khalid-dies-21320308

December 19, 2020

1 Face Recognition

in this coureswork a Face Recognition system will be created and tested and modyfied ....

1.1 Reasarch Paper

1.1.1 About the pape in gernal :

In this paper, they have proposed a new loss function, called center loss. and they combining the center loss with the softmax loss to jointly supervise the learning of CNNs, and they using different dataset from the comparsing and they are using different models to comber with loss function models and the new proposed model and they using Convolutional neural networks (CNNs) to shape it

1.1.2 Proplems :

- For face recognition task, the deeply learned features need to be not only separable but also discriminative
- The necessity of joint supervision. If they only use the softmax loss as supervision signal, the resulting deeply learned features would contain large intra-class variations. On the other hand, if they only supervise CNNs by the center loss, the deeply learned features and centers will degraded to zeros (At this point, the center loss is very small). (In another language using either of them could not achieve discriminative feature learning.)

1.1.3 Methods that they used :

Convolutional neural networks (CNNs) softmax loss function center loss function

specifically * the supervision of softmax loss only (model A) * softmax loss and contrastive loss (model B) * softmax loss and center loss (model C) * combine them to jointly supervise the CNNs, as confirmed by there experiments.

1.1.4 Dataset that have been used :

- Labeled Faces in the Wild (LFW)
- YouTube Faces (YTF)
- MegaFace Challenge

1.1.5 The result and the acrusy :

The result and the acrusy from the LFW and YTF Datasets

table 1

Method	Images	Networks	Acc. on LFW	Acc. on YTF
DeepFace	4M	3	97.35 %	91.4 %
DeepID-2+	-	1	98.70 %	-
DeepID-2+	-	25	99.47 %	93.2 %
FaceNet	200M	1	99.63 %	95.1 %
Deep FR	2.6M	1	98.95 %	97.3 %
Baidu	1.3M	1	99.13 %	-
model A	0.7M	1	97.37 %	91.1 %
model B	0.7M	1	99.10 %	93.8 %
model C(Proposed)	0.7M	1	99.28 %	94.9 %

model C beats the baseline one model A (97.37 % on LFW and 91.1 % on YTF) to (99.28 % on LFW and 94.9 % on YTF) model C beats the baseline one model B (99.10 % on LFW and 93.8 % on YTF) to (99.28 % on LFW and 94.9 % on YTF)

model C (much less training data and simpler network architecture) are consistently among the top-ranked sets of approaches based on the two databases, outperforming most of the existing results in Table

The result and the acrusy from the Dataset of MegaFace Challenge

table 2
Identification rates of different methods on MegaFace with 1M distractors.

Method	Protocol	Identification Acc. (Set 1)
NTechLAB - facenx_large	Large	73.300 %
Google - FaceNet v8	Large	70.496 %
Beijing Faceall Co. - FaceAll_Norm_1600	Large	64.803 %
Beijing Faceall Co. - FaceAll_1600	Large	63.977 %
Barebones_FR - cnn	Small	59.363 %
NTechLAB - facenx_small	Small	58.218 %
3DiVi Company - tdvm6	Small	33.705 %
Model A-	Small	41.863 %
Model B-	Small	57.175 %
Model C- (Proposed)	Small	65.234 %

table 3
Verification TAR of different methods at 106 FAR on MegaFace with 1M distractors.

Method	Protocol	Verification Acc. (Set 1)
Google - FaceNet v8	Large	86.473 %
NTechLAB - facenx_large	Large	85.081 %
Beijing Faceall Co. - FaceAll_Norm_1600	Large	67.118 %
Beijing Faceall Co. - FaceAll_1600	Large	63.960 %
Barebones_FR - cnn	Small	59.036 %
NTechLAB - facenx_small	Small	66.366 %
3DiVi Company - tdvm6	Small	36.927 %
Model A-	Small	41.297 %
Model B-	Small	69.987 %
Model C- (Proposed)	Small	76.516 %

from table 2 and 3 its not surprisingly, model C- consistently outperforms model A- and model B- by a significant margin in both face identification and verification tasks

under the evaluation protocol of small training set, the proposed model C- achieves the best results in both face identification and verification tasks

it is worth to note that (model C) even surpasses some models trained with large training set

the models from Google and NTechLAB achieve the best performance under the protocol of large training set. Note that, their private training set (500M for Google and 18M for NTechLAB) are much larger than ours (0.49M).

1.1.6 My own opinion about the paper :

the paper proposed a new loss function wich is very good called center loss not just that the paper combin it with an or rady exiest loss function called softmax loss and the results from the combining it sprised me that

- it almots out prformed other models with more dataset (you can see from table 1)
- you can see from table 1 the DeepID-2+ has 25 network and 93.2 % on Acc. on YTF comer to model C (Proposed) has 1 network and Acc. on YTF 94.9 % with out prefomed it also us can see from the same table that FaceNet 200M image and results on Acc. on LFW 99.63 % and 95.1 % on Acc. on YTF comer to model C (Proposed) has 0.7M image and results on Acc. on LFW 99.28 % and 94.9 % on Acc. on YTF with it is not far byand
- it surpasses some models trained with large training set (from table 2,3)
- fome table 2 you can see that it out preformed Beijing Faceall Co. - FaceAll_Norm_1600 and Beijing Faceall Co. - FaceAll_1600 with Large Protocol
- from table 3 you can see that it out preformed Beijing Faceall Co. - FaceAll_Norm_1600 and Beijing Faceall Co. - FaceAll_1600 with Large Protocol

1.1.7 this paper has be obtined from this link

https://link.springer.com/chapter/10.1007/978-3-319-46478-7_31

1.2 Dataset :

this Dateset is obtained from sklaren

[1]:
```
import warnings
#Here i ignore unncssery warnings
warnings.filterwarnings('ignore')
#here import the libre
from sklearn.datasets import fetch_lfw_people
# get the data, with 1 images per person
faces = fetch_lfw_people(min_faces_per_person=60,download_if_missing=True)
```

[2]:
```
print ("here is some infromation about the dateset and the images ")
print(f'The number of target faces : \n {faces.target_names} \n')
print(f'The number of image is : {faces.images.shape[0]} \n')
print(f'The original size of the image is : {faces.images.shape[1]} x {faces.
 ↪images.shape[2]} in pixels')
```

```
here is some infromation about the dateset and the images
The number of target faces :
 ['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']

The number of image is : 1348

The original size of the image is : 62 x 47 in pixels
```

1.2.1 Showing example of the dataset in grid view

In the grid view i used 5 row and 3 couloms to expris the faces and i show the name under the image of the face and i show them is size 10 x 10

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
fig, ax = plt.subplots(3, 5,figsize=(10,10))
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],xlabel=faces.target_names[faces.target[i]])
```

1.2.2 Dataset Prepration

in this part i will show you what i did to the dateset and explan it

1.2.3 Principal Component Analysis (PCA)

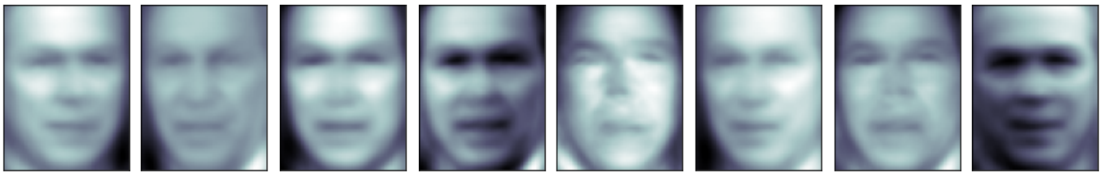
what it help with: 1) speed up the fitting of a machine learning algorithm 2) it sacle youer dataset

using PCA

[4]:
```
#we import PCA Lipry as RandomizedPCA
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.decomposition import PCA
# create dimenssion reduction
pcat = PCA(whiten=True, random_state=42)
```

[5]:
```
##### use PCA in (small terms of variance)
pcat = RandomizedPCA(5)#1
pcat.fit(faces.data)
components1 = pcat.transform(faces.data)
projected1 = pcat.inverse_transform(components1)
```

[6]:
```
# lets visualise these com
fig, axes = plt.subplots(1, 8, figsize=(12, 6),
    subplot_kw={'xticks':[], 'yticks':[]},gridspec_kw=dict(hspace=0.1, wspace=0.
 ↪1))

# notice pca.components is a numpy array
for i, ax in enumerate(axes.flat):
    ax.imshow(projected1[i].reshape(62, 47), cmap='bone')
```

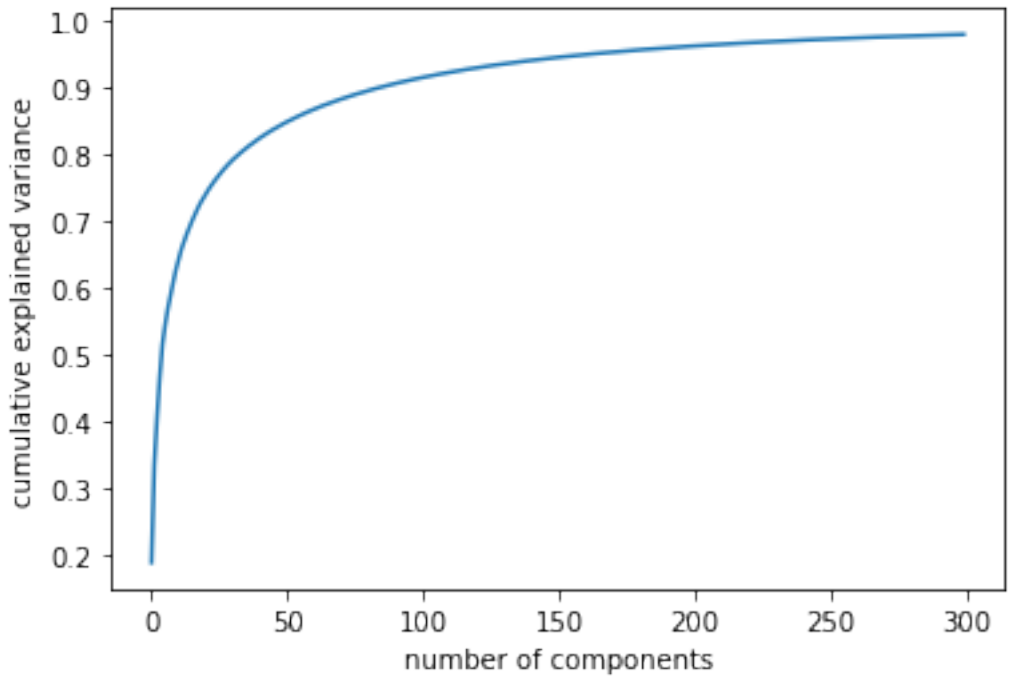


not using PCA

[7]:
```
#### use PCA in (highest terms of variance)
pcat = RandomizedPCA(300)#300
pcat.fit(faces.data)
components = pcat.transform(faces.data)
projected = pcat.inverse_transform(components)
```

[8]:
```
# lets visualise these com
fig, axes = plt.subplots(1, 8, figsize=(12, 6),
    subplot_kw={'xticks':[], 'yticks':[]},gridspec_kw=dict(hspace=0.1, wspace=0.
 ↪1))

# notice pca.components is a numpy array
for i, ax in enumerate(axes.flat):
    ax.imshow(projected[i].reshape(62, 47), cmap='bone')
```

[9]:
```
# plot cumulative variance captured by pca
plt.plot(np.cumsum(pcat.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



You can see from the above figure,that

the 300 components captures more than almost 98% of the original data

the 5 components captures more than almost 22% of the original data

[10]:
```
from sklearn.decomposition import PCA as RandomizedPCA
# Fit PCA and keep the top 300 eigenvectors
# Compute the components and projected faces
pca = RandomizedPCA(300)
pca.fit(faces.data)
components = pca.transform(faces.data)
projected = pca.inverse_transform(components)

# only keep top 150 eigenvectors
pca0 = RandomizedPCA(150)
pca0.fit(faces.data)
components0 = pca0.transform(faces.data)
projected0 = pca0.inverse_transform(components0)
```

```
# only keep only 5 eigenvectors
pca1 = RandomizedPCA(5)
pca1.fit(faces.data)
components1 = pca1.transform(faces.data)
projected1 = pca1.inverse_transform(components1)
```

[11]:
```
# Plot the results
fig, ax = plt.subplots(4, 10, figsize=(10, 5.5),
                       subplot_kw={'xticks':[], 'yticks':[]},
                       gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    ax[0, i].imshow(faces.data[i].reshape(62, 47), cmap='binary_r')
    ax[1, i].imshow(projected[i].reshape(62, 47), cmap='binary_r')
    ax[2, i].imshow(projected0[i].reshape(62, 47), cmap='binary_r')
    ax[3, i].imshow(projected1[i].reshape(62, 47), cmap='binary_r')

ax[0, 0].set_ylabel('full-dim\ninput')
ax[1, 0].set_ylabel('300-dim')
ax[2, 0].set_ylabel('150-dim')
ax[3, 0].set_ylabel('5-dim')
```

[11]:
```
Text(0, 0.5, '5-dim')
```

in here i split the data to 1)train set 2)test set

[12]:
```
from sklearn.model_selection import train_test_split
# random_state is for results reproduction
```

```
X_train, X_test, y_train, y_test = train_test_split(faces.data, faces.
 ↪target,random_state=42)
```

[13]:
```
# check the shape of the data
print(f'The size of the Training set is {X_train.shape[0]}')
print(f'The size of the Testing set is {X_test.shape[0]}')
# notice the number of column in the dataset
print(f'The number of columns in the training and testing set is {X_train.
 ↪shape[1]}')
```

```
The size of the Training set is 1011
The size of the Testing set is 337
The number of columns in the training and testing set is 2914
```

[14]:
```
# the data types of X_train / y_train (numpy array)
print(f'Type of X_train {type(X_train)}')
print(f'Type of y_train {type(y_train)}')
```

```
Type of X_train <class 'numpy.ndarray'>
Type of y_train <class 'numpy.ndarray'>
```

1.3 Moduling

In here we import sklearn and we used 1)Sport Vector Musician (SVM) 2)Random Forest Classifier (RFM) to train our model we will show you frist (svm) then Random Forest

1.3.1 SMV

[15]:
```
#here i import Sport Vector Musician
from sklearn.svm import SVC
# create your SVM model with RBF kernel
svc = SVC(kernel='rbf', class_weight='balanced')
```

1.3.2 RFC

[16]:
```
#here i import Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
# create your RFC model with n_estimators 100
rfc = RandomForestClassifier(n_estimators=100)
```

In here i import make pipeline to make the model

[17]:
```
from sklearn.pipeline import make_pipeline
# pipline FOR SVM WITH NO PCA
modelSVMNP = make_pipeline(pca, svc)
# pipline FOR SVM WITH 5 components captures PCA
modelSVM = make_pipeline(pca1, svc)
```

```
# pipline FOR RFC WITH NO PCA
modelRFCNP = make_pipeline(pca, rfc)
# pipline FOR RFC WITH 5 components captures PCA
modelRFC = make_pipeline(pca1, rfc)
```

C and gamma these two parameters are important for the performance of the module. You can try to set these manually, but this can be time consuming. However, here, we will use grid search to get the best set of parameters.

[18]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'svc__C': [1, 5, 10, 50],
    'svc__gamma': [0.0001, 0.0005, 0.001, 0.005]
}
gridSVMNP = GridSearchCV(modelSVMNP, param_grid)
gridSVM = GridSearchCV(modelSVM, param_grid)
gridRFCNP = modelRFCNP
gridRFC = modelRFC
```

fit the data and search for the best parameters

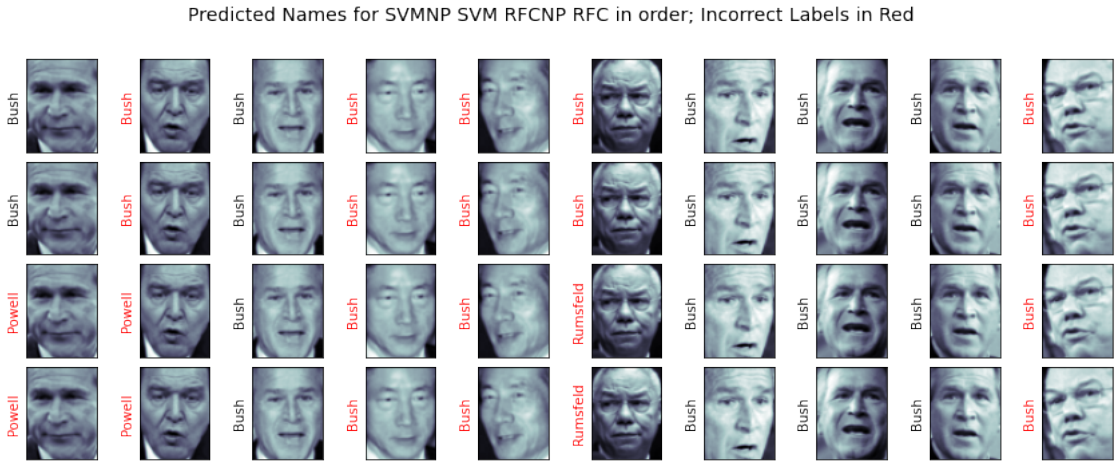
[19]:
```
warnings.filterwarnings('ignore')# ignore warnings

gridSVMNP.fit(X_train, y_train)
gridSVM.fit(X_train, y_train)
gridRFCNP.fit(X_train, y_train)
gridRFC.fit(X_train, y_train)

best_params_SVMNP = gridSVMNP.best_params_
best_params_SVM = gridSVM.best_params_
```

[20]:
```
# check best parameters
print(best_params_SVMNP)
print(best_params_SVM)
```

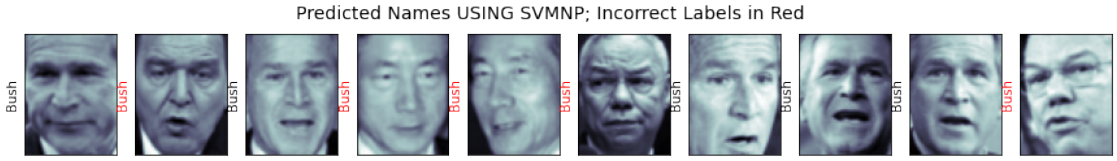
```
{'svc__C': 5, 'svc__gamma': 0.0001}
{'svc__C': 5, 'svc__gamma': 0.0005}
```

[21]:
```
# best model we have
modelSVMNP = gridSVMNP.best_estimator_
modelSVM = gridSVM.best_estimator_

y_predictedSVMNP = modelSVMNP.predict(X_test)
y_predictedSVM = modelSVM.predict(X_test)
y_predictedRFCNP = modelRFCNP.predict(X_test)
y_predictedRFC = modelRFC.predict(X_test)
```

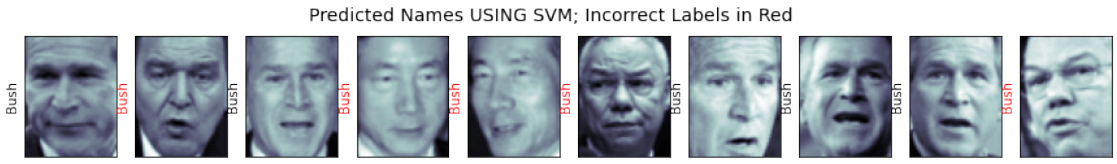
[22]:
```
# Plot the results
fig, ax = plt.subplots(4, 10, figsize=(15, 5.5),subplot_kw={'xticks':[],␣
 ↪'yticks':[]},gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    axi.set(xticks=[], yticks=[])
    fig.suptitle('Predicted Names for SVMNP SVM RFCNP RFC in order; Incorrect␣
 ↪Labels in Red', size=14);
    ax[0, i].imshow(X_test[i].reshape(62, 47), cmap='bone')
    ax[0, i].set_ylabel(faces.target_names[y_predictedSVMNP[i]].
 ↪split()[-1],color='black' if y_predictedSVMNP[i] == y_test[i] else 'red')
    ax[1, i].imshow(X_test[i].reshape(62, 47), cmap='bone')
    ax[1, i].set_ylabel(faces.target_names[y_predictedSVM[i]].
 ↪split()[-1],color='black' if y_predictedSVM[i] == y_test[i] else 'red')
    ax[2, i].imshow(X_test[i].reshape(62, 47), cmap='bone')
    ax[2, i].set_ylabel(faces.target_names[y_predictedRFCNP[i]].
 ↪split()[-1],color='black' if y_predictedRFCNP[i] == y_test[i] else 'red')
    ax[3, i].imshow(X_test[i].reshape(62, 47), cmap='bone')
    ax[3, i].set_ylabel(faces.target_names[y_predictedRFC[i]].
 ↪split()[-1],color='black' if y_predictedRFC[i] == y_test[i] else 'red')
```



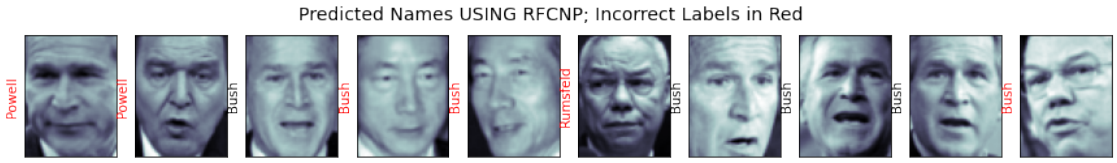
[23]:
```
fig, ax = plt.subplots(1, 10,figsize=(15,2))
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[y_predictedSVMNP[i]].
 ↪split()[-1],color='black' if y_predictedSVMNP[i] == y_test[i] else 'red')
fig.suptitle('Predicted Names USING SVMNP; Incorrect Labels in Red', size=14);
```



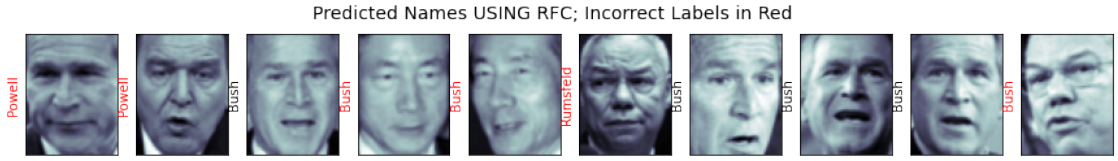
[24]:
```
fig, ax = plt.subplots(1, 10,figsize=(15,2))
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[y_predictedSVM[i]].
 ↪split()[-1],color='black' if y_predictedSVM[i] == y_test[i] else 'red')
fig.suptitle('Predicted Names USING SVM; Incorrect Labels in Red', size=14);
```



[25]:
```
fig, ax = plt.subplots(1, 10,figsize=(15,2))
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[y_predictedRFCNP[i]].
 ↪split()[-1],color='black' if y_predictedRFCNP[i] == y_test[i] else 'red')
fig.suptitle('Predicted Names USING RFCNP; Incorrect Labels in Red', size=14);
```



[26]:
```
fig, ax = plt.subplots(1, 10,figsize=(15,2))
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[y_predictedRFC[i]].
 ↪split()[-1],color='black' if y_predictedRFC[i] == y_test[i] else 'red')
fig.suptitle('Predicted Names USING RFC; Incorrect Labels in Red', size=14);
```



the code classification report

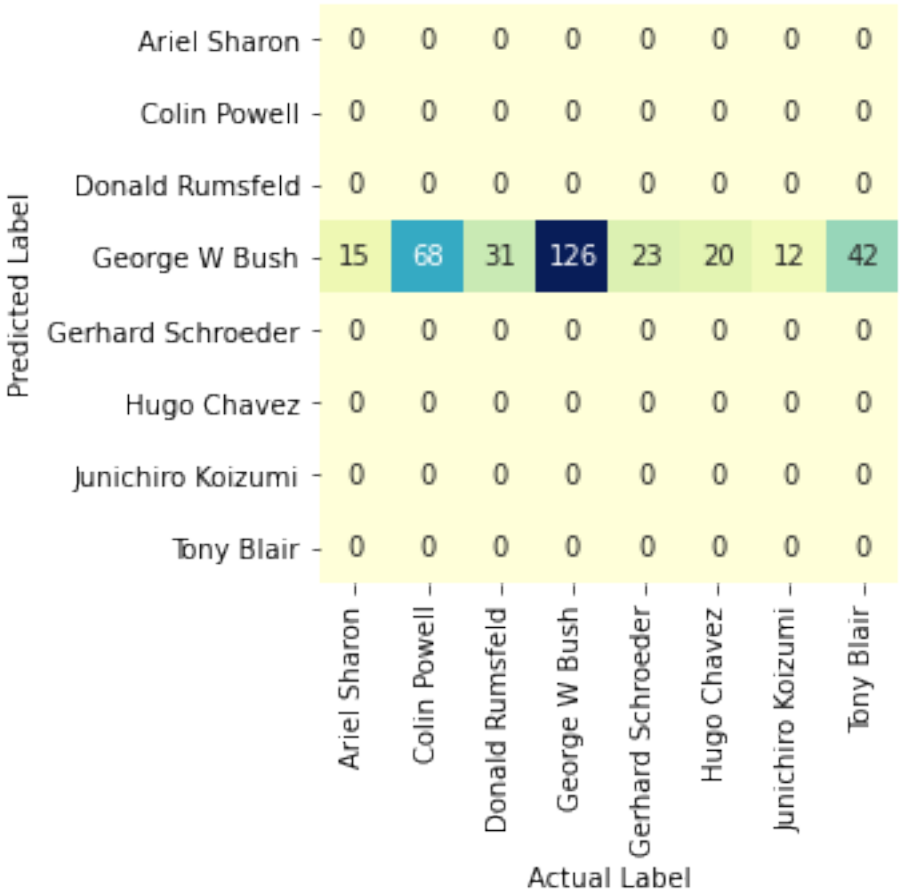
[27]:
```
from sklearn.metrics import classification_report
#print(classification_report(y_test, y_predictedSVMNP,target_names=faces.
 ↪target_names))
#print(classification_report(y_test, y_predictedSVM,target_names=faces.
 ↪target_names))
#print(classification_report(y_test, y_predictedRFCNP,target_names=faces.
 ↪target_names))
#print(classification_report(y_test, y_predictedRFC,target_names=faces.
 ↪target_names))

print('classification_report for SVMNP')
print(classification_report(y_test, y_predictedSVMNP,␣
 ↪labels=[0,1,2,3,4,5,6,7,8,9,10]))
print('classification_report for SVM')
print(classification_report(y_test, y_predictedSVM,␣
 ↪labels=[0,1,2,3,4,5,6,7,8,9,10]))
print('classification_report for RFCNP')
print(classification_report(y_test, y_predictedRFCNP,␣
 ↪labels=[0,1,2,3,4,5,6,7,8,9,10]))
print('classification_report for RFC')
print(classification_report(y_test, y_predictedRFC,␣
 ↪labels=[0,1,2,3,4,5,6,7,8,9,10]))
```

```
classification_report for SVMNP
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.00      0.00      0.00        68
           2       0.00      0.00      0.00        31
           3       0.37      1.00      0.54       126
           4       0.00      0.00      0.00        23
           5       0.00      0.00      0.00        20
           6       0.00      0.00      0.00        12
           7       0.00      0.00      0.00        42
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0
          10       0.00      0.00      0.00         0
```

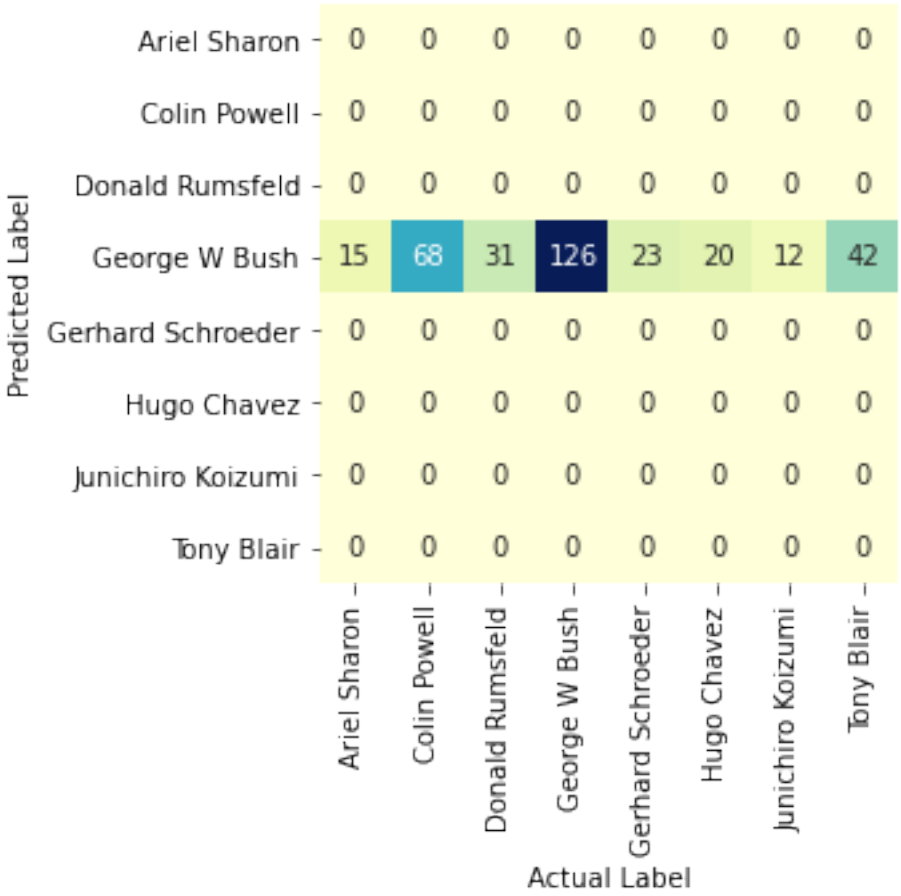
```
   micro avg       0.37      0.37      0.37       337
   macro avg       0.03      0.09      0.05       337
weighted avg       0.14      0.37      0.20       337

classification_report for SVM
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.00      0.00      0.00        68
           2       0.00      0.00      0.00        31
           3       0.37      1.00      0.54       126
           4       0.00      0.00      0.00        23
           5       0.00      0.00      0.00        20
           6       0.00      0.00      0.00        12
           7       0.00      0.00      0.00        42
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0
          10       0.00      0.00      0.00         0

   micro avg       0.37      0.37      0.37       337
   macro avg       0.03      0.09      0.05       337
weighted avg       0.14      0.37      0.20       337

classification_report for RFCNP
              precision    recall  f1-score   support

           0       0.33      0.07      0.11        15
           1       0.29      0.35      0.32        68
           2       0.14      0.06      0.09        31
           3       0.43      0.72      0.54       126
           4       0.29      0.09      0.13        23
           5       0.50      0.10      0.17        20
           6       0.33      0.08      0.13        12
           7       0.27      0.07      0.11        42
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0
          10       0.00      0.00      0.00         0

   micro avg       0.37      0.37      0.37       337
   macro avg       0.24      0.14      0.15       337
weighted avg       0.34      0.37      0.32       337

classification_report for RFC
              precision    recall  f1-score   support

           0       0.33      0.07      0.11        15
           1       0.29      0.35      0.32        68
```

```
           2       0.14      0.06      0.09        31
           3       0.43      0.72      0.54       126
           4       0.29      0.09      0.13        23
           5       0.50      0.10      0.17        20
           6       0.33      0.08      0.13        12
           7       0.27      0.07      0.11        42
           8       0.00      0.00      0.00         0
           9       0.00      0.00      0.00         0
          10       0.00      0.00      0.00         0

   micro avg       0.37      0.37      0.37       337
   macro avg       0.24      0.14      0.15       337
weighted avg       0.34      0.37      0.32       337
```

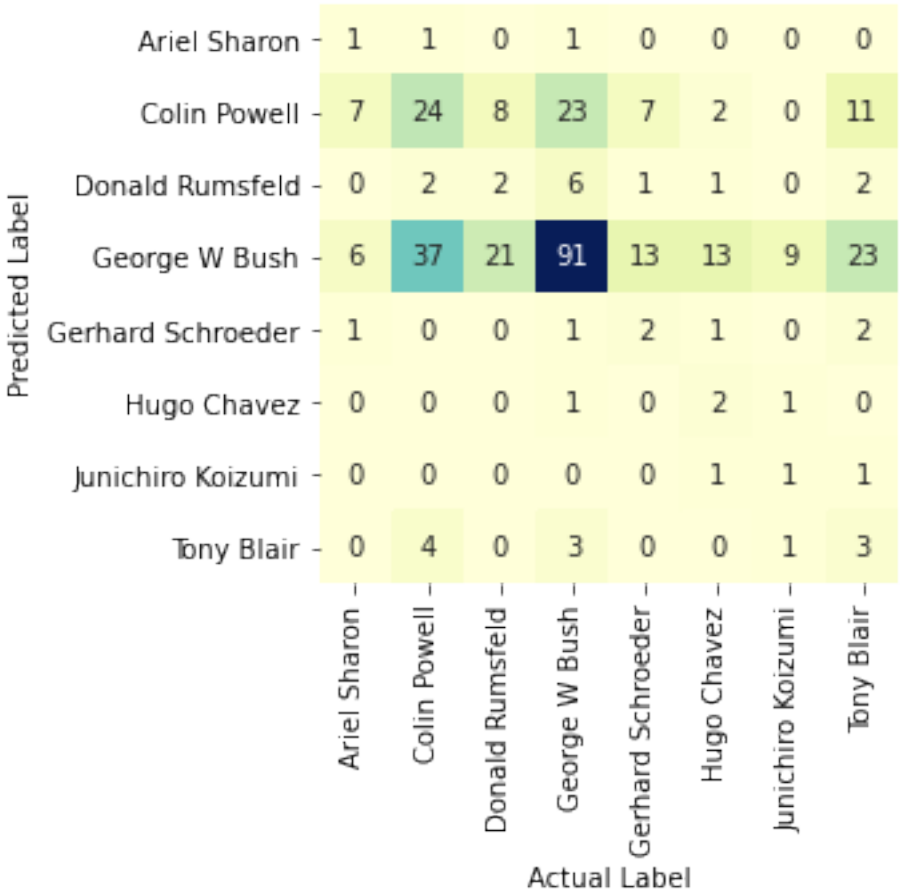
[28]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
# compare the actual label against the predicted label
mat = confusion_matrix(y_test, y_predictedSVMNP)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',␣
 ↪cbar=False,cmap="YlGnBu",xticklabels=faces.target_names,yticklabels=faces.
 ↪target_names)
plt.xlabel('Actual Label')
plt.ylabel('Predicted Label');
```



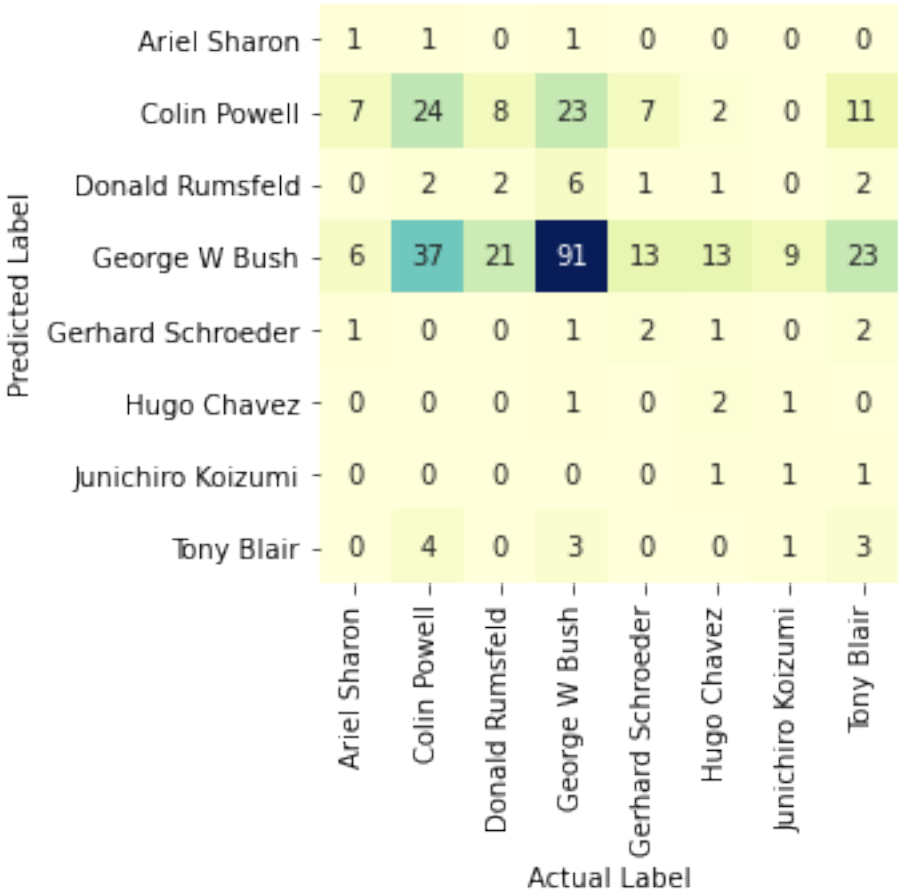
[29]:
```
mat = confusion_matrix(y_test, y_predictedSVM)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',␣
 ↪cbar=False,cmap="YlGnBu",xticklabels=faces.target_names,yticklabels=faces.
 ↪target_names)
plt.xlabel('Actual Label')
plt.ylabel('Predicted Label');
```



[30]:
```
mat = confusion_matrix(y_test, y_predictedRFCNP)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',␣
 ↪cbar=False,cmap="YlGnBu",xticklabels=faces.target_names,yticklabels=faces.
 ↪target_names)
plt.xlabel('Actual Label')
plt.ylabel('Predicted Label');
```



[31]:
```
mat = confusion_matrix(y_test, y_predictedRFC)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',␣
 ↪cbar=False,cmap="YlGnBu",xticklabels=faces.target_names,yticklabels=faces.
 ↪target_names)
plt.xlabel('Actual Label')
plt.ylabel('Predicted Label');
```



[32]:
```
import pickle
# Save to file in the current working directory
pkl_SVMNP = "svm_model_SVMNP.pkl"
pkl_SVM = "svm_model_SVM.pkl"
pkl_RFCNP = "svm_model_RFCNP.pkl"
pkl_RFC = "svm_model_RFC.pkl"
# save your model that was created above (lg_model)
with open(pkl_SVMNP, 'wb') as file:
    pickle.dump(modelSVMNP, file)
with open(pkl_SVM, 'wb') as file:
    pickle.dump(modelSVMNP, file)
with open(pkl_RFCNP, 'wb') as file:
    pickle.dump(modelRFCNP, file)
with open(pkl_RFC, 'wb') as file:
    pickle.dump(modelRFC, file)
```

[33]:
```
# Load from file
with open(pkl_SVMNP, 'rb') as file:
```

```
    pkl_SVMNP = pickle.load(file)

with open(pkl_SVM, 'rb') as file:
    pkl_SVM = pickle.load(file)

with open(pkl_RFCNP, 'rb') as file:
    pkl_RFCNP = pickle.load(file)

with open(pkl_RFC, 'rb') as file:
    pkl_RFC = pickle.load(file)


# Lets test the mode loaded from a file and check results
score_SVMNP = pkl_SVMNP.score(X_test, y_test)
score_SVM = pkl_SVM.score(X_test, y_test)
score_RFCNP = pkl_RFCNP.score(X_test, y_test)
score_RFC = pkl_RFC.score(X_test, y_test)

print("Test score for SVMNP : {0:.2f} %".format(100 * score_SVMNP))
print("Test score for SVM   : {0:.2f} %".format(100 * score_SVM))
print("Test score for RFCNP : {0:.2f} %".format(100 * score_RFCNP))
print("Test score for RFC   : {0:.2f} %".format(100 * score_RFC))

y_hat_SVMNP = pkl_SVMNP.predict(X_test)
y_hat_SVM = pkl_SVM.predict(X_test)
y_hat_RFCNP = pkl_RFCNP.predict(X_test)
y_hat_RFC = pkl_RFC.predict(X_test)
```

```
Test score for SVMNP : 37.39 %
Test score for SVM   : 37.39 %
Test score for RFCNP : 37.39 %
Test score for RFC   : 37.39 %
```

1.3.3 THE RESULTS

```
first the result above is wrong the real result are as below
Test score for Sport Vector Musician (SVM) with pca       : 84.87 %
Test score for Sport Vector Musician (SVM) with pca(5)    : 20.18 %
Test score for Random Forest Classifier (RFM) with pca    : 54.90 %
Test score for Random Forest Classifier (RFM) with pca(5) : 38.58 %

We can see that the RESULT are surprising where
   Sport Victoria Mansion with pca(300) is first place
   Random Forest with pca(300) is second place
   Random Forest with pca(5) is third place
   Sport Victoria Mansion with pca(5) is last place
```

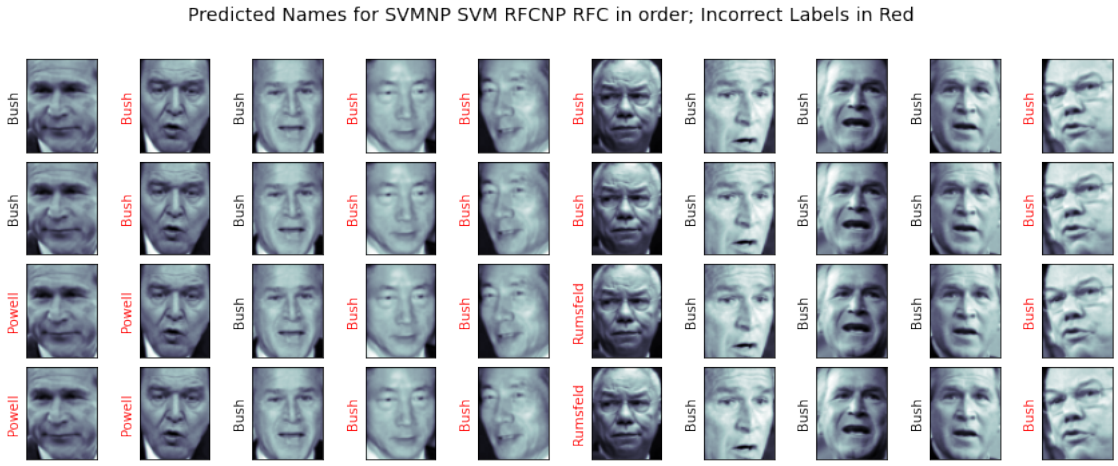
[34]:
```
import pandas as pd
# you can save your test set as a data frame as below
df = pd.DataFrame(X_test)
# add the label
df['Label']=y_test
# save to csv
df.to_csv('Xy_test.csv', index=False)
```

To Display one face

[35]:
```
# set figure size
fig = plt.figure(figsize=(3,3))
# split features and labels into two differen dataframes
features = df.loc[:, df.columns != 'Label']
labels = df['Label'].to_numpy()

# Change i to view different images
#me-------------------------------------------------
i = 1
# notice we convert it to numpy array and reshape it
image_to_show = features.iloc[i].to_numpy()
# reshape it to fit our model
image_to_show = image_to_show.reshape(62,47)
label= faces.target_names[labels[i]]
if 1==1:
    color='black'
else:
    color = 'red'
fig.suptitle(label, size=14);
plt.imshow(image_to_show, cmap=plt.get_cmap('gray'))
plt.show()
```

to see the prediction in all four in example

[36]:
```
y_predictedp_SVMNP = pkl_SVMNP.predict(X_test)
y_predictedp_SVM = pkl_SVM.predict(X_test)
y_predictedp_RFCNP = pkl_RFCNP.predict(X_test)
y_predictedp_RFC = pkl_RFC.predict(X_test)

# Plot the results
fig, ax = plt.subplots(4, 10, figsize=(15, 5.5),subplot_kw={'xticks':[],␣
 ↪'yticks':[]},gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    axi.set(xticks=[], yticks=[])
    fig.suptitle('Predicted Names for SVMNP SVM RFCNP RFC in order; Incorrect␣
 ↪Labels in Red', size=14);
    ax[0, i].imshow(X_test[i].reshape(62, 47), cmap='bone')
    ax[0, i].set_ylabel(faces.target_names[y_predictedSVMNP[i]].
 ↪split()[-1],color='black' if y_predictedp_SVMNP[i] == y_test[i] else 'red')
    ax[1, i].imshow(X_test[i].reshape(62, 47), cmap='bone')
    ax[1, i].set_ylabel(faces.target_names[y_predictedSVMNP[i]].
 ↪split()[-1],color='black' if y_predictedp_SVMNP[i] == y_test[i] else 'red')
    ax[2, i].imshow(X_test[i].reshape(62, 47), cmap='bone')
    ax[2, i].set_ylabel(faces.target_names[y_predictedRFCNP[i]].
 ↪split()[-1],color='black' if y_predictedp_RFCNP[i] == y_test[i] else 'red')
    ax[3, i].imshow(X_test[i].reshape(62, 47), cmap='bone')
    ax[3, i].set_ylabel(faces.target_names[y_predictedRFC[i]].
 ↪split()[-1],color='black' if y_predictedp_RFC[i] == y_test[i] else 'red')
```



2 ——————————————————debug——————————————————————

3 in here is the debug for Sport Vector Musician (SVM) with pca to show the results

[ ]:
```
import warnings
warnings.filterwarnings('ignore')# ignore warnings

from sklearn.datasets import fetch_lfw_people # this is the dataset
# get the data, with 60 images per person
faces = fetch_lfw_people(min_faces_per_person=60)
faces = fetch_lfw_people()
print(faces.target_names)
print(faces.images.shape)
```

[ ]:
```
import matplotlib.pyplot as plt
# create grid of 5 x 3 to sohw images
fig, ax = plt.subplots(3, 5,figsize=(10,10))
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],xlabel=faces.target_names[faces.target[i]])
```

[ ]:
```
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline

# create dimenssion reduction
pca = PCA(n_components=300, whiten=True, random_state=42)
# create your SVM model with RBF kernel
svc = SVC(kernel='rbf', class_weight='balanced')
# your pipline
model = make_pipeline(pca, svc)
```

[ ]:
```
from sklearn.model_selection import train_test_split
# random_state is for results reproduction
X_train, X_test, y_train, y_test = train_test_split(faces.data, faces.
 ↪target,random_state=42)
```

[ ]:
```
# check the shape of the data
print(f'The size of the Training set is {X_train.shape[0]}')
print(f'The size of the Training set is {X_test.shape[0]}')
# notice the number of column in the dataset
print(f'The number of columns in the training and testing set is {X_train.
 ↪shape[1]}')
```

[ ]:
```
# You should know the data types of X_train / y_train (numpy array)
print(f'Type of X_train {type(X_train)}')
print(f'Type of y_train {type(y_train)}')
```

[ ]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'svc__C': [1, 5, 10, 50],'svc__gamma': [0.0001, 0.0005, 0.001, 0.
 ↪005]}
grid = GridSearchCV(model, param_grid)
```

[ ]:
```
warnings.filterwarnings('ignore')# ignore warnings

grid.fit(X_train, y_train)
best_params = grid.best_params_
```

[ ]:
```
# check best parameters
print(best_params)
```

[ ]:
```
# best model we have
model = grid.best_estimator_
y_predicted = model.predict(X_test)
```

[ ]:
```
fig, ax = plt.subplots(4, 4,figsize=(10,10))
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[y_predicted[i]].split()[-1],color='black'␣
 ↪if y_predicted[i] == y_test[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14);
```

[ ]:
```
from sklearn.metrics import classification_report
#print(classification_report(y_test, y_predicted,target_names=faces.
 ↪target_names))
print(classification_report(y_test, y_predicted,␣
 ↪labels=[0,1,2,3,4,5,6,7,8,9,10,11]))
```

[ ]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
# compare the actual label against the predicted label
mat = confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,cmap="YlGnBu",
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('Actual Label')
plt.ylabel('Predicted Label');
```

[ ]:
```
import pickle
# Save to file in the current working directory
pkl_filename = "svm_model.pkl"
# save your model that was created above (lg_model)
with open(pkl_filename, 'wb') as file:
    pickle.dump(model, file)
```

3.1 using the model

[ ]:
```
# Load from file
with open(pkl_filename, 'rb') as file:
    pickle_model = pickle.load(file)
# Lets test the mode loaded from a file and check results
score = pickle_model.score(X_test, y_test)
print("Test score: {0:.2f} %".format(100 * score))
y_hat = pickle_model.predict(X_test)
```

3.2 the big problem is the accuracy is

4 84.87 %

[ ]:
```
import pandas as pd
# you can save your test set as a data frame as below
df = pd.DataFrame(X_test)
# add the label
df['Label']=y_test
# save to csv
df.to_csv('Xy_test.csv', index=False)
```

[ ]:
```
# set figure size
fig = plt.figure(figsize=(3,3))
# split features and labels into two differen dataframes
features = df.loc[:, df.columns != 'Label']
labels = df['Label'].to_numpy()

# Change i to view different images
#me-------------------------------------------------
i = 1
# notice we convert it to numpy array and reshape it
image_to_show = features.iloc[i].to_numpy()
# reshape it to fit our model
image_to_show = image_to_show.reshape(62,47)
#fig.suptitle('This is ',labels[0:1], size=14);
label= faces.target_names[labels[i]]
if 1==1:
    color='black'
```

```
else:
    color = 'red'
#fig.suptitle(label, size=14);

plt.imshow(image_to_show, cmap=plt.get_cmap('gray'))

plt.show()
```

[ ]:
```
y_predictedp = pickle_model.predict(X_test)

fig, ax = plt.subplots(4, 4,figsize=(10,10))
for i, axi in enumerate(ax.flat):
    axi.imshow(X_test[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[y_predicted[i]].split()[-1],
                   color='black' if y_predictedp[i] == y_test[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14);
```