EDS Assignment 6

Our Team Members:

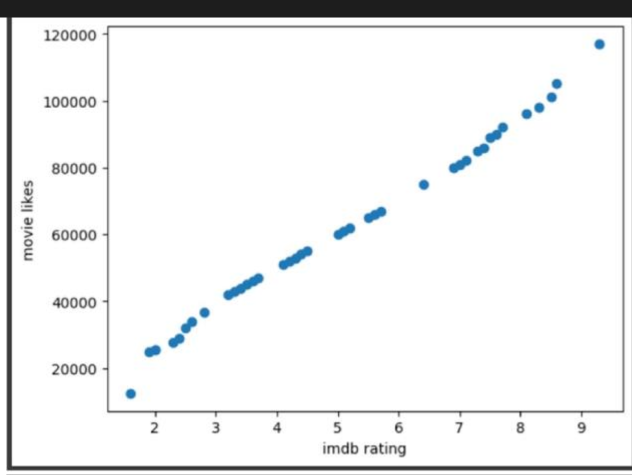
Tuhinansh Sharma (265)

Tushar Najawan (266)

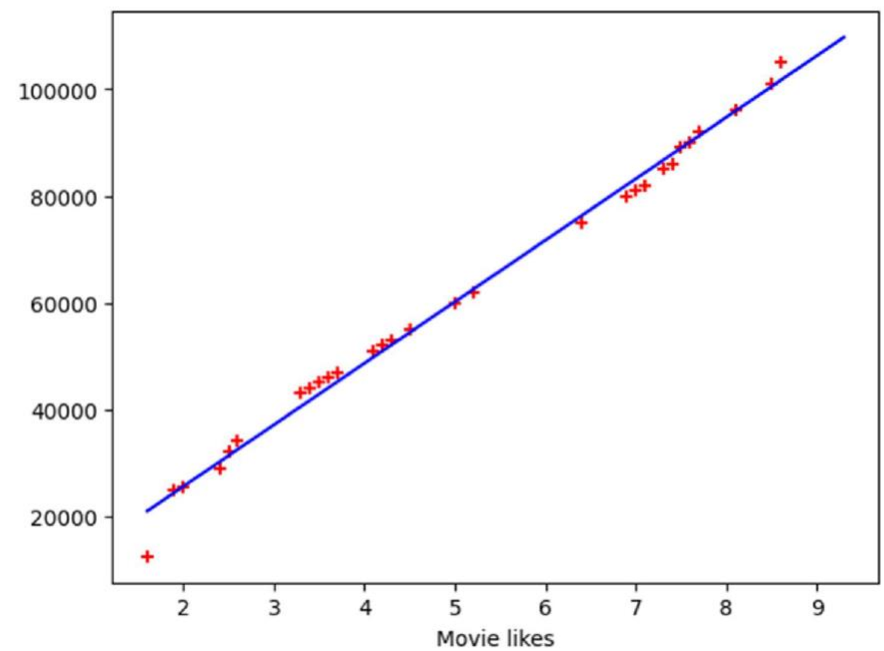
Atharva Pawar (279)

1. <u>Linear Regression:</u>

```
import nump
```



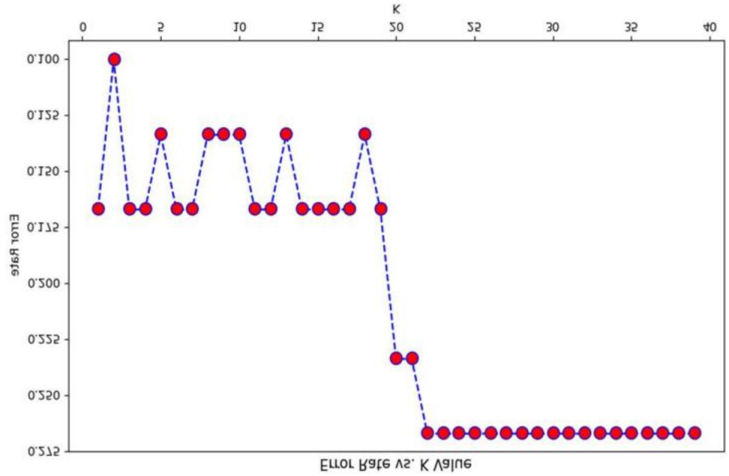
```
X = np.array(df1[['imdb_score']]).reshape(-1,1)
Y = np.array(df1[['movie_likes']]).reshape(-
1,1) X_train,X_test,Y_train,Y_test =
train_test_split(X,Y,test_size = 0.25)

# create
linear
regression
object reg =
linear_model.Li
```



2. KNN

```
3. import pandas as pd
4. import seaborn as sns
5. import matplotlib.pyplot as plt
6. import numpy as np
7. df = pd.read_csv("prostate.csv")
8. df.head()
11     scaler =
.
12     scaler.fit(df.drop('Target', axis=1))
.      scaled_features =
13
```

```
df_feat = pd.DataFrame(scaled_features,
          columns=df.columns[:-1])
df_feat.head()
from sklearn.metrics import classification_report,\
  confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

X_train, X_test,\
  y_train, y_test = train_test_split(scaled_features,
                    df['Taregt'],
                    test_size=0.30)

# Remember that we are trying to come up
# with a model to predict whether
# someone will Target or not.
# We'll start with k = 1.

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)

# Predictions and Evaluations
# Let's evaluate our KNN model !
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
error_rate = []

# Will take some time
for i in range(1, 40):

  knn = KNeighborsClassifier(n_neighbors=i)
  knn.fit(X_train, y_train)
  pred_i = knn.predict(X_test)
  error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10, 6))
plt.plot(range(1, 40), error_rate, color='blue',
    linestyle='dashed', marker='o',
    markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()
```




```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K = 1
knn = KNeighborsClassifier(n_neighbors = 1)

knn.
fit(
X_tr
ain,
y_tr
```

WITH K = 1

Confusion Matrix

[[19 3]

[2 6]]

Classification Report

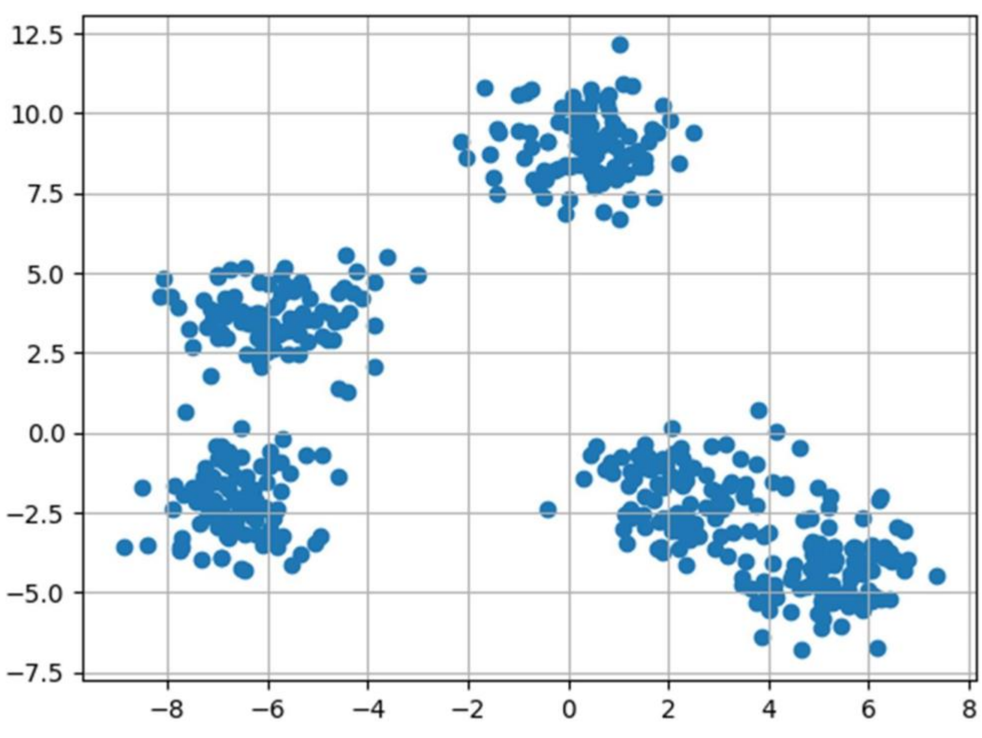
	precision	recall	f1-score	support
0	0.90	0.86	0.88	22
1	0.67	0.75	0.71	8
accuracy			0.83	30
macro avg	0.79	0.81	0.79	30
weighted avg	0.84	0.83	0.84	30

```
# NOW WITH K = 10
knn = KNeighborsClassifier(n_neighbors = 10)

knn.
fit(
X_tr
ain,
y_tr
ain)
```

```
WITH K = 10

Confusion Matrix

[[21 1]

 [ 3  5]]

Classification Report

              precision    recall  f1-score   support


           0       0.88      0.95      0.91        22

           1       0.83      0.62      0.71         8


    accuracy                           0.87        30

   macro avg       0.85      0.79      0.81        30

weighted avg       0.86      0.87      0.86        30
```

3. K means

```
#k-Means
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X,y = make_blobs(n_samples = 500,n_features = 2,centers =

5,random_state = 23)fig = plt.figure(0)
plt.grid(T
```

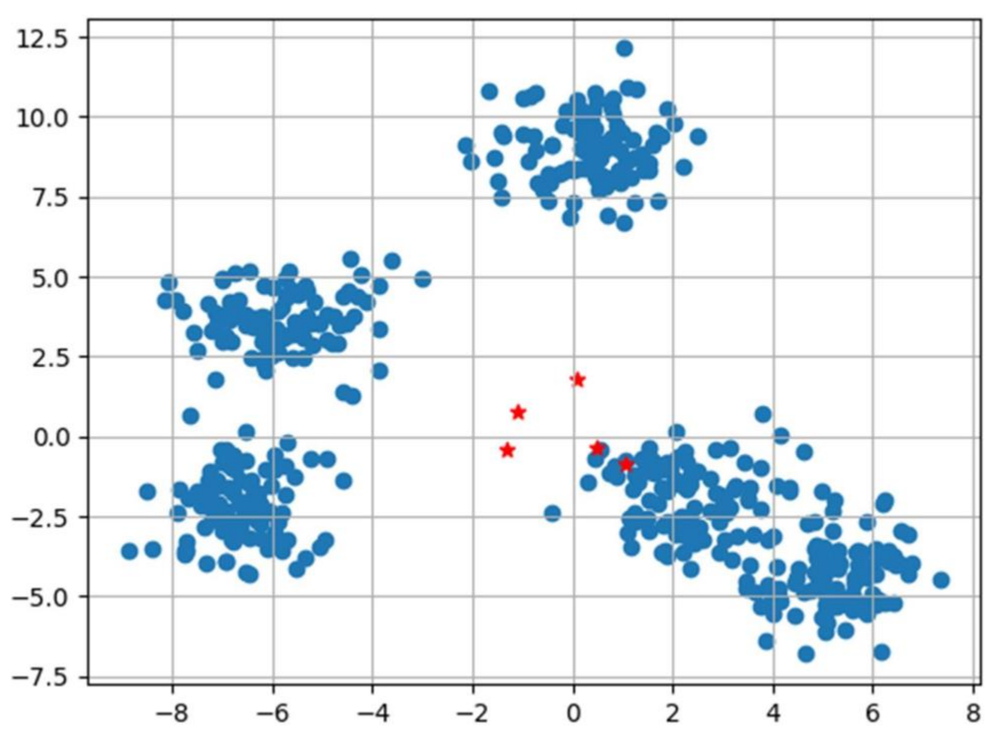



```
k = 5
```

```
clusters = {}
np.random.seed(23)

for idx in range(k):
  center = 2*(2*np.random.random((X.shape[1],))-1)
  points = []
  cluster = {
    'center' : center,
    'points' : []
  }

  clusters[idx] = cluster

clusters
```

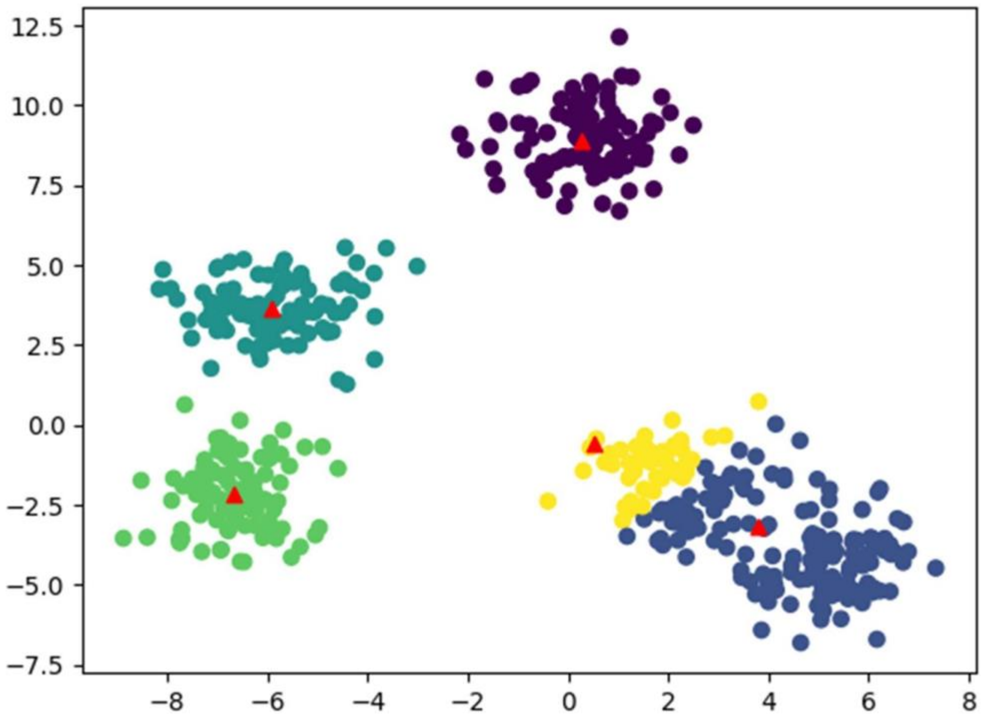
```
{0: {'center': array([0.06919154, 1.78785042]), 'points': []},
 1: {'center': array([ 1.06183904, -0.87041662]), 'points': []},
 2: {'center': array([-1.11581855,  0.74488834]), 'points': []},
 3: {'center': array([-1.33144319, -0.43023013]), 'points': []},
 4: {'center': array([ 0.47220939, -0.35227962]), 'points': []}}
```

```
plt.scatte
r(X[:,0],X
[:,1])
plt.grid(T
rue)
```




```
def distance(p1,p2):
  return np.sqrt(np.sum((p1-p2)**2))
```

```
#Implementing E step
def assign_clusters(X, clusters):
  for idx in range(X.shape[0]):
    dist = []

    curr_x = X[idx]

    for i in range(k):
      dis = distance(curr_x,clusters[i]['center'])
      dist.append(dis)
    curr_cluster = np.argmin(dist)
    clusters[curr_cluster]['points'].append(curr_x)
  return clusters

#Implementing the M-Step
def update_clusters(X, clusters):
  for i in range(k):
    points = np.array(clusters[i]['points'])
    if points.shape[0] > 0:
      new_center = points.mean(axis =0)
      clusters[i]['center'] = new_center

      clusters[i]['points'] = []
  return clusters

def pred_cluster(X, clusters):
  pred = []
  for i in range(X.shape[0]):
    dist = []
    for j in range(k):
      dist.append(distance(X[i],clusters[j]['center']))
    pred.append(np.argmin(dist))
  return pred

clusters = assign_clusters(X,clusters)
clusters = update_clusters(X,clusters)
pred = pred_cluster(X,clusters)

plt.scatter(X[:,0],X[:,1],c = pred)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.show()
```



12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−8
−6
−4
−2
0
2
4
6
8