ELEC3612 Pattern Recognition and Machine Intelligence Project Report 1

Hangyu Chen (490075948)

Abstract

The Sheffield Face Database was used to achieve a human face classification task in this project. Each person's face is shown in a range of poses, which are from profile to frontal views. The whole dataset is separated into directories labelled 1a, 1b, … 1t and images are numbered consecutively as they were taken.

Some basic Machine Learning techniques and algorithms are used and we got the final accuracy based on the confusion matrix:

- As for SVM with Polynomial kernel, we finally got the accuracy of 98%.
- As for manual binary Logistic Regression classification with OvR (OvA), we finally got an accuracy of 69%. And we compare the result using the sklearn Logistic Regression package, which got an accuracy of 99.1%.
- As for LDA, we finally got a classification accuracy of 99.1%.

Besides, PCA and LDA are used for feature reduction and classification, and we got the Eigen-faces and Fisher-faces respectively.

Section I: Aims and Background

According to Danijel. et.al (2006), visual learning and recognition have become an important and popular research topic in the computer vision area, and different methods have been proposed in recent years. [1]

Firstly, as we know that this dataset is made up of 20 subfolders, and each of them contains one person's face images in different poses, we could easily use the supervised algorithms for classification. Secondly, an image could be considered as a matrix with a size of 112 x 92, which means there are 10304 pixels and each pixel has its corresponding pixel value (in the range of 0 to 256 because of the grey-scale image). In this case, the label would be the index of the person's subfile, the data would be the image data (which is considered as a list after flattening the matrix).

As we define this classification task as supervised. The techniques/algorithms used in this project to implement the human face classification task for Sheffield Face Database are:

1. Using different sampling methods (Random split/sampling and Stratified sampling as improvement) from Dataset to get training and testing set.
2. Support-Vector-Machine (SVM) with Gaussian and Polynomial kernel for classification. Then we also consider the overfitting problem.
3. Principle-Component-Analysis (PCA) for feature extraction (image matrix dimension reduction) and we got the Mean-face and Eigen-face. Then, we used the eigenfaces for reconstructed the image.
4. Binary Logistic Regression and OvA (OvR) algorithms for classification.
5. LDA for multi-class classification and got the Fisher-face.

The whole process is separated into 5 phases which could be easily found in the. ipynb file:

1. Load the dataset and split it into training and testing datasets.
2. Using package SVM for classification with improvement.
3. PCA Calculation with Eigen-faces.
4. Binary Logistic Regression Calculation + (OvR/OvA) with improvement
5. Using LDA for classification

Finally, at the end of each technique/algorithm, we used different evaluation methods such as confusion matrix, accuracy value, *classification_report* package in sklearn which contains a precision, recall, F1 score, support number, and ROC curve with AUC area.

Section II: Team Contributions

There are four people in our group. Each of us has done some checking and updating tasks for the whole project,

but the main task for each person is:

- Hangyu Chen is responsible for separating the dataset randomly, using SVM for classification, setting up binary Logistic Regression algorithm and using OvA (OvR) for classification and package LDA for classification.
- Mason Gao is responsible for using PCA to get the Eigen-faces.
- Steve Wu is responsible for training the binary Logistic Regression classifiers (number 13 to 19).
- Haoran Yang is responsible for using Stratified sampling to get the training and testing dataset as the improvement of Logistic Regression. And training the binary Logistic Regression classifiers (number 7 to 12).

Section III: Methodology

To achieve the classification task, we have to implement the techniques/algorithms needed first. In this part, the techniques used are introduced in general, and a detailed explanation will be shown in **Section IV**

As for data preparation (including data loading, training set and test set separating), we used the *torch.utils.data.random_split()* function in PyTorch and *train_test_split()* function in sklearn for implementing random split and Stratified sampling respectively.

As for the SVM, we used the package from sklearn with different kernels in order to solve for high-dimension data. However, the proper parameters are needed to choose so that it will avoid the overfitting problem.

Besides, to extract the main features from the images, reducing the calculation and time needed and further help with training, we used Principal Component Analysis (PCA) for the data. This algorithm is developed by ourselves and an example was made to check its feasibility. After having the same result with the PCA package from sklearn, it could be denoted that our PCA algorithm is correct. In this case, we used our PCA and extract the top 10 features (eigenvectors) and make the Eigen-face.

Then, we used Logistic Regression for solving this multi-class classification problem. The way is to set up and train 20 binary Logistic Regression classifiers and apply OvR for each of them. Then we could combine the prediction results and get the final one based on the probability.
Finally, we used Linear Discriminant Analysis (LDA) package in sklearn for classification.

Section IV: Results

A. Data Organization

1. Data Modeling

1) This step is sometimes considered to be a high-level and abstract design phase also referred to as conceptual design [3], which could describe the data contained in the database, their relationships, and the constraints on data.

2) We used *ImageFolder()* function to read all the images inside the cropped.zip file as our dataset. In this case, each subfile (containing images of a person in different poses) will correspond to a number which is the label of a certain person. Besides, we could find the length of the dataset is 575, which is the total number of images. And 20 subfiles are corresponding to 20 people (labels from 0 to 19). However, the number of images of a certain person is not always the same.

3) We set up the dataset in this way because we aimed to use some supervised algorithms for classification. As we assign people's images with a certain label (number), we could easily train and test the data using different classifiers later.

2. How to organize the training set and test set.

1) The training set is the dataset that we used for training the model we used so that the model would learn the hidden features/patterns in the data.
The test set is the dataset that we used to test the performance of the model after training. Based on the accuracy result, we could check if the model performance is reliable.

2) In this project, we separate the training set and test set in two ways: random split and Stratified sampling.

a) We first separate the dataset and randomly make 80% of the data as the training set and the rest of 20% the test set.

b) After using the datasets after random splitting for SVM and Logistic Regression, it is found that the training performance for Logistic Regression is not good enough. In this case, to have more reasonable and reliable performance, these two datasets are separated by using Stratified sampling based on the labels. And 80% of the images in each subfile (each person) will be selected as the training set. The rest of them will be selected as the test set.

3) The reason we choose 80% of the images as a training set is that the classifier could learn more unseen features. Besides, the reason we used Stratified sampling is that the dataset we used is small (only 575 images). In this case, if the number of images on each label has a large difference, it will have a negative effect when training the model, which means the model will not learn many features and further affect the test result.

B. Training, Testing and Evaluation

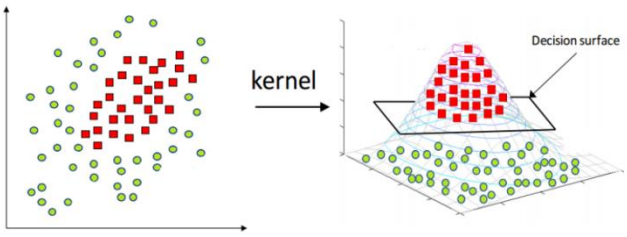
1. SVM

1) Introduction

Support Vector Machine (SVM) is a supervised machine learning algorithm that is widely used in classification objectives. Each feature of the data is the value of a particular coordinate. In this project, as we have multi-dimension data after flattening the images, we need to have a method to find the hyperplane with SVM to separate the data and complete the classification.

2) Implement methods with reason and Overfitting:

We used the package SVM from sklearn. But kernel SVM is needed to use for this un-separable data.
One simple method for solving un-separable data is to map the sample features to the high-level space. However, when there are more and more dimensions, computations within that space become more and more expensive.
In this case, the kernel trick offers a more efficient and less expensive way to transform data into higher dimensions so that it will help with classification. [2]



1 Kernel trick

https://miro.medium.com/max/1400/1*mCwnu5kXot6buL7jeIafqQ.png

In this project, we used Polynomial and Gaussian kernels for classification.

- Gaussian kernel SVM with RBF as Radial Basis Function uses the function below to map the original data:

  $$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

  And here as we used package SVM, we used the RBF function below:

  $$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma\|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

  where $\gamma = 1/2\sigma^2$ for $\gamma > 0$

- Polynomial kernel SVM uses the function below:

  $$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

  The basic idea is mapping low-dimension data to higher-dimension. The value of degree (d) is a regularization variable. To avoid too much computation needed using too many dimensions, we set the maximum degree as 3.

When using the default value of Polynomial kernel SVM, the result accuracy is 100%. However, it is not a proper result in machine learning which is called "overfitting". It means our model fits exactly against this dataset. However, it means this model cannot perform accurately against new/unseen data which is not in this dataset. So, we changed the parameter of C (which is called the Regularization parameter) for both Polynomial and Gaussian kernel SVM, and this will be shown in section **D. Improvement.**

The current result accuracy when using the default parameter is 0.982609 for Gaussian kernel and 1.00 (overfitting) for Polynomial kernel respectively.

2. Logistic Regression

1) Introduction

Logistic Regression is a supervised method for predictive analysis. It is used to predict a binary

outcome based on a set of independent variables. And in this project, as we have a multi-class dataset, OvR will be used with 20 binary Logistic Regression classifiers so that we could implement the classification task.

2) Implement methods with reason:

We implement it for multi-class classification manually. The overall steps are:

a) Set up binary Logistic Regression class.

b) To achieve the OvR algorithm, we separate each training and test set into 20 subsets based on the label.

c) Training 20 different classifiers so that each classifier could only classify images of a certain label.

d) Based on the predicted results from these 20 classifiers and the corresponding probability results from the sigmoid function in the Logistic Regression, develop an algorithm to get the final prediction result.

A general explanation of our manual Logistic Regression:

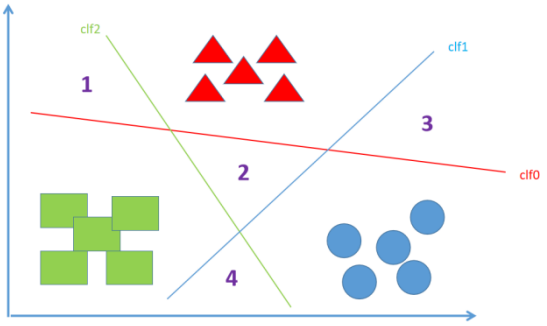
a) Hypothesis function: we used the sigmoid function which would help with prediction. The general representation is: $y = \frac{1}{1+e^{-z}}$. Replace z as $\omega^T x + b$ we could get: $y = \frac{1}{1+e^{-(\omega^T x+b)}}$. And if the probability result returned by sigmoid function is larger than 0.5, we assign this prediction result as 0 (True).

b) Loss function: we aim to minimize the loss between the prediction and ground truth. The loss function (log-loss) in Logistic Regression is: $L = -[y log\hat{y} + (1-y)\log(1-\hat{y})]$, and because it is a binary classification method, y can only have two values: 0 and 1. In this case, L would be $-\log(\hat{y})\ when\ y = 1$ or $-\log(1-\hat{y})\ when\ y = 0$

c) Optimization: We use gradient descent to find optimal $Omega\ and$ b step by step.

And the algorithm we designed for solving the trick that OvR with binary classifiers is shown below:

The trick is shown in the diagram below. For example, when we deal with the multi-class classification (3 classes below), the new data which are in the area marked as 1, 3 and 4 will have two prediction results. And the new data which are in the area marked as 2 will have no prediction result. So, we have to develop an algorithm to solve this problem.



2 OvR trick

The algorithm is shown below:

Algorithm: Solving the binary classification trick

Input:

1. Lists of prediction results (i.e., y_test_pred_0, …, y_test_pred_19) from each of the binary Logistic Regression classifiers.
2. Lists of probability results (i.e., y_test_pred_proba_0, …, y_test_pred_proba_19) from each of the binary Logistic Regression classifiers.

Step 1: Combine the value in the prediction results and the probability results into two new arrays in order (from index 0 to index 115 which is the length of the test set), named *combine_labellist* and *prob_list*.

Step 2: In the for loop, firstly find the index of the 0s in each of the prediction results. (0 means the prediction is True) and save the index into a new list named *equallist*

Step 3: If the length of the *equallist* is equal to 1, which means there is only one classifier that predicts the image as True. We then directly label this image as the classifier's index.

Step 4: If the length of the *equallist* is larger than 1, which means there are at least two classifiers that predict the image as True. We then find the most-likely label that the image will be predicted as 0 according to their probabilities from the sigmoid function. (i.e., find the maximum probability of 0)

Step 5: If the length of the *equallist* is smaller than 1, which means no classifier predicts the image as True. We then find the most unlikely that the image will be predicted as 1 according to their probabilities from the sigmoid function. (i.e. find the minimum probability of 1)

Overall idea: The final prediction is based on the prediction results and the corresponding probability results from each classifier.

C. Evaluation

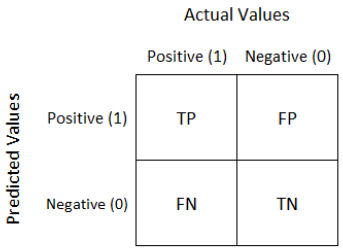
1) We need to evaluate the performance using the test set to check if a model is reliable/suitable for the classification. In this project, we used the confusion matrix to evaluate the performance of each classification technique/algorithm. The idea of the confusion matrix is to count the number of times when instances of class A are classified as class B.

2) There are four areas of a confusion matrix: TP, TN, FP, and FN.

- TP (True Positive): represents the data in which both prediction and ground truth are True.
- TN (True Negative): represents the data in which both prediction and ground truth are False.
- FP (False Positive): represents the data in which prediction is True and ground truth is False.
- FN (False Negative): represents the data in which prediction is False and ground truth is True.

3) The are some matrices for representing the performance based on the four values (TP, TN, FP, FN).

- Accuracy = (TP+TN)/(TP+TN+FN+FP), represents the ratio that the data is correctly classified.
- Precision = TP/(TP+FP), represents the ratio of the number of samples that are correctly predicted to be positive to the total number of samples that are predicted to be positive. In other words, “How many retrieved items are relevant”.
- Recall = TP/(TP+FN), represents the ratio of the number of samples that are correctly predicted to be positive samples to the number of samples that are actually positive. In other words, “How many relevant items are retrieved”.
- Sensitivity (True Positive Rate), refers to the probability of a positive test, conditioned on truly being positive.
- Specificity (True Negative Rate), refers to the probability of a negative test, conditioned on truly being negative.
- F1-Score = 2/ (1/Precision + 1/Recall) = TP/ (TP + (FN+FP)/2), represents the harmonic mean of precision and recall.
- The confusion matrix shows the number of correct and incorrect predictions with count values and is broken down by each class.
  For a binary classification:



But here we have a multi-class classification with 20 different classes. So, the confusion matrix should be 20*20.

Actual / Predict	Label 0	Label 1	…	Label 19
Label 0				
Label 1				
…				
Label 19				

A perfect (100% accuracy) confusion matrix only has values in its diagonal.

- ROC is a curve whose x-axis is the False Positive Rate, and the y-axis is the Sensitivity (True Positive Rate). The larger the FPR, the more actual negative classes are in the predicted positive class. Similarly, the larger the TPR, the more actual positive classes are in the predicted positive class.
- AUC is the area under the ROC curve, the larger the AUC, the better the performance of this classifier.

4) After each of the classification techniques, we used confusion matrix, accuracy_score, and classification report (including precision, recall, f1-score) functions in sklearn. Besides, we used the ROC curve to represent the relationship between TPR and FPR for the binary Logistic Regression classifier.
We implement them by importing these packages from sklearn:

```
from sklearn import datasets, metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss
from sklearn.metrics import roc_curve, auc
```

1 Matrix used for result

And for each matrix, we simply use the prediction result from a certain classifier and the ground truth as our two inputs for the matrix. And the images below show the result when using Gaussian kernel SVM:

```
Confusion Matrix :
[[ 7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  4  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 10  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  1  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 10  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 10]]
Accuracy Score is 0.9826086956521739
```

```
Accuracy Score is 0.9826086956521739
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00         7
           2       1.00      1.00      1.00         6
           3       1.00      1.00      1.00         4
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         4
           7       1.00      0.80      0.89         5
           8       0.50      1.00      0.67         1
           9       1.00      1.00      1.00        10
          10       1.00      1.00      1.00         4
          11       1.00      1.00      1.00         4
          12       1.00      1.00      1.00         9
          13       1.00      1.00      1.00         9
          14       1.00      0.67      0.80         3
          15       0.67      1.00      0.80         2
          16       1.00      1.00      1.00         8
          17       1.00      1.00      1.00         6
          18       1.00      1.00      1.00        10
          19       1.00      1.00      1.00        10

    accuracy                           0.98       115
   macro avg       0.96      0.97      0.96       115
weighted avg       0.99      0.98      0.98       115
```

2 Gaussian kernel SVM result

And the ROC curve for binary Logistic Regression is shown in the "**Section D**. Improvement **Part5** Improve the performance of Logistic Regression".

5) Reason for using each matrix:

a) The reason why we used the confusion matrix is that we could easily check out how many images are correctly predicted for each label. A 100% accuracy confusion matrix only has values on its diagonal.

b) The accuracy score shows the accuracy of the classifier, which is calculated by:

$$\frac{number\ of\ data\ which\ correctly\ predicted}{total\ number\ of\ data}$$

c) The classification report contains the value of precision, recall, F1-score and the number of support data.

d) Depending on different situations, we usually have different weights for precision and recall. But in this project, we aim to classify the people's image, so we simply consider these two values have the same weights, which means the higher the F1-score, the better the classifier's performance on classifying this certain label.

e) And support shows how many samples of this label are in the test set, when this value is larger, the F1 score has higher reliability.

f) As ROC curve and only be used in binary classification. In this case, we used it in the binary Logistic Regression classifier and shows the AUC of those 20 binary classifiers used in OvR.

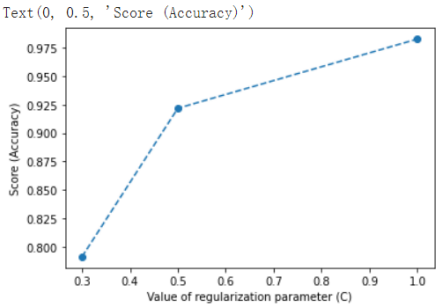
D. Improvement

4. Improve the performance of SVM

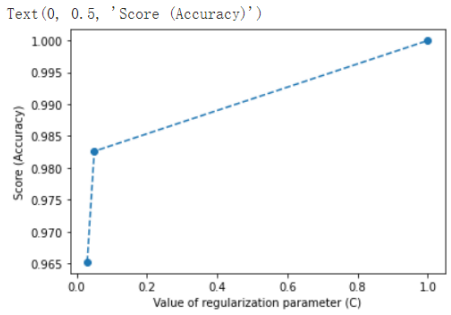
Because the dataset is small, we need to change the parameter of the SVM so that we could improve the generalization ability/performance of the original model. So that the model will have a good prediction for those data which are not in this dataset.

In this case, we changed the parameter of C (which is called the Regularization parameter) for both Polynomial and Gaussian kernel SVM. It is used to balance Support Vector Complexity and Misclassification Rate. When C is larger, our loss function will also be larger, which means that we are reluctant to drop farther away outliers. In this way, we will have more support vectors, which means that the model of support vectors and hyperplanes will become more complex and easier to overfit.

In this case, we reduced the value of the regularization parameter and keep the other parameters the same. Finally, we got an accuracy of 0.93 and 0.96 (using confusion matrix) after setting the regularization parameter as 0.5 and 0.03 respectively for Gaussian and Polynomial kernel.



3 Gaussian kernel Result



4 Polynomial kernel Result

And we could visualize the results (Ground Truth and Prediction) for the first 5 images in the test set.

5 Visualization

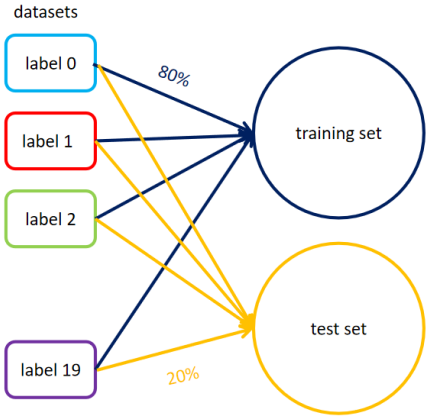
5. Improve the performance of Logistic Regression

Besides, the reason we used Stratified sampling is that the dataset we used is small (only 575 images). In this case, if the number of images on each label has a large difference, it will have a negative effect when training the model, which means the model will not learn many features and further affect the test result.

As for the improvement, we used Stratified sampling for our dataset. In this part, we take 80% of the images randomly from each label (i.e., in each subfile in the original dataset, we randomly take 80% of the images inside and combine them as the training set, the rest of them are in the test set).

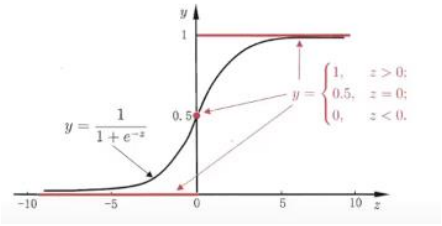
The code used for Stratified Sampling is train_test_split function in sklearn below:

```
sklearn.model_selection.train_test_split(all_img_list,all_label_list,test_size = 0.2, random_state = 42, shuffle = True, stratify = all_label_list)
```



After using Stratified sampling, the accuracy increased from 64% to 69%.

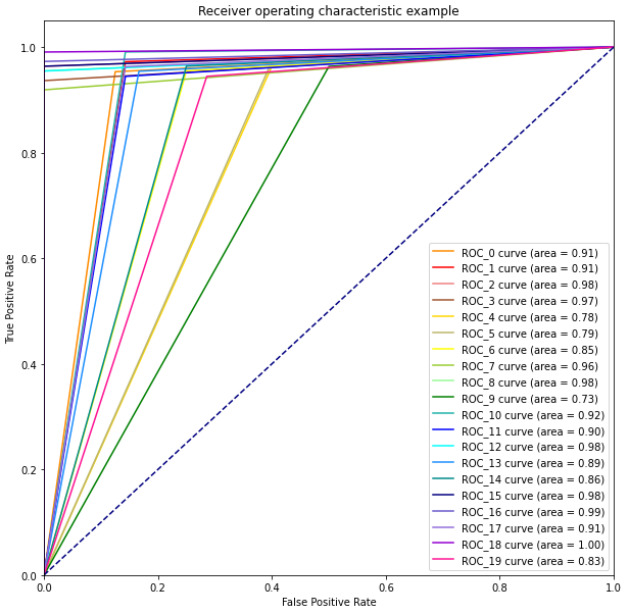
Besides, we could find the accuracy of using binary Logistic Regression and OvR is not as good as using other techniques. It is because we used the sigmoid function in the Logistic Regression and set 0.5 as the threshold (i.e., If the possible is larger than 0.5, this data will be considered as Positive).



6 Sigmoid function

But in our manual Logistic Regression, in order to balance the weight of Precision and Recall, we set this value as 0.5.

Finally, we could use the confusion matrix and ROC curves of each classifier to show the result:



7 ROC curve for OvR Logistic Regression

E. PCA

1. Use PCA for training

1) PCA (Principal Component Analysis) is a common unsupervised data analysis method, which is often used for dimensionality reduction of high-dimensional data and can be used to extract the main feature components of the data, minimizing the information loss.

2) We implement the PCA by coding using NumPy.

Algorithm: Manual PCA
Input: 1. x: the input data 2. n_component: the number of the eigenvalue/eigenvector will be saved
Step 1: Zero-centered each row and minus the mean value of this row **Step 2**: Using *np.cov()* to calculate the covariance matrix **Step 3**: Find the eigenvalues and eigenvectors using *np.linalg.eigh()* **Step 4**: Find the maximum *n_component* number of eigenvalue and their corresponding eigenvectors. Each of the eigenvectors is an Eigen-face.

Besides, to check our manual PCA's feasibility, we set up a small example to check if the result is the same as using the package PCA from sklearn. And we find that the results are the same, which means our PCA is correct.

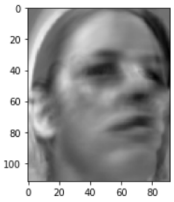
3) Then, we could apply PCA to our datasets.
Firstly, we try to extract the top 10 features from the first person's images (label 0) and reconstructed the image using these 10 principal components.

```
for i in range(0,len(new_label_0_array)):
    reconstructed_img_0 = np.mean(new_image_0_array) + np.matmul(img_0_pca[i],pca.components_)
print(reconstructed_img_0.shape)
```

```
(10304,)
```

```
plt.imshow(reconstructed_img_0.reshape(112,92),cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7fae2d302450>
```

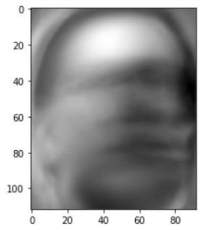


8 Reconstructed image for the first person

Then, we extract the top 100 features from all the images in the dataset, which means we need to combine the largest 100 eigenvectors. The result is shown below, and the component has a size of 112*92.

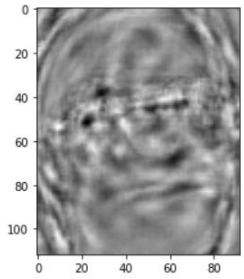
```
plt.imshow(pca.components_[0].reshape(112,92),cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7fae2d21cd50>
```



9 The first principal component

```
(10304,)
<matplotlib.image.AxesImage at 0x7fae2d4bc550>
```



10 Combining all the first 100 components

3) With PCA, we extract the top 100 features of the images, which means we could significantly reduce the computation needed when training and testing the data (compared with using original images in the size of 112*92). In other words, the dataset after PCA has a size (575, 100) rather than (575, 10304), which means we represent the high-dimension data as low-dimension.

F. LDA

1. Use LDA for training

1) Linear discriminant analysis (LDA) is a useful tool for dimension reduction and classification. Different from PCA, it is a supervised algorithm.
It selects a projection hyperplane in the n-dimensional space so that the distance between the projections of the same classes on the hyperplane is as close as possible, and the distance of different classes is as far as possible.

2) In this case, we firstly used package LDA from sklearn to do the classification task, and use manual LDA to find the eigenvectors and eigenvalues. Firstly, we choose the number of components as 10. This value represents the dimensions that we want to reduce to, and this value has to be smaller than the number of classes minus one. And as LDA is a supervised algorithm, we have to make the data with their labels as inputs. Different with the binary LDA, as we have a multi-classes classification, the projection will not be a line, but a hyperplane.

Algorithm: Solving multi-class LDA classification

Input:

1. X which is the data
2. y which is the corresponding label
3. n_components = d which is the dimension reduced to

Step 1: Calculate

$\mu_j(j=1,2...k)$: the mean vector of sample jth

$\Sigma_j(j=1,2...k)$: the covariance matrix of sample jth

Step 2: Find matrix W which is an n*d matrix that contains a basis vector (w1,w2,...wd)

Step 3: The equation we would optimize is $\frac{W^T S_b W}{W^T S_w W}$

Where $S_b = \sum_{j=1}^{k} N_j(\mu_j - \mu)(\mu_j - \mu)^T$ where μ is the mean vector of all the samples.

Besides $S_w = \sum_{j=1}^{k} S_{wj} = \sum_{j=1}^{k} \sum_{x \in X_j} (x - \mu_j)(x - \mu_j)^T$

Step 4: The equation we want to optimize become:

$$arg\ max\ J(w) = \frac{\prod_{diag} W^T S_b W}{\prod_{diag} W^T S_w W}$$

3) As we used the package LDA in sklearn for classification firstly, we simply input our images in the training set as X and our labels in the training set as y, and the n_components as 10.

```
lda = LinearDiscriminantAnalysis(n_components=10)
lda.fit(X, y)
```

11 Use LDA for training

Then we could predict the result in the test set using this LDA classifier.

```
pre_y = lda.predict(test_my_np_array)
```

12 Do the predication

Finally, we could check the results of this classifier using the confusion matrix and other matrices mentioned before.

```
Confusion Matrix :
[[ 7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  5  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 10  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  4  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  3  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 10  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  9]]
Accuracy Score is 0.991304347826087
```

```
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00         7
           2       1.00      1.00      1.00         6
           3       1.00      1.00      1.00         4
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         4
           7       1.00      1.00      1.00         5
           8       1.00      1.00      1.00         1
           9       1.00      1.00      1.00        10
          10       1.00      1.00      1.00         4
          11       1.00      1.00      1.00         4
          12       0.90      1.00      0.95         9
          13       1.00      1.00      1.00         9
          14       1.00      1.00      1.00         3
          15       1.00      1.00      1.00         2
          16       1.00      1.00      1.00         8
          17       1.00      1.00      1.00         6
          18       1.00      1.00      1.00        10
          19       1.00      0.90      0.95        10

    accuracy                           0.99       115
   macro avg       0.99      0.99      0.99       115
weighted avg       0.99      0.99      0.99       115
```

13 Result for LDA classification

However, we could not get the eigenvalue and the eigenvector corresponding to them by using the package LDA. In this case, we set up our LDA using the algorithm above which will help us to find the Fisher-face. The detail is shown in the final part.

Section V: Discussion and Conclusion

Overall, in this project, we used different machine learning techniques/algorithms to implement the classification task for this Sheffield Face Database.

- Load the images in the file and make our dataset. Using Stratified Sampling for Logistic Regression and random split for other techniques. The training set is 80% and the test set is 20% of the total samples.
- We compared the result using SVM with Gaussian kernel and Polynomial kernel, and get the best accuracy of 98%.
- Use PCA for feature extraction and save the computation in training and make Eigen-faces. Then reconstruct the image using the Eigen-faces.
- Set up manual binary Logistic Regression and apply OvR algorithm to implement the multi-class classification, got final accuracy of about 69%.
- Use LDA for multi-class classification and get the final accuracy of about 99%. And set up manual LDA to find the eigenvectors which will help with finding Fisher-faces.

References:

[1] Danijel. S, Martina. U, Ales. L and Horst. B. *Why to combine reconstructive and discriminative information for incremental subspace learning*. In *Computer Vision Winter Workshop 2006.*

[2] Grace. Chen (Nov 2018), *What is the kernel trick? Why is it important?* Available at: https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d

[3] Adrienne Watt, DATABASE DESIGN – 2ND EDITION Chapter 5 Data Modelling, Available at: https://opentextbc.ca/dbdesign01/chapter/chapter-5-data-modelling/