ISEN 613: Engineering Data Analysis

Fall 2019 – Course Project

Image Classification on CIFAR 10 Dataset

TEXAS A&M UNIVERSITY

Group:

Jathin Dhulipalla (527002408)

Pooja Chourey (527002563)

Sai Siva Ganesan (627006061)

Soma Sunder Gutala (927006092)

Contents

[1. Technical Report: Linear kernel SVM (SVC) 2](#_Toc26065111)

[2. Quadratic Discriminant Analysis (QDA) 4](#_Toc26065112)

[3. Technical Report: Boosting 6](#_Toc26065113)

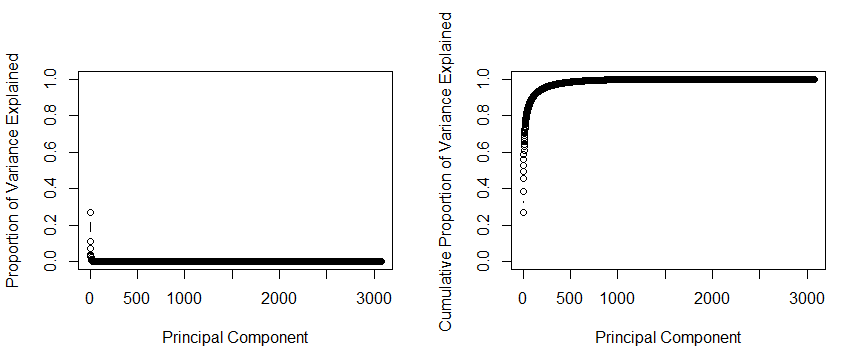
[4. Choosing the Best Model: QDA 8](#_Toc26065114)

# Technical Report: Linear kernel SVM (SVC)

**Data Preprocessing:**

The aim of building this model was to predict the object in each image provided in the data set. We have been given a training data set of 50,000 images with RGB pixel values which directly correspond to 3072 (1024 X 3) features. Since the computation time required to train a model across all the given features is extremely long, we decided to use the **Principal Component Analysis** dimensionality reduction technique with scaling set to ‘TRUE’ to work with lesser number of features.

In order to choose the number of principal components, we fixed a threshold variance of 95%. To explain 95% of variance in the data, we ended up choosing the first 220 principal components. The same has been illustrated by the scree plot and cumulative PVE (proportion of variance explained) plot.



With 220 features and respective training labels available for each observation, a support vector machine with a linear kernel function (support vector classifier) was fit to 80% of shuffled data. An interesting point to note here is that the linear kernel SVM seemed to perform much better than other nonlinear kernel functions like radial and polynomial SVMs. Training data corresponded to 40,000 observations in the given dataset. We are using the validation set approach to assess the model performance. 20% of data is used as a holdout set that the model will make predictions on.

**Training the model**

Since SVC is a soft margin classifier and margin violations are kept possible to provide robustness to the model, the cost parameter had to be set while training the model. The cost value acts like a budget for the magnitude of violations that will be acceptable by the model. We decided to go with a cost parameter value of 0.1. This particular cost value yielded better results compared to other values in our testing. A seed value is set so that the model results are reproducible.

> set.seed(10)

> svmlin.fit = svm(as.factor(train\_Y)~., data=train\_PCA.df, kernel='linear', cost=0.1)

> summary(svmlin.fit)

Taking a look at the summary of our fit linear SVM model, we see a total of 35,583 support vectors from all 10 classes. We now run the fit model on our training data to check the training accuracy.

> svmlin.train.predict = predict(svmlin.fit, train\_PCA.df)



From the table, we can see that we get a training accuracy of **44.92%** from linear SVM. The model has low variance but a relatively higher bias.

**Validating the model**

Now running the fit model on validation set:

> svmlin.val.predict = predict(svmlin.fit, test\_PCA.df)  


We end up with a validation set accuracy of **41.40%** from our model.

# Quadratic Discriminant Analysis (QDA)

**Data Preprocessing:**

For Quadratic Discriminant Analysis, data preprocessing was done in a similar way as it was done for the linear SVM model. Since the number of features in our dataset is 3072, we performed principal component analysis to identify the features that explain the maximum proportion of variance in data. We found that 95% of the variance in data could be explained by 220 features. These features were used to train our model using quadratic discriminant analysis and then further to validate the model. Following the validation set approach, the data was divided into an 80-20 ratio to train and validate, respectively. There was a total of 40,000 rows in our training data and 10,000 rows in the validation data. Since the labels are readily available for this data, it was convenient to check the prediction accuracy of the model on our validation set.

We preferred QDA for this dataset because the training data is large and as compared to some other models that we tried, QDA seemed to perform well in terms of computational speed as well as accuracy.

**Training the model**

After performing the principal component analysis and split on data, the QDA model was trained. There are no hyper parameters to tune when it comes to training a QDA classifier. It is important to note that the QDA classifier assumes observations from each class take the form of Gaussian distribution and each class has its own co-variance matrix. Hence the resulting classifier boundary is a much more flexible quadratic decision boundary compared to an LDA classifier. A seed value is also being set before training so that the results from our model are reproducible.

> set.seed(10)

> qda.fit = qda(as.factor(train\_Y)~., data=train\_PCA.df)

The training accuracy of the model was obtained as:

> qda.train.pred = predict(qda.fit, train\_PCA.df)



We got a training accuracy of **68.19%** from QDA.

**Validating the model**

After performing fit on the training data, the model was run on the other 20% data - validation set. The validation set accuracy that we got was **50.69%**

> qda.val.pred = predict(qda.fit, newdata=test\_PCA.df)



These results were significantly better than the results we achieved with an LDA classifier. Hence this turned out to be one of our top 3 models on the CIFAR 10 dataset.

# Technical Report: Boosting

**Data Preprocessing:**

For Boosting Analysis, data preprocessing has been done using Principal Component Analysis. This is similar to what has been done for the above two methods. The total number of features in the data is 3072 which causes problems for computing the models and associated predictions. It was found that 95% of the variance in data could be explained by 220 features which were used to train the model. The complete data set has been split into training data that contains 80% of observations and a test set of data containing 20% of the observations. Using the given labels, test error has been used to compare this model with the other models.

After using random forests, boosting and bagging since they fall under the same family of methods, we found that boosting had the most potential.

**Training the model**

Boosting model has been trained after Principal Component Analysis of the data as described in the previous section. Boosting has three tuning parameters that makes this assessment one of the most time consuming. These parameters are the number of trees, shrinkage parameter and the depth of trees grown. For the depth, we take 4 as the value since shorter trees have better results as boosting grows the trees based on the information from previous trees. For the number of trees, we take a value of 2000. We consider the tuning of overall model mainly by focusing on the tuning parameter. Since the number of trees is not being changed, the shrinkage parameter is picked by evaluating test errors for the values of 0.001, 0.005, 0.01, 0.05, 0.1 and 0.5. We found that 0.005 gives a good amount of accuracy compared to other shrinkage parameters.

After this, we finally train the model with the chosen parameters using a multinomial distribution since we require classification on more than two classes. We then calculate the training and test errors.

> boost.fit

gbm(formula = as.factor(train\_Y) ~ ., distribution = "multinomial",

data = train\_PCA.df, n.trees = 4000, interaction.depth = 4,

shrinkage = 0.005)

A gradient boosted model with multinomial loss function.

4000 iterations were performed.

There were 220 predictors of which 220 had non-zero influence.

The accuracy on training set has been found to be **60.0825%.**

In both instances of predicting on training and validation sets, we have used the label that came out with the maximum probability.

> boost.train.predprob = predict(boost.fit,newdata=train\_PCA.df,type="response", n.trees=4000)

> boost.train.pred = apply(boost.train.predprob, 1, which.max)



**Validating the model**

After performing fit on the training data, the model was run on the other 20% data - validation set. The validation set accuracy that we got was **45.27%.** Also, note that the predictions were bases on which label had he highest probability.

> boost.val.predprob = predict(boost.fit,newdata = test\_PCA.df,type="response",n.trees=2000)

> boost.val.pred = apply(boost.val.predprob, 1, which.max)



The training and more importantly the validation error turned out to be one of the best on the CIFAR 10 dataset using boosting.

# Choosing the Best Model: QDA

After considerable analysis as outlined in the previous sections, the following table has been compiled comprising of the training and test accuracies for each of the models. Note that these accuracies are achieved after tuning the models.



The primary methodology that has been used in selecting and training the best model is validation set approach. We have calculated both test and training accuracies for multiple models and the above three have been found to be performing the best for the given dataset. We have used Principal Component Analysis (PCA) for preprocessing before each of the models since the number of features in the given dataset is overwhelming to perform the computations.

Finally, after deciding the required model, we have used PCA once again to train the model on the entire dataset as this would give more data for the model to be trained. Since we would be using this model on data that is completely new, there is no concern that the data is going to cause an underestimate of errors on the new data.

We cannot do much than speculate on how QDA turned out to be the best models among all the classification models we have tried out like Linear Discriminant Analysis, KNN, Random Forests among others. We found that the least test error was obtained for boosting, linear SVM and Quadratic Discriminant Analysis. However, it is important to note that boosting is extremely powerful and showed great potential. If we could have more computations, perhaps we could have achieved higher accuracy. This observation is being made since we noticed a trend of increasing accuracy while increasing the number of trees and reducing the shrinkage parameter. Boosting as a method learns slowly to make better predictions. Based on our literature review, we found that image classification works best when using machine learning models that learn slowly like neural networks. Boosting could come close to perhaps mimic such a learning pattern and shows promise if provided enough resources.

For now, based on the given conditions, data and constraints, we have found Quadratic Discriminant Analysis to be the model to go forward with.