Predicting How Much Interest a Rental Listing on RentHop Will Receive

Andrew Ju

3/9/2017

Introduction

An apartment rental website like RentHop has tens of thousands of listings with millions of views - how accurately can one predict how much interest a particular listing will receive? Using data provided by RentHop and Two Sigma in a competition hosted by Kaggle (https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries), this exercise outlines an approach using multinomial logistic regression and gradient boosting in R to predict how much interest a rental listing generates.

RentHop provides us data on 14 different features of a rental listing, in addition to whether the listing received low, medium, or high interest. We're provided a training data set of 49,352 observations where we know how much interest a listing receives, and a test data set of 74,659 observations where we don't know the interest level.

Setup

We begin by first importing the data from a JSON file to a data frame we can work with. The package "jsonlite" takes care of the difficulty of parsing a JSON formatted file, but we also use the package "purrr" to help deal with the fact that some features have more than one variable for each listing (i.e. there is only one price but many photos and features for each listing).

```
vars_train <- setdiff(names(train), c("photos", "features"))
train <- map_at(train, vars_train, unlist)
train <- tibble::as_tibble(train)

vars_test <- setdiff(names(test), c("photos", "features"))
test <- map_at(test, vars_test, unlist)
test <- tibble::as_tibble(test)
```

And though it's a rich set of data, we also remove the photos feature, as scraping and analyzing images is outside the scope of this project. We also remove the street address and display address variables, as we'll be relying on the more precise and less error-prone latitude and longitude data provided. Some other initial clean-ups include setting the variable we're predicting, interest level, as a factor, and setting the created time-stamp variable to the standard POSIXlt format.

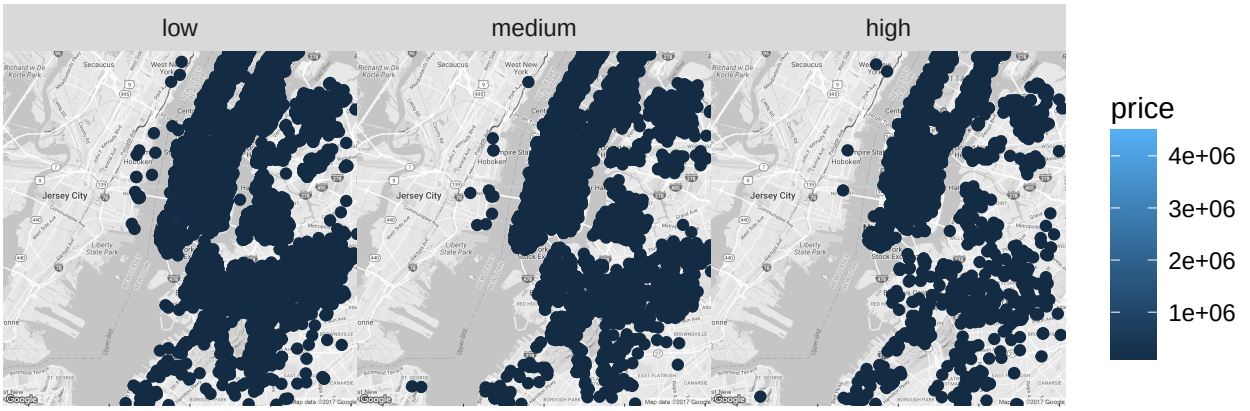
```
train_w <- subset(train, select = -c(photos, features, display_address,
    street_address))
train_w$interest_level <- as.factor(train_w$interest_level)
train_w$interest_level <- factor(train_w$interest_level, levels = c("low",
    "medium", "high"))
train_w$created <- ymd_hms(train$created)

test_w <- subset(test, select = -c(photos, features, display_address,
    street_address))
test_w$created <- ymd_hms(test$created)
test_w$interest_level <- factor(NA)
```

Geospatial Exploration of Data

The first part of the data we'll explore and make use of is the geolocational data. Using the "ggmaps" package, we plot the listings in the training dataset and find that the listings center aroud NYC and Newark. Beyond that, though, it's hard to discern any useful signal out of the map because of the sheer number of observations we're examining in such a concentrated area.

```
nyc_map <- qmap("new york city", zoom = 12, color = "bw")

nyc_map + geom_point(aes(x = longitude, y = latitude, color = price),
    data = train_w) + facet_wrap(~interest_level)
```

```
## Warning: Removed 5867 rows containing missing values (geom_point).
```



Instead of looking at each observation's exact location, it might be more useful to just use a listing's neighborhood as a factor variable in our models. This'll retain the listing's geolocational features but generalizes enough to give us something we can work with.

Although the "ggmaps" package has a very useful reverse geocoding function that can return a wealth of information about a particular location, including neighborhood, it relies on Google Maps API which limits queries to 2500 requests a day. Given that we have a total of about 124,000 listings, we won't be able to rely on this method, convenient though it is. Instead, we find a workaround by manually finding a list of neighborhoods in the New York Metropolitan Area and find the coordinates for the center of each of those neighborhoods. From there, we apply a K-Nearest Neighbor model for each listing's lat/lon data against each of the different neighborhood's central coordinates to find which neighborhood it's in.

```
# ggmaps revgeocode method - does not scale
test1 <- train_w[1, ]
test1 <- cbind(test1$longitude, test1$latitude)
revgeocode(test1, output = "more")$neighborhood
```

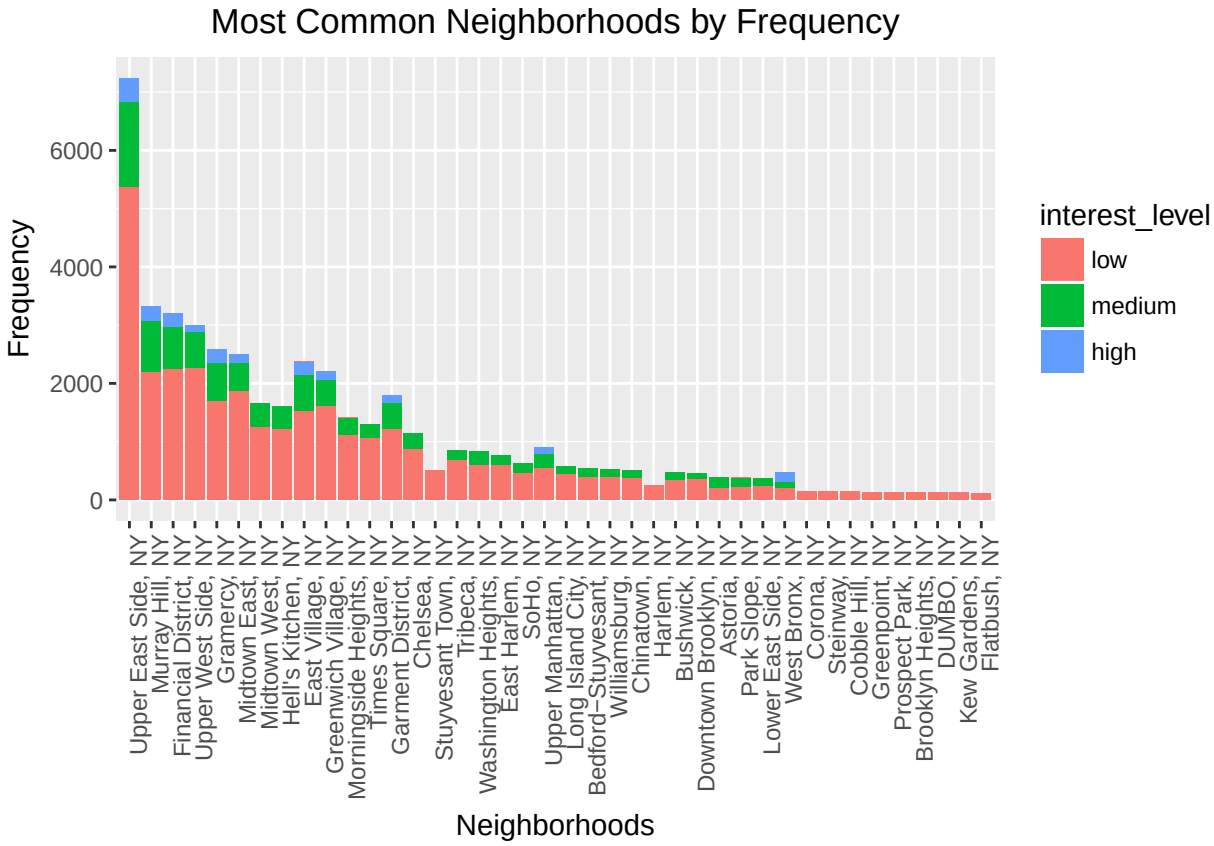
```
## [1] Williamsburg
## Levels: Williamsburg
```

```
m_neighborhoods <- c("Chelsea", "Washington Heights", "Harlem",
    "East Harlem", "Upper West Side", "Upper East Side", "Midtown West",
    "Midtown East", "Greenwich Village", "Lower East Side", "Murray Hill",
    "Stuyvesant Town", "Upper Manhattan", "Hell's Kitchen", "East Village",
    "SoHo", "Financial District", "Gramercy", "Garment District",
    "Morningside Heights", "Tribeca", "Chinatown", "Times Square")

b_neighborhoods <- c("Bay Ridge", "Sunset Park", "Bensonhurst",
    "Sheepshead Bay", "Borough Park", "Midwood", "Flatbush",
```

```
    "East Flatbush", "Park Slope", "East New York", "Bedford-Stuyvesant",
    "Williamsburg", "Greenpoint", "Red Hook", "Downtown Brooklyn",
    "DUMBO", "Brownsville", "Prospect Park", "Fort Hamilton",
    "Cypress Hills", "Bushwick", "Canarsie", "Brooklyn Heights",
    "Cobble Hill")

q_neighborhoods <- c("Astoria", "Long Island City", "Steinway",
    "Ridgewood", "Woodside", "Elmhurst", "Jackson Heights", "Corona",
    "Murray Hill", "Flushing", "Kew Gardens", "Fresh Meadows",
    "Jamaica", "Bayside", "Whitestone")

s_neighborhoods <- c("West New Brighton", "Mariners Harbor")


bx_neighborhoods <- c("West Bronx", "Yankee Stadium")

nj_neighborhoods <- c("Newark")

getCoords <- function(neighborhoods) {
    num_n <- length(neighborhoods)
    if (neighborhoods[1] == "Newark") {
        neighborhoods <- paste0(neighborhoods, ", NJ")
    } else {
        neighborhoods <- paste0(neighborhoods, ", NY")
    }

    lat <- rep(0, num_n)
    lon <- rep(0, num_n)

    for (i in 1:num_n) {
        n <- neighborhoods[i]
        reply <- suppressMessages(geocode(n))
        lat[i] <- reply$lat
        lon[i] <- reply$lon
    }

    return(data.frame(n = neighborhoods, lat = lat, lon = lon))
}

# getCoords(nj_neighborhoods)

neighb_df <- do.call("rbind", list(getCoords(m_neighborhoods),
    getCoords(b_neighborhoods), getCoords(bx_neighborhoods),
    getCoords(q_neighborhoods), getCoords(s_neighborhoods), getCoords(nj_neighborhoods)))

neighborhoods_train <- knn(neighb_df[, c("lat", "lon")], train_w[,
    c("latitude", "longitude")], neighb_df$n, k = 1)
train_w$neighborhood <- neighborhoods_train

neighborhoods_test <- knn(neighb_df[, c("lat", "lon")], test_w[,
    c("latitude", "longitude")], neighb_df$n, k = 1)
test_w$neighborhood <- neighborhoods_test
#
```

As one might suspect, we can see that the overwhelming majority of the listings are in NY rather than NJ and that there are far more listings with low and medium interest than with high interest.

```
train_neighb_count <- count(train_w, neighborhood, interest_level)

ggplot(subset(train_neighb_count, n > 100), aes(x = reorder(neighborhood,
    -n), y = n, fill = interest_level, order = interest_level)) +
    geom_bar(stat = "identity") + geom_col(position = position_stack(reverse = TRUE)) +
    ggtitle("Most Common Neighborhoods by Frequency") + xlab("Neighborhoods") +
    ylab("Frequency") + theme(axis.text.x = element_text(angle = 90,
    hjust = 1), plot.title = element_text(hjust = 0.5))
```



Data Exploration

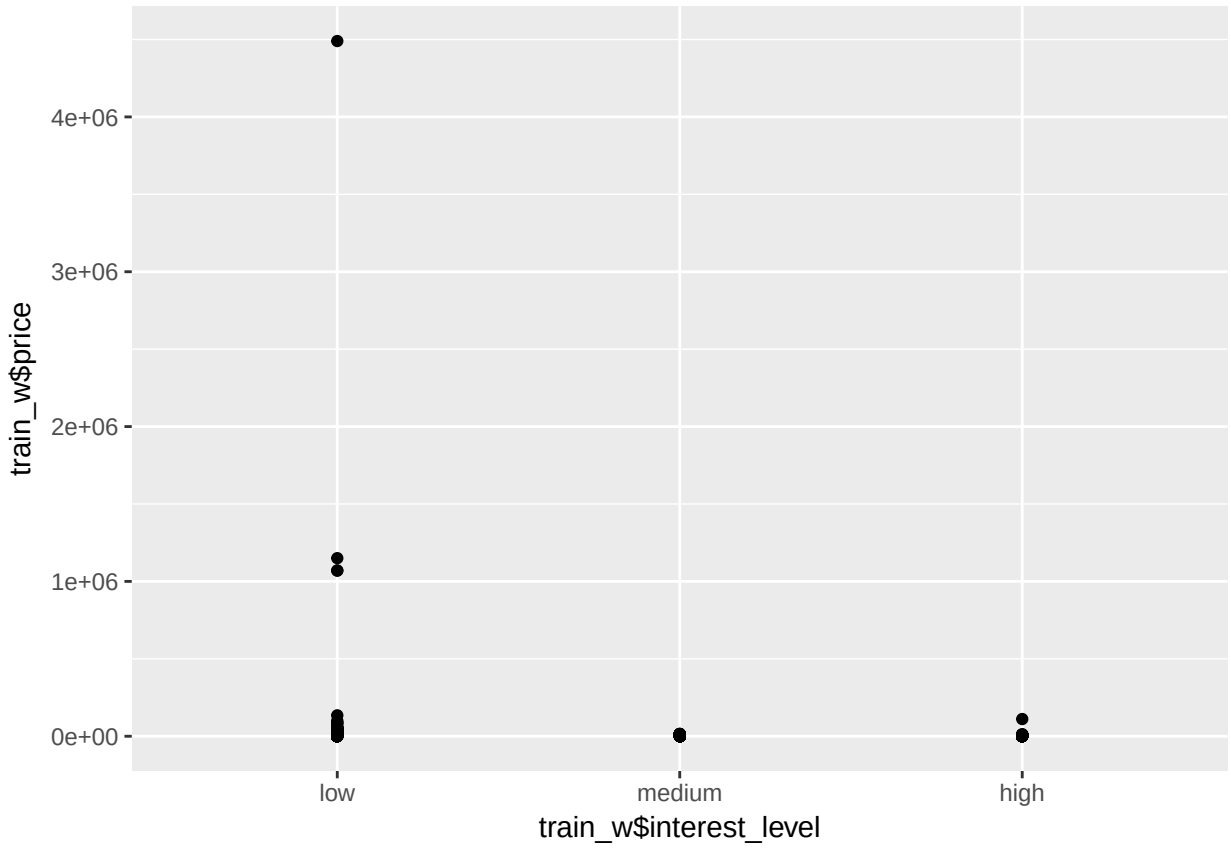
We start by exploring the data and finding a few worthwhile insights:

Though it is a complete dataset with no NAs, making our modeling process much cleaner, it's quite heavily skewed. Looking at the feature we'll be predicting, there are far more observations of listings receiving low and medium interest than high interest. We can also see that higher priced listings tend to receive low interest.

```
lapply(train_w, function(x) sum(is.na(x)))
lapply(test_w, function(x) sum(is.na(x)))
```

```
qplot(train_w, y = train_w$price, x = train_w$interest_level)
```

```
## Warning: Ignoring unknown parameters: NA
```




```
# train_w[-which(train_w$price>1000000), ] %>% nrow
# price_plot <- train_w[-which(train_w$price>1000000), ] #
# %>% nrow qplot(price_plot, y = price_plot$price, x =
# price_plot$interest_level)
```

We can also see if there's any pattern underlying the interest levels a listing receives based on when the listing is posted. Ignoring a few aberrations, interest seems fairly steady across time, so a time series analysis may not be very useful.

```
time_train <- train_w %>% group_by(created) %>% count(interest_level)
ggplot(time_train, aes(as.Date(created), fill = interest_level)) +
    geom_bar(aes(weight = n), position = "fill", size = 0.75) +
    ylab("Proportion of Listings") + ggtitle("Time Series of Interest Level") +
    theme(plot.title = element_text(hjust = 0.5))
```

And though we're not using the photos and features data extensively, we can at least count how many photos and features a listing provides. Perhaps just the amount of information and photos provided is correlated with how much interest a listing receives. Looking just briefly at the managers_id and buildings_id features, we also see that only about 7% and 15% of the IDs are unique, telling us that it's quite possible that listings associated with particular managers and buildings receive more interest than others.

```
train_w$photos_num <- lengths(train$photos)
train_w$features_num <- lengths(train$features)

test_w$photos_num <- lengths(test$photos)
test_w$features_num <- lengths(test$features)

length(unique(train_w$building_id))/length(train_w$building_id)
```

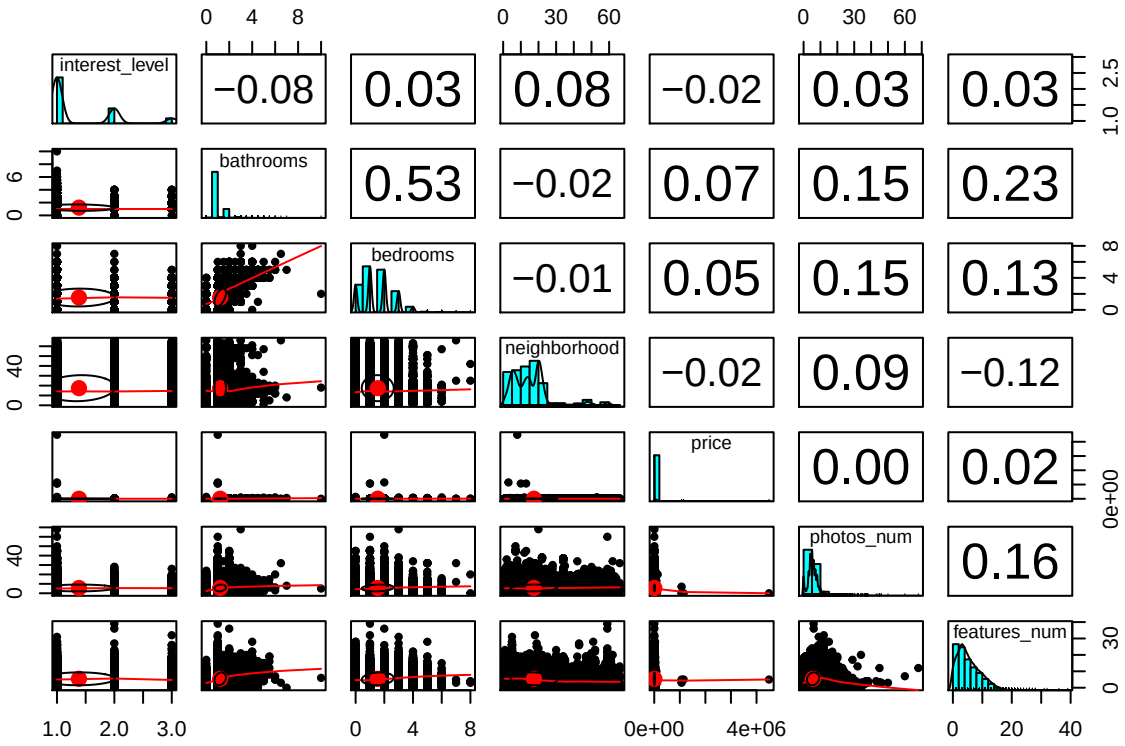
```
## [1] 0.1536918
```

```
length(unique(train_w$manager_id))/length(train_w$manager_id)
```

```
## [1] 0.07053412
```

To cap off the initial data exploration, we examine the correlation the different features have between each other to check for multicollinearity and to see which particular variables might be especially correlated with interest levels. None of the variables we've examined so far seem obviously correlated with interest level, but we can trust our modeling process later to make use of this information.

```
pairs.panels(subset(train_w, select = c(interest_level, bathrooms,
    bedrooms, neighborhood, price, photos_num, features_num)))
```



Basic Sentiment Analysis of Description

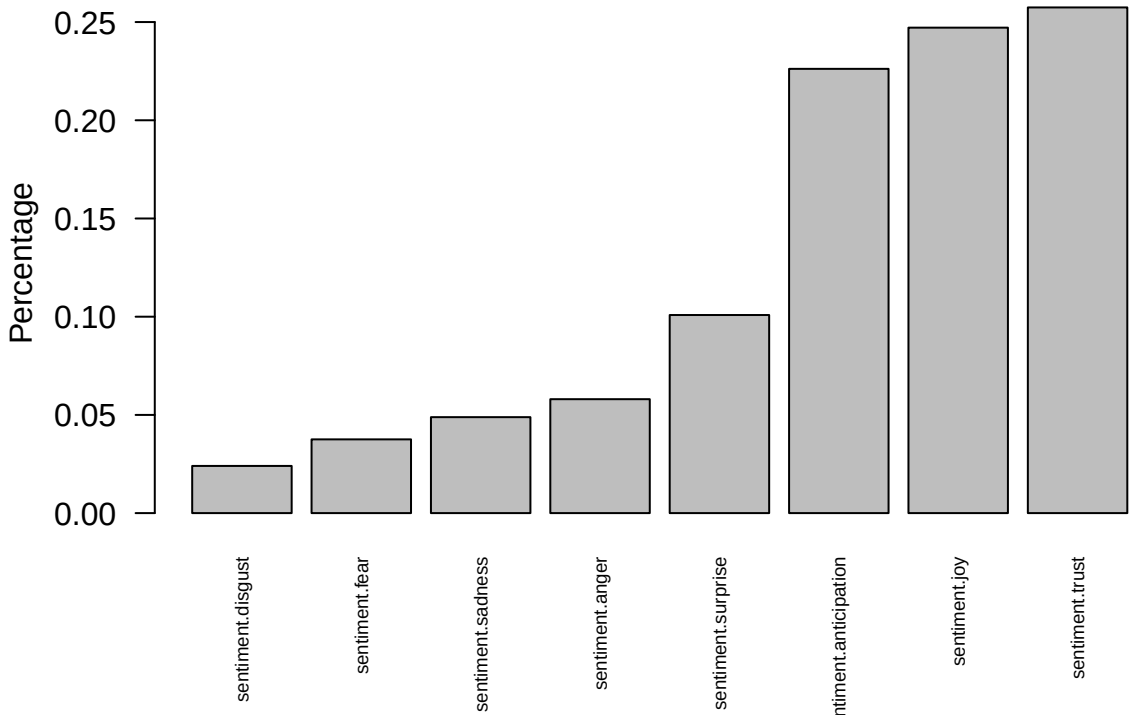
We can also try to extract some value out of the written descriptions a listing provides. Though potentially a rich set of data, it's also tricky to work with as it features natural language. A typical example is as follows:

```
train_w$description[1]
```

```
## [1] "Spacious 1 Bedroom 1 Bathroom in Williamsburg!Apartment Features:- Renovated Eat in Kitchen Wit
```

We can utilize the "syuzhet" package, which provides us a sentiment analysis of each description and returns a score along eight emotional values: anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. We can use these sentiment scores in our models and hopefully improve our predictions. As one might expect, the most popular sentiments are trust, joy, and anticipation.

```
sentiment_train <- get_nrc_sentiment(train_w$description)
colnames(sentiment_train) <- paste0("sentiment.", colnames(sentiment_train))
train_w <- cbind(train_w, sentiment_train)

sentiment_test <- get_nrc_sentiment(test_w$description)
colnames(sentiment_test) <- paste0("sentiment.", colnames(sentiment_test))
test_w <- cbind(test_w, sentiment_test)


# all((train_w[, colnames(sentiment_train)] ==
# sentiment_train) == TRUE)

barplot(sort(colSums(prop.table(sentiment_train[, 1:8]))), horiz = FALSE,
    cex.names = 0.6, las = 2, main = "Sentiment in Descriptions of Listings",
    ylab = "Percentage")
```




```
# barplot( sort(colSums(prop.table(sentiment_test[, 1:8]))),
# horiz = FALSE, cex.names = 0.6, las = 2, main = 'Sentiment
# in Descriptions of Listings', ylab='Percentage' )
```

Multinomial Logistic Regression Model

Having explored and extracted some features out of our data, we can try fitting a model to predict interest level. We start with a basic multinomial logistic regression model, which predicts categorical variables and is an extension of a binary logistic regression. Using our model's predictions on the test dataset, we find that our model has a multiclass loss of .693.

```
logm1 <- multinom(data = train_w, formula = interest_level ~
    bathrooms + bedrooms + created + price + neighborhood + photos_num +
        features_num + sentiment.anger + sentiment.anticipation +
        sentiment.disgust + sentiment.fear + sentiment.joy +
        sentiment.sadness + sentiment.surprise + sentiment.trust +
        sentiment.negative + sentiment.positive)
```

```
## # weights:  249 (164 variable)
## initial  value 54218.713670
## iter  10 value 43809.660693
## iter  20 value 39962.709750
## iter  30 value 37408.352784
## iter  40 value 36494.444605
## iter  50 value 35229.005324
## iter  60 value 34570.153421
## iter  70 value 34291.566584
## iter  80 value 34140.180395
```

```
## iter  90 value 34039.959826
## iter 100 value 34013.857041
## final  value 34013.857041
## stopped after 100 iterations
```

```
fitted.results <- predict(logm1, newdata = test_w, type = "prob")
test_w$logm1pred <- fitted.results
logm1_subm <- cbind(test_w$listing_id, test_w$logm1pred)
colnames(logm1_subm) <- c("listing_id", "low", "medium", "high")

# write.csv(logm1_subm, file = 'logm1_subm.csv', row.names =
# FALSE) first attempt w/ no features gives us multiclass
# loss of .734 w/ new features multiclass loss of .69309
```

XGBoost Model - WORK IN PROGRESS

We can also try to improve predictive performance with a more powerful model. Let's try on Extreme Gradient Boosting run by xgmboost, a package in R that implements a gradient boosting decision tree algorithm very quickly and efficiently. This is really a combination of two techniques, boosting and gradient descent. Boosting is an ensemble technique creates new models sequentially to improve the errors made by previous models; gradient boosting uses the gradient descent algorithm to minimize errors and guide which models are built sequentially.

```
str(train_w)
train_xgb <- subset(train_w, select = -c(description))
test_xgb <- subset(test_w, select = -c(description))


train_xgb$neighborhood <- as.integer(factor(train_xgb$neighborhood))
train_xgb$building_id <- as.integer(factor(train_xgb$building_id))
train_xgb$manager_id <- as.integer(factor(train_xgb$manager_id))
train_xgb$listing_id <- as.integer(factor(train_xgb$listing_id))

train_xgb$month <- month(train_xgb$created)
train_xgb$day <- day(train_xgb$created)
train_xgb$hour <- hour(train_xgb$created)
train_xgb$created <- NULL


test_xgb$neighborhood <- as.integer(factor(test_xgb$neighborhood))
test_xgb$building_id <- as.integer(factor(test_xgb$building_id))
test_xgb$manager_id <- as.integer(factor(test_xgb$manager_id))
test_xgb$listing_id <- as.integer(factor(test_xgb$listing_id))

test_xgb$month <- month(test_xgb$created)
test_xgb$day <- day(test_xgb$created)
test_xgb$hour <- hour(test_xgb$created)
test_xgb$created <- NULL


train_xgb$photos_num <- log(train_xgb$photos_num + 1)
train_xgb$features_num <- log(train_xgb$features_num + 1)
train_xgb$price <- log(train_xgb$price + 1)
```

```
test_xgb$photos_num <- log(test_xgb$photos_num + 1)
test_xgb$features_num <- log(test_xgb$features_num + 1)
test_xgb$price <- log(test_xgb$price + 1)


train_xgb$interest_level <- as.integer(factor(train_xgb$interest_level))
intrst_lvl <- train_xgb$interest_level
intrst_lvl <- intrst_lvl - 1
train_xgb$interest_level <- NULL
test_xgb$interest_level <- NULL


seed = 1985
set.seed(seed)
xgb_parameters = list(colsample_bytree = 0.7, subsample = 0.7,
    eta = 0.1, objective = "multi:softprob", max_depth = 4, min_child_weight = 1,
    eval_metric = "mlogloss", num_class = 3, seed = seed)

dtest <- xgb.DMatrix(data.matrix(test_xgb))

kfolds <- 10
folds <- createFolds(intrst_lvl, k = kfolds, list = TRUE, returnTrain = FALSE)
fold <- as.numeric(unlist(folds[1]))

x_train <- train_xgb[-fold, ]  #Train set
x_val <- train_xgb[fold, ]  #Out of fold validation set

y_train <- intrst_lvl[-fold]
y_val <- intrst_lvl[fold]

dtrain = xgb.DMatrix(as.matrix(x_train), label = y_train)
dval = xgb.DMatrix(as.matrix(x_val), label = y_val)

gbdt = xgb.train(params = xgb_parameters, data = dtrain, nrounds = 475,
    watchlist = list(train = dtrain, val = dval), print_every_n = 25,
    early_stopping_rounds = 50)

allpredictions = (as.data.frame(matrix(predict(gbdt, dtest),
    nrow = dim(test_xgb), byrow = TRUE)))
allpredictions %>% str

allpredictions = cbind(allpredictions, test$listing_id)
names(allpredictions) <- c("low", "medium", "high", "listing_id")
allpredictions = allpredictions[, c(4, 1, 2, 3)]
write.csv(allpredictions, paste0(Sys.Date(), "-BaseModel-20Fold-Seed",
    seed, ".csv"), row.names = FALSE)

# b/c of log transforms maybe ... or the feature engineering
# ...
```