

CS492 Coursework1-part1

20170528 HyeAnn Lee
School of Electrical Engineering
KAIST
Daejeon, Republic of Korea
hyeann4688@kaist.ac.kr

20190647 Donghyeon Choi
School of Electrical Engineering
KAIST
Daejeon, Republic of Korea
hyun000@kaist.ac.kr

Abstract—Coursework 1 part 1 of 2022 KAIST CS492E ‘Machine Learning for Computer Vision’ course.

Keywords—Manifold Learning, Online Learning, Discriminant Analysis, Ensemble Learning, Randomized Forests

I. INTRODUCTION

The experiment was done with the provided face data (face.mat), which stores raster-scanned face images (46×56 pixels) in columns. We performed PCA by calculating eigenfaces of the given data, and compared PCA with other methods such as incremental PCA. Face recognition performance is further improved by using LDA after PCA, or using random forest. Implementation for the coursework is done with Python, and entire code of us is shown in GitHub(url in Appendix 1).

II. COMPUTATIONALLY EFFICIENT EIGENFACES

A. Implementation method

The first step to classify images with PCA is to get the covariance matrix $S = \frac{1}{N}AA^T$, where A is the deviation of the given image vectors x_i from the average image vector \bar{x} . Here, A has dimension of $D \times N$ since there is N image vectors each with $D \times 1$ dimension. Thus the covariance matrix S has dimension of $D \times D$, where $N \ll D$ for the given image set ‘face.mat.’

The next step is to get eigenspace. There are two ways to get the eigenspace. First is the direct way using $S = \frac{1}{N}AA^T$ with dimension of $D \times D$, and the other is the indirect way using $\frac{1}{N}A^TA$ which has much lower dimension of $N \times N$. With the eig() function in the numpy.linalg library, the eigenvalues and eigenvectors can be easily calculated and they follow the dimension of the input matrix i.e. eigenvalue and eigenvector of S has dimension of $D \times 1$ and $D \times D$, respectively, and in the case of $\frac{1}{N}A^TA$, they have dimension of $N \times 1$ and $N \times N$, respectively.

Lastly, we select M components of eigenvectors with largest eigenvalues and get the reproduction of the image by $\tilde{x} = \bar{x} + WU^T$ where $U = Au$, $W = A^TU$.

The evaluation process of this model gets through four steps. The first step is getting the deviation of the input image with the average image of the training set. Next step is getting representation of the image in the trained eigenspace. Then, get the distance of the representations between the input image and the trained classes. Finally, classify the image by getting the class with the minimum distance. Our implementation achieved 62.5% prediction accuracy for the given face image dataset.

B. Comparing eigenspaces of two methods

Since eigenvalue λ_i and eigenvector v_i for given matrix P always satisfy $Pv_i = \lambda_i v_i$, $Su_i = \lambda_i u_i$ and $A^T Av_i = \lambda_i^* v_i$. Here, multiply A to both side of $A^T Av_i = \lambda_i^* v_i$ so that $A^T Av_i = SAV_i = \lambda_i^* Av_i$. By letting $u_i = \frac{Av_i}{\|Av_i\|}$, $SAv_i = \lambda_i^* Av_i$ becomes $\|Av_i\|Su_i = \|Av_i\|\lambda_i^* u_i$. Then, by dividing both side with $\|Av_i\|$, $Su_i = \lambda_i^* u_i = \lambda_i u_i$. Thus, AA^T and $A^T A$ have the same eigenvalues and their eigenvectors are related with $u_i = Av_i$. We proved this by showing that there are no zero eigenvalues in both eigenspaces.

C. Pros and Cons of each method

The pros of the low-dimension method is the computation efficiency. Getting eigenspace of a matrix becomes much harder as the dimension of the target matrix becomes larger. As $N \ll D$ in typical datasets, the low-dimension method has a far smaller matrix, therefore, theoretically, it will consume less time than the default PCA. We measured the time consumption of both methods and there was a significant advantage of the low-dimension method. The default method takes about 50 times longer than the low-dimension method in our experiment.

The cons of the low-dimension method is the less number of principal components that can be selected. The default PCA method makes D eigenvectors, which is even more than N eigenvectors that low-dimension PCA makes. Thus, low-dimension can replace the default method with selecting up to N principal components.

D. Effect of the number of PCA bases

The objective of the PCA method is getting high speed inference with less loss of accuracy by reducing the bases (i.e., only use the ‘Principal’ bases.) In other words, selecting too few bases causes significant loss of accuracy while choosing too many bases induces no gain on the time efficiency. We implemented an experiment to show the change of the reproduced image by various base numbers (Appendix 2). At 200 bases, the reproduced image is almost similar to the original image.

III. INCREMENTAL PCA

A. Incremental PCA

For incremental PCA, we divided the 416 training data pairs into 4 subsets, each containing 2 images per person. We used *sklearn* to implement. *incrementalPCA()* is used for incremental PCA, *PCA()* is used for batch PCA, and *KNeighborsClassifier()* with parameter *n_neighbors=1* is used for NN classifier.

B. Comparison between incremental PCA, batch PCA and PCA trained only by the first subset

Comparison between i) incremental PCA, ii) batch PCA and iii) PCA trained only by the first subset is shown in Table 1. Training time of incremental PCA includes a dataset dividing step. Their confusion matrices are also shown. `accuracy_score()` is used for accuracy and `ConfusionMatrixDisplay()` is used for confusion matrix. Violet, green and yellow correspond to 0, 1, and 2, respectively. Training time of i) is long due to the data splitting time, and that of iii) is very short due to the quartered training data. Accuracy of incremental PCA and batch PCA are similar, while iii) shows low prediction accuracy. This is also explainable through the size of training data. Accuracy values are in line with the confusion matrices. Confusion matrices of i) and ii) show clear diagonal tendency, while iii) does not.

Changing a parameter `whiten` of incremental PCA from `False` to `True` gives accuracy of 0.606, which is a little bit better than original incremental PCA. Whitening adjusts scales of components to be the same. When comparing faces, the face position can make a big difference if the image is moved one pixel to the right so that recognizing as a completely different face. PCA whitening might be helpful in this case.

IV. PCA-LDA FOR FACE RECOGNITION

A. PCA-LDA

We used the `LinearDiscriminantAnalysis()` for LDA models. Various M_{PCA} and M_{LDA} values were tested under the condition of $M_{PCA} \leq N - c = 364$ and $M_{LDA} \leq \min\{M_{PCA}, c - 1\} = \min\{M_{PCA}, 51\}$. 10, 50, 100 and 200 are chosen for M_{PCA} , and 3, 8 and 40 are chosen for M_{LDA} . With

the same M_{PCA} value, PCA-LDA with $M_{LDA} = 3$ performs worse than PCA-only method for all 4 cases. It is because many multiple classes are projected into the same axis, so that it makes distinction harder. With the same M_{LDA} value, all models except $M_{PCA} = 10$ perform similarly well. This tendency corresponds with the performance of vanilla PCA models with various M_{PCA} values. Results at $M_{PCA} = 50$ are shown in Table 2, and results at other M_{PCA} values are shown in the Appendix 3.

Within-class scatter matrix S_W is spanned by $\{x - m_i\}$, where m_i is class means. Rank of the within-class scatter matrix is $N - c = 364$. Between-class scatter matrix S_B is spanned by $\{m_i - m\}$, where m is global mean. Rank of the between-class scatter matrix is $c - 1 = 51$.

We also visualized success and failure cases of PCA-LDA based face recognition. The images are in the Appendix 4. The model misclassified one of face label 22 into label 39. Visualization shows 22 and 39 both wear glasses, which is an understandable misprediction.

With the same M_{PCA} value, training time is not much different, which implies PCA plays an important role by reducing dimensionality. Resource usage report of Google Colab tells that the RAM usage was almost similar (about 1GB) for all (M_{PCA}, M_{LDA}) pairs.

B. PCA-LDA Ensemble

We firstly tested the error of the committee machines and the average error of the individual models. Here, 2 fusion rules

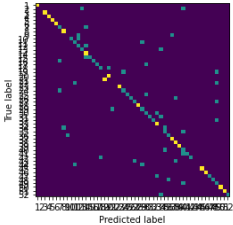
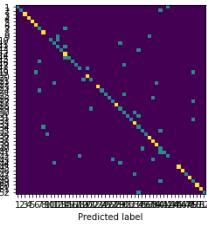
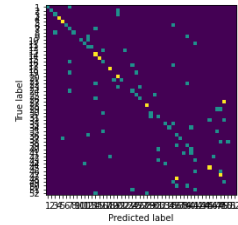
	i) incremental PCA	ii) batch PCA	iii) PCA trained only by the first subset
training time (sec)	0.591	0.193	0.033
reconstruction error ($\times 10^{-11}$)	1.791	104.5	39.42
accuracy	0.596	0.625	0.365
confusion matrix			

Table 1 Comparison between i) incremental PCA, ii) batch PCA and iii) PCA trained only by the first subset

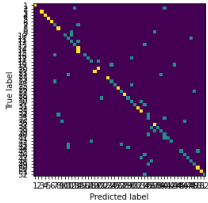
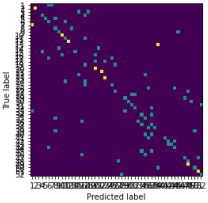
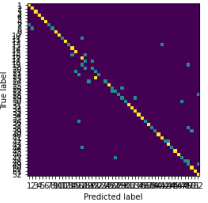
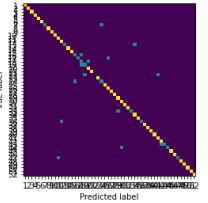
	PCA only	$M_{LDA}=3$	$M_{LDA}=8$	$M_{LDA}=40$
Accuracy	0.567	0.337	0.740	0.865
Time (sec)	0.157	0.163	0.172	0.135
Confusion matrix				

Table 2 Classification performance with $M_{PCA} = 50$ and varying M_{LDA} values

for committee machines were used: majority voting and summation rules. The base model for this experiment was $T=8$, $M_0=40$, $M_1=10$ and $M_{LDA}=40$. The result is shown in Table 3. Accuracy and confusion matrix of summation rule show that it is not appropriate for multi-class classification problems. It averages predictions from T models, so predicted labels tend to be centered, compared to the majority voting case where confusion matrix shows exact diagonal line. Average error of T individual models was output as 182.34, which is larger than the error of the committee machines. This is the same result with the lecture note — E_{com} should be at least less than E_{av} — although $E_{com} = \frac{1}{T}E_{av}$ is ideal. Hereafter, majority voting is used for confusion rule of our rest experiments.

Randomization of training data was experimented with varying numbers of bootstrap $T = 4, 8, 20$ and 50 . Here, the base model uses $M_0=40$, $M_1=10$ and $M_{LDA}=40$. The result is shown in Table 4. The result implies that the performance is good as long as T is not too small as 4 or 8.

Randomization of feature space was experienced with varying (M_0, M_1) pairs - $(40, 10)$, $(50, 0)$, $(90, 10)$ and $(100, 0)$. Here, the base model uses $T=20$ and $M_{LDA}=8$. The result is shown in Table 5. The result implies that the performance is not much affected by the ratio between M_0 and M_1 . Instead, $M_{PCA} = M_0 + M_1$ seems to affect performance much more.

It can be easily inferred that the smaller the randomness parameter, more similar the component models are. Models generated with small randomness parameter are highly correlated to each other, so setting the randomness parameter too small might results no effect of ensemble method.

V. RF CLASSIFIER

A. Implementation of RF classifier

Random Forest classifier uses many decision trees and weak learners, or test functions, for the nodes of trees to classify an input. We implemented the algorithm with

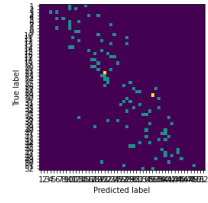
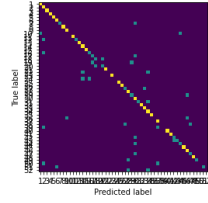
	Summation	Majority voting
Accuracy	0.048	0.683
Error	74.990	114.63
Confusion matrix		

Table 3 Comparison of the error of the committee machines: majority voting and summation rules

`sklearn.ensemble.RandomForestClassifier()` and vanilla coding of weak learner classes with feature extractor. The information gain used to train the model is based on the gini index.

B. Effect of Changing parameters — the number of trees

The number of trees in the forest is the most important parameter that affects the performance of the classifier. The more trees, the more accurate the prediction is. But the accuracy gain efficiency for calculation time drops as the number of used trees increases. We've conducted this with the `sklearn` RandomForest Classification without any parameter change from the default except the $n_estimator$ which is the number of the used decision trees. We have measured the time consumption and test set prediction accuracy (Fig. 1).

C. Effect of Changing parameters — the maximum depth of trees

The maximum depth of trees affects accuracy until the overfitting occurs. The maximum number of leaf nodes of a tree is 2 to the power of the maximum depth ($(Max \# of leaf) = 2^{maxDepth}$). Thus, if this parameter is too small, the number of splitting becomes not enough to

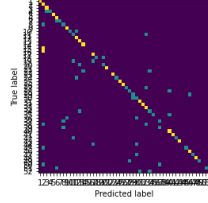
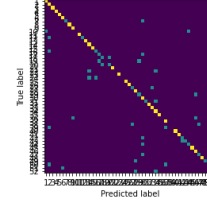
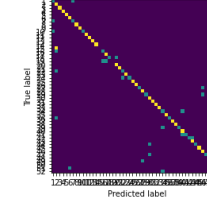
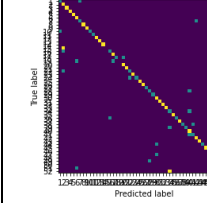
	$T=4$	$T=8$	$T=20$	$T=50$
Accuracy	0.615	0.683	0.779	0.75
Confusion matrix				

Table 4 Randomization of training data

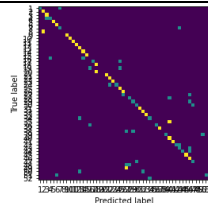
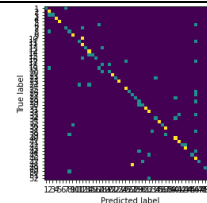
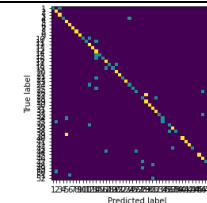
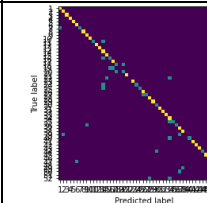
(M_0, M_1)	$(40, 10)$	$(50, 0)$	$(90, 10)$	$(100, 0)$
Accuracy	0.644	0.577	0.692	0.769
Confusion matrix				

Table 5 Randomization of feature space

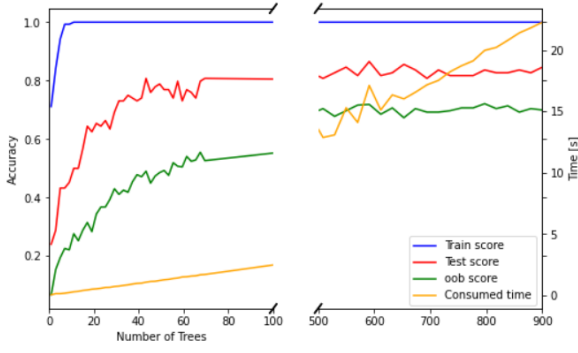


Figure 1 Accuracy score and consumed time for number of trees

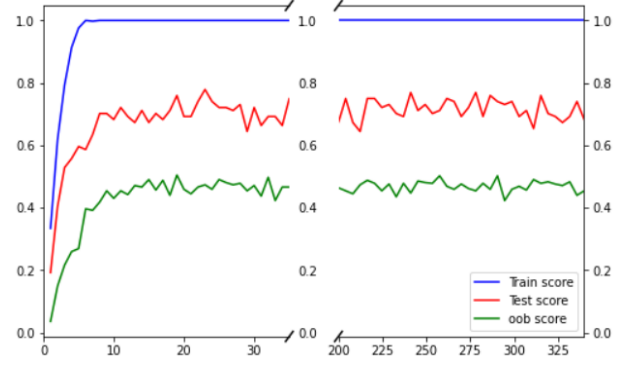


Figure 2 Accuracy score for max depth

classify the given dataset. However, it does not mean that the deeper is the better. If a random forest used too deep trees, the overfitting occurs as unnecessary splits conducted by some nodes so that the algorithm becomes concerned about the “fake” features that actually the objects in that label don’t have commonly. Also, as this algorithm is a greedy algorithm, if an additional split at a node gives no advantages on information gain, the node becomes leaf node, therefore the number of leaf nodes is limited. We tested depth of under 35 and over 200 until 340 where major parameters are number of trees of 100, type of weak learner of ‘linear’. The graph is shown in Fig. 2.

D. Effect of Changing parameters — the degree of randomness

We implemented additional weak-learner with fixed feature selection to reduce the randomness of the algorithm. It gives 0.0192 accuracy score and almost random confusion matrix as shown in Appendix 5. Although we used extremely less randomness, it is quite predictable that less randomness makes classification performance worse.

E. Effect of Changing parameters — the weak-learner type

We implemented 5 weak-learners of Axis Aligned, Linear, Conic, Parabola, TwoPoints. All weak-learners have an array

‘tests’ that contains information of selected features, coefficients that will be used in the test equation, and threshold values. All threshold values are within the boundary of the maximum and minimum of selected features. Axis aligned function selects a single random feature and splits the input by two groups, larger ones and smaller or equal ones. Linear function selects two random features and three scalars values which are in the same scale with the input points. Then, it compares the linear calculation $Ax + By + C$ with zero. We also implemented two nonlinear functions which are in Conic shape and in Parabola shape. The tests are conducted through the equations $Ax^2 + Bxy + Cy^2 + Dx + Ey + F > 0$ and $x^2 < Ay$, respectively. The TwoPoints weak-learner calculates the distance of two features and compares it with the threshold which is the difference between two feature thresholds i.e., $|v_i - v_j| < |\tau_i - \tau_j|$. Specific comparison is shown in Table 6. By comparing with the results of PCA-LDA based face recognition, random forest model can give better performance depending on the type of weak learner. Major model parameters are set as *max_depth* = 100 and *the_number_of_trees* = 1000. More details are in our code.

	Axis Aligned	Linear	Conic	Parabola	TwoPoints
Out-of-bag score	0.695	0.596	0.481	0.464	0.683
Accuracy	0.837	0.788	0.663	0.76	0.875
Confusion matrix					

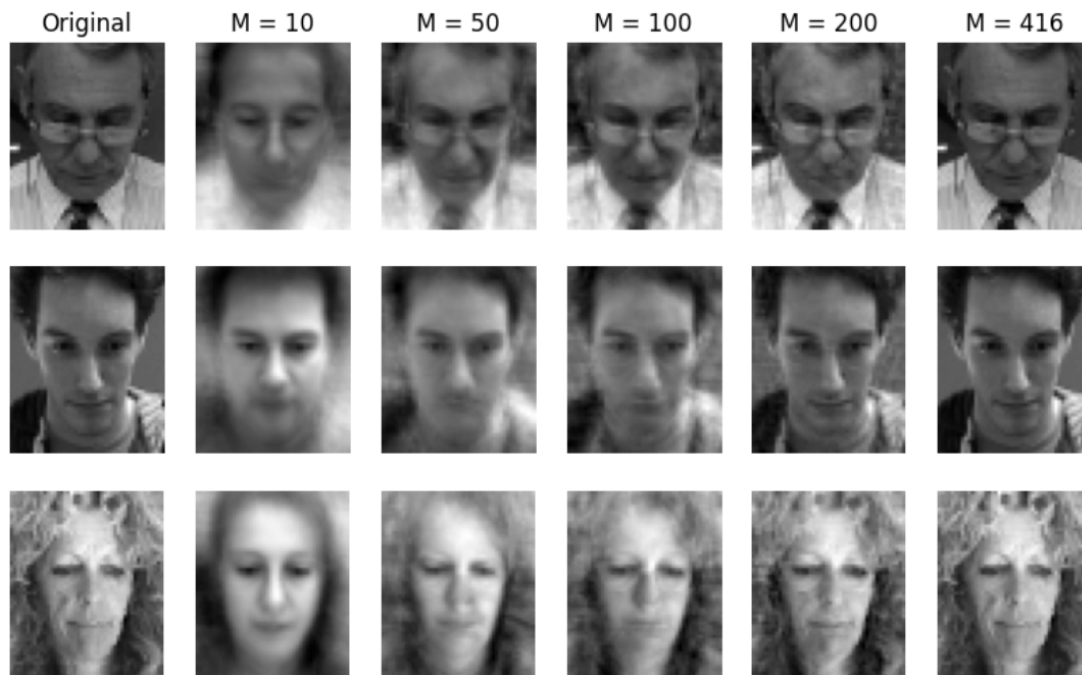
Table 6 Comparison between multiple weak-learners

Appendix

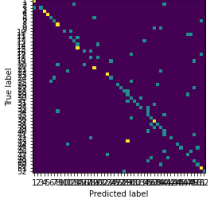
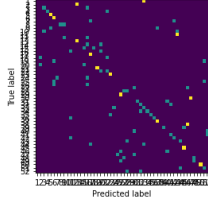
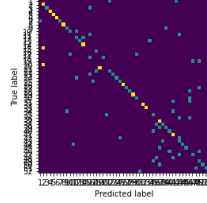
Appendix 1 Entire implementation

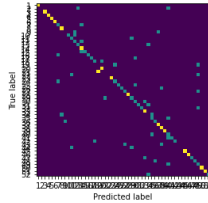
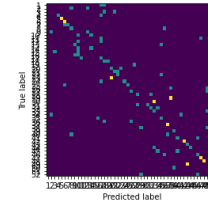
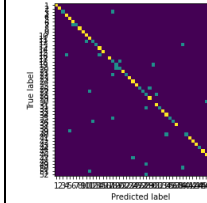
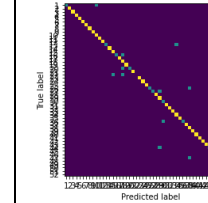
<https://github.com/DongHyunnn/CS492E-ML4CV-Courseworks>

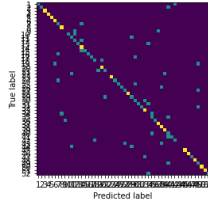
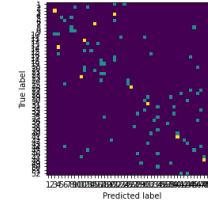
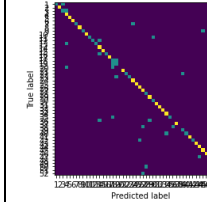
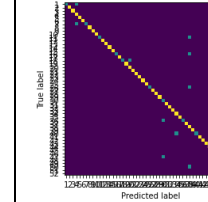
Appendix 2 Change of the reproduced image by various number of bases



Appendix 3 Classification performance with varying M_{PCA} and M_{LDA} values.




$M_{PCA}=10$	PCA only	$M_{LDA}=3$	$M_{LDA}=8$	$M_{LDA}=40$
Accuracy	0.423	0.317	0.471	unavailable since M_{LDA} should be smaller than M_{PCA} .
Time (sec)	0.145	0.145	0.141	
Confusion matrix				

$M_{PCA}=100$	PCA only	$M_{LDA}=3$	$M_{LDA}=8$	$M_{LDA}=40$
Accuracy	0.596	0.346	0.721	0.884
Time (sec)	0.237	0.255	0.291	0.298
Confusion matrix				

$M_{PCA}=200$	PCA only	$M_{LDA}=3$	$M_{LDA}=8$	$M_{LDA}=40$
Accuracy	0.615	0.240	0.731	0.894
Time (sec)	0.419	0.461	0.487	0.436
Confusion matrix				

Appendix 4 Example success and failure cases of PCA-LDA

One success case: the model predicted the face with label 39 as label 39 correctly.

test image	mean face of label 39	one of training image of label 39
		

One failure case: the model predicted the face with label 22 as label 39.

test image	mean face of label 22	mean face of label 39	one of training image of label 22
			

Appendix 5 Confusion matrix of weirdLinear model

