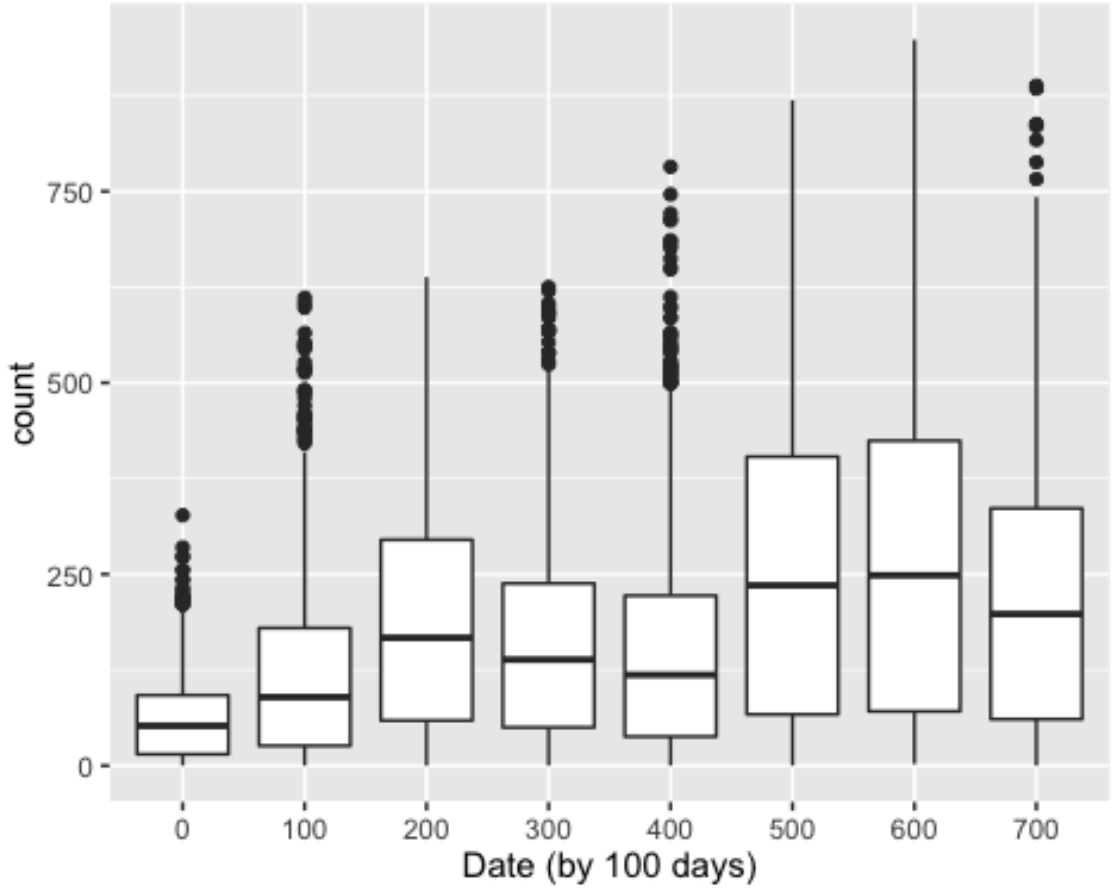
Machine Learning Homework 2

Patrick Miller, Vandana Ramakrishnan, Nathaniel Matare, Ernie Mori, Jordan Bell-Masterson
2/1/2017

To begin our analysis, we spent some time looking over the data and thinking about how we should treat our variables. The first question was - what variables should we treat as continuous and what should we treat as discrete. Some were obvious (we should treat temperature as continuous - 10 degrees is closer to 11 degrees but farther from 20 degrees), but we had some discussion about a few. In particular, whether we treat month as continuous (we tried both in various models - 12 is actually similar to 1, moreso than 8; however, 6 is similar to 7 and 5, so by treating month as discrete you risk losing that commonality). The most interest decision we made was how we treat daylabel. We initially thought to treat the days as factors and let them have some fixed effect, but when graphing the data (see the next section), we found a significant day trend:

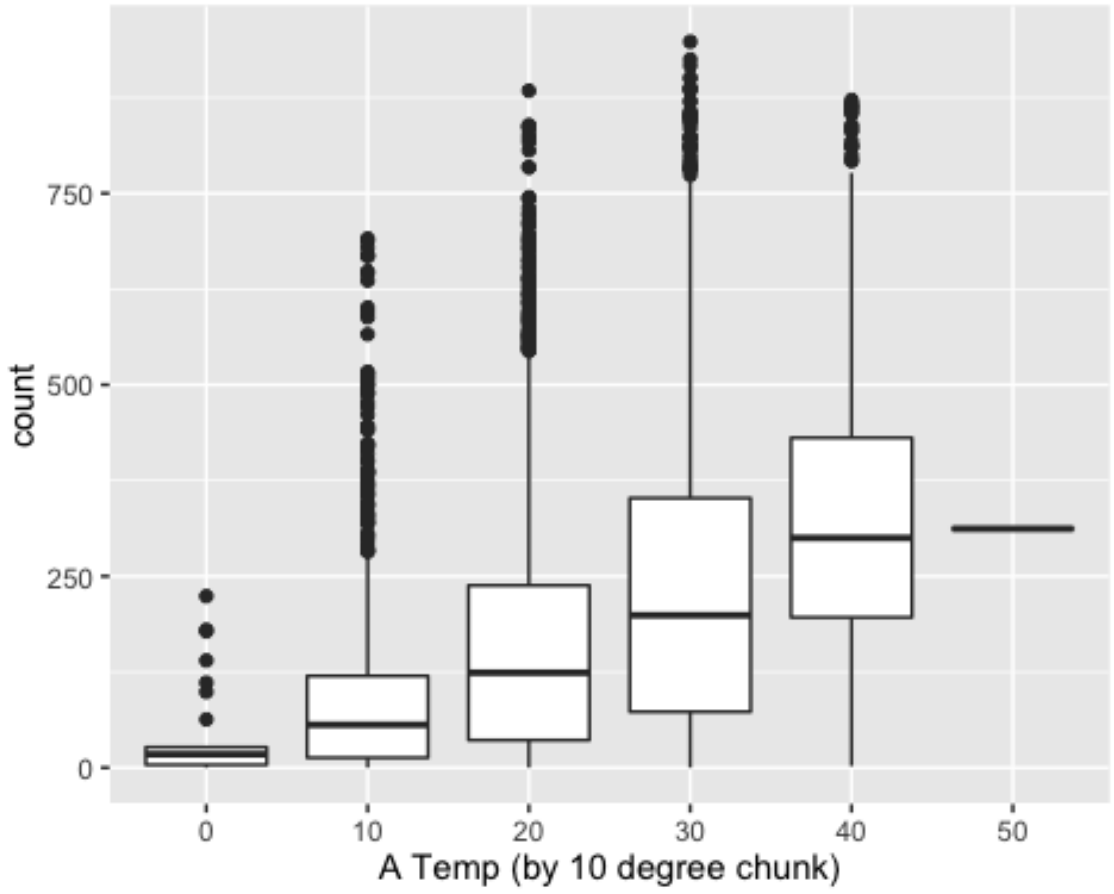
```
ggplot(dt.bike.train,aes(x=as.factor(round(daylabel,-
2)),y=count))+geom_boxplot()+xlab('Date (by 100 days)')
```



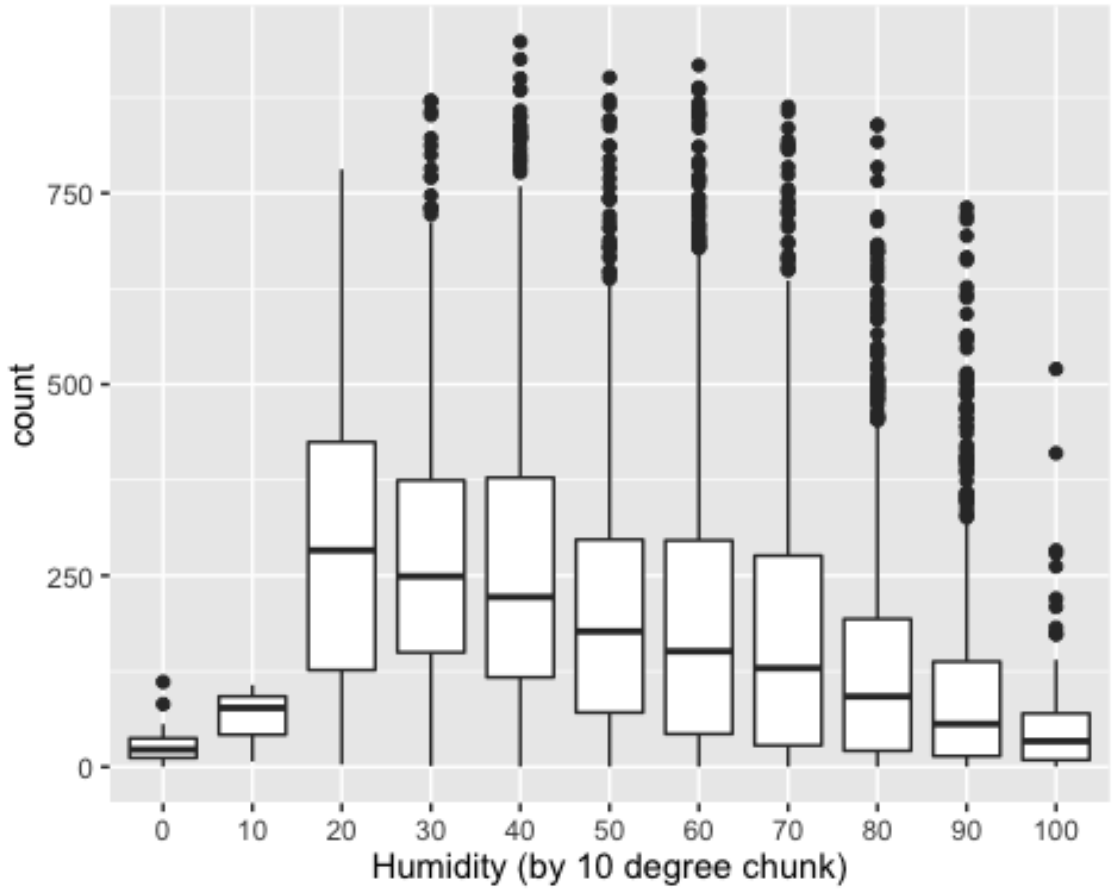
As you can see, there is significant growth over time, both in terms of peak count and average daily count.

We performed similar analysis to look at many of the different characteristics. We've included a few below:

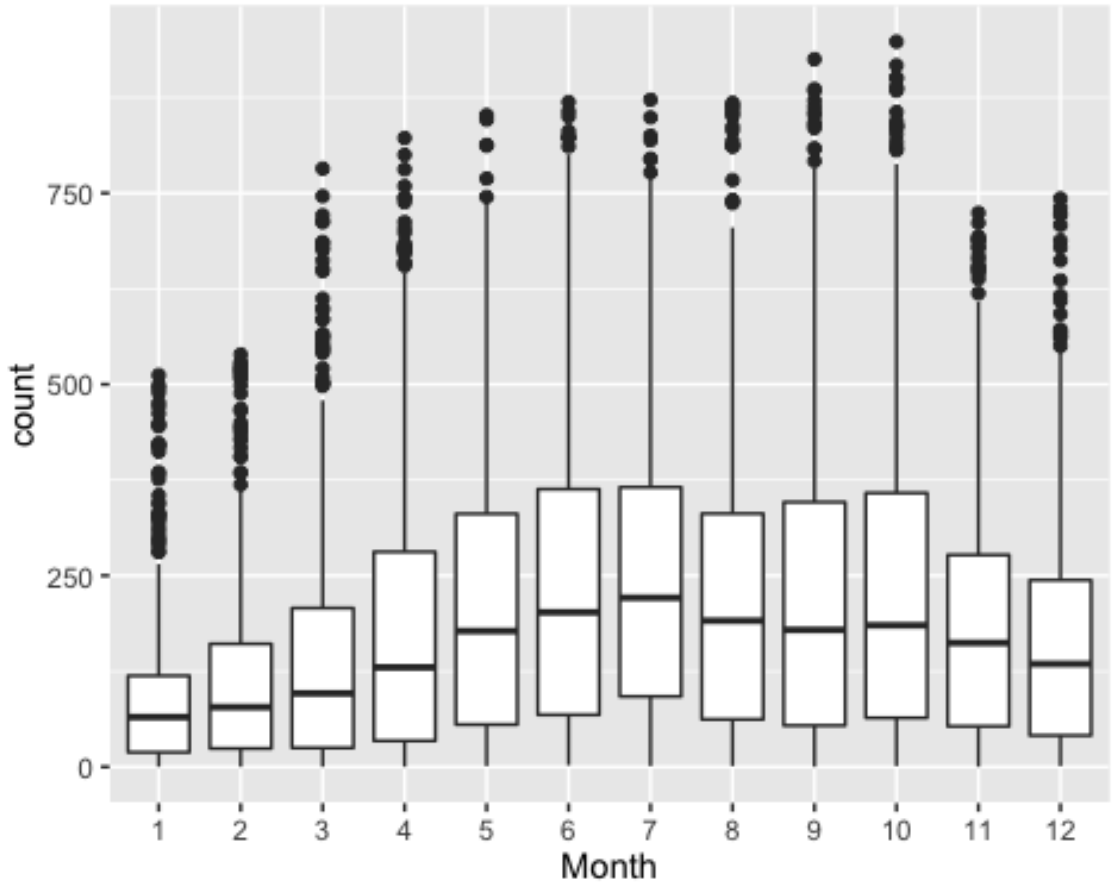
```
ggplot(dt.bike.train,aes(x=as.factor(round(atemp,-
1)),y=count))+geom_boxplot()+xlab('A Temp (by 10 degree chunk)')
```




```
ggplot(dt.bike.train,aes(x=as.factor(round(humidity,-
1)),y=count))+geom_boxplot()+xlab('Humidity (by 10 degree chunk)')
```




```
ggplot(dt.bike.train,aes(x=month,y=count))+geom_boxplot()+xlab('Month')
```



As you can see from the graphs, there are strong relationships with atemp and humidity and there are definitely some seasonal trends with usage falling in the winter. We chose not to remove outliers as we found most of this data generally informative and did not want to bias this sample.

Our next step was deciding/determining whether to predict count or log(count+1). In our initial models we tried both, but quickly discovered that it made little difference for our more machine-learning focused models. Because those models were better (see later sections), we ultimately decided to focus on predicting count. The other factor that influences that decision is that our final judgment is based on the absolute error of our model, not the percentage error. Thus, predicting 250 when the answer is 300 is actually worse than predicting 0 when the answer is 40. Though not all business problems will be this way, focusing on log(count+1), we would be emphasizing the wrong metric.

The last step before going through the models was to think about what type of model would theoretically fit this data best. If we look at some of the data trends, the relationship is not strictly linear, or even really continuous. For example, we saw how higher temperature generally resulted in more biking, but the highest hours were not the highest temperatures. This means that a simple linear model would be unlikely to perform well. In diving into the data more, we found that the biggest hours were hours that were primetime (just after work), weekdays, and good weather days. In fact, there were interesting patterns

like how the count variable would only be high when you saw all three conditions - good weather at 3 am did not really matter. Because these sorts of interactions are so important, hypothesized that some of the more advanced models (trees,random forests, boosting) would be better able to capture these relationships.

For our initial model exploration, we just divided the data into a training and testing set. At the end, once we honed in on which models performed better, we used full cross-validation to assess the best parameters. With all that in mind, we began with a simple linear model.

```
fit_base=lm(log(count+1)~poly(as.numeric(daylabel),3)+season+holiday+workingd
ay+temp+atemp+windspeed,data=dt.bike.train)
summary(fit_base)

##
## Call:
## lm(formula = log(count + 1) ~ poly(as.numeric(daylabel), 3) +
##     season + holiday + workingday + temp + atemp + windspeed,
##     data = dt.bike.train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -4.609 -0.634  0.313  0.881  2.827
##
## Coefficients:
##                                Estimate Std. Error t value
Pr(>|t|)
## (Intercept)                     2.59639    0.06490   40.01 <
0.0000000000000002 ***
## poly(as.numeric(daylabel), 3)1 21.12613    1.58259   13.35 <
0.0000000000000002 ***
## poly(as.numeric(daylabel), 3)2 -1.67945    1.32147   -1.27
0.20380
## poly(as.numeric(daylabel), 3)3  1.85799    1.64874    1.13
0.25981
## season2                        -0.44408    0.05228   -8.49 <
0.0000000000000002 ***
## season3                        -0.94735    0.06753  -14.03 <
0.0000000000000002 ***
## season4                        -0.01443    0.05868   -0.25
0.80582
## holiday1                       -0.09577    0.08663   -1.11
0.26899
## workingday1                    -0.11650    0.03124   -3.73
0.00019 ***
## temp                            0.08815    0.01158    7.61
0.000000000000029 ***
## atemp                           0.01396    0.01021    1.37
0.17152
## windspeed                       0.02281    0.00182   12.55 <
0.0000000000000002 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.28 on 8187 degrees of freedom
## Multiple R-squared:  0.227,  Adjusted R-squared:  0.226
## F-statistic:  218 on 11 and 8187 DF,  p-value: <0.0000000000000002
```

```
y_predict=predict(fit_base,newdata=dt.bike.test)
sqrt(sum((dt.bike.test$count-(exp(y_predict)+1))^2)/nrow(dt.bike.test))
```

```
## [1] 178
```

As you can see, our simple linear model had a RMSE of 178. We didn't expect the linear model to perform well, so we did not spend much more time with it.

We then tried a knn model. However, we ran into issues with reweighting - how should we weight various discrete variables? Should being in the same month be as valuable as being in a similar hour? Did humidity matter as much as our daylable variable in terms of evaluating how "close" a neighbor was? We ultimately built a model using some of the data:

As you can see from the graph, k=4 was best using cross validation. Looking then at the full model on the training set, we could assess the RMSE:

```
kfbest=kknn(y~.,ddf,adjusted.test.x,k=4,kernel = "rectangular")
sqrt(sum((dt.bike.test$count-kfbest$fitted)^2)/nrow(dt.bike.test))
```

```
## [1] 109
```

Which turned out to be 109. Because of the number of variables relative to the size of the data set (and the ambiguity around how we should relatively value the different variables), we expected we could do better with trees/forests.

Our first step was a simple pruned tree:

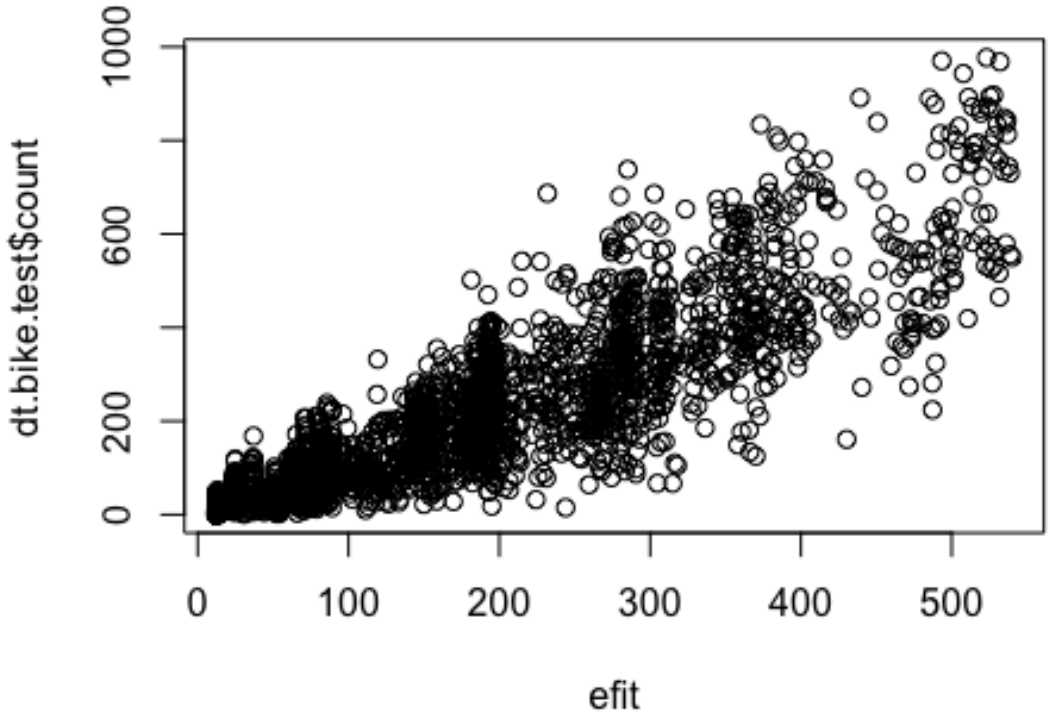
```
new.fit=predict(best.tree,newdata=dt.bike.test)
sqrt(sum((dt.bike.test$count-new.fit)^2)/nrow(dt.bike.test))
```

```
## [1] 56.2
```

As you can see, this tree did much better than any of our previous predictions. It found a RMSE of 56.

That was pretty good, so we went ahead and also looked at bagging and random forests (we have hid the majority of the code because it was crowding out much of the writeup - happy to provide upon request):

```
efit = apply(fmat,1,mean)
plot(efit,dt.bike.test$count)
```




```
sqrt(sum((efit-dt.bike.test$count)^2)/nrow(dt.bike.test))
```

```
## [1] 95.3
```

Bagging did a significantly worse job - with a RMSE of 95, potentially due to the complexity of the data vs the simplicity of the bagging process. For random forests:

```
print(cbind(parmrf,olrf,ilrf))
```

```
##    mtry ntree olrf ilrf
## 1 21.00   100 46.3 46.5
## 2  4.58   100 57.8 58.8
## 3 21.00   500 46.1 45.5
## 4  4.58   500 56.5 56.4
```

As you can see, depending on the inputs, random forests gave RMSE in the range of 46-58.

While random forests clearly show potential, we thought boosting might provide the right mix of complexity and ability to handle the various relationships without wildly overfitting.

```
print(cbind(parmb,olb,ilb))
```

```
##   tdepth ntree   lam   olb   ilb
## 1      4  1000 0.001 118.6 116.4
```

```
## 2     10  1000 0.001 105.2 102.8
## 3      4  5000 0.001  69.2  66.3
## 4     10  5000 0.001  51.6  47.2
## 5      4  1000 0.200  46.5  26.9
## 6     10  1000 0.200  44.9  13.1
## 7      4  5000 0.200  46.4  13.4
## 8     10  5000 0.200  45.7   1.5
```

As you can see, boosting gave the best results, with RMSE as low as 45.

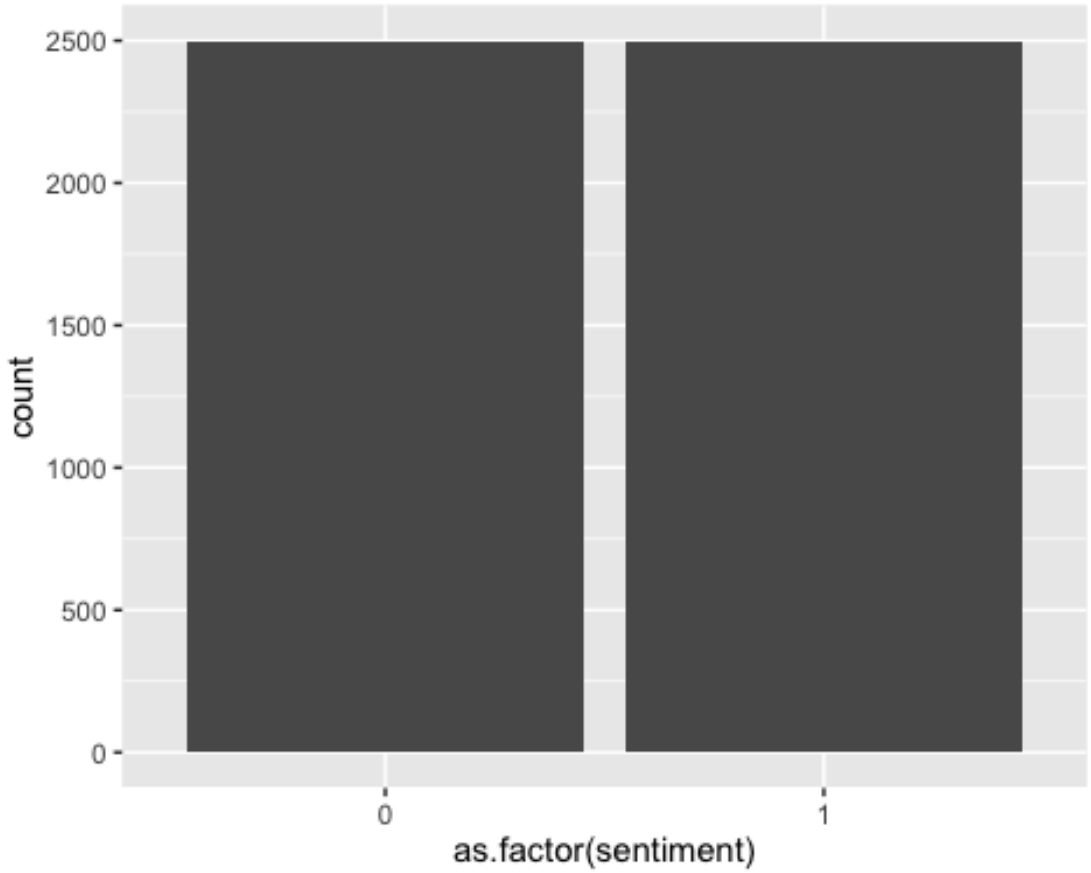
Our last step was then fine-tuning the model and applying it to the sample data. Because we had the best initial luck (and believe the logic is sound) around boosting, we attempted to cross-validate and fine tune our boosting model. This is a difficult balance - we wanted to set all of the parameters (lambda, max_depth) but be careful not to overfit. As a result, we set aside a testing sample (not to be used until the end), then tweaked our parameters using cross validation. The net result then, when we ran with our best set of parameters, was a **RMSE of 38.3** using the parameters gamma=.05 and max_depth=8. We have included that model below:

```
params <- list(gamma = 0.05, max_depth = 8, booster = "gbtree", objective =
"reg:linear") # 41.8

XGBST.cv <- xgb.cv(params = params, data = as.matrix(trainX.fit), label =
as.vector(unlist(trainY.fit)),  nthread = detectCores() - 1, verbose = 1,
nfold = 5, nrounds = 2500) # this takes a while to run: about an hour on the
BoothGrid
yhat.XGBST <- predict(XGBST, as.matrix(trainX.OOS))
XGBST.outsample.RSME <- sqrt(mean((as.matrix(trainY.OOS) - yhat.XGBST) ^ 2));
print(XGBST.outsample.RSME) # OOS RMS
```

Question 2

In looking at this data, our first step was to analyze what the data and see if there were any key trends. For this sort of 1/0 variable prediction, we know our best model options were likely to be various forms of logistic regression or trees. Before we could start to evaluate different models though, we needed to determine two things. First, how should we value our errors (misclassification rate, some sort of weighting, deviance?) and second do we need to make sample size adjustments? To answer these questions, our first step was to characterize the data - what sort of distribution of data did we have:

```
ggplot(movies.train,aes(x=as.factor(sentiment)))+geom_bar()
```



As you can see, the data is perfectly evenly split. Assuming the test data is drawn similarly, we would assume it would be evenly split. This, combined with the fact that we will be judged on misclassification rate, makes a clear case that we should judge our models based on their misclassification rate. This means that for models that generate a probability, we will treat p>.5 as 1 and p<.5 as 0 and then look at the misclassification rate to assess

accuracy. Also, because the data is evenly split, we do not need to worry about selecting a smaller data set or adjusting our models to account for any biases in one direction or the other.

Before we began our model building process, we wanted to estimate what models we believed would perform best. Unlike in question 1, we guessed that there would be more limited relationships between variables - having a "worst" in your review is probably bad and a having a "classic" is probably good, but having both a "classic" and "worst" together wouldn't obviously be any different than the cumulative sum of their individual effects. Thus, we did not know that a tree/random forest model would necessarily ourperform a more simple logistic regression.

On the other hand, the big issue with this data is likely the number of irrelevant columns. The number of "and"s would have no obvious impact on the review. This leads us to believe a model with can properly adjust the impact of columns would likely perform better - something like boosting or a lasso model.

For our initial model exploration, we just divided the data into a training and testing set. At the end, once we honed in on which models performed better, we used full cross-validation to assess the best parameters.

Logistic Regression Model

Our first step was to try a logistic model. Our results were similar to what professor Kolar found.

```
lgfit = glm(sentiment~., dt.movies.train, family=binomial)

temp.predict=predict(lgfit,newdata=dt.movies.test)
lossMR(dt.movies.test$sentiment,temp.predict)

## [1] 0.221
```

As you can see, we found about a 21% misclassification rate. While this isn't terrible, we wanted to see if we could do better with some of the more complex machinery.

Random Forest Model

We build a model a random forest model. There are two parameters:

1. number of steps to use when building each tree
2. total number of trees

We ran for a set of these variables and found a range of misclassification rates:

```
p=ncol(dt.movies.train)-1
mtryv = c(p/2, sqrt(p)/2)
ntreev = c(100,5000)
```

```
(setrf = expand.grid(mtryv,ntreev))  # this contains all settings to try
colnames(setrf)=c("mtry","ntree")
phatL = matrix(0.0,nrow(dt.movies.test),nrow(setrf))  # we will store results
here

###fit rf
for(i in 1:nrow(setrf)) {
  #fit and predict
  frf = randomForest(as.factor(sentiment)~., data=dt.movies.train,
                     mtry=setrf[i,1],
                     ntree=setrf[i,2],
                     nodesize=10)
  phat = predict(frf, newdata=dt.movies.test, type="prob")[,2]
  phatL[,i]=phat
}
```

These models resulted in a range of misclassification rates, but none performed significantly better than the simple logistic regression.

Boosting

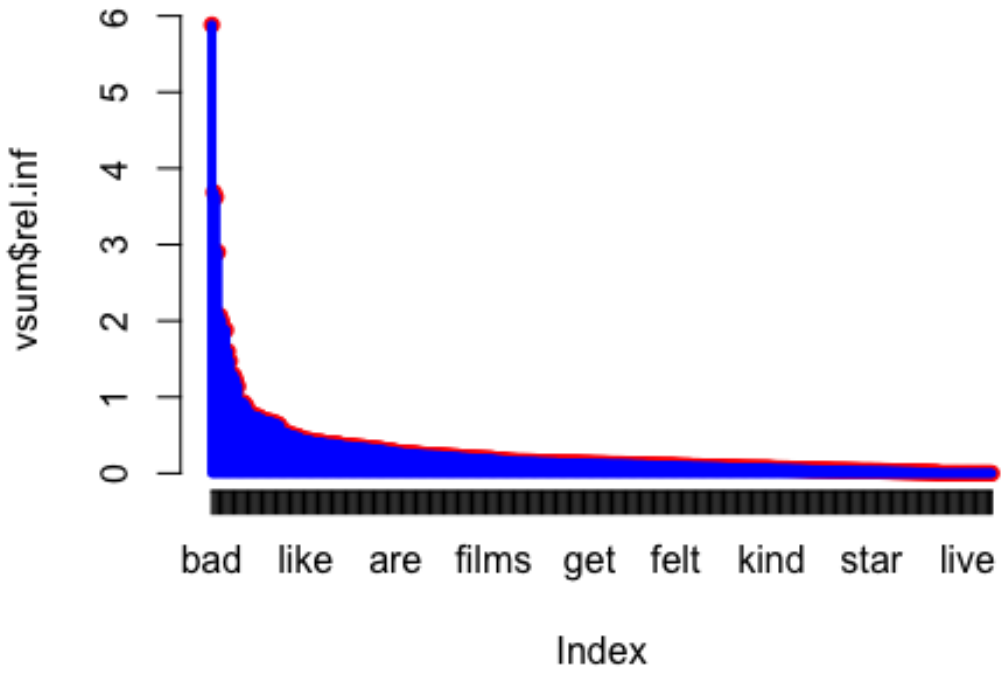
We thought boosting might better adjust for variable importance (something we had guessed would be an issue at the beginning). Again, we looped through a range of values for our parameters for boosting to see if boosting was likely to perform well once we fine-tuned the parameters.

```
for(i in 1:nrow(setboost)) {
  fboost = gbm(sentiment~., data=dt.movies.train, distribution="bernoulli",
               n.trees=setboost[i,2],
               interaction.depth=setboost[i,1],
               shrinkage=setboost[i,3])

  phat = predict(fboost,
                 newdata=dt.movies.test,
                 n.trees=setboost[i,2],
                 type="response")

  phatL[,i] = phat
}
```

The boosting model still gives us an error of 20.4% in the best case. This surprised us, so we wanted to dig more - we had thought results would be better. To look into the boosting model, we then looked at variable importance:

```
boostfit = gbm(sentiment~.,data=dt.movies.train,
               distribution='bernoulli',
               interaction.depth=4,
               n.trees=500,
               shrinkage=.2)
```

```
vsum=summary(boostfit, plotit=F)
p=ncol(dt.movies.train)-1
plot(vsum$rel.inf,axes=F,pch=16,col='red')
axis(1,labels=vsum$var,at=1:p)
axis(2)
for(i in 1:p) lines(c(i,i),c(0,vsum$rel.inf[i]),lwd=4,col='blue')
```



As you can see, our model gives importance to many of the variables. Let's zoom in to the top few and look at the relative values:

```
vsum=vsum[1:20,]
plot(vsum$rel.inf,axes=F,pch=16,col='red')
axis(1,labels=vsum$var,at=1:20)
for(i in 1:20) lines(c(i,i),c(0,vsum$rel.inf[i]),lwd=4,col='blue')
```


```
print(vsum)
```

```
##                 var rel.inf
## bad             bad   5.884
## length       length   3.686
## worst         worst   3.623
## great         great   2.902
## awful         awful   2.067
## best           best   1.996
## the             the   1.893
## waste         waste   1.879
## and             and   1.599
## wonderful wonderful   1.473
## perfect     perfect   1.304
## boring       boring   1.284
## terrible   terrible   1.222
## excellent excellent   1.140
## love           love   0.937
## minutes     minutes   0.931
## this           this   0.926
## script       script   0.892
## poor           poor   0.839
## was             was   0.773
```

While some of these values make sense (bad, wosrt, great, maybe even length), a few are surprising. Why are "the" and "and" important? We had two hypotheses.

1. These variables were indicative of length, and the model was ascribing some of the importance of the length variable to the "the" and "and" variables which are correlated with length.
2. There is something important about the "the" and "and" variables - these values might indicate a certain type of review or a certain reviewer (maybe more formal reviewers use more "the"s and also give more positive reviews?)

Adjusting the data

To test these competing theories we reran the models in two ways. First, we tried normalizing the data by length. That is, we adjusted each word count to be a fraction, word count/length. In theory this would better normalize the data and we would judge reviews by word frequency rather than word count.

```
dt.movies.train.adj=dt.movies.train[,2:391,with=FALSE]/dt.movies.train$length
dt.movies.train.adj$length=dt.movies.train$length
dt.movies.train.adj$sentiment=dt.movies.train$sentiment
```

However, when we reran our models using these adjusted values, the model consistently performed worse.

Second, we tried eliminating the so called "stopwords" from the regression. These are small, generally irrelevant words for meaning parsing. Thus, we wanted to see if removing them from the models would make for a more "true" and hopefully accurate model.

```
library(tm)
new.names=colnames(dt.movies.train)[!(colnames(dt.movies.train) %in%
stopwords("english"))]
dt.movies.train2=dt.movies.train[,new.names,with=FALSE]
```

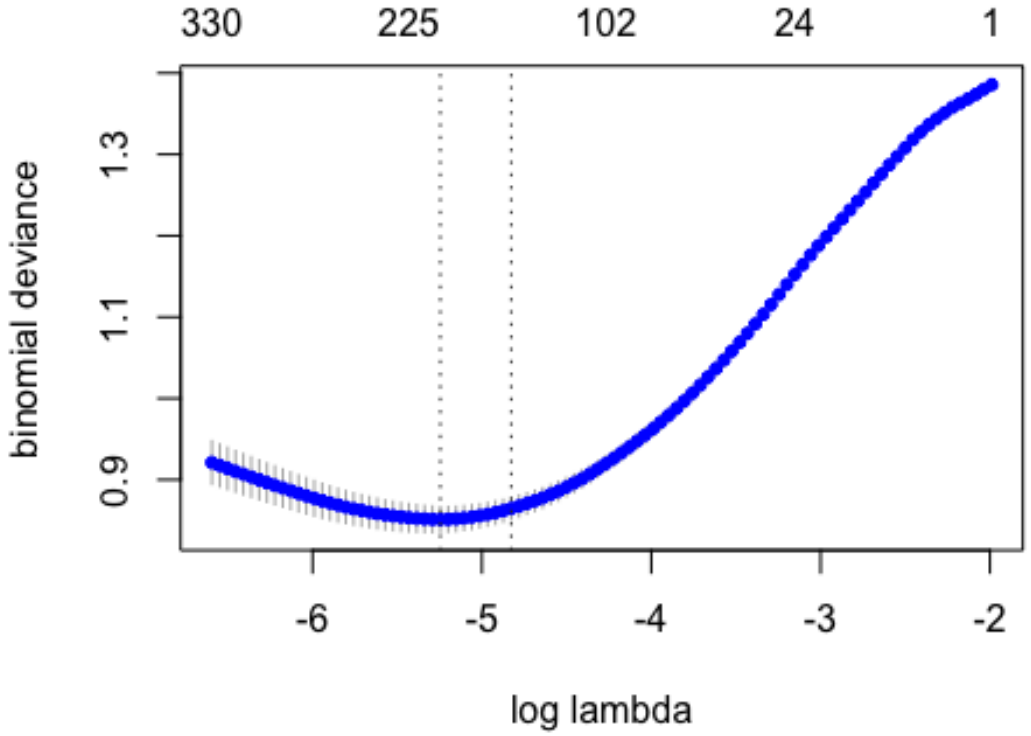
However, again this resulting in less accurate models. This result was not as surprising given that many of our models would theoretically account for having irrelevant variables and adjust accordingly. Thus, eliminating variables which might provide some relevant information could only hurt accuracy.

Either way, both attempts failed, leading us to conclude that there must be some relevance with these "and" and "the" variables.

Lasso

To deal with the high number of variables but hopefully maintain the same benefits we saw with the initial logistic model we built a LASSO model. This should better with the relative importance of the different variables.

```
gamlr.model=cv.gamlr(dt.movies.train.without.sent,dt.movies.train$sentiment,f
amily="binomial")
plot(gamlr.model)
```




```
newpredict=predict(gamlr.model,newdata=dt.movies.test,type="response")
lossMR(dt.movies.test$sentiment,newpredict)

## [1] 0.209
```

This model again gave us a model with around a 20% misclassification rate. As you can see from the plot, our CV lasso uses between 102 and 225 variables (depending on the exact specifications we use). This aligns with the earlier significance levels we saw in our boosting model.

Improving the model with increased dimensionality

With none of these models making any significant improvement over the initial values, we wanted to try something more advanced. We tried binning numeric features into distinct groups: we tried binning them into 5, 10, 20, and 30 groups. We settled on 30 distinct groups. The intuition being that we're able to account for potential skew and hetroskedastictiy in the feature space. We also differenced columns to extract more features. So, we wrote a function (binCols) that does this. We focused on the length column.

```
    binCols <- function(target, bins, data = all.data){
                    binned.col <- cut(as.matrix(data[ ,target, with =
FALSE]), bins, include.lowest = TRUE, labels = paste(target, ".bin.", 1:bins,
sep = ""))
                    return(binned.col)
    }

    all.data <- cbind(all.data, lengthAsbin = binCols(target = 'length', bins
= 30))



    toDouble <- function(data){ # transforms data from integer to double
                    ind.num <- names(which(sapply(data, is.numeric)))
                    ind.factor <- names(which(!sapply(data, is.numeric)))

                    data.double <- apply(data[, ind.num, with = FALSE], 2,
as.double)
                    dummies <- model.matrix(~ lengthAsbin, naref(data[,
ind.factor, with = FALSE]))[ ,-1] # currently not adaptive

                    data.out <- cbind(data.double, dummies)
                    return(data.out)
    }

    trainX.fit <- toDouble(trainX.fit)
```

After we made those changes, we reran through our same models and found that this significantly improved the LASSO model. Our final model had an out of sample error rate between **15.8% and 21%** depending which data was used for training and testing; that is, the randomness of the CV and test set. You can see the code below.

```
trainOOS.idx <- sample(1:NROW(trainX), NROW(trainX) * 0.20) # sample x% of
the training data in order to get a true OOS estimate

trainX.fit <- trainX[-trainOOS.idx, ] #use these for true testing and
evaluating

trainY.fit <- trainY[-trainOOS.idx, ]

trainX.OOS <- trainX[trainOOS.idx, ]

trainY.OOS <- trainY[trainOOS.idx, ]

 LASSO <- cv.gamlr(trainX.fit, trainY.fit, family = 'binomial', gamma = 0,
verb = TRUE, nfold = 10) # 10 fold cross-validation
    prob.LASSO <- predict(LASSO, trainX.OOS, type = 'response') # 20.1 % w/
binning length 15.8% - 21% miss classification; depends on randomness from
CV, and random test sample
    loss.LASSO <- lossMR(trainY.OOS, prob.LASSO)
    print(loss.LASSO)
```

```
#15.8% best up to 21% worst
```