

Assignment 3

Hagar Hesham Kamel Ismail Negm
Ekhlās Soliman Khlil Mosa

1 DATASET

| Load data | Split data |
|---|---|
| <pre>4]: train_df = pd.read_csv('Pokemon_train.csv') test_df = pd.read_csv('Pokemon_test.csv')</pre> <p>Encoder</p> <pre>5]: ord_enc = OrdinalEncoder() train_df["type1"] = ord_enc.fit_transform(train_df[["type1"]]) test_df["type1"] = ord_enc.fit_transform(test_df[["type1"]]) train_df["type1"] = train_df["type1"].astype("int") test_df["type1"] = test_df["type1"].astype("int") test_df.head()</pre> | <p>Split Data</p> <pre>]: X_train = train_df.iloc[:, 0:32] y_train = train_df.iloc[:, -1] X_test = test_df.iloc[:, 0:32] y_test = test_df.iloc[:, -1]</pre> |

First, we load train and test files, after that we encode target using ordinalEncoder method to convert target to numeric, then we split the data.

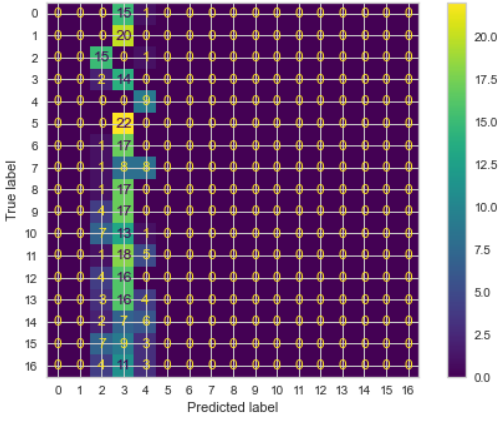
2 BASELINE GNB AND SVM

Gaussian Naïve Bayes

| Code | Output |
|--|--------|
| <pre>gnb_clf = GaussianNB() gnb_clf.fit(X_train,y_train) y_pred_gnb = gnb_clf.predict(X_test) print('\nClassification Report:\n') print(classification_report(y_test, y_pred_gnb)) accuracy_gnb = accuracy_score(y_test, y_pred_gnb)*100 print(accuracy_gnb) print('\nConfusion Matrix:\n') plot_confusion_matrix(gnb_clf, X_test, y_test)</pre> | |

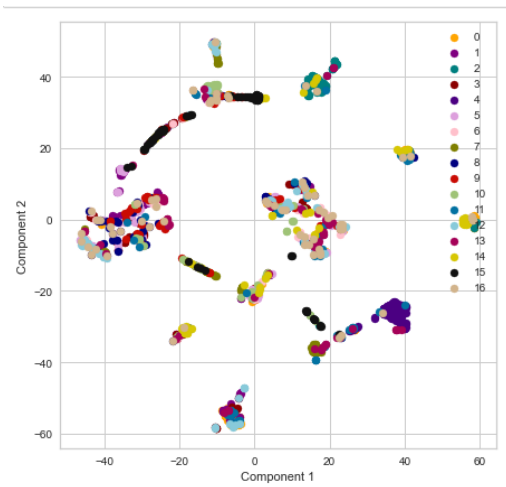
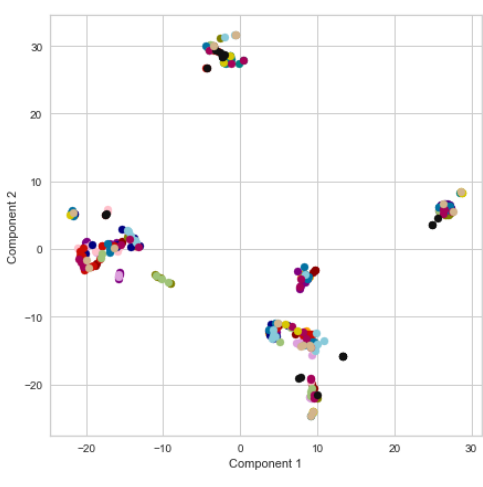
After we fit the data to the GNB model, the baseline accuracy is low as it equals 51.44%.

Support Vector Machine

| Code | Output |
|---|--|
| <pre>SVM j): svm_model = svm.SVC(random_state=0) svm_model.fit(X_train, y_train) y_pred_svm = svm_model.predict(X_test) print('\nClassification Report:\n') print(classification_report(y_test, y_pred_svm)) accuracy_svm = accuracy_score(y_test, y_pred_svm)*100 print(accuracy_svm) print('\nConfusion Matrix:\n') plot_confusion_matrix(svm_model, X_test, y_test)</pre> |  |

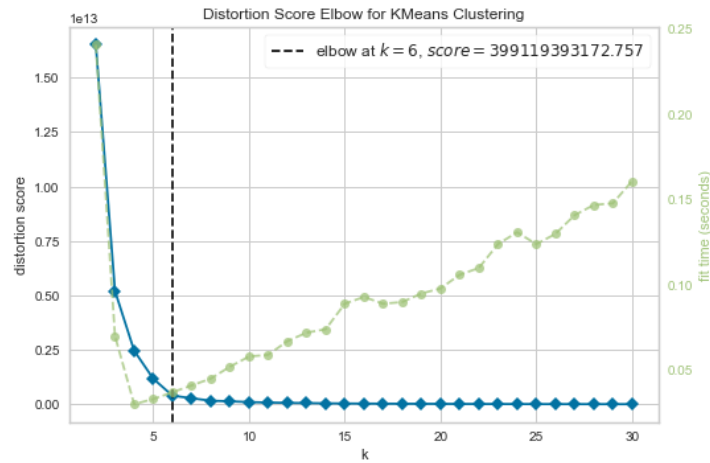
After we fit the data to SVM model, we conclude that SVM isn't suitable for this data as it's nonlinear data, and the baseline accuracy equals 12.14%.

TSNE

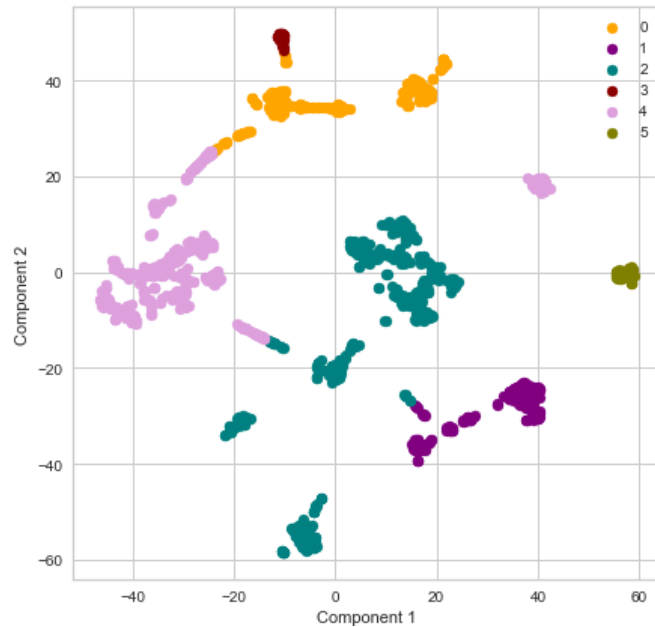
| Train | Test |
|---|--|
|  |  |

3 KMEANS CLUSTERING

Elbow Method



According to the elbow rule, the optimum number of clusters is 6 clusters.



4 DIMENSIONALITY REDUCTION

LDA with GNB

```
: clf_gnb = GaussianNB()

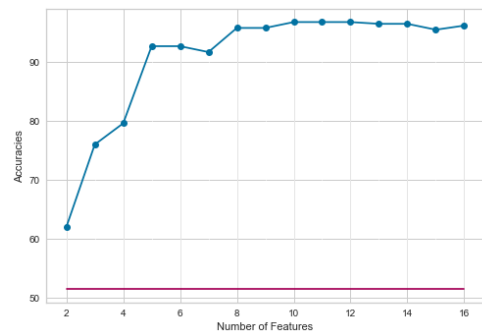
num_features_lda_gnb = []
accuracy_gnb_list_lda = []

for i in range(2, 17):
    lda = LinearDiscriminantAnalysis(n_components= i)
    lda.fit(X_train, y_train)
    lda_train = lda.transform(X_train)
    lda_test = lda.transform(X_test)
    num_features_lda_gnb.append(i)
    clf_gnb.fit(lda_train, y_train)
    y_pred_gnb = clf_gnb.predict(lda_test)
    accuracy = round(accuracy_score(y_test, y_pred_gnb),3) *100
    accuracy_gnb_list_lda.append(accuracy)
```

```
(markers, stemlines, baseline) = plt.stem(num_features_lda_gnb, accuracy_gnb_list_lda , bottom = accuracy_gnb)
plt.setp(stemlines, linestyle="-", color="white", linewidth=0.5 )
plt.plot(num_features_lda_gnb, accuracy_gnb_list_lda)
plt.ylabel('Accuracies')
plt.xlabel('Number of Features')
plt.show()
```

```
: max_3 = np.max(accuracy_gnb_list_lda)
max_index_gnb_lda = accuracy_gnb_list_lda.index(max_3)
num_components_lda_gnb = num_features_lda_gnb.__getitem__(max_index_gnb_lda)
print("Max accuracy : ", max_3)
print("Index of max accuracy : ", max_index_gnb_lda)
print("Best num of components : ", num_components_lda_gnb)
```

```
Max accuracy : 96.8
Index of max accuracy : 8
Best num of components : 10
```



Here, we applied the dimensionality reduction algorithm LDA to our data and fit the resulted data to GNB and the accuracy got better as it equals 96.8% using 10 components.

LDA with SVM

```
svm_model = svm.SVC(random_state=0)

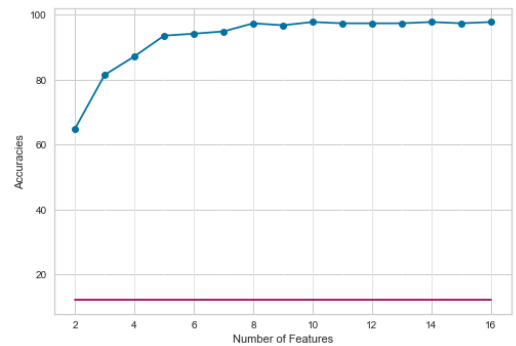
num_features_lda_svm = []
accuracy_svm_list_lda = []

for i in range(2, 17):
    lda = LinearDiscriminantAnalysis(n_components=i)
    lda.fit(X_train, y_train)
    lda_train = lda.transform(X_train)
    lda_test = lda.transform(X_test)
    num_features_lda_svm.append(i)
    svm_model.fit(lda_train, y_train)
    y_pred_svm = svm_model.predict(lda_test)
    accuracy = round(accuracy_score(y_test, y_pred_svm), 3) * 100
    accuracy_svm_list_lda.append(accuracy)
```

```
: max_4 = np.max(accuracy_svm_list_lda)
max_index_svm_lda = accuracy_svm_list_lda.index(max_4)
num_components_lda_svm = num_features_lda_svm.__getitem__(max_index_svm_lda)
print("Max accuracy : ", max_4)
print("Index of max accuracy : ", max_index_svm_lda)
print("Best num of components : ", num_components_lda_svm)
```

```
Max accuracy : 97.8
Index of max accuracy : 8
Best num of components : 10
```

```
: (markers, stemlines, baseline) = plt.stem(num_features_lda_svm, accuracy_svm_list_lda, bottom = accuracy_svm)
plt.setp(stemlines, linestyle='-', color='white', linewidth=0.5)
plt.plot(num_features_lda_svm, accuracy_svm_list_lda)
plt.ylabel('Accuracies')
plt.xlabel('Number of Features')
plt.show()
```



Here, we applied the dimensionality reduction algorithm LDA to our data and fit the resulted data to SVM and the accuracy got better as it equals 97.8% using 10 components.

5 FEATURE SELECTION

Initially, we apply LDA to our data using 10 components which is the optimum number of components that would maximize our accuracy.

```
lda = LinearDiscriminantAnalysis(n_components=10)
lda.fit(X_train, y_train)
lda_train = lda.transform(X_train)
lda_test = lda.transform(X_test)
```

5.1 FILTER METHOD USING INFORMATION GAIN

GNB

```
gnb = GaussianNB()

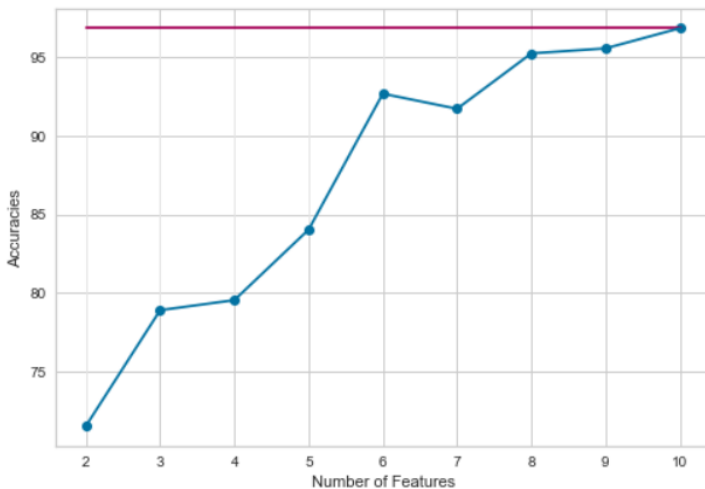
feature_number = []
accuracies = []

for nf in range(2,11):
    fsm = SelectKBest(mutual_info_classif, k=nf)
    fsm.fit(lda_train, y_train)
    fsm_train = fsm.transform(lda_train)
    fsm_test = fsm.transform(lda_test)
    gnb.fit(fsm_train, y_train)
    gnb_pred = gnb.predict(fsm_test)
    accuracy = accuracy_score(y_test, gnb_pred)*100
    feature_number.append(nf)
    accuracies.append(accuracy)
```

```
max_5 = np.max(accuracies)
max_index_gnb_filter = accuracies.index(max_5)
print("Max accuracy : ", max_5)
print("Index of max accuracy : ", max_index_gnb_filter)
```

```
Max accuracy : 96.80511182108627
Index of max accuracy : 8
```

```
(markers, stemlines, baseline) = plt.stem(feature_number, accuracies, bottom = max_3)
plt.setp(stemlines, linestyle="-", color="white", linewidth=0.5 )
plt.plot(feature_number, accuracies)
plt.ylabel('Accuracies')
plt.xlabel('Number of Features')
```



Using the filtering method with the GNB model to choose the best k components, gave us the same optimum performance as the baseline optimum accuracy of GNB with LDA using 10 components. An accuracy of 96.8% was obtained.

SVM

```
svm_model = svm.SVC(random_state=0)

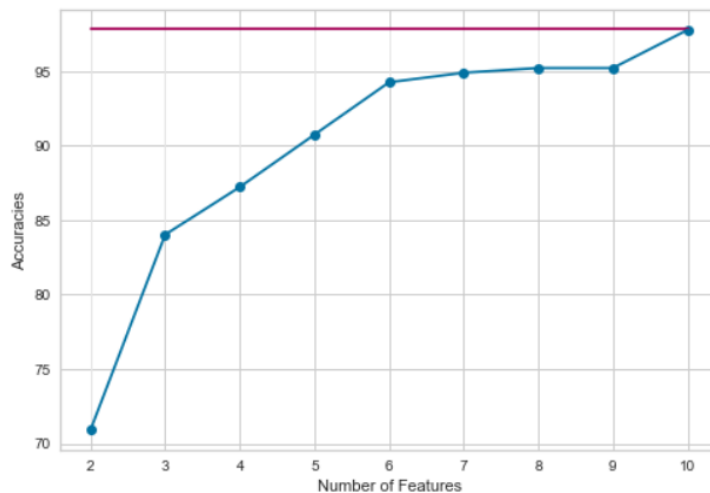
feature_number_svm_filter = []
accuracies_svm_filter = []

for nf in range(2,11):
    fsm = SelectKBest(mutual_info_classif, k=nf)
    fsm.fit(lda_train, y_train)
    fsm_train = fsm.transform(lda_train)
    fsm_test = fsm.transform(lda_test)
    svm_model.fit(fsm_train, y_train)
    svm_pred = svm_model.predict(fsm_test)
    accuracy = accuracy_score(y_test, svm_pred)*100
    feature_number_svm_filter.append(nf)
    accuracies_svm_filter.append(accuracy)
```

```
max_6 = np.max(accuracies_svm_filter)
max_index_svm_filter = accuracies_svm_filter.index(max_6)
print("Max accuracy : " , max_6)
print("Index of max accuracy : " , max_index_svm_filter)
```

```
Max accuracy : 97.76357827476039
Index of max accuracy : 8
```

```
(markers, stemlines, baseline) = plt.stem(feature_number_svm_filter, accuracies_svm_filter , bottom = max_4)
plt.setp(stemlines, linestyle="-", color="white", linewidth=0.5 )
plt.plot(feature_number_svm_filter, accuracies_svm_filter)
plt.ylabel('Accuracies')
plt.xlabel('Number of Features')
```



Using the filtering method with the SVM model to choose the best k components, gave us the same optimum performance as the baseline optimum accuracy of SVM with LDA using 10 components. An accuracy of 97.8% was obtained.

5.2 WRAPPER METHOD USING FORWARD FEATURE ELIMINATION

GNB

```
gnb = GaussianNB()

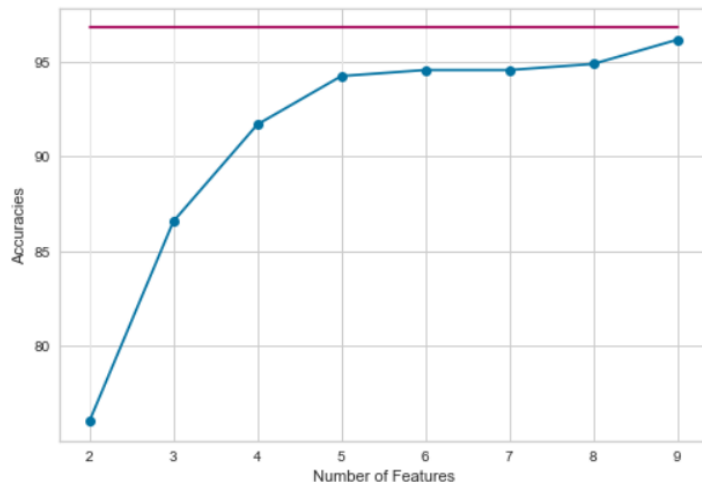
feature_number_gnb_wrapper = []
accuracies_gnb_wrapper = []

for nf in range(2,10):
    sfs = SequentialFeatureSelector(gnb, n_features_to_select=nf)
    sfs.fit(lda_train, y_train)
    sfs_train = sfs.transform(lda_train)
    sfs_test = sfs.transform(lda_test)
    gnb.fit(sfs_train, y_train)
    gnb_pred = gnb.predict(sfs_test)
    accuracy = accuracy_score(y_test, gnb_pred)*100
    feature_number_gnb_wrapper.append(nf)
    accuracies_gnb_wrapper.append(accuracy)

max_7 = np.max(accuracies_gnb_wrapper)
max_index_gnb_wrapper = accuracies_gnb_wrapper.index(max_7)
print("Max accuracy : ", max_7)
print("Index of max accuracy : ", max_index_gnb_wrapper)

Max accuracy : 96.1661341853035
Index of max accuracy : 7

(markers, stemlines, baseline) = plt.stem(feature_number_gnb_wrapper, accuracies_gnb_wrapper , bottom = max_3)
plt.setp(stemlines, linestyle="-", color="white", linewidth=0.5 )
plt.plot(feature_number_gnb_wrapper, accuracies_gnb_wrapper)
plt.ylabel('Accuracies')
plt.xlabel('Number of Features')
```



When the forward feature elimination method was used with the GNB model, it can be seen that it's optimum accuracy is 96.17% using 9 features, however the accuracy is lower than the baseline accuracy of the GNB model with LDA using 10 components.

SVM

```
svm_model = svm.SVC(random_state=0)

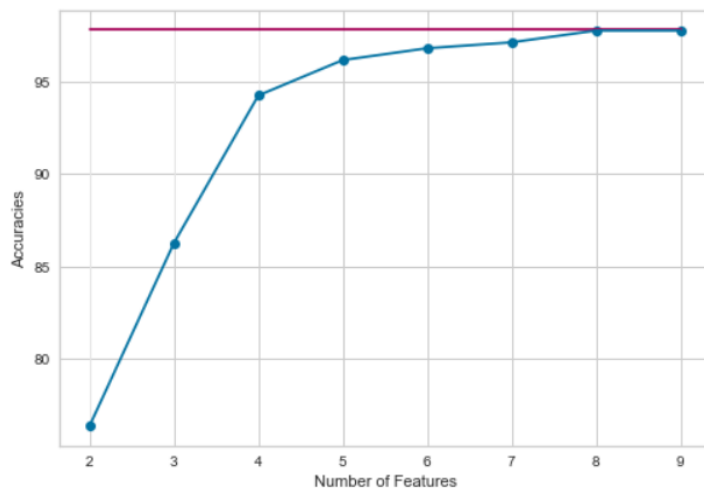
feature_number_svm_wrapper = []
accuracies_svm_wrapper = []

for nf in range(2,10):
    sfs = SequentialFeatureSelector(svm_model, n_features_to_select=nf)
    sfs.fit(lda_train, y_train)
    sfs_train = sfs.transform(lda_train)
    sfs_test = sfs.transform(lda_test)
    svm_model.fit(sfs_train, y_train)
    svm_pred = svm_model.predict(sfs_test)
    accuracy = accuracy_score(y_test, svm_pred)*100
    feature_number_svm_wrapper.append(nf)
    accuracies_svm_wrapper.append(accuracy)
```

```
max_8 = np.max(accuracies_svm_wrapper)
max_index_svm_wrapper = accuracies_svm_wrapper.index(max_8)
print("Max accuracy : " , max_8)
print("Index of max accuracy : " , max_index_svm_wrapper)
```

```
Max accuracy : 97.76357827476039
Index of max accuracy : 6
```

```
(markers, stemlines, baseline) = plt.stem(feature_number_svm_wrapper, accuracies_svm_wrapper , bottom = max_4)
plt.setp(stemlines, linestyle="-", color="white", linewidth=0.5 )
plt.plot(feature_number_svm_wrapper, accuracies_svm_wrapper)
plt.ylabel('Accuracies')
plt.xlabel('Number of Features')
```



However, it can be seen that when the forward feature elimination method was used with the SVM model, the optimum accuracy was equal to the baseline accuracy of SVM with LDA but with a fewer number of components – 8 components. It achieved an optimum accuracy of 97.8%.

6 KMEANS CLUSTERING AFTER DIMENSIONALITY REDUCTION

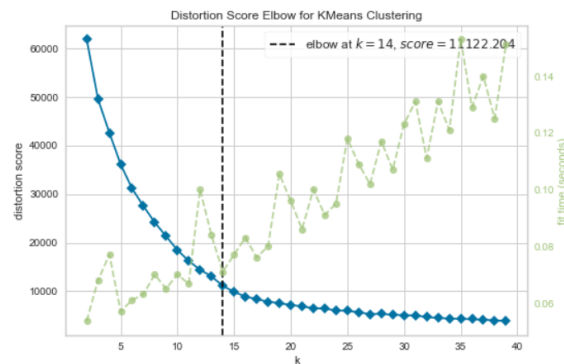
It can be concluded from the previous question that in order to achieve the optimum accuracy with least number of components, the dimensionality of the data should first be reduced to 10 components using LDA and then using forward features elimination with an SVM estimator, an accuracy of 97.8% can be obtained using 8 components.

When the elbow method is used on the transformed data, it can be seen that the optimum number of clusters is 14 clusters.

```
lda = LinearDiscriminantAnalysis(n_components=10)
lda.fit(X_train, y_train)
lda_train = lda.transform(X_train)

svm_model = svm.SVC(random_state=0)
sfs = SequentialFeatureSelector(svm_model, n_features_to_select=8)
sfs.fit(lda_train, y_train)
sfs_train = sfs.transform(lda_train)
sfs_test = sfs.transform(lda_test)
```

```
model = KMeans(random_state=0)
visualizer = KElbowVisualizer(model, k=(2,40),metric='distortion')
visualizer.fit(sfs_train) # Fit the data to the visualizer
visualizer.show()
```



7 SOM ALGORITHM

According to the elbow rule, the optimum number of neurons is 17 neurons, which is equal to the number of classes present in the data.

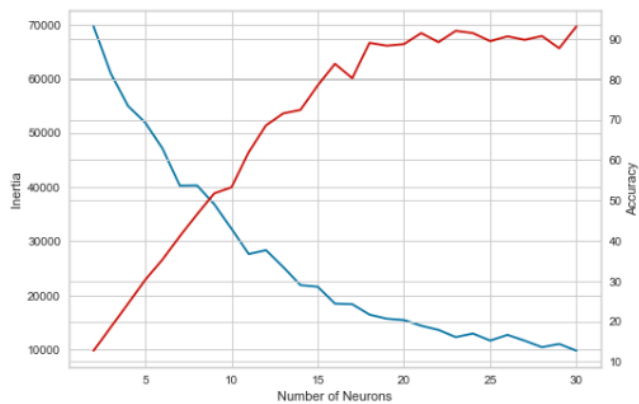
```
: wss = []
  accuracies = []

  for neurons in range(2,31):
      som_model = SOM(m=neurons,n=1,dim=8, random_state=0)
      som_labels = som_model.fit_predict(sfs_train)
      predY = usLabels2sLabels(som_labels, y_train)
      accuracy = accuracy_score(y_train, predY)*100
      accuracies.append(accuracy)
      inertia = som_model.inertia_
      wss.append(inertia)

  fig, ax1 = plt.subplots()
  ax2 = ax1.twinx()
  ax1.plot(range(2,31), wss, c="b")
  ax2.plot(range(2,31), accuracies, c="r")

  ax1.set_xlabel('Number of Neurons')
  ax1.set_ylabel('Inertia')
  ax2.set_ylabel('Accuracy')
```

```
: Text(0, 0.5, 'Accuracy')
```



Using the Self Organizing Feature-Map algorithm, we initially build a 17x17 matrix to initialize the weights of the neurons randomly. Before training our SOM model on the data, the initial positions of the neurons with random weights are plotted along with the data points. The gradient color in the background represents the clustering density measure of the neurons or proximity of each neuron to the data points. Initially it can be seen that the neurons are randomly positioned, hence the data points are scattered along the bone diagram.

```
: som = MiniSom(17, 17, sfs_train.shape[1], random_seed=0)
initial_weights = som.random_weights_init(sfs_train)

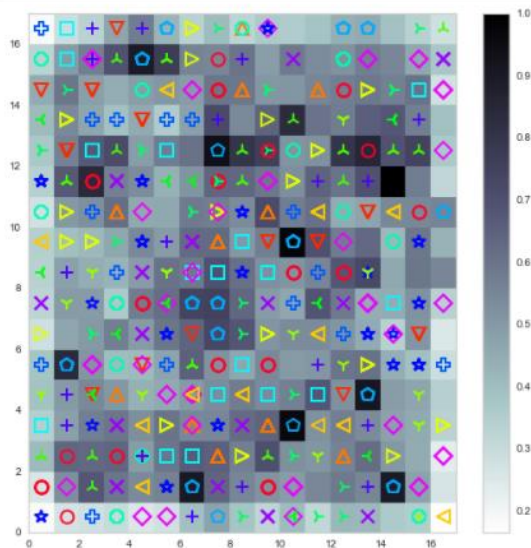
: plt.figure(figsize=(9, 9))

plt.pcolor(som.distance_map().T, cmap='bone_r') # plotting the distance map as background
plt.colorbar()

target_lbl = np.unique(train_df['type1'])
NUM_COLORS = 17
cm = plt.get_cmap('gist_rainbow')
colors = [cm(1.*i/NUM_COLORS) for i in range(NUM_COLORS)]
markers = ["o", "v", "^", "<", ">", "1", "2", "3", "4", "8", "s", "p", "P", "=", "+", "x", "D"]

for index, xx in enumerate(sfs_train):
    w = som.winner(xx)
    plt.plot(w[0]+.5, w[1]+.5, markers[y_train[index]], markerfacecolor='None', markeredgecolor=colors[y_train[index]], markersize=100)

plt.show()
```



After training the model on the data for 4000 iterations, the weights of the neurons have been updated and the positions of the neurons have changed and it can be seen that the data has become clustered around certain neurons. The white cells indicate cluster centers.

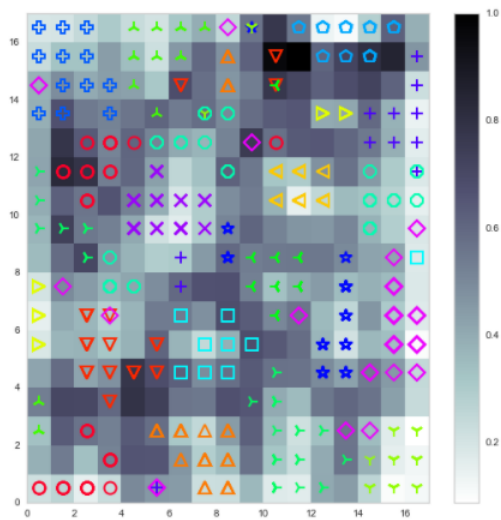
```
: plt.figure(figsize=(9, 9))

plt.pcolor(som.distance_map().T, cmap='bone_r') # plotting the distance map as background
plt.colorbar()

target_lbl = np.unique(train_df['type1'])
NUM_COLORS = 17
cm = plt.get_cmap('gist_rainbow')
colors = [cm(1.*i/NUM_COLORS) for i in range(NUM_COLORS)]
markers = ["o", "v", "A", "<", ">", "1", "2", "3", "4", "8", "s", "p", "P", "*", "+", "x", "D"]

som.train(sfs_train, 4000)
for index, xx in enumerate(sfs_train):
    w = som.winner(xx)
    plt.plot(w[0]+.5, w[1]+.5, markers[y_train[index]], markerfacecolor='None', markeredgecolor=colors[y_train[index]], markersize=100)

plt.show()
```



8 DBSCAN ALGORITHM

```
: epslist, mslist, acclist, n_cluster = list(), list(), list(), list()
for eps in tqdm(np.arange(.2, 3.1, 0.05)):
    for ms in range(2, 16):
        model = DBSCAN(eps=eps, min_samples=ms)
        predClusters = model.fit_predict(sfs_train)
        predY = usLabels2sLabels(predClusters, y_train)
        accuracy = accuracy_score(y_train, predY)*100
        n_clusters_ = len(set(predClusters)) - (1 if -1 in labels else 0)
        n_cluster.append(n_clusters_)
        epslist.append(eps)
        mslist.append(ms)
        acclist.append(accuracy)

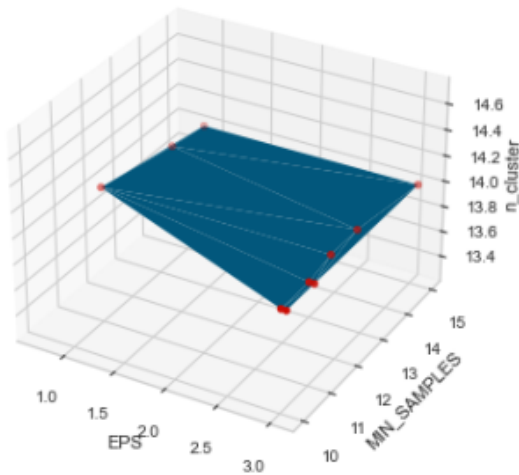
epslist, mslist, acclist, n_cluster = np.array(epslist), np.array(mslist), np.array(acclist), np.array(n_cluster)

i = np.where(n_cluster==14)
max_index = np.argmax(acclist[i], -10)[-10:]

# xx, yy = np.meshgrid(range(epslist.shape[0]), range(mslist.shape[0]))
# zz = 14
# ax.plot_surface(xx, yy, np.atleast_2d(zz), color="r")

ax = plt.axes(projection='3d')
ax.plot_trisurf(epslist[i][max_index], mslist[i][max_index], n_cluster[i][max_index])
ax.scatter(epslist[i][max_index], mslist[i][max_index], n_cluster[i][max_index], color="r")
ax.set_xlabel('EPS')
ax.set_ylabel('MIN_SAMPLES')
ax.set_zlabel('n_cluster')
plt.show()

print("Number of Clusters: ", n_cluster[i][max_index])
print("Epsilon: ", epslist[i][max_index])
print("Minpoint: ", mslist[i][max_index])
print("Accuracy: ", acclist[i][max_index])
```



```
Number of Clusters: [14 14 14 14 14 14 14 14 14 14]
Epsilon: [0.7 0.85 0.9 3. 3.05 3.05 3. 3.05 2.95 2.95]
Minpoint: [12 14 15 10 10 11 11 15 13 12]
Accuracy: [33.57314149 40.76738609 41.48681055 68.58513189 68.58513189 68.58513189
68.58513189 72.58193445 74.10071942 74.02078337]
```

The red points of the 3D plot indicate the 10 combinations of minpoint and epsilon values that give us 14 clusters with the highest accuracies, according to question 6.