THE STARBUCKS DATA HUNT

PREDICTING STARBUCKS YELP SCORES TO FIND THE ISSUES THAT MATTER

Christopher Jose

Springboard Data Science Intensive
Capstone Project Report
January 2017

A Lengthy Introduction: Why I care about Starbucks

Starbucks has two membership levels that offer rewards: green level and gold level. I consider myself as extra-gold platinum plus premier level. The level doesn't exist, but in my opinion, it's needed. A level that only the most loyal of coffee-holics can ever hope to reach. Reaching gold level from green is like learning to do a three-point turn. What about the customers that can parallel park blind in a blizzard? That's me. There's others too. We're out there. On my days off, I often spend more hours at my local Starbucks than the actual baristas. I occasionally ask them if they've begun renting out cots. It'd save me a 15-minute ride home. I've become quite acquainted with Starbucks over the past several years. I've logged hundreds of hours at different locations. I've spent hundreds (perhaps thousands) of dollars on coffee and parking meters. I've gone out drinking with baristas, Snapchat friended them, accepted illegal freebies on an almost daily basis… They ask when I'll start working there. I'm like a permanent fixture hanging on the wall.

I've seen things happen at Starbucks. Fights, breakups, baristas from hell, business deals devolving into awkward silences, a job interview where the current employer showed up and said hello (much to the chagrin of the interviewee), homeless people overdosing in the bathrooms. A regular named Debby would tell me everyday that I reminded her of her father, and that he had nice hair and "was a handsome man". I could eventually finish the sentence for her, and I did, since she said the same words to me everyday. Debby talked to herself loudly, and sometimes this talking was racist, but at least she gave friendly hellos to everyone. I haven't seen Debby lately.

You see, it's not all sunshine and rainbows in Starbucks. The bathrooms can be disgusting and will run out of hand soap. I joke with the baristas that the next guy who comes out of there will undoubtedly have hands that you won't want to shake. Homeless people come in, talk to themselves loudly, and carry with them an intense wall of stench that remains even after they've departed. For the longest time, I had wondered where the stench had been originating from. Other homeless people stand nearby the front door and smoke. I am sometimes infiltrated with a big whiff of narcotics on breezier days whenever the front door opens.

Yet, not all is bad with Starbucks. It seems like a focal point for communities—a hub of community activity. A place where people get together and talk, do business, relax, where students do their homework, where first dates go horribly awkward…It's a melting pot of experiences and a place where people from different walks of life come and co-exist. Given how much of my life I spend at Starbucks, I am interested in investigating and improving the brand and customer experience.

Identifying Issues that Customers care most about

Given my own plethora of experiences at Starbucks, I have thoughts on how the customer experience can be improved. I am curious as to whether other customers agree with me. To maintain its status as a leading coffee provider and hub of community activity, it seems obvious that Starbucks should continually monitor and consider the collective feedback of its customers.

My hunch is that store cleanliness, homeless people, barista friendliness, order accuracy, consistency of drink quality, and running out of inventory are among the biggest issues for customers and are therefore things that can be improved a lot at certain locations. I often sit at the bar, the area where baristas make drinks, and I've observed that customers frequently complain about the wrong drink having been made, that the drink tasted weird, or that it was missing an ingredient. I've also observed that trash cans are not frequently stocked enough and overfill to the point of trash falling onto the ground. Another common occurrence is that the barista making drinks at the bar will inform the customer that the store is out of an ingredient (after the order had already been paid for).

Coincidentally, as I was writing the last sentence in my local Starbucks, a barista informed a guest that they ran out of strawberry puree (after the drink had been purchased), which goes into Frappuccinos. I let the barista know how perfectly what just happened fits in with the contents of my report. These issues are incredibly commonplace and happen throughout each and every day, as evidence by the above incident.

Yelp

A place where Starbucks can monitor customer feedback is Yelp. Yelp is a website that lets customers give public feedback to businesses. This feedback is in the form of written reviews and star scores ranging from 1-5, where 5 indicates a perfect score and high level of regard for the business, and 1 indicates the worst score and a low level of regard for the business. Each business for a particular location (if it's a chain store) is given an overall star score, which is an aggregation of individual star scores Yelp deems as legitimate.

By monitoring Yelp, Starbucks can improve its brand by focusing on improving factors that Yelp users care most about concerning Starbucks. This means determining factors that most influence Starbucks Yelp scores. These factors would best be determined through the development of models. The models would predict Starbucks star scores using the considered factors, and then factors that contribute the most towards predicting star scores within each model would be considered as more important to customers.

What the raw data looks like

The Yelp data is available at https://www.yelp.com/dataset_challenge, and consists of multiple files. I use the yelp_academic_dataset_business.json and yelp_academic_dataset_review.json files. I construct one table from each file, the *business* table and *reviews* table, respectively. The *business* table has a record for each business, and the *reviews* table has a record for each individual review of each business. Fields that I make use of in the business.json file include: *business_id* which uniquely identifies each store, store name, several location fields, the overall yelp star rating assigned to the store, and number of reviews. Fields that I make use of in the review.json file include: *business_id*, a field containing individual review text content, the date of the review, and the number of stars that the user assigned to their respective review. Thus, this file contains multiple records for each store, where each record is a review and the associated number of stars that the user gave that store location.

Files that I do not use include: a file containing information about each user, a file containing the total number of times yelp users indicated that they checked into the store during each hour of each day of the week, and a file containing photos.

Data Dictionaries of *business* and *reviews* tables

Business table

	Field Name	Type	Description	Example
0	business_id	object	encrypted business id	-8x21h1bkRD9uwEw1ec1xw
1	full_address	object	localized address	8150 S Maryland Pkwy\nSte D-1\nSoutheast\nLas ...
2	open	bool	True / False (True if store still operates)	True
3	categories	object	localized category names	[Food, Coffee & Tea]
4	city	object	city	Las Vegas
5	review_count	int64	review count	47
6	name	object	business name	Starbucks
7	neighborhoods	object	hood names	[Southeast]
8	longitude	float64	longitude	-115.135398
9	state	object	state	NV
10	stars	float64	star rating, rounded to half-stars	3.5
11	latitude	float64	latitude	36.041659
12	type	object	business	business

Reviews table

	Field Name	Type	Description	Example
0	votes	object	useful: count, funny: count, cool: count	{u'funny': 0, u'useful': 0, u'cool': 0}
1	user_id	object	encrypted user id	7DeM4C66zVflYNrZn_Vv9g
2	text	object	review text	This is a tiny Starbucks and it locations like...
3	business_id	object	encrypted business id	3gmBc0qN_LtGbZAjTtHWZg
4	stars	int64	star rating	3
5	date	object	date, formatted like 2012-03-14	2009-06-10
6	type	object	review	review

Data Wrangling: Preparing the data for Analysis

In preparing the data, I read the json files as a list of strings and then convert them to pandas DataFrames. I only use rows in the *business* table with 'Starbucks' contained within the store name field. The *reviews* table does not have a store name field, so I only use rows whose business id corresponds to a business id with 'Starbucks' in the store name field of the *business* table. The *reviews* table, even after filtering for only Starbucks stores, is large and takes awhile to import, so I pickle it to allow a much faster import of the table. Henceforth, in referring to table names *business* and *reviews*, I am referring to the Starbucks filtered versions of these tables.

After performing EDA, as outlined in the next section, I settle on the following predictor variables (except for region), whose construction is as follows:

In constructing the *mean_review_year* variable, I make a *year* column in the *reviews* table based on the *date* column. Then, for each unique business id in *reviews*, I calculate the mean year. This is the mean review year.

In constructing the *unclean* variable, I assign a new column in the *business* table with value 1 if the business id has either the words 'dirty', 'filthy', or 'messy' in any of its text reviews in the *reviews* table.

In constructing the *homeless* variable, I assign a new column in the *business* table with value 1 if the business id has the word 'homeless' in any of its text reviews in the *reviews* table.

In constructing the *unfriendly* variable, I assign a new column in the *business* table with value 1 if the business id has either the words 'rude' or 'mean' in any of its text reviews in the *reviews* table.

In including *state* as a predictor, I make dummy variables for each state, except for the state with the largest number of stores in the data, which turns out to be AZ. A value of 0 for every state dummy then represents AZ. I do this to avoid the dummy variable trap.

In constructing the *region* variable to explore in EDA, for a given row in *business*, I assign a region value based on the *state* column's value.

Exploratory Data Analysis

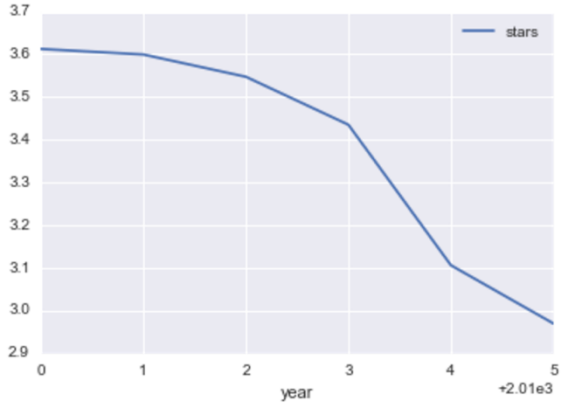
After preparing the data, I observe that there are 494 stores and typically 18 reviews per store. Most of the stores are located in Arizona and Nevada, which have 201 and 161 stores, respectively. Data is missing for many states, like for my home state Pennsylvania, so clearly, Yelp has not provided all of its data.

I decide to consider the following variables in predicting a store's star score: mean review year, region, state, clean vs not clean, friendly vs unfriendly baristas, homeless people problems vs no problems, and review count (number of reviews). I visualize the relationship between these variables and star score to see if I'd like to include them in the models.

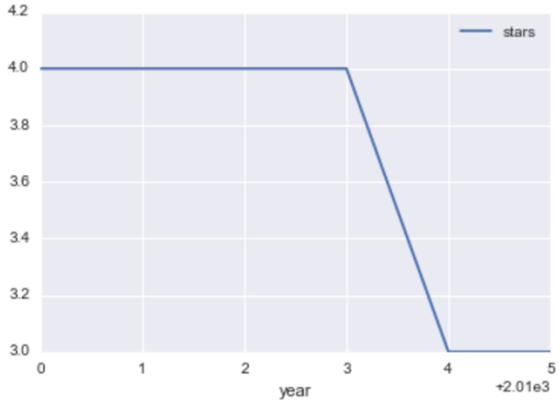
Investigating stars by review year

I investigate whether the average and median star score by year shows a clear pattern over time. Thus, for any given year in which users give Starbucks star scores, I look at the average and median star score for that year. The average and median star score decreases as review year increases (mean scores are graphed on the left, and median scores are graphed on the right). I conclude that review year deserves consideration as a predictor, however, because most stores have reviews that take place in different years, I use the *average review year* as a predictor.

Average star score by year

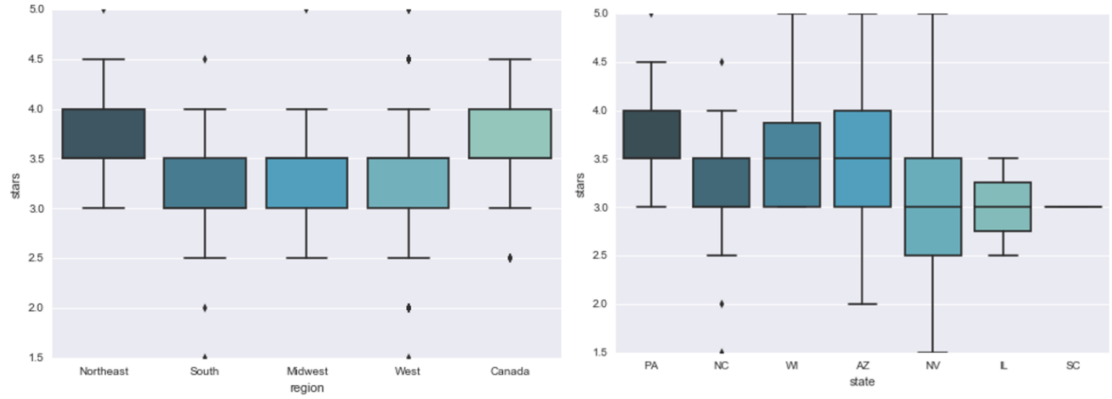


Median star score by year



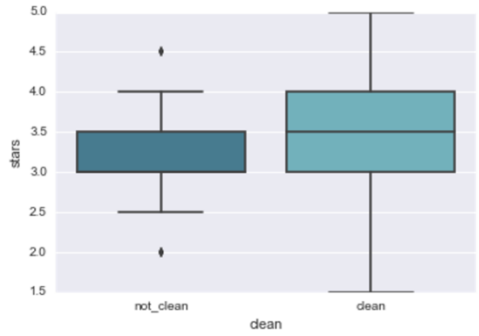
Investigating whether stars vary by region and state

I investigate whether either region (Northeast, West, Midwest, South, Canada) or state deserve consideration as predictor variables. Does the average star score differ across regions and states? Judging by the boxplots below, region does not appear to have differing average star scores, while state does appear to have differing average star scores. I will only consider state as a predictor.

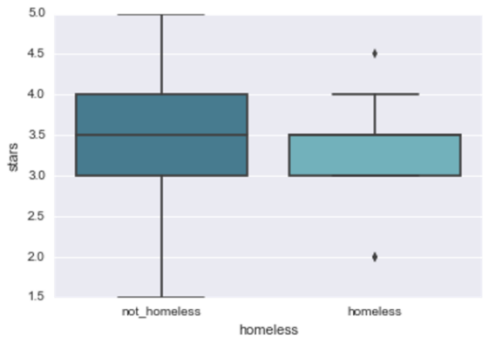


Investigating text review content

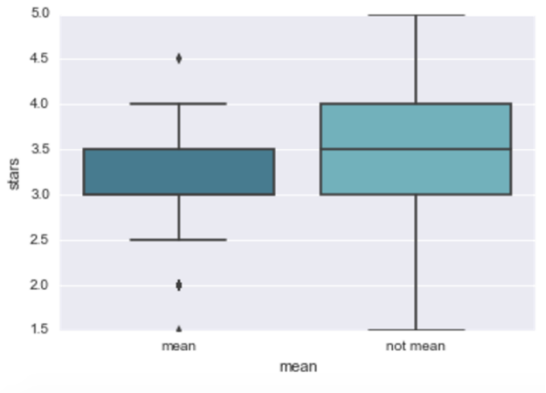
I investigate whether the presence of certain words in the text reviews changes the overall star score. I look into star score distributions for stores whose reviews contain the words "dirty", "filthy", and "messy" versus stores that do not have these words in their reviews. The presence of these words indicates that the store is not regarded as clean and so I am considering store cleanliness as a predictor variable. The below box plots indicate that store cleanliness should be considered as a predictor variable, since the distribution of star scores for perceived non-clean stores is lower than that of clean stores.



Similarly, I investigate whether the presence of the word "homeless" in text reviews affects the yelp score of a store. The below boxplot suggests that it does, so I will consider the presence of this word as a predictor variable:



I investigate whether the presence of the words “mean” or “rude” in text reviews affect the yelp score of a store, and they appear to, so I will consider the presence of these words as a predictor variable:

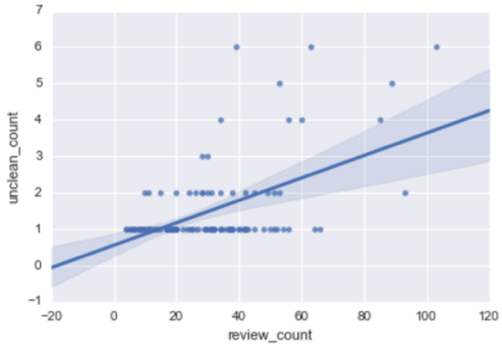


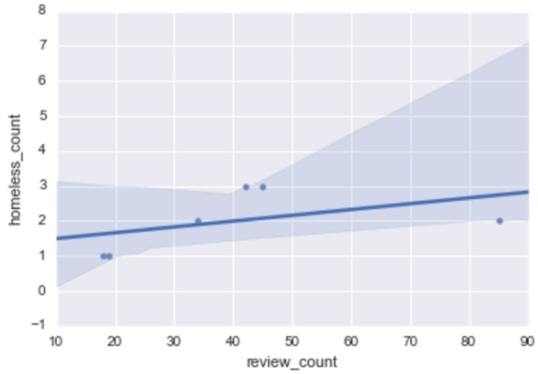
Investigating review count

Regarding the dummy variables, a potential issue is that the greater the number of reviews, the more opportunities for reviewers to use any of the words indicating homeless problems, uncleanliness, or unfriendliness. Thus, it seems plausible that a store with more reviews will also have a greater chance of having dummy variable values indicating the presence of the above problems. This would mean that the dummy variables are correlated with review count, which, if excluded from the model, would be a lurking variable that is contributing to the change in star scores associated with a change in the dummy variable values.

I find that review count is strongly positively correlated with the unfriendliness variable, mildly positively correlated with the uncleanliness variable, and that there are no conclusions to be drawn on the homeless variable due to too few stores labeled as having a homeless problem. The spearman correlation coefficients are .784, .535, and .46, respectively. The scatterplots between review count and a count of the number of reviews containing the words indicating problems is shown below. Least squares regression lines are shown as well, along with 95% confidence intervals of the predicted y values surrounding the line (the shaded area). I am going to included review count as a predictor variable due to its correlation with some of the dummy variables.


unfriendly_count
review_count


unclean_count
review_count


homeless_count
review_count

Predictors to use

mean_review_year
review_count
unclean
homeless
unfriendly
State dummy variables:
    BW, EDH, IL, NC, NV, ON, PA, QC, SC, WI
    where a value of 0 for every dummy represents AZ
    QC stands for Quebec, ON stands for Ontario, EDH stands for Edinburgh, BW=?

The Models

In predicting Starbucks Yelp scores, I generate and compare four models created using the following techniques: Linear Regression, Principal Component Regression, Random Forests, Gradient Boosted Regression Trees.

Models will be compared and ranked by their root mean square error (rmse), the typical amount by which a model's predictions deviate from the actual values. The Linear Regression and Principal Component Regression models will be built by splitting the data into train and test splits. 70% of the data will be randomly selected for training the models, while the remaining 30% will be used to evaluate the performance of the models.

The Random Forest and Gradient Boosting methods will be built using 5-fold cross validation and Grid search. Grid search evaluates different model parameters and determines the values that lead to optimal model performance.

Each model's rmse generated from testing the model on the test split data will be used to rank it against the other models.

Statistical graphs and outputs for the models are shown at the end of the report.

Linear Regression

I run a multiple linear regression model which generates the following output:

Predicted Overall Star Score for a given Starbucks store =
190.226 - .0927(mean_review_year) + .0027 (review_count) - .2268(unclean)
+ .2897 (homeless) - .2754 (unfriendly) - .9562 (BW) + .1339 (EDH) - .3327 (IL) - .2543 (NC)
- .2223 (NV) - .3317 (ON) + .2439 (PA) + .3968 (QC) - .1314 (SC) + .0944(WI)

There is a different model for each value of each dummy variable. If a store is located in NC (NC=1), and is regarded as being unclean, having a homeless problem, and unfriendly baristas, then unclean = homeless = unfriendly = 1, and the model changes as follows:

Predicted Stars =
(190.226-.2268+.2897-.2754-.2543) - .0927(mean_review_year) + .0027 (review_count)
=189.7592 - .0927(mean_review_year) + .0027 (review_count)

The coefficient of -.2268 for variable *unclean* is interpreted as a measure of the difference in the mean star score resulting from changing from value 0 (a clean store) to value 1 (an unclean store). This value is the change in the intercept between clean and unclean stores. The predicted star score for unclean stores is .227 lower than the predicted star score for clean stores.

The null hypothesis of the F test, that all regression coefficients are zero, is rejected at the .001 level with an F value of 4.632. This indicates that the variation in star scores explained by the model (the regression mean square) is 4.632 times the variation in star scores that is not explained by the model (the mean squared error). The intercept, mean_review_year, unclean, unfriendly, NC, NV, and QC variables are statistically significant at the .05 level, indicating that these predictors add a statistically significant amount of predictability to overall star score.

In inspecting the significant predictors, we expectedly observe that when customers regard baristas as unfriendly and the store as unclean, the predicted star score drops by .2754 and .2268, respectively (and holding all other variables constant). The predicted score also drops as the average year in which that store receives text reviews increases. Location also impacts star score. Apparently, people are happier with Starbucks in Quebec, which results in a .3968 increase in the predicted score.

The adjusted R-squared of .137 indicates that the model explains 13.7% of the variation in star score. The intercept of 190.226 does not have a practical interpretation because it would not make sense for the variables mean_review_year and review_count to have value zero.

In examining regression model assumptions, I inspect the residual plot and Q-Q plot (see the model output section at the end of this paper). In the residual plot, residuals do not seem to have a larger or smaller spread as the predicted star score increases, so I do not see evidence of heteroskedasticity. Residuals appear to be normally distributed in the Q-Q plot. In examining whether residuals are uncorrelated, I observe 8 diagonal bands of residuals within the residual plot, which is also the number of distinct star score values in the data. This concerns me, so I

further check for auto-correlation by calculating the Durbin Watson statistic, which equals 1.996. Values close to 2 indicate uncorrelated residuals, so it looks like the 8 bands of residuals are not indicating autocorrelation and regression assumptions appear to hold reasonably well.

I inspect variance inflation factors (VIFs) for multicollinearity problems, meaning whether the predictors are highly correlated. Values greater than either 4 or 5 may indicate a strong presence of multicollinearity. The VIFs for mean_review_year and review_count are the only VIFs greater than 4 (4.84 and 4.32, respectively). This indicates that the square of the standard errors for these estimated coefficients (aka the estimated variances of these estimated coefficients) are 4-5 times higher than they otherwise would have been because of correlations between these predictors and other predictors.

I test the model on the test split and calculate the rmse, which is .6544. This indicates that the typical error in this model's predictions on the test split versus the actual values is .6544.

Principal Component Regression

I run a principal component regression (PCR) to diminish the presence of multicollinearity and potentially improve the predictability of the model, as measured by the rmse. Principal components are derived using the correlation matrix. In choosing the number of principal components to retain in the model, I consider several criterion:

1. The minimum number of principal components to retain such that at least 80% of the variation of the predictor variables is retained is 11. This is observed by consulting the Variance Explained Plot.
2. The number of retained principal components after excluding those whose eigenvalues are less than the average eigenvalue is 9.
3. Using the elbow rule while examining the scree plot indicates at least 3-5 principal components are needed, though selecting only 5 principal components would only explain 45% of the variation in the data, which is not enough.
4. In excluding all principal components whose eigenvalues are close to zero, the last two principal components must be excluded or else multicollinearity won't be reduced, since these principal components will be too correlated with the intercept.

I will select 10 principal components, so that 79% of the variation in the data is explained and components with eigenvalues close to zero are excluded. We want to select as few principal components as possible to reduce the dimensionality of the data as much as possible, which will lead to a greater reduction in multicollinearity and model complexity. We also want to retain as much of the variation in the data as possible, so we can't select too few principal components.

Unlike in the linear regression model, the presence of multicollinearity is greatly diminished in PCR, due to having excluded principal components with eigenvalues close to zero. While all VIFs equal 1 because principal components are uncorrelated, if we had not excluded principal components with eigenvalues close to zero, then we would have merely masked the presence of multicollinearity.

The model is as follows, where Prin j is the principal component score for principal component j:
Predicted Overall Star Score =
3.3044 - .1194(Prin1) + .0493(Prin2) - .0180(Prin3) + .0885(Prin4) - .0201(Prin5) - .0340 (Prin6) + .0287 (Prin 7) + .0158(Prin 8) - .0555(Prin 9) - .0796(Prin 10)

Principal component scores can be interpreted as the values of the principal components. A given score for a principal component is calculated as a linear combination of the standardized values of a given observation. Thus, each principal component reflects information from multiple predictor variables. Any new data fed to this model should be standardized first, since the model was built from standardized predictor variables (principal components were derived from the correlation matrix, which is the same thing as building the model using the covariance matrix and standardized variables). Eigenvector values of each principal component (also called loadings) are applied to the standardized observations to calculate principal component scores for each component. The variance of each principal component is its eigenvalue.

Principal components that are statistically significant at the .05 level include: principal component 1, 4, 9, and 10.

Comparing Principal Component Regression (PCR) to Linear Regression (LR):
1. The adjusted R-squared for PCR is lower than that of LR (.108 vs .137).
2. The Akaike Information Criterion (AIC) is higher in PCR (649.1 vs 642.6).
3. The Bayesian Information Criterion (BIC) is lower in PCR (691.4 vs 704.1).

AIC is lower for models with a better goodness-of-fit and less complexity. We want to minimize AIC. BIC is similar to AIC, and we also want to minimize it, except it rewards less complexity even more (by increasing the penalty amount for the number of predictors). It makes sense that PCR has a lower BIC than LR's value, since while PCR has a worse fit as judged by its adjusted R-squared, it also has less complexity (less variables), which is more rewarded in the BIC value.

The presence of multicollinearity has been reduced, due to having excluded principal components with eigenvalues close to zero.

The rmse has decreased to .645 from .654, meaning this model has less error in predicting the test data.

Determining the most important variables in the PCR
We can attempt to interpret and assign labels to each principal component by examining their eigenvectors. This might allow us to get a sense of which variables are contributing the most to PCR. Components that explain more variation in the data (components with the largest eigenvalues, meaning those with the largest variance) will have more meaning. After examining the eigenvectors for the first several principal components however, I notice that there is a mixture of signs for different variables, and I find it difficult to form any meaningful labels. I won't try to make conclusions regarding variance importance for this model.

Random Forests

I implement a Random Forest regression model, where I tune the *max_features* parameter using 5-fold cross-validation and grid search. I first investigate the effect on the rmse of changing the *max_features* and *n_estimators* parameters.

n_estimators sets the number of regression trees to create in the random forest. A higher value improves model performance, but increases computation time and eventually offers diminishing returns.

max_features sets the size of the random subset of features generated at each split in every tree, so all trees and splits will have the same number of randomly selected features. This parameter affects the feature bagging randomization step in constructing trees. A higher value can lead to over-fitting and less generalizability of the model, though too low of a value can also negatively affect model performance.

A plot of rmse versus *n_estimators* for six different *max_features* values:

The rmse stay fairly level for a given *max_features* value after about 100 trees, so we won't see much (if any) of a performance gain by increasing the number of trees beyond this. I will use 300 trees to be certain that there is no loss in performance from using too small of an n_estimators value.

There is a different curve for each considered max_features value. Generating random subsets whose size is 50% of the total number of features results in the lowest rmse values. 30% is the next best max_features value. All other max_features values result in curves that have rmse values around .65 for tree values greater than 100. Clearly, *max_features* makes a difference, so I will tune this parameter using GridSearchCV.

The optimal *max_features* value, selected using 5-fold cross validation and grid search, is .1. The procedure calculates an average mean squared error (avg MSE) for each considered *max_features* value, and then selects the *max_features* value that generates the lowest avg MSE.

The procedure works as follows: the training data is split into 5 folds. Each fold is treated as a validation fold from which to evaluate a model that is trained on the other 4 folds. Thus, five models are created for each *max_features* value (one model for each fold), and 5 MSEs' are averaged from having tested each model on its respective validation fold.

The algorithm I use, *GridSearchCV*, actually calculates the negative of the mean squared error (neg MSE) for each considered *max_features* value, so it selects the *max_features* value that maximizes the neg MSE.

I then fit the random forest with the tuned *max_features* value of .1 on the test split, which results in an rmse of .6495.

I assess variables that contribute the most to the model by determining *feature importance* (or *variable importance*) values for each variable. These values are normalized to sum to 1. Interestingly, *mean_review_year* and *review_count* contribute the most, with values above .3. All other variables have values below .1. The code does not calculate feature importance as the gini importance, which is often done for classification trees. Documentation is unclear, but it appears that a variable's feature importance is based on its contribution to reducing the error rate within each tree that it is used.

Gradient Boosted Trees

I implement a gradient boosted regression tree, where I tune the *max_depth*, *learning_rate, subsample*, and *max_features* parameters using 5-fold cross-validation and grid search.

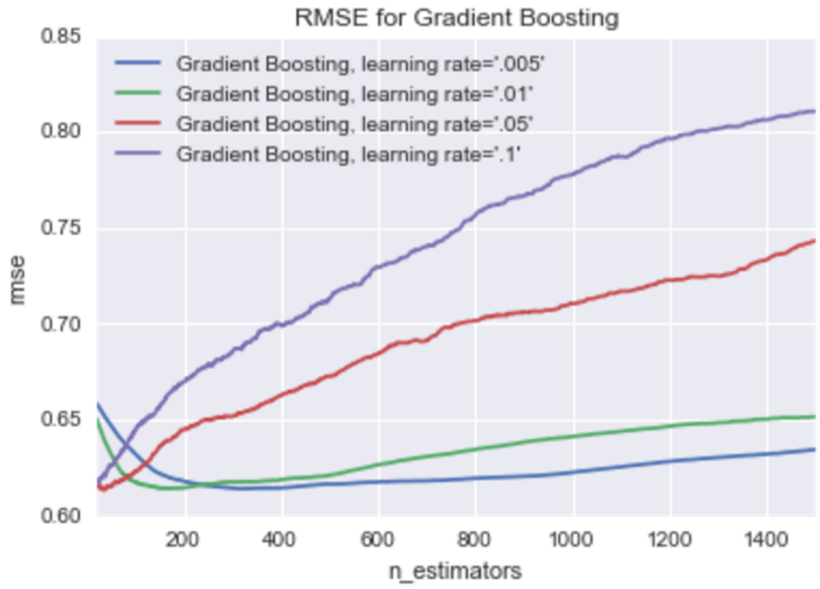
max depth controls the size (number of layers) of each tree where you can think of layer j as being composed of nodes or decision values arrived at from having gone through j decisions or splits. Less layers can mean a worse prediction, though too many layers can lead to over-fitting and less generalizability.

learning rate controls the impact of each subsequent tree on the current model. Using a lower learning rate prevents over-fitting, but the code takes longer.

subsample sets the percentage of observations (rows) randomly sampled in fitting the model. We use a value lower than 1 to prevent over-fitting.

max_features sets the size of the random subset of features generated at each split. Use a lower value to prevent over-fitting.

I first investigate the effect on the rmse of changing the *learning_rate* and *n_estimators* parameters. A plot of rmse versus *n_estimators* for four different *learning_rate* values is shown below:



Observe that the rmse increases as trees increase for higher values of the learning rate (.05 and .1), indicating that these values are too fast and the tree is over-fitting. Over-fitting is a problem with gradient boosted decision trees, which can quickly learn and over-fit the data without many trees. By lowering the learning rate, we can control and lesson the impact that each tree has on the overall model, and thus prevent over-fitting. Notice that even for lower learning rate values,

over-fitting eventually occurs, but it takes more trees for this to happen than for higher learning rate values, where over-fitting occurs immediately.

It looks like about 300 trees should be adequate with a learning rate tuned to this value. Regarding learning rate values to consider with grid search, I use the rule of thumb provided by the creator of XGBoost, Tianqi Chen, which is tuning the learning rate across the following range: (2-10/number of trees). Tianqi Chen also recommends values to consider for *max depth*, *subsample, and max_features,* so I consider them in the grid search.

A rule of thumb for the parameter *min_samples_split*, the minimum number of samples required to split an internal node, is setting this value to .5-1% of the number of observations in your data. In this case, the training data consists of 70% of the original data, so this means 345 observations. The recommended *min_samples_split* values should then range from 1.7-3.5. I will set the value at 2, which is the default value.

Grid search considers every combination of the specified values for the parameters being tuned and selects the following combination of values. These values yield the optimal negative mean squared error (just like for the random forest model):

learning_rate: .01333333
max_depth: 4
max_features: 1
subsample: .5

I then fit the gradient boosted regression tree with the tuned parameter values, which results in an rmse of .62185.

Variables with the largest feature importances are *mean_review_year* and *review_count* (where the values are normalized to sum to 1), which have values above .2. The *unfriendly* and *unclean* variables see their variable importances go up a little, but their values are still below .1. Feature importances are similar to those in the random forest, except *unclean* contributes more in this model.

Results – Most Important Features

Variables that significantly contributed to each model (except PCR) are shown below:
LR: mean_review_year, unclean, unfriendly, NC, NV, and QC
RF: mean_review_year, review_count
GBRT: mean_review_year, review_count

Variables *unclean* and *unfriendly* are important in LR, whose rmse is worse than the other models, but not by much. The LR model thus suggests these variables might be more important than the *homeless* variable to Yelp reviewers. Improving them for stores with low star scores might make customers happier and improve these stores' star scores.

The fact that *mean_review_year* and *review_count* are important variables in RF and GBRT does not seem as interesting and informative to me as if, say, *unclean* and *unfriendly* had been important variables.

I think improving the models, such as with more features and better constructed features, would potentially improve these results in that: some of the additional variables might be important and thus a better assessment of the relative importance of variables can be made. Also, if existing variables stay just as important after adding new variables, then this fact would add to their legitimacy as important variables. Investigating and potentially implementing additional variables, as well as trying to improve existing variables is definitely needed before taking any action, in my opinion.

Results – Predictability of Models

As for the most predictive model, GBRT produces the lowest rmse, then PCR, RF, and LR. What this means is that for a given prediction of the GBRT (for example), we expect it to be off in either direction by about .622 points (the typical error amount when the model is tested on the test split). These models have good enough predictability to tell us roughly whether customers regard a store favorably or unfavorably (where perhaps a value greater than 3 is favorable). If a store is predicted as having a score of 4.5, then it's actual score is probably at least a 3.5, so we are pretty sure that the store is regarded very favorably. Similarly, a predicted score of 2.0 would indicate a store is regarded unfavorably, since even if this value is too low by .65 or a little more (LR's rmse), the store would still have a low score.

Gradient Boosted Regression Tree (GBRT): 0.62185
Principal Component Regression (PCR): 0.64525
Random Forest (RF): 0.64949
Linear Regression (LR): 0.65436

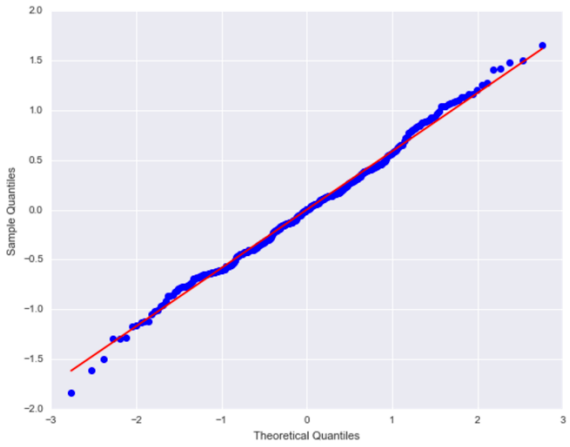
The plot below shows RMSE values centered around the mean RMSE value, which highlights the fact that GBRT does a bit better than the other methods:

Next Steps: Further Research and Recommendations

The models I created provide rough, yet informative predictions of Starbucks Yelp scores, in my opinion. I suspect that improving the models through better feature engineering of the variables I already created, adding more variables from the Yelp data, and adding more variables from outside the Yelp data will potentially create a much more predictive model, as well as lead to a better determination of which variables are important and are the most important.

1) Perform a more sophisticated text analysis or sentiment analysis method of text reviews to better measure the *unfriendly*, *unclean*, and *homeless* variables. Even my simplified approach yielded significant predictors in the linear regression model, so I would expect a fuller capturing of the emotions underlying review content would yield a more predictive model for these variables.
2) Include more variables constructed from Yelp's data, such as from the text review content. Other variables might include a measure of order accuracy, consistency of drink quality, and how well inventory is stocked.
3) Include more variables constructed from data outside Yelp's data, like the number of competitors that are within a certain distance of the store. Also, consider using internal company data at Starbucks, if available.
4) Examine the most important variables within these updated models and for stores with low star scores, consider focusing store improvement efforts on these variables. For instance, if a variable that measures drink quality is more important than a variable measuring store cleanliness in improving star score, then re-train baristas on making drinks.
5) Consider constructing models for subgroups of the Yelp data, like a model for each region or state. Perhaps certain predictors are more of a problem in some states versus others.
6) Use predicted star score as a predictor in a model that predicts a metric that is correlated with star score. This would be needed for stores not yet on Yelp or lacking Yelp activity. Use Starbucks internal company data as a proxy for Yelp data. Perhaps star score is a predictor of profitability five years down for a new store that has no Yelp data.

Linear Regression Model Outputs



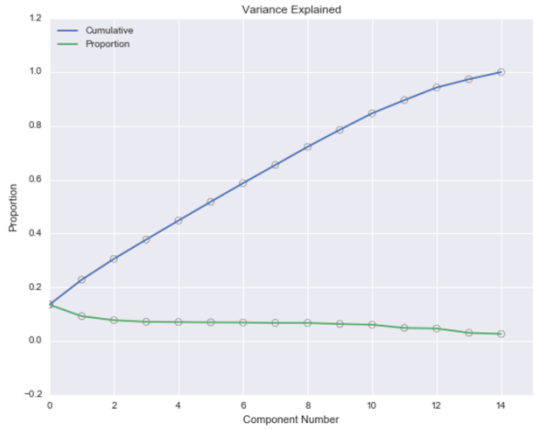
OLS Regression Results

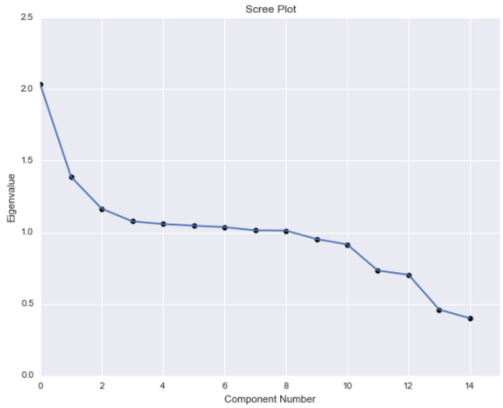
Dep. Variable:	Y_train	R-squared:	0.174
Model:	OLS	Adj. R-squared:	0.137
Method:	Least Squares	F-statistic:	4.632
Date:	Fri, 06 Jan 2017	Prob (F-statistic):	4.62e-08
Time:	14:35:41	Log-Likelihood:	-305.28
No. Observations:	345	AIC:	642.6
Df Residuals:	329	BIC:	704.1
Df Model:	15		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
Intercept	190.2266	75.886	2.507	0.013	40.944	339.509
X_train[0]	-0.0927	0.038	-2.461	0.014	-0.167	-0.019
X_train[1]	0.0027	0.003	0.990	0.323	-0.003	0.008
X_train[2]	-0.2268	0.085	-2.677	0.008	-0.394	-0.060
X_train[3]	0.2897	0.168	1.728	0.085	-0.040	0.619
X_train[4]	-0.2754	0.079	-3.498	0.001	-0.430	-0.121
X_train[5]	-0.9562	0.629	-1.521	0.129	-2.193	0.281
X_train[6]	0.1339	0.176	0.763	0.446	-0.211	0.479
X_train[7]	-0.3327	0.351	-0.947	0.344	-1.024	0.358
X_train[8]	-0.2543	0.126	-2.024	0.044	-0.502	-0.007
X_train[9]	-0.2223	0.077	-2.870	0.004	-0.375	-0.070
X_train[10]	-0.3317	0.282	-1.177	0.240	-0.886	0.223
X_train[11]	0.2439	0.161	1.510	0.132	-0.074	0.562
X_train[12]	0.3968	0.148	2.681	0.008	0.106	0.688
X_train[13]	-0.1314	0.606	-0.217	0.829	-1.324	1.062
X_train[14]	0.0944	0.219	0.430	0.667	-0.337	0.526

Omnibus:	0.774	Durbin-Watson:	1.996
Prob(Omnibus):	0.679	Jarque-Bera (JB):	0.641
Skew:	-0.102	Prob(JB):	0.726
Kurtosis:	3.053	Cond. No.	4.73e+06

Principal Component Regression Model Output





Eigenvalues

```
pd.DataFrame(list(eigenvalues.sort_values(ascending=False)))[0].values
```

```
array([ 2.03111317,  1.38783544,  1.16545276,  1.07699152,  1.05815894,
        1.04701224,  1.03661903,  1.01388638,  1.01261583,  0.95365015,
        0.91596182,  0.7336701 ,  0.70481659,  0.46132689,  0.40088915])
```

Cumulative Percentage of Variance Explained

```
Cum_Variance_Explained
```

```
array([ 0.13540754,  0.22792991,  0.30562676,  0.37742619,  0.44797012,
        0.51777094,  0.58687887,  0.6544713 ,  0.72197902,  0.7855557 ,
        0.84661982,  0.89553116,  0.94251893,  0.97327406,  1.        ])
```

Eigenvectors

```
evals,evecs=scipy.linalg.eig(X.corr())
pd.DataFrame(evecs).ix[:,:10]
```

	0	1	2	3	4	5	6	7	8	9	10
0	0.028407	-0.215979	-0.246843	-0.652006	-0.611572	-0.127973	-0.249898	0.052371	-0.090657	0.009113	0.021932
1	0.580833	-0.762709	0.013750	0.143558	0.071954	0.118615	0.017723	-0.150921	0.098352	-0.028052	-0.031110
2	0.358037	0.308736	-0.056895	0.198740	-0.122576	-0.564572	-0.230091	-0.445720	0.056732	0.268385	-0.000596
3	0.195960	0.063761	-0.039427	0.170444	-0.091368	-0.271491	-0.153967	0.571654	0.521656	-0.063805	-0.254856
4	0.506827	0.467154	-0.172231	0.016039	-0.190056	0.643903	-0.088100	-0.017395	0.055271	-0.021637	0.152139
5	-0.062862	-0.010927	-0.223634	0.385781	-0.470199	-0.050476	0.393868	0.064155	-0.236328	-0.092305	-0.186933
6	-0.135999	-0.126067	-0.342619	0.205413	-0.055733	-0.096568	0.109626	0.034074	0.167147	0.074977	0.809068
7	-0.021255	0.010934	-0.091540	-0.081920	0.159570	-0.152359	-0.082813	-0.107765	0.192614	-0.102942	0.212165
8	-0.065771	-0.007311	-0.399199	0.207581	0.155067	-0.032926	-0.470614	0.017002	-0.169306	-0.658000	-0.063324
9	0.310493	0.137603	-0.471202	-0.360412	0.404012	-0.226116	0.516231	0.124662	-0.126101	-0.050002	-0.098180
10	-0.161264	-0.052438	-0.260945	0.182866	-0.240805	0.081120	0.274298	-0.203859	0.231485	-0.098855	-0.186166
11	-0.107230	-0.087351	-0.342104	0.196370	0.045665	0.103948	-0.139405	0.040978	-0.100544	0.543543	-0.239661
12	-0.272045	-0.047861	-0.349133	-0.161857	0.189191	0.238843	-0.133999	-0.183366	0.480348	0.204580	-0.200335
13	0.056416	-0.047473	-0.053607	0.096956	0.056363	0.026283	-0.140385	0.574466	-0.187655	0.298206	0.159123
14	-0.005911	-0.062996	-0.193285	0.084589	0.159184	0.008439	-0.241890	-0.095559	-0.456741	0.162375	-0.068595

Principal Component Regression Results

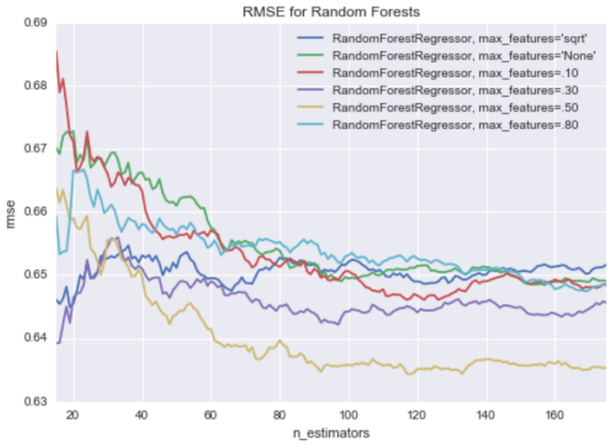
OLS Regression Results

Dep. Variable:	Y_train	R-squared:	0.134
Model:	OLS	Adj. R-squared:	0.108
Method:	Least Squares	F-statistic:	5.156
Date:	Fri, 06 Jan 2017	Prob (F-statistic):	5.01e-07
Time:	14:58:17	Log-Likelihood:	-313.57
No. Observations:	345	AIC:	649.1
Df Residuals:	334	BIC:	691.4
Df Model:	10		
Covariance Type:	nonrobust		

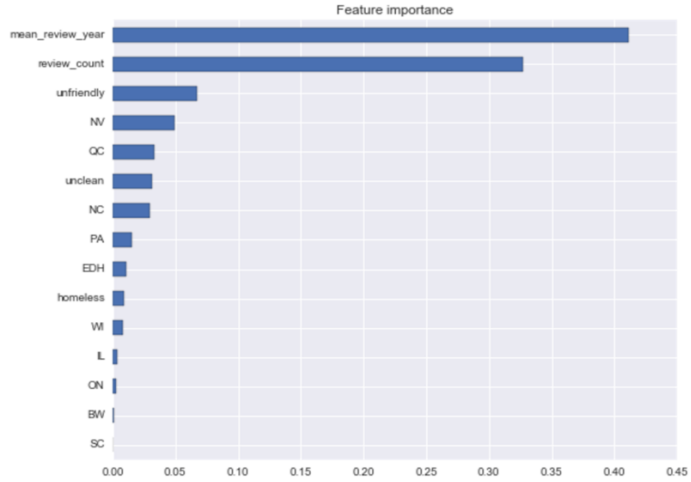
	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
Intercept	3.3044	0.033	100.457	0.000	3.240	3.369
X_train[0]	-0.1194	0.023	-5.245	0.000	-0.164	-0.075
X_train[1]	0.0493	0.028	1.732	0.084	-0.007	0.105
X_train[2]	-0.0180	0.030	-0.595	0.552	-0.077	0.041
X_train[3]	0.0885	0.031	2.830	0.005	0.027	0.150
X_train[4]	-0.0201	0.032	-0.635	0.526	-0.082	0.042
X_train[5]	-0.0340	0.030	-1.147	0.252	-0.092	0.024
X_train[6]	0.0287	0.031	0.916	0.360	-0.033	0.090
X_train[7]	0.0158	0.031	0.503	0.615	-0.046	0.077
X_train[8]	-0.0555	0.029	-1.945	0.053	-0.112	0.001
X_train[9]	-0.0796	0.033	-2.415	0.016	-0.144	-0.015

Omnibus:	0.790	Durbin-Watson:	2.036
Prob(Omnibus):	0.674	Jarque-Bera (JB):	0.687
Skew:	-0.108	Prob(JB):	0.709
Kurtosis:	3.029	Cond. No.	1.56

Random Forest Model Output



Feature Importances

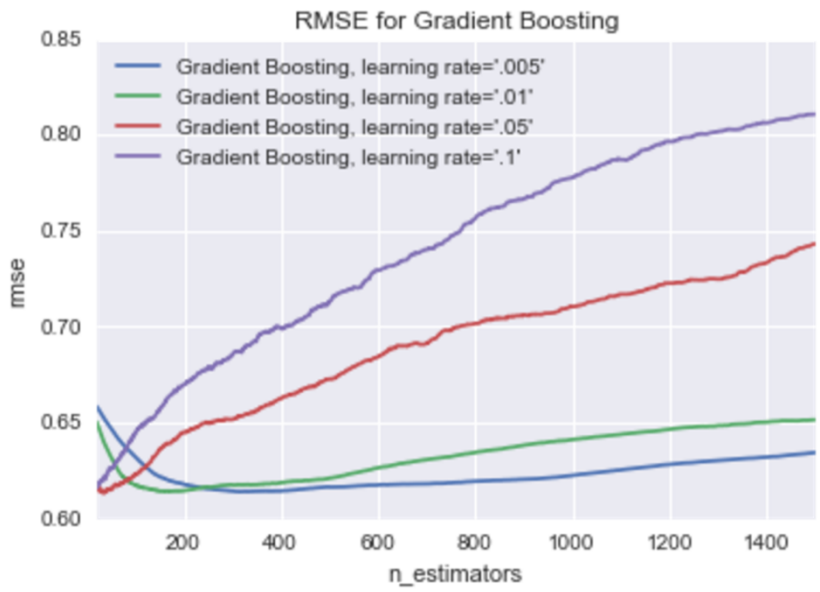


Grid Search Results

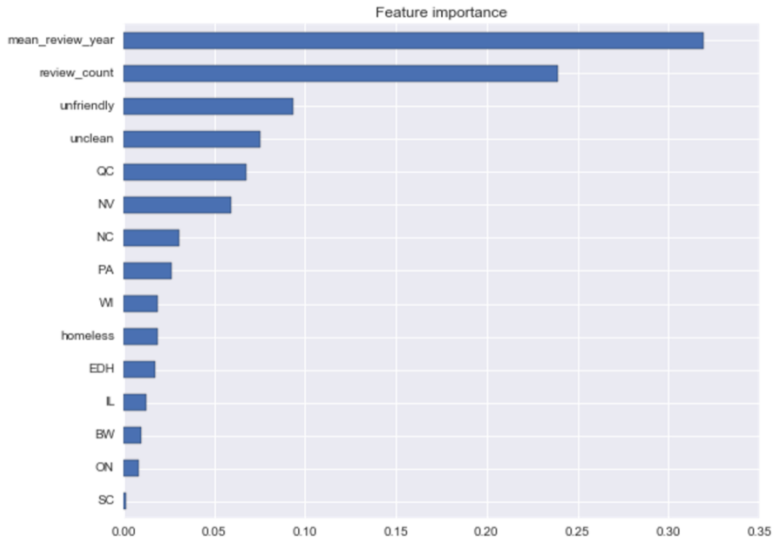
```
from sklearn.grid_search import GridSearchCV
clf=RandomForestRegressor(warm_start=True, oob_score=True,random_state=RANDOM_STATE,
                          n_estimators=300)
parameters={"max_features":[None,'sqrt',.1,.2,.3,.4,.5,.6,.7,.8,.9]}
fitmodel = GridSearchCV(clf, param_grid=parameters, cv=5, scoring="neg_mean_squared_error")
fitmodel.fit(X_train, Y_train)
fitmodel.best_estimator_, fitmodel.best_params_, fitmodel.best_score_, fitmodel.grid_scores_
```

```
(RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
            max_features=0.1, max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=300, n_jobs=1,
            oob_score=True, random_state=112, verbose=0, warm_start=True),
 {'max_features': 0.1},
 -0.3917144605567583,
 [mean: -0.43751, std: 0.07967, params: {'max_features': None},
  mean: -0.39686, std: 0.05727, params: {'max_features': 'sqrt'},
  mean: -0.39171, std: 0.05333, params: {'max_features': 0.1},
  mean: -0.39686, std: 0.05727, params: {'max_features': 0.2},
  mean: -0.40097, std: 0.06106, params: {'max_features': 0.3},
  mean: -0.40865, std: 0.06498, params: {'max_features': 0.4},
  mean: -0.41391, std: 0.06526, params: {'max_features': 0.5},
  mean: -0.41478, std: 0.07162, params: {'max_features': 0.6},
  mean: -0.41954, std: 0.07386, params: {'max_features': 0.7},
  mean: -0.42564, std: 0.07820, params: {'max_features': 0.8},
  mean: -0.42943, std: 0.07796, params: {'max_features': 0.9}])
```

Gradient Boosted Trees Model Output



Feature Importances



Grid Search Results

```
(GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
              learning_rate=0.0133333333333, loss='ls', max_depth=4,
              max_features=1, max_leaf_nodes=None, min_impurity_split=1e-07,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=300,
              presort='auto', random_state=200, subsample=0.5, verbose=0,
              warm_start=False),
 {'learning_rate': 0.013333333333333334,
  'max_depth': 4,
  'max_features': 1,
  'subsample': 0.5},
 -0.3544635981971545,
```