Using Data to Predict Interest in NYC Apartments

Trevor Wexner

Stat 471

04/29/2017

Professor Zhao

Contents

I: Executive Summary

This project was done to compete in the Kaggle Data Science competition hosted by an online real estate company called RentHop.com. The aim of the competition was to predict interest a given property listed on the website. Specifically, entrants competed to build the model that most accurately predicts interest in an apartment from that property's listing information and pictures[1]. Competitors were provided a dataset of 49352 listings in New York City to build their models. For each apartment, 15 variables containing associated listing information and pictures were given[2].

After data cleaning and preparation, I tried using four methods to predict interest. I excluded photos because my computer was unable to store their corresponding file sizes. For all of these algorithms, I built separate models for each possible apartment size and aggregated the results. I then submitted my most accurate model to the competition. My final model had a misclassification rate of 29.6% out of sample and placed 2155 out of 2489. While this model was better than a trivial guessing procedure, it was still not optimal. This non-ideal accuracy and low placement are likely due to the fact that photos, which are a key element of the decision process of looking for an apartment, were excluded from my analysis.

II: Problem Definition

An interesting area in which data science is being applied is real estate. Specifically, companies and websites can now leverage online data to more accurately anticipate the demand for a given property. In turn, these websites can list the more desirable properties first and make sure that the first listings that users see are the ones they're most likely to pursue. Thus, the website will be maximizing its potential leads and sales, and, therefore its revenue.

This revenue optimization through the analysis of website data was precisely the goal of the RentHop Data Challenge. By accurately predicting interest in a given property from its listing information, RentHop could more effectively order its listings. As such, the goal of this project was to build the most accurate possible model to predict interest in a property. Specifically, this project aimed to accurately classify properties from RentHop.com as "high", "medium", or "low" interest.

[1] Here "interest" is measured by the number of inquiries a given listing receives

[2] For a full description of the competition, see https://www.kaggle.com/c/two-sigma-connect-

[2] For a full description of the competition, see https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries

While there are many possible criteria with which to evaluate accuracy, only the multi-class logarithmic loss was used for final model selection in this project[3]. Since this metric aims to minimize incorrect predictions, misclassification rate was also used as criteria for preliminary model comparison.

Final results were evaluated for the competition by having entrants run a testing set through their models and submitting the corresponding class probabilities. The multi-class log loss was then computed on these probabilities, and the lowest score won.

III: Description of the Data

The (training) data analyzed consisted of 49352 apartment listings from RentHop.com in Manhattan, Queens, and Brooklyn. For each property, the following information corresponding to the following 15 variables was provided:

Variable Name	**Description**
bathrooms	Number of Bathrooms
bedrooms	Number of Bedrooms
building_id	Building id on renthop.com
created	Timestamp the listing was created
description	Description of the property, as it appears on the website
display_address	Address, as it appears on the website
features	A list of features about the apartment
latitude	Latitude of the apartment
listing_id	Listing id on renthop.com
longitude	Longitude of the apartment
manager_id	Manager id on renthop.com
photos	A list of photo links
price	Price of rent, in USD
steeet_address	Full street address
interest_level	Target variable. Has levels "high", "medium", and "low", based on the number of inquiries a given property gets

All listings considered were created between April 1st and June 29th of 2016. The average rent for a property was $3830 per month, with an interquartile range of $2500 to $4100. Additionally, most listings were for 1 bathroom and 1 and 2 bedroom units. The vast majority of properties (69.5%) of properties were low interest, with 22.5% being medium interest and only 7.8% being high interest.

[3] Multi-Class Logarithmic Loss was the metric used by RentHop to rank the models of all competitors. It penalizes models based on the log of their predicted probabilities that a given point belongs to the incorrect class

Exact distributions of *bathrooms, bedrooms, price,* and *interest_level* are shown below.

Distribution of *Bathrooms*

0	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5	6	6.5	7	10
313	39422	645	7660	277	745	70	159	29	20	5	4	1	1	1

Distribution of *Bedrooms*

0	1	2	3	4	5	6	7	8
9475	15752	14623	7276	1929	247	46	2	2

Distribution of *Price*

Min.	1st Qu.	Median	Mean	3rd Qu.	Max
43	2500	3150	3800	4100	4490 e3

Distribution of *interest_level*

Low	Medium	High
.695	.228	.078

Distribution of *interest_level* by number of bedrooms

	low	medium	high
Studio	0.688	0.223	0.089
1br	0.744	0.196	0.06
2br	0.67	0.24	0.089
3br	0.65	0.269	0.081
4br	0.639	0.284	0.077
5br	0.984	0.008	0.008
6+br	0.94	0.06	0

We can see that apartments with 4 or more bedrooms were far less likely to have high interest than those with 3 or less.

IV: Methods and Results

Data Cleaning and Preparation

Initial Cleaning

Prior to any meaningful analysis being performed, some data cleaning and further preparation needed to be done to transform the data into a useful form.

First, *building_id*, *listing_id,* and *manager_id* were removed from the dataset, since each of these took on too many values to yield any meaningful insight during analysis and would be computationally expensive to include. Additionally *photos* were not used because the corresponding file was too large to download and store on the computer used for analysis[4]. Thus, this variable was discarded from the dataset.

Converting Date-Time Values

Next, because *created* was a unique date-time timestamp for each observation, these values needed to be converted prior to use in analysis. As such, *created* was separated into the time of day, day of week, and time of month when a listing was posted. The original variable was then removed.

Converting Location

Similarly, there were too many addresses, and longitudinal and latitudinal coordinates to be immediately used. To standardize and more effectively compare the locations of various properties, the postal code (*zipcode)* and distance to the nearest subway (*dist.to.subway)* for each listing were added[5]. *Display_address, street_address, latitude,* and *longitude* were all then removed.

Splitting the Dataset by Number of Bedrooms

At this point the only remaining necessary data preparation involved transforming *features* and *description*. However, both of these variables were text-based and had too many possible values to be used immediately. Thus, text mining needed to be performed to cut uninformative words. Before such analysis could be performed, data needed to be separated into testing and training sets, where only the training set would be used for text mining.

Before this testing-training split could occur, there was one more technicality to address. If we are trying to build a model that imitates the decision process of perspective renter looking for an apartment, then it is incorrect for us fit just one model predicting interest in units of all sizes. Rather, someone looking for an apartment searches for properties with a set number of bedrooms. So, because this decision process is distinct across apartment sizes, we must build distinct models based on the number of bedrooms in a given property. As such, we split our original dataset into seven separate datasets based on number of bedrooms. 0,1,2, 3, 4, and 5

[4] The inclusion of these photos would undoubtedly improve results. See "V: Conclusions" for further discussion

[5] Both *zipcode* and *dist.to.subway* were computed from the provided longitude and latitude of each apartment. For exact implementation of these calculations, see Appendices 2 & 3. The referenced datasets for coordinates of postal codes and subway stops were provided by Google Maps (via gaslampmedia.com) and data.ny.gov, respectively. See bibliography for details

bedrooms were all considered distinctly. Due to the small numbers of observations of listings with 6 or more bedrooms, these were combined into one dataset.

For each of these datasets, 70% of observations were randomly selected for training, with the remaining 30% reserved for testing.

Text Mining: Features

Initially, 1104 possible features appeared in the dataset. However, not all of these features would contribute to the predictive power of the model and would interfere with computational efficiency. Thus, I aimed to reduce the number of features input to the model to include only those that contributed meaningfully to prediction. First, I eliminated all words appearing less than one percent of the time, as these likely occurred too infrequently to have impact[6]. Then, backwards elimination on a multiple regression model was performed to remove all features that were not significantly predictive of *interest_level* at the 5 percent level. Backwards elimination was used for its effective ability to reduce the size of the dataset to small dimension. Multiple-regression was chosen for its computational simplicity. Given that this step was solely for data preparation, algorithms with longer runtimes were not used (e.g. logistic regression). The regression was valid because *interest_level* is simply a binned continuous variable – the number of inquiries a given property receives. So changing its values from "high", "medium", and "low" to 0, 1, and 2 and using continuous models is a valid procedure. After backwards elimination was performed, 33 features remained for analysis. These features can be found in Appendix 4.

Text Mining: Descriptions

Similarly, *descriptions* needed to be transformed before model construction. Initially each listing had its own unique description, making it impossible to gain insight from the variable. Thus, text mining was performed to identify key words included in descriptions that were predictive of *interest_level*. Again only words occurring at least 1% of the time were considered and *interest_level* was treated as a numerical variable. LASSO was run with lambda tuned to "lambda1se". After this procedure 397 of the original 61487 words were included.

Final Dataset for Modeling

Having all of the data preparation complete, there were 438 predictor variables were considered for model construction: *bathrooms, bedrooms, price, interest_level, dayOfMonth, dayOfWeek, timeListed, zipcode, dist.to.subway,* 33 features, and 397 keywords from descriptions. The target variable remained *interest_level.* A complete version of the dataset is attached separately.

[6] Before analysis, words from both *features* and *description* were standardized and cleaned by being converted to lower case and having non-content words and punctuation removed

Methods Used

Five models were used in trying to predict interest level.

Random Forest for Classification

The first model used was random forest for classification. Separate random forests were built for each apartment size (by number of bedrooms). For computational simplicity, parameter tuning was performed only on the random forest built for studio apartments, and optimal parameters were taken to be the same across all apartment sizes. Mtry was tuned to be 21 and ntree was set to 100 in all cases. (See Appendix 5 for details and plots about tuning).

Random Forest for Regression

Trying to account for the ordinal relationship of *interest_level*, a random forest for regression was also constructed for each apartment size. The parameters were tuned to the same values as the random forests above, due to the similarities in the input datasets. Since logloss is computed using probabilities, I needed to invent a metric acting as a probability proxy that was based on the output of the regression. I defined the "probability" of being in a certain class as:
Let $\hat{y} = predicted\ interest\ s.t.\ \hat{y} \in [0, 2]. Let\ i\ \in \{0, 1, 2\}$. Then

$$P(interest_{level} = i) = \frac{\left(\sum_{j=0}^{2}|\hat{y} - j|\right) - |\hat{y} - i|}{2 \cdot \sum_{j=0}^{2} |\hat{y} - j|}$$

In other words, we take the probability of interest being a given class as proportional to how far away the predicted interest value is from a that class relative to the other classes. Thus, we take the predicted class to be the class (i.e. 0, 1, 2) to which the predicted interest is closest.

Reducing Dataset Further for Input Into Logistic Regression

The next model attempted was logistic regression. However, before attempting to fit the model, more variables were cut from the dataset (due to the computational complexity of fitting 7 separate logistic regressions with 438 predictors). PCA was attempted, but failed to substantially explain the variability of the data in a reduced number of dimensions (see Appendix 5 for plots).

Then, LASSO was performed, (again taking interest to be numerical for efficiency). Lambda was set to lambda1se. First, 7 separate LASSO's were run for each of the possible apartment sizes. Some of the models still yielded over 300 predictors. Thus, I instead used one LASSO with all of the apartment sizes. This dimensionality reduction still gave 378 predictors, so I then used backwards elimination with linear regression separately for each apartment size and kept only those variables that were significant at the 5% level. The resulting datasets had between 44 and 89 variables included for input into logistic regression.

Multi-Class Logistic Regression

A multi-class logistic regression was then fit for each of these reduced datasets (separated by apartment size).

Ordinal Logistic Regression

Because *interest_level* is ordinal, an ordinal logistic regression was then fit for each apartment size, again using the reduced inputs.

Results

The remaining 30% of the original dataset reserved for testing was run through each model. First, misclassification errors of the various models were compared to get a preliminary comparison of model performance. Results are shown below.

Method	**RF Classification**	**RF Regression**	**Multinomial Logit**	**Oridnal Logit**
Testing Error	0.296	0.308	0.319	0.316

We can see that, based on misclassification error out of sample, random forest for classification was the most accurate model, followed by random forest for regression, ordinal logistic regression, and multinomial logistic regression. A possible explanation for the superior performance of the random forests is that they had more variables as inputs than the logistic regressions. Still, the performance of all models was very similar, and yielded misclassification rates of roughly 30%-32%.

For a more detailed comparison, misclassification errors of each model on each apartment size were compared.

	RF Classification	RF Regression	Multinomial Logit	Oridnal Logit
Studio	0.305	0.315	0.324	0.319
1br	0.252	0.255	0.259	0.258
2br	0.318	0.333	0.346	0.341
3br	0.328	0.363	0.369	0.366
4br	0.362	0.375	0.43	0.433
5br	0.012	0.012	0.024	0.055
6+br	0.071	0.071	0.107	0.214

Again, random forest for classification was the most accurate across all bedroom sizes, and the relative performances of all models were roughly the same.

Lastly, given that the goal of the competition was to submit predictions from a model with the lowest log-loss, the log-losses of various models were compared.

Method	RF Classification	RF Regression	Multinomial Logit
Log-Loss	0.705	0.902	0.823

Given that the ordinal logistic regression had a consistently higher misclassification rate than the random forests, its log-loss was not computed. Again we see that the random forest for classification was the most accurate model. In this case random forest for regression had a higher log-loss than the logistic regression, likely because there was no true probability computed, but rather a proxy was used.

By all measures, the random forest for classification was the most optimal model. It's confusion matrix and positive prediction rates are shown below:

Confusion Matrix

Predicted\Observed	low	medium	high
low	23093	6739	2013
medium	714	1019	489
high	69	185	178

Positive Prediction Rates

Interest Level	low	medium	high
Pos. Prediction Rate	0.725	0.459	0.412

Since the random forest for classification was the most accurate model, this was the model used in the competition. The provided testing set of 74659 observations was then passed into the random forest, and predictions were submitted. A log-loss of .710 was obtained, and awarded 2155 out of 2489 submissions. The winning submission achieved a log-loss of .492.

V: Conclusions

In the end, my best model was a a random forest for classification. It is possible that this model was the best because there is no true well-behaved functional form of *interest_level*, as logistic regression assumes. Still, the random forest was not extremely accurate. It had a 29.6% misclassification rate out of sample. This error rate is very high, especially considering that, if we simply guessed "low" for the *interest_level* of every property, we would have had an error rate of 30.8%. Still, our model was able to classify high and medium interest properties with positive prediction rates of 45.9% and 41.2%, respectively, making it superior to a trivial guessing rule. Still, these positive prediction rates are not very high, and the model's accuracy likely suffered from several simplifying procedures taken.

Perhaps the biggest barrier to the accuracy of the model was the fact that photos were not included in the analysis. The pictures for a given listing are a key component in a person's decision of whether or not to inquire about a given apartment, so excluding them prevented my analysis from truly modeling someone's interest while searching the site.

Additionally, many simplifying procedures were taken for the purposes of computational simplicity. Perhaps using a continuous scale for interest was an oversimplifying assumption, and better results would have been instead cutting variables with a more computationally expensive classification algorithm. Similarly, the dataset may have been too simplified prior to analysis and too many variables were cut. This was likely the case with input into the logistic regression models, where only 40-89 variables were included for analysis.

Ultimately, my best model proved superior to a trivial model, but still less than ideal. Given the simplifying assumptions made and exclusion of photos, this sub-optimal performance and lower placement in the competition make sense. A model using photos and more variables and complex models, such as a neural net, would likely have obtained better results.

VI: Appendices

Contents

1: Bibliography

(a) *RentHop Challenge and Data:* https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries
(b) *Postal Code Data*: geocoded with Google Maps and accessed via https://www.gaslampmedia.com/download-zip-code-latitude-longitude-city-state-county-csv
(c) *Subway Location Data*: New York State Office of IT Services. https://data.ny.gov/Transportation/NYC-Transit-Subway-Entrance-And-Exit-Data/i9wp-a4ja

2: Calculation of Zip Codes

To convert longitudinal and latitudinal coordinates to zip codes, the following procedure was done. The coordinates of each zip code provided in the dataset only gave one point (presumably the center) of each zipcode. So, the zipcode of a given property was taken to be the zipcode centered at the closest longitudinal and latitudinal coordinate pair.

3: Calculation of Distance to Subway

The distance to the subway for a given property was taken to be the Euclidean distance between the coordinates of the nearest subway stop (in longitude and latitude) and the coordinates of the property.

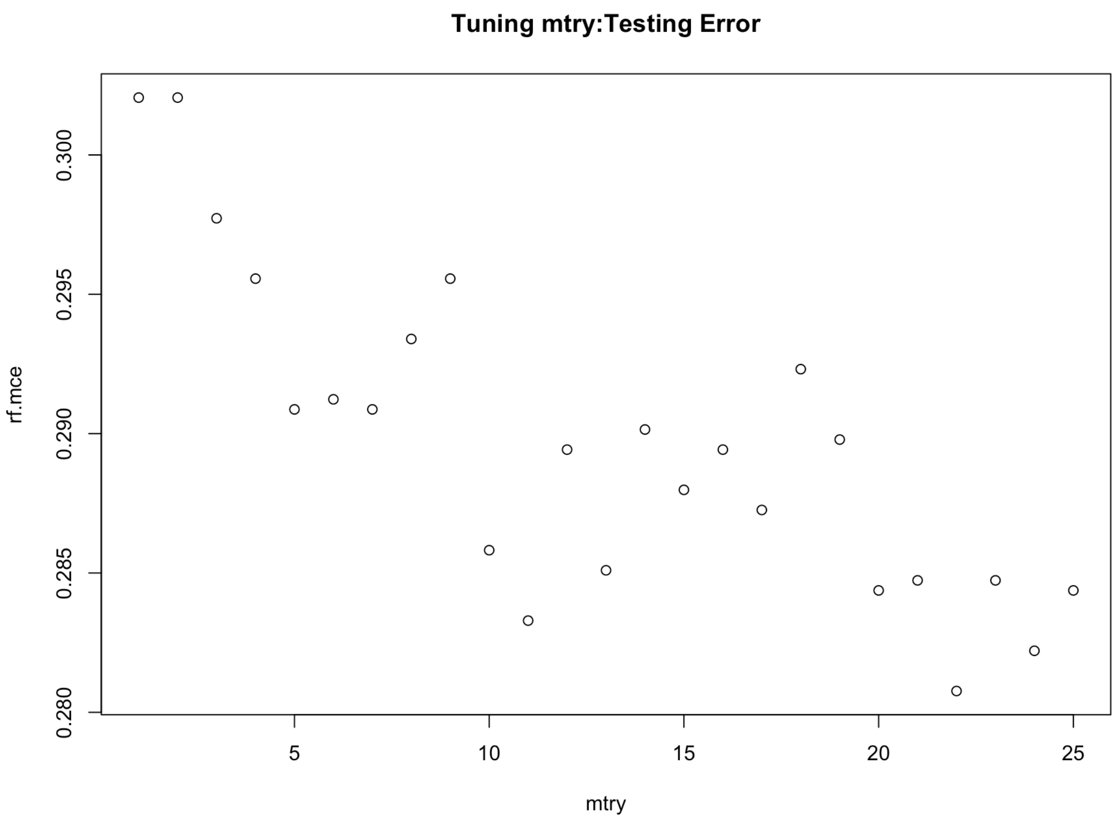
4: Features Included in the Analysis

"access"	"allowed"	"building"	"cats"	"ceilings"	"center"
"common"	"deck"	"dogs"	"doorman"	"exclusive"	"fee"
"fireplace"	"fitness"	"floors"	"furnished"	"gardenpatio"	"hardwood"
high"	"internet"	"multilevel"	"null"	"outdoor"	"prewar"
"private"	"reduced"	"renovated"	"roof"	"space"	"super"
"terrace"	"unit"	"wheelchair"			

5: Parameter Tuning

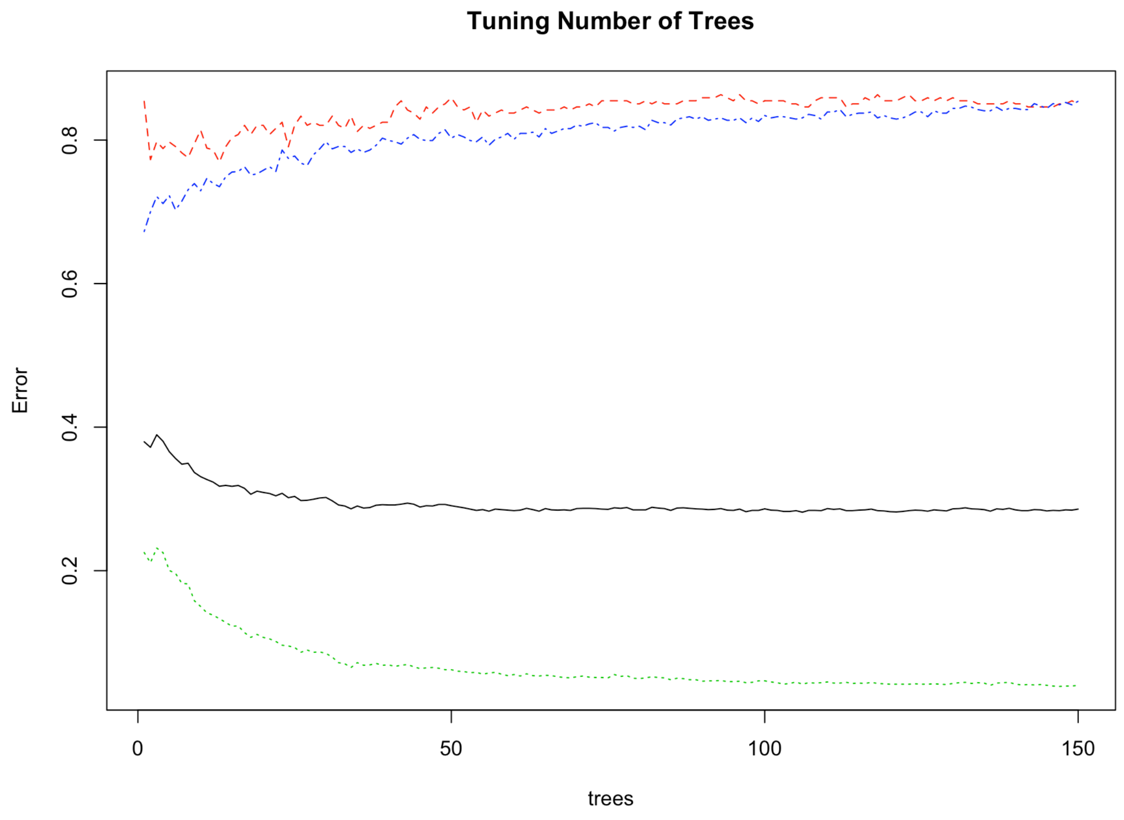
Random Forest

Tuning mtry



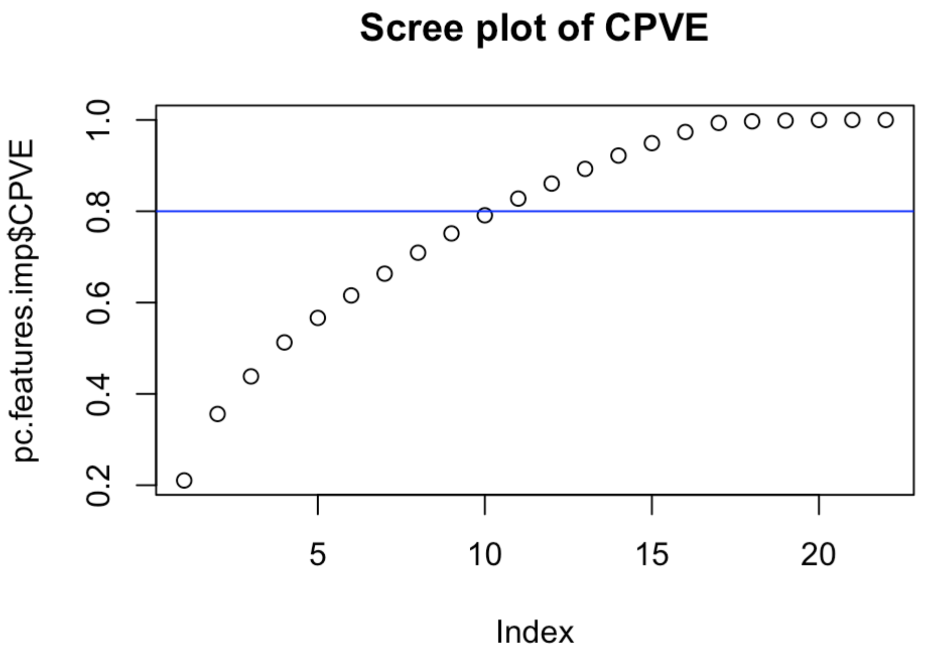
Since the default value of mtry was 21, I varied mtry from 1 to 25. The optimal value of mtry turned out to be 21. We can see that this corresponds to the minimum testing error.

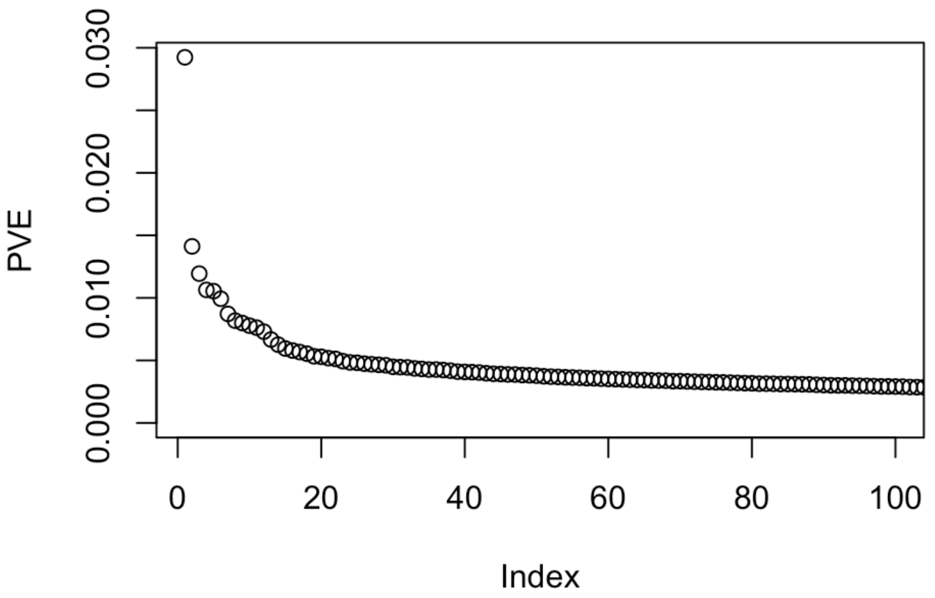
Tuning ntree



The plot above shows that testing error plateaus at around 100 trees. So, ntree = 100 is sufficient.

PCA





We can see that there is no clear threshold where a certain number of dimensions explains a substantial portion of the variance, as adding more components seems to continually increase the variance explained at a steady rate.

6: R Code

```
# Script for RentHop Challenge
#install.packages('rjson')

# Plan:
#   1st, we will exclude pictures & descriptions. Then, we will add the two
#   (1) Fit a multinomial logistic regression --> do LASSO or PCA first to reduce
dimension of data (maybe)
#   (2) Fit a random forest for classification
#   (3) Fit a nearest neighbors analysis
#   (4) Fit a support vector machine

# load data
library(jsonlite) #loading json
library(lubridate)
library(ggmap)
library(tables) #table formatting
library(tm); library(SnowballC)
library(glmnet);library(randomForest)
library(nnet) #multinomial logistic regression
library(car)
library(ModelMetrics) #multiclass log loss
library(ordinal) #ordinal glm

json.file <- "/Users/TrevorWexner/Downloads/train.json"
data.train <- fromJSON(json.file)

# PART 0: EDA

# get info on data
class(data.train) # list
varnames <- names(data.train)

# will add pictures back in later, so remove photos from the list
data.train <- subset(data.train, ! names(data.train) == "photos")
varnames <- names(data.train)

#unlist all variables except for "features"
data.train[!names(data.train) == 'features'] <- lapply(subset(data.train,
!names(data.train)
                                        == 'features'), unlist)

#get variable summaries for bedrooms, bathrooms, display_address, & interest level
lapply(subset(data.train, names(data.train) %in% c("bedrooms", "bathrooms",
"interest_level")), table)
```

```
#get summary stats for price
summary(data.train$price)

# a quick EDA about bathrooms, bedrooms, price, and created
table(data.df$bathrooms);
table(data.df$bedrooms)
summary(data.df$price); boxplot(data.df$price)
summary(dates)
round(table(data.df$interest_level)/49352, 3)


#### DATA PREPARATION #############

# CREATED
# Seperate "created" into day of week, time of month, and time of day
# convert "created" to a date variable
nobs <- length(data.train$created)
dates <- unlist(sapply(data.train$created, strsplit, split = " "))[seq(1, 2*nobs, by =
2)]
dates <- strptime(dates, format = '%Y-%m-%d')
# only 30 dates are available on this dataset: 2016-05-31 through 2016-06-29

# get times
times <- unlist(sapply(data.train$created, strsplit, split = " "))[seq(2, 2*nobs, by =
2)]
# convert times to a vector of h + mins/60 throughout the day
times <- unlist(sapply(times, strsplit, split = ":"))
hrs <- as.integer(times[seq(1, length(times), by = 3)])
mins <- round((1/60) * as.integer(times[seq(2, length(times), by = 3)]), 3)
timeListed <- hrs + mins

# day of month and day of week
dayOfMonth <- day(dates)
dayOfWeek <- weekdays(dates)

# replace "created" in a new data frame (contains everything except features)
data.df <- data.frame(data.train[! names(data.train) %in% c("features", "created")])
data.df <- cbind(data.df, dayOfMonth, dayOfWeek, timeListed)
nobs <- nrow(data.df) #49352 observations
###

#get zipcodes (longitude & latitude)
zip_codes_states <-
read.csv("/Users/TrevorWexner/Downloads/zip_codes_states.csv")
```

```
zipcode.data <- subset(zip_codes_states, state == "NY" & county %in% c("New
York", "Kings", "Queens"))
zipcode.data <- zipcode.data[, colnames(zipcode.data) %in% c("zip_code",
"latitude", "longitude")]

# round the directory to the same number of decimal places
zipcode.data$latitude <- round(zipcode.data$latitude, 3)
zipcode.data$longitude <- round(zipcode.data$longitude, 3)
zipcode.data <- as.data.frame(zipcode.data)

# assign each observed longitude and latitude coordinate to the correspoinding
zipcode
zipcode.observed <- as.data.frame(cbind(round(data.train$latitude, 3),
round(data.train$longitude, 3),
                                        rep(0, nobs)))
colnames(zipcode.observed) <- c("latitude", "longitude", "zipcode")

# for a given zipcode, set "observed.zipcode" = zipcode for all
for(i in 1:nrow(zipcode.data)) {
  lat.range <- c(zipcode.data$latitude[i] + .001, zipcode.data$latitude[i] - .001)
  lon.range <- c(zipcode.data$longitude[i] + .001, zipcode.data$longitude[i] - .001)
  indices <- which(((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
                   & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
  zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}

for(i in 1:nrow(zipcode.data)) {
  lat.range <- c(zipcode.data$latitude[i] + .003, zipcode.data$latitude[i] - .003)
  lon.range <- c(zipcode.data$longitude[i] + .001, zipcode.data$longitude[i] - .001)
  indices <- which((zipcode.observed$zipcode == 0)&
                    ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
                   & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
  zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}


for(i in 1:nrow(zipcode.data)) {
  lat.range <- c(zipcode.data$latitude[i] + .001, zipcode.data$latitude[i] - .001)
  lon.range <- c(zipcode.data$longitude[i] + .003, zipcode.data$longitude[i] - .003)
  indices <- which((zipcode.observed$zipcode == 0)&
                    ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
```

```
               & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
  zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}


for(i in 1:nrow(zipcode.data)) {
  lat.range <- c(zipcode.data$latitude[i] + .003, zipcode.data$latitude[i] - .003)
  lon.range <- c(zipcode.data$longitude[i] + .003, zipcode.data$longitude[i] - .003)
  indices <- which((zipcode.observed$zipcode == 0)&
                ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
               & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
  zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}


for(i in 1:nrow(zipcode.data)) {
  lat.range <- c(zipcode.data$latitude[i] + .005, zipcode.data$latitude[i] - .005)
  lon.range <- c(zipcode.data$longitude[i] + .003, zipcode.data$longitude[i] - .003)
  indices <- which((zipcode.observed$zipcode == 0)&
                ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
               & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
  zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}


for(i in 1:nrow(zipcode.data)) {
  lat.range <- c(zipcode.data$latitude[i] + .005, zipcode.data$latitude[i] - .005)
  lon.range <- c(zipcode.data$longitude[i] + .005, zipcode.data$longitude[i] - .005)
  indices <- which((zipcode.observed$zipcode == 0)&
                ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
               & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
  zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}


for(i in 1:nrow(zipcode.data)) {
  lat.range <- c(zipcode.data$latitude[i] + .05, zipcode.data$latitude[i] - .05)
  lon.range <- c(zipcode.data$longitude[i] + .005, zipcode.data$longitude[i] - .005)
  indices <- which((zipcode.observed$zipcode == 0)&
          ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
         & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
 zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}

# expand the longitudinal radius for those without a zipcode
for(i in 1:nrow(zipcode.data)) {
 lat.range <- c(zipcode.data$latitude[i] + .005, zipcode.data$latitude[i] - .005)
 lon.range <- c(zipcode.data$longitude[i] + .05, zipcode.data$longitude[i] - .05)
 indices <- which((zipcode.observed$zipcode == 0)&
          ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
         & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
 zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}

# expand the longitudinal radius for those without a zipcode
for(i in 1:nrow(zipcode.data)) {
 lat.range <- c(zipcode.data$latitude[i] + .05, zipcode.data$latitude[i] - .05)
 lon.range <- c(zipcode.data$longitude[i] + .05, zipcode.data$longitude[i] - .05)
 indices <- which((zipcode.observed$zipcode == 0)&
          ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
         & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
 zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}

for(i in 1:nrow(zipcode.data)) {
 lat.range <- c(zipcode.data$latitude[i] + .5, zipcode.data$latitude[i] - .5)
 lon.range <- c(zipcode.data$longitude[i] + .05, zipcode.data$longitude[i] - .05)
 indices <- which((zipcode.observed$zipcode == 0)&
          ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
         & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
 zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}

for(i in 1:nrow(zipcode.data)) {
 lat.range <- c(zipcode.data$latitude[i] + .05, zipcode.data$latitude[i] - .-5)
 lon.range <- c(zipcode.data$longitude[i] + .5, zipcode.data$longitude[i] - .5)
 indices <- which((zipcode.observed$zipcode == 0)&
            ((zipcode.observed$latitude <= lat.range[1]) &
(zipcode.observed$latitude >= lat.range[2]))
           & ((zipcode.observed$longitude <= lon.range[1]) &
(zipcode.observed$longitude >= lon.range[2])))
  zipcode.observed$zipcode[indices] <- zipcode.data$zip_code[i]
}
as.factor(zipcode.observed$zipcode)
View(zipcode.observed)
# missing classifications for 61 zipcodes. leave blank
###

# remove "street address" & display address
data.df <- data.df[! colnames(data.df) %in% c("street_address", "display_address",
"latitude", "longitude", "description")]
data.df <- cbind(data.df, zipcode.observed$zipcode)
colnames(data.df[ncol(data.df)]) <- "zipcode"

# Proximity to subway
subway_data <-
read.csv("/Users/TrevorWexner/Downloads/NYC_Transit_Subway_Entrance_And_
Exit_Data.csv")
subway_data <- subway_data[, c("Station.Latitude", "Station.Longitude")] #keep
only longitude and latitude
colnames(subway_data) <- c("latitude", "longitude")

# for each property, calculate the minimum euclidian distance to the nearest
subway station
distance <- function(x1, y1, x2, y2) {
  return(sqrt((x1 - x2)**2 + (y1 - y2)**2))
}

dist.to.subway <- rep(NA, nobs)
for(i in 1:nobs) {
  dist.to.subway[i] <- min(distance(x1 = data.train$latitude[i], y1 =
data.train$longitude[i],
                    x2 = subway_data$latitude, y2 = subway_data$longitude))
}

data.df <- cbind(data.df, dist.to.subway)
###

# Split the data up by number of bedroooms
split.by.bedrooms <- function(preprocessed.data) {
  zero_br <- subset(preprocessed.data, bedrooms == 0)
  one_br <- subset(preprocessed.data, bedrooms == 1)
  two_br <- subset(preprocessed.data, bedrooms == 2)
  three_br <- subset(preprocessed.data, bedrooms == 3)
  four_br <- subset(preprocessed.data, bedrooms == 4)
  five_br <- subset(preprocessed.data, bedrooms == 5)
  six_plus_br <- subset(preprocessed.data, bedrooms >= 6)
  return(list(zero_br, one_br, two_br, three_br, four_br, five_br, six_plus_br))
}
data.by.br <- split.by.bedrooms(data.df)

#Get testing and training indices
set.seed(10)
training.indices <- list(NULL)
for(i in 1:length(data.by.br)) {
  training.indices[[i]] <- sample(1:nrow(data.by.br[[i]]), .7*nrow(data.by.br[[i]]),
replace = F)
}
training.indices.vector <- unlist(training.indices)

# Features: Text Mining
features <- data.train$features
# Convert each feature entry to a single character, then unlist to a vector
for(i in 1:length(features)) {
  features[[i]] <- paste(features[[i]][1:length(features[[i]])], collapse = ", ")
}
features <- unlist(features)

# Input vector to make a corpus
mycorpus.features <- VCorpus( VectorSource(features))

# 2, Change to lower case
mycorpus.features <- tm_map(mycorpus.features, content_transformer(tolower))

# 3, Remove some non-content words
mycorpus.features<- tm_map(mycorpus.features, removeWords,
stopwords("english"))

# 4, Remove punctuations
mycorpus.features <- tm_map(mycorpus.features, removePunctuation)

# 5, Ready to get word frequency matrix
dtm1 <- DocumentTermMatrix(mycorpus.features)
#str(dtm1) #1104 possible features

# Cut the bag to only include the words appearing at least 1% of the time
threshold <- .01*length(mycorpus.features)   # 1% of the total documents
words.01 <- findFreqTerms(dtm1, lowfreq=threshold)  # words appearing at least
among 1% of the documents
```

```
length(words.01)

dtm.01<- DocumentTermMatrix(mycorpus.features, control=list(dictionary =
words.01))
dtm.01 <- as.matrix(dtm.01)
dtm.01.train <- dtm.01[training.indices.vector,]

# NUMERICAL BACKWARDS ELIMINATION (USING OLS) for FEATURES
# keep only those words who have a significant influence on interest
# treat interest as a numerical variable in this case, and run a linear regression
numerical.interest <- as.character(data.df$interest_level[training.indices.vector])
numerical.interest[which(numerical.interest == "low")] <- "0"
numerical.interest[which(numerical.interest == "medium")] <- "1"
numerical.interest[which(numerical.interest == "high")] <- "2"
numerical.interest <- as.numeric(numerical.interest)

interest.matrix <- cbind(as.data.frame((as.matrix(dtm.01.train))),
numerical.interest)
colnames(interest.matrix)[ncol(interest.matrix)] <- "interest"

# backwards selection function
backwards.select.lm <- function(y, data) {
 p <- ncol(data) - 1 # start with all variables
 data.new <- data
 fit.temp <- lm(as.formula(paste(y, "~.", sep = "")) , data = data.new)
 print(dim(coef(summary(fit.temp))))
 largestp <- max(coef(summary(fit.temp))[2:nrow(coef(summary(fit.temp))), 4]) #
largest p-values of all the predictors

 while(largestp > .05) {
   p <- p - 1
   # get var with largest p value & remove from model
   var.remove <- rownames(subset(coef(summary(fit.temp)),
coef(summary(fit.temp))[,4] == largestp))
   data.new <- data.new[,!(names(data.new) == var.remove)]
   fit.temp <- lm(as.formula(paste(y, "~.", sep = "")), data = data.new)
   largestp <- max(coef(summary(fit.temp))[2:nrow(coef(summary(fit.temp))), 4]) #
largest p-values of all the predictors
 }
 return(fit.temp)
}


final.vars <- backwards.select.lm(y = "interest", data = interest.matrix)
final.vars <- names(final.vars$coefficients)[2:length(final.vars$coefficients)]
```

```
# Keep only those words who are significant at the .05 level
dtm.keep <- as.matrix(dtm.01)[,which(colnames(as.matrix(dtm.01)) %in%
final.vars)]
dtm.keep <- as.data.frame(dtm.keep)

# Append to data.df
data.df <- cbind(data.df, dtm.keep)
data.df <- data.df[! colnames(data.df) %in% c("building_id", "manager_id")] #
remove building id & manager id

# TEXT MINING THE DESCRIPTIONS
descriptions <- data.train$description

# Input vector to make a corpus
mycorpus.descriptions.train <- VCorpus( VectorSource(descriptions))

# 2, Change to lower case
mycorpus.descriptions.train <- tm_map(mycorpus.descriptions.train,
content_transformer(tolower))

# 3, Remove some non-content words
mycorpus.descriptions.train<- tm_map(mycorpus.descriptions.train, removeWords,
stopwords("english"))

# 4, Remove punctuations
mycorpus.descriptions.train <- tm_map(mycorpus.descriptions.train,
removePunctuation)

# 6, Stem words
mycorpus.descriptions.train <- tm_map(mycorpus.descriptions.train,
stemDocument, lazy = TRUE)

# 7, Ready to get word frequency matrix
dtm1 <- DocumentTermMatrix(mycorpus.descriptions.train)

# Cut the bag to only include the words appearing at least 1% of the time
threshold <- .01*length(mycorpus.descriptions.train)   # 1% of the total documents
words.01 <- findFreqTerms(dtm1, lowfreq=threshold)  # words appearing at least
among 1% of the documents
length(words.01)
words.01

dtm1.01<- DocumentTermMatrix(mycorpus.descriptions.train,
control=list(dictionary = words.01))
dim(as.matrix(dtm1.01)) #716 words
dtm1.01 <- as.matrix(dtm1.01)[,!colnames(dtm1.01) == "interest"]
```

```
dtm1.01.train <- dtm1.01[training.indices.vector,]

# keep only those words who have a significant influence on interest
# treat interest as a numerical variable in this case, and run a linear
regression/LASSO
# first do as a linear regression, then do as a multinomial and compare results
interest.matrix <- cbind(as.data.frame((as.matrix(dtm1.01.train))),
numerical.interest)
colnames(interest.matrix)[ncol(interest.matrix)] <- "interest"

# Do LASSO against a numerical scale of interest to cut variables. Choose lamda1se
X <- model.matrix(~.+0, interest.matrix)
Y <- X[, "interest"]
X <- X[, - ncol(X)]
fit.descriptions.cv <- cv.glmnet(X, Y, alpha=1, nfolds=10)
plot(fit.descriptions.cv)
#save(fit.descriptions.cv, file = "descriptionsCV_regression.Rdata")

#comparing lambdamin & lambda1se
coef.1se <- coef(fit.descriptions.cv, s="lambda.1se")  # 51 terms
coef.1se <- coef.1se[which(coef.1se !=0),]
fit.descriptions.cv$lambda.1se #lambda1se error of .00398, or .199% of max (2)
lasso.vars <- rownames(as.matrix(coef.1se)) # 397 total variables left

# run backwards selection on remaining variables
interest.matrix <- interest.matrix[,colnames(interest.matrix) %in% c(lasso.vars,
"interest")]
final.vars <- lasso.vars
#final.vars <- backwards.select(y = "interest", data = interest.matrix)
#final.vars <- names(final.vars$coefficients)[2:length(final.vars$coefficients)]

# keep subset of freq matrix with these terms
dtm.keep <- as.matrix(dtm1.01)[,which(colnames(as.matrix(dtm1.01)) %in%
final.vars)]
dtm.keep <- as.data.frame(dtm.keep)

# Attatch to original df
data.df <- cbind(data.df, dtm.keep) #Except for images, data frame ready for
predictive modeling
# make listing_id the rowname and remove as a variable
listing_id <- data.df$listing_id
rownames(data.df) <- listing_id
data.df <- data.df[! colnames(data.df) == "listing_id"]
write.csv(data.df, file = "cleaned_testing&training_data.csv")
```

```
### TEMPORARY AND REMOVE LATER.. KEEP ONLY COLUMNS PRESENT IN
TESTING DATA
#data.df <- data.df[,-missing.columns[3:length(missing.columns)]]

# loading data
# data.df <- as.data.frame(full_testing_training_data)
# listing_id <- data.df[,1]
# rownames(data.df) <- listing_id
# data.df <- data.df[,-1]

### NOW we have:
# Split into testing and training sets & divide by number of bedrooms
temp.split <- split.by.bedrooms(data.df)
training.set <- data.df[training.indices.vector,]
testing.set <- data.df[- training.indices.vector,]
train <- list(NULL)
test <- list(NULL)
for(i in 1:length(temp.split)) {
  train[[i]] <- subset(temp.split[[i]], rownames(temp.split[[i]]) %in%
rownames(training.set))
  test[[i]] <- subset(temp.split[[i]], rownames(temp.split[[i]]) %in%
rownames(testing.set))
}

# remove "bedrooms" as a variable from testing and training sets
# also change the name of the zipcode varaible to zipcode
for(i in 1:length(test)){
  train[[i]] <- train[[i]][, !colnames(train[[i]]) == 'bedrooms']
  test[[i]] <- test[[i]][, !colnames(test[[i]]) == 'bedrooms']
  colnames(train[[i]])[7] <- "zipcode"
  colnames(test[[i]])[7] <- "zipcode"
}

# there are no instances of "high" interest in 6+ bedroom propertees, so exclude if
from levels
train[[7]]$interest_level <- as.factor(as.character(train[[7]]$interest_level))
test[[7]]$interest_level <- as.factor(as.character(test[[7]]$interest_level))

# for completeness of the analysis, add one instance of "high" to [[6]], and "medium"
to [[7]]
# cant do analysis on 5 bedrooms, since there is no "high" interest in our training set
train[[7]] <- rbind(train[[7]], test[[7]][27,]) # a random instance of medium
train[[6]] <- rbind(train[[6]], test[[6]][51,]) # a random instance of high
test[[7]] <- test[[7]][-27,] # remove from testing data
test[[6]] <- test[[6]][-51,] # remove from testing data
```

```
# 460 explanatory variables included

# PART 1: BUILD A RANDOM FOREST CLASSIFIER FOR EACH BEDROOM DATASET
start.time <- Sys.time()
rf.fits <- list(NULL)

# fitting studio appartments. first tune on mtry, then # of trees
# mtry defaults to 21, so try up to 25
rf.mce <- vector()
for (i in 1:25) {
  rf.fit <- randomForest(interest_level~., data=train[[1]], mtry=i, ntree=100)
  rf.mce[i] <- rf.fit$err.rate[rf.fit$ntree, 1]    # we only need to pull out the last error
}
plot(rf.mce, main = "Tuning mtry:Testing Error", xlab = "mtry")
## We see from the plot that tuning mtry to be 21 yields the minimum MCE
(roungly .285)

# now, tuning for ntree
rf.fits[[1]] <- randomForest(interest_level~., data = train[[1]], mtry = 21, ntree =
150)
plot(rf.fits[[1]], main = "Tuning Number of Trees")
# we see that after about 100 trees, performance plateaus

# now fit all models
start.time <- Sys.time()
for(i in 1:length(train)) {
  rf.fits[[i]] <- randomForest(interest_level~., data = train[[i]], mtry = 21, ntree =
100)
  print(i)
  print(Sys.time() - start.time)
}
#rf.fits[[7]] <- randomForest(interest_level~., data = train[[7]], mtry = 21, ntree =
100)
#rf.fits[[6]] <- randomForest(interest_level~., data = train[[6]], mtry = 21, ntree =
100)
#plot(rf.fits[[1]]); plot(rf.fits[[2]]); plot(rf.fits[[3]]); plot(rf.fits[[4]])

# See how we do with prediction (and probabilities)
rf.pred <- list(NULL)
rf.pred.label<- list(NULL)
for(i in 1:length(train)) {
  rf.pred[[i]] <- predict(rf.fits[[i]],test[[i]],type="prob")
  rf.pred.label[[i]] <- predict(rf.fits[[i]],test[[i]],type="class")
}

# confusion matrices and error rates
```

```
cm.rf <- list(NULL)
mce.rf <- list(NULL)
for (i in 1:length(train)) {
  print(i)
  cm.rf[[i]] <- table(rf.pred.label[[i]], test[[i]]$interest_level)
  mce.rf[[i]] <- (mean(rf.pred.label[[i]] != test[[i]]$interest_level))
}
mce.rf <- unlist(mce.rf)
print(mce.rf)
#0.30399761 0.24984147 0.31789971 0.32698790 0.37435520 0.01212121
0.07142857

# compute overall misclassification rate for random forest
# overall MCE out of sample of 2.2% (1072 misclassifications out of 49352)
tot.mce.1 <- sum(mce.rf*unlist(lapply(train, nrow)))/sum(unlist(lapply(train,
nrow))) #mce= .296

#compute multiclass log loss
mlogloss.rf <- list(NULL)
for(i in 1:length(test)) {
  mlogloss.rf[[i]] <- mlogLoss(test[[i]]$interest_level, rf.pred[[i]])
}
mlogloss.rf <- unlist(mlogloss.rf) # vector
tot.logloss.1 <- sum(mlogloss.rf*unlist(lapply(train,
nrow)))/sum(unlist(lapply(train, nrow)))
# total log loss of .7046

# PART 2: Trying a random forest against numerical predictiors

# make interest numericac and put in a new matrix
numerical.interest <- as.character(data.df$interest_level)
numerical.interest[which(numerical.interest == "low")] <- "0"
numerical.interest[which(numerical.interest == "medium")] <- "1"
numerical.interest[which(numerical.interest == "high")] <- "2"
numerical.interest <- as.numeric(numerical.interest)
numerical.df <- cbind(data.df[,!colnames(data.df) == "interest_level"],
numerical.interest)
colnames(numerical.df)[ncol(numerical.df)] <- "interest"
colnames(numerical.df)[7] <- "zipcode"

# seperate numerical interest matrix into testing and training data
temp.split <- split.by.bedrooms(numerical.df)
train.numeric <- list(NULL)
test.numeric <- list(NULL)
for(i in 1:length(temp.split)) {
  train.numeric[[i]] <- subset(temp.split[[i]], rownames(temp.split[[i]]) %in%
rownames(training.set))
  test.numeric[[i]] <- subset(temp.split[[i]], rownames(temp.split[[i]]) %in%
rownames(testing.set))
}

## make sure there is a level of everything in both training sets
train.numeric[[7]] <- rbind(train.numeric[[7]], test.numeric[[7]][27,]) # a random
instance of medium
train.numeric[[6]] <- rbind(train.numeric[[6]], test.numeric[[6]][51,]) # a random
instance of high
test.numeric[[7]] <- test.numeric[[7]][-27,] # remove from testing data
test.numeric[[6]] <- test.numeric[[6]][-51,] # remove from testing data


# numerical random forest (assuming same performance as before for tuning, for
simplicity)
rf.fits.numeric <- list(NULL)
start.time <- Sys.time()
for(i in 1:length(train)) {
  rf.fits.numeric[[i]] <- randomForest(interest~., data = train.numeric[[i]], mtry = 21,
ntree = 100)
  print(i)
  print(Sys.time() - start.time)
}

rf.pred.numeric<- list(NULL)
for(i in 1:length(train)) {
  rf.pred.numeric[[i]] <-
as.factor(round(predict(rf.fits.numeric[[i]],test.numeric[[i]],type="response")))
}

# confusion matrices and error rates
cm.rf2 <- list(NULL)
mce.rf2 <- list(NULL)
for (i in 1:length(train)) {
  print(i)
  cm.rf2[[i]] <- table(rf.pred.numeric[[i]], test.numeric[[i]]$interest)
  mce.rf2[[i]] <- (mean(rf.pred.numeric[[i]] != test.numeric[[i]]$interest))
}
mce.rf2 <- unlist(mce.rf2)
print(mce.rf2)
tot.mce.2 <- sum(mce.rf2*unlist(lapply(train, nrow)))/sum(unlist(lapply(train,
nrow)))
print(tot.mce.2)
# actually better predictions
```

```
#[1] 0.32085322 0.25156264 0.33110038 0.36347412 0.39130435 0.01212121
0.07142857
#0.3090083 MCE, about the same

# in order to get log loss, we need to compute probabilities
# take probability as normalized distance to 0, 1 and 2
rf2.probs <- list(NULL)
for(i in 1:length(train)) {
  rf2.distances <- cbind(abs(predict(rf.fits.numeric[[i]],
test.numeric[[i]],type="response") -2),
                  abs(predict(rf.fits.numeric[[i]], test.numeric[[i]],type="response") -0),
                  abs(predict(rf.fits.numeric[[i]], test.numeric[[i]],type="response") -1))

  tot.distance <- (rf2.distances[,1] + rf2.distances[,2] + rf2.distances[,3])

  rf2.probs[[i]] <- (tot.distance - rf2.distances)/(2*tot.distance)
}

# get log loss
mlogloss.rf2 <- list(NULL)
for(i in 1:length(test)) {
  mlogloss.rf2[[i]] <- mlogLoss(test[[i]]$interest_level, rf2.probs[[i]]) # log loss of
.736
}
mlogloss.rf2 <- unlist(mlogloss.rf2) # vector
tot.logloss.2 <- sum(mlogloss.rf2*unlist(lapply(train,
nrow)))/sum(unlist(lapply(train, nrow)))
# log loss of .209, way better --> I don't know how accurate this is


# PART 2: LASSO TO CUT VARIABLES - given the fact that these are ordinal data, use
linear model
start.time <- Sys.time()
cv.lasso <- list(NULL)
for(i in 1:length(train)) {
  x.temp <- model.matrix(interest~.,
                  train.numeric[[i]][,-which(colnames(train.numeric[[i]]) ==
"bedrooms")])[, -1]
  y.temp <- train.numeric[[i]]$interest
  cv.lasso[[i]] <- cv.glmnet(y=y.temp, x=x.temp, alpha = 1, nfolds = 10)
  print(i)
}
print(Sys.time() - start.time)

plot(cv.lasso[[1]]) #.32 error. lambda1se
plot(cv.lasso[[2]]) #.24 error. lambda1se
plot(cv.lasso[[3]]) #.35 error
plot(cv.lasso[[4]]) #.3 error (mse)
plot(cv.lasso[[5]]) # .25 error
plot(cv.lasso[[6]]) #.05 error
plot(cv.lasso[[7]]) #.05 error

# still masssive models (300 vars). Maybe we can dwindle down further using lasso
against all
y.temp <- numerical.df$interest[training.indices.vector]
x.temp <- model.matrix(interest~., numerical.df)[training.indices.vector,-1]
overall.lasso.cv <- cv.glmnet(y = y.temp, x = x.temp)
plot(overall.lasso.cv)
#still almost 383 variables included

## Take overall LASSO coeffs
coef.1se <- coef(overall.lasso.cv, s="lambda.1se")
coef.1se <- coef.1se[which(coef.1se !=0),]
lasso.vars <- rownames(as.matrix(coef.1se))[-1]
# 378 vars included


# use only these varibles for a given bedroom setup
for(i in 1:length(train)) {
  train[[i]] <- train[[i]][,colnames(train[[i]]) %in% c(lasso.vars, "interest_level")]
  test[[i]] <- test[[i]][,colnames(test[[i]]) %in% c(lasso.vars, "interest_level")]
  train.numeric[[i]] <- train.numeric[[i]][,colnames(train.numeric[[i]]) %in%
c(lasso.vars, "interest")]
  test.numeric[[i]] <- test.numeric[[i]][,colnames(test.numeric[[i]]) %in%
c(lasso.vars, "interest")]
}

# attempting backwards selection. gives linear model
backwards.selection.models <- list(NULL)
backwards.selection.vars <- list(NULL)
for(i in 1:length(train)) {
  backwards.selection.models[[i]] <- backwards.select.lm(y = "interest", data =
train.numeric[[i]])
  backwards.selection.vars[[i]] <- names(coef(backwards.selection.models[[i]]))
}
print(unlist(lapply(backwards.selection.vars, length)))
# [1] 65 68 93 77 76 71 vars per model

# no model avaiable for 6+ bedroooms, so use variables from 5 br
backwards.selection.models[[7]] <- lm(interest~., data =
train.numeric[[7]][,colnames(train.numeric[[7]]) %in%
```

```
                                        c(backwards.selection.vars[[6]],
"interest")]])

backwards.selection.vars[[7]] <- backwards.selection.vars[[6]]

## See linear model predictions
lm.predictions <- list(NULL)
for(i in 1:length(train)) {
  lm.predictions[[i]] <-
as.factor(round(predict(backwards.selection.models[[i]],test.numeric[[i]],type="res
ponse")))
}

# confusion matrices and error rates
cm.lm <- list(NULL)
mce.lm <- list(NULL)
for (i in 1:length(train)) {
  print(i)
  cm.lm[[i]] <- table(lm.predictions[[i]], test.numeric[[i]]$interest)
  mce.lm[[i]] <- (mean(lm.predictions[[i]] != test.numeric[[i]]$interest))
}
mce.lm <- unlist(mce.lm)
print(mce.lm)
tot.mce.lm <- sum(mce.lm*unlist(lapply(train, nrow)))/sum(unlist(lapply(train,
nrow)))
print(tot.mce.lm)
# [1] 0.36843675 0.28743546 0.38498670 0.41701368 0.48489315 0.05454545
0.07142857 MCE for each model
# 0.3573412 total MCE, somehow not terrible


################# STOP HERE BEFORE LOGISTIC REG ############
# Part 3: Multiclass Logistic Regression
glm.fits <- list(NULL)
for(i in 1:length(train)) {
  glm.fits[[i]] <- multinom(interest_level~., data=train[[i]][,colnames(train[[i]])
%in%
                                  c(backwards.selection.vars[[i]], "interest_level")])
}

glm.pred <- list(NULL)
glm.pred.label <- list(NULL)
for(i in 1:length(train)) {
  glm.pred[[i]] <- predict(glm.fits[[i]], test[[i]][,colnames(test[[i]]) %in%
                             c(backwards.selection.vars[[i]], "interest_level")],
type="prob")
```

```
  glm.pred.label[[i]] <- predict(glm.fits[[i]], test[[i]][,colnames(test[[i]]) %in%
                                        c(backwards.selection.vars[[i]], "interest_level")],
type="class")
}


cm.glm <- list(NULL)
mce.glm <- list(NULL)
for (i in 1:length(train)) {
  cm.glm[[i]] <- table(glm.pred.label[[i]], test[[i]]$interest_level)
  mce.glm[[i]] <- (mean(glm.pred.label[[i]] != test[[i]]$interest_level))
}
mce.glm <- unlist(mce.glm)
print(mce.glm)
# [1] 0.31279833 0.25409910 0.34489213 0.36981955 0.43699337 0.03636364
0.07142857

tot.mce.3 <- sum(mce.glm*unlist(lapply(train, nrow)))/sum(unlist(lapply(train,
nrow)))
# tot mce of .315. Not great

mlogloss.glm <- list(NULL)
for(i in 1:length(test)) {
  mlogloss.glm[[i]] <- mlogLoss(test[[i]]$interest_level, glm.pred[[i]]) # log loss of
.736
}
mlogloss.rf <- unlist(mlogloss.glm) # vector
tot.logloss.3 <- sum(mlogloss.rf*unlist(lapply(train,
nrow)))/sum(unlist(lapply(train, nrow)))

##### NOW, REDUCE DIMENSIONALITY OF TRAINING AND TESTING DATA WITH
PCA #########
# PCA on features
pc.features <- prcomp(training.set[, 10:(ncol(training.set)-408)], scale. = TRUE)
pc.features.imp <- t((summary(pc.features))$importance)   # this is a matrix
pc.features.imp <- as.data.frame(pc.features.imp)
names(pc.features.imp) <- c("Sdev", "PVE", "CPVE")
plot(pc.features.imp$PVE, xlim=c(1, 100))
plot(pc.features.imp$CPVE, main="Scree plot of CPVE")
abline(h = .8, col = "blue")

# with the first 15 features components, we get 95% of variance expliained
# still, not using this

# PCA on descriptions
```

```
pc.descriptions <- prcomp(training.set[, (ncol(training.set)-408):ncol(training.set)],
scale. = TRUE)
pc.descriptions.imp <- t((summary(pc.descriptions))$importance)   # this is a
matrix
pc.descriptions.imp <- as.data.frame(pc.descriptions.imp)
names(pc.descriptions.imp) <- c("Sdev", "PVE", "CPVE")
attach(pc.descriptions.imp)
plot(PVE, xlim=c(1, 100))
plot(CPVE, main="Scree plot of CPVE")
abline(h = .8, col = "blue")

#### ORDINAL LOGISTIC REGRESSION ON REDUCED DATASET
clm.fits <- list(NULL)
for(i in 1:length(train.numeric)) {
  train.numeric[[i]]$interest <- as.factor(train.numeric[[i]]$interest)
  test.numeric[[i]]$interest <- as.factor(test.numeric[[i]]$interest)
  clm.fits[[i]] <- clm(interest~., data =
train.numeric[[i]][,colnames(train.numeric[[i]]) %in%
                                            c(backwards.selection.vars[[i]], "interest")])
}

clm.pred <- list(NULL)
clm.pred.label <- list(NULL)
for(i in 1:length(train)) {
  clm.pred[[i]] <- predict(clm.fits[[i]], test.numeric[[i]][,colnames(test.numeric[[i]])
%in%
                                           c(backwards.selection.vars[[i]], "interest_level")],
type="prob")
  clm.pred.label[[i]] <- predict(clm.fits[[i]],
test.numeric[[i]][,colnames(test.numeric[[i]]) %in%
                                               c(backwards.selection.vars[[i]],
"interest_level")], type="class")
}


cm.clm <- list(NULL)
mce.clm <- list(NULL)
for (i in 1:length(train)) {
  cm.clm[[i]] <- table(unlist(clm.pred.label[[i]]), test.numeric[[i]]$interest)
  mce.clm[[i]] <- (mean(unlist(clm.pred.label[[i]]) != test.numeric[[i]]$interest))
}

mce.clm <- unlist(mce.clm)
print(mce.clm)
# [1] 0.30996420 0.25437087 0.34144419 0.36406901 0.43773029 0.05454545
0.07142857
```

```
tot.mce.4 <- sum(mce.clm*unlist(lapply(train, nrow)))/sum(unlist(lapply(train,
nrow)))
# tot.mce = .313, not very good

mlogloss.clm <- list(NULL)
for(i in 1:length(test)) {
  mlogloss.clm[[i]] <- mlogLoss(test[[i]]$interest_level, clm.pred[[i]]) # log loss of
.736
}
mlogloss.clm <- unlist(mlogloss.clm) # vector
tot.logloss.4 <- sum(mlogloss.clm*unlist(lapply(train,
nrow)))/sum(unlist(lapply(train, nrow)))


# for fast back testing, save:
data.df.subsets <- data.df[, colnames(data.df) %in% c(backwards.selection.vars,
"interest_level")]
write.csv(data.df, "data_df.csv") # data.df
save(rf.fits.numeric, file = "numeric_rf.Rdata") # rf numeric fits
save(rf.fits, file = "rf.Rdata")# rf fits
save(glm.fits, file = "glm.Rdata")# glm.fits

# remove stuff
# rm(dtm.01); rm(dtm.01.train); rm(dtm.keep); rm(dtm1); rm(dtm1.01);
rm(dtm1.01.train);
# rm(interest.matrix); rm(numerical.df); rm(X); rm(testing.set); rm(training.set);
# rm(cm.rf); rm(coef.1se); rm(data.by.br); rm(data.train); rm(dates);
rm(dayOfMonth); rm(dayOfWeek)
# rm(descriptions); rm(features); rm(fit.descriptions.cv);
rm(mycorpus.descriptions.train); rm(mycorpus.features)
#


# OTHER: Tables for the report
model.names <- c("RF Classification", "RF Regression", "Multinomial Logit", "Oridnal
Logit")
#MCE table - overall (overall.mces.csv)
overall.mces <- data.frame(unlist(lapply(c(tot.mce.1, tot.mce.2, tot.mce.3, tot.mce.4),
round, 3)))
rownames(overall.mces) <- model.names
# MCE table- by nbedrooms
mces <- cbind(round(mce.rf,3), round(mce.rf2,3), round(mce.glm,3),
round(mce.clm,3))
rownames(mces) <- c("Studio", "1br", "2br", "3br","4br", "5br", "6+br")
colnames(mces) <- model.names
```

```
write.csv(mces, "mces.csv")
# logloss table
overall.loglosses <- rbind(c(round(tot.logloss.1,3), round(tot.logloss.2,3),
round(tot.logloss.3,3)))
colnames(overall.loglosses) <- model.names[-4]

# view the distribution of interest levels across apartment sizes
interest.by.size <- matrix(nrow = 7, ncol = 3)
for(i in 1:6){
  interest.by.size[i,] <- round(as.vector(table(test[[i]]$interest_level))/
                              sum(as.vector(table(test[[i]]$interest_level))), 3)
}
low <- interest.by.size[,2]
medium <- interest.by.size[,3]
high <- interest.by.size[,1]
interest.by.size <- cbind(low, medium, high)
rownames(interest.by.size) <- rownames(mces)
write.csv(interest.by.size, "interest_by_size.csv")

#final model confusion matrices
for(i in 1:7) {
  filename <- paste("cm", i, sep = "")
  write.csv(cm.rf[[i]], file = paste(filename, ".csv", sep = ""))
}
```