STAT4249 HW3: Team 9

Alexandria Seybold (aes2236) - Thomas Darricau (td2413)
Daniel Rieman(djr2144) - Drew Limm (dtl2119)

Introduction

Given a data set extracted from the census bureau database, our job was to predict whether or not a person makes over $50K a year. The data set contains the following variables: age, workclass, fnlwgt, education, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country, and a binary value indicating if the subject made over $50K a year. We separated the data set, using the first half as training data, and the second half to test. We each explored different classification and regression methods, but kept our data manipulation and dummy variable construction consistent.

Data Loading and Manipulation

We first wrote a script to manipulate the data into a manageable form. This involved loading the data from the provided URL and then splitting the data into two groups, test and train. Each group held half of the data. Next we converted categorical data into matrices of dummy variables so that we could run a selection of classification techniques.

Data Set:	Train	Test
Rows:	16281	16280
Columns:	15	15
Number Positive Class:	3897	3944
Number Negative Class:	12384	12336

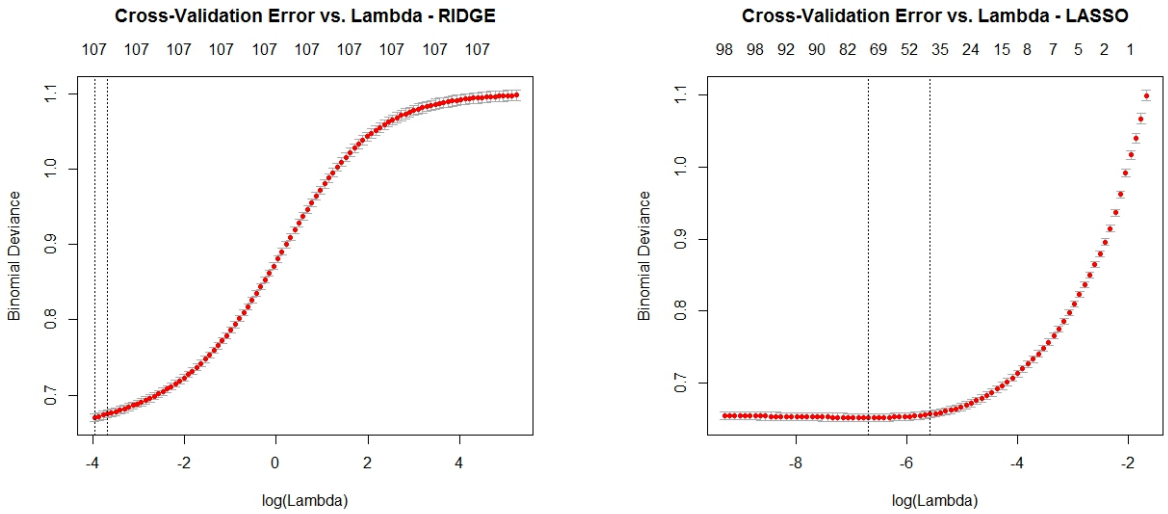
Prediction Methods and Results

Ridge and Lasso

With higher dimensional datasets, the need to prevent overfitting of data becomes more critical. Some variables are more meaningful than others, and to determine their importance, we must impose some sort of penalty to our model. Both the Ridge and Lasso are linear regressions that focus on regularization and shrinkage.

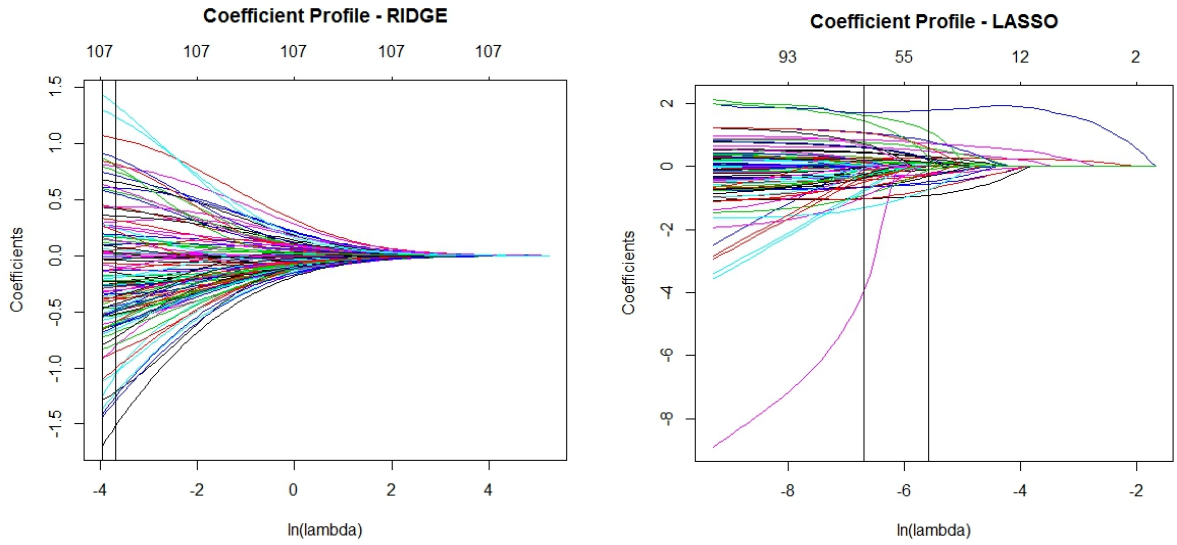
Both the Ridge and the Lasso perform reduction in dimension, however the difference between the two lies in the penalty they impose. The Lasso uses a L1 penalty regression model, while the Ridge uses the L2 in order to shrink the coefficients. We understand that the Ridge cannot zero out the coefficients, and therefore the Coefficient matrix will include a value for each

variable. However, with a large number of variables, it is not necessarily obvious which method will perform better. We will use the *glmnet* package in R to see which regression performs better. The Elastic Net combines the Ridge and Lasso penalties together by implementing a continuous alpha parameter from 0-1. To directly compare the two, we use strictly α = 0 (complete ridge) and α=1 (complete lasso). After doing cross validation, we can compare the coefficient profiles and see how well each model predicted our test data.



Figures 1 and 2: Cross Validation curves using glmnet for Ridge and Lasso

For a given complexity parameter, lambda, the cross-validation error is denoted by a red dot. The numbers on top show how many variables were used at that point. Notice that for the ridge regression, there are always 107 parameters because it does not zero out the coefficients. For each plot, the dotted vertical lines on the left shows the value of λ that minimizes error, and the line on the right shows the largest value of λ that is still within a standard error of the minimum.



Figures 3 and 4: Coefficient profile plots for ridge (left) and lasso (right) using glmnet.

The above figures show the different coefficient values for a certain λ. Again, the vertical line on the left is the λ where error is minimized, while the right vertical line is the largest λ still within a standard error of the minimum.

For comparison, we made sure to choose the λ where the error is minimized before obtaining our coefficients. After applying the coefficient matrices to the appropriate test input matrix, we got a predicted output. Since -1 corresponds to <=50K and 1 corresponds to >50K, the output was classified by whether it was positive or negative. A contingency matrix was constructed to calculate the prediction error for each regression:

Linear Regression Model:	Ridge	Lasso
Number of Predictors:	107	75
Test Misclassification Rate:	14.78501%	14.82187%

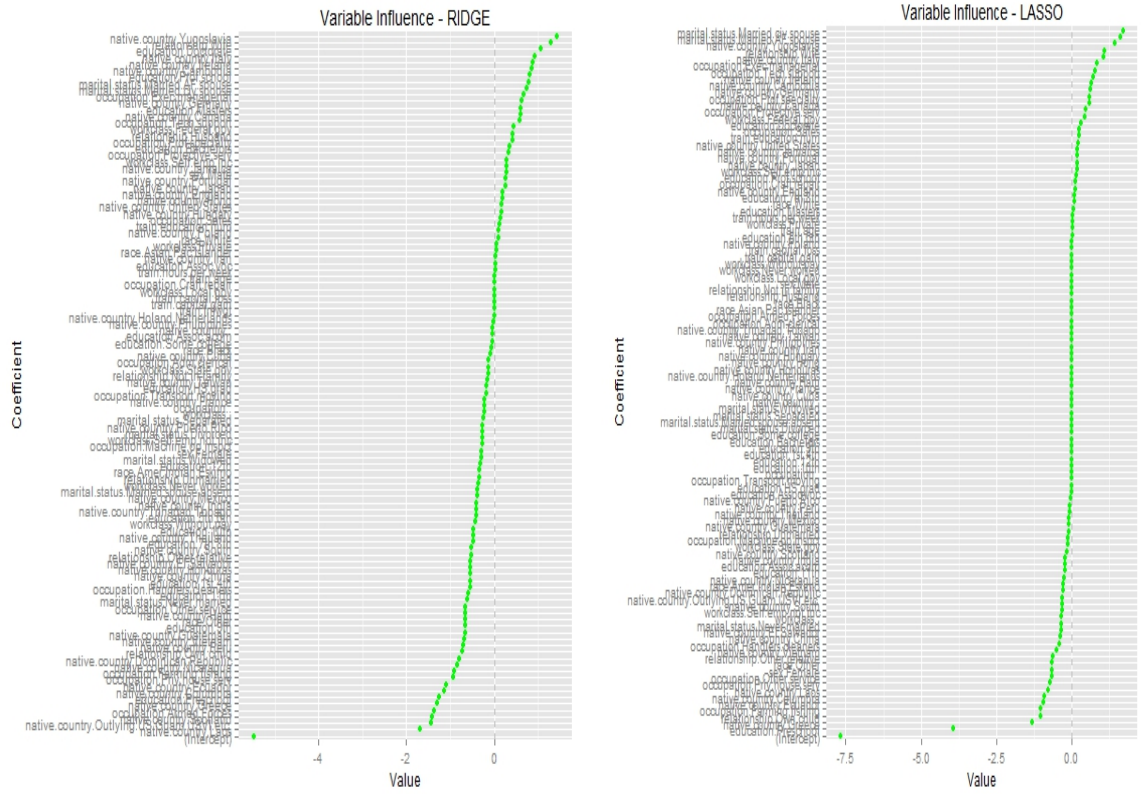


Figure 5 and 6: Influence of each indicator on income level

Again, with high-dimensional datasets, variable selection is critical. Both the Ridge and Lasso are regularization regressions, and attempt to shrink coefficients. Figures 5 and 6 value each indicator variable by how much of an influence it was on predicting the desired output. Although the plot is difficult to see, by expanding them we can see that the variables on the top are the highest indicators of having an income of greater than 50K.

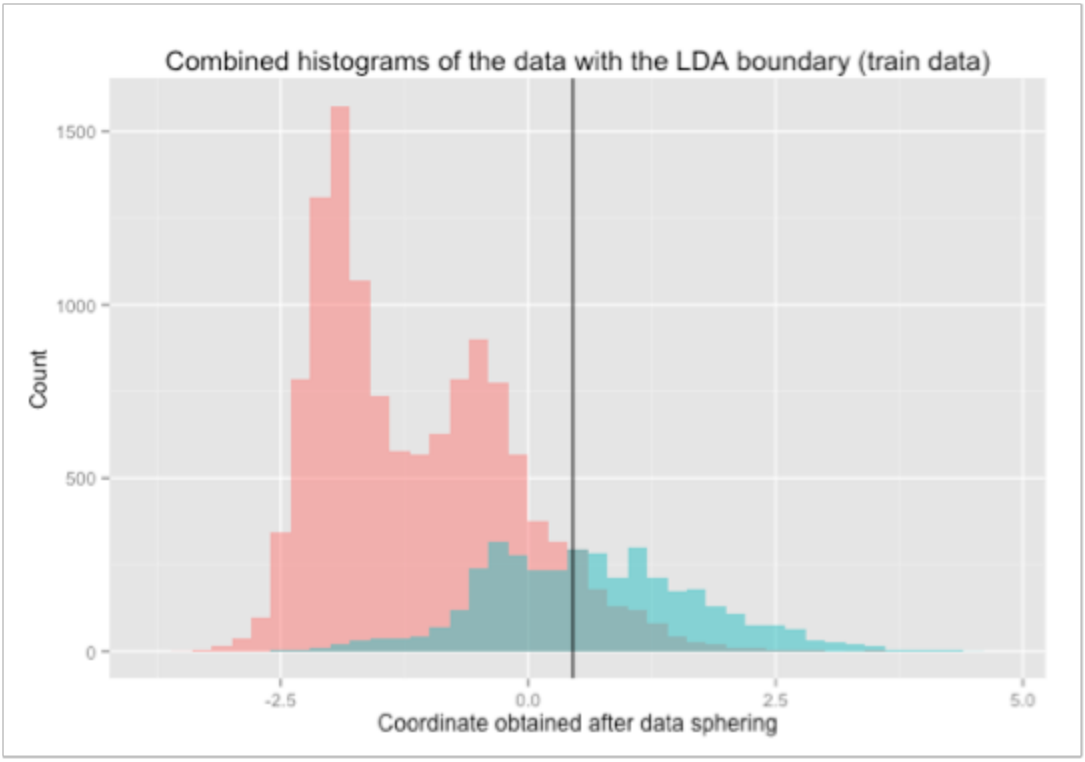
Naïve Bayes Classifier

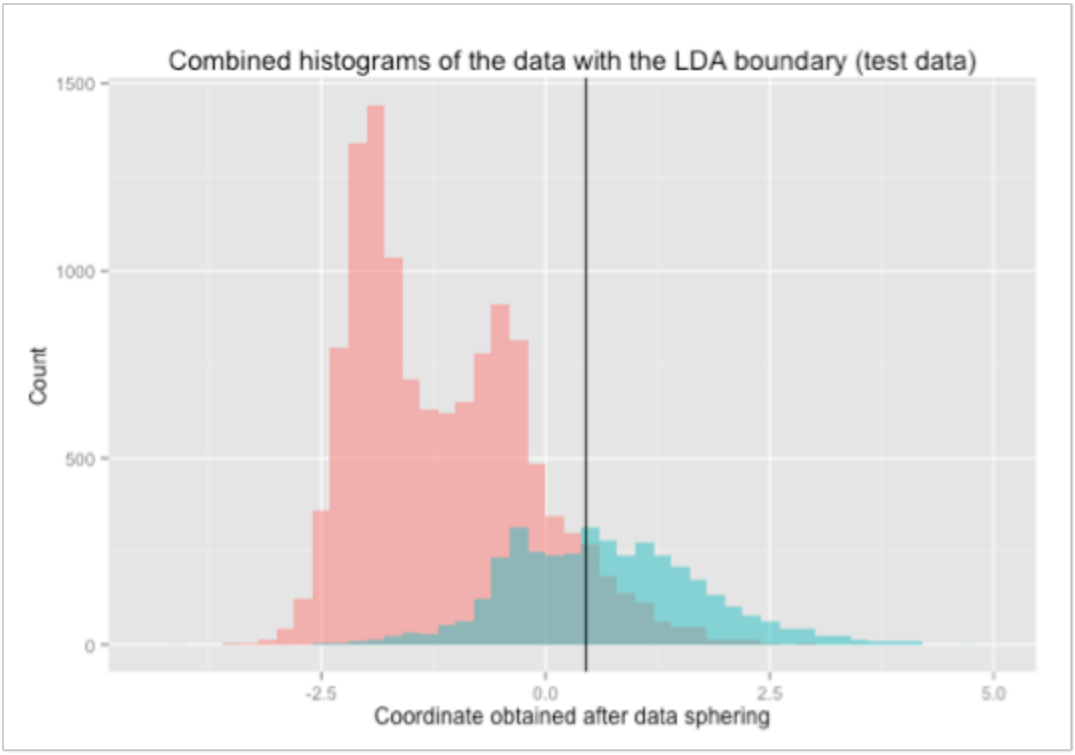
A logical choice as a classifier for our data was the Naïve Bayes method. Handling easily categorical data in R, this simple method often achieves good results.
As said, it is very easy to use in R even when the independent variables are categorical with the naiveBayes function from the “e1071” package. Its implementation was not an issue and we obtained a misclassification rate of **16.93%**, close to the rate the writers of the paper obtained.

Linear Discriminant Analysis

Once all the dummy variables created, we could run another linear classifier with the LDA method, where we assume that the class-conditional densities are Gaussian with a common

covariance matrix. When there are only two different classes as in here, this method only differs from the linear regression method in its intercept. But since we make probabilistic assumptions here unlike in the linear regression, we expect to have a better result here.
The misclassification rate using LDA was **15.8%**.
By sphering the data, we obtain a graphic visualization of our data on which we may see the classification boundary obtained with this method.
We observe that for the training data, the boundary is located at the point where the two classes are at approximately the same count, providing the optimality for our boundary.
On the test data, the same boundary is obviously not optimal but still provides a good misclassification rate (15.8%).





Linear OLS Regression

Issues encountered: Although we removed the first column in each dummy matrix before running any linear regressions, we still found some of the matrices to not be full rank. This resulted in the exclusion of some columns but will not affect the predictive power of the model.

Models: We ran a full model with all data included then ran the forward and backward stepwise and forward stagewise regression model selection algorithms. All three returned the same model so we chose to report only one model, the forward stagewise. Within the model selection process we restricted the algorithms to only add a whole dummy matrix or not add it at all – we didn't want an arbitrary column of the matrix appearing in the final model. The final forward stagewise model included the predictors relationship, education level, capital gain, occupation, capital loss, age, work class, and marital status.

Since we used income as the response variable with the positive class defined as income >50K and the negative class <=50K, we decided on the following classification rule:
class = sign(y.prediction). This gave the following results:

Linear Regression Model:	Full Model	Forward Stagewise Model
Number of Predictors:	95	54
Training Misclassification Rate:	15.90197%	15.95725%
Test Misclassification Rate:	16.00737%	16.09337%

We recommend the stagewise selected model over the full model because it is computationally less complex and achieves similar misclassification rates.

Fitted Hyperplane using Support Vector Machine (SVM)

We used the best.svm() function with 10-fold cross validation from R's e1071 package to fit the SVM with soft margin. With the entire training data set included we got following misclassification rates.

Method:	SVM with Soft Margin
Number of Support Vectors:	13283
Train Misclassification Rate:	10.12899%
Test Misclassification Rate:	25.13514%

K-Nearest Neighbors, K-Nearest Neighbors Cross-Validation

We used the knn() function in R's class package to classify the test data.
We also used the knn.cv() function

Error Rates:	K = 1	K = 2	K = 3	K = 5	K = 10	K = 15	K = 20
KNN:	28.403%	29.687%	26.161%	25.178%	23.888%	23.440%	23.587%
KNN-CV:	27.769%	28.507%	26.345%	25.147%	23.857%	23.102%	23.213%

Random Forest

We used R's randomForest package to make a random forest classifier. We also ran randomForest() with settings for regression, but the error rate was huge (greater than 80%).

Class:	<=50K	>50K
Error Rate:	6.0642765%	42.5455479%

Overall Test Misclassification Rate: 14.4226%

Conclusion

We found the Random Forest method to have the lowest test set misclassification rate. Below is a summary of all methods and misclassification rates studied.

Method	**Test Misclassification Rate**
Ridge Regression	14.79%
Lasso Regression	14.82%
Naïve Bayes	16.93%
Linear Discriminant Analysis	15.8%
OLS Full Model	16.01%
OLS Forward Stagewise	16.09%
SVM with Soft Margin	25.14%
KNN	23.44%
KNN with Cross-Validation	23.102%
Random Forest	14.42%