Face recognition using PCA and clustering

Diego Bened, Yuanrong Han, Mohamed Mohamed, Mohamed El Sayed

***Abstract*—Facial recognition technology has gained immense popularity due to its diverse applications, from security systems to personalized user experiences. This project introduces an assessment of clustering methodologies in order to introduce a lightweight and fast algorithm specifically for this important and demanding domain. Given the complexity of dealing with image datasets, particularly regarding memory and time complexity, our study focuses on supervised clustering techniques to classify face images. Specifically, KNN, KMeans with regression, Nearest Centroids, and Agglomeration Clustering algorithms are evaluated for their efficacy in this topic. To moderate computational challenges, Principal Component Analysis (PCA) is employed to reduce the dimensionality of the dataset, thereby enhancing the algorithm's efficiency. The number of Principal Components is chosen thoughtfully to help balance retaining data variance and computational feasibility. Comprising 400 face image data points of 40 individuals, each with a corresponding label. The various clustering algorithms are trained and evaluated using test data accuracy, precision, recall, and F1 score as performance metrics. Results indicate that with lower principal component counts, KNN exhibits favorable performance, while KMeans outperforms with higher principal component counts. The agglomeration clustering demonstrates promising results, showcasing its potential for lightweight face recognition applications. Overall, the findings show the feasibility of developing simple face recognition algorithms using supervised clustering techniques, offering insights into model selection, dimensionality reduction, and performance optimization for efficient real-world deployment.**

***Index Terms*—ECE 1513, Introduction to Machine Learning**

I. Introduction

Face recognition is used everywhere, from identifying individuals in public spaces for security to biometric access control in airports to facilitating the work of border agents.

Various techniques, such as Gaussian Mixture Models, Neural Networks, and Support Vector Machines, are used for face recognition. Face recognition can serve different purposes, such as predicting whether an individual is known (useful for building access, for example) or assigning a face image to a known individual. Our project focuses on identifying a person's selfie in a smartphone album. Modern phones can already recognize faces and group photos accordingly. We aim to develop an algorithm for this task. The goal is to create a lightweight algorithm that performs well for selfie identification, making sure that both training and prediction are not computationally expensive. While face recognition is a valuable feature for smartphones, it should not consume excessive energy. Therefore, using techniques like Neural Networks, which are computationally expensive, is not ideal. We will restrict our project, considering that all photos in a smartphone's album are selfies of known individuals.

Our project focuses on supervised algorithms to assign each selfie to a particular person, using lightweight algorithms and assessing their performance. We will specifically study non-computationally expensive clustering techniques for data classification, including KNN, K Means with regression, Nearest centroids, and Agglomeration Clustering.

Working with images means dealing with a large dataset due to the numerous features each image has. For instance, for an image size of 40x40, we would have 1600 features for each data point, which is quite significant. Time complexity will increase significantly with many features as the model will try to find patterns between the feature values and the labels. To solve this issue, we will use PCA (Principal Component Analysis) to reduce the dataset's dimensions.

II. System Design

A. Dataset

The dataset used for this project is *fetch_olivetti_faces*. It is part of the sklearn datasets based on *The Database of Face* from AT&T Laboratories Cambridge. It contains 400 face images of 40 distinct individuals. Each image is 64x64 pixels, which gives a dimensionality of 4096. Each pixel has a real value between 0 and 1, corresponding to its shade of grey. Each image has a corresponding label (from 0 to 39) that identifies the face of the person on the photo.

Fig. 1: First 50 Faces

B. Design of the experiment

As we mentioned earlier, there are different ways to do face recognition. Neural networks are one option, but we wanted something faster and simpler. That's why we went with clustering techniques. They're lightweight compared to neural networks, which need more computation for training and predictions. Instead of choosing one clustering technique and assessing the results on the dataset provided, we wanted to investigate which clustering technique would be more relevant for face recognition. We thus decided to focus on clustering algorithms that could be relevant to use for face recognition, that is to say, KNN, K Means, Nearest Centroids, and Agglomeration Clustering. For some algorithms, such as K Means, we also needed to use regression to predict the target of the image, as they may be unsupervised algorithms.

To train the model for each clustering algorithm, we first needed to preprocess the data. To apply machine learning, we first needed to flatten the images of size 64*64 to vector data containing 4096 dimensions. As you may guess, we will not use all these dimensions to make our predictions as it would be computationally expensive. To reduce the dimension of the data, we decided to use PCA, which stands for Principal Component Analysis. Using PCA, we are trying to reduce the number of dimensions needed while still being able to describe the data well (conserve as much variance as possible).

C. PCA/SVD

PCA is an efficient algorithm that can be used to reduce the number of dimensions while retaining as much information as possible. In our case, we have a dataset of size 400x4096, with 4096 Principal Components.

We started our analysis by looking at how many PCs would be necessary to retain as much variance as possible. We figured out that to retain 95% of the variance, we needed to have 250 Principal Components. However, this is still an important number of dimensions. We also visualized the variance captured by the first 30 PCs, as shown in the scree plot in Figure 2. As you can notice, after the first 20 PCs, the variance captured is quite small. Therefore, we decided to analyze the clustering techniques using the first 5, 10, and 30 Principal Components, as well as using all 250 PCs that capture 95% of the variance.

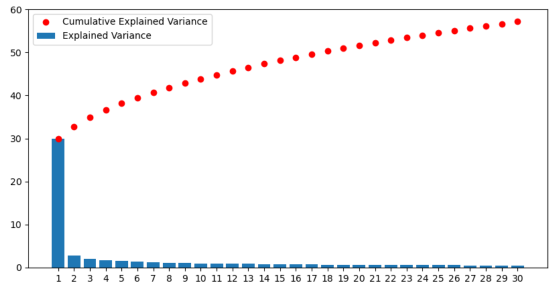

Fig. 2: Scree Plot

In our case, we choose to compute PCA with Singular Value Decomposition (SVD), which is another widely used matrix decomposition method that is closely related to Principal Component Analysis(PCA). SVD on matrix A can be expressed as

$$A = U\Sigma V^T \tag{1}$$

with $A \in \mathbb{R}^{m\times n}$, $U \in \mathbb{R}^{m\times m}$, $\Sigma \in \mathbb{R}^{m\times n}$, $V \in \mathbb{R}^{n\times n}$. U and V are orthonormal matrices. [1]

SVD is closely related to Principal Component Analysis (PCA), and we could use SVD to perform PCA. Looking at A^TA

$$\begin{aligned} A^TA &= V\Sigma U^T U\Sigma V^T \\ &= V\Sigma^2 V^T \end{aligned} \tag{2}$$

We know that the covariance matrix of A is $C = \frac{1}{m-1}V\Sigma^2V^T$. From PCA, since the covariance matrix is symmetric, it can be decomposed to $C = V\Lambda V^T$. Then $\frac{1}{m-1}\Sigma^2 = \Lambda$, and the orthogonal matrix V is the eigenvectors of the covariance matrix. Using SVD, the column of matrix V is the eigenvector of the image. [2]

D. KNN

We are employing a range of classifiers from the K-Nearest Neighbour to Decision Trees, KNN is straightforward yet powerful, it doesn't make assumptions about the shape of data and can handle complex patterns. Rather than the conventional use of KNN in direct face recognition, we are using it to cluster faces in the dataset based on their similarities. Grouping similar faces and clustering them would help in understanding the pattern of the dataset by considering similar data points based on their features. By examining which faces ended up in the same clusters, we get insights into the natural groupings present within the dataset. While this doesn't directly assign identities to faces, it lays the groundwork for understanding facial similarities, which is crucial for subsequent recognition tasks. We will iterate values from 1 to 15 to find the best number of neighbors and ensure a consistent starting point for clustering. The KNN would help find the best approach for clustering similar data points (faces).

Algorithm 1: K Nearest Neighbor Classifier

```
input: X: Training datapoints
         ${(x_{11}, \dots, x_{1n}), \dots, (x_{d1}, \dots, x_{dn})}$
       k: number of neighbors
       dist(x,y): Euclidean Distance between x and y
       p: a testing data point $(p_1, \dots, p_n)$
foreach $x \in X$ do
    Compute $dist(x, p)$;
end
Let $N \in X$ be the set of points of the first k shortest
  distance;
return The majority label of $N$
```

E. KMeans

The K-means clustering algorithm's main purpose is to cluster similar data in the K number of clusters. In the fetch_olivetti_faces dataset, we will use k-means to group data of each person into a cluster; in other words, similar data points can be clustered together in one cluster, and this cluster is for the target face we need to recognize; it also could be an important step in higher precision face recognition. Since K-means is not a supervised classification method, we must map the cluster label from the K-means training process to our target label. We tried three types of regression: logistics regression(LR), decision tree(DT), and random forest(RF).

```
Algorithm 2: KMeans Classifier
  input: $X$: Training data, $X = \{(x_1, \dots, x_n), \dots\}$
          $\vec{y}$: Training data labels $y_i \in 0, \dots, 39$
          p: a testing data point $(p_1, \dots, p_n)$
          regression: A Regression method on data x
  with label y
          clustering number: N
  $Kmeans(N).fit(X)$;
  $regression.fit(Kmeans.labels, \vec{y})$;
  return $regression.predict(Kmeans.predict(p))$
```

1) Logistics Regression(LR): We first used logistic regression as the post-processing method. We have tried three types of input data for logistics regression with K Means. We called them 3 different modes: full, partial, and hybrid. In partial mode, we used only the label from K-Means as the input of logistics regression, but the results were not ideal. In full mode, we used the labels from K-Means as a feature for each data point and used this as the input for logistics regression, and we got promising results shown in Table III. Lastly, in hybrid mode, we took the first 30 percent of the training data and the label classified from K-Means as the input for logistics regression.

2) Decision Tree(DT): Another method we have tried is the decision tree. The decision tree is a type of hierarchical model. The data will be represented in a tree-like structure based on the feature of the dataset; the tree will split the data into different nodes. In our case, we use the compressed dataset and the label from K-means as our input to the decision tree; figure 3 shows the first 2 depths of the tree. [3]

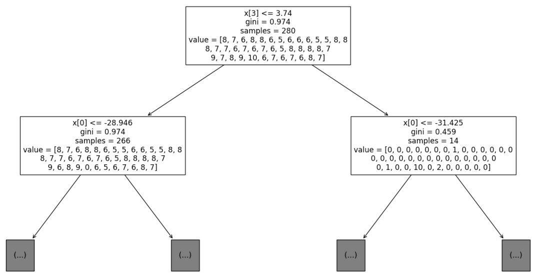

Fig. 3: Decision Tree, First 2 Depth

3) Random Forest(RF): Random forest can be considered a more robust decision tree. It uses many decision trees and averaging to get better accuracy on the prediction. A different number of trees used in the Random Forest model is set to default 100. [4]

As we go from 5 to 50 principal components, we should notice an increase in accuracy; hence, the approach is on the right path.

F. Nearest Centroids Classifier

The nearest Centroid Classifier is a naive supervised classification method. It requires the least calculation among the other classification methods. Thus, the time to train and predict takes a short time. Its idea is to calculate the average of each feature in one class, which will be the centroid of that class. By comparing the distances to all centroids, the closest centroid to the data point will be the predicted class. We follow the same path as on other algorithms. First, we feed the faces dataset to the NCC; we train the model to let it learn and organize the corresponding data points for each target label and cluster them. The prediction here means sorting the data points to each target based on the distance between each data point and centroid to calculate the closest centroid. In our case, we use Euclidean distance as our metric to calculate distances. We use the same accuracy determination method as the previous two clustering methods. The results section will compare all the clustering algorithms as we change principal components from 5 to 30.

```
Algorithm 3: Nearest Centroid Classifier
  input: $X_i$: Class i datapoints
           $\{(x_{i1}, \dots, x_{in}), \dots, (x_{i1}, \dots, x_{in})\}$
           p: a testing data point $(p_1, \dots, p_n)$
           dist(x,y): Euclidean Distance between x and y
  foreach $i \in class$ do
    | Calculate the centroid of all points $x \in X_i$;
  end
  Let C be the Centroids;
  foreach $c \in C$ do
    | Calculate the distance: $dist(c, p)$
  end
  return The class of the centroid with the shortest
    distance to the test point.
```

G. Agglomeration Clustering

Clustering mainly consists of two types: hierarchy and non-hierarchy. As a hierarchical, unsupervised learning technique, agglomerative clustering groups similar data points into a tree or a dendrogram. The algorithm continues to iterate until it finds a hierarchy of similar features in the dataset, and the closer hierarchies agglomerate together, forming one.

$$d = \sqrt{\sum_{i=1}^{n} |u_i - v_i|}$$

where, d is the distance or dissimilarity measure between two images u and v in the dataset, and n is the total number of variables. The distance between hierarchies is calculated mainly by single linkage, complete linkage, and average linkage methods. The single linkage clusters data using the minimum distances between hierarchy, complete linkage uses the maximum distances between them, and the average linkage classifies data points based on the average distance.

III. Results

A. Evaluation metrics

We trained the models with compressed images using 4 principal component numbers: 5, 10, 30, and 250. We also trained our model with the non-compressed images. To compare the effectiveness of each model, we used test data accuracy, precision, recall, and F1 score as our metrics.

In our case, we have multiclass classification. Our precision and recall will ultimately take an average strategy, so we use macro-averaging. In a binary case, precision is the percentage of True Positive(TP) cases among True Positive and False Positive(FP) cases; recall is the percentage of True Positive cases among True Positive and False Negative(FN) cases. The F1 score considers both precision and recall, which gives us a better evaluation of the model.

For class 0:

$$Precision_0 = \frac{TP_0}{TP_0 + FP_0} \tag{3}$$

$$Recall_0 = \frac{TP_0}{TP_0 + FN_0} \tag{4}$$

$$\mathbb{F}_{1_0} = 2 \cdot \frac{precision_0 \cdot recall_0}{precision_0 + recall_0} \tag{5}$$

In our case, the precision, recall, and F1 score will take an average, and N is the number of classes

$$Precision = \frac{1}{N}\sum_{i=0}^{N} precision_i \tag{6}$$

$$Recall = \frac{1}{N}\sum_{i=0}^{N} recall_i \tag{7}$$

$$\mathbb{F}_1 = \frac{1}{N}\sum_{i=0}^{N} \mathbb{F}_{1i} \tag{8}$$

B. K-means regression

Since the K-means classifier uses 5 regression methods, we will discuss their differences using the F1 score and training/predicting time as metrics.

Clearly, with lower principal components, the decision tree and random forest outperform the linear regression methods. However, with more principal components used, the performance of the decision tree and the random forest did not increase by a huge margin compared to the full-mode linear regression. Full-mode linear regression performs the best with 250 principal components with an f1 score of 0.958. The partial linear regression couldn't predict any with any principal component count since the only feature of the regression input is the label from k-means.

Decision trees and random forests outperform the training/predicting time with fewer principal components. With 30 principal components, they still outperform by a little. However, with 30 principal components, the time difference between linear regression and the decision tree is only 0.02 seconds, and the F1 score is 72% higher. Using 250 principal components doubles the time to train and predict the result; comparing 30 principal components at 0.05785 seconds to 0.13963 seconds, it is over 200% slower. In the comparisons in the later part, kmeans will be referred to as kmeans with full-mode linear regression. Details can be found in Figure 4 and Figure 5

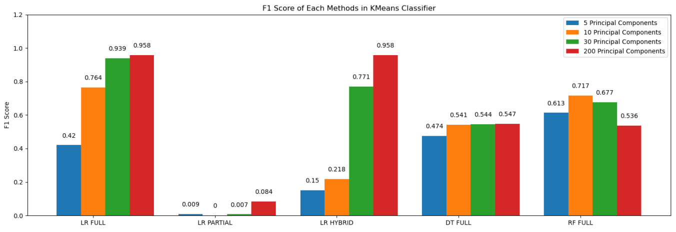

Fig. 4: Kmeans Methods F1 Scores

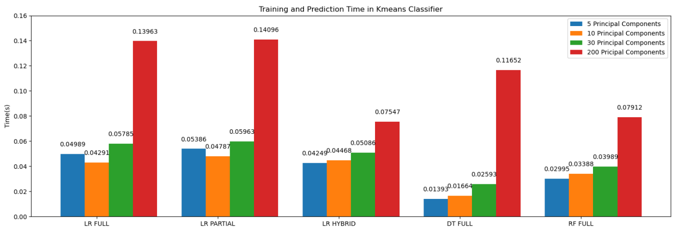

Fig. 5: Kmeans Methods Time

C. Evaluation Metrics Comparison

	KNN	Nearest Centroid	Kmeans	Agglomeration Clustering
Precision	0.774	0.497	0.428	0.336
Recall	0.747	0.472	0.427	0.352
Accuracy	0.717	0.475	0.442	0.367
F1 Score	0.716	0.460	0.391	0.316

TABLE I: 5 Principal Components

	KNN	Nearest Centroid	Kmeans	Agglomeration Clustering
Precision	0.850	0.670	0.832	0.594
Recall	0.829	0.693	0.842	0.609
Accuracy	0.825	0.708	0.833	0.600
F1 Score	0.811	0.663	0.814	0.564

TABLE II: 10 Principal Components

	KNN	Nearest Centroid	Kmeans	Agglomeration Clustering
Precision	0.943	0.854	0.948	0.793
Recall	0.932	0.852	0.944	0.818
Accuracy	0.925	0.842	0.942	0.783
F1 Score	0.926	0.841	0.939	0.779

TABLE III: 30 Principal Components

We have noticed the best model with lower principal component counts is KNN. KNN has a decent average in most cases, with an average F1 score of 0.817. When 10 principal components are used, precision, recall, accuracy, and F1 scores are close between the KNN and KMeans classifier. With 30 Principal Components, Kmeans outperforms the KNN and

	KNN	Nearest Centroid	Kmeans	Agglomeration Clustering
Precision	0.959	0.891	0.959	0.857
Recall	0.955	0.894	0.969	0.856
Accuracy	0.950	0.883	0.958	0.842
F1 Score	0.951	0.882	0.958	0.834

TABLE IV: 250 Principal Components

Nearest Centroid Classifier by 3.5% and 13.9%, respectively. When we increase the principal component drastically to 250 to obtain over 95% of variance, we notice that the test f1 score does not increase by a lot; the test accuracies are very close to each other between using 30 principal components and 250 principal components. We used only 30 principal components in the later part to compare kmeans with other clustering methods.

Details are in Table I, Table II, Table III and in Table IV

D. Time Metrics Comparison

	KNN	Nearest Centroid	Kmeans	Agglomeration Clustering
Total Time(s)	0.031	0.008	2.218	2.154
Predicting Time(s)	0.030	0.005	2.007	1.774
F1	0.94	0.882	0.955	0.815

TABLE V: Time of Training with the full dataset

	KNN	Nearest Centroid	Kmeans	Agglomeration Clustering
Total Time(s)	0.01	0.003	0.062	0.055
Predicting Time(s)	0.010	0.001	0.0440	0.038
F1	0.926	0.841	0.939	0.779

TABLE VI: Time of Training with compressed dataset, 30 PC

Since our ultimate goal is to have lightweight models, we have compared the total time to train the model and give a prediction, as well as the duration of giving a prediction. It is obvious that in models trained with compressed data, the performances are very close to those of the model trained with uncompressed data. The F1 score for each model, for example, KNN trained on the full dataset, is only 1.5% more effective than the one trained on the compressed dataset, Nearest Centroid is 4.8% more, and KMeans is only 1.7% more effective.

Principal component numbers have a huge impact on the time needed to train and predict, especially for KMeans. For KMeans, the total time to train and predict with the full dataset takes 2.218 seconds, but the total time to train and predict with the compressed dataset only takes 0.062 seconds, which is 35 times faster.

Details are in Table V, Table VI

E. Interpretation

We obtained promising results using KNN and K-Means for face recognition, achieving an F1 score of 0.926 and 0.939, respectively, with only 30 Principal Components and a training time of 0.031 and 2.218 seconds. Using a more complex model does not necessarily mean getting better results, as KNN is a simplistic model compared to Agglomerative Clustering but provides better results. Indeed, KNN and Nearest centroid are the more simplistic models and, therefore, give the least training time. K-means and Agglomerative clustering are more complex models as they are unsupervised and need regression to predict a person's face. Finally, KNN may be the most suitable model to use as it is a simplistic model that provides great results. Note that in our experiment, we only had 40 faces and 400 images. In a real-world application, we could have thousands of images and a larger number of people, thus needing more time to train the model. In that sense, KNN model may be more relevant to use.

The good results may be attributed to the dataset containing similar selfie photos. This can be seen in Figure 1, where each image contains the same proportion of a person's face. Using selfies with varying face proportions could potentially yield different results. Therefore, in future experiments, it would be relevant to use selfies with diverse face proportions.

IV. Concluding Remarks

Utilizing clustering methods, particularly K-means, we successfully developed a lightweight algorithm with rapid training times, which is crucial for identifying a person's selfie in a smartphone album. The importance of this work stems from the practical need for a fast and efficient face recognition algorithm in scenarios where models must be frequently retrained. Traditional approaches like Neural Networks may offer accuracy but have high computational costs and prolonged training times, making them less suitable for rapid retraining needs. The choice to focus on clustering techniques, especially K-means, is strategic as it aligns to create a quick-to-train algorithm while still maintaining reasonable performance metrics such as accuracy, precision, recall, and F1 score. The findings from evaluating various clustering algorithms, including KNN, K-means with regression, Nearest Centroids, and Agglomeration Clustering, provide insights into model selection, dimensionality reduction, and performance optimization, offering a practical framework for efficient real-world deployment. Accordingly, we filled the gap between theoretical advancements and practical applications by providing a solution that is not only effective in solving a real-world problem but also feasible and cost-effective for deployment in scenarios where rapid training and retraining are paramount.

References

[1] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[2] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 01 1991.

[3] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, pp. 81–106, 1986.

[4] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.