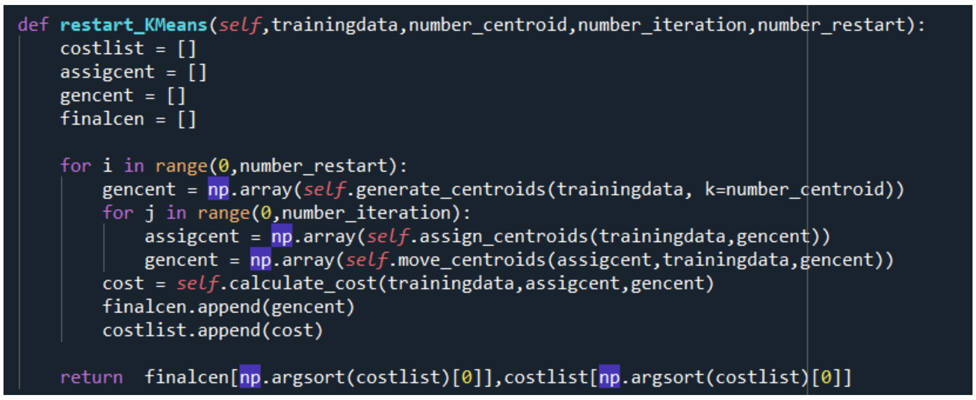


```
def calculate_cost(self,trainingdata ,assigcent,movecentroid):
    c =(np.sum(np.square(self.calculate_distance(movecentroid[assigcent],trainingdata))))*(1/len(trainingdata))
    return c
```

5.restart_kmeans

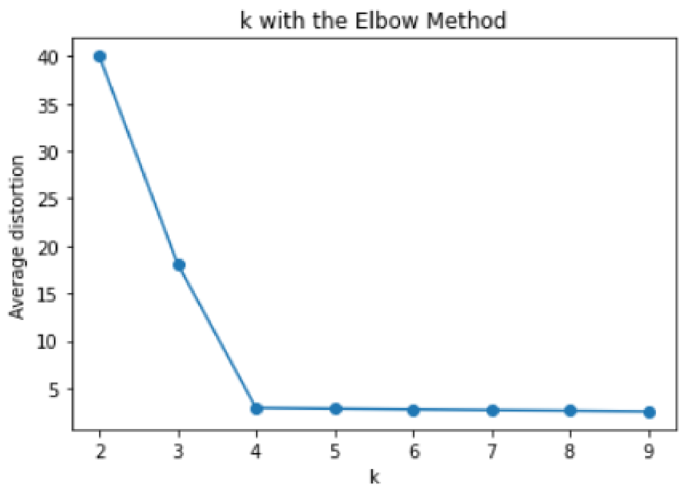
1.accepts all feature data, number of centroids,number of iteration,number of restarts.

2 The implementation is done in the below code



```
def restart_KMeans(self,trainingdata,number_centroid,number_iteration,number_restart):
    costlist = []
    assigcent = []
    gencent = []
    finalcen = []

    for i in range(0,number_restart):
        gencent = np.array(self.generate_centroids(trainingdata, k=number_centroid))
        for j in range(0,number_iteration):
            assigcent = np.array(self.assign_centroids(trainingdata,gencent))
            gencent = np.array(self.move_centroids(assigcent,trainingdata,gencent))
        cost = self.calculate_cost(trainingdata,assigcent,gencent)
        finalcen.append(gencent)
        costlist.append(cost)

    return  finalcen[np.argsort(costlist)[0]],costlist[np.argsort(costlist)[0]]
```

Below the elbow plot of the kMeans cluster



```
 Cost : [40.00875322733565, 18.076085822636838, 2.922150566084583, 2.838176780916482, 2.756806171941531, 2.6816137165045273, 2.620443142500415, 2.538474627793665]
```

The curve is going down on each iteration

Part 2(B) – Researching Alternatives to KMeans

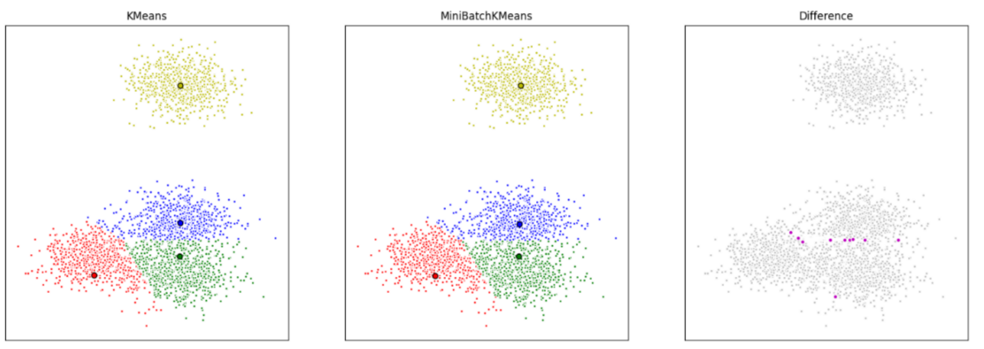
This algorithm is alternative of KMeans algorithm for clustering dataset. When u compare the both algorithm ,Mini-Batch KMeans reduce the computational cost by not using all dataset at one time,it takes small batches of the data in fixed size. It reduces the number of distance per iteration at the cost of lower cluster quality.

Working of Mini-Batch KMeans algorithm :

1. This algorithm accepts for each iteration small random number of batches
2. Depending on the previous location of the cluster centroid, it assigns each data to the cluster
3. updates the clusters with the combination of the data and apply learning rate(inverse the number of data) decrease the number of iteration.
4. if the number of iteration increases ,the effect of new data is reduced.

Advantages :

1. Reduce Computation time
2. The simple unsupervised learning
3. Use the subset of input data in each iteration
4. Reduce the amount of computation required to coverage to a local solution.



References :

https://www.geeksforgeeks.org/ml-mini-batch-k-means-clustering-algorithm/#:~:text=Mini%20Batch%20K%2Dmeans%20algorithm's%20main%20idea%20is%20to,this%20is%20repeated%20until%20convergence.

http://ijsetr.com/uploads/462153IJSETR4282-245.pdf

https://scikit-learn.org/stable/modules/preprocessing.html#normalization
https://scikit-learn.org/stable/modules/feature_selection.html