# Question 2

## a)

If we look at a single row in $\tilde{X}$ we see what the effect of centering is on the coeffiecients $\beta$. Let $\tilde{X}_i$ be the ith row of $\tilde{X}$. Then $\tilde{X}_{ij} = X_{ij} - \mu_j$ where $\mu_j$ is the mean of the jth column in our data.

If we run regression on $\tilde{X}$, we get the coefficients $\beta$. Then the prediction for the jth observation is given by.

$\hat{Y}_j = \beta_0 + \sum_{i=1}^{p} \beta_i \tilde{X}_{ji}$
$= \beta_0 + \sum_{i=1}^{p} \beta_i \left(X_{ji} - \mu_i\right)$
$= \beta_0 - \sum_{i=1}^{p} \beta_i * \mu_i + \sum_{i=1}^{p} \beta_i X_{ji} = \beta_0' + \sum_{i=1}^{p} \beta_i X_{ji}$

We know that the predictions $\hat{Y}$ will find the values of $\beta_i$ to minimize the RSS. Since there is a linear combination of the original $X$ that also produces this $\hat{Y}$ , with the exact same coefficients for $\beta_i, \quad i \in 1...p$, we know that OLS on the original data will also find these coefficients. Therefore the OLS coefficients for the variables do not change when we center, but the intercept does.

## b)

We know that the predicted vector $\hat{y}$ lives in the column space of $X_s$ and that therefore the residual $r = y - \hat{y}$ is perpendicular to this space, as $\hat{y}$ minimizes the residual squared. This means that the variable that will reduce the residual-sum of squares the most is the one that has the smallest angle with $r$. When we add this new vector to our variable basis, the new residual will be the component of the old residual orthogonal to the chosen variable. We will find our next variable $z$ by selecting the one that results in the smallest residual after augmenting $X_s$ with $z$.

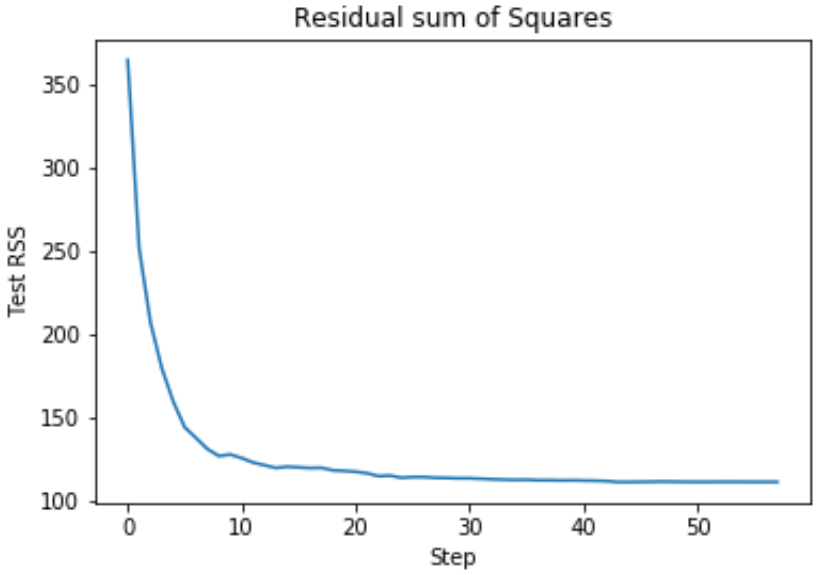
## c)

- Center data
- Compute intercept as the mean of y.
- Center y
- Initialize Q to hold the adjusted data columns in the picked order
- Initialize an upper triangular matrix, coeff, to hold the coefficients for each step
- Initialize a list to keep track of the variables that have been selected
- For each step in (0, p - 1):
    - For all $x_j$ such that $j$ has not yet been selected:

        - Regress y on Q U $\{x_j\}$ and choose the $x_j^*$ that leads to the lowest RSS
    - Add $j^*$ to the selected list
    - Add $x_j^*$ to Q
    - For all $x_i$ such that $i$ has not yet been selected:
        - Remove the component of $x_i$ in the direction of $x_j$
    - Compute the regression coefficients of regressing $y$ on $\{x_k\}$ such that $k$ has been selected and add to the upper triangular coefficient matrix
- Return the columns selected, the coefficient matrix, and the training means vector

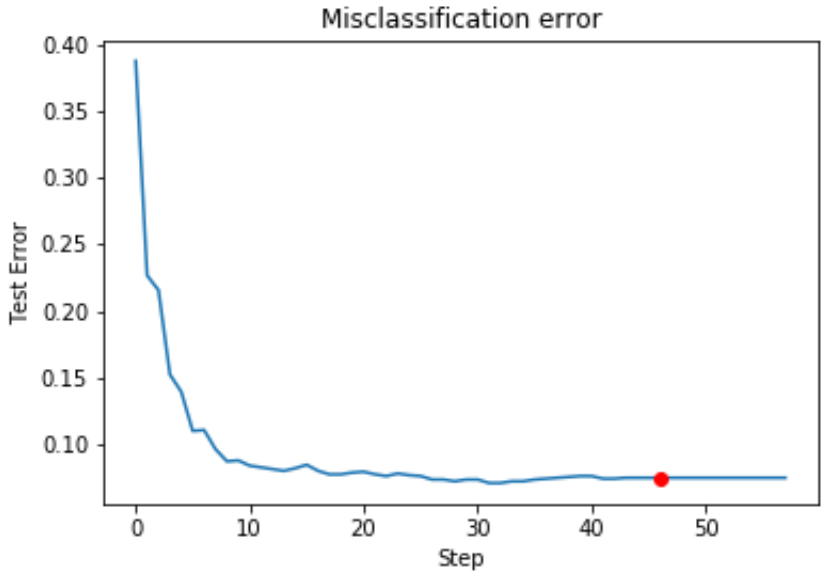
## f)

If we plot the RSS as a function of the step size, we get the following plot. We see that the RSS almost monotonically decreases with the number of steps. However, since the algorithm is greedy on the training data there are steps where the RSS increases.



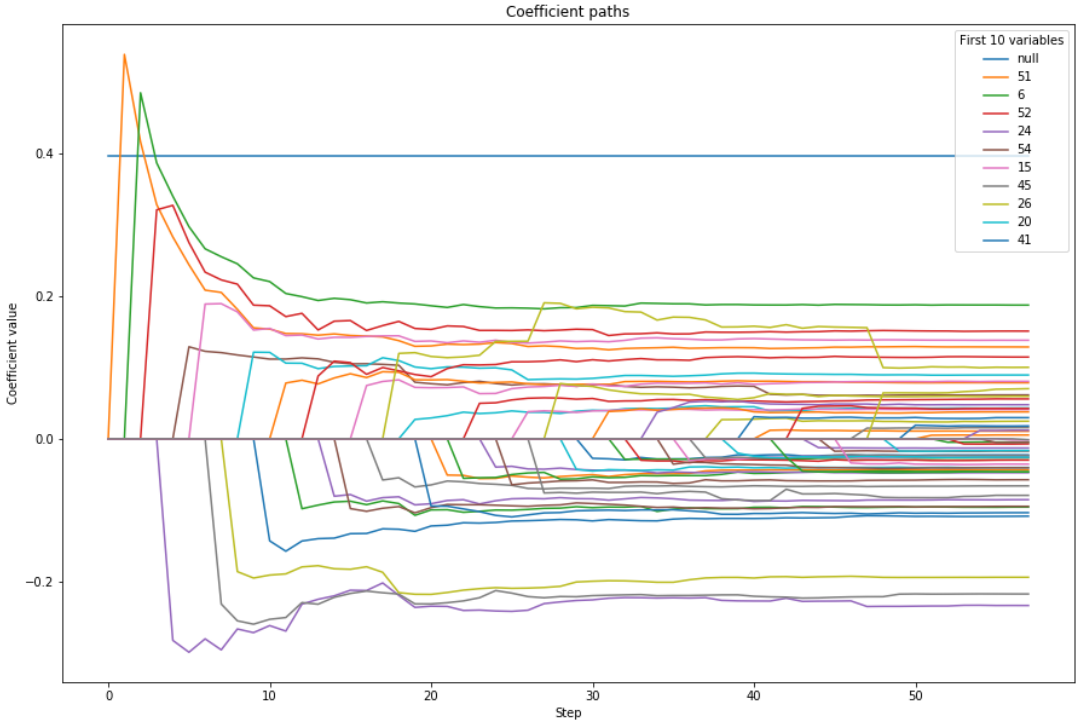

## h)

Like the RSS, the misclassification rate also almost monotonically decreases with number of steps. The red dot indicates the result from cross validation. The optimal number of variables is 46.

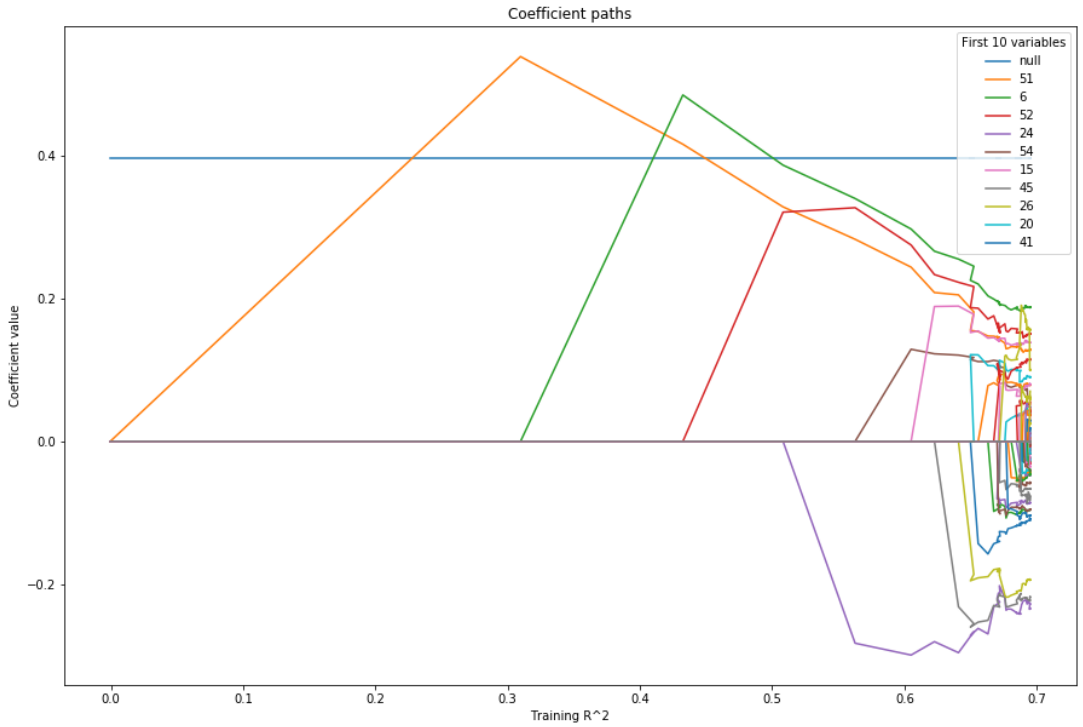



## i)

When we plot the coefficient values against the number of steps we see that out of the first 10 coefficients 7 have positive sign and 3 negative sign. We also not that when the coefficient gets first added its value peaks and then decays as more coefficients get added.



When the coefficient size is plotted against the training $R^2$ we see that the most significant improvement is given by the first couple of variables and that the remaining ones only marginally improve the $R^2$.




# Code

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
from copy import deepcopy

def get_residual(predictors, col):
    '''Helper function to get the residual of a column when regressed with a
set of predictors'''
    reg = LinearRegression(fit_intercept=False).fit(predictors, col)
    x_hat = reg.predict(predictors)
    return col - x_hat

def find_next_variable(X, y, q, Q, cols):
  '''Helper function to get the next variable in forward stepwise'''
  # set default value
  rss = np.inf
```

```
    next_var = - 1

    #iterate over the remaining
    for i in np.setdiff1d(range(X.shape[1]),cols):
      #Create a new data matrix with one of the columns appended
      Z = np.hstack((Q[:,:q], X[:,i].reshape(-1,1)))

      rs = get_residual(Z, y)

      #Get the rss
      rss_ = np.sum((rs)**2)

      #See if it is an improvement
      if rss_ < rss:
        rss = rss_
        next_var = i

    return next_var


def stepwise_forward(X,y):

    X = deepcopy(X)
    y = deepcopy(y)

    #Get the sizes
    p = X.shape[1]
    N = X.shape[0]
    means = np.mean(X,0)

    # Center the data
    for i in range(p):
      X[:,i] -= np.mean(X[:,i])

    # Get another copy to orthogonalize
    X_tilde = deepcopy(X)

    # Get the constant intercept
    beta_0 = np.mean(y)

    # Center the y's
    y = y - beta_0

    # define containers
    cols = []
    coeff = np.zeros((p+1,p+1))
```

```
  Q = np.zeros((N,p))

  #initialize the first row of coeff
  coeff[0,:] = beta_0

  #iterate over the columns
  for i in range(0,p):

    #Find the next variable
    next_var = find_next_variable(X_tilde,y,i,Q,cols)

    #Store the identity of the next column in cols
    cols.append(next_var)

    # Regress the centered X against y
    reg =
LinearRegression(fit_intercept=False).fit(X[:,np.array(cols).astype(int)], y)

    # Update the coefficient matrix
    coeff[1:,i+1] = np.pad(reg.coef_,(0, p - len(reg.coef_)),'constant')

    #Add that column to Q.
    Q[:,i] = X_tilde[:,next_var]

    # Orthogonalize the remaining vectors
    for j in np.setdiff1d(range(X_tilde.shape[1]),cols):

        X_tilde[:,j] = get_residual(Q[:,:i+1], X_tilde[:,j])

  return Q, cols, coeff, means


def load_spam_data():
    X = np.loadtxt('spam.data', usecols=range(57))
    y = np.loadtxt('spam.data', usecols=[57])

    ### Convert features 0-53 to binary responses
    X[:,:54][X[:,:54] != 0] = 1

    ### Convert features 54-57 via log transform
    X[:,54:] = np.log(X[:,54:])

    test_ind = np.loadtxt('test_ind.data')
    X_test, y_test = X[test_ind==1,:], y[test_ind==1]
    X_train, y_train = X[test_ind==0,:], y[test_ind==0]
    return X_train, y_train, X_test, y_test
```

```
def predict(X, cols, coeff, means):

    ## Subtract off the training means
    X = X - means

    ## Reshuffle X columns based on order in the coefficient matrix
    X = X[:,np.array(cols).astype(int)]

    ## Prepend column of 1's to X for intercept term
    X = np.concatenate([np.expand_dims(np.ones(X.shape[0]),1),X],axis=1)

    ## Predict Y
    Y = X @ coeff

    return Y


def plot_metric(y_hat, y, dataset = "Test", metric = "RSS", cv_idx = None,
plot=True):
    n_samples, n_steps = y_hat.shape
    rss = np.zeros(n_steps)
    classification_error = np.zeros(n_steps)

    for i in range(n_steps):
        rss[i] = np.sum((y - y_hat[:,i])**2)
        classification_error[i] = 1 - np.sum(y ==
np.round(y_hat[:,i]))/n_samples

    if plot:
        plt.figure()

        if metric == "RSS":
            plt.plot(range(n_steps),rss)
            plt.title("Residual sum of Squares")
            plt.xlabel("Step")
            plt.ylabel(dataset + " RSS")
            plt.savefig("metric "+dataset+metric)

        elif metric == "Misclassification Error":
            plt.plot(range(n_steps),classification_error)
            plt.title("Misclassification error")
            plt.xlabel("Step")
            plt.ylabel(dataset + " Error")
            if cv_idx:
                plt.plot(cv_idx, classification_error[cv_idx], 'ro')
```

```
            plt.savefig("metric "+dataset+metric)

    return classification_error, rss


def cross_validation(X, y):
    X = deepcopy(X)
    y = deepcopy(y)
    _, n_steps = X.shape
    classification_error = np.zeros((10,n_steps))
    k = 0

    kf = KFold(n_splits=10)
    for train_idx, val_idx in kf.split(X):
        X_train, X_val = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]
        Q, cols, coeff, means = stepwise_forward(X_train, y_train)
        y_hat_val = predict(X_val, cols, coeff, means)
        n_samples, _ = y_hat_val.shape
        for i in range(n_steps):
            classification_error[k,i] = 1 - np.sum(y_val ==
np.round(y_hat_val[:,i]))/n_samples
        k = k + 1

    subset_error_estimates = np.sum(classification_error,0) * 0.1
    best_subset = np.argmin(subset_error_estimates)
    return best_subset

def plot_coeff_path(coeff, cols, x = None):
    if x is None:
        plt.figure(figsize=(15,10))
        plt.title("Coefficient paths")
        plt.xlabel("Step")
        plt.ylabel("Coefficient value")
        series = plt.plot(coeff.T)
        plt.legend(series[:11],["null"] + [str(col) for col in cols][:10],
title= "First 10 variables",loc=1)
        plt.savefig("coeff_path1")
    else:
        plt.figure(figsize=(15,10))
        plt.title("Coefficient paths")
        plt.xlabel("Training R^2")
        plt.ylabel("Coefficient value")
        series = plt.plot(x, coeff.T)
        plt.legend(series[:11],["null"] + [str(col) for col in cols][:10],
title= "First 10 variables",loc=1)
```

```
        plt.savefig("coeff_path2")


X_train, y_train, X_test, y_test = load_spam_data()
Q, cols, coeff, means = stepwise_forward(X_train, y_train)
y_hat_train = predict(X_train, cols, coeff, means)
y_hat_test = predict(X_test, cols, coeff, means)

cv_idx = cross_validation(X_train, y_train)

_, rss_train = plot_metric(y_hat_train, y_train, "Train", "RSS", plot=False)
plot_metric(y_hat_train, y_train, "Train", "Misclassification Error",
plot=False)
#_, _ = plot_metric(y_hat_test, y_test, "Test", "RSS", plot=True)
#_, _ = plot_metric(y_hat_test, y_test, "Test", "Misclassification Error",
cv_idx, plot=True)

#plot_coeff_path(coeff, cols)
#plot_coeff_path(coeff, cols, rss_train)
r2 = np.zeros(58)
for i in range(58):
    r2[i] = r2_score(y_test,y_hat_test)
plot_coeff_path(coeff, cols, r2)
```

# Homework 2, STATS 315A

Stanford University, Winter 2019

*Joe Higgins, Jess Wetstone, Remmelt Ammerlaan*

## Question 7

7. Obtain the zipcode train and test data from the ESL website.

i. Compare the test performance of a) linear regression b) linear discriminant analysis and c) multiclass linear logistic regression.

```
rm(list = ls())

read_data_as_matrix <- function(file_string){
  table <- read.table(file_string)
  output <- matrix(, nrow=dim(table)[1], ncol=dim(table)[2])
  for(i in 1:ncol(table)){
    output[,i] <- table[,i]
  }
  return(output)
}

get_accuracy <- function(y_hat, y){
  correct <- y_hat == y
  pct_correct <- sum(correct)/length(correct)
  return(pct_correct)
}

#import data
train <- read_data_as_matrix("zip.train")
y_train <- train[,1]
x_train <- train[,2:dim(train)[2]]
test <- read_data_as_matrix("zip.test")
y_test <- test[,1]
x_test <- test[,2:dim(test)[2]]

#encode y_train as one hot matrix
to_one_hot_matrix <- function(vector){
  K <- 10
  one_hot_matrix <- c()
  for(i in 1:length(vector)){
    class <- vector[i]
    new_row <- rep(0,K)
    new_row[class+1] <- 1
    one_hot_matrix <- rbind(one_hot_matrix, new_row)
  }
  return(one_hot_matrix)
}
y_train_1hot <- to_one_hot_matrix(y_train)


#train models
```

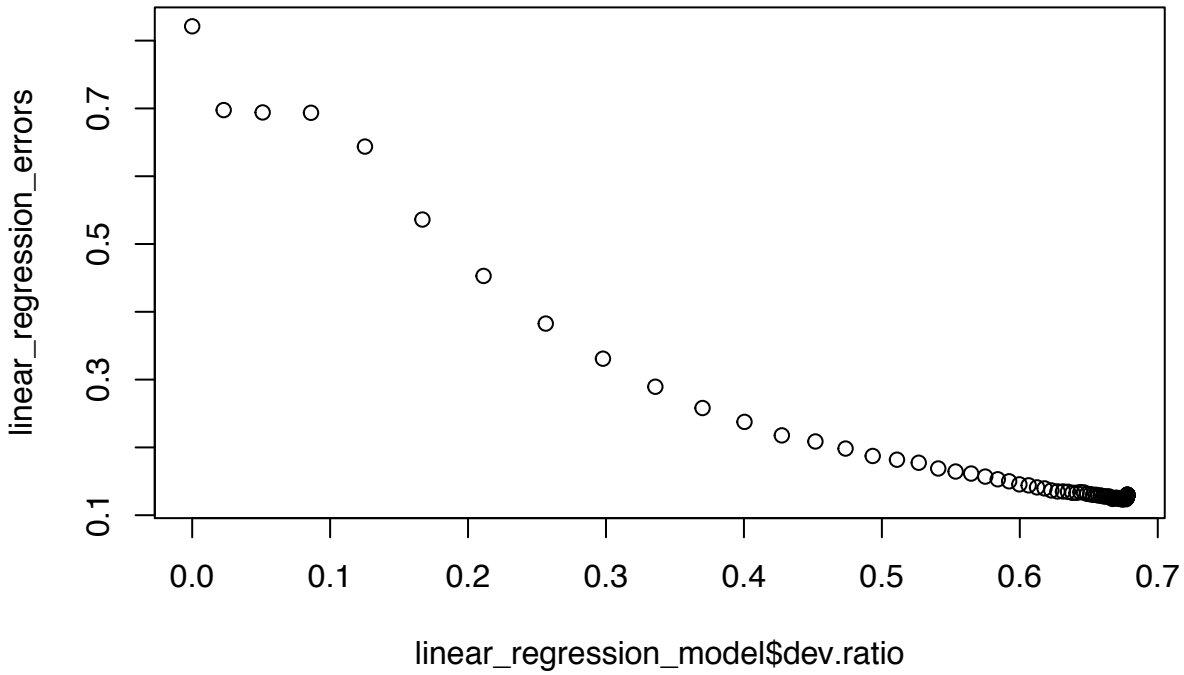
```r
linear_regression_model <- glmnet(
  x_train, y_train_1hot,
  family=c("mgaussian"), alpha = 0.3
)
multinomial_regression_model <- glmnet(
  x_train, y_train,
  family=c("multinomial"), alpha = 0.3
)
lda_model <- lda(
  y_train ~ ., data=data.frame(x_train),
  na.action="na.omit", CV=FALSE
)

#create predictions
linear_regression_output <- predict(
  linear_regression_model, x_test, type=c("response")
)
linear_regression_prdct <- apply(
  linear_regression_output, 3, function (x) apply(
    x, 1, function(y) which.max(y) - 1
  )
)
multinomial_regression_output <- predict(
  multinomial_regression_model, x_test, type=c("response")
)
multinomial_regression_prdct <- apply(
  multinomial_regression_output, 3, function (x) apply(
    x, 1, function(y) which.max(y) - 1
  )
)
lda_output <- predict(lda_model, data.frame(x_test))
lda_prdct <- lda_output$class

#get errors
linear_regression_errors <- apply(
  linear_regression_prdct, 2,
  function(x) 1 - get_accuracy(x, y_test)
)
multinomial_regression_errors <- apply(
  multinomial_regression_prdct, 2,
  function(x) 1 - get_accuracy(x, y_test)
)
lda_error <- 1 - get_accuracy(lda_prdct, y_test)

#report errors
cat("Test performance:\n")
cat(
  "Linear Regression, min error over Lambda:",
  min(linear_regression_errors), "\n"
)
cat(
  "Linear Discriminant Analysis error:",
  lda_error, "\n"
```

```
)
cat(
  "Multinomial Regression, min error over Lambda:",
  min(multinomial_regression_errors), "\n"
)
```

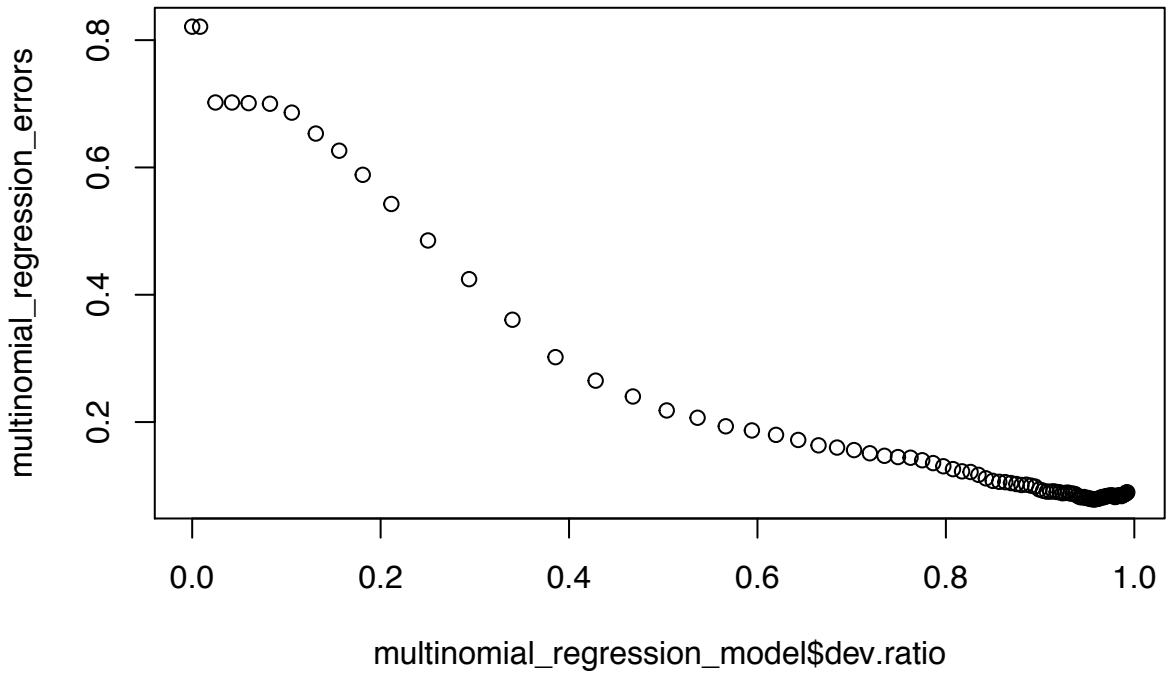
```
## Test performance:
## Linear Regression, min error over Lambda: 0.1235675
## Linear Discriminant Analysis error: 0.1145989
## Multinomial Regression, min error over Lambda: 0.07822621
```

ii. For a) and c), use the package `glmnet` (available in R, matlab and python) to run elastic-net regularized versions of each (use $\alpha = 0.3$). For these two, plot the test error as a functions of the training $R^2$ for a) and $D^2$ for c) (% training deviance explained).

```
plot(linear_regression_model$dev.ratio, linear_regression_errors)
```




```
plot(multinomial_regression_model$dev.ratio, multinomial_regression_errors)
```




iii. In ii., what is the optimization problem being solved?

$\min_\beta ||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda[\alpha||\beta||_2^2 + (1-\alpha)||\beta||_1]$