Dimensionality Reduction, Classification and Clustering on Face Data

Stephen Jonany[1]

***Abstract*— We perform a variety of analyses on Yale face dataset [1]. We use SVD to analyze the low-rank structure of the faces under a variety of lighting conditions. We attempt to classify pictures into person identities as well as genders. Finally, we conclude by clustering the photos to see if separation based on lighting conditions can be achieved in an unsupervised manner.**

I. Introduction and Overview

The Yale face dataset [1] is a collection of a variety of people, subjected to different lighting conditions and poses. They provide an interesting dataset to compare different classification and clustering algorithms. In the following sections we will tackle dimensionality reduction, identity classification, gender classification, and clustering using a variety of data science methods.

II. Theoretical Background

A. Singular Value Decomposition

Singular value decomposition (SVD) is a dimensionality reduction technique that aims to decompose a matrix M into the product of three matrices $U\Sigma V^*$. The implementation that we are using is the economy SVD, where if M, the original matrix, is of dimension $m \times n$ and we have chosen a low rank $r \leq min(m,n)$, then both U and V are composed of orthonormal column vectors, with U of dimension $m \times r$, and V of dimension $n \times r$. Σ is a square diagonal matrix of dimension $r \times r$, and the diagonal values are often referred to as the singular values.

The singular values can be thought of as a measure of the importance of the corresponding column vectors of U and V in reconstructing the original matrix M. By selecting the top r values of Σ and the corresponding r columns of U and V, one can reconstruct an approximation of the matrix M. One useful metric for deciding on how many top singular values to keep is the total energy, as defined in Equation 1.

$$Energy(r) = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2} \tag{1}$$

The energy metric measures how much of the original variance of the data is retained if we only keep the top U and V components corresponding to the top r singular values. There are also some intuitive interpretations of U and V. The columns of U serve as the basis vectors of the column space of M. Likewise, the columns of V serve as the basis vectors of the row space of M.

[1]Stephen is a Master's student in Applied Mathematics at the University of Washington. Email: sjonany@uw.edu

B. Singular Value Decomposition on Face Dataset

In the face dataset, we preprocess a collection of images into the matrix M. Each 2D grayscale image is converted into a single vector by concatenating each row one after the other. Thus, each letting m be the number of pixels per image, and n be the number of images, each of the n columns of M is a vector of length m.

Before performing the SVD, we took the extra preprocessing step of removing the average face from every column. That is, we compute the average across all the columns, and from each column in M, we deduct this average face. Let the mean-subtracted matrix be called M'. The goal of SVD is to capture the covariance of the original matrix with as few ranks as possible. If one does not subtract off the mean, most of the variance that is captured will be mostly derived from the deviation of the mean from the origin. By removing the average face, we ensure that the covariance we are seeking to capture are the deviations from the average face.

Going back to the SVD background from Section II-A, then the columns of U, which serve as a basis for the column space of M', serve as a collection of face templates, the linear combination of which allows one to reconstruct all the original faces. We will refer to the columns of U as the eigenfaces from now on. Note that each eigenface is $m \times 1$. The diagonal entries in Σ, also referred to as singular values, inform us on how much variance is captured by the corresponding eigenface, where an example correspondence is that the first diagonal entry of Σ corresponds to the first column of U. Thus, if one wants to have a faithful reconstruction of an image under a low rank, one would choose the basis set to be the eigenfaces corresponding to the highest singular values. Finally, the n rows of V, of dimension $1 \times r$, corresponds to the projection of each corresponding n images into the r eigenfaces, scaled by the singular values. They can be thought of as the low-dimensional representation of the original high dimensional images.

C. Low-Dimensional Face Reconstruction

To reconstruct a face using with a low dimensional approximation, we note that $U^T M'$ gives you an $r \times n$ matrix, which correspond to the low-dimensional $(r \times 1)$ projections of the original n images. To see why, note that the matrix multiplication involves dot products between columns of M', with various eigenfaces. Then, $UU^T M'$ will yield an $m \times n$ matrix, corresponding the low-rank reconstructed images, with the same dimension as the original images. To see why, note that the matrix multiplication will involve taking linear combinations of columns of U, the eigenfaces, according to the low-dimension representations of each image.

Note that we are not done yet, because the SVD was computed on M', which already had the original average face removed. To complete our low-rank approximation, we simply add back the average face as computed in II-B to all the columns of UU^TM'.

D. Classification Task and Evaluations

Classification is the problem of identifying a relationship between an input feature vector and an output label from a predetermined discrete set of classes. An example would be as follows: Given a feature vector representing a photo of a person, predict the identity among a predetermined set of 10 people.

A commonly used performance metric is the classification accuracy.

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i). \tag{2}$$

Where N is the number of samples, x_i is a feature vector, y_i is the true label, and f is the classification model, the accuracy measure captures the average number of data points that are classified correctly.

However, the accuracy metric will not be that useful when there is an imbalance among the label distributions. This will be relevant for our binary gender classification experiment. For instance, assuming that among a random population sample, we are tasked with predicting if the person has cancer or not, one way to achieve high accuracy is to consistently predict the most popular label, where the person is free of cancer. One approach to get around this is to use the F-score, which considers the tradeoff between true positive (TP), false positive (FP), and false negative (FN). The exact formulation can be seen in Equation 3.

$$\begin{aligned} Fscore &= \frac{2}{Recall^{-1} + Precision^{-1}} \\ Precision &= \frac{TP}{TP+FP} \\ Recall &= \frac{TP}{TP+FN} \end{aligned} \tag{3}$$

The intuition is that we highly reward the correct identification of the rare label by assigning true positive on the numerators of both precision and recall. Precision will reward the model on its selectivity, whereas recall will reward the model on its comprehensiveness.

E. SVM for Multi-class Classification

The first approach that we will discuss is Support Vector Machine (SVM)[2]. In particular, we will talk about linear SVM. We first discuss the binary classification case, where the true class for each sample, y_i, can only be 1 or -1. Given a feature vector x, the SVM model is comprised of a set of weights w, and makes its predictions simply by computing the dot product $w \cdot x + w_0$, and outputs 1 if the resulting number is positive, and -1 otherwise. Given a training data, the linear SVM will find a set of weights w that minimize the objective function given in Equation 4.

$$argmin_{w,w_0} \frac{1}{2} w \cdot w + C \sum_{i=1}^{N} max(0, 1 - y_i(w \cdot x + w_0)) \tag{4}$$

The $w \cdot w$ term can be thought of as regularization, where we penalize weights that are big in magnitude, which might cause the model to overfit the data. The summation term, which contains y_i' (the true labels) and $w \cdot x + w_0$ (the predicted confidences of the positive class) is the hinge loss summed over the training data. Finally, C is a hyperparameter that captures the trade-off between data fit and regularization penalty.

Extending this to multi-class classification, one formulation we can adopt is one-vs-rest. The idea is that for each of the K classes, we train a binary classifier independently. The binary classifier for class i would be trained with a training set where the labels have been transformed to 1 if the original class matches i, or -1 otherwise. After training all the K binary classifiers, one makes a prediction by simply computing all the $w \cdot x + w_0$ for each binary classifier, which represents the confidence that the sample is of a particular class i, and simply picks the class with the highest confidence value.

F. KNN for Multi-class Classification

K-nearest neighbor (KNN)[3] is a multi-class classification algorithm that works as follows. During training, it simply stores all the training samples, along with the sample labels, without any additional parameters. During testing, given an input feature vector, it will compute the distances (let us assume Euclidean) from that feature vector to all the training samples. It will then select the top K neighbors that have the smallest distances, and do a majority vote on the K neighbors' labels to produce the predicted label.

G. Stratified K-fold Cross Validation

We have described cross validation extensively in our previous reports, but it is worth mentioning an additional tweak. For the gender dataset, we will have a great imbalance in the distribution of labels, whereby there are a lot more males than females. The tweak is that when dividing into various folds, we also take the label distribution into consideration, and ensure that each fold preserves the same class distribution as the that of the whole dataset.

H. Clustering

In a clustering task, one is presented with a list of unlabelled samples, and is asked to cluster the samples into collections that hopefully contain some meaningful structures. This is an example of an unsupervised task, where one is not presented with the labels, and must discover latent structure in the dataset by optimizing against objective functions that only rely on the relative positioning of the samples.

In the analysis section, we will be focusing on just 2 clusters. Since we also happen to know the true binary labels of each sample, a simple evaluation method is to simply treat the 2 clusters as a supervised learning prediction, where the samples of cluster 1 are labelled as class 1, and the samples of cluster 2 are labelled as class 2. Then, we simply compute the classification accuracy as defined in Equation 2. Since we do not know the right mapping between clusters and the supervised labels, we simply try the two possible label-to-cluster assignments, and take the higher accuracy value. Note that the binary labels are not provided during the clustering process, but only used for evaluation.

I. *K-means*

The first clustering approach that we discuss is the k-means algorithm [4]. The algorithm assumes that clusters are represented by centers, and points can only belong to one cluster whose center it is the closest to. The objective function is to find cluster centers that minimize the within-cluster sum of squares, formally described in Equation 5.

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 \tag{5}$$

Assuming two clusters, k is 2, S is a collection of sets S_1, S_2, μ_i is the center of each cluster, and x is the unlabelled data point that has been assigned to a cluster.

The optimization approach is based on expectation maximization, and will be discussed briefly. We start by picking random data points to be our initial centers. Then, up to a finite number of iterations, we will keep repeating between two phases. The first phase is cluster assignment, where we go through every point and assign each one to the cluster with the closest center. The second phase is center adjustment, where for each cluster, we adjust the center by taking the mean of all the points assigned to the cluster.

J. *Gaussian Mixture Models*

Gaussian mixture models (GMM)[5] is the second clustering approach that we will apply. Unlike k-means, which assign points to 1 cluster each, GMM uses soft assignment, where each point is softly assigned to multiple clusters simultaneously. Clusters are modeled as multi-dimensional gaussians, with a center μ_z, a covariance matrix Σ_z, and a weight π_z. The assumption is that the observed points are each independently generated by picking a random cluster z according to the cluster weights π_z, then sampling a point from the cluster's gaussian probability distribution $\mathcal{N}(x|\mu_z, \Sigma_z)$. The objective function is then to find gaussian mixtures that maximize the log likelihood of the data, as formally described in Equation 6.

$$\sum_{i=1}^{n} \log \sum_{z=1}^{k} \pi_z \mathcal{N}(x_i|\mu_z, \Sigma_z) \tag{6}$$

The approach to finding the parameters that maximize the log likelihood is based on expectation maximization, where the high level intuition is such that we take turn solving for the soft assignment of each point to the different clusters, and re-estimating the cluster parameters based on the soft assignments.

III. Algorithm Implementation and Development

All the codes are written in Python. The following Python libraries are used.

- SciPy is used for the clustering, classification and SVD algorithms.
- NumPy is used for efficient 2D matrix manipulations.
- Matplotlib is used to generate all the plots.

We invite interested readers to refer to the code appendices for more detail.

IV. Computational Results

A. *Dataset Overview*

There are two types of datasets: cropped and uncropped. We will be using both only for the SVD analyses, and only cropped for the remaining classification and clustering tasks.

The cropped dataset consists of 38 people with 64 grayscale images whose pixel dimensions are 168 wide and 192 tall. The differing setups are due to a variety of pose and lighting conditions. For the purpose of the gender classification task, we have manually labelled the gender as follows. There are 9 identified females, with person IDs of 5, 15, 16, 24, 27, 28, 32, 34, 37, and the remaining are males. We provide a sample of the dataset in Figure 1.

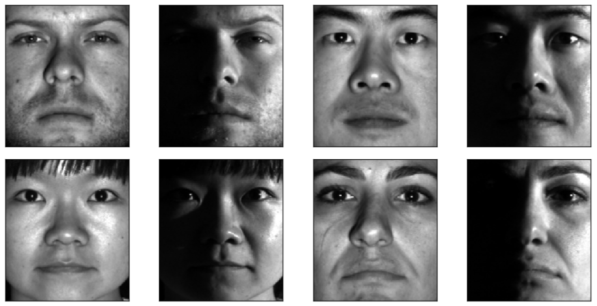

Fig. 1: 2 males and 2 females from the cropped dataset, with 2 different lighting conditions each.

The uncropped dataset consists of 15 people with 11 grayscale images whose pixel dimensions are 320 wide and 243 tall. The differing setups are due to a variety of pose and lighting conditions. Unlike the cropped dataset, most of the image area is consumed by the background, and the face locations within each image are not aligned with each other. We provide a sample of the dataset in Figure 2.

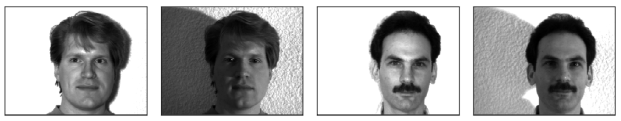

Fig. 2: 2 individuals from the uncropped dataset with 2 different lighting conditions each.

B. *Low-Dimensional Analyses*

As per Section II-B, we perform SVD on both the cropped and uncropped dataset, which have been preprocessed to be in matrix forms.

1) Cropped Dataset: The data matrix we are performing the SVD on is of the dimension 32256×2432, corresponding to the 38×64 images and 168×192 pixels. Before performing SVD, we subtract the mean face as shown in Figure 3 from all the image columns in the matrix. The average face seems to be reasonable, since one can see the key facial features such as nose, eyes, and mouth, and the image is blurry, as expected.

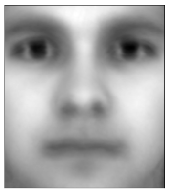

Fig. 3: The average face of the cropped dataset, to be subtracted from all the images before performing SVD.

The energy distribution, as explained in II-A, can be seen in Figure 4. We need 22 modes to cover 90% energy, 62 modes to cover 95% energy and 295 modes to cover 99% energy. It seems rather impressive that one needs so little modes considering there are 2432 images to work with.

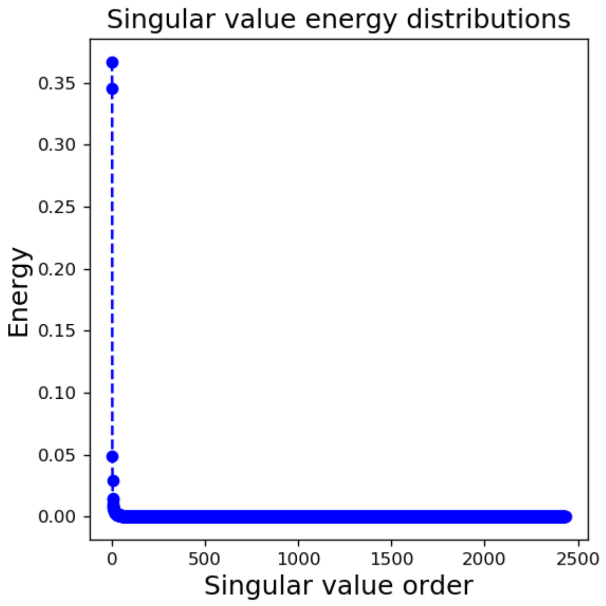

Fig. 4: The energy distribution of the cropped dataset after mean subtraction.

We now inspect the eigenfaces. As seen from Figure 5, the top 4 eigenfaces seem to be blurry, but capture various kinds of lightings. For example, the second eigenface captures lightings that are biased towards either the left or the right (keeping in mind that eigenvectors capture directions, and so the signs do not really matter). We compare this to Figure 6, which shows the 4 eigenfaces from mode 22 onward, which captures 90% energy. As we can see, the pictures are less blurry, and capture the more refined details that distinguish each person, such as the shape of the nose and the boldness of the eyebrows.

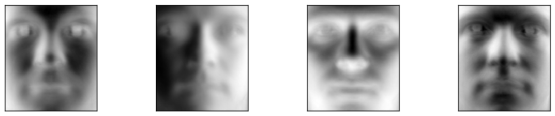

Fig. 5: First 4 eigenfaces for the cropped dataset, ordered from left to right in decreasing singular values.

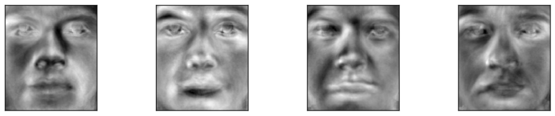

Fig. 6: 4 eigenfaces from mode 22 onward, which capture 90% energy for the cropped dataset. The eigenfaces are ordered from left to right in decreasing singular values.

We finally consider reconstructing an image with varying number of modes, the process of which was described in Section II-C. As seen in Figure 7, there is a clear progression from a blurry image to the final face. At each phase, we add more detailed features, where on the second image we see a clearer eye definition, on the third image we see a clearer brow definition, and finally in the final portrait, we were able to see small wrinkles.

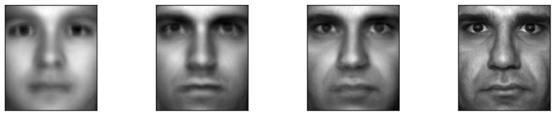

Fig. 7: Reconstructing an image with varying number of modes. From left to right, the number of modes are 1, the number of modes that capture 90% energy, 95% energy, and 99% energy.

2) Uncropped Dataset: We repeat the same analysis to the uncropped dataset, and will not repeat commentaries that are redundant with our discussion on the cropped dataset, but will instead be focusing on the differences.

The data matrix we are performing the SVD on is of the dimension 77760×165, corresponding to the 15×11 images and 243×320 pixels. Before performing SVD, we subtract the mean face as shown in Figure 9 from all the image columns in the matrix. The mean face of the uncropped dataset is much blurrier than that of the cropped dataset. This is reasonable because the face positions of each image are no longer aligned.

Figure 9 shows the energy distribution. We need 26 modes to cover 90% energy, 50 modes to cover 95% energy and 106 modes to cover 99% energy. The SVD rank has increased much more compared to the cropped dataset. Even though there are only 165 images in the uncropped dataset (as

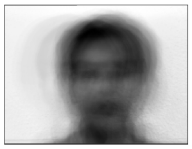

Fig. 8: The average face of the uncropped dataset, to be subtracted from all the images before performing SVD.

compared to the 2432 images in the cropped dataset), we need 26 modes to capture 90% energy. It is impressive that the cropped dataset, while having more than 10 times the number of samples, would require less number of modes (22) to capture the same energy. This also makes sense because the difference in face alignments are equivalent to scrambling the grayscale values of each image, and this transformation is not well captured by SVD.

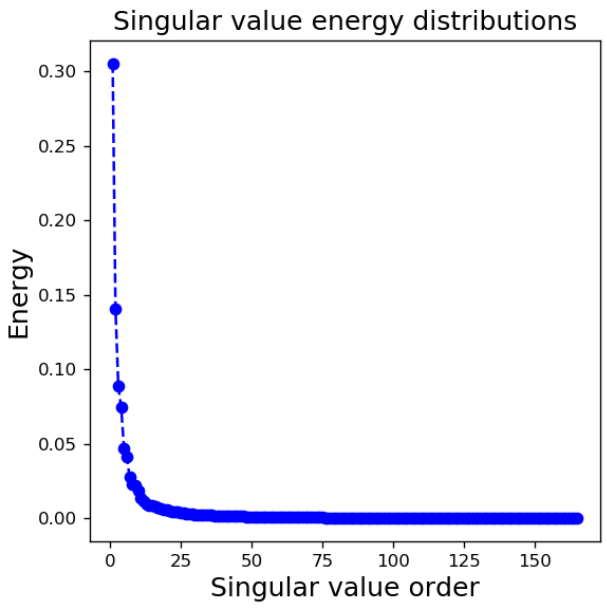

Fig. 9: The energy distribution of the cropped dataset after mean subtraction.

We now inspect the eigenfaces. As seen from Figure 10, the top 4 eigenfaces are qualitatively similar to those of the cropped dataset in that the faces are blurry. However, the main difference is that we see multiple shadows of a head, which are caused by the non-aligned images. We compare this to Figure 11, which shows the 4 eigenfaces from mode 26 onward, which captures 90% energy. The main difference is we now see more detailed facial features such as the nose. We still see the multiply-aligned shadows, just like the top 4 eigenfaces.

An example of a reconstructed image can be seen in Figure 12. Comparing to Figure 7, while we still see a progression from a less blurry image to a more refined image, the 99% energy image is still blurry. This is perhaps because most of the variance being captured in the uncropped dataset comes from the background and the misaligned faces, rather than the distinguishing facial features.

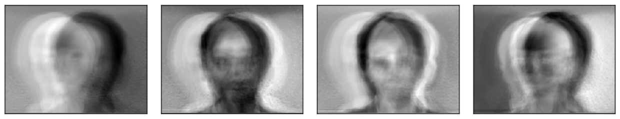

Fig. 10: First 4 eigenfaces for the uncropped dataset, ordered from left to right in decreasing singular values.

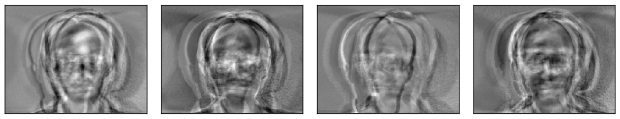

Fig. 11: 4 eigenfaces from mode 26 onward, which capture 90% energy for the uncropped dataset. The eigenfaces are ordered from left to right in decreasing singular values.

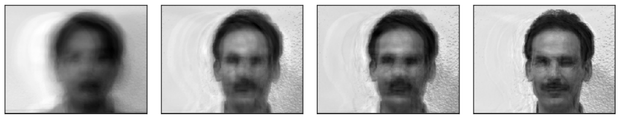

Fig. 12: Reconstructing an image with varying number of modes. From left to right, the number of modes are 1, the number of modes that capture 90% energy, 95% energy, and 99% energy.

Overall, the result reminds us of the importance of preprocessing one's images before performing low dimensionality analyses. In this case, background removal and alignment would help remove the variance from variables that we care less about, and instead allow the algorithms to focus on the more important variances, such as the distinguishing facial features.

C. Face Classification

We describe the setup for the face classification task. From the cropped dataset, we pick 10 people as shown in Figure 13. That is, the classification task involves 10 discrete classes, ranging from 1 to 10, and each input vector is a flattened image as described in Section II-B.

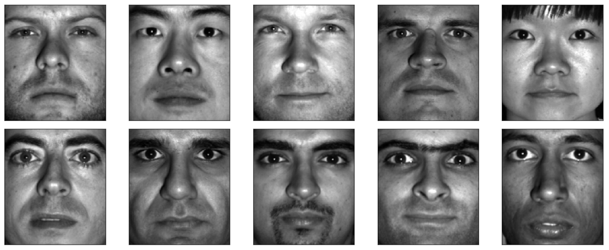

Fig. 13: 10 people used for face classification.

For the training set, we pick 7 lighting conditions, one of which has slight darkness on the left side, and another one has slight darkness on the right aside, as shown in Figure 14.

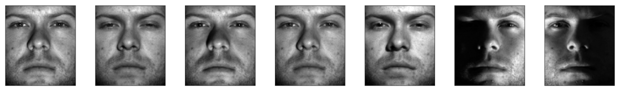

Fig. 14: 7 lighting conditions used for the face classification training set.

For the test set, we pick 2 lighting conditions, which correspond to darkness on the left and right side of the face, as shown in Figure 15. Thus, this setup intuitively tests the classifier's ability to remove face occlusion, given training samples of un-occluded, and weakly occluded faces.

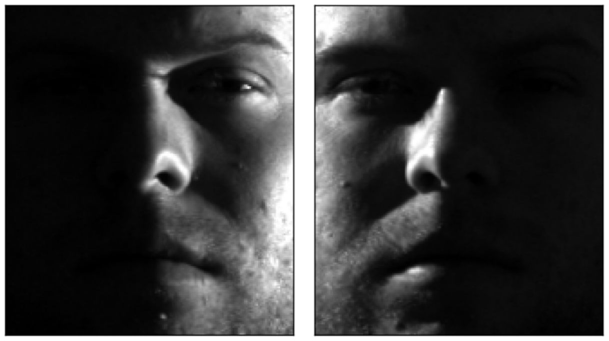

Fig. 15: 2 lighting conditions used for face classification test set.

To select the SVM regularization hyperparameter C as described in Equation 4, we perform 3-fold cross validation, and pick the C that minimizes the average accuracy score as described in Equation 2. We found that ranging C among 0.1, 1, 10, 100 all give the same accuracy of 71.4%, and so we just settle with $C = 1$. For KNN, we similarly perform 3-fold cross validation for the number of nearest neighbors to be considered. As seen in Table I, $K = 1$ gives the highest accuracy, and so we simply choose to use 1-NN for our evaluation.

TABLE I: Cross-validation for KNN on Face Classification

Number of neighbors	Accuracy
1	67.1
3	64.2
5	60
7	54
9	39.9

We train the models on the entirety of the training set, and report our results in Table II. We first note that although both approaches were able to achieve perfect training accuracy, neither of them were able to do just as well during cross validation. This is perhaps because the training data has very few samples of the darker images, and so training folds that contain only the bright images are unlikely to succeed in predicting on the darker images. We also note that the test performance of both approaches are poor. This attests to the difficulty of removing dark patches that are more severe than those encountered in the training set. KNN has a slightly better test performance than SVM, perhaps because KNN will simply try to match the dark test images to the dark train images that it has seen before, whereas SVM, which has also been trained on the bright images, will have a more complicated weighting that takes into account both bright and dark images.

TABLE II: Accuracy % of KNN and SVM on Face Classification

	SVM	KNN
Cross validation	71.4	67.1
Train	100	100
Test	30	40

Looking at an example error, we can see how challenging this task is. Figure 16 shows two test samples which managed to confuse both models. From a quick glance, it is possible that a human annotator might mistakenly judge both pictures to correspond to the same person.

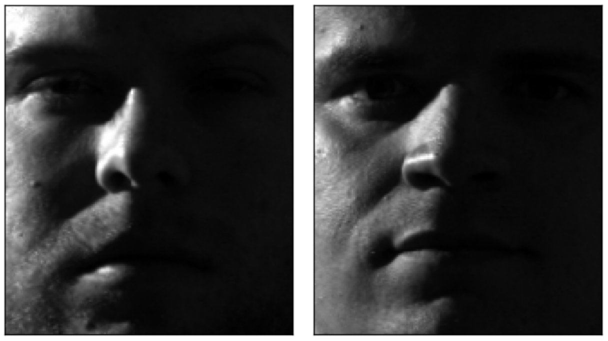

Fig. 16: An example of challenging test samples.

We seek out the most important pixels by inspecting the linear SVM weights that have the highest absolute values. As shown in Figure 17, we see that eyebrows, nose, mouth, and eyes seem to be the distinguishing features of the photo. This also points out one advantage SVM presents over KNN, despite its lower test performance for this setup. The availability of the pixel weightings allows one to understand which aspects of the input are important for the task.

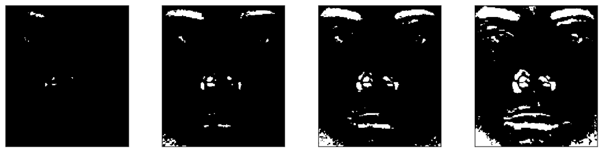

Fig. 17: From left to right, in white are the top 100, 1000, 2000, 4000 most important pixels according to SVM for face classification.

D. Gender Classification

We describe the setup for the gender classification task. From the cropped dataset, we manually labelled the gender of each of the individual as mentioned in Section IV-A. Unlike face classification, this task is now a binary classification task, where each sample image is either labelled as male or female. The feature vector representation of flattened images, however, remains unchanged. We divided the dataset such that the training set contains 23 males and 6 females, and the test set contains 6 males and 3 females.

To select the SVM regularization hyperparameter C as described in Equation 4, we perform 2-fold stratified cross validation as described in Section II-G, and pick the C that minimizes the average F-score as described in Equation 3. We chose $k = 2$ because there are only 6 females in our training set. We also use the F-score as described by Equation 3, where the positive class is female, to encourage the models to be able to detect the rarer female class. We found that ranging C among 0.1, 5, 1, 2 all give the same F-score of 0.65, and so we just settle with $C = 1$. For KNN, we similarly perform 2-fold stratified cross validation for the number of nearest neighbors to be considered. As seen in Table III, $K = 3$ gives the highest F-score, and so we use that for our evaluation. We note that there are several 0 F-scores. These are when the model fails to detect any of the (less than 3) female samples correctly.

TABLE III: Cross-validation for KNN on Gender Classification

Number of neighbors	F-score
1	0.4
3	0.5
5	0
7	0
9	0

We then train the models on the entirety of the training set, and report our results in Table IV. We see that although SVM was able to fully fit the training set, its test performance is lower than the test performance of KNN. On close inspection, it turns out that KNN managed to identify the female sample shown in the rightmost picture of Figure 18. One might argue that the label for the image is rather ambiguous, and so the classification error might not be that severe. It is also interesting to see that both algorithms managed to classify all the males correctly as males. Thus, the main challenge of this task setup is for the algorithms to correctly predict the rare female samples without misclassifying the males as females.

TABLE IV: F-scores of KNN and SVM on Gender Classification

	SVM	KNN
Cross validation	0.65	0.5
Train	1.0	0.5
Test	0.5	0.8

We seek out the most important pixels by inspecting the linear SVM weights that have the highest absolute values. As shown in Figure 19, we see that eyes are the most important features, then followed by the lighting surrounding the eye, and finally the moustache area. It is interesting that the important facial features for gender classification is different from those for face classification, as shown in Figure 17. In particular, it seems that eyes and moustache area are more important for gender classification, but the nose or mouth less so. Thus, a good model for one task setup might not be necessarily good for a different task setup.

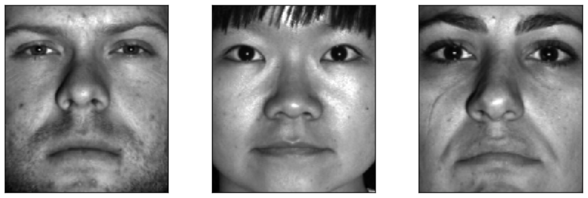
Fig. 18: From left to right, is a correctly identified male sample, a correctly identified female sample, and a female sample that was misclassified as male by SVM.

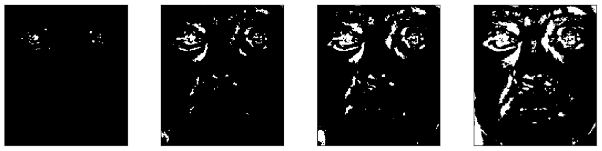
Fig. 19: From left to right, in white are the top 100, 1000, 2000, 4000 most important pixels according to SVM for gender classification.

E. Brightness Clustering

For the clustering formulation as described in Section II-H, we decided to see if we can segregate 2 different lighting conditions from one another. In particular, we use all the people from the cropped dataset, but only 2 lighting conditions as shown in Figure 20.

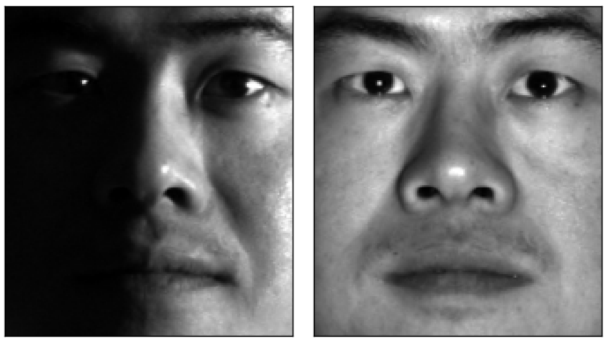
Fig. 20: The two lighting conditions to be segregated by clustering. One is a bright lighting, and the other is dark on the left side.

We apply k-means and GMM as described in Sections II-I and II-J. Surprisingly enough, both approaches were able to obtain perfect accuracy score as described in Section II-H. That is, without any labels, both approaches were able to group the images for all individuals based on the lighting condition. This task is perhaps well-suited to clustering because taking a euclidean distance between two images that have a lot of dark pixels on the left is highly likely to be a lot lower than comparing a dark image with a bright image.

We then investigate if the perfect performance still holds if we project each image to 2D as described in Section II-C. As seen in Figure 21, we still achieve perfect separation even with just 2-dimensional representations of the photos. We also observe that whereas k-means clusters are axis-aligned, GMM was able to identify cluster rotations that better aligns with the actual cluster shapes. This indicates that GMM is a more flexible model, and can capture more cluster shapes than K-means.

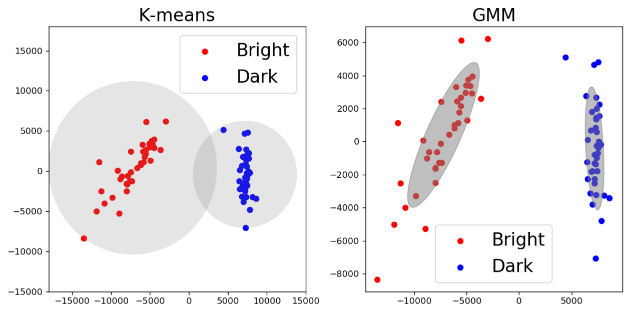

Fig. 21: 2-clustering of bright and dark photos. The sample colors are derived from the true labels. On the left are the k-means centers, and a visualization of how each center recruits the points closest to it. On the right are the GMM centers, and a visualization of the covariances of each cluster.

V. Summary and Conclusions

We have explored a variety of common machine learning tasks on the Yale face dataset, ranging from dimensionality reduction, classification, to clustering. For classification and clustering, we explored two different methods, and compared the performance of each method to one another not only in terms of test performance, but also on model interpretability. Overall, we showed that at each task setup, all algorithms were able to perform reasonably well.

References

[1] A. Georghiades, P. Belhumeur, and D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Trans. Pattern Anal. Mach. Intelligence*, vol. 23, no. 6, pp. 643–660, 2001.

[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[3] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[4] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[5] G. J. McLachlan and K. E. Basford, *Mixture models: Inference and applications to clustering*, vol. 38. M. Dekker New York, 1988.

Appendix A: Key Python functions

Eigenface Analysis

```
def perform_svd_analysis(images_mat, image_shape,
  ranks_to_reconstruct_with, file_prefix):
  avg_face = images_mat.mean(axis = 1)
  avg_removed_mat =  (images_mat.T - avg_face).T
  u, s, vh = np.linalg.svd(avg_removed_mat, full_matrices = False)

  # Average face.
  fig, ax = plt.subplots(figsize = (5, 5))
  show_image(ax, image_util.vector_to_img(avg_face, image_shape))
  fig.tight_layout()
  fig.savefig(IMG_PATH_PREFIX + file_prefix + '_avgface.png',
    format='png', dpi=DPI)

  # Top 4 eigenfaces
  fig, axes = plt.subplots(nrows = 1, ncols = 4, figsize = (10, 2))
  for i in range(4):
    show_image(axes[i], image_util.vector_to_img(u[:,i],
    image_shape))
  fig.tight_layout()
  fig.savefig(IMG_PATH_PREFIX + file_prefix + '_topeigens.png',
    format='png', dpi=DPI)

  fig = core.plot_energy_from_singular_values(avg_removed_mat)
  fig.tight_layout()
  fig.savefig(IMG_PATH_PREFIX + file_prefix + '_energy.png',
    format='png', dpi=DPI)

  # 4 eigenfaces after the 90 energy mark. I'm assuming ranks
  # contains 90 energy mark at index 1.
  fig, axes = plt.subplots(nrows = 1, ncols = 4, figsize = (10, 2))
  for i in range(4):
    show_image(axes[i],
    image_util.vector_to_img(u[:,ranks_to_reconstruct_with[1]+i], image_shape))
  fig.tight_layout()
  fig.savefig(IMG_PATH_PREFIX + file_prefix + '_90eigens.png',
  format='png', dpi=DPI)

  # Reconstruction with various ranks
  person_id = 0
  fig, axes = plt.subplots(nrows = 1, ncols = 4, figsize = (10, 2))
  for i, rank in enumerate(ranks_to_reconstruct_with):
    recon = image_util.get_reconstructed_face(u, rank=rank,
      original_vec=avg_removed_mat[:,person_id])
    show_image(axes[i],
      image_util.vector_to_img(recon + avg_face, image_shape))
  fig.tight_layout()
  fig.savefig(IMG_PATH_PREFIX + file_prefix + '_recon.png',
  format='png', dpi=DPI)
```

Plot Important Pixels

```
fig, axes = plt.subplots(nrows=1, ncols = 4, figsize=(12, 3))
for i, num_pixels_to_plot in enumerate([100,1000,2000,4000]):
  important_pixel_indices =
    np.argsort(np.abs(best_svm.coef_[0]))[::-1][:num_pixels_to_plot]
  important_pixels = np.zeros(data_loader.CROPPED_IMG_SHAPE)
  for pixel_index in important_pixel_indices:
    r = pixel_index // data_loader.CROPPED_IMG_SHAPE[1]
    c = pixel_index % data_loader.CROPPED_IMG_SHAPE[1]
    important_pixels[r,c] = 1
  image_util.show_image(axes[i], important_pixels)

fig.tight_layout()
fig.savefig(IMG_PATH_PREFIX + 'gender_importance.png',
  format='png', dpi=DPI)
```

SVM Training and Testing

```
from sklearn.svm import SVC
for kernel in ['linear']:
  for C in [0.1, 1,10,100]:
    model = SVC(kernel = kernel, C = C)
    print(f"Kernel = {kernel}, C = {C},
      Accuracy = {core.get_kfold_accuracy(model,
      train_mat, train_labels)}")
from sklearn.metrics import f1_score
best_svm = SVC(kernel = 'linear')
best_svm.fit(train_mat.T, train_labels)
train_preds = best_svm.predict(train_mat.T)
train_acc = np.sum(train_labels == train_preds) / len(train_labels)
print(f"Train acc = {train_acc}")
test_preds = best_svm.predict(test_mat.T)
test_acc = np.sum(test_labels == test_preds) / len(test_labels)
print(f"Test acc = {test_acc}")
```

KNN Training and Testing

```
from sklearn.neighbors import KNeighborsClassifier
for n_neighbors in [1,3,5,7,9]:
  model = KNeighborsClassifier(n_neighbors=n_neighbors)
  print(f"n_neighbors = {n_neighbors}, acc =
    {core.get_kfold_accuracy(model, train_mat, train_labels)}")
best_knn = KNeighborsClassifier(n_neighbors=1)
best_knn.fit(train_mat.T, train_labels)
train_preds = best_knn.predict(train_mat.T)
train_acc = np.sum(train_labels == train_preds) / len(train_labels)
print(f"Train acc = {train_acc}")
test_preds = best_knn.predict(test_mat.T)
test_acc = np.sum(test_labels == test_preds) / len(test_labels)
print(f"Test acc = {test_acc}")
```

Plot GMM and K-means Decision Boundary

```
def plot_2d(ax):
  # Plot on 2D.
  bright_coords = []
  dark_coords = []
  for img_i in range(dim_reduced_mat.shape[1]):
    coord = dim_reduced_mat[:, img_i]
    if data_labels[img_i] == 1:
      bright_coords.append(coord)
    else:
      dark_coords.append(coord)

  bright_coords_xy = list(zip(*bright_coords))
  dark_coords_xy = list(zip(*dark_coords))

  ax.scatter(bright_coords_xy[0], bright_coords_xy[1],
    c = 'red', label = 'Bright')
  ax.scatter(dark_coords_xy[0],
    dark_coords_xy[1], c = 'blue', label = 'Dark')
  ax.legend()

from scipy.spatial import distance
# Kmeans
kmeans = KMeans(n_clusters=2, random_state=0).fit(dim_reduced_mat.T)
print(core.score_cluster(kmeans.labels_, data_labels))

# Plot kmeans centers
# Radius of each center = to reach the furthest points.
cluster_radii = []
for ci, center in enumerate(kmeans.cluster_centers_):
  distances = []
  pts = dim_reduced_mat[:,kmeans.labels_ == ci]
  for pt_i in range(pts.shape[1]):
    distances.append(distance.euclidean(pts[:, pt_i], center))
  cluster_radii.append(np.max(distances))

fig, axes = plt.subplots(ncols = 2, figsize=(10,5))
ax = axes[0]
ax.set_title("K-means")
plot_2d(ax)
for c, r in zip(kmeans.cluster_centers_, cluster_radii):
  ax.add_patch(plt.Circle(c, r, alpha=0.2, fc = "grey"))
ax.set_xlim(-18000, 15000)
ax.set_ylim(-15000, 18000)

from matplotlib.patches import Ellipse
# GMM
gmm = mixture.GaussianMixture(n_components=2, random_state=0)
gmm_labels = gmm.fit_predict(dim_reduced_mat.T)
print(core.score_cluster(gmm_labels, data_labels))

ax = axes[1]
plot_2d(ax)
for i in range(len(gmm.means_)):
  center = gmm.means_[i]
  covars = gmm.covariances_[i]

  v, w = np.linalg.eigh(covars)
  u = w[0] / np.linalg.norm(w[0])
  angle = np.arctan2(u[1], u[0])
  angle = 180 * angle / np.pi  # convert to degrees
  v = 2. * np.sqrt(2.) * np.sqrt(v)
  ell = Ellipse(center, v[0], v[1],
                            180 + angle, color="grey", alpha=0.2)
  ell.set_clip_box(ax.bbox)
  ell.set_alpha(0.5)
  ax.add_artist(ell)
ax.set_title("GMM")
fig.tight_layout()
fig.savefig(IMG_PATH_PREFIX + 'cluster_lowd.png',
  format='png', dpi=DPI)
```

Dimensionality Reduction on Face Matrix

```
def get_dim_reduced_mat(data_mat, r):
  """Perform dimensionality reduction through SVD.
  Parameters:
  - data_mat (d x n np array), each column is a flattened image.
  Return;
  - reduced dimension data_mat (r x n np array), where r << d.

  How:
  - Get average face by getting column mean.
  - Subtract that average face from every column.
  - Perform SVD on the average-subtracted matrix.
  - The U from SVD is of shape d x r.
  - Each of U's columns is a d x 1 vector an eigen face.
  - Given an image vector, d x 1, you can then convert to a
    (r x 1) low D vector by projecting
    against each of the r eigenfaces, each also of dimension (d x 1).
  - That is, given image_vec = d x 1, you just do Ut @ image_vec
    to get an r x 1 low-dimensional vector.
  - So, given the entire data matrix of size d x n,
    you just do Ut @ data_mat to get an r x n matrix.
  """
  avg_face = data_mat.mean(axis = 1)
  avg_removed_mat =  (data_mat.T - avg_face).T
  u, s, vh = np.linalg.svd(avg_removed_mat, full_matrices = False)
  ur = u[:,:r]
  return ur.T @ avg_removed_mat
```

Reconstruct Face in Low Ranks

```
def get_reconstructed_face(u, rank, original_vec):
  """ Get a low-rank reconstruction of a face.
  Parameters:
  - u - the eigenface matrix. The columns are eigenfaces.
  - rank - the low rank to do reconstruction in.
  - original_vec - the vector representation of a face.
    Mean is already subtracted.
  Return:
  Low-rank reconstruction of original_vec.
  """
  ur = u[:,:rank]
  return ur @ (ur.T @ original_vec)
```

Plot Energy Distribution

```
def plot_energy_from_singular_values(data):
  U, singular_vals, Vh = np.linalg.svd(data, full_matrices = False)
  squared_singular_vals = np.power(singular_vals, 2)
  energy_values = squared_singular_vals / np.sum(squared_singular_vals)
  fig, ax = plt.subplots(figsize = (5, 5))
  ax.plot(range(1 , len(energy_values) + 1), energy_values,
  linestyle='--', marker='o', color='b')
  ax.set_xlabel("Singular value order")
  ax.set_ylabel("Energy")
  ax.xaxis.label.set_fontsize(15)
  ax.yaxis.label.set_fontsize(15)
  ax.set_title("Singular value energy distributions", fontdict={'fontsize': 15})
  energy_cumsum = np.cumsum(energy_values)
  for energy_percent in [90, 95, 99, 99.9]:
    # Index of first item greater than percent.
    mode = np.argmax(energy_cumsum > energy_percent / 100) + 1
    print(f"Need {mode} modes to cover {energy_percent} energy")

  return fig
```

Stratified K-fold F-score

```
def get_kfold_f1(model, train_mat, train_labels):
  """
  Get F1 score from k-fold validation.
  We use stratified k-fold to deal with the imbalanced class distribution.
  """
  num_fold = 2
  kf = StratifiedKFold(n_splits=num_fold, random_state = 1, shuffle = True)
  splits = kf.split(train_mat.T, train_labels)
  fold = 1
  total_f1 = 0
  for train_indices, test_indices in splits:
    fold_train_images = train_mat[:, train_indices].T
    fold_test_images = train_mat[:, test_indices].T
    fold_train_labels = train_labels[train_indices]
    fold_test_labels = train_labels[test_indices]
    model.fit(fold_train_images, fold_train_labels)
    preds = model.predict(fold_test_images)
    f1 = f1_score(fold_test_labels, preds)
    fold += 1
    total_f1 += f1
  return 1.0 * total_f1 / num_fold
```

Appendix B: Supporting Python codes

The rest of the supporting Python codes can be found on https://github.com/sjonany/AMATH563-public/tree/master/hw3/code