ENEE633: STATISTICAL PATTERN RECOGNITION

PROJECT 1

Implementing fundamental classifiers for face classification

By:
Anirudh Nakra
UID: 118134444

ENEE 633: Statistical Pattern Recognition
Project 01: Implementing Fundamental Face Classifiers

Anirudh Nakra

University of Maryland,
College Park
Prof. Behtash Babadi

November 14, 2021

Abstract

In this project, I implement various fundamental classifiers such as Bayes, KNN, Kernel SVMs and Boosted SVM. Due to the constraint of working on a high dimensional data, I also apply dimensionality reduction techniques such as PCA and MDA and study their effect on the misclassification rates of the classifiers. I conduct experiments to find the significance and impact of hyperparamaters such as K in KNN and the number of classifiers in Adaboost algorithm.

1 Data Preprocessing

1.1 Dataset manipulation

The data pipeline is a very interesting part of the project. I am working with two datasets, DATA and POSE although the algorithms that have been developed can easily be made to work on ILLUMINATION through appropriate data restructuring. I use structures as a way to encapsulate all the data in a format that is very easily accessible and rarely inconvenient. This allows me to index over the structure as if I am indexing a 1d array but in fact each element of that array is in itself a 2d matrix. This allows me easily visualise what data I have present and how I am passing it through. After reshaping the data and putting it into a struct the size of the observation data, I do the labeling according to the classification task that is to be solved. The program accepts an input from the user and labels the data according to that input. This means that if task 1 has to be performed, the structure looks like a 200*3 array with each column of the struct representing Neutral, Expressive and Illuminated images according to the specifications of the dataset DATA. Further since we need to classify with respect to the subject labels, this means that each row corresponds to a particular label in my struct.

For classification task 2, the preprocessing pipeline is a little more complicated. For task 2, we need to classify between the Neutral and Expressive faces of a subject. The struct paradigm overcomplicates things here. It is easy enough to label each column to have a label but the segmentation into disjoint training and testing sets after that is tricky. From the 200*3 array of structs, I go to a 600*1 array of structs where each element in the struct is a 504*1 array by itself. This allows me to use standard indexing practices to segment my dataset into training and validation sets when I require them to train and test my classifiers.

1.1.1 Training-Testing Partition

Developing an efficient training and testing algorithm was something I was really exploring. Rather than brute forcing training and validation sets, I opt for a more autonomous approach. Using standard random generator functions, I transform my complete dataset to a test-train partition depending on the size provided by a PRNG program. This allows me to not worry about the train-test procedure while at the same time enforces that I am not testing or training on the same subset again and again.

1.1.2 Standard Estimators

Another important step that we need to take into account before starting the classification algorithm development is the calculation of various standard estimators. For our case this implies the mean as well as the covariances of the data. The mean and the covariances are very important statistical tools that will be used in both dimensionality reduction as well as the Bayes Classifier. I opt to compute these estimators before I start developing the code for any other applications. In order to compute the covariances and the means, I create a new function called "standardestimators.m" that can be used to solve the means and covariances for both the classification tasks by calculating either the class wise means or the whole dataset means whichever are desired.

1.1.3 MDA

An MDA solver has been implemented in the project to reduce the dimensionality of the data I am working with. The MDA solver operates on the full dataset and reduces the dimensions of each image in the presented dataset to a specified dimension size that is passed as a parameter. The program takes as input the mean of all the classes as well as each class covariance (calculated by the program beforehand) and then solves for the within and between scatter covariances. It also computes the anchor point mu0 which is a linear combination of all of the class means and is essential to design the between class scatter matrix. We can then simply calculate the eigenvalues of the matrix $\Sigma_w^{-1}\ \Sigma_b$ and then project our data on to as many dimensions as we want. However there are two major catches here. The first catch is the fact that the Class covariances for the dataset are singular and as a result the within scatter matrix is singular as well. This will create big issues when moving forward with classifiers that need some covariance information to classify the data points (especially Bayes). Also we need to compute the inverse of the within class matrix to get the desired eigenvalues. If the matrix is singular, this inverse will not be defined and our MDA will fail.

To fix this problem, I tried out two techniques. One way to solve the singularity of the covariances (and the within scatter matrix) is to add a small non zero element to the diagonals until the determinant of the desired matrix is non zero. I ran a loop that keeps on adding a particular positive constant to the diagonals until the determinant of the covariance is non zero and is thus regularised. Another way to regularise the covariances is to just use their peso inverses instead of the actual inverses wherever the inverses are required. However I found out that the pseudoinverses, although good generalisations, were still not able to provide a non zero determinant and even if they did, the classifiers and the dimensionality reduction techniques' performance suffered a lot. Therefore, I chose to go ahead with the first option.

We need to keep in mind that the multiclass LDA or the MDA method can only project the data onto dimensions that are lower than the number of the classes. This is because the rank of the matrix $\Sigma_w^{-1}\ \Sigma_b$ is at most M-1 if there are M classes. Thus for the second classification task, we can project to only a single dimension. This condition felt very restrictive in me designing future techniques. I, therefore, adopted a new approach to the problem. Instead of just considering the case of the Neutral vs the Expressive faces, I also considered the Illuminated faces. Thus I converted the two class problem into a generalised three class problem according to the given data. This allowed me to project my data onto either 1 or 2 dimensions depending on my choice since I now had 3 classes. As we will see later, this generalisation has the benefit of me being able to generalise my code very well without losing any accuracy in the results. The MDA of 3 classes rather than 2 classes does not affect our classification task hugely and there are only minor increments in the error rates. Classification task 1 can however be decreased to any dimension in the range [1,200].

1.1.4 PCA

Implementation of the PCA algorithm is really straightforward. For implementing PCA, we just need to calculate the covariance matrix of all the data and then perform eigendecomposition on the matrix. We then sort the eigenvalues according to their magnitude and truncate them to however many dimensions we want to project our data onto. The only really tricky part while implementing PCA is to make sure to recenter the data. Recentering the data implies that we subtract the mean of the ith component of all observations from the ith component of each observation. This kind of represents a normalisation procedure which can be explained in a very intuitive way. We can think of this procedure as normalising each pixel of all the images in a dataset that consists of a huge amount of pictorial information.

After performing the aforementioned techniques, we now have a dataset that is reshaped into a format that is easy to work with using struct manipulation, has its standard estimators computed and has a lower dimension than what we started with. Now we can move on to the main part of the project, which is implementing various classifiers.

2 Classifiers

2.1 Bayes' Classifier

Bayes classifier is the first fundamental classifier I implement. Since I already have an implementation of the means and covariances of the various classes, I can just go ahead and implement the real core algorithm.

$$f_Y(x) = \frac{1}{\sqrt{(2\pi)^n|\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(x-m)^T\mathbf{\Sigma}^{-1}(x-m)\right)$$

Using the above formula,I calculate the likelihoods of the various classes and then find out the index that corresponds to the class that maximises this likelihood function. The class with the highest likelihood is the class the testing sample gets assigned to. This function is really easy to solve since we have already gotten rid of the major issues such as singular covariances, etc. It was interesting that the Bayes classifier was performing so well with and without MDA for the classification tasks. At first, I was getting an increase in error for the Bayes classifier after the MDA based dimensionality reduction. I attributed this error to one of either of two issues. Either the data columns of data were correlated in some aspect or that one dimension was too low for the Bayes Classifier to predict class labels accurately. This was handled by me as explained above. Replacing the pseudo inverse with the regularised covariance inverse estimates and generalising the two class problem to the three class problem we originally started with in the data resolved the issues instantly.

The Bayes classifier did not perform very well on the DATA dataset for classification task 1. However I attribute this finding to another aspect. The DATA dataset is a weird data for classification task 1. We only have three observations per sample and the obvious choice for the training-testing partition is reduced to just a single case where we take two images as our training data and test on the remaining one. This inherent scarcity of data to train an test on makes me think that the issue is not with the classifier but rather with the dataset itself. This was one of the main reasons I wanted to implement the first task on another dataset such as POSE. The 4 % error in POSE signifies that the classifier works with the utmost accuracy.

Other than just implementing the normal Bayes Classifier, I also needed to implement something which I call the "Normalized Bayes Classifier". This classifier normalised the value of the exponential according to a parameter which is passed through the function as an argument. I came up with the idea while exploring fixes for my program. When I was going through the implementation for the Bayes Classifier on POSES for classification task 1, I noticed that I was getting values inside the exponential that were really high and which ultimately drove the likelihood function to 0 regardless of the actual value due to overflow conditions. I then modified my Bayes function to circumvent those overflow conditions by creating a modified form of the exponential that would normalize the values to a specified range. Although its really application specific, I found this trick to be something that allowed me to curb issues that might have arose in the future implementations.

2.2 K Nearest Neighbours

My KNN system performs really well for both classification tasks although it also suffers from the same issues for dataset DATA as the Bayes classifier does in classification task 1. Many metrics can be used for designing the KNN classifier. I experimented with a couple of them. Although I was really keen on moving forward with Mahalanobis and Minkowski distances, the algorithm performed way better with just a simple pairwise euclidean distance than either of the other ones. Through the experiments, I also noticed that the theoretical bound of the KNN classifier as a classifier that performs at most twice as worse as the Bayes classifier to be very true. In fact, in some instances, the difference in misclassification rates in the two classifiers was merely 0.02.

The trend with different choices of k's is another interesting observation I had. I noticed that the KNN classifier performed better for odd values of k. This is very intuitive and probably because of the way my experiment is set up. Since I do not implement a concrete tie breaking strategy and just *toss a coin*, there must be cases when k is even where there are multiple classes having equal proportion of neighbours. A weighted KNN might be an improvement along with a tie breaking policy. This might be explored in future revisions of the project. Usually with an increase in k from 1 to 10, there is an increase in accuracy with the classifier performing the best for values k=5,7 and 9 when looking at average errors across classical, PCA and MDA implementations.

2.3 Kernel Support Vector Machine

The kernel SVM is an interesting classifier to implement. In the project, I implemented the SVM classifier using Linear, polynomial and RBF kernels. Rather than resorting to solving the primal SVM problem, I approached the dual problem instead. The dual problem can be neatly reconstructed into the following equation:

$$\min_{\alpha} \frac{1}{2}\alpha^T G\alpha - \alpha^T \mathbb{1}$$

This is now in the form of a standard quadratic programming optimization problem. There are several solvers available in the form of packages. I use the inbuilt MATLAB function 'quadprog' function which accepts the equality constraints of the dual problem and gives an optimal alpha for the convex optimization problem. This alpha can then be used to predict the labels according to the following equation:

$$y' = sign(\sum_n \alpha_n y_n K(x_n, x'))$$

2.3.1 K-Fold Cross Validation

For K Fold Cross Validation, I need to divide my dataset into k partitions. Then I need to use k-1 partitions for training my SVM and the last partition for testing for all possible permutations of the training and testing sets. This is made very easy with an object called cvpartition. The cvpartition object essentially returns an array of indices depending on the number of observations and the value of k passed through the function. Using these indices, the generation of a training-testing partition is a trivial matter.

The issue however is the optimal parameter calculation. My designed program, although functional, is quite memory intensive and resource hungry. Even for as low as three values of iteration for the optimal r for polynomial case, it becomes a humongous time consuming task. Therefore to optimise the system, I have logarithmically incremented my values of both r and sigma to find the one that has the least average error over all of the cvpartition generated partitions.

2.4 Linear Support Vector Machine with AdaBoost

Implementing AdaBoost is a matter of implementing the 4 main steps of the algorithm. These are as follows:

- INITIALISATION Initialise the weights of the samples in the training data as the reciprocal of the number of observations. For i=1,2,3,4,.......k, do:
- PROBABILISTIC WEIGHTS Normalize the weights to form a PMF.
- WEAK CLASSIFICATIONS Find a weak classifier that performs at least better than a coin toss
- ASSIGNMENT Calculate epsilon as the weighted error and the coefficients as the satndard logarithmic formula.
- FINAL STEP Return to the second step

Here I implemented a neat little optimization idea. Since, I had a working prototype of a linear SVM that was giving me error percentages of around 20 %, the 2nd step became trivial. The explanation is rather simple. The formula in step 3 wants us to find a classifier such that the weighted misclassification rate is less than 0.5. To get a straightforward, efficient and easy solution to the above problem, I went ahead and implemented my 80% accurate linear SVM on 66% of the data, sampled according to the distribution provided by step 1. This essentially ensured that I would meet the stopping condition in step 2 on the first check itself and I would not have to employ techniques such as gradient descent to recursively find optimal solutions.

The AdaBoost SVM performs very well for the most part. It rivals the accuracies reached by high order polynomial and RBF kernels and greatly improves from the 80% accurate Linear SVM as well. Although the patterns for Adaboost applied on the PCA reduced data are hard to figure out with respect to the projected dimensions, we can still clearly see that the error rates achieved are very close to 10 percent which is better than the RBF kernel with σ=1 and almost as good as the polynomial kernel with r=20.

3 Experimental Observations

3.1 Task 1

3.1.1 DATA

Bayes and KNN on data reduced by PCA and MDA have been plotted.

3.1.2 POSE

Bayes and KNN on data reduced by PCA and MDA have been plotted.

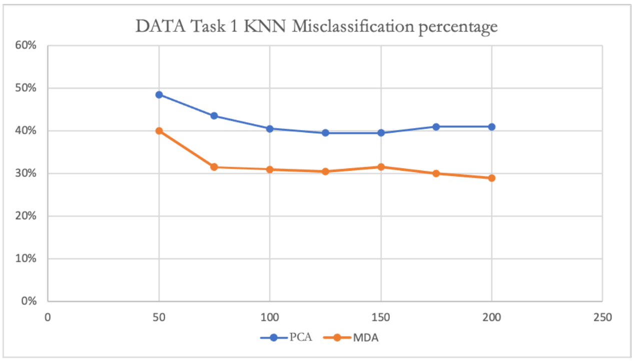

Figure 1: KNN Error Percentage v/s Dimension size

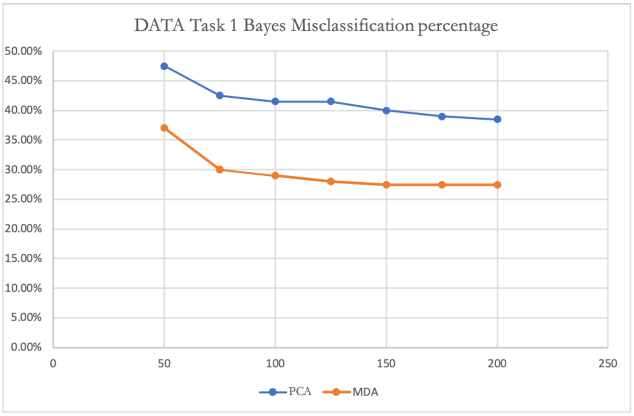

Figure 2: Bayes Error Percentage v/s Dimension size

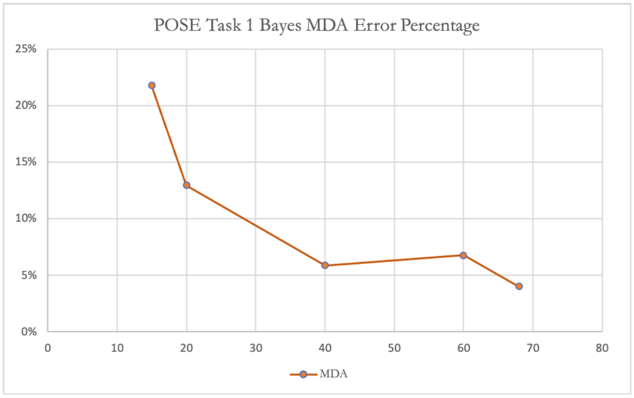

Figure 3: Bayes Error Percentage MDA v/s Dimensions

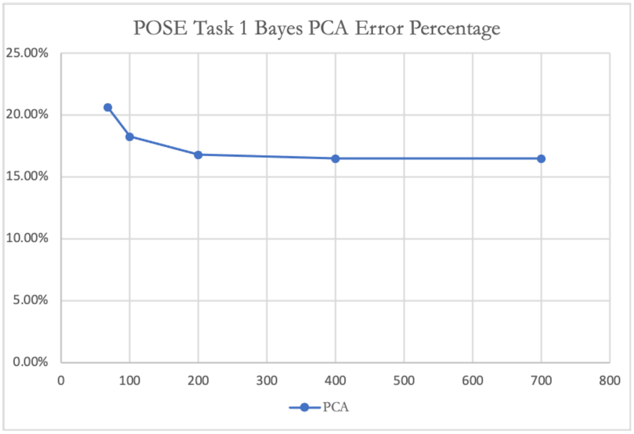

Figure 4: Bayes Error Percentage PCA v/s Dimensions

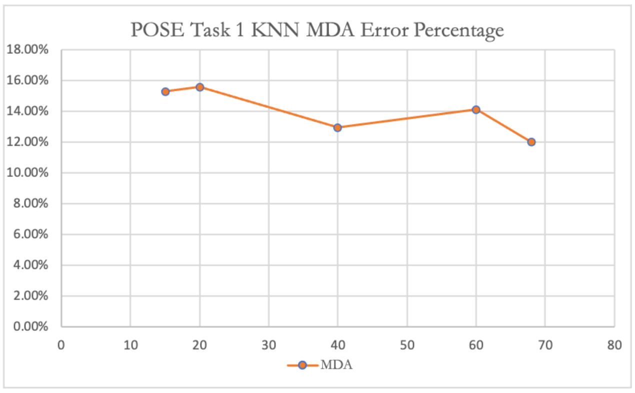

Figure 5: KNN Error Percentage MDA v/s Dimensions

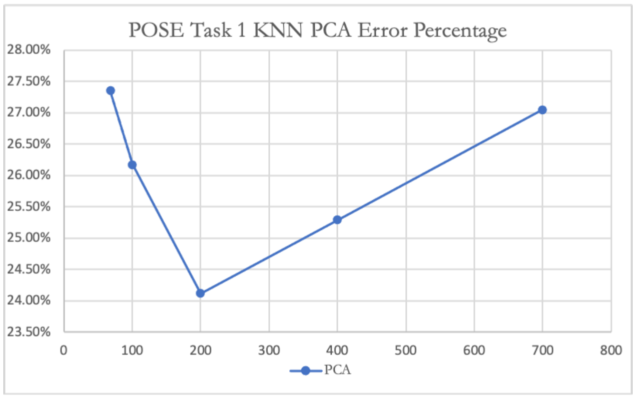

Figure 6: KNN Error Percentage PCA v/s Dimensions

3.2 Task 2

Every Classifier using both MDA and PCA has been implemented.

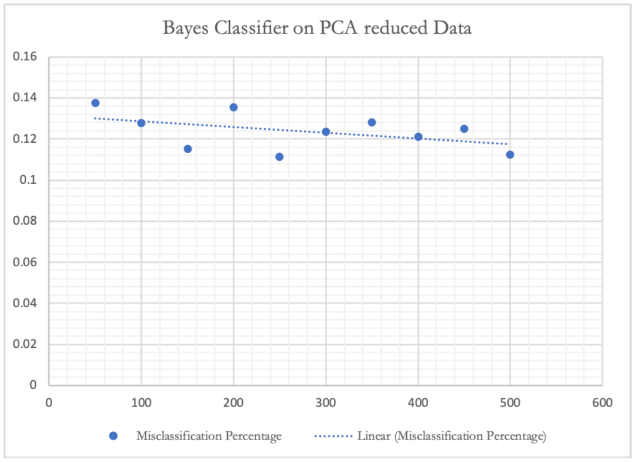

Figure 7: Bayes Error Percentage PCA v/s Dimensions

3.3 Features and Limitations

Over the course of the project, I implemented some good optimisations (both logic and algorithm based) and there were some issues that still need to be fixed. These are encapsulated in the bullet points below:

- THE ADABOOST TRICK By designing a system algorithm that trains a very good weak Linear SVM

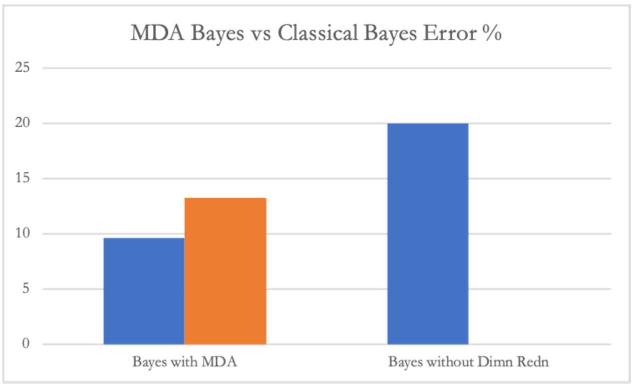

Figure 8: Bayes MDA v/s Bayes w/ no dimn redn

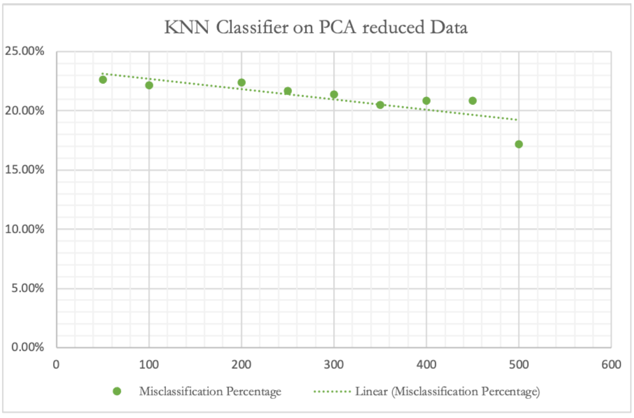

Figure 9: KNN PCA v/s Dimensions

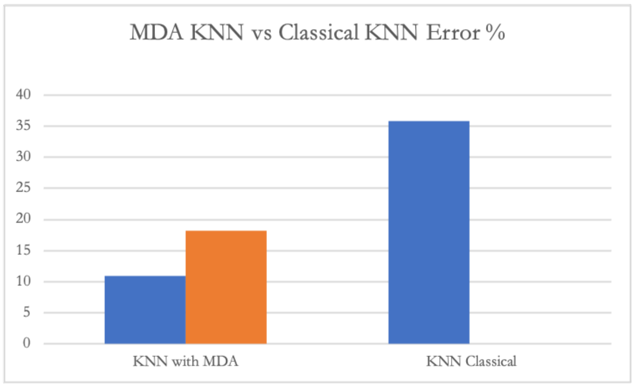

Figure 10: KNN MDA v/s KNN w/ no dimn redn

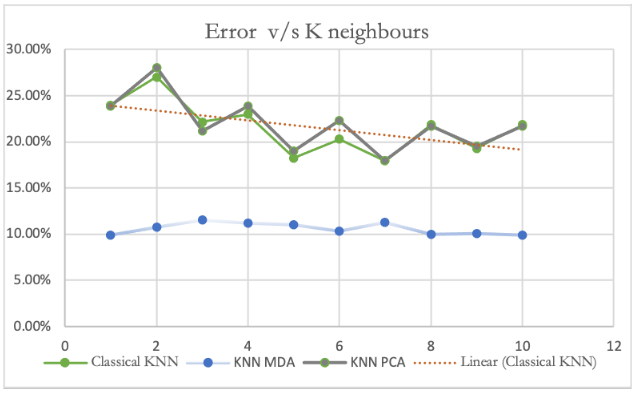

Figure 11: Effect of K on KNN performance

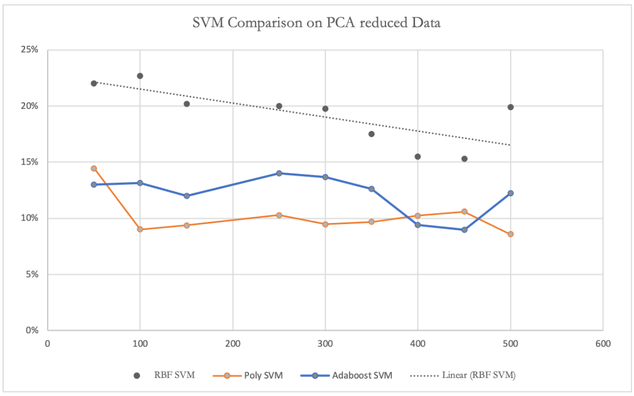

Figure 12: Comparing Boosted SVM with Kernel SVM

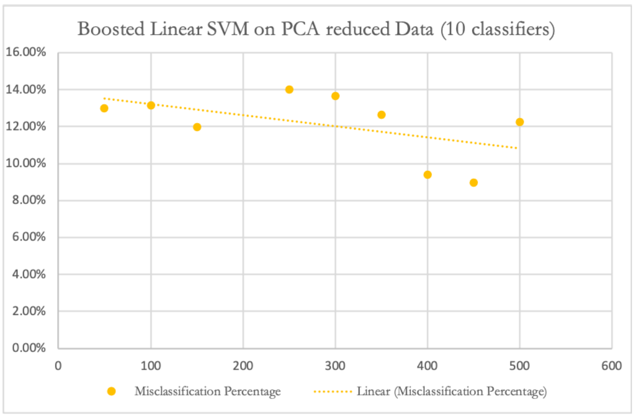

Figure 13: Boosted SVM on PCA reduced data

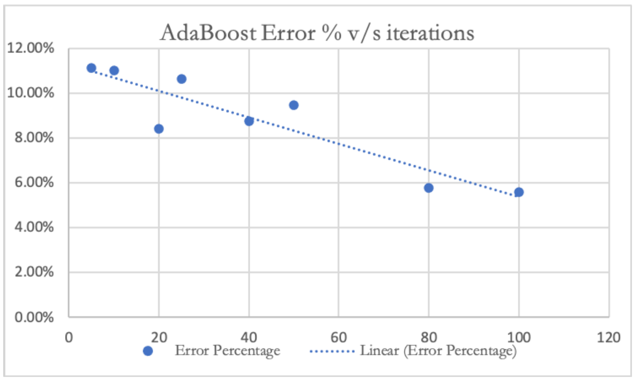

Figure 14: Changing the AdaBoost iterations

on 2/3rd of the data, circumvented the iterations in AdaBoost algorithm step 2.

- Generalising Task 2 Rather than only work on the two faces problem, generalised the MDA and the Classifiers to discriminate between the 3 faces with high accuracy. This can be easily converted to a system that discriminates between only Neutral and Expressive as well.

- The SVM Dual Rather than solving the primal problem or the dual problem using iterative and complicated procedures such as gradient ascent/descent, converted the problem to a simplistic quadratic programming solution with many solvers available.

RBF SVM	Polynomial SVM	Boosted SVM
6.25%	9.23%	5.97%

Table 1: Error for SVM's with MDA

- Normalizing Bayes Without changing the essence of the likelihood calculation, I introduced a normalising coefficient in the formula to help prevent overflow issues.

- Randomizing Test-Train Randomized the test-training data each time the program is run.

- Struct Manipulation Cleaner data extraction using structures which help in visualising the data really well.

- Modularity and using Excel Unfortunately due to time constraints, a lot of my code was not very modular and therefore I used Excel to plot curves after running experiments many times.

4 Key Conclusions

I can conclude that we have created an efficient face classification system that accurately estimates both csubject labels and Neutral v/s Expressive faces while also getting very low misclassification rates. I observe error rates that are in the range of 10-20 % for both KNN and Bayes given that we reduce the data using MDA. In the case of PCA, the classifiers still perform really well but there is a loss in performance due to the data not being projected along the direction of maximum separation. Through plots, I analyse the significance of k in the KNN algorithm for our problem and figure out some other relationships such as the impact of the number of iterations in AdaBoost.

The system proposed has also been optimised heavily using both algorithmic knowledge as well as some "tricks". The next step would be to optimise the code memory-wise and make it more modular to work with. I would also like to explore the ILLUMINATION dataset in future work on task 1.