uOttawa

Faculté de génie
Faculty of Engineering

Dimensionality Reduction Assignment 3

Applied Machine Learning ELG5255[EG]

Group name: Group 6

Ahmed Fares Saad Eldin Khalifa

Abdallah Medhat Mohamed Rashed

Contents

1. Load Iris Dataset

```
[ ]  pokemon_train = pd.read_csv("Pokemon_train.csv")
     pokemon_test= pd.read_csv("Pokemon_test.csv")
```

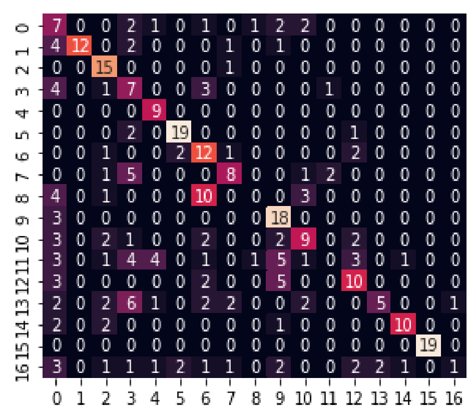
```
[ ]  x_pokemon_train=pokemon_train.iloc[:,:-1].to_numpy()
     y_pokemon_train=pokemon_train.iloc[:,-1]
```

```
[ ]  x_pokemon_test=pokemon_test.iloc[:,:-1].to_numpy()
     y_pokemon_test=pokemon_test.iloc[:,-1]
```

Figure 1: loading the dataset

2. Apply Gaussian Naïve Bayes classifier (GNB) and Support Vector Machine (SVM) to Pokémon dataset

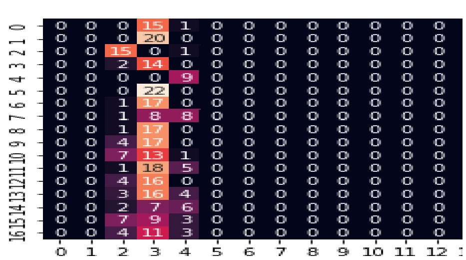
Confusion Matrix_Gaussian Naïve Bayes:



```
naive_clf = GaussianNB()
naive_clf.fit(x_pokemon_train,y_pokemon_train)
y_pred_naive=naive_clf.predict(x_pokemon_test)
#Making the Confusion Matrix and Classification Report
acc_GNB1=naive_clf.score(x_pokemon_test,y_pokemon_test)*100
print("Accuracy for Gaussian Naïve Bayes  in testing data: ",acc_GNB1)
print('\nClassification Report_Gaussian Naïve Bayes:\n')
print(classification_report(y_pokemon_test,y_pred_naive))
print('Confusion Matrix_Gaussian Naïve Bayes:\n')
hm=sn.heatmap(confusion_matrix(y_pokemon_test, y_pred_naive), annot=True)
plt.show()
```

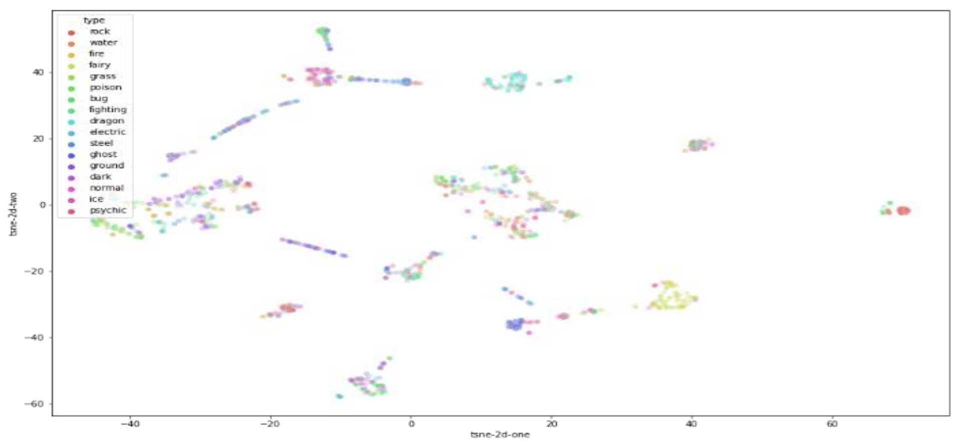
Accuracy for Gaussian Naïve Bayes in testing data: 51.43769968051119

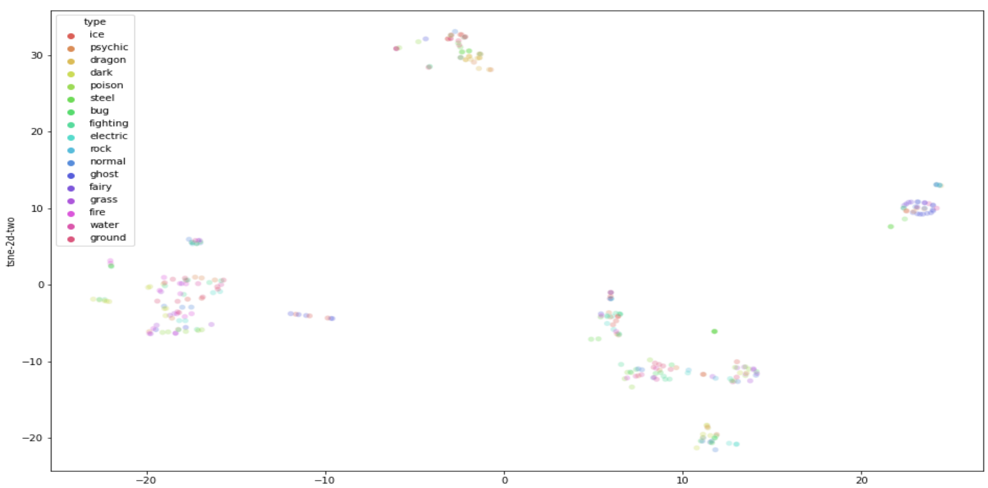
Confusion MatrixSVM:



```
svm_clf= SVC(kernel='rbf',random_state=0)
svm_clf.fit(x_pokemon_train,y_pokemon_train)
# Predicting the Test set results
y_pred_svm=svm_clf.predict(x_pokemon_test)
#Making the Confusion Matrix and Classification
acc_SVC1=svm_clf.score(x_pokemon_test,y_pokemon_
print("Accuracy for model_SVM:  ",acc_SVC1)
print('\nClassification Report_SVM:\n')
print(classification_report(y_pokemon_test,y_pr
print('Confusion MatrixSVM:\n')
# plot_confusion_matrix(svm_clf,x_pokemon_test,
hm=sn.heatmap(confusion_matrix(y_pokemon_test,y_
plt.show()
```

```python
x_tsne_train=TSNE(n_components=2, random_state=0).fit_transform(x_pokemon_train)
x_tsne_train
pokemon_train_ts=pd.DataFrame()
pokemon_train_ts['tsne-2d-one'] = x_tsne_train[:,0]
pokemon_train_ts['tsne-2d-two'] = x_tsne_train[:,1]
pokemon_train_ts['type']=y_pokemon_train
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="type",
    palette=sns.color_palette("hls", 17),
    data=pokemon_train_ts,
    legend="full",
    alpha=0.3
)
```

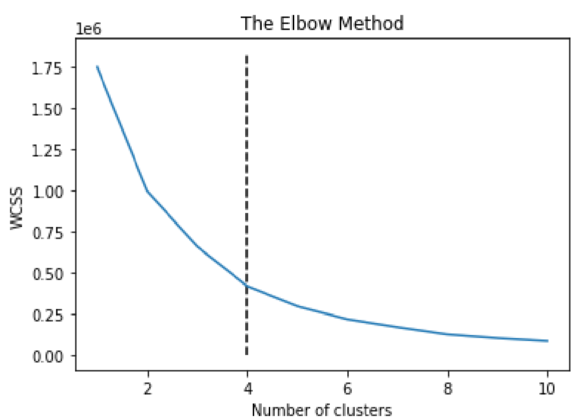


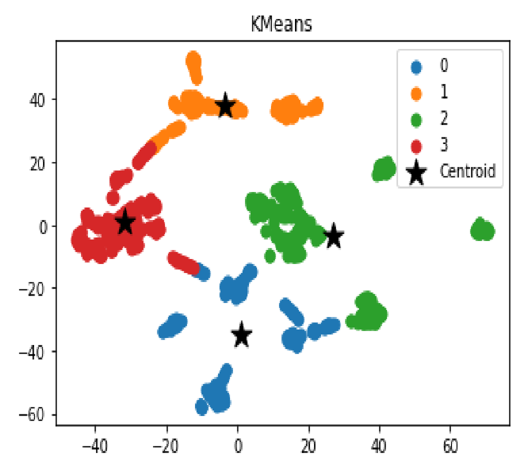


After applying the Gaussian Naïve Bayes classifier and Support Vector Machine that we know the Gaussian Naïve Bayes (51.44%) is better than Support Vector Machine (12.14%) and applying TSNE on training and testing data to visualize data with number of components= 2.

3. Choose the best number of cluster for k-means clustering algorithm

```
inertia1 = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 0)
    kmeans.fit(x_train_tsne)
    inertia1.append(kmeans.inertia_)
kn1 = KneeLocator(range(1,11), inertia1, curve='convex', direction='decreasing')
plt.plot(range(1, 11), inertia1)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.vlines(kn1.knee, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
plt.show()
```



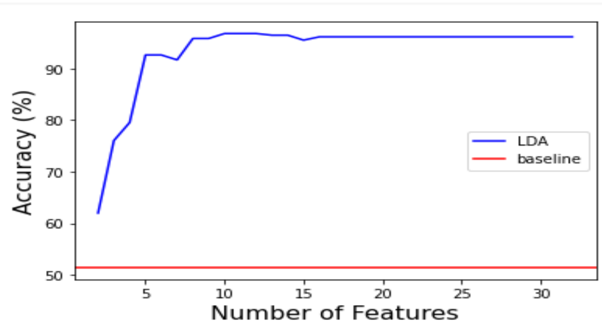


After applying k-means with elbow method on the dataset to Determine the optimal number of clusters. Optimal number of clusters =4. Then plotting data with the optimal number of clusters (4).

4. Apply one of the following Dimensionality Reduction (DR) methods to data. Find the best value for n_components based on the GNB and SVM classifiers test accuracies.

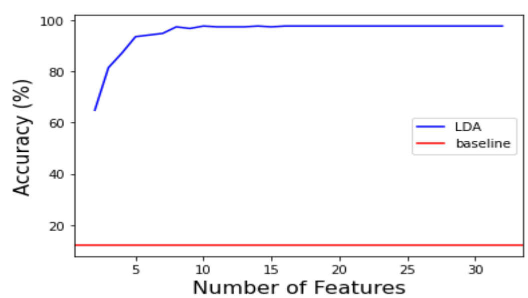
```
# Models
SVC_model=SVC(kernel='rbf',random_state=0)
GNB_model=GaussianNB()
```

```
[ ]  def apply_LDA(model,baseModel_acc):
         model_scores=[]
         for i in range(1,33):
           LDA = LinearDiscriminantAnalysis(n_components=i)
           x_train_lda = LDA.fit_transform(x_pokemon_train,y_pokemon_train)
           x_test_lda = LDA.transform(x_pokemon_test)
           model.fit(x_train_lda,y_pokemon_train)
           model_scores.append(model.score(x_test_lda,y_pokemon_test)*100)
         max_accuracy=max(model_scores)
         index_best_com = model_scores.index(max_accuracy)+1
         # Plot a simple line chart
         plt.plot(range(1,33), model_scores, 'b', label='LDA')
         # Plot another line on the same chart/graph
         plt.axhline( y=baseModel_acc,c='r', label='baseline')
         plt.xlabel("Number of Features", fontsize=16)
         plt.ylabel("Accuracy (%)", fontsize=16)
         plt.legend()
         plt.show()
         return index_best_com,max_accuracy
```




```
Maximum accuracy: 96.8%
Best number of features: 10
```

Figure 2: LDA with GNB




```
Maximum accuracy: 97.8%
Best number of features: 10
```

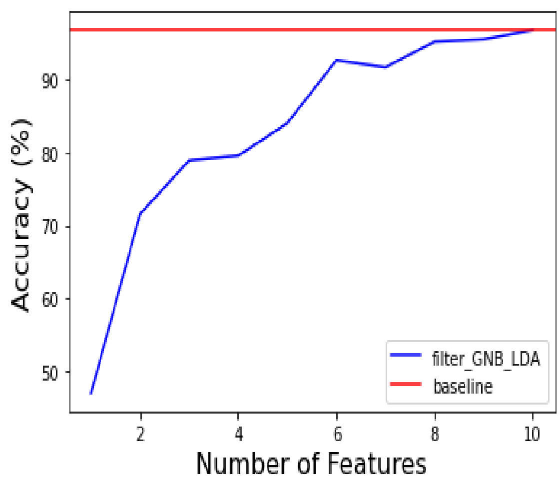
Figure 3: LDA with SVM

We use these parameters with the models because that affect to our accuracy better and after applying LDA on the Pokémon dataset with GNB and SVM classifiers. We get the best accuracy = 97.8% and number of features = 10 in LDA with SVM.

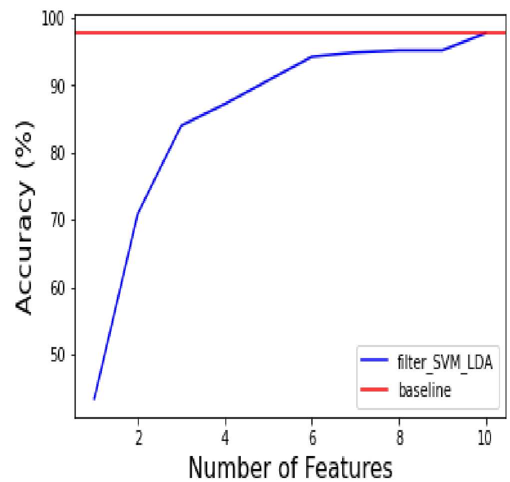
5. Use the following Feature Selection methods (one for each method). Find the best number of features based on the GNB and SVM classifiers' test accuracies on the data that is obtained after Q4. Plot the number of features versus accuracy graph for each case with the improved baseline performance as shown in Q4.

```
# Data after transformation with 10 features
LDA = LinearDiscriminantAnalysis(n_components=10)
x_train_lda = LDA.fit_transform(x_pokemon_train,y_pokemon_train)
x_test_lda = LDA.transform(x_pokemon_test)
```

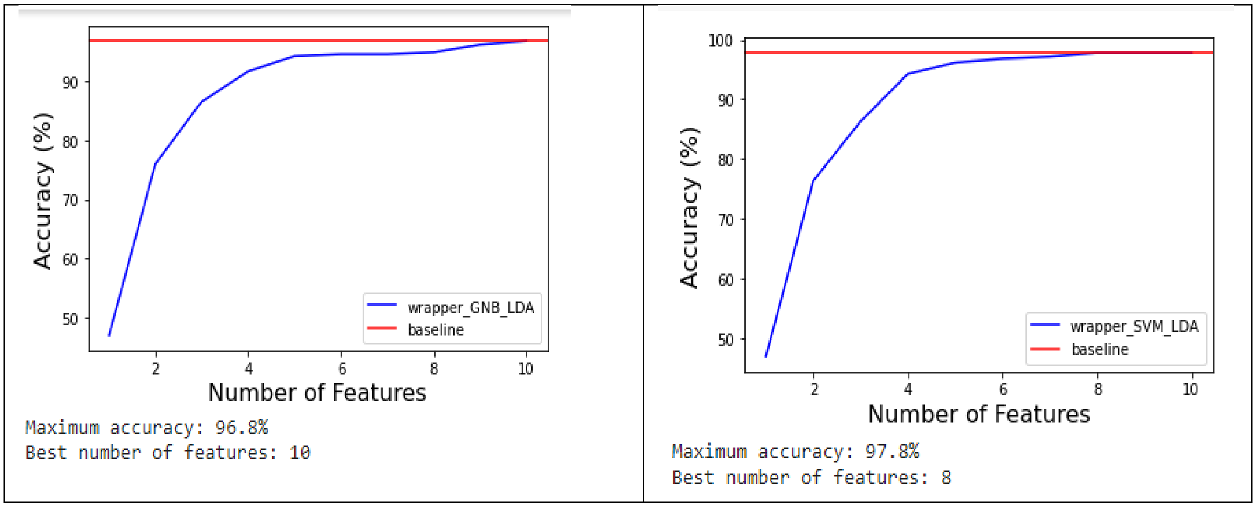
```
def select_feature(fs, model):
    fs.fit(x_train_lda, y_pokemon_train)
    train_new = fs.transform(x_train_lda)
    test_new = fs.transform(x_test_lda)
    model.fit(train_new, y_pokemon_train)
    yPred = model.predict(test_new)
    acc = accuracy_score(y_pokemon_test, yPred) * 100
    return acc
```




```
Maximum accuracy: 96.8%
Best number of features: 10
```

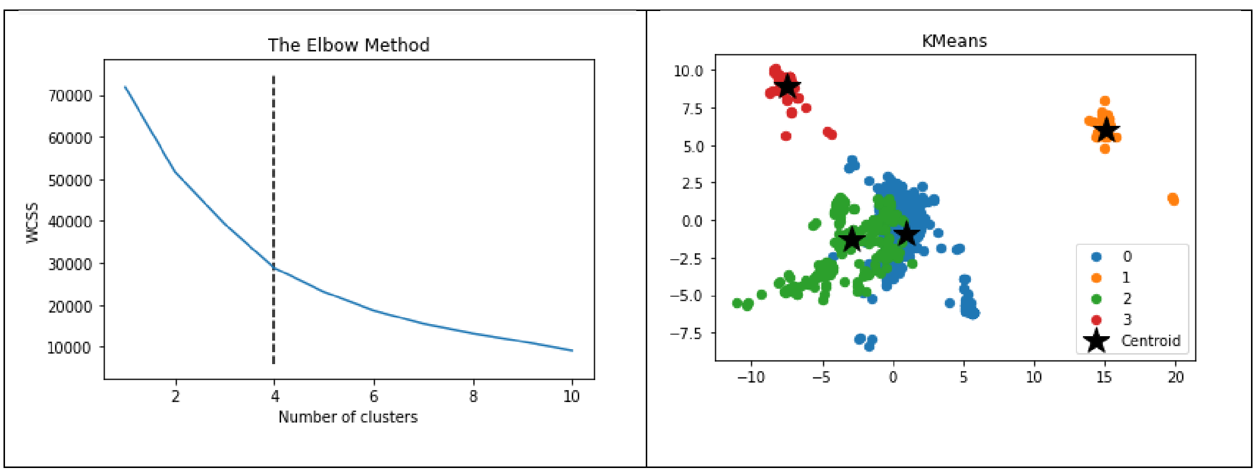



```
Maximum accuracy: 97.8%
Best number of features: 10
```



Applying Filter Method (Information Gain) and Wrapper Method (Forward) with the data from LDA, GNB and SVM classifiers. We get the best accuracy = 97.8% and the number of features = 8 when using wrapper method with LDA and SVM.

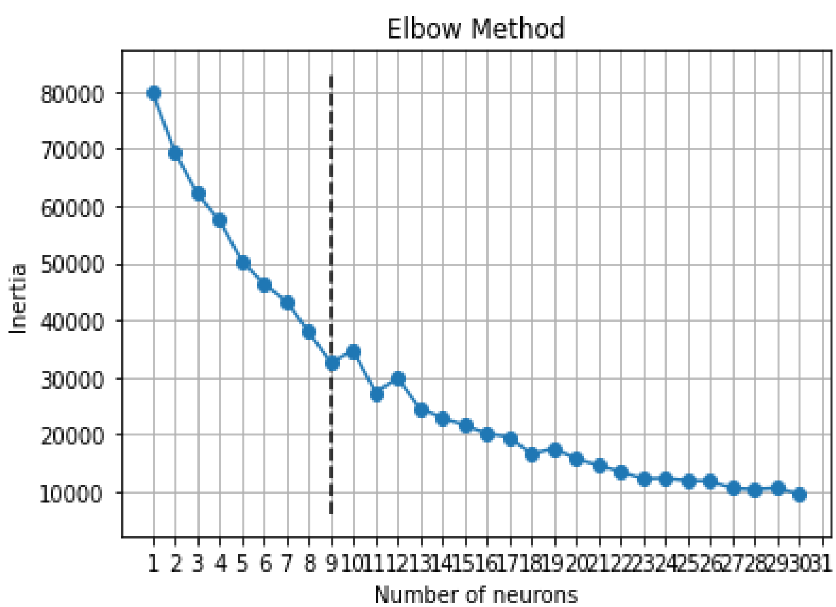
6. Choose the best number of clusters for k-means clustering algorithm on the processed data (using the best features or dimensionality from Q4 and Q5)



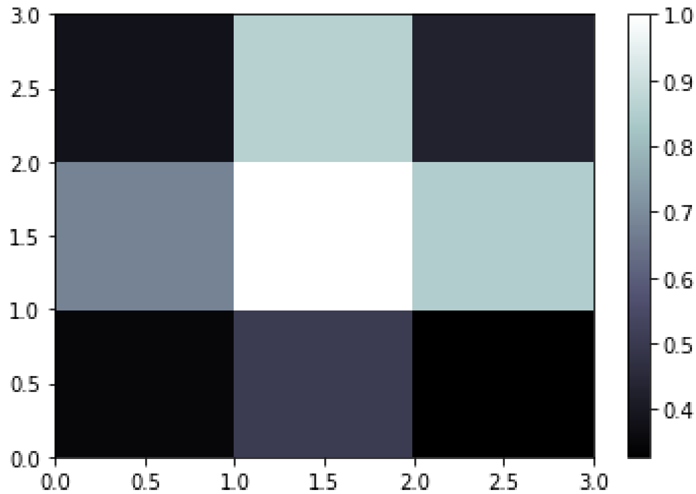
After applying LDA in Q4 and filter and wrapper method in Q5 that we get the best accuracy (97.8%) with the least number of feature (8) in wrapper method with LDA and SVM. After applying k-means on the dataset from wrapper method with LDA and SVM and using elbow method to determine the best number of cluster (4).

7. Choose the best number of neurons for SOM algorithm using the best features or dimensionality from Q4 and Q5

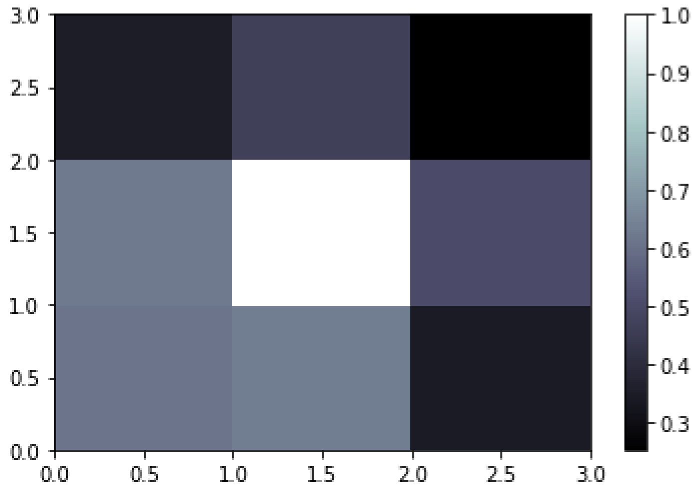
```
score = []
df_som = pd.DataFrame(test_sfs_SVM)
for i in range(1,31):
  SOM_model = SOM(m=i, n=1, dim=train_sfs_SVM.shape[1],max_iter=3000,random_sta
  SOM_model.fit(train_sfs_SVM)
  SOM_lables=SOM_model.predict(test_sfs_SVM)
  score.append((SOM_model.inertia_,i))
df_som['SOM_lables'] = SOM_lables
df_score = pd.DataFrame(score, columns=['score','neurons'])
kn3 = KneeLocator(range(1,31), df_score['score'], curve='convex', direction='de
plt.plot(df_score['neurons'], df_score['score'], marker = 'o')
plt.title("Elbow Method")
plt.xlabel("Number of neurons")
plt.ylabel("Inertia")
plt.xticks([i for i in range(1,33)])
plt.grid(True)
plt.vlines(kn3.knee, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
plt.show()
```



```
np.random.seed(43)
som_rows=3
som_columns=3
som_iterations=3000
sigma=1
learning_rate=0.5
np.random.seed(42)
som_init = MiniSom(x = som_rows, y = som_columns, input_len=8, sigma=sigma, lea
som_init.random_weights_init(train_sfs_SVM)
bone()
pcolor(som_init.distance_map().T)
colorbar()
show()
```




```
np.random.seed(42)
som_final=MiniSom(x = som_rows, y = som_columns, input_len=8, sigma=sigma, learn
som_final.train_random(train_sfs_SVM,som_iterations)
som_final.distance_map()
bone()
pcolor(som_final.distance_map().T)
colorbar()
show()
```



After applying LDA in Q4 and filter and wrapper method in Q5 that we get the best accuracy (97.8%) with the least number of feature (8) in wrapper method with LDA and SVM. After applying the SOM on the dataset from wrapper method with LDA and SVM and using elbow rule by inertia to determine the best number of neurons (9) then plot neurons in the initial (before training) and final positions (after training) on the shape gird 3*3 because number of neurons = 9.

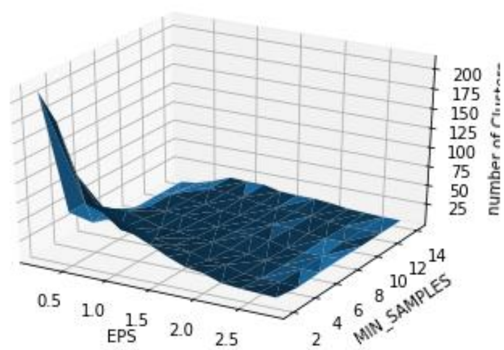
8. Tune the epsilon (0.2-3) and minpoints (2-15) values in the given intervals to obtain same number of clusters in Q6 by using DBSCAN

```
] def unsupervisedLabelMap(labels, y):
      labelDict = dict()
      for label in unique_labels(labels):
          tmpY = y[labels == label]
          unique, count = np.unique(tmpY, return_counts=True)
          trueLabel = unique[np.argmax(count)]
          labelDict[label] = trueLabel
      return labelDict
  def usLabels2sLabels(labels, y):
      sLabels = np.empty(labels.shape, labels.dtype)
      labelDict = unsupervisedLabelMap(labels, y)
      for usl, tl in labelDict.items():
          sLabels[labels == usl] = tl
      return sLabels
  encoder = LabelEncoder()
  encoder_train_y = encoder.fit_transform(y_pokemon_train)
```

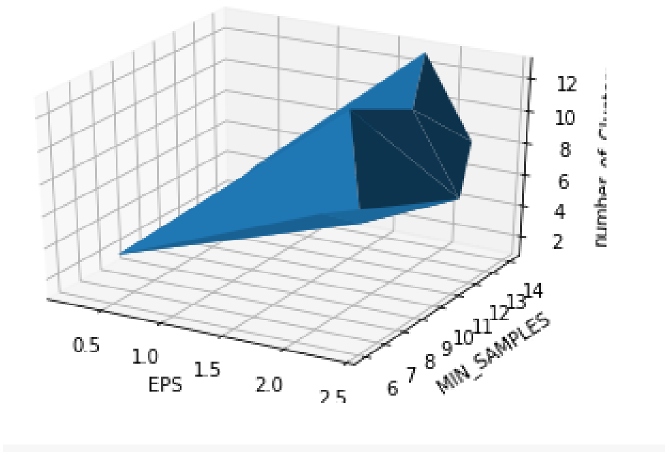
s

```
epsList, mpList, clustersList, accuracyList,noiselist = list(), list(), list(), list(),list()
for eps in tqdm(np.arange(0.2, 3,0.2)):
        for mp in range(2, 15,2):
            model = DBSCAN(eps=eps, min_samples=mp)
            y_pred_db = model.fit_predict(train_sfs_SVM)
            labels = model.labels_
            n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
            n_noise_ = list(labels).count(-1)
            predY = usLabels2sLabels(y_pred_db, encoder_train_y)
            accuracy = accuracy_score(encoder_train_y, predY)
            epsList.append(eps)
            mpList.append(mp)
            clustersList.append(n_clusters_)
            accuracyList.append(accuracy)
            noiselist.append(n_noise_)
epsList, mpList, accuracyList, clustersList,noiselist = np.array(epsList), np.array(mpList),
     np.array(accuracyList), np.array(clustersList),np.array(noiselist)
ax = plt.axes(projection='3d')
ax.plot_trisurf(epsList, mpList, clustersList)
ax.set_xlabel('EPS')
ax.set_ylabel('MIN_SAMPLES')
ax.set_zlabel('number of Clusters')
plt.show()
x = accuracyList.argmax()
print(epsList[x], mpList[x], accuracyList[x],noiselist[x],clustersList[x])
```

```
100%|██████████| 14/14 [00:01<00:00,  8.12it/s]
```




```
0.8 2 0.8984812150279776 104 90
```



	epsilon	minpoint	number of cluster
6	0.2	14	1
3	0.2	8	2
13	0.4	14	4
95	2.8	10	6
81	2.4	10	7
12	0.4	12	8
71	2.2	4	9
69	2.0	14	11
11	0.4	10	12
54	1.6	12	13

After applying tune, the epsilon (0.2-3) and minpoints (2-15) values in DBSCAN, plotting the epsilon, minpoints and the number of clusters in 3D figure, we know the best epsilon (0.4) and minpoints (14) to obtain the same number of clusters = 4 in (Q6). Showing the 10 combinations from epsilon and minpoints with number of clusters in dataframe to be organized and plotting the 10 combinations from epsilon and minpoints in 3D figure.

9. Conclusion

In conclusion, this report could be summarized by loading the data from Pokémon dataset, after we applied the GNB and SVM classifiers and k-means, then applied TSNE to visualize data and LDA with the GNB and SVM classifiers. After applied LDA then applied filter and wrapper methods based on data from LDA. Then we applied k-means and SOM with LDA and wrapper method because the accuracy of wrapper with LDA is the best = 98.1% and the least of number of features = 5 and we determined the optimal number of clusters in k-means and number of neurons in SOM by elbow. We applied tuning the epsilon and minpoints in DBSCAN to obtain the same number of clusters in k-means (Q6) and plotting the epsilon, minpoints and number of clusters in 3D figure.