Authors

Our group identifier is P2.

Introduction

The Problem

Humans make many eye movements every second. The two most common types of these are fixations and saccades. Fixations are periods of relative oculomotor stability that center the high-resolution component of the retinal surface (the fovea) over targets of interest. Saccades are fast motions used to re-center the fovea over new targets of interest between fixations. The planning and deployment of these movements is driven by a wide range of "bottom-up" (roughly, "internal") and "top-down" ("external" or "environmental") factors. The placement of the human oculomotor-system at the intersection of these forces makes the study of eye movement an excellent window into the cognitive and attentional state of the observer during a particular task.

Recently, sharp reductions in the cost of eye movement tracking equipment (typically infrared camera-based systems) have pushed the study of eye movement behavior into fields beyond psychology and neuroscience. The rising temporal precision (sampling rate) of many such systems also enables the user to collect gaze position data at a rate of thousands of samples per second (Hz). This not only significantly refines the spatio-temporal profile of recorded eye movement behavior, but also allows eye tracking systems to drive the behavior of hardware and software in real time.

One likely outcome of the intersection of these developments could be the incorporation of eye movement information in the behavior of user interfaces. For example, consider the potential advantages of monitors which could track a user's eye movements, detect whether or not they were reading, and adapt their display characteristics in real time to reduce eye strain associated with reading text on a backlit surface. Our objective for this project was to develop an eye movement data based classifier–specifically, a binomial logistic regression model–that could serve a similar function by distinguishing between two different patterns of user behavior (reading text and following a randomly moving target).

The Data

The data for this project are taken from the BioEye 2015 "Competition on Biometrics via Eye Movements" hosted by the University of Texas[1]. The goal of this competition differed from ours – participants were asked to develop biometric algorithms for identifying individual users based on their eye movement behavior across tasks. For the competition data set, 306 subjects completed two trials for each of two tasks: a reading task and a random target pursuit task. The data for 153 of these subjects were assigned to a "development set"; the rest were assigned to a "test set". These groups of subject data did not differ in any way, but could be used to separately develop and evaluate the performance of algorithms if the contest participant so chose.

We chose a subset of the data supplied for competitors and slightly adapted it to make it more suitable to our purposes. We used only the "development" data set, which consisted of 612 data files (two tasks, two trials per task, 153 subjects – 153*2*2=612 files) containing eye position records. This data was collected at 1000hz for each subject/trial, then down-sampled to a rate of 250hz in post processing, yielding between 15000 and 25000 eye position recordings per file (the free-viewing task was slightly shorter than the object-pursuit task). Each position record contained a time stamp,the horizontal and vertical deviation of the subject's gaze from the center of the display (measured in degrees of visual angle), and a "quality flag" indicating the system's confidence in the validity of the recorded gaze point.

"Non-Technical Methods Introduction"

A large amount of pre-processing was required to make the data raw data suitable for analysis. These steps are described in detail in the methods section. The output of the data pre-processing procedure was 306 records – these were difference scores calculated on summary statistics associated with the fixations in each subject's pair of trials for a particular task. As an example, row "201" of the resultant data set could contain data from subject "25" for their two reading task trials. The values in this row are produced by subtracting the values of the predictor

variables associated with trial 1 from those associated with trial 2 for their reading task trials. Although the use of such aggregate scores represents a substantial reduction in the size of the usable data set, it was necessary to avoid violations of the independence of observations for logistic regression models produced by the design of the experiments. The dependent variable for all analyses was a binary variable indicating whether a subject was performing a reading task or a random target pursuit task.

Once the data had been pre-processed, a binary logistic regression was fitted to the entire data set using all available predictor variables in order to check for the presence of outliers/influential observations. After these points were eliminated, the remaining data set was divided into two subsets: a "training" set containing 80% of the available data and a "test" set containing the rest.

Next, a two-stage model fitting process was used to evaluate the performance of the full model and to assess whether or not a subset of the available predictor variables might improve the model fit and minimize multi-collinearity between the predictor variables under the condition of a shrinkage penalty. In the first stage, the full model was fitted to the "training" set of data. Its performance was assessed in terms of the area under the curve (AUC) value associated with the receiver operating curve (ROC) for the full model's classification performance on the "test" set of data. An Adaptive LASSO (ALASSO) version of the binomial regression model was then fitted to the training data using the same set of predictors to evaluate whether any of the predictor values could be selected out of the model design. Its classification performance was also evaluated in terms of the AUC. The AIC and AUC values for the full and LASSO models were then compared, and the winning model was selected.

Finally, the model-fitting process was wrapped in a 800-fold cross-validation procedure. The distributions of AIC and AUC for each model type across all iterations of the cross-validation procedure were then plotted and compared in order to make a final selection of the model.

Methods

Data Preprocessing

The scale of the raw data available for this project is tremendous – with an average of 20000 points of gaze position data available for each trial for every subject, a data frame containing all of the raw gaze position data would span roughly 24,480,000 rows by 4 columns, or nearly 100 million points total.

Because it is technically challenging to manipulate raw data at this scale, and because individual gaze points are considered relatively noisy measurements–even when collected using very robust eye tracking systems under controlled experimental conditions–raw gaze position data is typically collapsed or summarized into records of fixations and saccades. Our first objective was therefore to collapse our raw eye position data into a much smaller set of eye movement records. This was accomplished in R using a package called *Saccades*[2].

Although *Saccades* can identify both fixations and saccades in raw eye position data, we examined only the fixations. This is because computational methods for identifying eye movements in raw gaze position data are still not sufficiently powerful to robustly discriminate between fixations, saccades, and a third eye movement type called smooth-pursuits. The nature of smooth-pursuits and the specific reasons for this difficulty are beyond the scope of this project, but as a result, most eye movement detection algorithms treat all eye position recordings as parts of either fixations or saccades. In other words, when raw gaze position data is converted into eye movement data of a particular type, all the remaining data is assumed to be associated with movements of the other type. Saccades and fixations therefore become perfectly, negatively correlated in time – when one is happening, the other is not. Selecting one eye movement type and focusing on that subset of the data is a reasonable way to get around this problem.

Once the fixation detection algorithm had identified fixations in each subject's trial data, a second algorithim–also from *Saccades*–was applied to further compress our data by summarizing fixations for every trial for all subjects. It returned the following aggregate fixation behavior variables:

1. Number of fixations in the trial.
2. Mean duration of fixations during the trial (in milliseconds).
3. Mean horizontal dispersion of fixations during the trial (in pixels).
4. Mean vertical dispersion of fixations (in pixels).

5. Peak horizontal velocity within recorded fixations.

6. Peak vertical velocity within recorded fixations.

Aggregating the data in this way is necessary because it eliminates a second potential violation of the independence of observations assumption for logistic regression in our data. This would result because eye movement records within a trial constitute a time-series – any two fixations adjacent to one-another in time are likely to be more similar to one another than any two non-adjacent fixations. With the exception of the "number of fixations", these variables therefore served as the basis of those subsequently used as predictors in all of our models. "Number of fixations" was excluded because the others are mean scores derived partially from the count, and so are highly correlated with it.

Because the design of the experiment involved a test-retest procedure, individuals' performance on all trials was also likely to also be highly correlated. We addressed this last potential violation of the independence of observations assumption by making the final versions of the variables for our analyses mean-centered difference scores calculated between individuals' two trials on each task. This resulted in the following set of variables:

1. Mean centered difference in mean duration of fixations between trials within each task (X1 in model formulae).

2. ... difference in mean horizontal dispersion between trials ... (... X2 in model formulae)

3. ... difference in mean vertical dispersion between trials ... (... X3 in model formulae)

4. ... difference in mean horizontal velocity within recorded fixations between trials ... (... X4 in model formulae)

5. ... difference in mean vertical velocity within recorded fixations between trials ... (... X5 in model formulae)

Taking a difference score between the two trials within each task eliminates the violation of the independence of observations assumption for logistic regression associated with the experiment's test-rest design. The decision to mean-center the data was made in order to facilitate subsequent model fitting in R.

Initial Model Fit & Outlier Detection/Removal

After the data had been pre-processed in this way, it was checked for outliers by fitting a binomial logistic regression model to the data set. This fitted model took the following form:

$$condition = 0.09_{\text{intercept}} - 0.01X_{\text{X1}} - 0.18X_{\text{X2}} + 1.95_{\text{X3}} + 0.01_{\text{X4}} - 0.83_{\text{X5}}$$

The model was then re-applied to the data and the plots of the residuals against the fitted values, a QQplot, a scale-location plot, and Cook's D values were used to check for outliers. The results of these analyses are displayed in Figure 1.

As a result of these tests, rows 214 and 79 were removed from the data set as outliers/potential influential data points. The resultant data set was used for all subsequent analyses.

Model Fitting

Full Model Fitting

In order to fit and evaluate the performance of our classifier, we first split the data into training and test sets. The training set contained a random sample of 80% of our data; the test contained the remaining 20%. A binary logistic regression model containing the full set of predictor variables was then fitted to the training data set and the AIC value for the model was recorded.

Next, the classifier was applied to the test set of data and used to predict the task to which each observation belonged. The predictive accuracy of the model on the test data set was then evaluated using an ROC curve, the AUC for which was also recorded.

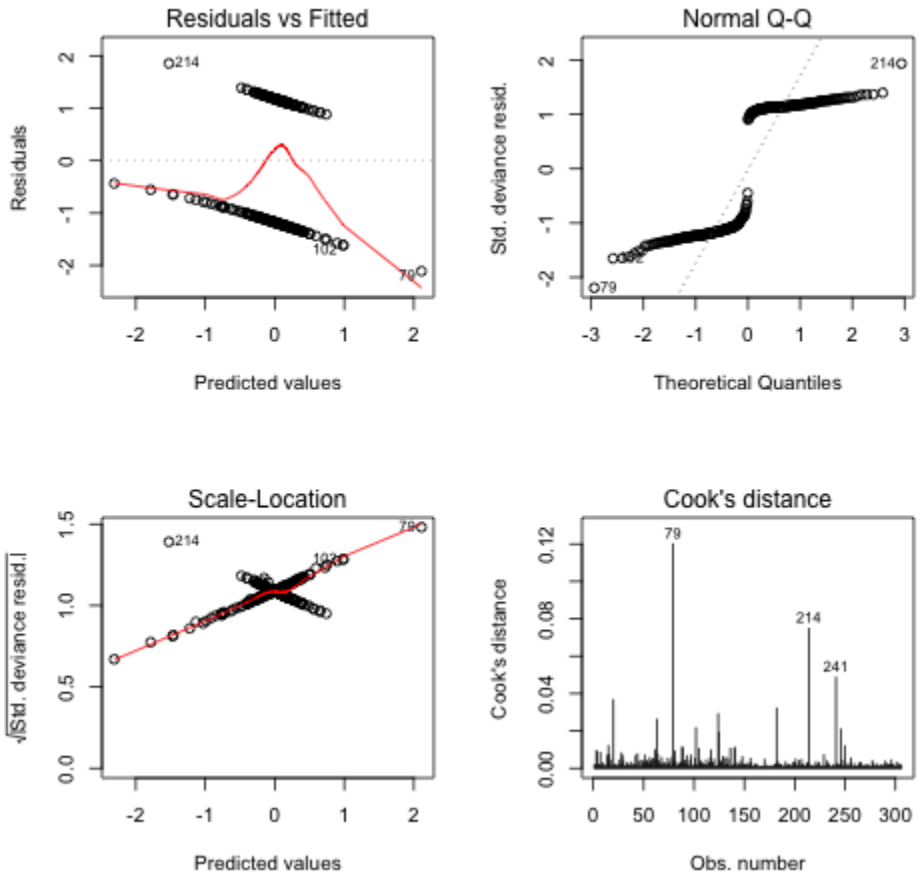


Figure 1: Outlier Plotting. Note that points 214 and 79 are likely outliers.

Lasso Model Fitting

As can be seen in Figure 2, the predictor variables for this data remained at least moderately correlated even after the data had been extensively cleaned and pre-processed. One option for reducing this degree of multi-collinearity is to use a shrinkage-based method such as the LASSO to reduce the size of the predictor space.

We used a modification of the "vanilla" LASSO called the "Adaptive LASSO" (ALASSO), the derivation of which is described is described in Wang and Leng (2006); code for implementing this method is available at the link included in reference 3. The chief advantage of this method over the "vanilla" LASSO is that it does not require the user to pre-specify a λ value; the optimum value for the shrinkage penalty is estimated using the "known variance version of BIC with full model MSE for estimating the error variance".

The non-zero terms in the ALASSO fit were used to refit a classifier on the same training and test data sets used for the full model fitting. The performance of this classifier was then tested in the same fashion as for the full model, and the resultant AIC and AUC values were recorded.

Model Performance Evaluation

Cross-validation

Finally, to assess the robustness and generalizability of the full and ALASSO models, an 800-fold "repeated random sub-sampling validation" cross-validation procedure was used to refit both the full and ALASSO models. Unlike "$k-fold$" or "$leave-p-out$" cross-validation methods, this approach uses the entirety of the available data set for generating training and test sets. Although this technique possesses undesirable Monte Carlo variability, it was chosen because we were concerned that $k-fold$ methods with even moderately sized ks would make the training and test set sizes too small, and because $leave-p-out$ methods were unlikely to be computationally tractable

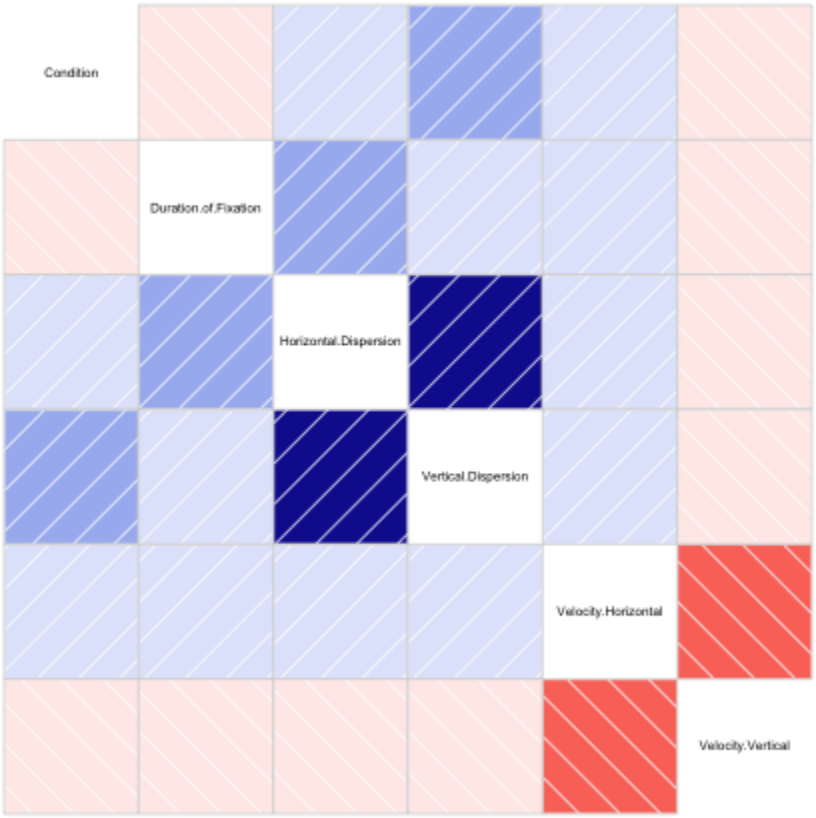


Figure 2: "Correlogram" of Predictor & Dependent Variables. Color values are scaled to represent the magnitude of the correlation (darker = greater magnitude) – blue are positive correlations, red are negative correlations.

with a data set of this size. Our cross-validation procedure was therefore performed with a very large number of iterations in the hope of obtaining an adequately sized slice of the possible combinations of training/test data sets.

During each iteration, the steps performed for our model fitting were repeated. Again, at the end of each iteration, the AIC and AUC values for both models were recorded. The resulting distributions of AIC and AUC values for the two models were then compared using paired-samples Student's t tests on the cross-validation generated data.

Results

The initial fitting of the full model on the pre-cross-validation training set took the following form:

$$condition = 0.23 - 0.02X_1 - 0.02X_2 + 2.63X_3 - 0.15X_4 - 0.59X_5$$

The AIC value for this model was 335.89; its AUC value was 0.69. The ALASSO procedure recommended the elimination of all predictor variables except for "mean centered difference in mean duration of fixations between trials" and "mean centered difference in mean vertical dispersion of fixations between trials". This model was refitted to the training data and took the following form:

$$condition = 0.23 - 0.03X_1 + 2.48X_3$$

The AIC value for the ALASSO subsetted model was 330.08; its AUC value was 0.68. Figure 3 shows superimposed ROC curves for the two models in this iteration of the model. Given the relatively small difference in AUC

values between the models, and the slightly reduced AIC value for the ALASSO refitted relative to the full model, the ALASSO refitted model appears to have turned out to be the better model, at least for this set of data.

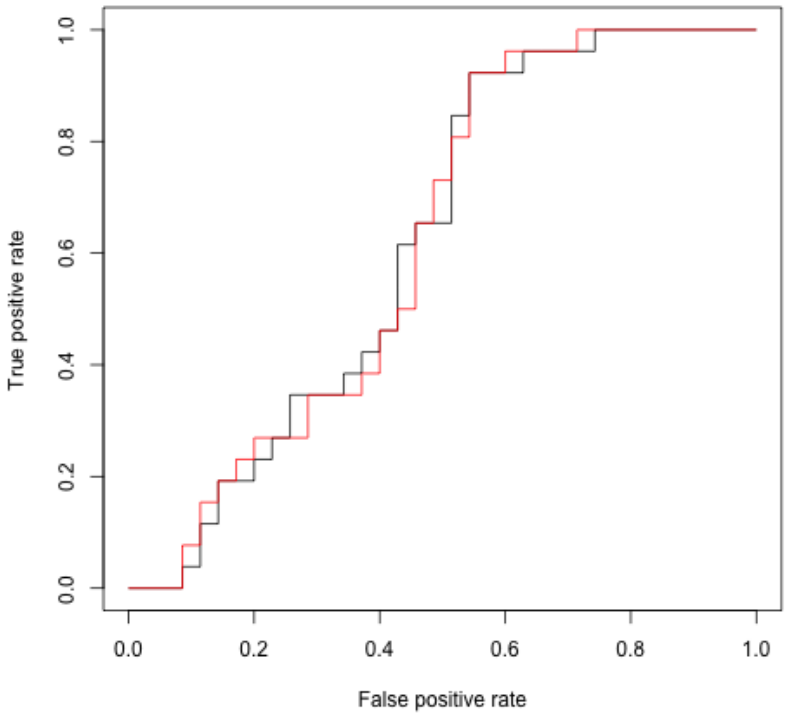


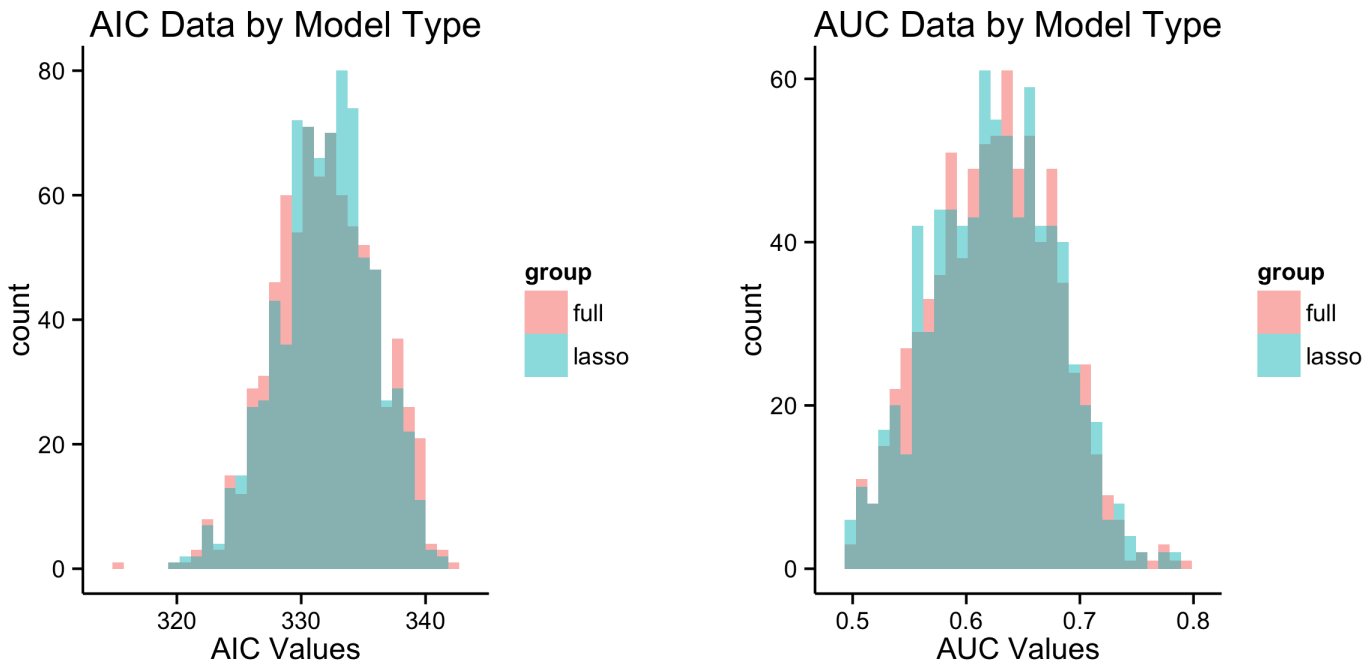
Figure 3: Comparison of the ROC curves for the full model (shown in red) and the ALASSO subsetted model (shown in Black).

Figure 4 presents superimposed histograms of the AIC and AUC values for both model types for the 800 iterations of the cross validation procedure. Although there is no clear graphical difference between the two distributions, paired t-tests indicate significant differences between the two models for both the cross-validation generated AIC values ($t_{1,799} = 148.52, p \leq 0.001$) and the AUC values ($t_{1,799} = 20.23, p \leq 0.001$). These tests were not directional, but a comparison of the man values for both distributions suggest that the ALASSO model had significantly lower AIC scores in the cross-validated data sets than the full model, but that the full model had slightly higher AUC values.

Discussion

The results of our analyses suggest that our classifier's performance is better than chance (65% correct classification, where 50% would represent chance performance), but perhaps not substantially so. Attempts to improve the model fit through subset selection methods generated models that were "better" models with respect to an information criterion value, but which also had slightly reduced predictive accuracy. The choice of one of these approaches over the other would therefore likely depend on an a-priori assessment of the costs/benefits associated with a tradeoff between "model quality" (broadly construed) and classification power.

Given that AIC scores do not assess model quality in an *absolute* sense, one possibility we must consider given these results is that much of predictive power for discriminating between tasks in this case lies in variables we did not assess. Preliminary results using a mixed-effect model version of this classifier (not included in this project) suggest that this variability may be attributable in individual differences which our methods would have failed to capture.



(a) AIC Scores

(b) AUC Values

Figure 4: Distributions of AIC & AUC Values by Group

What did you learn from this analysis?

The primary lesson we took from this analysis is that there is a significant penalty to be paid for the use of summary statistics to overcome violations of the independence of observations assumption. Although we were pleased we were able to reduce the scale of our data from nearly 100 million points to only several thousand, a substantial amount of power is lost in the process of compressing data to this extent.

What additional steps could be potentially performed to improve your analysis?

Had there been extra time and resources, as well as coverage of the relevant material in class, we would have used a hierarchical model based method instead of using summaries of data for the extent to which we were forced. By including a by-subject random effect term in our model, we could have used data summarized to the level of by-trial eye movement aggregate statistics, rather than being forced to use a between-trials performance change score.

It would also have been interesting to use the same set of procedures on either the raw eye movement data set containing information about saccades. The set of predictor variables associated with saccade data would have been slightly larger than for fixation data, access to which may have improved the predictive power of our models.

Finally, given that we did not refit the candidate models in every iteration of the cross-validation procedure, but rather before it, it is impossible for us to assess whether their performance in the context of the cross-validation method is not a product of Monte Carlo variability alone. That is, it is possible that a clearer or more consistent relationship may have emerged between the models' predictive power and their "quality" as assessed in terms of the AIC values if the parameters of the full model had been re-calculated during each iteration, then down-sampled using the ALASSO and refitted to the training data.

References

[1] BioEye 2015. (n.d.). Retrieved April 16, 2015, from https://bioeye.cs.txstate.edu/
[2] Malsburg, T. (n.d.). CRAN - Package saccades - Saccades. Retrieved April 16, 2015, from http://cran.r-project.org/web/packages/saccades/index.html
[3] Wang, H., & Leng, C. (2007). Unified LASSO Estimation by Least Squares Approximation. Journal of the American Statistical Association, 102(479), 1039–1048. http://doi.org/10.1198/016214507000000509
[4] Leng, C. (n.d.). Adaptive LASSO in R. Retrieved April 16, 2015, from http://tinyurl.com/kctun8k

Appendix

Part A: Data Pre-Processing & Cleaning Code

```
#Install needed libraries:
install.packages("CircStats")
install.packages("saccades")
library(CircStats)
library(saccades)
library(plyr)
library(dplyr)

# This is the path of the files in the system
# This value will change depending upon system to system.
#directory.dummy <- "/Users/mavezsinghdabas/StatisticsForBigData-/eyemovement_data/BioEye2015_DevSets

#Get all the names in the data folder
random_files_list<-list.files(random_files,full.names=TRUE)
reading_files_list<-list.files(reading_files,full.names=TRUE)

#Define a function to parse the name of text files of data and return subject/trial/condition informa
#from the filename.
#subject_information<-function(text_file){
#  split_name <- unlist(strsplit(text_file,"_"))
#  subject <- split_name[6]
#  trial_num <- substr(split_name[7],1,1)
#  subject_trial_info<-c(subject,trial_num)
#  return(subject_trial_info)
#}

#Define a function that will parse a table of eye movement position data and identify saccades.
detect.saccades <- function(samples, lambda, smooth.saccades) {
  # Calculate horizontal and vertical velocities:
  vx <- stats::filter(samples$x, -1:1/2)
  vy <- stats::filter(samples$y, -1:1/2)
  # We don't want NAs, as they make our life difficult later
  # on. Therefore, fill in missing values:
  vx[1] <- vx[2]
  vy[1] <- vy[2]
  vx[length(vx)] <- vx[length(vx)-1]
  vy[length(vy)] <- vy[length(vy)-1]
  msdx <- sqrt(median(vx**2, na.rm=T) - median(vx, na.rm=T)**2)
  msdy <- sqrt(median(vy**2, na.rm=T) - median(vy, na.rm=T)**2)
  radiusx <- msdx * lambda
```

```
  radiusy <- msdy * lambda
  sacc <- ((vx/radiusx)**2 + (vy/radiusy)**2) > 1
  if (smooth.saccades) {
    sacc <- stats::filter(sacc, rep(1/3, 3))
    sacc <- as.logical(round(sacc))
  }
  samples$saccade <- ifelse(is.na(sacc), F, sacc)
  samples$vx <- vx
  samples$vy <- vy
  samples
}

#Define a function that takes a text file as input and drops unnecessary columns, and renames
#columns of interest. Adds these values to new data frame and returns it.
data_load_and_cleanup<-function(text_file){
  data <- data.frame(read.table(text_file,header = TRUE,fill = TRUE))
  data <- data[c(-5,-6,-7,-8,-9,-10)]
  colnames(data) <- c("sample","x_degree","y_degree","validity")
  return(data)
}

#Define a formula to convert degrees of visual angle to pixels.
#This formula will return a value which is the number of pixels away
#from the center of the display, which will then need to be adjusted using the second formula in this
pixel_conversion_formula<-function(data){
  #Define variables needed for the calculation:
  d=55 #Distance between viewers and monitor in cm. Don't modify this.
  h=297 #Monitor height in cm. Don't modify this.
  r=1050 # Vertical resolution of the monitor. Don't modify this.
  #Calculate the number of pixels per degree of visual angle:
  deg_per_pix<-deg(atan2(.5*h,d)/(.5*r))
  #Calculate the size of data object in pixels instead of degrees
  size_in_pix<-data/deg_per_pix
  return(size_in_pix)
}

#Takes pairs of x & y pixel values and shifts their origins to the upper
#left hand corner of the monitor. Works by adding a constant to the pixel coordinate value for each d
convert_to_origin_formula_x <- function(x){
  original_horizontal_coordinate<-840 # The value of the pixel at the horizontal locaiton of the orig
  con_hori_crd<-x+original_horizontal_coordinate #Adds the re-centering adjustment to the horizontal
  return(con_hori_crd) # Generate a tuple of converted coordinate values.

}

convert_to_origin_formula_y <- function(y){
  original_vertical_coordinate<-525 # The value of the pixel at the vertical location of the original
  con_verti_crd<-y+original_vertical_coordinate #Adds the re-centering adjustment to the vertical val
  return(con_verti_crd) # Generate a tuple of converted coordinate values.

}
```

```
#A wrapper for the conversion and re-centering formulas using APPLY functions for speed
coordinate_transformation_origin_recentering<-function(data_frame){
  pixel_terms_x <- unlist(lapply(data_frame$x_degree,pixel_conversion_formula))
  pixel_terms_y <- unlist(lapply(data_frame$y_degree,pixel_conversion_formula))
  recentered_x <- unlist(lapply(pixel_terms_x,convert_to_origin_formula_x))
  recentered_y <- unlist(lapply(pixel_terms_y,convert_to_origin_formula_y))

  #Generate a "trial" column (a list of 1's given the design of our analyses to this point)
  trial<-unlist(rep(1,length(recentered_x)))
  trial<-as.factor(trial)

  output<-data.frame(cbind(data_frame$sample,recentered_x,recentered_y,trial))
  colnames(output)<-c("time","x","y","trial")
  return(output)
}

#A wrapper function that calls all functions defined to this point on a single text file and returns
preprocessing_wrapper<-function(text_file){
  #subject_trial_info<-subject_information(text_file)
  cleaned_data<-data_load_and_cleanup(text_file)
  cleaned_transformed_data<-coordinate_transformation_origin_recentering(cleaned_data)
  return(cleaned_transformed_data)
}

#Takes cleaned/pre-processed data and returns eye movement data sets
eyemovement_wrapper<-function(cleaned_transformed_data){
  fixation_data<-detect.fixations(cleaned_transformed_data)
  fixation_data_summary<-calculate.summary(fixation_data)
  eyemovement_means<-as.numeric((fixation_data_summary[,1]))
  eyemovement_sd<-as.numeric((fixation_data_summary[,2]))
  fixation_data_summary<-c(eyemovement_means,eyemovement_sd)
  #saccade_data<-detect.saccades(cleaned_transformed_data,lambda=15,smooth.saccades = FALSE)
  return(fixation_data_summary)
}

#Final wrapper; contains all functions previously defined -- intended to be passed to an APPLY functi
final_wrapper<-function(text_file){
  #subject_info<-subject_information(text_file)
  cleaned_transformed_data<-preprocessing_wrapper(text_file)
  eyemovement_data<-eyemovement_wrapper(cleaned_transformed_data)
  return(eyemovement_data)
}

####With these functions defined, generate the actual data for the experiment using a faster APPLY me
random_eyemovement_data_list<-lapply(random_files_list,final_wrapper)
random_eyemovement_output<-as.data.frame(do.call(rbind,random_eyemovement_data_list))
condition<-rep("random",length(random_eyemovement_output$V1))
random_eyemovement_output<-cbind(random_eyemovement_output,condition)
reading_eyemovement_data_list<-lapply(reading_files_list,final_wrapper)
reading_eyemovement_output<-as.data.frame(do.call(rbind,reading_eyemovement_data_list))
condition<-rep("reading",length(random_eyemovement_output$V1))
reading_eyemovement_output<-cbind(reading_eyemovement_output,condition)
```

```
#Cleanup junk variables:
random_eyemovement_output<-select(random_eyemovement_output,V3,V4,V5,V6,V7,V8,condition)
reading_eyemovement_output<-select(reading_eyemovement_output,V3,V4,V5,V6,V7,V8,condition)

####Generate factors for mixed model analysis:
#Trial data:
trial_num<-rep(seq(1:2),153)
#Subject id:
subj_id<-rep(seq(1:153),each=2)
random_eyemovement_output<-cbind(random_eyemovement_output,as.factor(subj_id),trial_num)
reading_eyemovement_output<-cbind(reading_eyemovement_output,as.factor(subj_id),trial_num)

####Cleanup variable names
names(random_eyemovement_output)<-c("num_fixations","mean_fix_dur","disp_horz","disp_vert","peak_vel_
names(reading_eyemovement_output)<-c("num_fixations","mean_fix_dur","disp_horz","disp_vert","peak_vel

####Merge data set and write-out:
full_eyemovement_set<-rbind(random_eyemovement_output,reading_eyemovement_output)
write.csv(full_eyemovement_set,"full_eyemovement_set.csv",row.names=FALSE)
```

Part B: Cross-validation & Model Performance Evaluation Code

```
Glm function on the full model
Function: glm
Input: The glm function will take the data frame and perform general linear model.
Output: The model after fitting the glm model using the
     family = binomial()
     data = eyeMovements_3
     Condition as Response variable
     Duration.of.Fixation as Predictable variable
     Horizontal.Dispersion as Predictable variable
     Vertical.Dispersion as Predictable variable
     Velocity.Vertical as Predictable variable
eyeMovements_3.GLM <- glm(Condition ~ Duration.of.Fixation + Horizontal.Dispersion +
              Vertical.Dispersion +
                Velocity.Horizontal +
               Velocity.Vertical,data = eyeMovements_3,
             family = binomial())
plot(eyeMovements_3.GLM)

Splitting the data into training set and test set
splitdata <- function(dataframe, seed=NULL) {
 if (!is.null(seed)) set.seed(seed)
 index <- 1:nrow(dataframe)
 trainindex <- sample(index, trunc(length(index)*.80))
 trainset <- dataframe[trainindex, ]
 testset <- dataframe[-trainindex, ]
 list(trainset=trainset,testset=testset)
}
wholeData4 <- splitdata(eyeMovements_4,123)
eyeTraining4 <- wholeData4$trainset
eyeTest4 <- wholeData4$testset
```

```
Fitting the glm to the training set
Function: glm
Input: The glm function will take the training dataframe and perform genaralized linear model.
Output: The model after fitting the glm model using the
    family = binomial()
     data = eyeTraining4
     Condition as Response variable
    Duration.of.Fixation as Predictable variable
     Horizontal.Dispersion as Predictable variable
    Vertical.Dispersion as Predictable variable
     Velocity.Vertical as Predictable variable
training.GLM <- glm(Condition ~ Duration.of.Fixation +
         Horizontal.Dispersion +
             Vertical.Dispersion +
             Velocity.Horizontal +
             Velocity.Vertical,data = eyeTraining4,family = binomial())
plot(training.GLM)

Predicting the values of the test set
predict_training_glm <- round(predict(training.GLM,eyeTest4,type = "response"))
predict_training_glm_continous <- predict(training.GLM,eyeTest4,type = "response")

Generate the confusion matrix
cMatrix <- confusion.matrix(eyeTest4$Condition,predict_training_glm,threshold = .95)
cAccuracy <- accuracy(eyeTest4$Condition,predict_training_glm,threshold = .95)
cAccuracy$AUC

Create ROCR prediction and performance objects
pred_training <- prediction(predict_training_glm_continous,eyeTest4$Condition)
performance_training <- performance(pred_training,"tpr","fpr")
x <- unlist(performance_training@x.values)
y <- unlist(performance_training@y.values)
plot(performance_training)
lines(x,y,col = "yellow")


Fit the lasso model for the logistic regression
source("LSA.R.txt")
training.GLM.LASSO <- lsa(training.GLM)
training.GLM.LASSO
training.GLM.LASSO$beta.aic

Fit glm after lasso
training.lasso.glm <- glm(Condition ~ Duration.of.Fixation+
               Vertical.Dispersion,
             data = eyeTraining4,family = binomial())
training.lasso.Summary <- summary(training.lasso.glm)

Predicting the values in the Test data
predict_lasso_training_glm <- round(predict(training.lasso.glm,eyeTest4,type = "response"))
predict_lasso_training_glm_continous <- predict(training.lasso.glm,eyeTest4,type = "response")

Generate the confusion matrix
```

```
cMatrixLasso <- confusion.matrix(eyeTest4$Condition,predict_lasso_training_glm,threshold = .95)
cAccuracyLasso <- accuracy(eyeTest4$Condition,predict_lasso_training_glm,threshold = .95)

Create ROCR prediction and performance objects on lasso model
pred_lasso_training <- prediction(predict_lasso_training_glm_continous,eyeTest4$Condition)
performance_lasso_training <- performance(pred_lasso_training,"tpr","fpr")
x.lasso <- unlist(performance_lasso_training@x.values)
y.lasso <- unlist(performance_lasso_training@y.values)
plot(performance_lasso_training)
lines(x,y,col = "red")


Cross validation function
Function: Converts the data frame into training set and the test set fit the glm model and extract th
Input: The entire Data frame
Output: The AIC and AUC values of different training and test data data sets for Full and Lasso fit m
for(i in 200:1000){
 splitdataWORK <- function(dataframe, seed=NULL){
  if (!is.null(seed)) set.seed(seed)
  index <- 1:nrow(dataframe)
  trainindex <- sample(index, trunc(length(index)*.80))
  trainset <- dataframe[trainindex, ]
  testset <- dataframe[-trainindex, ]
  list(trainset=trainset,testset=testset)
 }
 # Entire Dataset
 push1 <- splitdataWORK(eyeMovements_4,i)
 # Extracting the Training Data
 pushTraining <- push1$trainset
 # extracting the Test Data
 pushTest <- push1$testset
 # Implementing the glm on the Training set
 pushTraining.glm <- glm(Condition ~ Duration.of.Fixation +
              Horizontal.Dispersion +
              Vertical.Dispersion +
              Velocity.Horizontal +
              Velocity.Vertical,data = pushTraining,family = binomial())
 # Saving the AIC value of each model into aicValue
 aicValue <- rbind(aicValue,pushTraining.glm$aic)

 # Predicting the values in the test data
 predict_training_glm <- round(predict(training.GLM,
                    pushTest,
                    type = "response"))
 # Measuring the accuracy.
 cAccuracy <- accuracy(pushTest$Condition,
            predict_training_glm,
            threshold = .95)

 # Saving the AUC value from each model into aucValue
 aucValue <- rbind(aucValue,cAccuracy$AUC)
 #==========================LASSO MODEL============================
```

```
 pushTraining.glm.lasso <- glm(Condition ~ Duration.of.Fixation+
                 Vertical.Dispersion,
                data = pushTraining,family = binomial())
 # Saving the AIC value of each model into aicValueLasso
 aicValueLasso <- rbind(aicValueLasso,pushTraining.glm.lasso$aic)

 # Predicting the values in the test data from the lasso model
 predict_lasso_training_glm <- round(predict(pushTraining.glm.lasso,
                       pushTest,
                       type = "response"))

 # Getting the accureacy
 cAccuracyLasso <- accuracy(pushTest$Condition,
              predict_lasso_training_glm,
              threshold = .95)
 # Saving the AUC value from each model into aucValueLasso
 aucValueLasso <- rbind(aucValueLasso,cAccuracyLasso$AUC)
}
```

Statement of contributions

- Mavez Singh-Dabas programmed the functions used to divide the data into training and test sets, as well as those used to analyze the performance of the original full model fit used to detect outliers. He also assisted Pushpinder in developing the functions used to perform the cross-validation procedure.
- Pushpinder Singh formulated the function to perform the cross validation on the data set. He also developed the code used to pinpoint outliers and influential observations. Along with Mavez, he helped generate the function to perform the cross-validation procedure.
- Dylan Rose assisted Mavez and Pushpinder in identifying and customizing the dataset for the purposes of the project. He also selected the statistical methods used for the project and wrote/developed graphs/text used to display the results of the model performance, and assisted Mavez in analyzing the ALASSO model performance.