STATISTICAL PATTERN RECOGNITION

PROJECT - 1

November 16, 2018

Rohit Kashyap
Directory ID: rohit94
University of Maryland College Park
MS. in Telecommunications

Contents

1 PRE CLASSIFICATION

1.1 Data-Extraction

Preprocessing the data before classification is the lowest level of abstraction to data classification. It involves removing noise and distortion from the data set and also allows to control the feature dimensionality. The computation time can blow up if dimensionality reduction is not performed on the data set.

The first dataset has been considered for different classification tasks. First dataset *data.mat* is unrolled into a $2D$ matrix of the form, X as opposed to the *face* matrix. *face* is a $3D$ matrix spanning over N, $2D$ images.

$$X = \begin{bmatrix} — & x_1^T & — \\ — & x_2^T & — \\ & \vdots & \\ — & x_N^T & — \end{bmatrix}$$

which is an N x M matrix containing N samples over M features.

After unrolling X we are presented with 600 test images to build our classifier. The classification can be performed over either the 200 subjects whose *neutral* or *expression* face test sample is presented. Note the *illuminated* sample can be discarded as it is just a scaled version of either the *neutral* or *expression* sample, by doing so we improve upon the rank of our feature matrix.

1.2 Dimensionality Reduction

To reduce the dimensionality of the feature space we perform *Principal Component Analysis* (PCA). The goal of dimensionality reduction is to find a linear sub-space of the feature space that has a reduced dimensionality.

This is done by procuring the first M' different principal components or eigen vectors ordered by their decreasing eigen values λ_i where, $i = 1, ..., M$. The sample covariance matrix $\bar{C}$ for centered random variable X is,

$$\bar{C} = \mathbb{E}(XX^T) \tag{1}$$

The covariance matrix $\bar{C}$ is a Hermitian matrix and it supports the fact that there exists M

real eigen values found from the eigen value decomposition of $\bar{C}$.

M' is chosen based on the tolerance of error, η calculated from these eigen values of the sample covariance matrix $\bar{C}$,

$$\frac{\sum_{i=1}^{M'} \lambda_i}{\sum_{i=1}^{M} \lambda_i} > \eta \tag{2}$$

Running PCA on X which initially has 504 feature components for 600 test samples gives us a reduced dimension of 139 components having a tolerance of, $\eta = 0.95$.

We can also view the dataset among the first 2 principal components in the figure below. Projecting the dataset on the first two significant eigenvectors allows us to get some clarity of the classification task ahead.

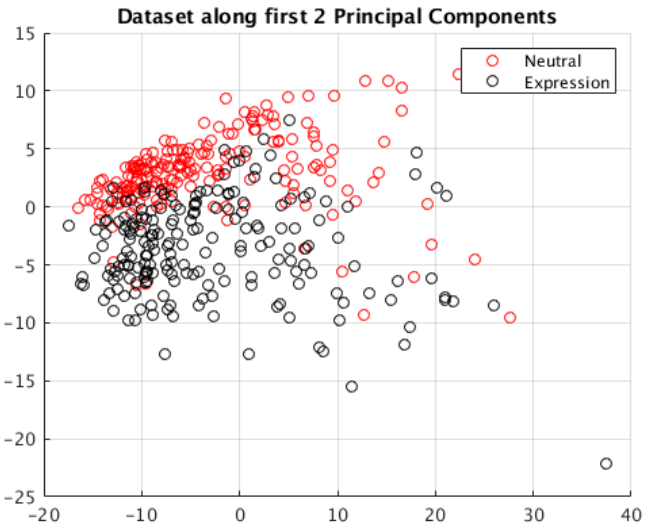


Figure 1: Projecting the data along 2 signinficant Principal components

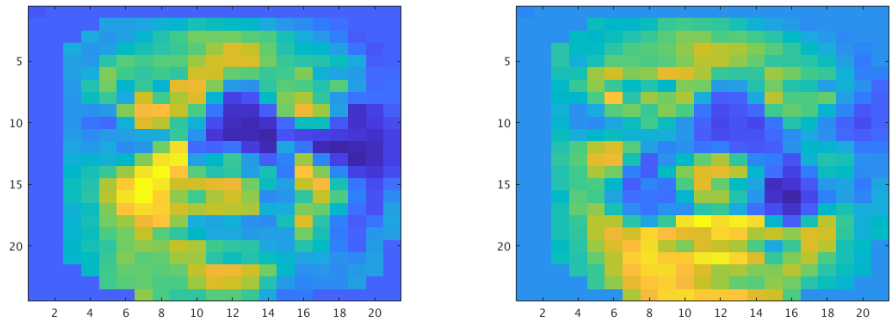
2 CLASSIFICATION

2.1 Bayes Classifier

The Bayes classifier was implemented by fitting the K class distribution to be a Multivariate Gaussian(MVN), i.e $\mathcal{N}(\mu_j, \Sigma_j)$ for j = 1....K. We find the class sample mean and variance as,

$$\mu_j = \frac{1}{N} \sum_{x \in w_j}^{N} \mathbf{x}_i \tag{3}$$

$$\Sigma_j = \frac{1}{N} \sum_{x \in w_j}^{N} (\mathbf{x}_i - \mu_j) \cdot (\mathbf{x}_i - \mu_j)^T \tag{4}$$



(a) Neutral **(b)** Expression

Figure 2: Sample mean of data points

The classification is done by evaluating the MVN pdf at training and testing data points. To avoid singularity in the evaluation of the determinant we bump the diagonal of the determinant by some positive constant $\alpha \cdot I$, where I is the identitiy matrix. Also to evaluate the inverse of the covariance matrix the *pseudo inverse* is chosen.

Bayes Classifier with Linear Discriminant Analysis(LDA)

LDA can be used as a tool for data reduction. LDA can also aid classification as projecting the dataset on a line learnt from the dataset increases the class separation and reduces the within class variance. This is ideal for Bayes classification as its a statistical inferencing technique. This abates the fitting MVN to a Gaussian distribution and reduces the computational overhead. It is observed that running Bayes classification with LDA improves the performance of classification in the course of simulation.

2.2 k-Nearest Neighbors

kNN classifies the test sample based on the voting evaluated by k of its nearest neighbors. The *closeness* is defined by a similarity measure and in this course of simulation the ℓ_2 norm or the Euclidean distance.

It is observed that as the value of k is increased the accuracy of performance increases but is bounded above by the *Bayes classifiers* accuracy shown in the experimental result section which reinforces the theoretical error bound.

2.3 Kernel SVM

Kernel SVM makes use of the Representer theorem to find the optimal θ and bias b in the RKHS. It was found that appending another feature to the dataset X doesn't solve the problem of finding the bias. The bias is calculated using the *average* of all the values of b found among non zero support vector from the constraint,

$$y_n(\theta^T \mathbf{x}_n + b) = 1 \tag{5}$$

The support vectors are found using MATLABs inbuild quadratic programming function *quadprog*. The regularization parameter C is found using cross-validation performed using *line search*.

During the course of simulation RBF and polynomial Kernel was chosen to compute the test and training accuracy. The parameter, $\gamma = 1/2\sigma^2$ for an RBF Kernel was chosen using the line search. Line search was performed by reducing γ in steps of 0.001 until a good training and testing accuracy was acheived. Intuitively γ controls the similarity of closeness chosen by the RBF Kernel. In other words lot of points will fall similar for a smaller γ or a larger σ^2.

For the polynomial Kernel, the bias of $c = 1$ was hard coded and the value of r was found using line search. Line search was performed by increasing r in steps of 2 and finding the optimal value which gave good training and testing error performance.

In the figure below we can see a non linear trend learnt from the RKHS separating the two classes.

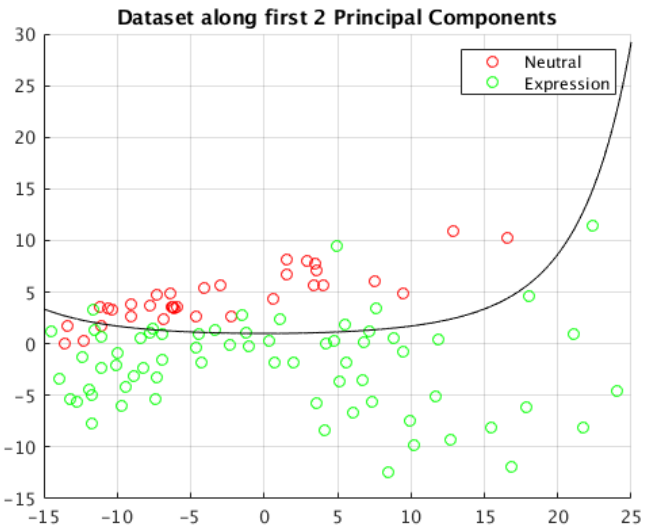


Figure 3: Projecting the data along 2 signinficant Principal components after Kernel SVM using RBF

2.4 Boosting Linear SVM

We perform classification by boosting the performance of weak Linear SVMs. We use adaboost to improve the performance of the linear SVMs. The initial weigths are taken to be $w_i = 1/N$ $\forall N$. The misclassification samples are learnt and weighted accordingly to compute the misclassification error P_j for the j^{th} iteration of adaboost. We then learn the parameter α_j to scale the weak classifier. This is done for T epochs. We incorporate the weights into the dual problem as,

$$\min \quad f(\alpha) = \frac{1}{2}\alpha^{\mathrm{T}} Q\alpha - M \cdot D^{\mathrm{T}}\alpha \tag{6}$$

$$s.t \quad y^{\mathrm{T}}\alpha = 0, \tag{7}$$

$$0 \leq \alpha_i \leq \frac{1}{M}, \ i = 1, ..., M. \tag{8}$$

where, D is the discrete distribution learnt from Adaboost. We can observe the improvement in the classifier performance aftere Adaboost.

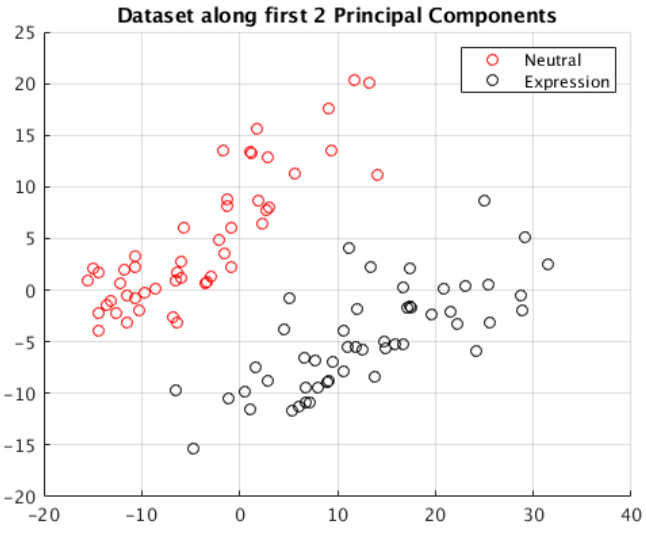


Figure 4: Projecting the data along 2 signinficant Principal components after boosting

3 EXPERIMENT RESULT

Classifier	**Test Accuracy**	**Train Accuracy**	**Script**
Bayes	84%	99.33%	bayesTest.m
kNN(K = 8)	84%	NA	kNNTest.m
kNN + LDA(K=8)	88%	NA	kNNTest.m
Kernel SVM, RBF: $\gamma = 0.0004$, C=3.9	83%	88.33%	kernelSVMTest.m
Kernel SVM, RBF: $r = 1$, C=2	83%	82%	kernelSVMPTest.m
Adaboost(C = 1.5)	99%	98.67%	adaboostTest.m