Machine learning implementations on different datasets using python

Submitted by: Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

Problem formulation

Electricity data set	Olivetti Dataset
• This is classification problem of electrical grid stability into two classes; • 1.Stable • 2.Unstable	• Recognizing face with highest accuracy using machine learning algorithms • Comparison of their accuracies to select the most fitted face recognition algorithm.

About Electricity Data Set

- Electrical Grid data set in which we have different attributes for examine the stability of the system.

- We will examine the response of the system stability depends on 10,000 observations and 13 attributes, 1 classes attribute (stabf). Attributes are given in dataset as;

- tau[x] : Reaction time of participant (real from the range [0.5,10]s).
- p[x] : Nominal power consumed(negative)/produced(positive)(real).
- g[x] : Coefficient (gamma) proportional to price elasticity (real from the range [0.05,1]s^-1).
- stab: The maximal real part of the characteristic equation root (if positive - the system is linearly unstable)(real)
- stabf: The stability label of the system (stable/unstable)

Head of the data set

	tau1	tau2	tau3	tau4	p1	p2	p3
0	2.959060	3.079885	8.381025	9.780754	3.763085	-0.782604	-1.257395
1	9.304097	4.902524	3.047541	1.369357	5.067812	-1.940058	-1.872742
2	8.971707	8.848428	3.046479	1.214518	3.405158	-1.207456	-1.277210
3	0.716415	7.669600	4.486641	2.340563	3.963791	-1.027473	-1.938944
4	3.134112	7.608772	4.943759	9.857573	3.525811	-1.125531	-1.845975
5	6.999209	9.109247	3.784066	4.267788	4.429669	-1.857139	-0.670397
6	6.710166	3.765204	6.929314	8.818562	2.397419	-0.614590	-1.208826

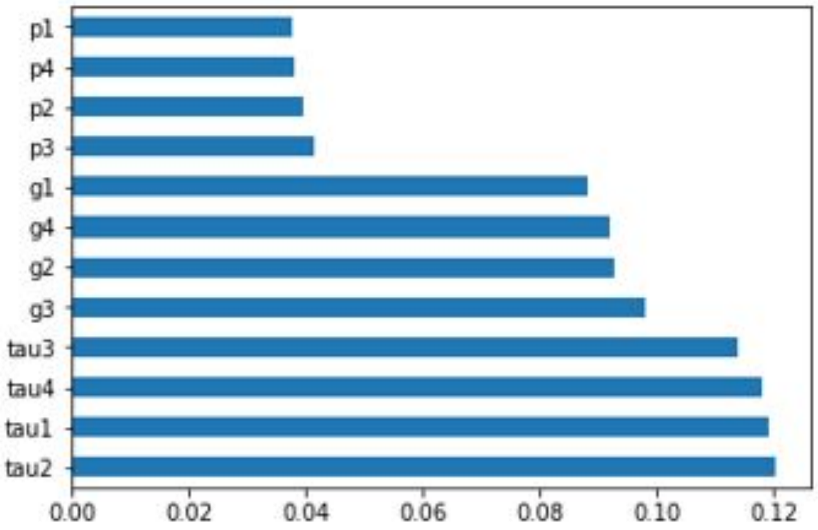
	p4	g1	g2	g3	g4	stab	stabf
0	-1.723086	0.650456	0.859578	0.887445	0.958034	0.055347	unstable
1	-1.255012	0.413441	0.862414	0.562139	0.781760	-0.005957	stable
2	-0.920492	0.163041	0.766689	0.839444	0.109853	0.003471	unstable
3	-0.997374	0.446209	0.976744	0.929381	0.362718	0.028871	unstable
4	-0.554305	0.797110	0.455450	0.656947	0.820923	0.049860	unstable
5	-1.902133	0.261793	0.077930	0.542884	0.469931	-0.017385	stable
6	-0.574004	0.177890	0.397977	0.402046	0.376630	0.005954	unstable

Preprocessing

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(12).plot(kind='barh')
plt.show()
```

```
C:\machinelearning\lib\site-packages\sklearn\ensemble\forest.py:246: Futu
ge from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
[0.11936582 0.12028137 0.11408567 0.11818652 0.03791636 0.03951985
 0.04155688 0.03816714 0.08810775 0.09275132 0.09799064 0.09207067]
```



Head of the data set

	tau1	tau2	tau3	tau4	g1	g2	g3	\
0	2.959060	3.079885	8.381025	9.780754	0.650456	0.859578	0.887445	
1	9.304097	4.902524	3.047541	1.369357	0.413441	0.862414	0.562139	
2	8.971707	8.848428	3.046479	1.214518	0.163041	0.766689	0.839444	
3	0.716415	7.669600	4.486641	2.340563	0.446209	0.976744	0.929381	
4	3.134112	7.608772	4.943759	9.857573	0.797110	0.455450	0.656947	

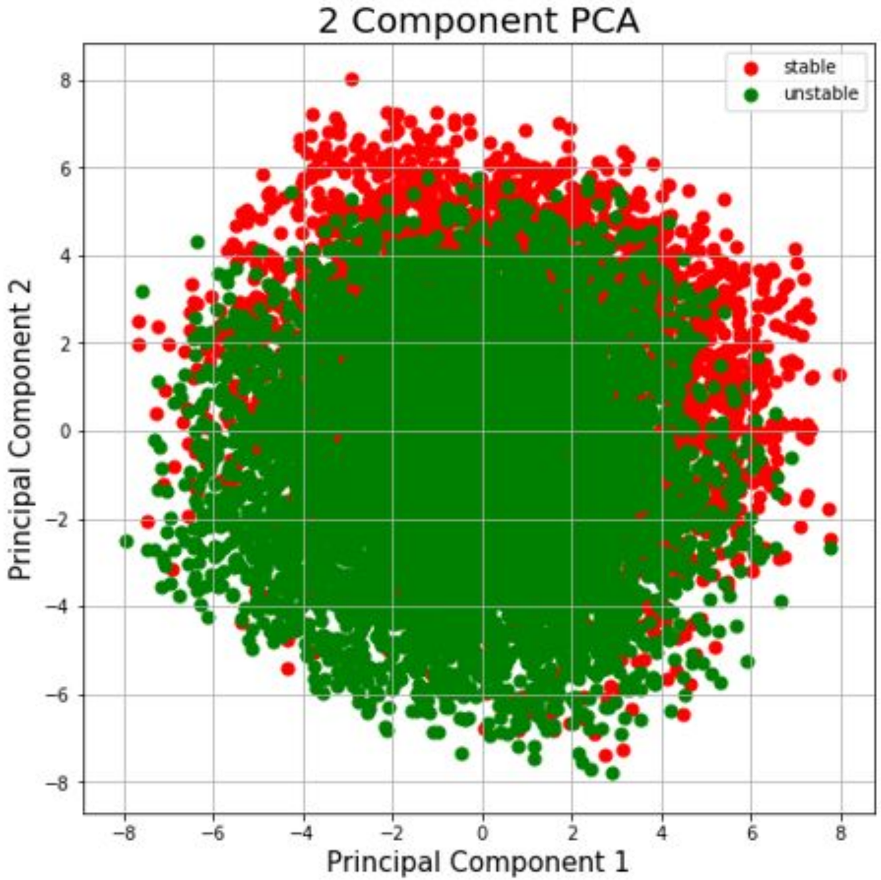
	g4	stabf
0	0.958034	unstable
1	0.781760	stable
2	0.109853	unstable
3	0.362718	unstable
4	0.820923	unstable

PCA transformation

	principal component 1	principal component 2	stabf
0	0.203014	-1.079259	unstable
1	-0.264106	1.845271	stable
2	1.862118	1.181113	unstable
3	1.347015	-0.300821	unstable
4	-0.168866	-0.561726	unstable

```
pca.explained_variance_ratio_
```

```
array([0.25353413, 0.25093078])
```



Classification using PCA

Method	90:10	80:20	75:25	70:30	Confusion Matrix
Naive bayes	**0.706**	0.6985	0.6932	0.6997	confusion matrix: [[151 210] [84 555]]
KNN (N=12)	**0.686**	0.6735	0.666	0.6723	confusion matrix: [[170 191] [123 516]]
SVM	**0.694**	0.689	0.6848	0.6897	confusion matrix: [[142 219] [87 552]]
Decision Tree	0.618	0.622	**0.6352**	0.6216	confusion matrix: [[449 482] [430 1139]]
Random Forest	0.661	0.6655	0.6612	**0.6703**	confusion matrix: [[490 608] [381 1521]]

Classification using without PCA,use standardization

Method	90:10	80:20	75:25	70:30	Confusion matrix
Naive Bayes	0.831	0.832	0.833	**0.83466**	confusion matrix: [[744 354] [142 1760]]
KNN (N=12)	0.906	0.9102	**0.9108**	0.909	confusion matrix: [[783 148] [75 1494]]
SVM	0.81	**0.812**	0.8108	0.8123	confusion matrix: [[507 220] [156 1117]]
Decision Tree	**0.879**	0.8715	0.869	0.85766	confusion matrix: [[294 67] [54 585]]
Random Forest (n_estimators=60)	**0.923**	0.9215	0.916	0.9186	confusion matrix: [[323 38] [39 600]]

Classification by ratio 95:5

Method	Accuracy	Confusion Matrix
Naive Bayes	0.832	confusion matrix: [[122 62] [22 294]]
KNN	0.92	confusion matrix: [[160 24] [16 300]]
SVM	0.92	confusion matrix: [[160 24] [16 300]]
Decision Tree	0.878	confusion matrix: [[153 31] [30 286]]
Random Forest	0.826	confusion matrix: [[129 55] [32 284]]

Feature selection using R project features and results

Test: 0.05

Naive Bayes:
Accuracy - 0.982

```
confusion matrix:
[[180   4]
 [  5 311]]
```

KNN:
Accuracy - 0.964

```
confusion matrix:
[[175   9]
 [  9 307]]
```

Decision Tree
Accuracy - 1.0

```
confusion matrix:
[[184   0]
 [  0 316]]
```

Random Forest:
Accuracy - 1.0

```
confusion matrix:
[[184   0]
 [  0 316]]
```

SVM
Accuracy - 99.2

```
confusion matrix:
[[180   4]
 [  0 316]]
```

```
features = ['tau1','tau2','tau3','p1','p2','p4','g1','g3','g4','stab']
```

Best algorithm fit for electricity simulated data Set for classification

Random Forest(Using n_estimators = 60) and Decision Tree :

- Test dataset: 95:5
- Accuracy = 1.0
- Confusion Matrix:

```
confusion matrix:
[[184   0]
 [  0 316]]
```

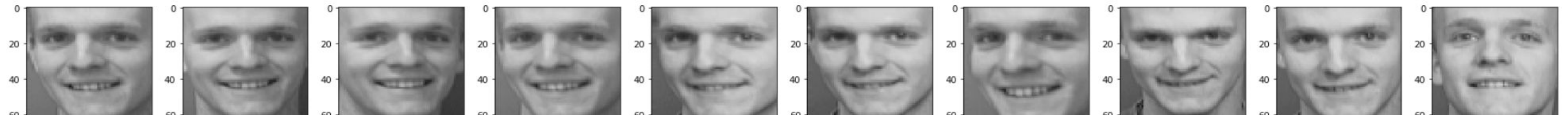
Step 1: Load and explore the data

```
# Input data files consisting of the images
pics = np.load("olivetti_faces.npy")
labels = np.load("olivetti_faces_target.npy")
```

```
print("pics: ", pics.shape)
print("labels: ", labels.shape)
```

```
pics:  (400, 64, 64)
labels:  (400,)
```

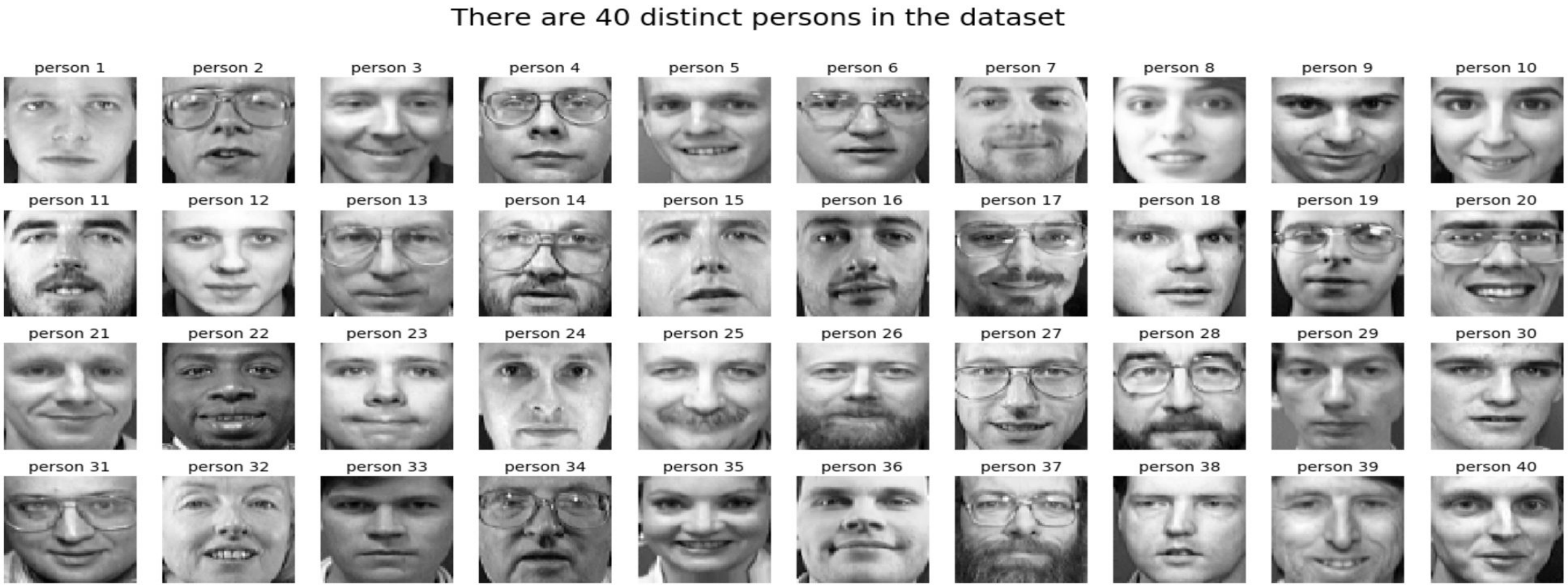
```
# Sample images of a subject
img_cnt = 10
plt.figure(figsize=(24,24))
for i in range(img_cnt):
    plt.subplot(1,10,i+1)
    x=pics[i+40] # 4th subject
    imshow(x)
plt.show()
```



Step 2: Visualize the data Olivetti Data set

```
fig = plt.figure(figsize=(24, 10))
columns = 10
rows = 4
for i in range(1, columns*rows +1):
    img = pics[10*(i-1),:,:]
    fig.add_subplot(rows, columns, i)
    plt.imshow(img, cmap = plt.get_cmap('gray'))
    plt.title("person {}".format(i), fontsize=14)
    plt.axis('off')

plt.suptitle("There are 40 distinct persons in the dataset", fontsize=24)
plt.show()
```



Python Packages Used

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from skimage.io import imshow
import matplotlib.image as mpimg
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
```

Reshape data

In [6]:

```
#Machine Learning models can work on vectors. Since the image data is in the matrix form, it must be converted to a vector.

Y = labels.reshape(-1,1) # store labels in Y
X=pics.reshape(pics.shape[0], pics.shape[1]*pics.shape[2]) # reshape and store images in X

print("X shape:",X.shape)
print("Y shape:",Y.shape)
```

```
X shape: (400, 4096)
Y shape: (400, 1)
```

The data set contains 10 face images for each subject. Of the face images, 80 percent will be used for training, 20 percent for testing. Uses stratify feature to have equal number of training and test images for each subject. Thus, there will be 8 training images and 2 test images for each subject. You can play with training and test rates.

In [7]:

```
#Split data for train and test purposes

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state=46)

print("x_train: ",x_train.shape)
print("x_test: ",x_test.shape)
print("y_train: ",y_train.shape)
print("y_test: ",y_test.shape)
```

```
x_train:  (280, 4096)
x_test:  (120, 4096)
y_train:  (280, 1)
y_test:  (120, 1)
```

Step 4: Use Scikit Learn to execute Naïve Bayes, SVM, KNN, Decision Tree, Random forest

- Use train_test_split from sklearn.cross_validation to shuffle and split the features and prices data into training and testing sets. Split the data into 90%-80%-80% training and 10% -20%-30% testing.
- Assign the train and testing splits to X_train, X_test, y_train, and y_test.
- Run the model Naïve Bayes, SVM, KNN, Decision Tree, Random forest
- Caculate Accurency and confusion matrix

```python


rf = RandomForestClassifier(n_estimators = 400, random_state = 1)
rf.fit(x_train, y_train)
RF_accuracy = round(rf.score(x_test, y_test)*100,2)

y_pred = rf.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
print("confusion matrix:")
print(cm)

print("RF_accuracy is %", RF_accuracy)

list_names.append("Random Forest")
list_accuracy.append(RF_accuracy)
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  after removing the cwd from sys.path.
```

```
confusion matrix:
[[3 0 0 ... 0 0 0]
 [0 6 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 2 0 0]
 [0 0 0 ... 0 3 0]
 [1 0 0 ... 0 0 2]]
RF_accuracy is % 93.33
```

Accuracy Metrics Comparison without PCA

Method	90:10	80:20	70:30
Naive_Bayes	92.5	87.50	73.33
KNN	90.0	91.25	90.00
Random Forest	90.0	93.75	93.33
SVM	95.0	96.25	96.67
Logistic Regression	95.0	**97.50**	**97.50**

Dimensionality Reduction using PCA

```
pca = PCA(.95)
X_train_pca = pca.fit_transform(x_train)
X_test_pca = pca.transform(x_test)


print('Original dataset:',x_train.shape)
print('Dataset after applying PCA:',X_train_pca.shape)
print('No of PCs/Eigen Faces:',len(pca.components_))
print('Eigen Face Dimension:',pca.components_.shape)
print('Variance Captured:',np.sum(pca.explained_variance_ratio_))
```

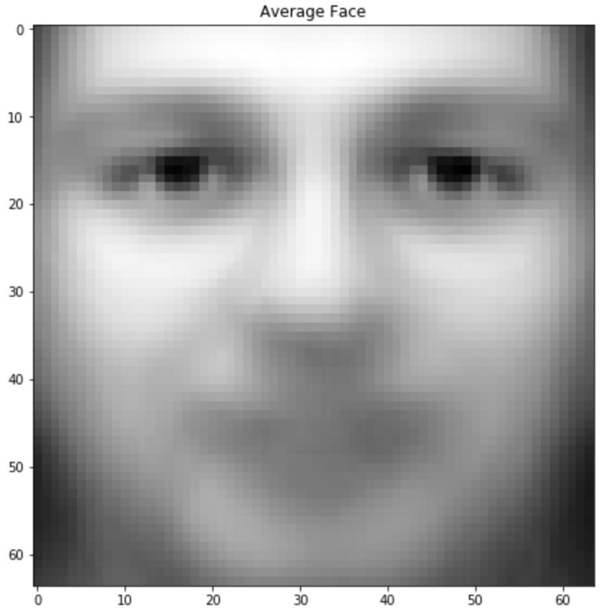
```
Original dataset: (280, 4096)
Dataset after applying PCA: (280, 103)
No of PCs/Eigen Faces: 103
Eigen Face Dimension: (103, 4096)
Variance Captured: 0.95040184
```

Mean Face of the Samples

```
# Average face of the samples

plt.subplots(1,1,figsize=(8,8))
plt.imshow(pca.mean_.reshape((64,64)), cmap="gray")
plt.title('Average Face')
```

```
Text(0.5, 1.0, 'Average Face')
```



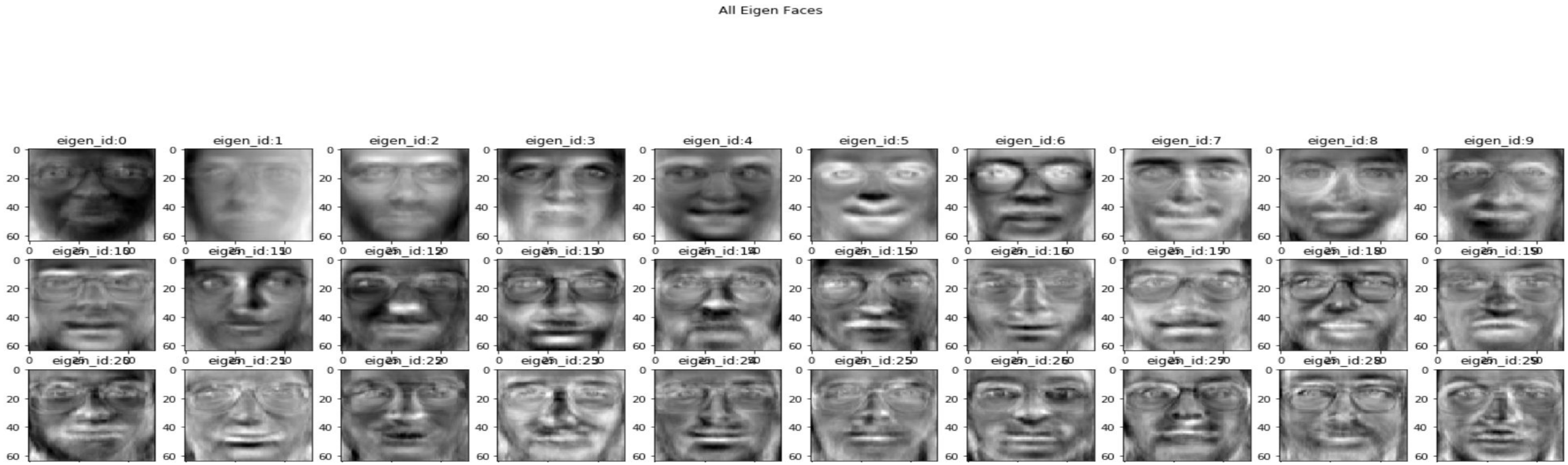
Eigen Faces

```
number_of_eigenfaces=len(pca.components_)
eigen_faces=pca.components_.reshape((number_of_eigenfaces, pics.shape[1], pics.shape[2]))

cols=10
rows=int(number_of_eigenfaces/cols)
fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(24,24))
#axarr=axarr.flatten()
for i in range(number_of_eigenfaces):
    axarr[i].imshow(eigen_faces[i],cmap="gray")

    axarr[i].set_title("eigen_id:{}".format(i))
plt.suptitle("All Eigen Faces".format(10*"=", 10*"="))
```

```
Text(0.5, 0.98, 'All Eigen Faces')
```



Metrics with different Principle Components

Method	**PC = 90**	**PC =103**	**PC = 200**
Naive Bayes	76.67	**77.50**	69.17
KNN	90.83	**90.83**	90.00
Random Forest	93.33	**92.50**	91.67
Logistic Regression	96.67	**98.33**	98.33
SVM	96.67	**96.67**	96.67

Accuracy Metrics Comparison with PCA

METHOD	90:10		80:20		70:30	
	Without PCA	With PCA	Without PCA	With PCA	Without PCA	With PCA
Naive Bayes	92.5	92.5	87.50	90.0	73.33	77.50
KNN	90.0	90.0	91.25	92.5	90.00	90.83
Random Forest	90.0	100.0	93.75	95.0	93.33	92.50
Logistic Regression	95.0	97.5	96.25	97.5	97.50	98.33
SVM	95.0	95.0	97.5	97.5	96.67	96.67

Best fit for Olivetti data set

Random Forest : PCA = 103

Test dataset = 90:10

Accuracy = 100

Conclusion

Use of PCA improved the accuracy metrics.

Project Enhancement Idea : Identify smile and no Smile face/

Identify male and female

Reference

- https://www.kaggle.com/serkanpeldek/face-recognition-on-olivetti-dataset
- https://colab.research.google.com/notebooks/welcome.ipynb
- https://www.youtube.com/watch?v=PitcORQSjNM
- https://www.youtube.com/watch?v=_VTtrSDHPwU&t=86s
- https://www.kaggle.com/elikplim/eergy-efficiency-dataset/kernels
- Class notes: Introduction to data mining and machine learning

Thank you for your attention