Final Project: Machine Learning **UCSC-Spring 2019**
Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

Machine learning implementations on different datasets using Python

Introduction:

In this paper, two different datasets are being collected to implement the machine learning classification techniques introduced from introduction to data mining and machine learning coursework. Both datasets are collected by analyzing their output and according to the author's interest. Following are the datasets are named as,
Electricity grid stability simulated dataset and Face Recognition on Olivetti Dataset

Electricity grid stability simulated dataset: This dataset is available in UCI dataset repository named as "Electricity grid stability simulated dataset". Click here to get the dataset. This dataset has different 14 attributes and 10,000 instances. This dataset has float values. Selection of the dataset is done by personal background interest of electrical engineering and power system to explore the energy electricity dataset to extract more information from them.

This contains 11 predictive attributes, 1 non – predictive (p1) , 2 goal fields and other attributes are tau(x) which indicates the reaction time of participant (real from the range [0.5,10]s). Tau1 - the value for electricity producer, p[x]: nominal power consumed (negative) /produced (positive) (real) in which consumers are from the range [-0.5,-2]s^-2; p1 = abs(p2 + p3 + p4), another is g[x]: coefficient (gamma) proportional to price elasticity (real from the range [0.05,1]s^-1) where g1 - the value for electricity producer, another named as stab which is the maximal real part of the characteristic equation root (if positive - the system is linearly unstable)(real), last one is stabf: the stability label of the system (categorical: stable/unstable).

In this dataset to classify which are observations are stable and which among them are unstable all different type of classification techniques is being used. The techniques used are named as Naïve Bayes, SVM, KNN, Decision Tree, Random forest implemented using python. Preprocessing of the dataset is done by using top 10 feature selection inbuilt class that gives tree-based classifiers, and extra tree classifier is being to extract the top 10 features for the dataset which are shown below;

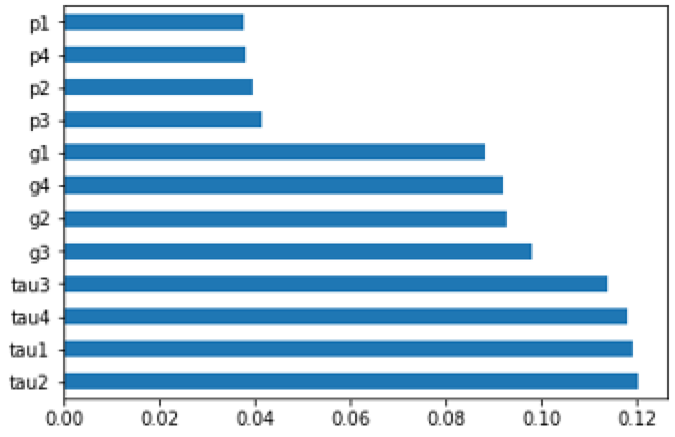
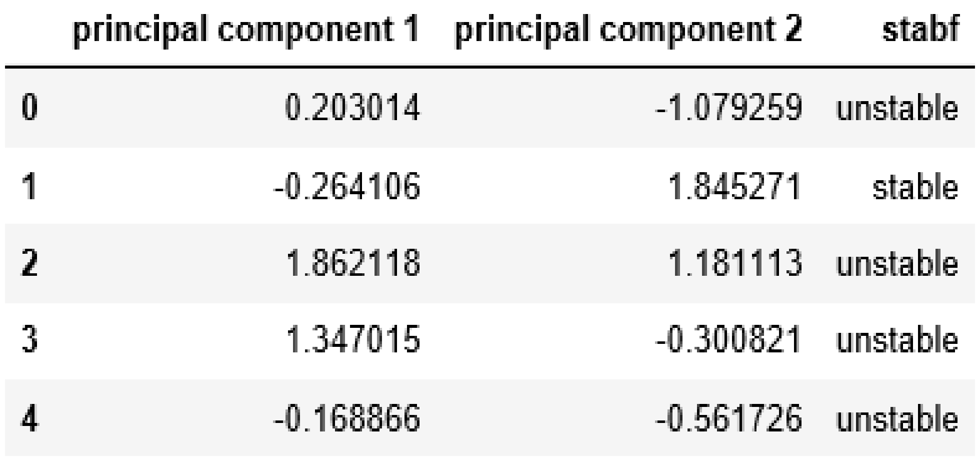


Figure 1: Feature selection using top ten feature technique

Further PCA is done to add variance to observations and it adds approx. 25% variance to principal component 1 and principal component 2. Following are the changes shown in the dataset by the principal component 1 and principal component 2;



	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

Table 1: It shows the changes in dataset by principal component 1 and 2

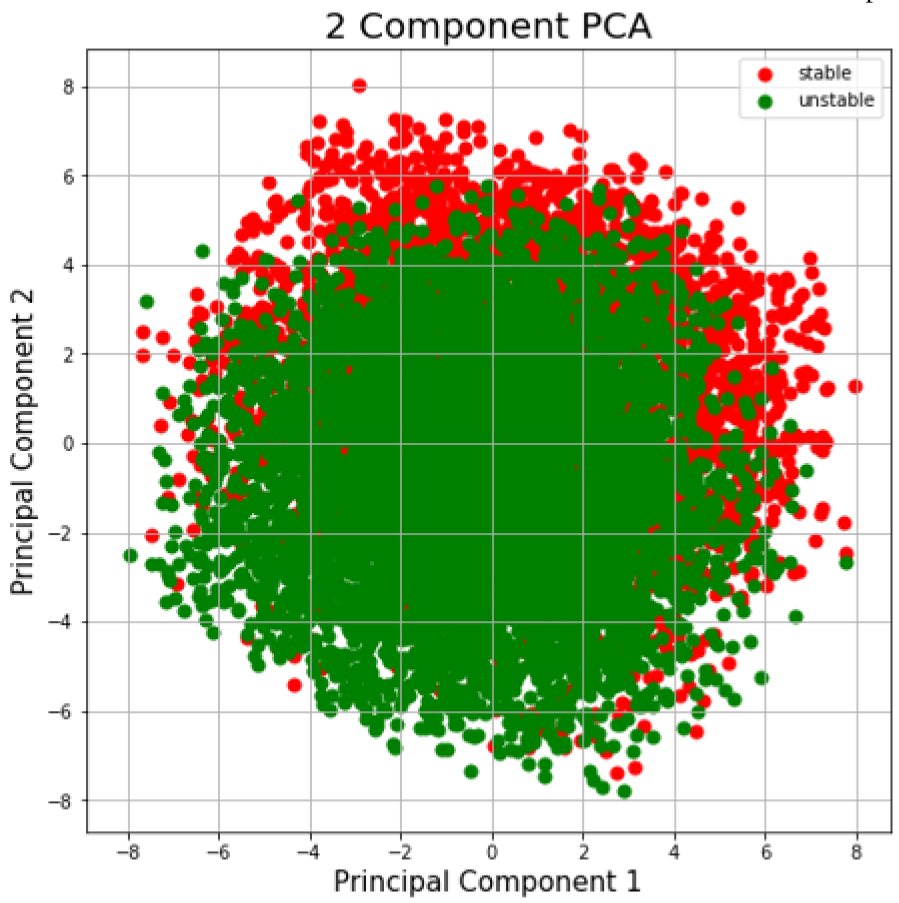


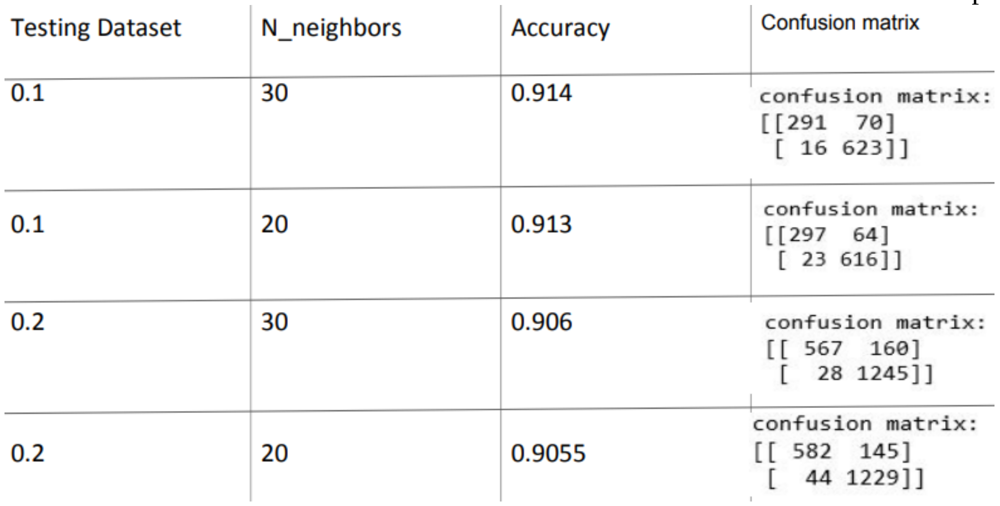
Figure 2: Representation of Principal component 1 and principal component 2

The results of without PCA using Naïve Bayes are given below in table;

Testing dataset	Accuracy	Confusion matrix
0.3	0.83466	confusion matrix: [[744 354] [142 1760]]
0.25	0.83	confusion matrix: [[619 312] [111 1458]]
0.2	0.832	confusion matrix: [[481 246] [90 1183]]

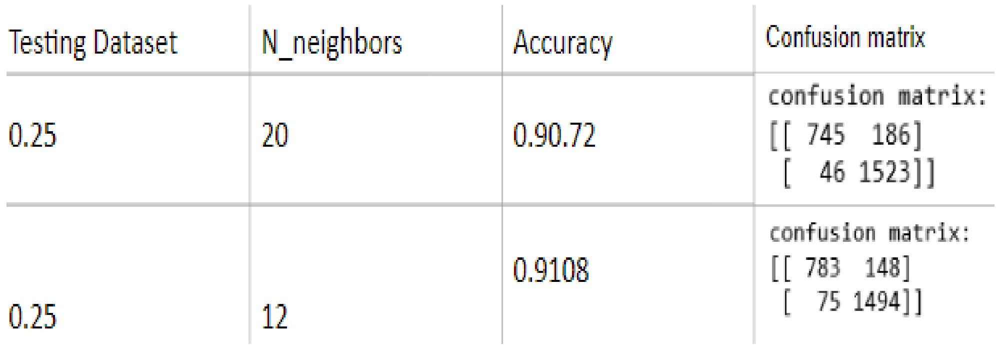
Table 2: Naïve Bayes using testing dataset 0.3, 0.25 and 0.2

Further, the results are shown without PCA using KNN using different n neighbors;



Testing Dataset	N_neighbors	Accuracy	Confusion matrix
0.1	30	0.914	confusion matrix: [[291 70] [16 623]]
0.1	20	0.913	confusion matrix: [[297 64] [23 616]]
0.2	30	0.906	confusion matrix: [[567 160] [28 1245]]
0.2	20	0.9055	confusion matrix: [[582 145] [44 1229]]

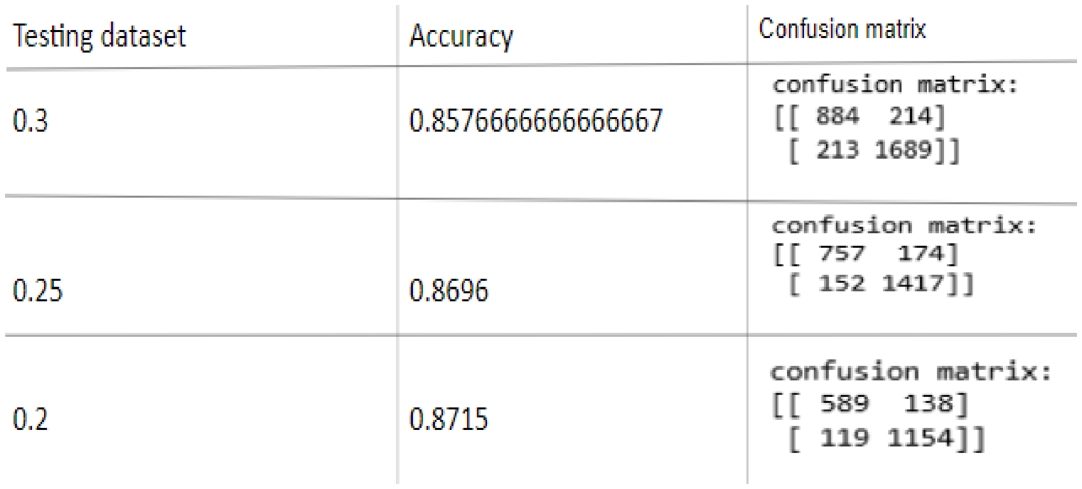
Table 3: KNN using different test dataset and different n neighbors



Testing Dataset	N_neighbors	Accuracy	Confusion matrix
0.25	20	0.90.72	confusion matrix: [[745 186] [46 1523]]
0.25	12	0.9108	confusion matrix: [[783 148] [75 1494]]

Table 4: Using different training and testing dataset by varying n Neighbors

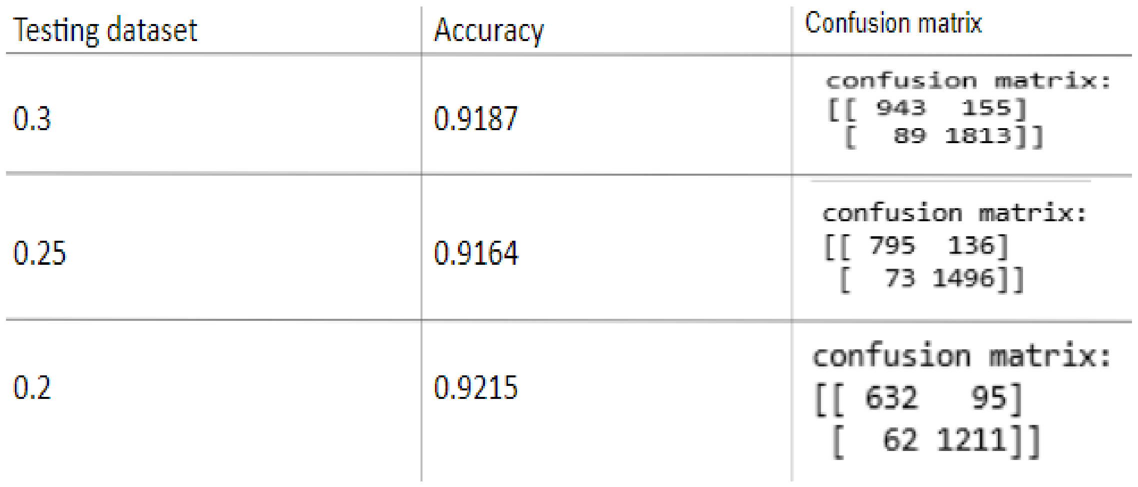
Below are the results of using different split for train and test using decision tree algorithm;



Testing dataset	Accuracy	Confusion matrix
0.3	0.8576666666666667	confusion matrix: [[884 214] [213 1689]]
0.25	0.8696	confusion matrix: [[757 174] [152 1417]]
0.2	0.8715	confusion matrix: [[589 138] [119 1154]]

Table 5: Results using Decision Tree algorithm

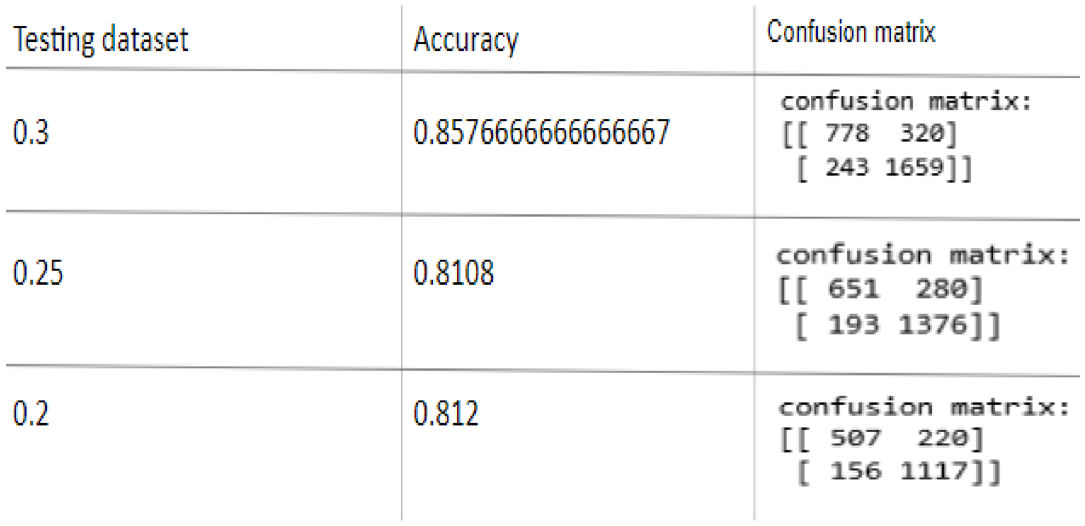
Below are the results of using different split for train and test using Random Forest algorithm;



Testing dataset	Accuracy	Confusion matrix
0.3	0.9187	confusion matrix: [[943 155] [89 1813]]
0.25	0.9164	confusion matrix: [[795 136] [73 1496]]
0.2	0.9215	confusion matrix: [[632 95] [62 1211]]

Table 6: Results using Random Forest algorithm

Below are the results of using different split for train and test using SVM algorithm;



Testing dataset	Accuracy	Confusion matrix
0.3	0.8576666666666667	confusion matrix: [[778 320] [243 1659]]
0.25	0.8108	confusion matrix: [[651 280] [193 1376]]
0.2	0.812	confusion matrix: [[507 220] [156 1117]]

Table 4: Results using SVM algorithm

Below is the Comparison of 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]]

Table 5: Comparison of all algorithm according to test data and confusion matrix of test dataset is given for highlighted numbers.

Below is the Comparison of Classification without PCA:

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]]

Table 6: Comparison of all algorithm according to test data and confusion matrix of test dataset is given for highlighted numbers.

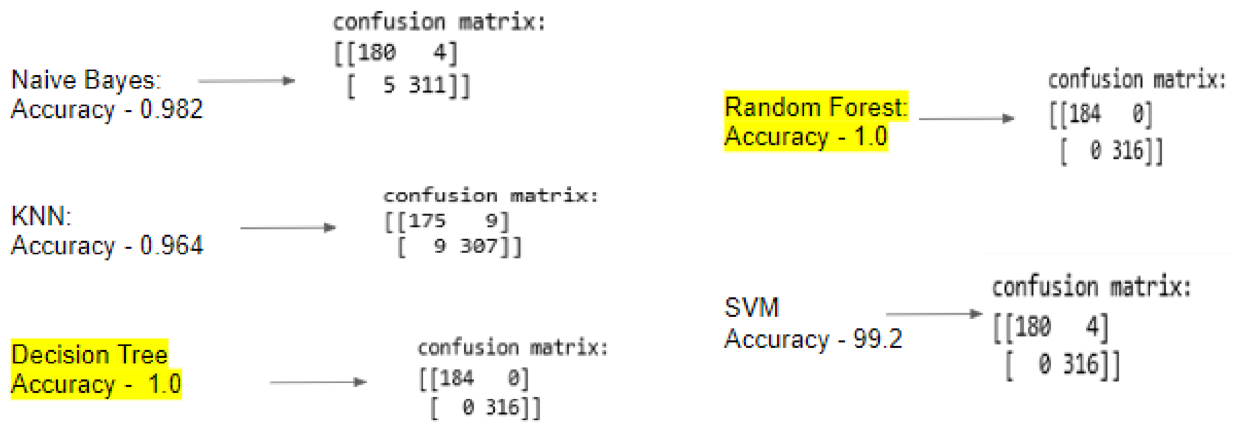
Below is the comparisons of all algorithms for testing dataset 0.05;

Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

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]]

Table 7: Comparison of all algorithm according to test data and confusion matrix of test dataset is given for highlighted numbers.

Now, to improve the accuracy, I have changed the features included tau1, tau2, tau3, p1, p2, p4, g1, g3, g4, stab and results are displayed below;



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]]
```

Figure 3: Comparison of different algorithms using test split 0.05 and different features

Conclusion: The feature selection by using top ten feature selection gives more accurate results but after changing the features the results having accuracy 1 are overfitted. When PCA is performed initially it gave very less accurate results and performance of algorithms improved by standardization of x and gave good results highest by random forest which is 0.92. Project scope is to do evaluation of training dataset performance on the observations. Second, preprocessing again and again try to fit the best algorithm to get more accurate results.
Note: Code of the paper is attached in zip file.

Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

Followed by the explanation of another data set names as Face Recognition on Olivetti Dataset. The data set contains 10 face images for each subject. The data set has 40 different person-owned, facial images. Size of each image is 64x64. There are 40 distinct subjects, and each has 10 images taken differently. (Hence the data set has 400 rows and pixel size of each image is 64 X 64). Names of 40 people were encoded to an integer from 0 to 39, the images were taken at different times, by varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses).

Project Idea:

Face recognition is a hot topic in industry.

This article is about performing multioutput regression on the Olivetti images and predicting the faces. Below are the algorithums to be used.

1. Logistics Regression
2. RandomForestRegressor
3. KNN
4. SVM
5. Naive Bayes

3. Requirements

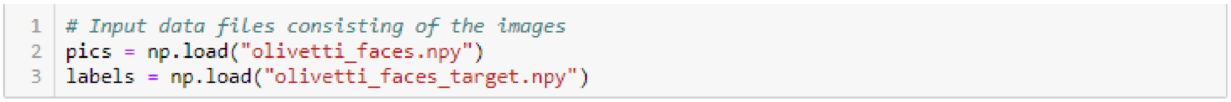
The python libraries are necessary for the rest of this mini project

```
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
```

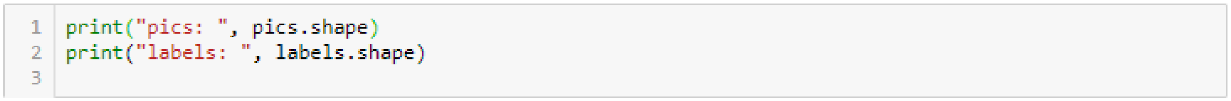
Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

4. Description of the Python program

Step 1: Load and explore the data

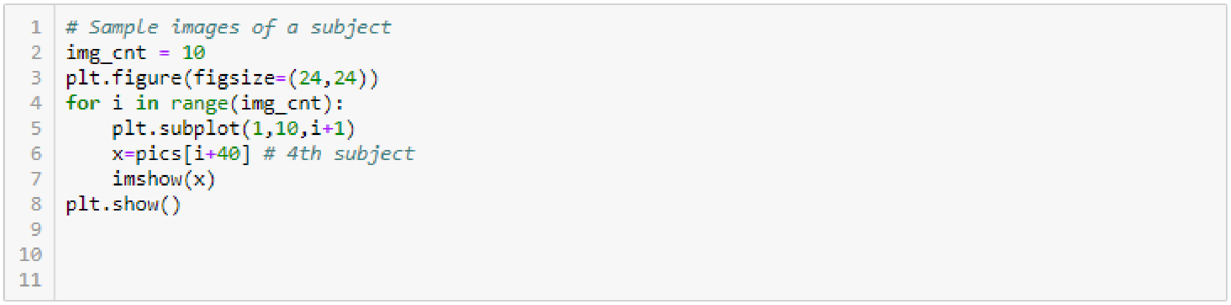


```python
# Input data files consisting of the images
pics = np.load("olivetti_faces.npy")
labels = np.load("olivetti_faces_target.npy")
```

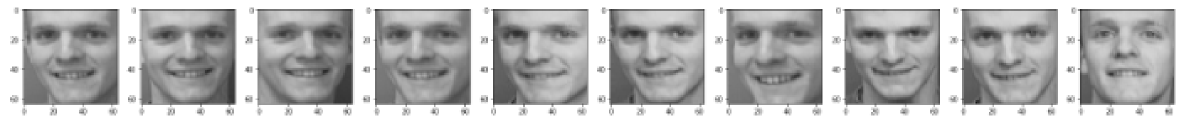


```python
print("pics: ", pics.shape)
print("labels: ", labels.shape)

```

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



```python
# 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()



```

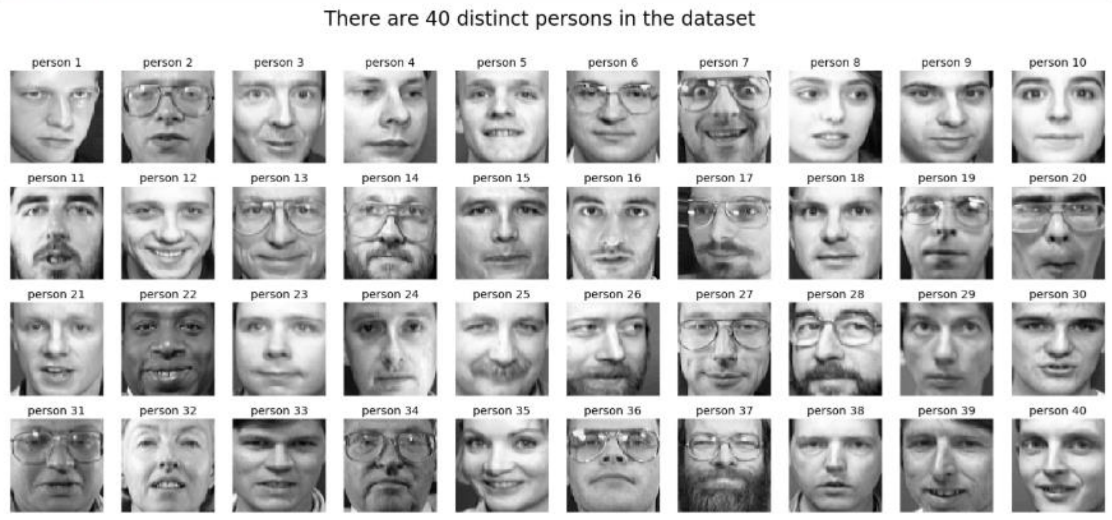


Step 2: Visualize the data

To see the dataset and model idea for the dataset.

Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

```python
# All unique faces in the sample
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()
```



Step 3: 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

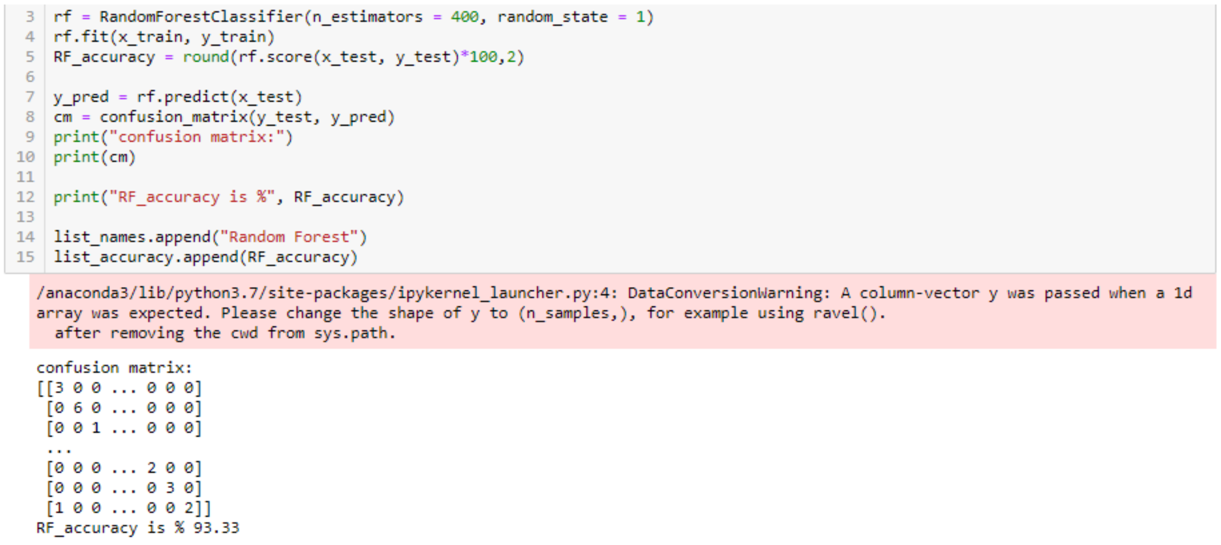
Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

1. 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.

2. Assign the train and testing splits to X_train, X_test, y_train, and y_test.

3. Run the model Naïve Bayes, SVM, KNN, Decision Tree, Random forest

4. Calculate Accuracy and confusion matrix



```
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)
```

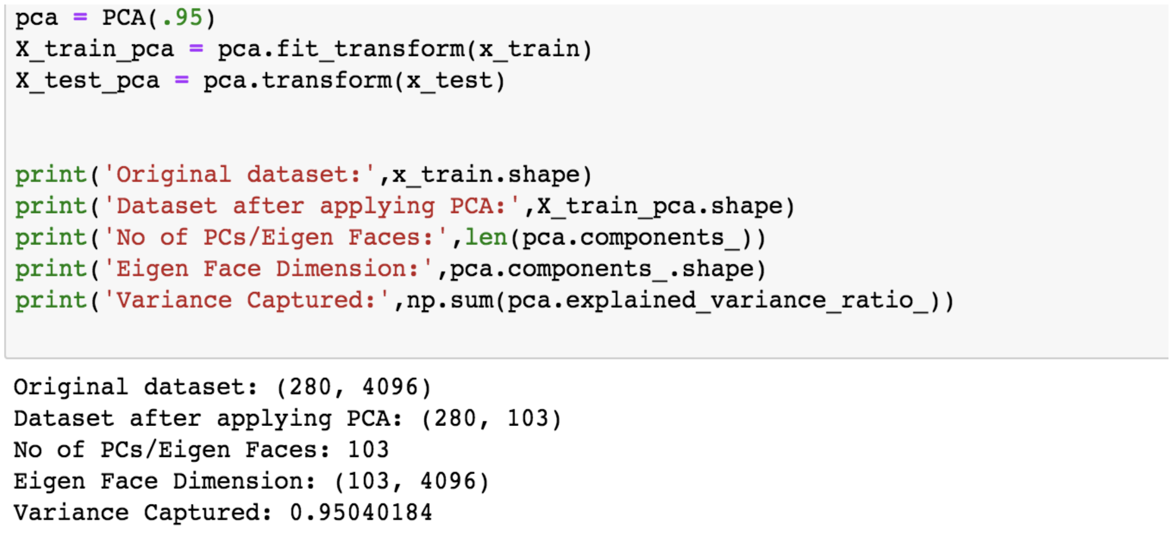
```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  after removing the cwd from sys.path.
```

```
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
```

Step 5: PCA

Introduce dimensionality reduction using PCA.

Our dataset is 64x64 image.Each pixel is considered a feature/dimension which ends up with 496



```
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
```

dimensions. With variance of 95% we could reduce to 103 principle components or dimensions.

Step 6:

Calculate Eigen Faces and Mean Face

When PCA is called a mean of the dataset is created. The mean is subtracted from the original dataset to create a covariance matrix. From the covariance matrix there is decomposition of Eigen face and Eigen values. This is an interesting dataset which can showcase the mean and Eigen face/Eigen Vector.

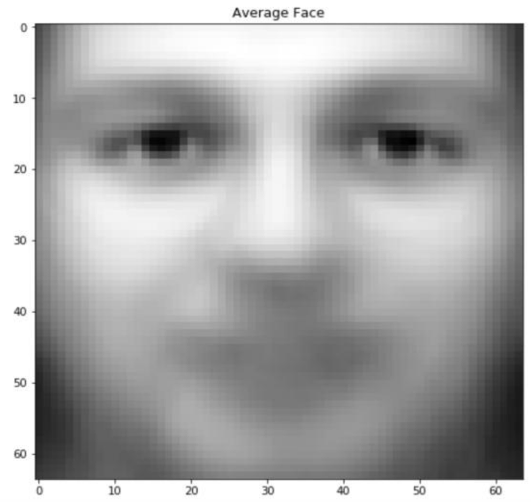
Mean Face is the average of all images in the dataset.

Eigen faces / principle components captures variations in the images. Variations can be different lighting, facial angle, hair length etc. There is one Eigen face for each principle component.

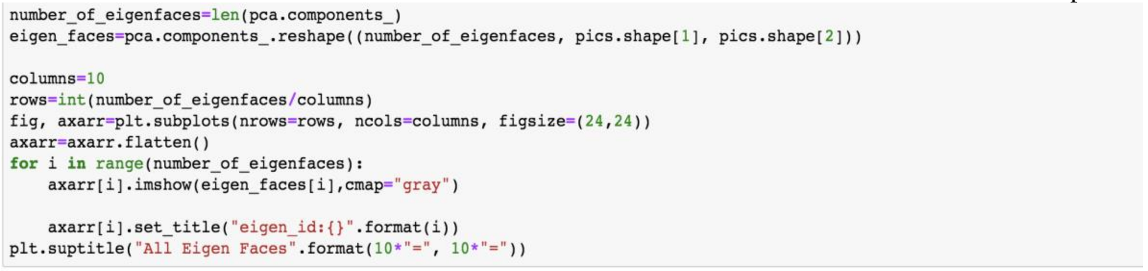
Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

```
# 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')
```

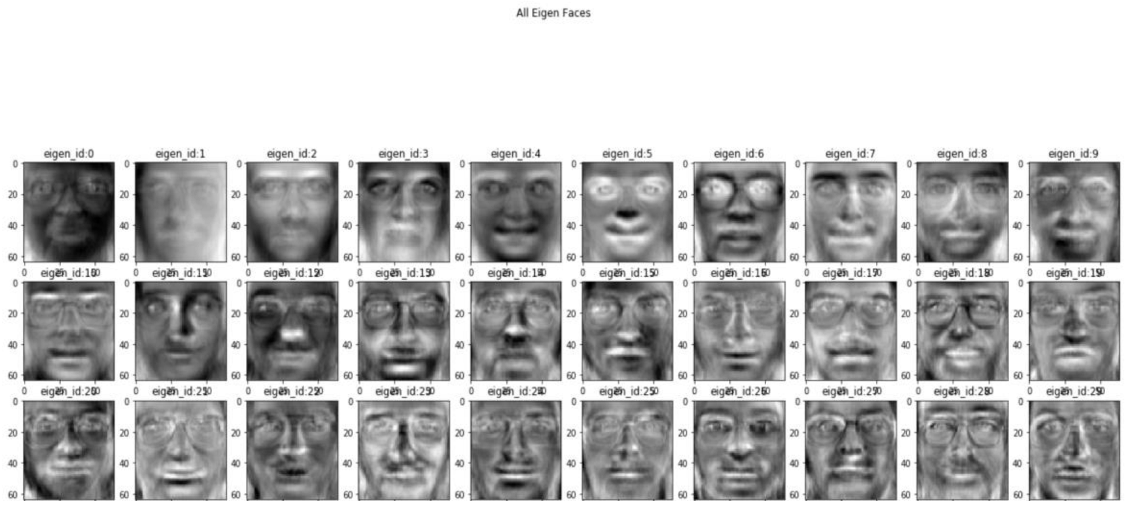


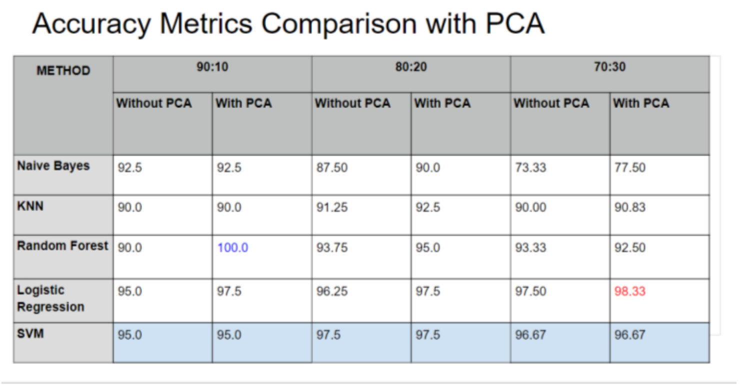
Step 7: Program output



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

columns=10
rows=int(number_of_eigenfaces/columns)
fig, axarr=plt.subplots(nrows=rows, ncols=columns, 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')
```





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

With PCA the SVM model have the same results, but other model's accuracy increases with use of PCA. There is accuracy of 100 for random forest. This could be because our data is small, and data is clean. With 90% data in the training it could have caused the algorithm to overfit.

Oanh Doan
Tess Mundadan
Ramandeep Kaur Bagri

Conclusion: PCA worked well with image dataset. We got a accuracy of 100 for Random forest with data split of 70:30 and PCA =103.This could be a result of overfitting with 90% data in training. Enhancement for this project would be to identify female - male learning/ Smile or not smile face.

References:

https://www.kaggle.com/serkanpeldek/face-recognition-on-olivetti-dataset
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