# Assignment 4

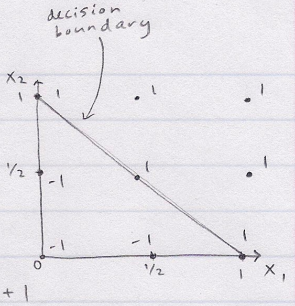
*Mikaela Hoffman-Stapleton*

*April 21, 2017*

See next page.

## Quantitative Questions

1. (a) $\hat{y}_0 = \begin{cases} -1 & \text{if } \eta(x) < 0.5 \\ 1 & \text{if } \eta(x) \geq 0.5 \end{cases} \longrightarrow \hat{y}_0 = \begin{cases} -1 & \text{if } \frac{x_1+x_2}{2} < 0.5 \\ 1 & \text{if } \frac{x_1+x_2}{2} \geq 0.5 \end{cases}$ where $0 \leq x_1, x_2 \leq 1$




$y = mx + b \longrightarrow 0 = -1(1) + b \longrightarrow 0 = -1 + b \longrightarrow b = 1 \longrightarrow y = -x + 1$

hence, the decision boundary is $\boxed{x_2 = -x_1 + 1}$

(b) $p(x_1, x_2) = p(x_1)\,p(x_2)$ since they are independent

$p(x_1) = 1$ and $p(x_2) = 3x_2^2$ for $0 \leq x_2 \leq 1$

$p(x_1, x_2) = (1)(3x_2^2) = \boxed{3x_2^2 \text{ for } 0 \leq x_2 \leq 1}$

(c) $\pi_1 = P(Y=1) = \int P(Y=1|X)\,P(X) = \int \eta(x)\,p(x_1, x_2) = \int_0^1 \int_0^1 \frac{x_1+x_2}{2} 3x_2^2\,dx_1\,dx_2 = \int_0^1 \left[\frac{3x_2^3}{2} + \frac{3x_2^2 x_1^2}{4}\right]_0^1 dx_2$

$= \frac{3x_2^4}{8} + \frac{3x_2^3}{12}\Big]_0^1 = \frac{9}{24} + \frac{6}{24} = \boxed{\frac{15}{24}}$

(d) $P(X|Y=1) = \frac{P(Y=1|X)\,P(X)}{P(Y=1)} = \frac{\eta(x)\,p(x_1, x_2)}{\pi_1} = \frac{\left(\frac{x_1+x_2}{2}\right)(3x_2^2)}{15/24} = \boxed{\frac{12x_2^2(x_1+x_2)}{5}}$

2. (a) $\text{logit}(\eta(x;\beta)) = \log\left(\frac{\eta(x;\beta)}{1-\eta(x;\beta)}\right) = \beta^T x \longrightarrow \frac{\eta(x;\beta)}{1-\eta(x;\beta)} = e^{\beta^T x}$

$\eta(x;\beta) = e^{\beta^T x}(1 - \eta(x;\beta))$

$\eta(x;\beta)(1 + e^{\beta^T x}) = e^{\beta^T x}$

$$\eta(x;\beta) = \frac{e^{\beta^T x}}{1+e^{\beta^T x}} = \frac{\frac{e^{\beta^T x}}{e^{\beta^T x}}}{\frac{1}{e^{\beta^T x}} + \frac{e^{\beta^T x}}{e^{\beta^T x}}} = \frac{1}{1+e^{-\beta^T x}}$$

(b) $\frac{\partial \log(\eta(x;\beta))}{\partial \beta} = \frac{\partial}{\partial \beta}\left(\log\left(\frac{1}{1+e^{-\beta^T x}}\right)\right)$

$= \frac{\partial}{\partial \beta}\left(\log(1) - \log\left(1+e^{-\beta^T x}\right)\right)$

$= -\frac{1}{1+e^{-\beta^T x}} \frac{\partial}{\partial \beta}\left(1+e^{-\beta^T x}\right)$

$\overset{*}{=} \frac{1}{1+e^{-\beta x}} e^{-\beta x}(-x) = \boxed{\frac{x e^{-\beta x}}{1+e^{-\beta x}}}$

* we can drop the T ($\beta^T \longrightarrow \beta$) because we are considering a one-dimensional situation

$$\frac{\partial^2 \log(\eta(x;\beta))}{\partial^2 \beta} = \frac{\partial}{\partial \beta}\left(\frac{x e^{-\beta x}}{1+e^{-\beta x}}\right) = \frac{x e^{-\beta x}(-x)(1+e^{-\beta x}) - x e^{-\beta x} e^{-\beta x}(-x)}{(1+e^{-\beta x})^2} = \boxed{\frac{-x^2 e^{-\beta x}}{(1+e^{-\beta x})^2}} < 0 \checkmark$$

(always −) / (always +)

this is a maximum $\longrightarrow$ this parameter estimate is the one for which the observed data would have the highest probability of occurrence

(c) i. $\frac{1}{1+e^{-(-6+0.05(40)+1(3.5))}} = \frac{1}{1+e^{6-2-3.5}} = \frac{1}{1+e^{0.5}} = .3775 \longrightarrow \boxed{37.75\%}$

ii. $\frac{1}{1+e^{-(-6+0.05x+1(3.5))}} = 0.5 \longrightarrow 1 = 0.5 + 0.5e^{-0.05x+2.5} \longrightarrow \log\left(\frac{1-0.5}{0.5}\right) = \log\left(e^{-0.05x+2.5}\right)$

$\hookrightarrow +2.5 = +0.05x \longrightarrow x = \frac{2.5}{0.05} = 50 \longrightarrow \boxed{50 \text{ hours}}$

## Computational Questions

### 1.

```
# set-up
data <- read.csv('~/Documents/MSAN 628/HW 4/OnlineNewsPopularityTraining.csv', header = T)
data$shares <- NULL
data$url <- NULL
data$timedelta <- NULL
x <- model.matrix(popular ~., data = data)[, -1]
y <- data$popular
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
y.train <- y[train]
y.test  <- y[test]
grid.lambda <- 10^seq(10, -2, length = 100)
```

#### (a)

```
nrow(data[which(data$popular == 0),])/nrow(data)
```

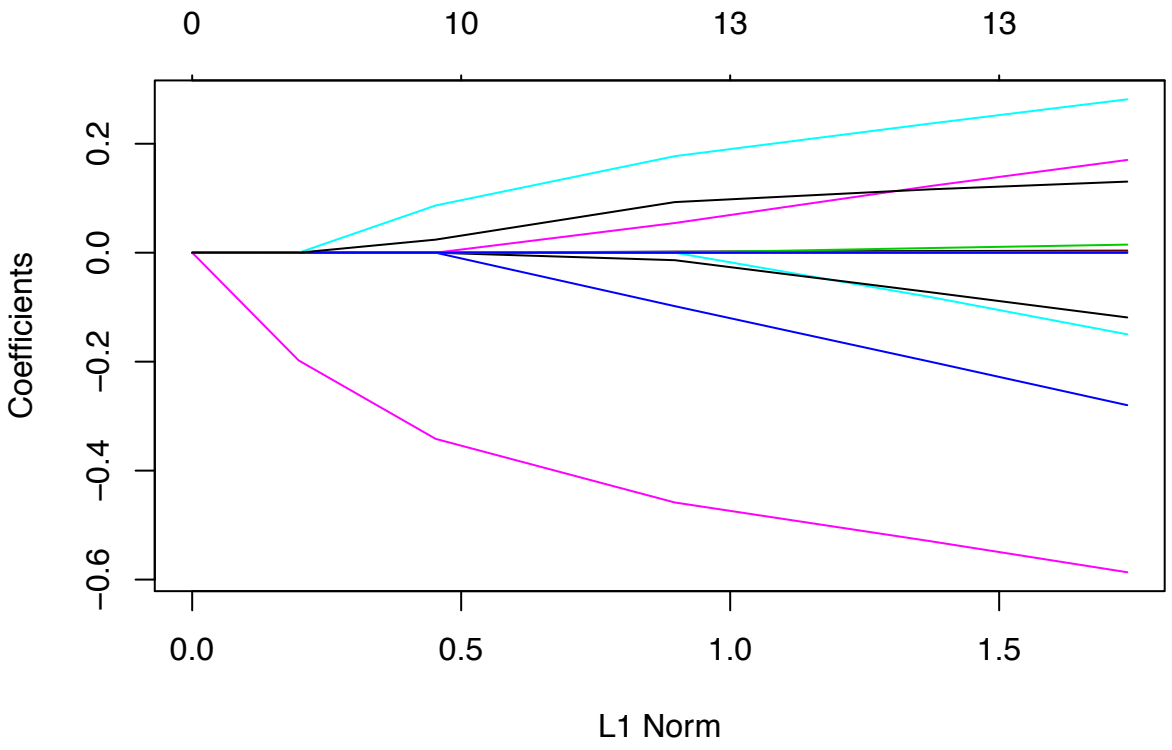
```
## [1] 0.797074
```

```
nrow(data[which(data$popular == 1),])/nrow(data)
```

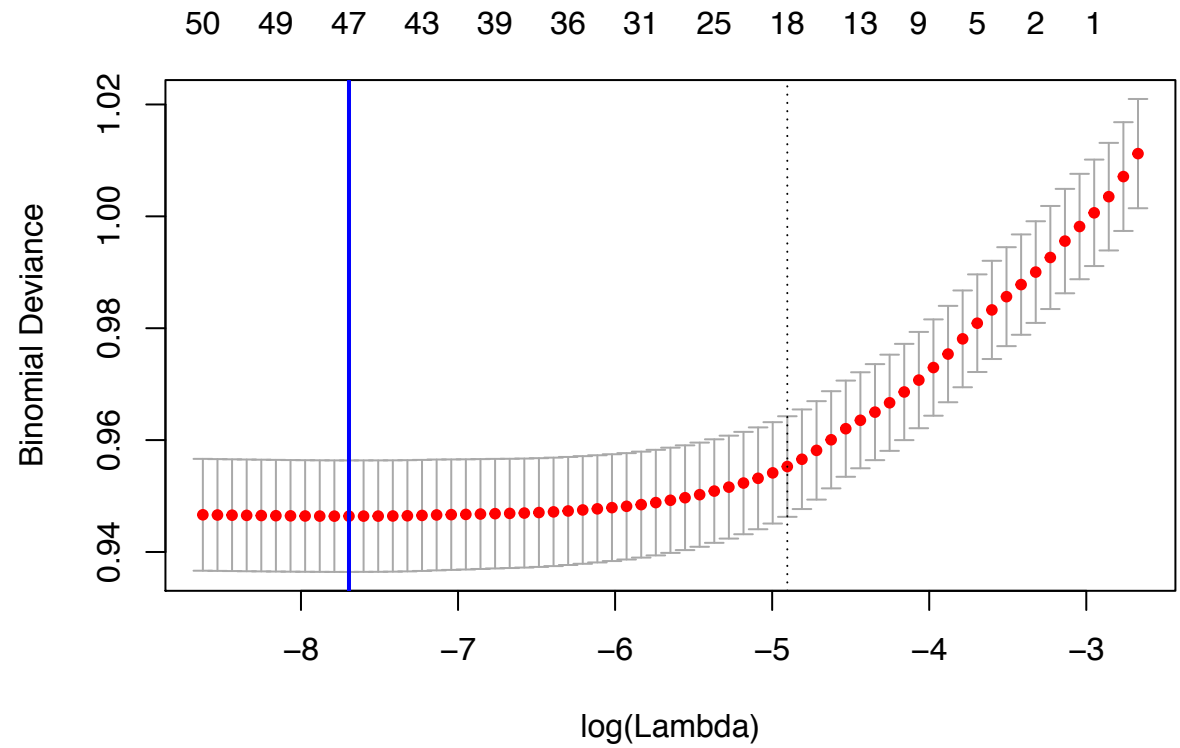
```
## [1] 0.202926
```

We see that the data is unbalanced, where 0.7971% of articles are not popular and 0.2029% of articles are popular. As a result, we will experiement with downsamping the negative cases, but first lets see the outcome of training a model with the entire dataset.

```
# all predictors included
all.model.train <- glm(popular ~ ., data = data, family = "binomial", subset = train)
all.probs.train <- predict(all.model.train, type = "response", newx = x[test,
    ])
d <- length(y.test)
all.pred.test <- rep(0, d)
all.pred.test[all.probs.train > 0.5] <- 1
mspe.all <- mean((all.pred.test - y.test)^2)
all.final.model <- glm(popular ~ ., data = data, family = "binomial")
# lasso
lasso.model <- glmnet(x, y, family = "binomial", alpha = 1, lambda = grid.lambda)
plot(lasso.model)
```




```
lasso.model.train <- glmnet(x[train, ], y.train, family = "binomial", alpha = 1,
    lambda = grid.lambda)
lasso.cv.out <- cv.glmnet(x[train, ], y.train, family = "binomial", alpha = 1)
lasso.best.lambda <- lasso.cv.out$lambda.min
plot(lasso.cv.out)
abline(v = log(lasso.best.lambda), col = "blue", lwd = 2)
```



```
lasso.probs.train <- predict(lasso.model.train, type = "response", s = lasso.best.lambda,
    newx = x[test, ])
lasso.pred.test <- rep(0, d)
lasso.pred.test[lasso.probs.train > 0.5] <- 1
mspe.lasso <- mean((lasso.pred.test - y.test)^2)
lasso.final.model <- glmnet(x, y, family = "binomial", alpha = 1, lambda = lasso.best.lambda)
coef.lasso <- coef(lasso.final.model)
exclude <- sum(coef.lasso == 0)
exclude
```

```
## [1] 8
```

```
# comparison
MSPE <- data.frame(All = mspe.all, Lasso = mspe.lasso)
MSPE
```

```
##         All     Lasso
## 1 0.2147181 0.2023584
```

As expected, using the lasso has sent 8 predictors to zero, simplifying the original model and making it more interpretable. We also end up getting a slightly lower MSPE, which is another advantage to using the lasso model. However, we notice that these MSPEs are very similar to the percentage of articles that are popular, indicating that the models are probably predicting mostly zeroes. To check this, we determine the model's specificity:

```
specificity(factor(all.pred.test), factor(y.test))
```

```
## [1] 0.0215423
```

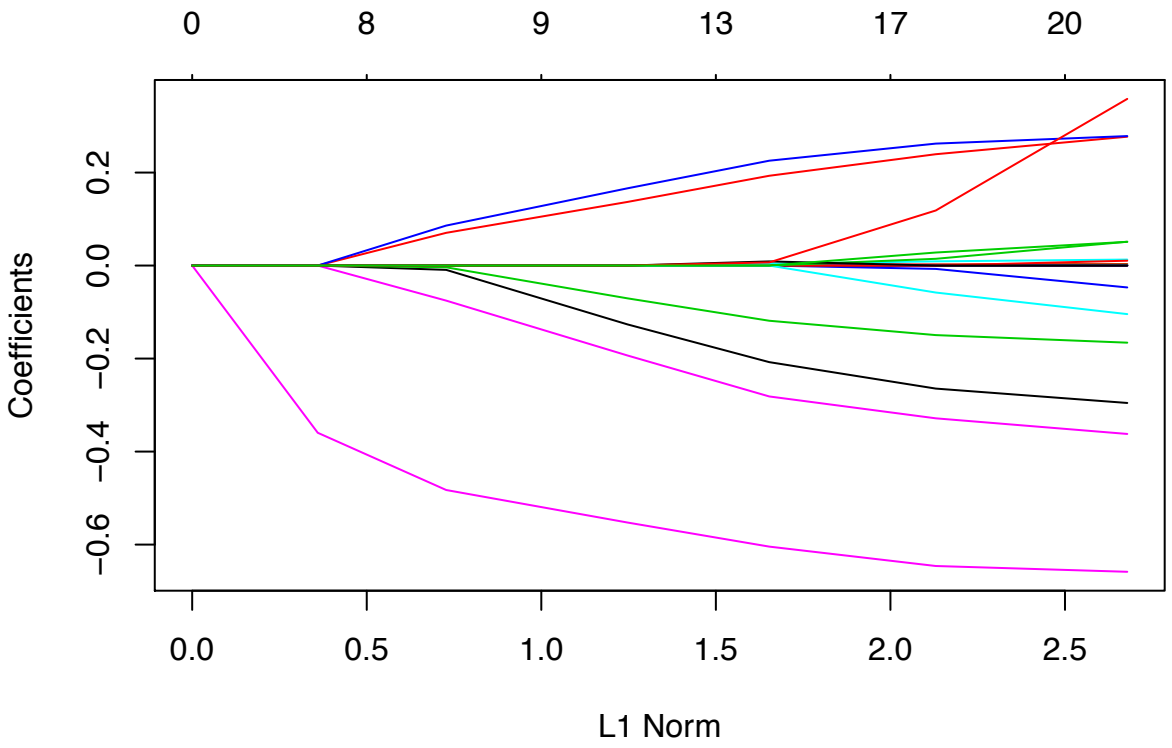
```
specificity(factor(lasso.pred.test), factor(y.test))
```

```
## [1] 0.006868561
```

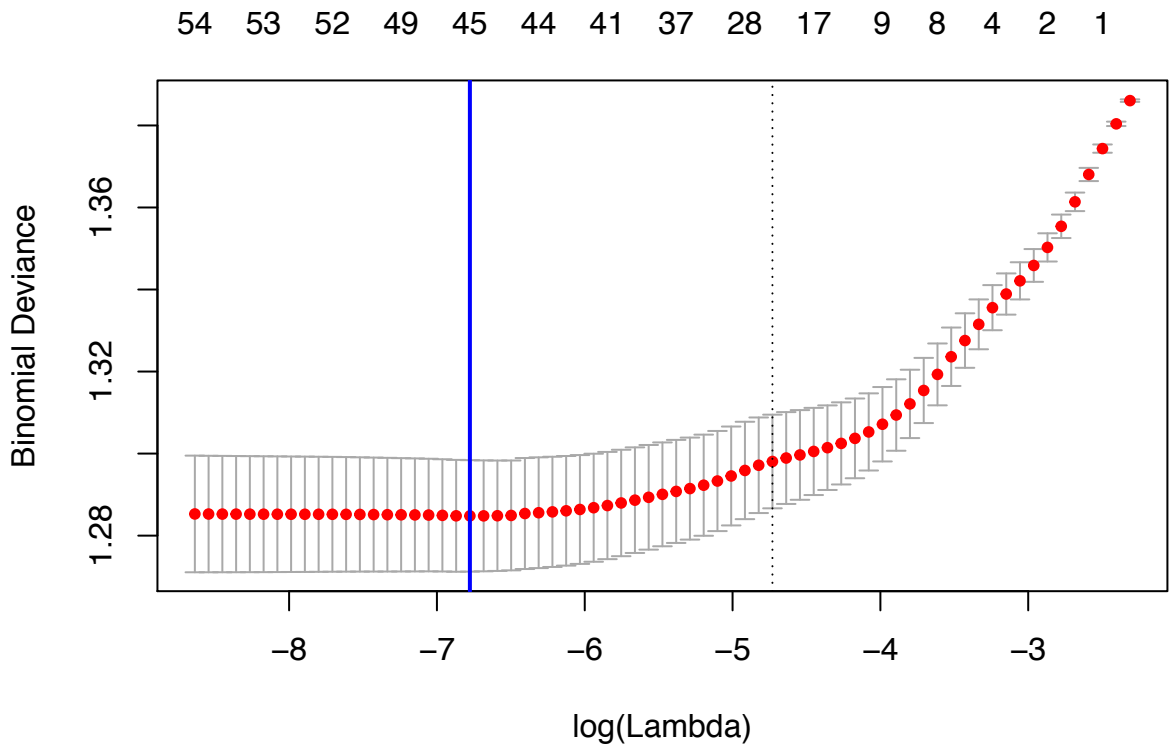
Indeed, these are very low, indicating that the models are hardly picking up any of the popular articles. We now turn to downsampling in an attempt to increase the models' specificity.

```
indices <- sample(which(data$popular == 0), nrow(data[which(data$popular ==
    1), ]))
indices <- c(indices, which(data$popular == 1))
indices <- sample(indices, length(indices))
new.data = data[indices, ]
x <- model.matrix(popular ~ ., data = new.data)[, -1]
y <- new.data$popular
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.train <- y[train]
y.test <- y[test]
# all predictors included
all.model.train <- glm(popular ~ ., data = data, family = "binomial", subset = train)
all.probs.train <- predict(all.model.train, type = "response", newx = x[test,
```

```
    ])
d <- length(y.test)
all.pred.test <- rep(0, d)
all.pred.test[all.probs.train > 0.5] <- 1
mspe.all <- mean((all.pred.test - y.test)^2)
spec.all <- specificity(factor(all.pred.test), factor(y.test))
all.final.model <- glm(popular ~ ., data = data, family = "binomial")
# lasso
lasso.model <- glmnet(x, y, family = "binomial", alpha = 1, lambda = grid.lambda)
plot(lasso.model)
```




```
lasso.model.train <- glmnet(x[train, ], y.train, family = "binomial", alpha = 1,
    lambda = grid.lambda)
lasso.cv.out <- cv.glmnet(x[train, ], y.train, family = "binomial", alpha = 1)
lasso.best.lambda <- lasso.cv.out$lambda.min
plot(lasso.cv.out)
abline(v = log(lasso.best.lambda), col = "blue", lwd = 2)
```




```
lasso.probs.train <- predict(lasso.model.train, type = "response", s = lasso.best.lambda,
    newx = x[test, ])
lasso.pred.test <- rep(0, d)
lasso.pred.test[lasso.probs.train > 0.5] <- 1
mspe.lasso <- mean((lasso.pred.test - y.test)^2)
spec.lasso <- specificity(factor(lasso.pred.test), factor(y.test))
lasso.final.model <- glmnet(x, y, family = "binomial", alpha = 1, lambda = lasso.best.lambda)
coef.lasso <- coef(lasso.final.model)
exclude <- sum(coef.lasso == 0)
exclude
```

```
## [1] 11
```

```
# comparison
MSPE <- data.frame(All = mspe.all, Lasso = mspe.lasso)
MSPE
```

```
##         All     Lasso
## 1 0.5004661 0.3609385
```

```
Specificity <- data.frame(All = spec.all, Lasso = spec.lasso)
Specificity
```

```
##          All     Lasso
## 1 0.01838579 0.6086008
```

As we can see, both MSPEs have increased, however, the lasso model's specificity has also increased. This means that, while the lasso model is technically worse at predicting, it is able to pick up the popular articles

way better than when we used the entire dataset. Since we are interested in interpreting the model and figuring out which predictors are relevant (and not necessarily in attaining the best predictions), we will stick with using the downsampled dataset. The lasso model, which has now sent 11 predictors to zero, is the clear choice. It gives the following coefficients:

```
coef(lasso.final.model)
```

```
## 59 x 1 sparse Matrix of class "dgCMatrix"
##                                        s0
## (Intercept)                   -1.537451e+00
## n_tokens_title                 1.264569e-02
## n_tokens_content               1.305070e-04
## n_unique_tokens                2.937274e-03
## n_non_stop_words               4.176399e-04
## n_non_stop_unique_tokens       6.206614e-04
## num_hrefs                      5.066877e-03
## num_self_hrefs                -1.465313e-02
## num_imgs                       2.653315e-03
## num_videos                     3.116622e-03
## average_token_length          -1.351833e-01
## num_keywords                   2.412380e-02
## data_channel_is_lifestyle     -8.263205e-02
## data_channel_is_entertainment -3.814078e-01
## data_channel_is_bus           -4.485052e-01
## data_channel_is_socmed         3.196872e-01
## data_channel_is_tech           1.877678e-01
## data_channel_is_world         -1.467968e-01
## kw_min_min                     5.523770e-04
## kw_max_min                    -6.932222e-06
## kw_avg_min                    -1.151851e-05
## kw_min_max                    -1.518826e-07
## kw_max_max                    -1.406061e-07
## kw_avg_max                    -7.405927e-07
## kw_min_avg                    -6.731368e-05
## kw_max_avg                    -7.185698e-05
## kw_avg_avg                     6.313530e-04
## self_reference_min_shares      5.154665e-06
## self_reference_max_shares      .
## self_reference_avg_sharess     6.655708e-06
## weekday_is_monday              1.212394e-01
## weekday_is_tuesday            -4.524725e-02
## weekday_is_wednesday          -4.425470e-02
## weekday_is_thursday           -3.418510e-02
## weekday_is_friday              2.615041e-02
## weekday_is_saturday            .
## weekday_is_sunday              1.638976e-01
## is_weekend                     3.088835e-01
## LDA_00                         4.444791e-01
## LDA_01                        -1.379078e-01
## LDA_02                        -4.595699e-01
## LDA_03                         .
## LDA_04                         3.460912e-02
## global_subjectivity            1.104831e+00
## global_sentiment_polarity      .
```

```
## global_rate_positive_words    -2.017912e+00
## global_rate_negative_words     .
## rate_positive_words            2.789398e-02
## rate_negative_words            .
## avg_positive_polarity         -3.588303e-01
## min_positive_polarity          .
## max_positive_polarity          .
## avg_negative_polarity         -2.084514e-01
## min_negative_polarity          .
## max_negative_polarity          .
## title_subjectivity             1.473072e-01
## title_sentiment_polarity       7.684977e-02
## abs_title_subjectivity         1.885332e-01
## abs_title_sentiment_polarity   .
```

**(b)**

For this part, we need to come up with a prior that makes sense. In class, we went over the Beta-Binomial model, so letting $p_j \sim \text{Beta}(\alpha_j, \gamma_j)$ is a good choice. However, we must provide the parameters $\alpha_j$ and $\gamma_j$. Looking at the Beta distribution on Wikipedia, we see that these two parameters can be written in terms of the expected value ($\mu$) and the variance ($\sigma$):

$$\alpha = \mu\sigma$$
$$\gamma = (1-\mu)\sigma$$

So we can solve for $\alpha$ and $\gamma$ using our best estimates of $\mu$ and $\sigma$. We can calculate the sample probabilities $p_0$ and $p_1$ and treat these as the expected value of each distribution. This would mean using $p_0$ and $p_1$ as our estimates of $\mu_0$ and $\mu_1$. We can also use the variance of the response column as our estimate for $\sigma$.

```
mu0 <- nrow(data[which(data$popular == 0),])/nrow(data)
mu1 <- nrow(data[which(data$popular == 1),])/nrow(data)
sigma <- var(data$popular)
alpha0 <- mu0*sigma
gamma0 <- (1-mu0)*sigma
alpha1 <- mu1*sigma
gamma1 <- (1-mu1)*sigma
```

Then we can find the posterior:

$$p(p_j|z_j) \propto p(p_j)p(z_j|p_j)$$
$$\propto p_j^{\alpha_j-1}(1-p_j)^{\gamma_j-1}p_j^{k_j}(1-p_j)^{n-k_j}$$
$$\propto p_j^{\alpha_j-1+k_j}(1-p_j)^{\gamma_j-1+n-k_j}$$

where we have $k_j$ successes in $n$ trials. This is another Beta distribution (as expected) with parameters $\alpha_j + k_j$ and $\gamma_j + n - k_j$. Hence, we have $p_j|z_j \sim \text{Beta}(\alpha_j + k_j, \gamma_j + n - k_j) =$. Now that we have a distribution for the posterior, we can simulate random variables to take as our $p_j$'s in $logit(p_j) = \beta X$.

```
n <- nrow(data)
k0 <- nrow(data[which(data$popular == 0),])
k1 <- nrow(data[which(data$popular == 1),])
p0 <- rbeta(k0,alpha0+k0,gamma0+n-k0)
p1 <- rbeta(k1,alpha1+k1,gamma1+n-k1)
p0[0:10]
```

```
##  [1] 0.7990317 0.7916089 0.7984708 0.8004172 0.7949250 0.7943946 0.7976616
##  [8] 0.7987219 0.7979575 0.7991695
```

```
p1[0:10]
```

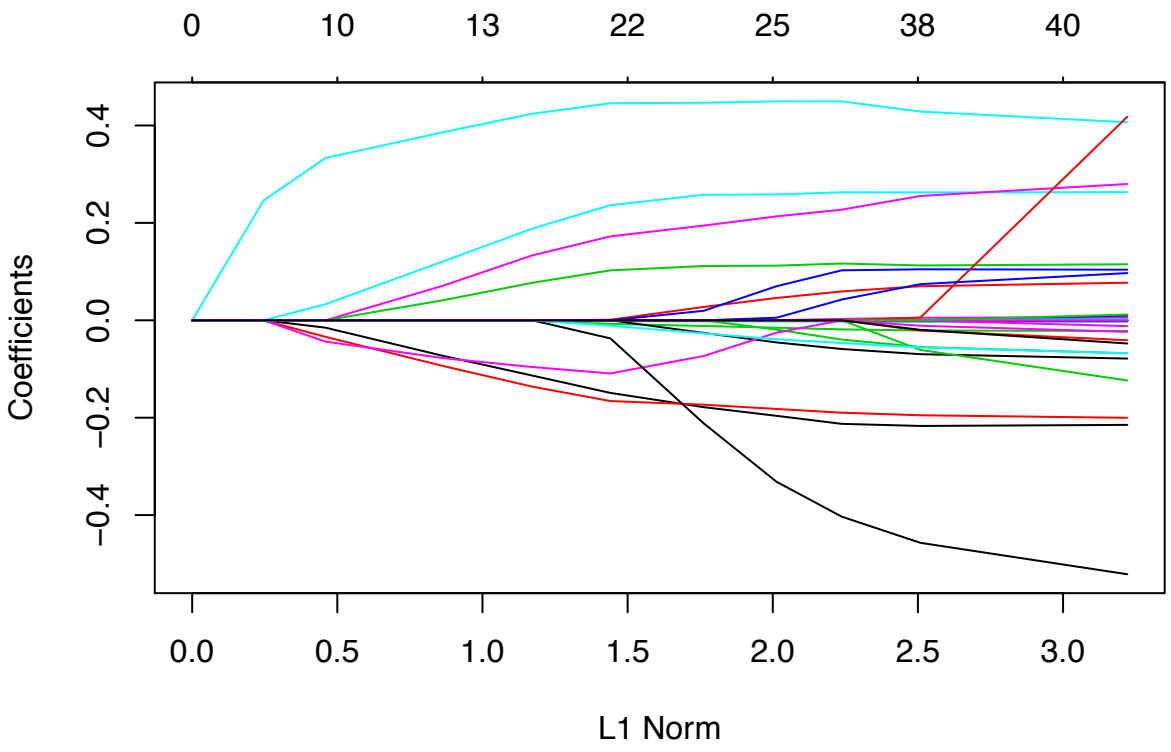
```
##  [1] 0.2034070 0.2015548 0.2047490 0.2035082 0.2030137 0.2044806 0.1986292
##  [8] 0.1998418 0.2008992 0.2033582
```

These seem reasonable, so we will stick with this strategy. Now we want to transform the response column. Those observations corresponding to zero will be populated by the set of $p_0$ random variables, and those corresponding to one will be populated by the set of $p_1$ random variables.

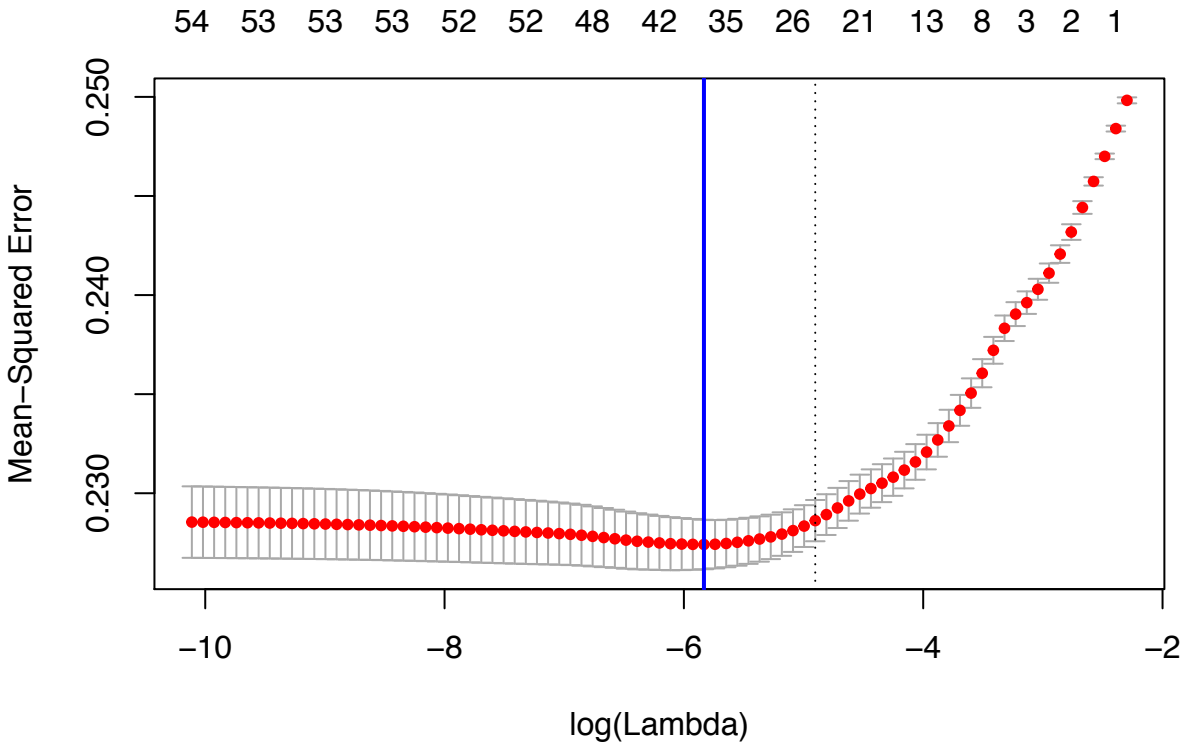
```
transformed <- new.data
for (i in 1:length(new.data$popular)) {
  if (transformed$popular[i] == 0) {
    transformed$popular[i] <- logit(p0[1])
    p0 <- p0[-1]
  }else {
    transformed$popular[i] <- logit(p1[1])
    p1 <- p1[-1]
  }
}
```

Now we can fit a linear model to this to get estimates for the $\beta$ coefficients. As we did in part (a), we can use the lasso to simplify the model and try to get a better fit.

```
# set-up
x.trans <- model.matrix(popular ~ ., data = transformed)[, -1]
y.trans <- transformed$popular
train.trans <- sample(1:nrow(x.trans), nrow(x.trans)/2)
test.trans <- (-train.trans)
y.train.trans <- y[train.trans]
y.test.trans <- y[test.trans]
# all predictors included
lm.all.model.train <- lm(popular ~ ., data = transformed, subset = train.trans)
lm.all.probs.train <- inv.logit(predict(lm.all.model.train, newx = x.trans[test.trans,
    ]))
lm.all.pred.test <- rep(0, d)
lm.all.pred.test[lm.all.probs.train < 0.5] <- 1
lm.mspe.all <- mean((lm.all.pred.test - y.test.trans)^2)
lm.all.final.model <- lm(popular ~ ., data = transformed)
# lasso
lm.lasso.model <- glmnet(x.trans, y.trans, family = "gaussian", alpha = 1, lambda = grid.lambda)
plot(lm.lasso.model)
```




```
lm.lasso.model.train <- glmnet(x.trans[train.trans, ], y.train.trans, family = "gaussian",
    alpha = 1, lambda = grid.lambda)
lm.lasso.cv.out <- cv.glmnet(x.trans[train.trans, ], y.train.trans, family = "gaussian",
    alpha = 1)
lm.lasso.best.lambda <- lm.lasso.cv.out$lambda.min
lm.lasso.best.lambda <- lm.lasso.cv.out$lambda.min
plot(lm.lasso.cv.out)
abline(v = log(lm.lasso.best.lambda), col = "blue", lwd = 2)
```




```
lm.lasso.probs.train <- inv.logit(predict(lm.lasso.model.train, s = lm.lasso.best.lambda,
    newx = x.trans[test.trans, ]))
lm.lasso.pred.test <- rep(0, d)
lm.lasso.pred.test[lm.lasso.probs.train < 0.5] <- 1
lm.mspe.lasso <- mean((lm.lasso.pred.test - y.test.trans)^2)
lm.lasso.final.model <- glmnet(x.trans, y.trans, family = "gaussian", alpha = 1,
    lambda = lm.lasso.best.lambda)
lm.coef.lasso <- coef(lm.lasso.final.model)
exclude <- sum(lm.coef.lasso == 0)
exclude
```

```
## [1] 13
```

```
# comparison
MSPE <- data.frame(All = lm.mspe.all, Lasso = lm.mspe.lasso)
MSPE
```

```
##         All     Lasso
## 1 0.5013984 0.4945618
```

This time, we don't see much difference in MSPE, so both models are on par with guessing randomly. While this is unfortunate, it is important to remember that this model was only trained on part of the balanced data – we will see how the final model (trained on all of the balanced data) does on the actual test set. The lasso model gives the following coefficients:

```
coef(lm.lasso.final.model)
```

```
## 59 x 1 sparse Matrix of class "dgCMatrix"
```

```
##                                       s0
## (Intercept)                 8.272343e-01
## n_tokens_title             -6.749169e-03
## n_tokens_content           -7.776591e-05
## n_unique_tokens            -2.300623e-03
## n_non_stop_words           -1.316920e-06
## n_non_stop_unique_tokens   -1.050886e-06
## num_hrefs                  -3.358301e-03
## num_self_hrefs              1.035976e-02
## num_imgs                   -1.806391e-03
## num_videos                 -1.801708e-03
## average_token_length        8.743096e-02
## num_keywords               -2.183279e-02
## data_channel_is_lifestyle   7.463212e-02
## data_channel_is_entertainment  2.848076e-01
## data_channel_is_bus         3.539732e-01
## data_channel_is_socmed     -1.883687e-01
## data_channel_is_tech       -7.018656e-02
## data_channel_is_world       1.386809e-01
## kw_min_min                 -3.731377e-04
## kw_max_min                  .
## kw_avg_min                  2.043850e-05
## kw_min_max                  2.728208e-07
## kw_max_max                  5.830397e-08
## kw_avg_max                  3.021636e-07
## kw_min_avg                  1.893881e-05
## kw_max_avg                  4.241157e-05
## kw_avg_avg                 -3.296100e-04
## self_reference_min_shares  -3.511293e-07
## self_reference_max_shares   .
## self_reference_avg_sharess -2.958805e-06
## weekday_is_monday          -7.941440e-02
## weekday_is_tuesday          2.683826e-02
## weekday_is_wednesday        2.365813e-02
## weekday_is_thursday         1.696689e-02
## weekday_is_friday          -1.906564e-02
## weekday_is_saturday         .
## weekday_is_sunday          -9.987756e-02
## is_weekend                 -1.948297e-01
## LDA_00                     -2.739443e-01
## LDA_01                      1.049351e-01
## LDA_02                      3.559063e-01
## LDA_03                      .
## LDA_04                      .
## global_subjectivity        -7.133381e-01
## global_sentiment_polarity   .
## global_rate_positive_words  1.371740e+00
## global_rate_negative_words  .
## rate_positive_words        -2.290075e-02
## rate_negative_words         .
## avg_positive_polarity       1.944225e-01
## min_positive_polarity       .
## max_positive_polarity       .
## avg_negative_polarity       1.340089e-01
```

```
## min_negative_polarity          .
## max_negative_polarity          .
## title_subjectivity            -9.213827e-02
## title_sentiment_polarity      -5.485631e-02
## abs_title_subjectivity        -1.187285e-01
## abs_title_sentiment_polarity   .
```

**2.**

```
# set-up
test.data <- read.csv("~/Documents/MSAN 628/HW 4/OnlineNewsPopularityTest.csv",
    header = T)
test.data$shares <- NULL
test.data$url <- NULL
test.data$timedelta <- NULL
x.test.data <- model.matrix(popular ~ ., data = test.data)[, -1]
y.test.data <- test.data$popular
# standard logistic regression
lasso.probs <- predict(lasso.final.model, type = "response", s = lasso.best.lambda,
    newx = x.test.data)
d2 <- length(y.test.data)
lasso.pred <- rep(0, d2)
lasso.pred[lasso.probs > 0.5] <- 1
mspe.lasso.test <- mean((lasso.pred - y.test.data)^2)
sens.lasso.test <- sensitivity(factor(lasso.pred), factor(y.test.data))
spec.lasso.test <- specificity(factor(lasso.pred), factor(y.test.data))
# bayesian logistic regression
lm.lasso.probs <- inv.logit(predict(lm.lasso.final.model, s = lm.lasso.best.lambda,
    newx = x.test.data))
lm.lasso.pred <- rep(0, d2)
lm.lasso.pred[lm.lasso.probs < 0.5] <- 1
lm.mspe.lasso.test <- mean((lm.lasso.pred - y.test.data)^2)
lm.sens.lasso.test <- sensitivity(factor(lm.lasso.pred), factor(y.test.data))
lm.spec.lasso.test <- specificity(factor(lm.lasso.pred), factor(y.test.data))
# comparison
MSPE <- data.frame(Standard = mspe.lasso.test, Bayesian = lm.mspe.lasso.test)
Sensitivity <- data.frame(Standard = sens.lasso.test, Bayesian = lm.sens.lasso.test)
Specificity <- data.frame(Standard = spec.lasso.test, Bayesian = lm.spec.lasso.test)
MSPE
```

```
##    Standard  Bayesian
## 1 0.3463673 0.3498991
```

```
Sensitivity
```

```
##    Standard  Bayesian
## 1 0.6591885 0.6552108
```

```
Specificity
```

```
##    Standard  Bayesian
## 1 0.6323798 0.6305539
```

We see that the final Bayesian model is actually very close to the standard model as both of these models have similar MSPEs, sensitivities, and specificities. As compared to models trained on the full, unbalanced dataset, we have larger MSPEs (less accurate), but also larger specificities (better at picking up the popular articles). We chose to train our final models on a balanced dataset at the cost of these larger MPSEs in order to get models that actually predict both classes. Given this set-up, our final models attain about ~66% accuracy, which is signficantly better than randomly guessing.

Given that standard logistic regression and Bayesian logistic regression end up performing so similarly, I don't have much of a preference between the two methods. If I had to choose, I would use standard logistic regression since it is more familiar to me (and requires less work!). Although, this might change as I get more comfortable with Bayesian statistics – ultimately, you have more flexibility and control over what you end up implementing.