Classifying emotions with Logistic regression and Softmax using CAFE dataset

Aswin Ansar
UC San Diego
PID: A53252914
`afm007@ucsd.edu`

Balaji Sathia Narayanan
UC San Diego
PID : A53237173
`bsathian@ucsd.edu`

Abstract

We attempt to classify six different emotions (happy, maudlin, afraid, surprised, anger, disgust) by first getting the principal components from the CAFE images and training a binary classifier using Logistic Regression. We then attempt a multi-class classifier using Softmax regression. We find that the maximum test accuracy is $80 \pm 24.4\%$ for the happy-maudlin binary classifier and $65 \pm 32\%$ for the afraid-surprised classifier. The multi-class classifier has a test accuracy of 65% We finally attempt to classify the ten subjects using the same dataset, and get a test accuracy of 97.5% with as little as 10 PCA components.

1 Loading the CAFE dataset

We load the CAFE dataset using the loader function given in `dataloader.py`. We use six emotions ((happy, maudlin, afraid, surprise, anger, disgust). A representative figure containing sample images is shown in Figure 1.

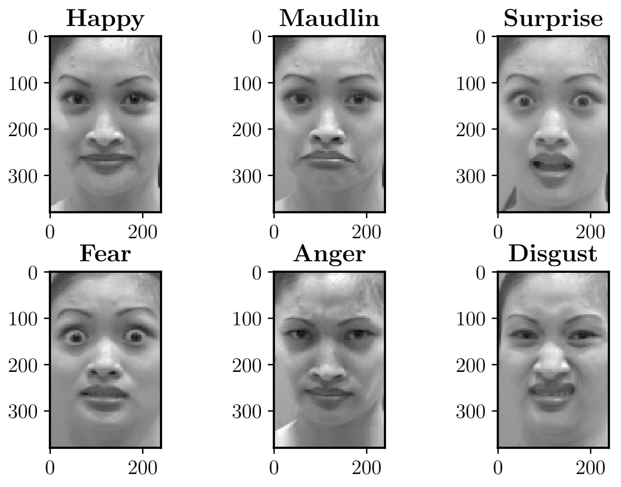


Figure 1: Representative figure showing the six emotions

2 Principal Component Analysis

The images we use have a dimension of 91200 (380 $\times$ 240), but the images themselves are only in a small subspace of this 91200 dimensional hyperspace. To use this information, we do a Principal Component Analysis (PCA) on the training images. We use only the training images because we do not "know" the test images yet, and incorporating them in PCA computation is indirectly using "forbidden" information, which is not a representation of the real world where we don't see the test dataset beforehand. Before doing PCA on the images, we "zero-mean" them by subtracting the average of each pixel from the images, followed by dividing them by the standard deviation of each pixel (aka "unit SD"-ing)

Since the covariance matrix has a dimension of 91200 $\times$ 91200, it is not possible to store it in memory. To combat this, we use the following technique - The covariance matrix is computed as

$$\mathrm{R} = \frac{(X - \langle X \rangle_c)^T (X - \langle X \rangle_c)}{N - 1}$$

where N is the number of images, X is the matrix containing the training data along each row, and $\langle X \rangle_c$ is a matrix whose rows are identically equal to the column average of X i.e., we take the average of each feature and subtract it from the features themselves. Now, we use the property that the eigenvalues of $A^T A$ and AA^T are the same, and the eigenvectors corresponding to nonzero eigenvalues of $A^T A$ can be obtained from those of AA^T by multiplying the latter with A^T.

In our case, the matrix $(X - \langle X \rangle_c)(X - \langle X \rangle_c)^T$ has a shape $N \times N$, where N is the number of images (does not exceed 60 in our case). This matrix can be computed very quickly and is compact enough to be stored in memory. Using the technique described above, we find the eigenvectors of the covariance matrix. Since the eigenvectors might not be normalized, we "scale" them by dividing them with the square root of the corresponding eigenvalue. The first six principal components of the "training dataset" of 48 images compiled by taking eight images from each of the six emotions (happy, maudlin, afraid, surprise, anger, disgust) are shown in Figure 2

We then project the images onto the normalized principal components, followed by dividing them by the square root of the corresponding eigenvalues to make them unit standard deviation after projection (the sanity check). We now obtain a set of "images" with reduced dimensions, scaled appropriately for logistic regression.

For the holdout and test sets, we subtract them with the mean of the training data, and project them using the principal components, followed by scaling them with the square root of the eigenvalues. This is to ensure the holdout and test datasets also "come from the same distribution", which is an important assumption in Machine Learning.

3 Binary classifier using Logistic Regression

We now implement a logistic regression based binary classifier following the instructions in the problem set. We just need to learn one set of weights because we only need to learn one probability i.e., the probability that the data falls into either of the two categories. The other probability is simply the difference of the former from 1.

As suggested, we split the two-emotion dataset containing 20 images into 16-image training, 2-image holdout and 2-image testing sets. We then train the classifier on the training set for ten epochs using three different learning rates. We use early stopping based on the holdout error, so that we have the best weights for that learning rate. We repeat this entire process 10 times, swapping the holdout and test images with images from the training set. This way, we get different training sets for each of the 10 iterations, where two images have been swapped out for the holdout and test images from the previous iteration.

3.1 Happy vs Maudlin

In this case, the two emotions classified are happy and maudlin. Following the steps outlined above, we obtain the training cost plot shown in Figure 3a for the three learning rates using 8 PCA

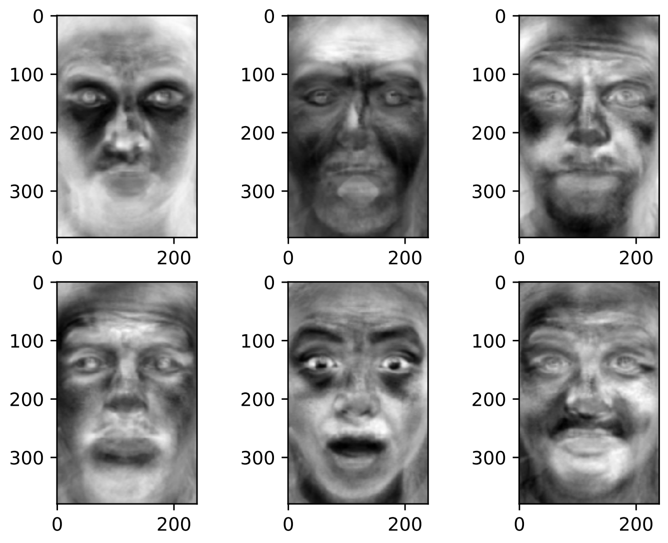
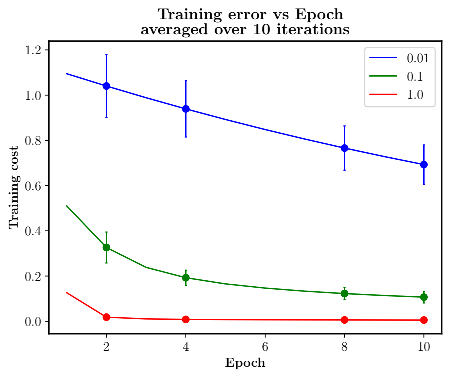
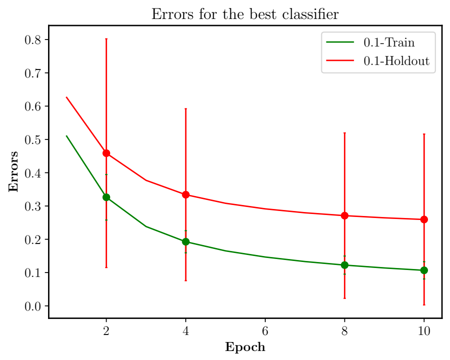


Figure 2: First six PCA components



(a) Happy vs Maudlin - Training error vs Epoch



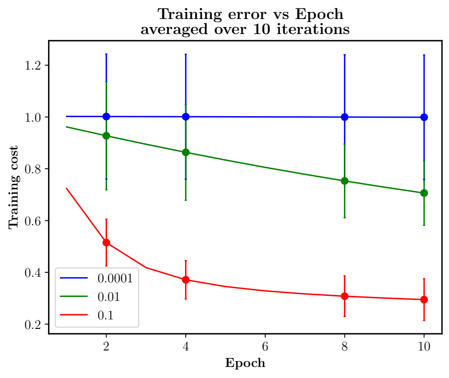
(b) Happy vs Maudlin - Best classifier training and holdout error

Figure 3: Happy vs Maudlin - Results using 8 PCA components averaged over 10 iterations

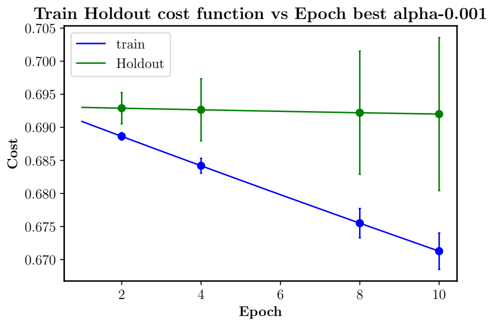
components. The training and holdout loss for the best classifier are shown in Figure 3b. The test results for the best classifier averaged over 10 runs using 4, 6 and 8 PCA components are shown in Table 1. The best test result was obtained with 8 PCA components and a learning rate of 0.01, and is equal to $80 \pm 24.4\%$. We think the high uncertainty is a statistical artifact i.e., we only had two images to test in each iteration, so the possible outcomes are simply 0%, 50% or 100%.

PCA_dim	α	Test accuracy (fraction)
4	0.1	0.7 (0.35)
6	0.1	0.75 (0.40)
8	0.1	0.8 (0.24)

Table 1: Best test results for Happy vs Maudlin using three PCA components. Standard deviation in parentheses



(a) Afraid vs Surprised - Training error vs Epoch



(b) Afraid vs Surprised - Best classifier training and holdout error

Figure 4: Afraid vs Surprised - Results using 8 PCA components averaged over 10 iterations

3.2 Afraid vs Surprised

We repeat the same process, but with the afraid and surprised emotions. The training cost functions are shown in Figure 4a, the best classifier costs in Figure 4b, and the table of results in Table 2. The best classifier has a test accuracy of 65 $\pm$ 32% .We find that the best classifier does only sparingly better than a random classifier (which has a performance of 50%). This could be because the afraid and surprised faces are closer to each other than the happy faces are to the maudlin ones. This can also be visually observed in the weight visualizations (Figure 7).

PCA_dim	α	Test accuracy (fraction)
4	0.01	0.55 (0.35)
6	0.01	0.65 (0.32)
8	0.01	0.65 (0.32)

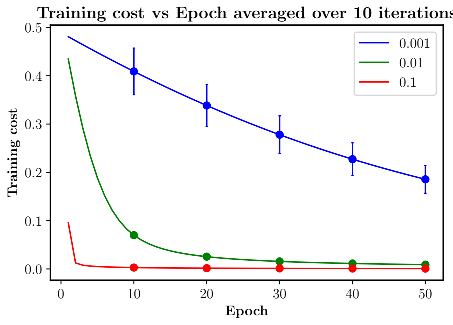
Table 2: Best test results for Afraid vs Surprised using three PCA components. Standard deviation in parentheses

4 Multi-class classification using Softmax

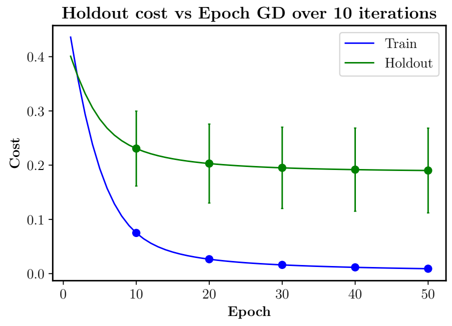
In order to classify all 6 types of emotions, we implemented a multi-class classifier using softmax regression. As opposed to logistic regression discussed in the previous section, Softmax lets us learn the probability that the data falls into one of the six emotion classes. Thus, Softmax is a more generalised version of logistic regression.

As instructed, we took all the 10 subject images given for each of the six emotions (happy, maudlin, surprise, fear, anger, disgust). So a total of 60 images were taken. We do not use both the happy faces because the training set becomes imbalanced in favour of the happy emotions otherwise, resulting in reduced performance on other emotions. During a run, one image for each emotion was considered as the test image, one random image which is not the test image was considered to be the holdout image from the remaining 9 images and all the other images were considered to be the train images. So overall, for each run the data was split as 48 train images (80%), 6 test images (10%) and 6 holdout images (10%). Using this method 10 runs iterating through each of the subject to be the test at least once was implemented. We did principal component analysis on the train images as explained in Section 2 to reduce the dimensions of all the images.

The training classes were fed in a one-hot encoded fashion, as a 48×6 matrix where ij^{th} entry is 1 if the i^{th} image belongs to the j^{th} class. The testing images were predicted to belong to class j if $w_j^T x^n$ is the maximum among all classes, where x^n is the n$^{\text{th}}$ testing image.



(a) Multi-class softmax - Training error vs Epoch



(b) Multi-class - Train and holdout cost for the best learning rate (0.01) vs Epoch

Figure 5: Multi-class - Results using 40 PCA components averaged over 10 iterations

4.1 Train vs Holdout Cost

We used the gradient descent algorithm to train our model. We tried out three learning rates - 0.001,0.01 and 0.1 . The Train performance and the hold out performance for all three learning rates were plotted and is shown in Figures 5a and 5b. 0.01 seemed to be the right learning rate as on using 0.001, train cost did not converge in the 50 epochs, and though the train cost converged, the holdout cost increased drastically for 0.1 after certain number of epochs.

4.2 Performance evaluation using a Confusion matrix

In a multi-class classification problem, in order to evaluate the performance of the model on the test data, it is important to use the confusion matrix as it keeps track of the mis-classification among the classes. This way we can get the true and false positives for each of the 6 emotions. The confusion matrix for the average performance for all the ten runs is shown in Table 3.

True(rows)/Predicted(Columns)	happy	maudlin	suprise	fear	anger	disgust
happy	80	0	0	20	0	0
maudlin	0	50	10	10	30	0
surprise	10	0	90	0	0	0
fear	10	10	20	50	10	0
anger	10	30	0	10	40	10
disgust	0	0	10	0	10	80

Table 3: Confusion matrix with % accuracy for each emotion along the diagonals

Each row in the confusion matrix represents the true class and the column the predicted class. The overall accuracy was calculated as the average of the sum of the diagonal elements. The overall accuracy obtained on using 40 PCA components and a learning rate of 0.01 is 65%, which is far better than a random classifier (16.66%). We can also understand from the confusion matrix the percentage of images from a particular class being mis-classified as some other class. For example, 20% of the total happy images were classified as fear.etc.,

4.3 Stochastic gradient descent vs Batch gradient descent

As instructed, we implemented the stochastic gradient descent (SGD) algorithm. We trained our model using the stochastic gradient descent and gradient descent now using 0.48 as the learning rate for the stochastic descent, and 0.01 as the rate for the batch descent, and the same set of train, test and hold out images for each run. The training costs for batch and stochastic gradient descent are plotted in Figure 6. We use different rates to compensate for the fact that the batch gradient descent sees the entire training set (containing 48 images), while the stochastic gradient descent sees only one image i.e, $(1/48)^{\text{th}}$ of the training set, and so the different learning rates "normalize" the differences

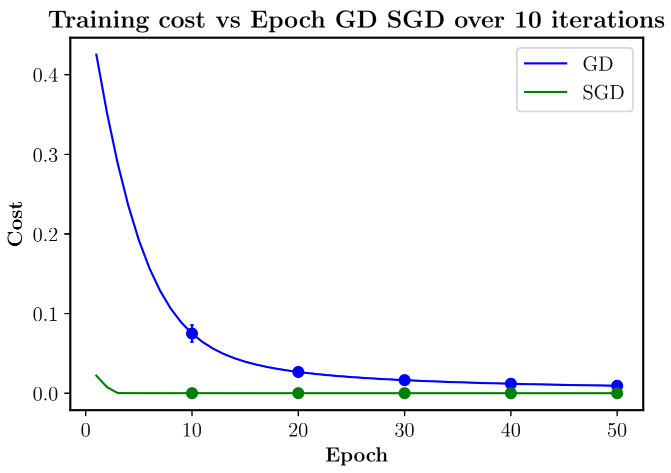


Figure 6: Batch gradient descent vs SGD for 40 PCA components

in the two processes under the assumption that every image is a good representation of the underlying sample.

We see that the stochastic gradient descent converges by around 10 iterations while the batch gradient descent takes nearly 50 iterations to converge to the same amount. This could be because of the frequent updates compared to the batch scenario, which help in propelling the system towards the minima faster than batch gradient descent.

4.4 Weights for each emotion

In order to visualise the weights obtained from training the model, we used the weight matrix of shape $d \times c$ where d is the no of PCA components used and c is the number of classes. The eigenvectors matrix has a dimension of $91200 \times d$. On multiplying the transpose of both matrices, we get an array of size $c \times 91200$. each row represents the weights learned for each emotion. The vectors are then scaled to 0-255 range so that they can correspond to pixel values and can be visualized. The weight image for each of the emotions were plotted by reshaping the 91200 vector to 380×240 array. The weights are shown in the Figure 7 for the scenario involving 40 PCA components, where the learned emotions can be verified visually (i.e., the "happy" image actually looks happy).

We see such images because what we have essentially done is a transformation from the reduced dimensions back to the original dimensions (i.e., projecting the manifold back into the bigger space). Mathematically, we can say the following:-

Let $\mathbf{\Phi}^(\mathbf{i})$ represent an image in the physical space. To transform it to the eigenspace, we multiply it by the matrix containing the eigenvectors along the rows. Let's call this unitary matrix U, i.e.,

$$\mathbf{\Phi_e^{(i)}} = \mathbf{U}\mathbf{\Phi^{(i)}}$$

What we would like to achieve is essentially learning a set of weights $\mathbf{w_j^P}$ that classify an image in the physical space, i.e.

$$C^{(i)} = \max_j \left(\mathbf{w_j^P}\right)^T \mathbf{\Phi^{(i)}}$$

Since doing it in the physical space is expensive, we do it in projected subspace, and hence to transfer it back we need to do the following

$$\begin{aligned} C^{(i)} &= \max_j \left(\mathbf{w_j^e}\right)^T \mathbf{\Phi_e^{(i)}} \\ &= \max_j \left(\mathbf{w_j^e}\right)^T \mathbf{U}\mathbf{\Phi^{(i)}} \end{aligned}$$

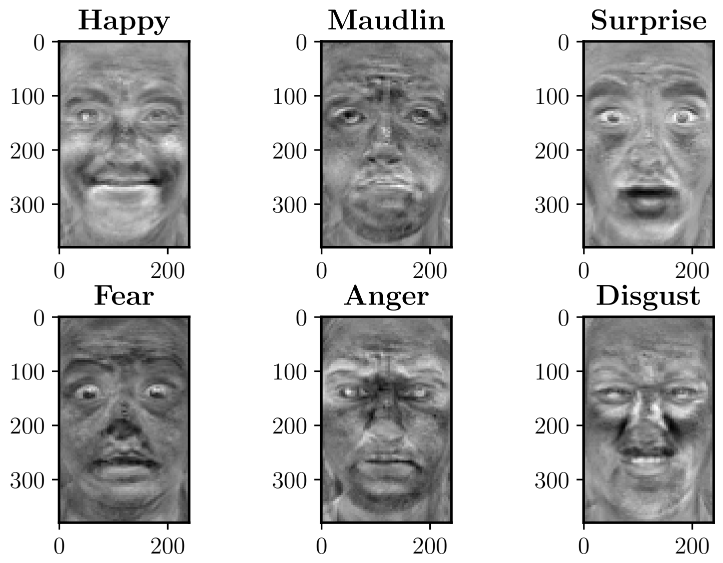


Figure 7: Weights for each of the emotion using 40 PCA components

where w_j^e are the weights in the eigensubspace. So, we are essentially doing a reverse-projection by multiplying with the eigenvectors. Since the classification merely involves taking the "projection" of the image on these weight vectors, these vectors are essentially the "happiest" or "saddest" faces that can be found.

Intuitively, we can say that each weight image is a ghostly representation of an emotion. While trying to classify, the brighter regions or the pixels with highest values help segment the features/dimensions of the image that are most vital in characterising a particular class. In other words, while testing, when a happy image comes in, It looks for the patterns that the lips, eyes and the face forms. These patterns are learned for every emotion and is used for classification.

4.5 Bonus - Classify subjects

In this task, we classify the subjects instead of the emotions. For each subject we have 8 images. Here we are including the happy with teeth and the neutral faces as well. Thus the total number of images for all the 10 subjects is 80. The performance and the cost should be averaged over 8 runs which considers one image from a particular subject as the test image (the iteration number), a random image as the holdout and the remaining 6 images to be the training set. Hence, for each run, We obtained a total of 60 training images, 10 test images and 10 holdout images. We then do a PCA on the training images (following Section 2) and for this task, take only 10 PCA components.

We used the batch gradient descent algorithm with a learning rate of 0.01 to train and obtained the learned set of weights which is a $d \times c$ matrix, c in this case being 10 (number of subjects) We also evaluated the performance over 8 runs and the average performance over these 8 iterations was captured in a confusion matrix, show in Table 4. We see that this classification is very efficient, with most subjects being classified perfectly. In this case, we obtain a test accuracy of 97.5%. With 40 PCA components, we achieve perfect test accuracy (100%). This could be because when it comes to subject detection, a lot of features are common among images with the same label, unlike in the previous case where only a few features were common (the expression of emotions). Hence, only a small number of principal components are required to capture all the essential information, resulting in near perfect classification.

The weights are visualised as images using the method described in Section 4.4 and 10 weight images are obtained, as shown in Figure 8. Each image looks like a ghostly representation of a subject. Weights for a particular subject represents the patterns of features recognised as the characteristic of a subject from the train images.

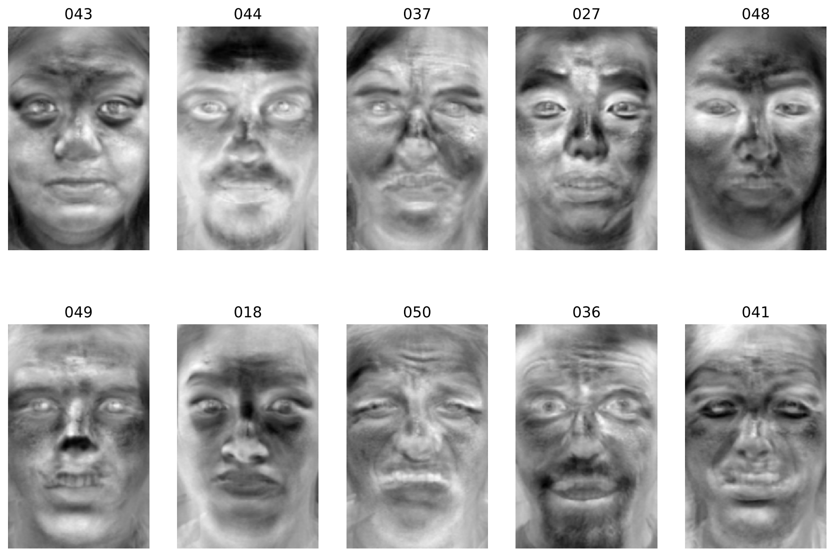


Figure 8: Weights for each subject trained using 10 PCA components with a learning rate of 0.01. The titles describe the subject number from the dataset

True(rows)/Predicted(Columns)	043	044	037	027	048	049	018	050	038	041
043	100	0	0	0	0	0	0	0	0	0
044	0	87.5	0	0	0	0	0	12.5	0	0
037	0	0	100	0	0	0	0	0	0	0
027	0	0	12.5	87.5	0	0	0	0	0	0
048	0	0	0	0	100	0	0	0	0	0
049	0	0	0	0	0	100	0	0	0	0
018	0	0	0	0	0	0	100	0	0	0
050	0	0	12.5	0	0	0	0	87.5	0	0
036	0	0	0	0	0	0	0	0	100	0
041	0	0	0	0	0	0	12.5	0	0	87.5

Table 4: Confusion matrix with % accuracy for each subject along the diagonals

5 Conclusions

We train a binary and a multi-class classifier using logistic regression and softmax regression respectively on the CAFE dataset. We find that the multi-class classifier beats out random classification, and the binary classifier only does a tad better than random picking when it comes to two classes that look very similar while beating out a random classifier on two classes that look significantly different. On the other hand, the multi-class classifier does a good job on subject classification even with very little data (10 PCA components).