|  |
| --- |
|  |
| MACHINE LEARNING PROJECT |
| Supervised Learning using Naïve Bayes and Unsupervised Learning using K-means Clustering |

|  |
| --- |
| Abhishek R Mishra – 1MS16IS003  Dhriti Wadhwa – 1MS16IS023 |

**UNSUPERVISED LEARNING**

**Unsupervised learning** is a type of **machine learning** algorithm used to draw inferences from datasets consisting of input data without labelled responses. A central application of unsupervised learning is in the field of [density estimation](https://en.wikipedia.org/wiki/Density_estimation) in [statistics](https://en.wikipedia.org/wiki/Statistics),though unsupervised learning encompasses many other domains involving summarizing and explaining data features.

Some of the most common algorithms used in unsupervised learning include:

* [Clustering](https://en.wikipedia.org/wiki/Data_clustering)
* [Anomaly detection](https://en.wikipedia.org/wiki/Anomaly_detection)
* [Neural Networks](https://en.wikipedia.org/wiki/Artificial_neural_network)
* Approaches for learning [latent variable models](https://en.wikipedia.org/wiki/Latent_variable_model)

We have used the Clustering approach using K-means Clustering

**Algorithm for K-means clustering**

Our algorithm works as follows, assuming we have inputs *x*1​, *x*2​, *x*3​,..., *xn*​ and value of K

* Step 1 - Pick K random points as cluster centers called centroids.

k = 3

C\_x = np.random.randint(0, np.max(X)-20, size=k)

* Step 2 - Assign each *xi*​  to nearest cluster by calculating its distance to each centroid.

distances = dist(X[i], C)

cluster = np.argmin(distances)

* Step 3 - Find new cluster center by taking the average of the assigned points.

C[i] = np.mean(points, axis=0)

* Step 4 - Repeat Step 2 and 3 until none of the cluster assignments change.

error = dist(C, C\_old, None)

**Program**

from copy import deepcopy

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

plt.rcParams['figure.figsize'] = (16, 9)

plt.style.use('ggplot')

data = pd.read\_csv('uds.csv')

print(data.shape)

data.head()

f1 = data['V1'].values

f2 = data['V2'].values

X = np.array(list(zip(f1, f2)))

plt.scatter(f1, f2, c='black', s=7)

def dist(a, b, ax=1):

return np.linalg.norm(a - b, axis=ax)

#Number of clusters

k = 3

C\_x = np.random.randint(0, np.max(X)-20, size=k)

C\_y = np.random.randint(0, np.max(X)-20, size=k)

C = np.array(list(zip(C\_x, C\_y)), dtype=np.float32)

print(C)

plt.scatter(f1, f2, c='#050505', s=7)

plt.scatter(C\_x, C\_y, marker='\*', s=200, c='g')

plt.show()

C\_old = np.zeros(C.shape)

clusters = np.zeros(len(X))

error = dist(C, C\_old, None)

while error != 0:

for i in range(len(X)):

distances = dist(X[i], C)

cluster = np.argmin(distances)

clusters[i] = cluster

C\_old = deepcopy(C)

for i in range(k):

points = [X[j] for j in range(len(X)) if clusters[j] == i]

C[i] = np.mean(points, axis=0)

error = dist(C, C\_old, None)

colors = ['r', 'g', 'b', 'y', 'c', 'm']

fig, ax = plt.subplots()

for i in range(k):

points = np.array([X[j] for j in range(len(X)) if clusters[j] == i])

ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])

ax.scatter(C[:, 0], C[:, 1], marker='\*', s=200, c='#050505')

plt.show()

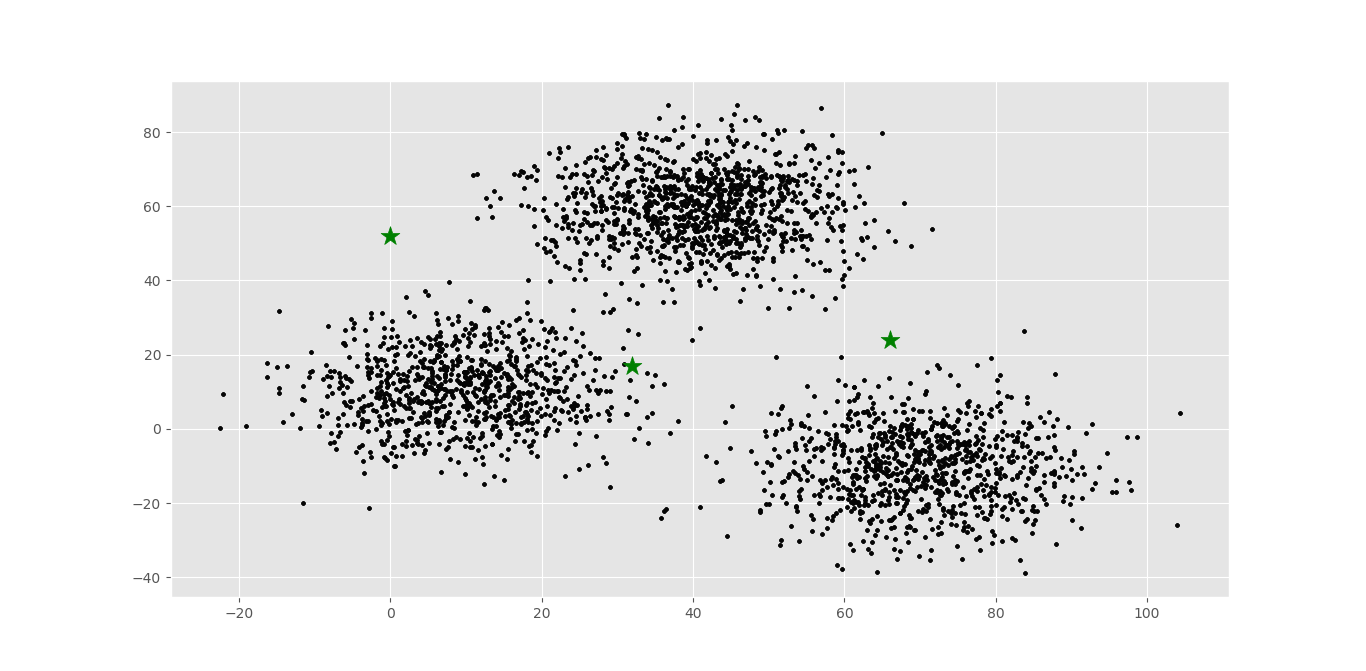


Fig 1 : Clusters with random centroid values

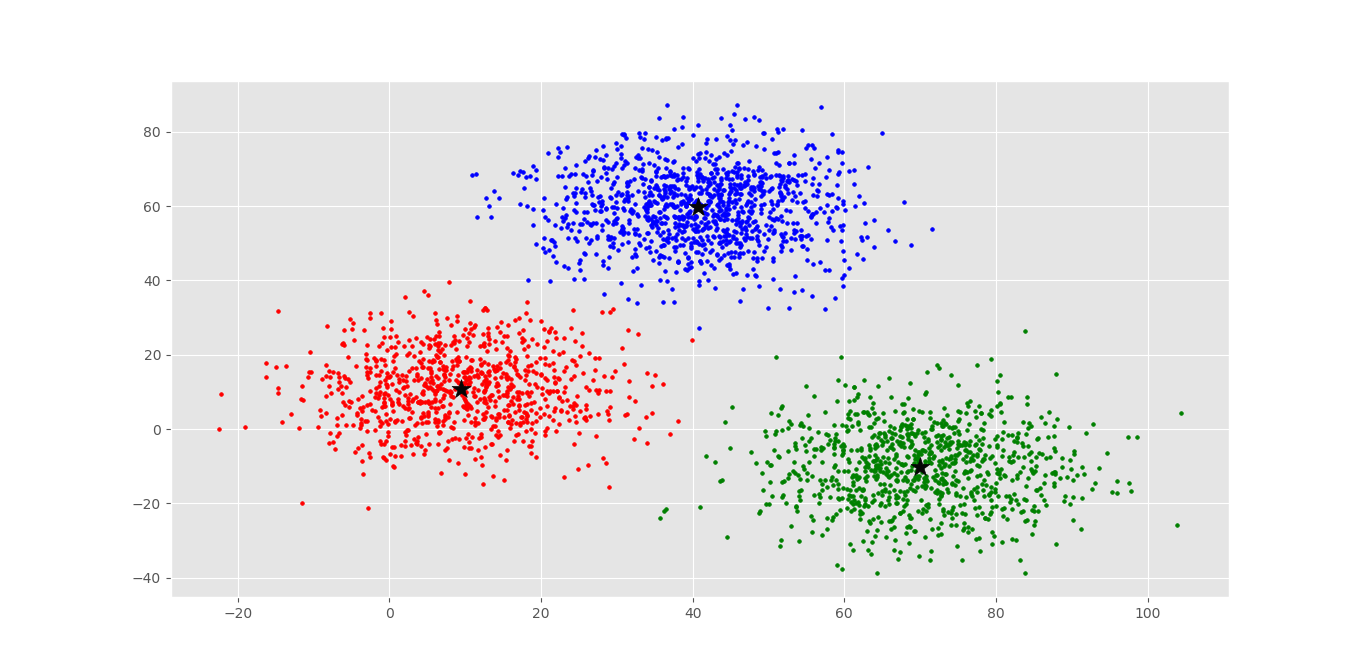


Fig 2 : Clusters with absolute centroid values

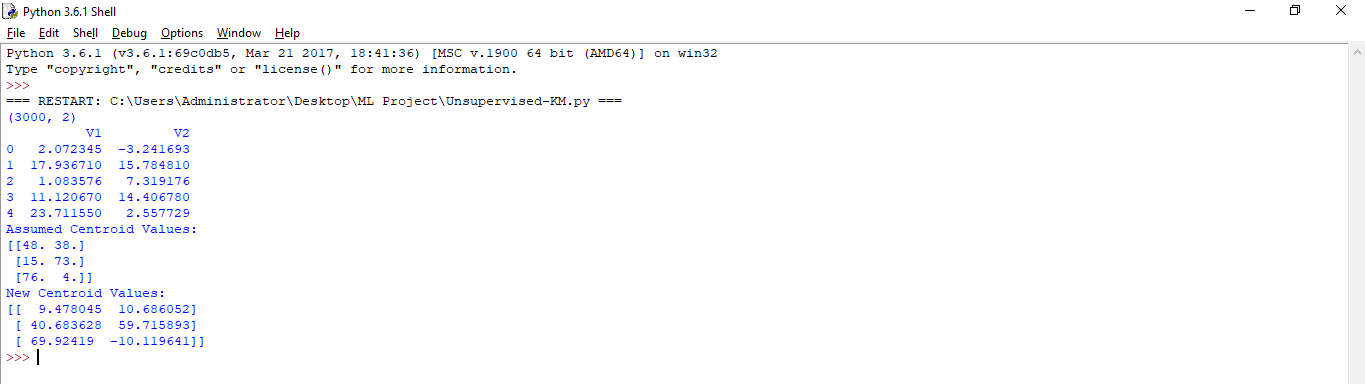


Fig 3 : Output of K means clustering

**SUPERVISED LEARNING**

**Supervised learning** is the [machine learning](https://en.wikipedia.org/wiki/Machine_learning) task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled [training data](https://en.wikipedia.org/wiki/Training_set) consisting of a set of training examples.

The most widely used learning algorithms are:

* [Support Vector Machines](https://en.wikipedia.org/wiki/Support_Vector_Machines)
* [linear regression](https://en.wikipedia.org/wiki/Linear_regression)
* [logistic regression](https://en.wikipedia.org/wiki/Logistic_regression)
* [naive Bayes](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)
* [linear discriminant analysis](https://en.wikipedia.org/wiki/Linear_discriminant_analysis)
* [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning)
* [k-nearest neighbor algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm)
* [Neural Networks](https://en.wikipedia.org/wiki/Artificial_neural_network) ([Multilayer perceptron](https://en.wikipedia.org/wiki/Multilayer_perceptron))
* [Similarity learning](https://en.wikipedia.org/wiki/Similarity_learning)

We have used Naïve Bayes Algorithm.

**Algorithm for Naïve Bayes Classifier**

* Step 1 - Traverse through the features and count the number of occurences of a nominal value with respect to the class variable and store them in a list.

def count(data,colname,label,target):

condition = (data[colname] == label) & (data['Outcome'] == target)

predicted = []

* Step 2 – Calculate the probabilities of 1 and 0.
* Step 3 – Train it by traversing through all the features and calculate the probabilities of all the unique categorial values and save them in a dictionary.
* Step 4 – Test the model and find the final probabilities of class 0 and class 1.
* Step 5 – Assign the highest probabilty of occurrence to the predicted list.
* Step 6 – Calculate the confusion matrix and find and display accuracy.

**Program**

import pandas as pd

data = pd.read\_csv('sds.csv')

labels = ['low','medium','high']

for j in data.columns[:-1]:

mean = data[j].mean()

data[j] = data[j].replace(0,mean)

data[j] = pd.cut(data[j],bins=len(labels),labels=labels)

#Split data set for training. 70 is 70% of data set used for training

split\_per = [70]

def count(data,colname,label,target):

condition = (data[colname] == label) & (data['Outcome'] == target)

return len(data[condition])

for i in split\_per:

predicted = []

probabilities = {0:{},1:{}}

#amount of data set used for training

train\_len = int((i\*len(data))/100)

#print(train\_len)

train\_X = data.iloc[:train\_len,:]

#extracting as many number of rows of data as specified by train\_len

test\_X = data.iloc[train\_len+1:,:-1]

#extracting the remaining rows for testing

test\_y = data.iloc[train\_len+1:,-1]

count\_0 = count(train\_X,'Outcome',0,0)

count\_1 = count(train\_X,'Outcome',1,1)

#Find probability of occurence of 0 and 1

prob\_0 = count\_0/len(train\_X)

prob\_1 = count\_1/len(train\_X)

for j in train\_X.columns[:-1]:

probabilities[0][j] = {}

probabilities[1][j] = {}

for k in labels:

count\_k\_0 = count(train\_X,j,k,0)

count\_k\_1 = count(train\_X,j,k,1)

probabilities[0][j][k] = count\_k\_0 / count\_0

probabilities[1][j][k] = count\_k\_1 / count\_1

for row in range(0,len(test\_X)):

prod\_0 = prob\_0

prod\_1 = prob\_1

#Find the probabilities in the test set for each column

for feature in test\_X.columns:

prod\_0 \*= probabilities[0][feature][test\_X[feature].iloc[row]]

prod\_1 \*= probabilities[1][feature][test\_X[feature].iloc[row]]

if prod\_0 > prod\_1:

predicted.append(0)

else:

predicted.append(1)

tp,tn,fp,fn = 0,0,0,0

#Calculate Accuracy Metrics in test set

for j in range(0,len(predicted)):

if predicted[j] == 0:

if test\_y.iloc[j] == 0:

tp += 1

else:

fp += 1

else:

if test\_y.iloc[j] == 1:

tn += 1

else:

fn += 1

print('Predicted Output: ')

print(predicted)

#for i in predicted :

# if(i==1):

# print('Has Diabetes')

# else:

# print('Does not have Diabetes')

print()

print('For Training Set ' + str(i) + '%')

print('Accuracy: ',((tp+tn)/len(test\_y))\*100)

print('TP:' + str(tp) + ' ' + 'FN:'+ str(fn))

print('FP:' + str(fp) + ' ' + 'TN:'+ str(tn))

print()

print('Specificity: ',(tn/(tn+fp))\*100)

print('Sensitivity: ',(tp/(tp+fn))\*100)

print('Precision: ',(tp/(tp+fp))\*100)

print('Recall: ',(tp/(tp+fn))\*100)

print('F1: ',(tp/(tp+((tp+fn)/2)))\*100)

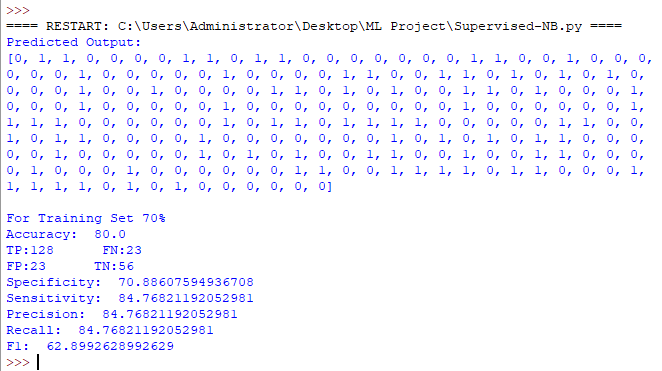


Fig 4 : Output for Naïve Bayes Classifier

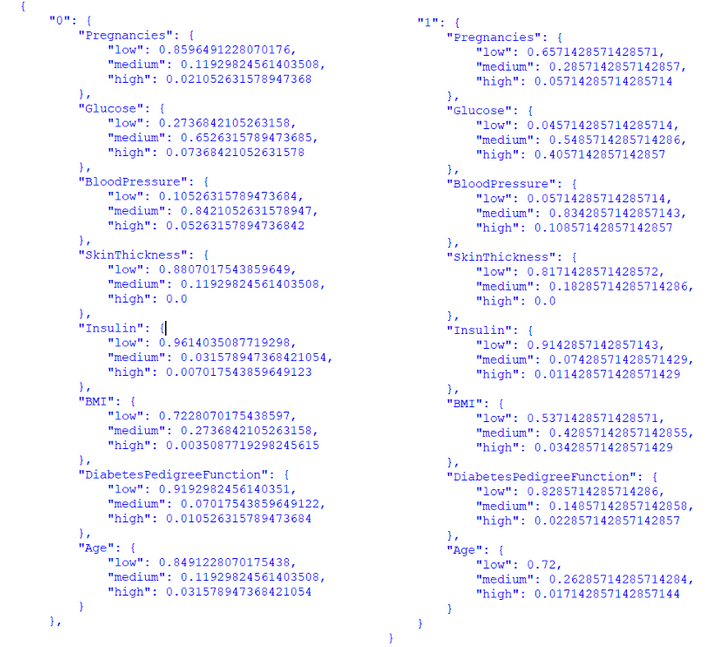


Fig 5 : An example for the probabilities in the dictionary