```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.svm import SVC


#MLP and SVC on kmeans data or the original data
```

```
data = pd.read_csv('falldetection_dataset.csv', header = None)
data = data.drop(0, axis = 1)
data = data.replace('F', 1)
data = data.replace('NF', 0)
labels = data[1]
cl = ['NF','F']
data = data.drop(1, axis = 1)
data.head()
```

	2	3	4	5	6	7	8	
0	-1.444006	51.897025	9.051206	39.154050	4.861414	30.582530	26.361643	5.0308
1	-2.336273	35.644388	6.443654	24.827069	2.555905	14.351492	17.849532	10.0188
2	-3.160453	40.378218	6.126165	25.891205	3.261484	21.788334	16.620108	11.2520
3	-2.991333	44.093847	6.691918	28.082497	4.566522	31.905741	16.683106	6.0051
4	-3.079600	45.901880	6.674186	32.691078	4.156527	26.843041	21.150885	7.7253

5 rows × 306 columns

```
scaler = StandardScaler()
data_norm = scaler.fit(data).transform(data)
pca = PCA()
pca.fit(data_norm)
```

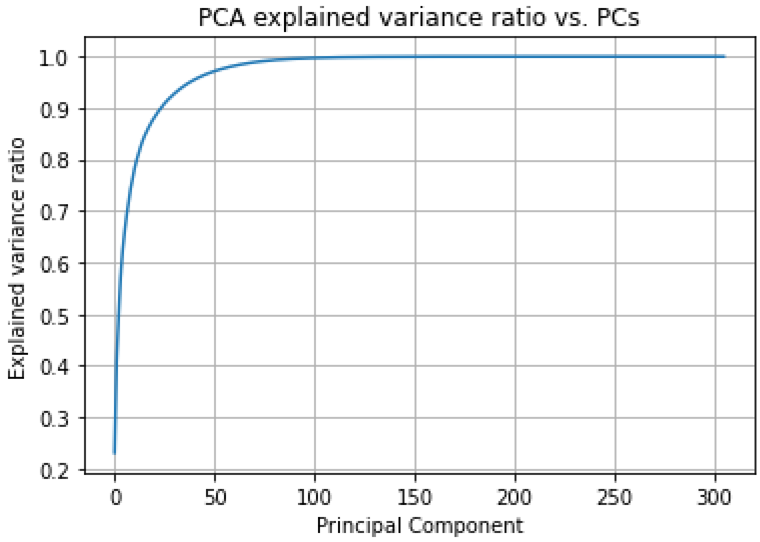
```
PCA()
```

```
# data_train, data_test, labels_train, labels_test = train_test_split(data_norm,
# data_train, data_val, labels_train, labels_val = train_test_split(data_train,

```

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Principal Component')
plt.ylabel('Explained variance ratio')
plt.title('PCA explained variance ratio vs. PCs')
```

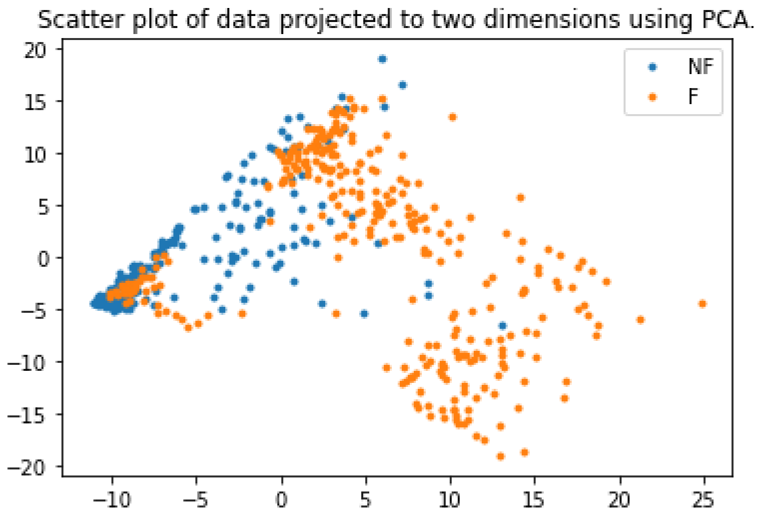
```
plt.grid()
print(pca.explained_variance_ratio_[0],pca.explained_variance_ratio_[1])
```

```
0.23101268308439282 0.17624389973531043
```




```
pca2 = PCA(2)
data_pca = pca2.fit_transform(data_norm)
```

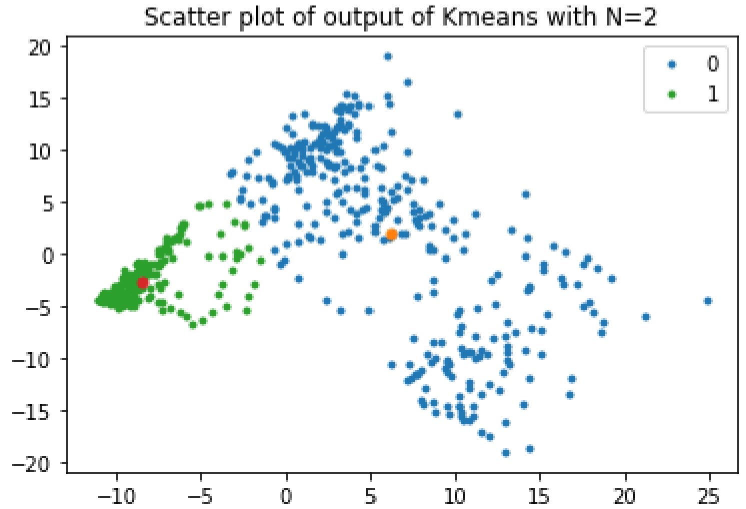
```
for i in range(2):
  ind = np.where(labels == i)[0]
  plt.plot(data_pca[ind,0],data_pca[ind,1], 'o',label = cl[i],markersize=3)

plt.title('Scatter plot of data projected to two dimensions using PCA.')
plt.legend()
plt.show()
```



Kmeans

```
from sklearn.cluster import KMeans
```

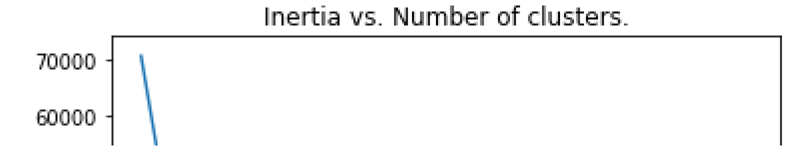
```
N = 2
kmeans = KMeans(n_clusters=N).fit(data_pca) #https://scikit-learn.org/stable/mod
for i in range(N):
  ind = np.where(kmeans.labels_ == i)[0]

  plt.plot(data_pca[ind,0],data_pca[ind,1], 'o',label = i, markersize = 3)
  plt.plot(kmeans.cluster_centers_[i,0],kmeans.cluster_centers_[i,1],'o', marker


plt.title('Scatter plot of output of Kmeans with N=2')
plt.legend()
plt.show()
sum(kmeans.labels_ == labels.to_numpy())/len(labels)
```




```
0.1872791519434629
```

```
j = []
for N1 in range(1,21):
  kmeans1 = KMeans(n_clusters=N1).fit(data_pca)
  j.append(kmeans1.inertia_)
```

```
plt.plot(np.arange(1,21),j)
plt.title('Inertia vs. Number of clusters.')
plt.xlabel('number of clusters')
plt.ylabel('Inertia')
plt.show()
```

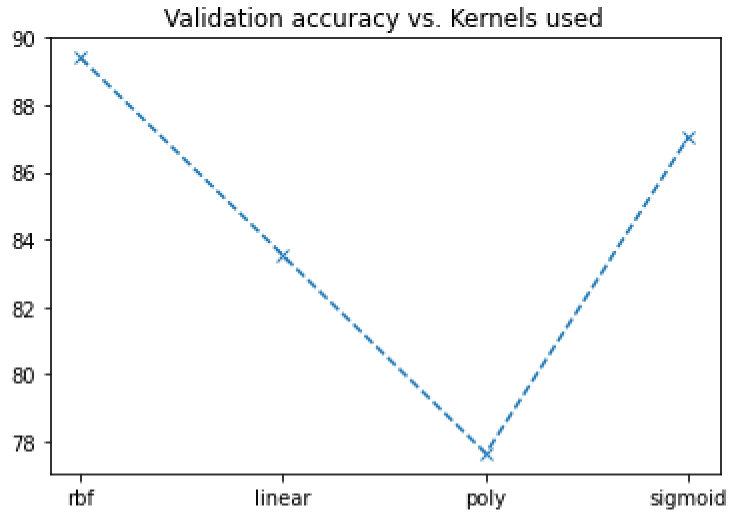


PCA Data

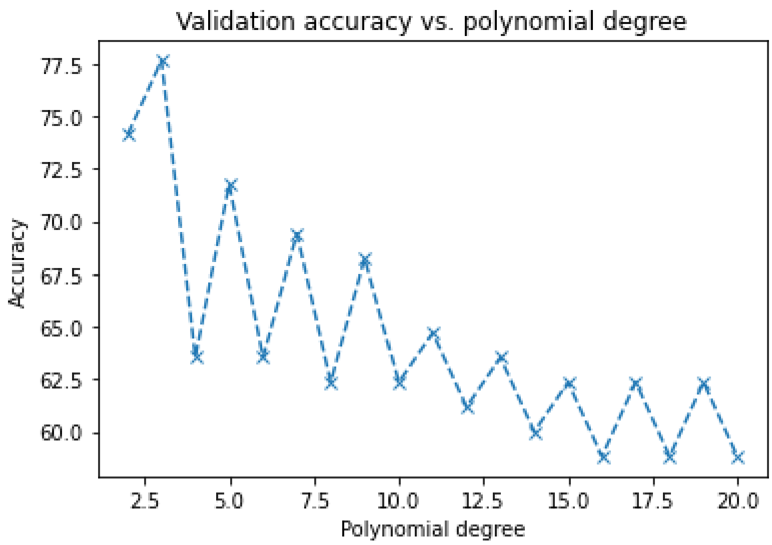
SVM

```
data_train, data_test, labels_train, labels_test = train_test_split(data_pca, la
data_train, data_val, labels_train, labels_val = train_test_split(data_train, la
```

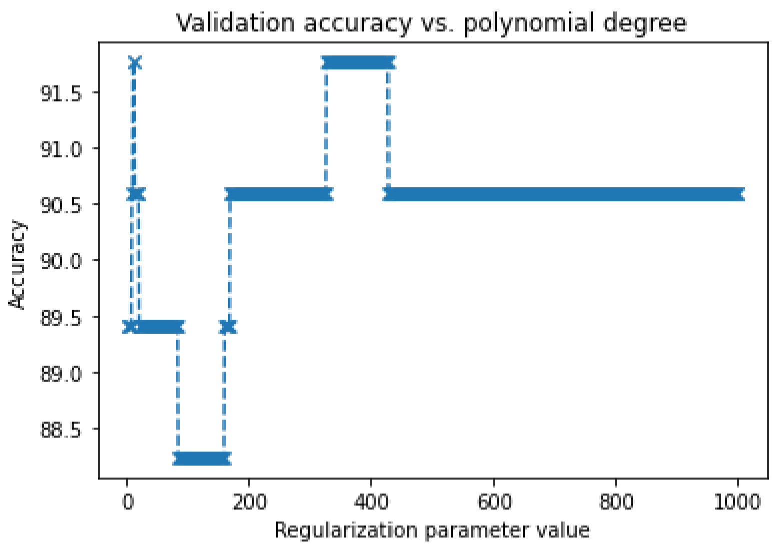
```
kernels = ['rbf','linear', 'poly',  'sigmoid']
score_list = []
for k in kernels:
  svm = SVC(kernel = k)
  svm.fit(data_train, labels_train)
  score_list.append(svm.score(data_val, labels_val)*100)
plt.plot(kernels, score_list, 'x--')
plt.title('Validation accuracy vs. Kernels used')
plt.show()
```




```
dim = np.arange(2,21)
score_list1 = []
for k in dim:
  svm = SVC(kernel = "poly", degree = k, C=1)
  svm.fit(data_train, labels_train)
  score_list1.append(svm.score(data_val, labels_val)*100)
plt.plot(dim, score_list1, 'x--')
plt.title('Validation accuracy vs. polynomial degree')
plt.xlabel('Polynomial degree')
plt.ylabel('Accuracy')
plt.show()
```




```
dim = np.arange(1,1000)
score_list1 = []
for k in dim:
  svm = SVC(kernel = "rbf",  C=k)
  svm.fit(data_train, labels_train)
  score_list1.append(svm.score(data_val, labels_val)*100)
plt.plot(dim, score_list1, 'x--')
plt.title('Validation accuracy vs. polynomial degree')
plt.xlabel('Regularization parameter value')
plt.ylabel('Accuracy')
plt.show()
```




```
svm = SVC(kernel = "rbf",  C=350)
svm.fit(data_train, labels_train)
score_list1.append(svm.score(data_val, labels_val)*100)
score_list1[-1]
```

```
91.76470588235294
```

```
# Best model is the rbf model or the poly w/ deg 3
svm = SVC(kernel = "rbf",  C=350)
svm.fit(data_train, labels_train)
```

```
score_lin = svm.score(data_test, labels_test)
score_lin = score_lin*100
print(score_lin)


# svm = SVC(kernel = "poly", degree = 3)
# svm.fit(data_train, labels_train)
# score_poly = svm.score(data_test, labels_test)
# score_poly = score_poly * 100
# print(score_poly)
```

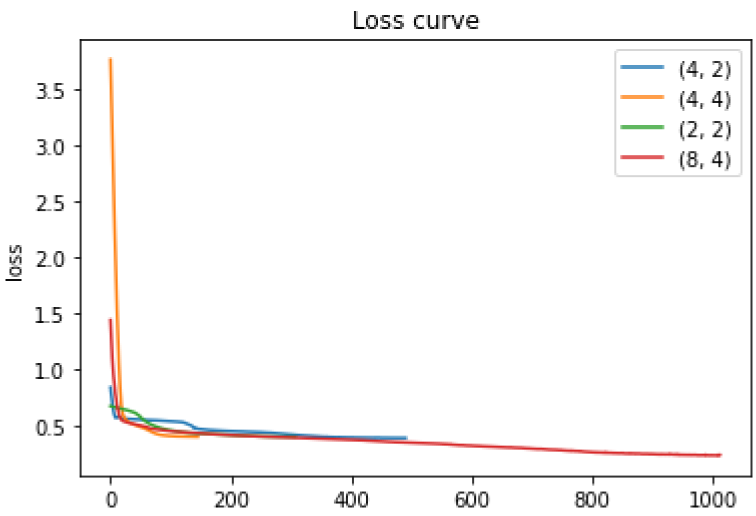
```
87.05882352941177
```

MLP

```
from sklearn.neural_network import MLPClassifier #https://scikit-learn.org/stabl
```

```
layers = [(4,2),(4,4),(2,2),(8,4)]
mlp_pca = []
mlp_loss = []

for l in layers:
  clf = MLPClassifier(hidden_layer_sizes=l, activation = 'relu', max_iter = 5000
                      solver='adam',learning_rate_init = 0.005,alpha = 0.0001,
                      shuffle = True, momentum = 0.9, learning_rate = 'adaptive'
  print(clf.score(data_val, labels_val))
  mlp_pca.append(clf.score(data_val, labels_val))
  mlp_loss.append(clf.loss_curve_)
  plt.plot(clf.loss_curve_, label = str(l))
plt.title('Loss curve')
plt.ylabel('loss')
plt.legend()
plt.show()
```

```
0.9058823529411765
0.8823529411764706
0.8823529411764706
0.8941176470588236
```



```
print(clf.score(data_test, labels_test))
```

```
0.8823529411764706
```

Original Data

```
data_train, data_test, labels_train, labels_test = train_test_split(data_norm, l
data_train, data_val, labels_train, labels_val = train_test_split(data_train, la
```

SVM

[ ] ↳ 3 cells hidden

MLP

[ ] ↳ 5 cells hidden

✓ 0s completed at 12:19 PM