```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score

raw_data = pd.read_csv('Q1_data.csv')
feature_matrix = raw_data.drop('class', axis=1).values
label_vector = np.where(raw_data['class'] == 'normal', 0, 1)

print(raw_data.head());
```

```
   duration  src_bclasstes  dst_bclasstes  count  srv_count
serror_rate  \
0         0            520              0    428        428
0.0
1         0              0              0    131         18
0.0
2         0              0              0     20          8
1.0
3         0           1235            404      1          4
0.0
4         0            224           1415      1          1
0.0

   srv_serror_rate  rerror_rate  srv_rerror_rate
same_srv_rate  ...  \
0              0.0          0.0              0.0           1.00  ...

1              0.0          1.0              1.0           0.14  ...

2              1.0          0.0              0.0           0.40  ...

3              0.0          0.0              0.0           1.00  ...

4              0.0          0.0              0.0           1.00  ...


   dst_host_srv_count  dst_host_same_srv_rate  dst_host_diff_srv_rate
\
0                 255                    1.00                    0.00

1                  18                    0.07                    0.06

2                  68                    0.27                    0.02

3                 179                    0.72                    0.12

4                  48                    1.00                    0.00
```

```
   dst_host_same_src_port_rate  dst_host_srv_diff_host_rate  \
0                         1.00                         0.00
1                         0.00                         0.00
2                         0.01                         0.00
3                         0.04                         0.02
4                         0.02                         0.00

   dst_host_serror_rate  dst_host_srv_serror_rate
dst_host_rerror_rate  \
0                   0.0                      0.00
0.00
1                   0.0                      0.00
1.00
2                   1.0                      1.00
0.00
3                   0.0                      0.01
0.04
4                   0.0                      0.00
0.00

   dst_host_srv_rerror_rate        class
0                       0.0  anomalclass
1                       1.0  anomalclass
2                       0.0  anomalclass
3                       0.0       normal
4                       0.0       normal

[5 rows x 23 columns]
```

```
print(raw_data.info());
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 23 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   duration                     200 non-null    int64
 1   src_bclasstes                200 non-null    int64
 2   dst_bclasstes                200 non-null    int64
 3   count                        200 non-null    int64
 4   srv_count                    200 non-null    int64
 5   serror_rate                  200 non-null    float64
 6   srv_serror_rate              200 non-null    float64
 7   rerror_rate                  200 non-null    float64
 8   srv_rerror_rate              200 non-null    float64
 9   same_srv_rate                200 non-null    float64
 10  diff_srv_rate                200 non-null    float64
 11  srv_diff_host_rate           200 non-null    float64
```

```
 12  dst_host_count               200 non-null    int64
 13  dst_host_srv_count           200 non-null    int64
 14  dst_host_same_srv_rate       200 non-null    float64
 15  dst_host_diff_srv_rate       200 non-null    float64
 16  dst_host_same_src_port_rate  200 non-null    float64
 17  dst_host_srv_diff_host_rate  200 non-null    float64
 18  dst_host_serror_rate         200 non-null    float64
 19  dst_host_srv_serror_rate     200 non-null    float64
 20  dst_host_rerror_rate         200 non-null    float64
 21  dst_host_srv_rerror_rate     200 non-null    float64
 22  class                        200 non-null    object
dtypes: float64(15), int64(7), object(1)
memory usage: 36.1+ KB
None
```

```
scaler = StandardScaler() # to normalize all the data
feature_matrix = scaler.fit_transform(feature_matrix)
print(feature_matrix); # feature_matrix has the training data without
class field from csv
```

```
[[-0.1180733  -0.07777228 -0.2117447  ... -0.67984097 -0.4467788
  -0.43567987]
 [-0.1180733  -0.07810672 -0.2117447  ... -0.67984097  2.41169081
   2.35177931]
 [-0.1180733  -0.07810672 -0.2117447  ...  1.47462258 -0.4467788
  -0.43567987]
 ...
 [-0.1180733  -0.07810672 -0.2117447  ...  1.47462258 -0.4467788
  -0.43567987]
 [-0.1180733  -0.07800703 -0.15243546 ... -0.65829634 -0.4181941
  -0.40780528]
 [-0.1180733  -0.07808743 -0.19969542 ... -0.67984097 -0.4467788
  -0.43567987]]
```

```
# split data
training_features, testing_features, training_labels, testing_labels =
train_test_split(
    feature_matrix,
    label_vector,
    test_size=0.3,
    stratify=label_vector,
    random_state=665
    );

print(f"Training Set: Normal Count = {np.sum(training_labels == 0)}")
print(f"Training Set: Anomaly Count = {np.sum(training_labels == 1)}")
print(f"Testing Set:  Normal Count = {np.sum(testing_labels == 0)}")
print(f"Testing Set:  Anomaly Count = {np.sum(testing_labels == 1)}")
```

```
Training Set: Normal Count = 70
Training Set: Anomaly Count = 70
Testing Set:  Normal Count = 30
Testing Set:  Anomaly Count = 30
```

```
def activation(val):
    # step function
    if val >= 0: return 1;
    return 0;

def train_perceptron(features, labels, learning_rate=0.1,
max_iterations=1000):
    num_samples, num_features = features.shape

    # init weights and threshold
    weights = np.zeros(num_features)
    bias = 0.0

    for it in range(max_iterations):
        for idx in range(num_samples):
            curr_sample = features[idx];
            true_label = labels[idx];

            # fn = x0 * w0 + x1 * w1 + .... + bias
            fn = np.dot(curr_sample, weights) + bias;
            predicted_val = activation(fn);

            # update weigths
            if true_label != predicted_val:
                err = true_label - predicted_val;

                weights = weights + (learning_rate * err *
curr_sample);
                bias = bias + (learning_rate * err);

    return weights, bias

def predict(features, weights, bias):
    pred_list = [];
    num_samples = len(features);

    for idx in range(num_samples):
        curr_sample = features[idx];

        fn = np.dot(curr_sample, weights) + bias;
        pred_val = activation(fn);

        pred_list.append(pred_val);

    return np.array(pred_list);
```

```
# training
trained_weights, trained_bias = train_perceptron(training_features,
training_labels)

predicted_test_labels = predict(testing_features, trained_weights,
trained_bias)
perceptron_accuracy = accuracy_score(testing_labels,
predicted_test_labels)

print(f"Perceptron Classification Accuracy: {perceptron_accuracy *
100:.2f}%")
```

```
Perceptron Classification Accuracy: 93.33%
```

```
best_k_value = 0
highest_training_accuracy = 0.0

# try different num of clusters
for num_clusters in [2, 3, 4]:

    # Initialize and fit K-Means
    kmeans_algorithm = KMeans(n_clusters=num_clusters,
random_state=665, n_init=10)
    kmeans_algorithm.fit(training_features)

    cluster_assignments = kmeans_algorithm.labels_

    print(f"\nanalysis for K = {num_clusters}:")

    # This map will store which class (0 or 1) each cluster represents
    id_class_map = {}

    for cluster_id in range(num_clusters):
        # get all points that belong to this cluster
        indices_in_cluster = np.where(cluster_assignments ==
cluster_id)
        true_labels_in_cluster = training_labels[indices_in_cluster]

        # calculate the num of elements in cluster
        normal_count = np.sum(true_labels_in_cluster == 0)
        anomaly_count = np.sum(true_labels_in_cluster == 1)
        total_count_in_cluster = normal_count + anomaly_count

        # make sure the cluster isn't empty
        if total_count_in_cluster == 0:
            continue;

        percent_normal = (normal_count / total_count_in_cluster) *
100;
        percent_anomaly = (anomaly_count / total_count_in_cluster) *
```

```
100;

        print(f"  cluster id {cluster_id}: normal={percent_normal:.1f}
%, anomaly={percent_anomaly:.1f}%")

        # give the clutter the label of the majority
        if normal_count > anomaly_count:
            id_class_map[cluster_id] = 0 # normal
        else:
            id_class_map[cluster_id] = 1 # anomaly

    # calc accuracy
    predicted_labels_from_clustering = []
    for assigned_cluster in cluster_assignments:
        predicted_label = id_class_map[assigned_cluster]
        predicted_labels_from_clustering.append(predicted_label)

    current_accuracy = accuracy_score(training_labels,
predicted_labels_from_clustering)
    print(f"  classification accuracy (on train set):
{current_accuracy * 100:.2f}%");

    # Keep track of the best K
    if current_accuracy > highest_training_accuracy:
        highest_training_accuracy = current_accuracy;
        best_k_value = num_clusters;

print(f"\noptimal number of clusters (K): {best_k_value}");


analysis for K = 2:
  cluster id 0: normal=0.0%, anomaly=100.0%
  cluster id 1: normal=71.4%, anomaly=28.6%
  classification accuracy (on train set): 80.00%

analysis for K = 3:
  cluster id 0: normal=25.0%, anomaly=75.0%
  cluster id 1: normal=0.0%, anomaly=100.0%
  cluster id 2: normal=86.5%, anomaly=13.5%
  classification accuracy (on train set): 88.57%

analysis for K = 4:
  cluster id 0: normal=25.0%, anomaly=75.0%
  cluster id 1: normal=0.0%, anomaly=100.0%
  cluster id 2: normal=86.3%, anomaly=13.7%
  cluster id 3: normal=100.0%, anomaly=0.0%
  classification accuracy (on train set): 88.57%

optimal number of clusters (K): 3
```