Top Business Attribute Identification

Team: Andrew Toscano 1000808071

Motivation and Objectives

The objective of my project was to identify trends of business attributes and categories, to determine which are the most important for a store to be rating highly. This interested me because I had a belief that things like atmosphere and mood would give influence to a people rating without them directly thinking about it, as opposed as most yelp businesses being rating by the quality of the service or product they receive. I was inspired to use MATLAB for this project after attending a seminar in which they showed Machine Learning techniques that I thought would be useful for my project.

Data mining/analysis tasks tackled

The main tasks I accomplished in this project include data wrangling to convert inconsistent JSON data into a MATLAB tables with usable data. I created decision tree classifiers by using n-fold cross validation sets as well as random forests using the bagged tree method. Within the bagged trees, I evaluated the relative importance of the trees by using out of bag feature importance, which evaluates each feature's importance based on getting the prediction accuracy of one tree, permuting each predictor and getting the change in accuracy for that tree, then averaging the change over every tree in the bag.

Design of methods

The first task was to parse the JSON data, and save it to a MATLAB variable, which is saved to file to read from later. Next was creating classes to convert this to a format which can be fed to built-in classification creators in MATLAB, which is a table. Classification of categories and attributes both went through a similar process, with only the implementation of the methods changing. First the names of the potential columns for the tables had to be found for the whole data set. The names that do not show up often are discarded, and the list of columns is used to create number arrays as the table's contents. String variables are converted to integer equivalents, so that the same string is always converted to the same number to be consistent with other rows of data. Columns that were not present in a row of data were either defaulted to 0, meaning either no or that a string was not specified if the column a string based descriptor. After a table is created, is was fed into MATLAB's Classification Learner, which would allow you to pick the type of classifier, as well as other specifications and compare the results. After a classifier was picked, it was exported as a function and used to predict the entire data set in order to create a confusion matrix showing the accuracy of that classifier. Then a

bagged tree classifier was created and the out of bag feature importance calculated. The feature importance is sorted and the top 10 features are displayed in a bar graph.

Implementation of methods

Proper order of function execution for final display:

Install JSONlab toolbox included in project

Install Customizable Heatmap toolbox included in project

readYelpJson.m

displayCategoryClassifierResults.m

displayAttributeClassifierResults.m

allCitiesAttributeResults.m

readYelpJson.m parses every line of JSON data separately and places it in a struct. Every line must be passed separately because when JSON parsing, all lines must use a similar format, or have a format allowing for all possibilities established in the beginning. Neither requirement can be met reasonably because the data is inconsistent with the number of values in key areas such as categories and attributes. It uses a MATLAB Toolbox add-on called JSONlab to perform JSON parsing.

Methods called in displayCategoryClassifierResults.m

getCategoryNames.m

setupCategoryBusinessData.m

trainCategoryClassifier.m

displayBarGraph.m

displayCategoryClassifierResults.m converts the business data into a usable format, creates a classifier and a confusion matrix of its accuracy, creates a bagged tree classifier then identifies and creates a graph containing the top important categories and the value of their importance.

It loads the business data from the variable created in readYelpJson. It calls **getCategoryNames.m** over the business data which returns a list of category names to be used as table columns. The categories of days of the week, Monday, Tuesday, etc. are added to the list at the beginning, as they are also used in the category classifier to see which days of the week its open on. It iterates over every row in business data and gets each category listed. Only the first word of the category is used to combine inconsistency throughout naming categories. It is added to a list of unique categories if it has not been used once yet, and it is added to a list of every category name listed, which allow duplicates purposely. At the end, every category in the unique list of attributes is used to count how many times it appears in the list with duplicates. If the category shows up less than 20 times, we assume it is an uncommon category and do not add it to the final list of categories. This results in there being over 200 unique categories being either merged into other categories or being ignored for being an underused category This list of categories is returned after adding a final value for high rating, which will be used as the response in classification..

The category table is created by **setupCategoryBusinessData.m** which is passed the category names and the business data. It establishes the table data as an array of zeroes with rows equal to the business data rows and columns equal to the size of category names. It goes through every row of the business data and reads through every category it is a part of, finds the proper column that it should belong it, and changes the table data in that position to a 1. The rating of the business is looked at, and a high rating is established if the rating is 3.0 stars or greater. Each row of table data will end up having a 1 for categories it belongs in and 0 for categories it doesn't. The table is created and returned.

The table is then sent to **trainCategoryClassifier.m**, which is code to generate a 5 fold cross validated decision tree. This code came from MATLAB's Classification Learner, and these specifications were tweaked until the optimal classification rate was found.

This classifier is used to predict over the entire business data set. The predicted values are compared to the known values in the business data, and a confusion matrix of the classification rate is created using the Customizable Heatmaps MATLAB toolbox and saved to file.

Next, the same predictor and response data from the classifier and used in the TreeBagger method, which creates an ensemble of bagged trees, with the variable importance computation set to on. The PermutedVarDeltaError is taken from the tree bag, which is used as a score to evaluate feature importance. The top 10 columns are taken, and sent to **displayBarGraph.m** which accepts the values and column names for the table as well as the x and y axis labels, the title, and the filename that it is saved to. This function saves it to categoryOOBImportance.fig. The final outputs of displayCategoryClassifierResults.m are the mentioned bar graph and confusion matrix. The total runtime of is about 30 minutes because the table created from

setupCategoryBusinesData.m has over 400 predictors, and creating decision trees over that many predictors is very expensive to compute.

Methods called in displayAttributeClassifierResults.m

getAttributeNames.m

setupAttributeBusinessData.m

trainAttributeClassifier.m

displayBarGraph.m

displayAttributeClassifierResults.m follows the same general structure as displayCategoryClassifierResults.m but has differences in implementation of methods. The classifiers will classify the attribute data instead of category data. The business data is loaded and then passed to **getAttributeNames.m**, which goes through every row of business data, looks at its attributes column, then finds the name of every field in attributes, and ads unique field names to the attribute list. The count of the attributes does not need to be limited by results because there is significantly less attribute names then possible category names, and they do not need to be merged because they are more consistent with each other so that duplicates with slightly different listings do not exist. At the end the last value for high rating is added to be used as a response in the classifier later.

setupAttributeBusinessData.m establishes the table data as an area of zeroes the same size of the length of business data and the number of names in the attribute list. It goes through every pair of field name and values in the attribute column in the business data and passes the value to the appropriate column of the data based on where the attribute list matches the field name. The area must be all integer values, so if the value is a string, it is added to an array of unique values if it isn't already a member and the index of the string in that array is used as a value. If the attribute has multiple values(ex. Good for: {breakfast = 0, lunch = 0, dinner = 1}), the string name of the field that contains 1 is used as a string value. The final column for rating is checked in the business data and set in the table data as 1 if it is higher than 3 stars and 0 if not. A table is created with the attribute names and the table data and returned.

The table is used as in input to **trainAttributeClassifier.m**. This is code generated from the Classification Learner to use the attribute table to create an ensemble of trees with 5 fold cross validation. The final result of displayAttributeClassifierResults.m is a confusion matrix of the

initial classifier and the Because this attribute table is smaller than the classification table because there are only about 40 predictors, it is possible to make more complex classification choices when determining prediction accuracy.

The trained classifier is then used to predict the entire dataset rather than the cross validated sets. These values are compared to the known values in the business data and a confusion matrix is created and saved.

The same data is used to create another classifier using the Bagged Tree model with Out of Bag Variable Importance set to true. We create another classifier because this does not use cross validation, allowing us to have more possible trees with similar runtime. The OOBPermutedVarDeltaError is taken from the tree bag, and the top 10 values and attribute names are sent to **displayBarGraph.m** which creates a bar graph of those names and values and saves it to 'attributeOOBImportanceGraph.fig'. The final outputs of displayAttributeClassifierResults.m are the mentioned confusion matrix and bar graph. The total run time of displayAttributeClassifierResults.m is about 15 minutes, even while using more complicated classifiers, the lower amount of predictors makes this process much faster to classify.

Methods called in allCitiesAttributeResults.m

getAttributeNames.m

getCityNames.m

getCityCount.m

setupCityAttributeBusinessData.m

displayAttributeResultsFunction.m

displayBarGraph.m

allCitiesAttributeResults.m separates the business data by city and then runs data for each city similarly to displayAttributeResults.m, and output a classification matrix and bar graph for each city. After importance for every city completes, the number of times a certain attribute appears is counted for all cities. The top 10 highest count cities are displayed in a bar graph.

It begins by loading the business data an passing it to **getAttributeNames.m** This is the same function used earlier. **getCityNames.m** returns a list of city names which were given on the Yelp Dataset Challenge page as the important cities.

The list of city names is then looped through to run each value once through the following. **getCityCount.m** gets the number of times the city appears in the business data, which is needed to create a table of the proper size for classification. The city is passed to **setupCityAttributeBusinessData.m** is the same function used earlier, setupAttributeBusinessData.m, that sets up a table for classification, but with provisions to only use the rows of data where the city matches the city passed in.

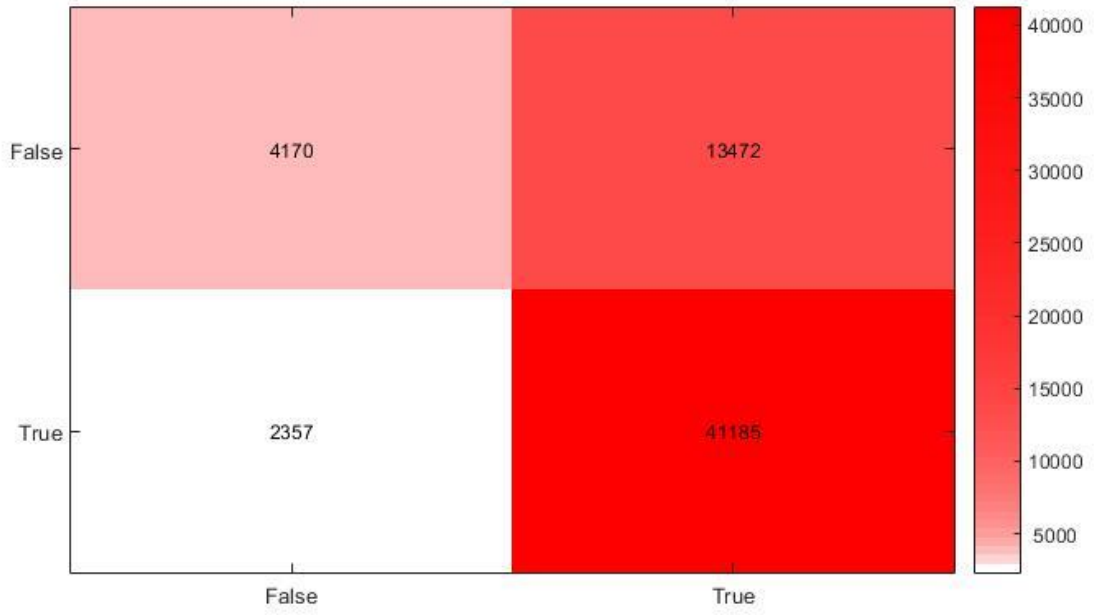
The table is sent to **displayAttributeResultsFunction.m**, which runs very similarly to displayAttributeResults.m. It creates a bagged tree classifier with cross validation to evaluate the classification rate and creates and saves a confusion matrix. It then creates a larger bagged tree classifier without cross validation to determine feature accuracy and creates a graph of the top 10 results and returns them. The results for every city are added to a list with repeats allowed.

After all the results have come through, the top 10 attributes by the count that they appeared are found, and passed to **displayBarGraph.m.** This creates a bar graph of these result and saves is as 'cityFeatureCount.fig'. The final outputs of cityAttributeResults.m are confusion matrices and bar graphs for every city and a final bar graph of top results for all cities. The total runtime is about 25 minutes.

Results and Visualization of Outcome

Because of the random factors in creating bagged tree models, the values of feature importance can slightly vary, and in some cases may change results (pushing a value out of top 10). However, we use a large enough number of trees in attribute classification to get an accurate result.

displayCategoryClassifierResults.m

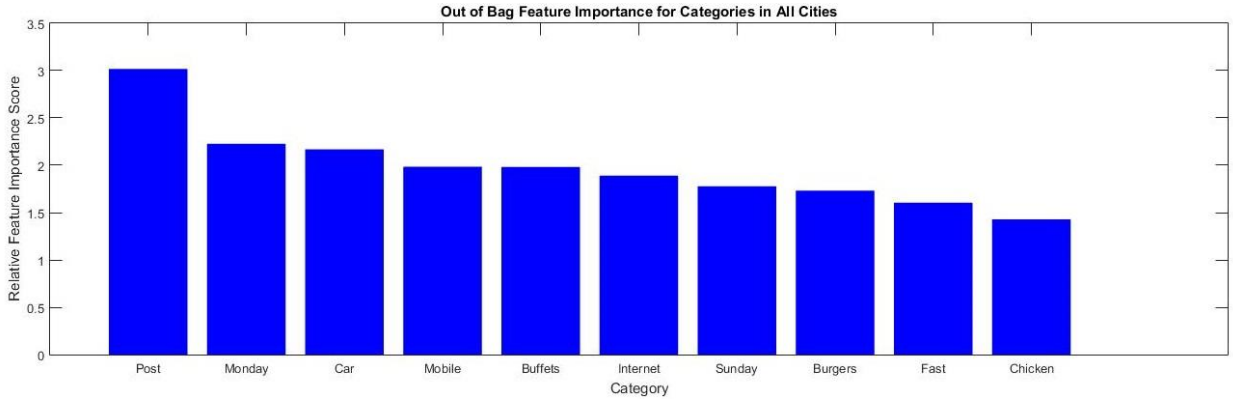


True Negative Rate – 24%

False Positive Rate – 76%

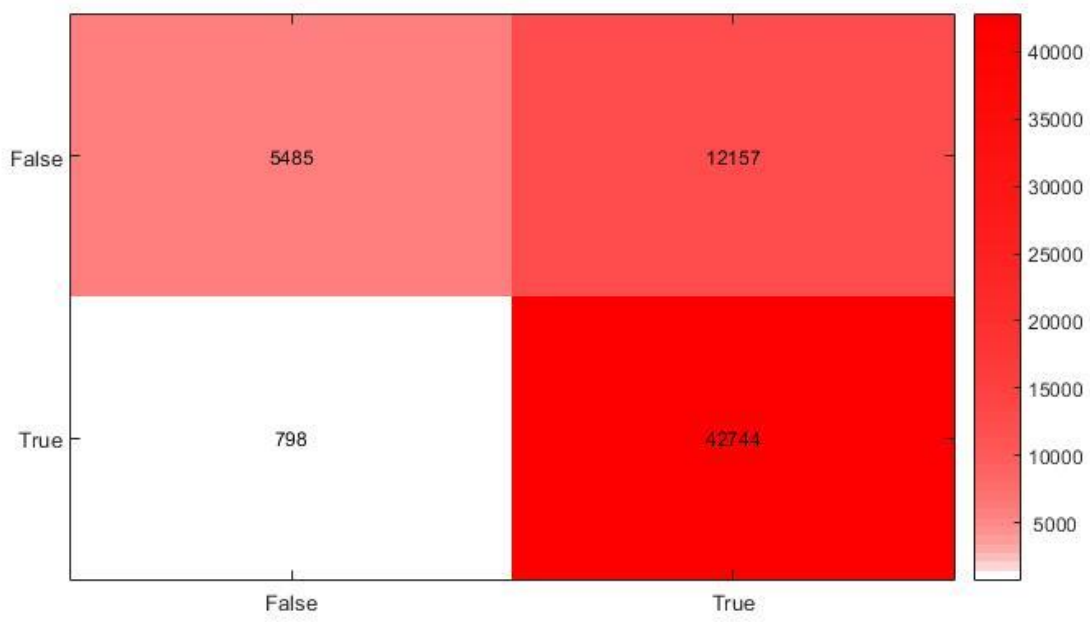
False Negative Rate – 5%

True Positive Rate – 95%



The results are somewhat random as expected, and the high false positive rate lowers my confidence in the accuracy of these results. It may not make sense to make a classifier over categories and opening times

displayAttributeClassifierResults.m

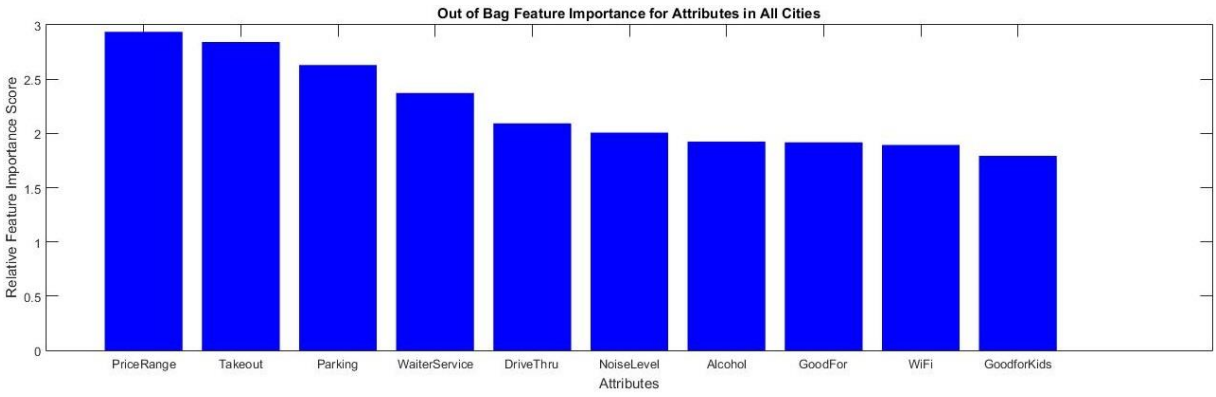


True Negative Rate – 31%

False Positive Rate – 69%

False Negative Rate – 2%

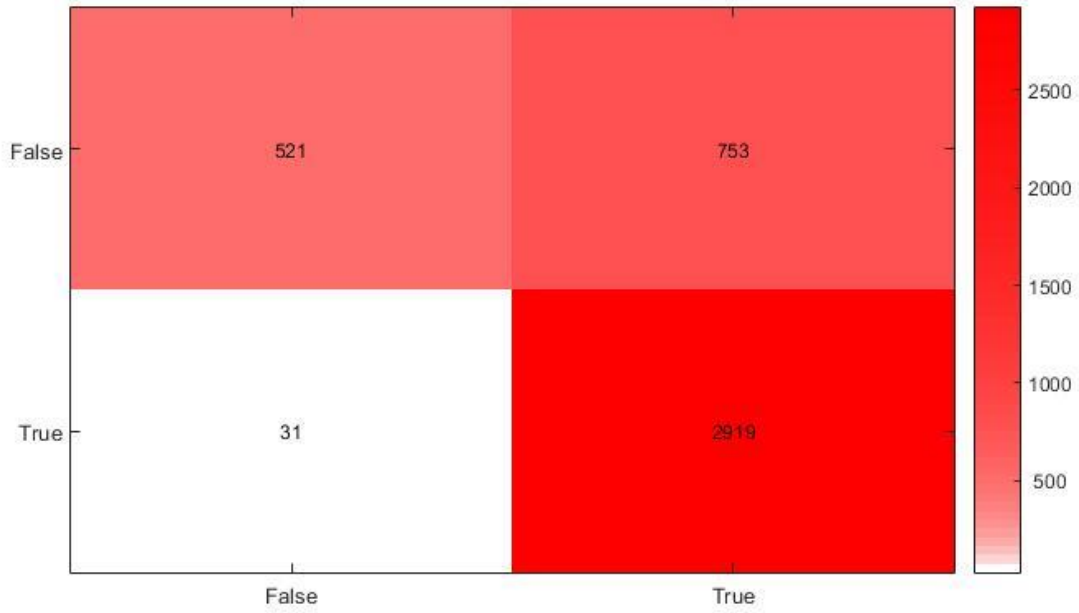
True Positive Rate – 98%



Although the classification rate is also poor in this class, the results seem to make sense from a common sense point of view. These results will be compared to every cities individual and combined results.

allCitiesAttributeResults.m

Charlotte

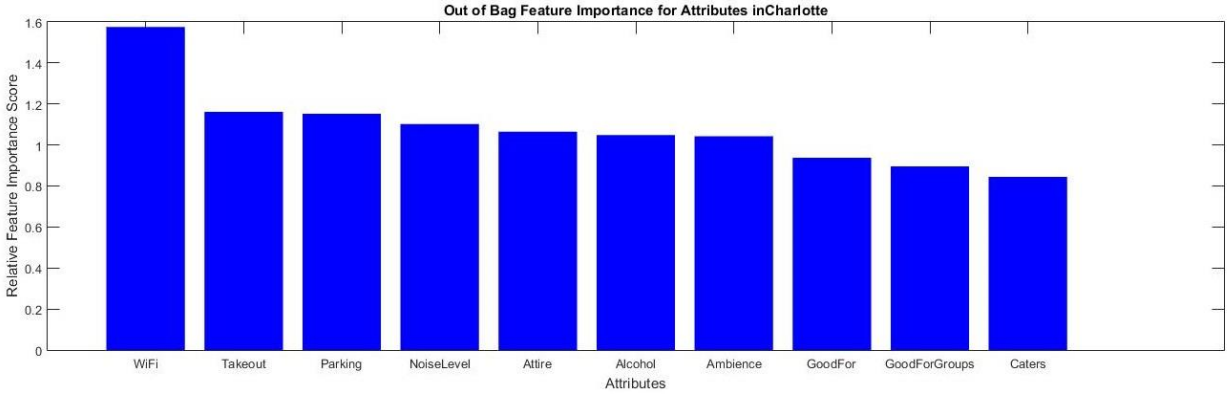


True Negative Rate – 40%

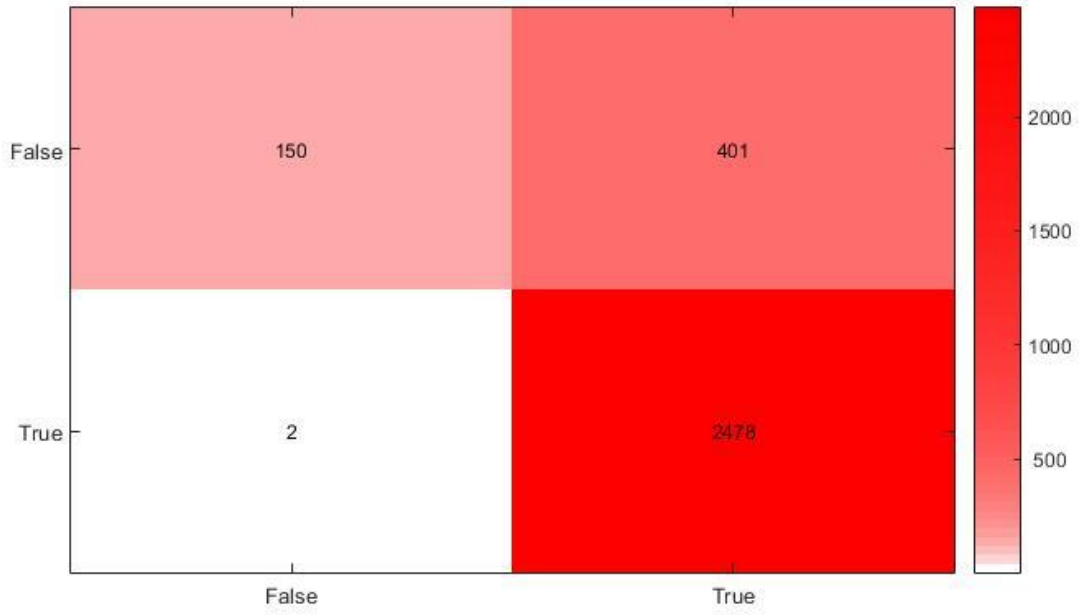
False Positive Rate – 60%

False Negative Rate - 1%

True Positive Rate – 99%



Edinburgh

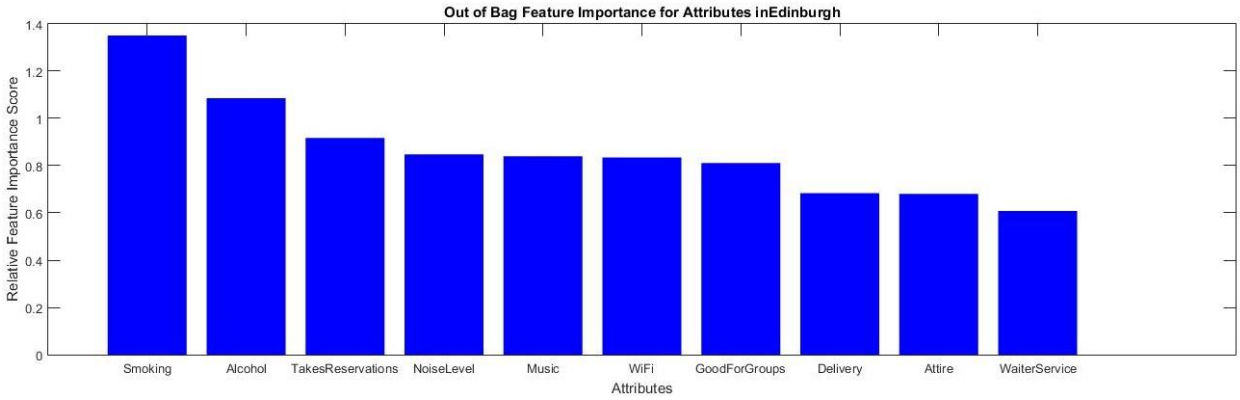


True Negative Rate – 27%

False Positive Rate – 73%

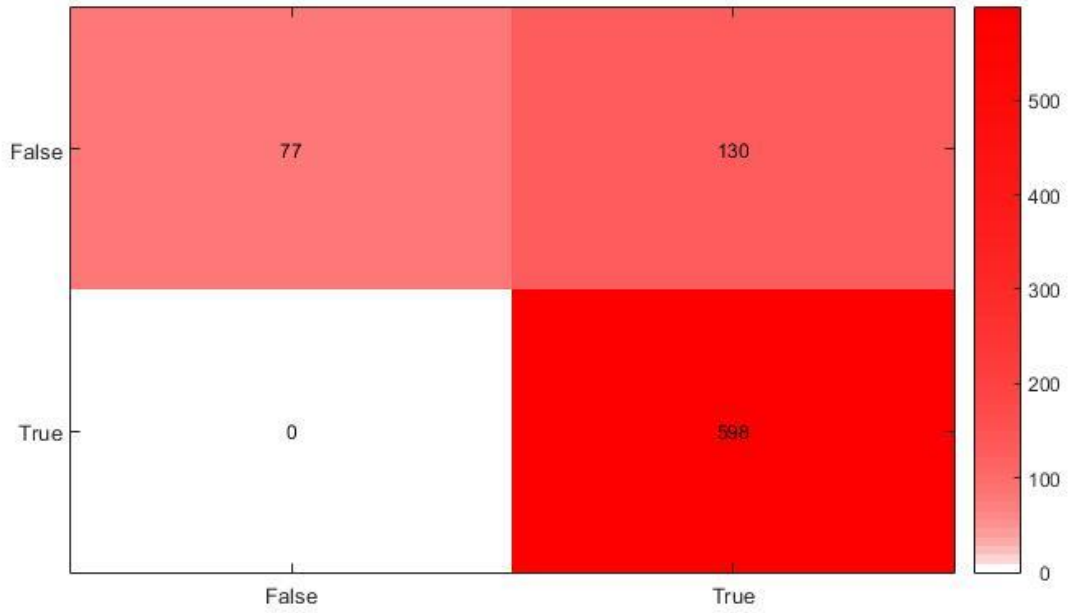
False Negative Rate - < 1%

True Positive Rate – 99%



It is important to note that this city is in Scotland and not in the United States. Although many of the top categories are similar to other cities, the smoking feature is very important in Edinburgh and not common in others. It is also the only city of the top cities which Takeout isn't in the top 10 features.

Karlsruhe

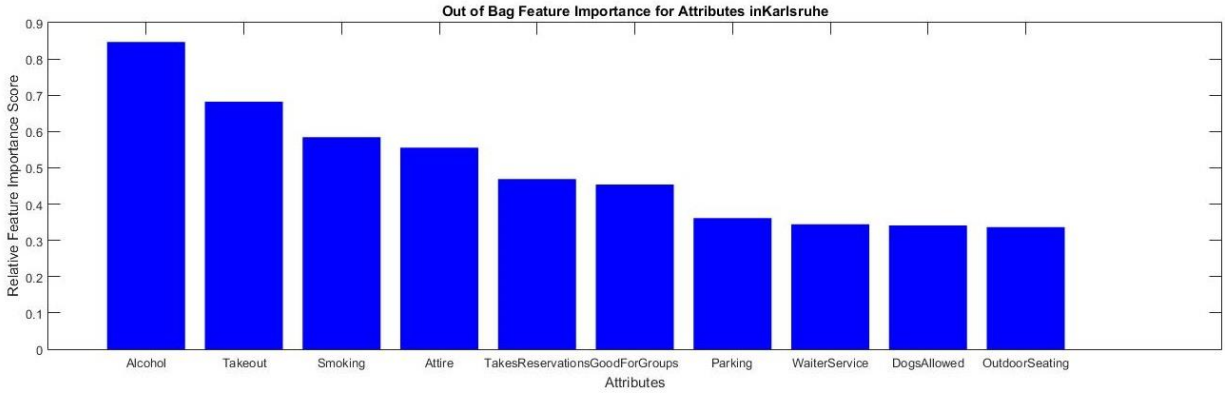


True Negative Rate – 37%

False Positive Rate – 63%

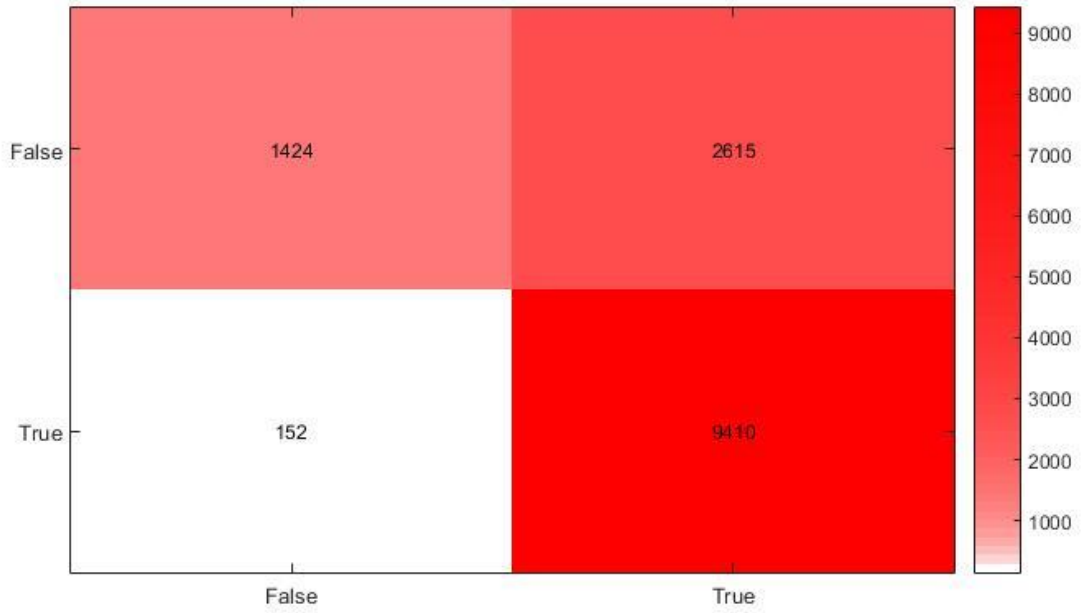
False Negative Rate – 0%

True Positive Rate – 100%



This city is in Germany and once again we see the smoking feature appear. It is interesting to note that smoking feature does not appear in any of the United States results with the exception of Las Vegas, but is very important in both cities outside of the US.

Las Vegas

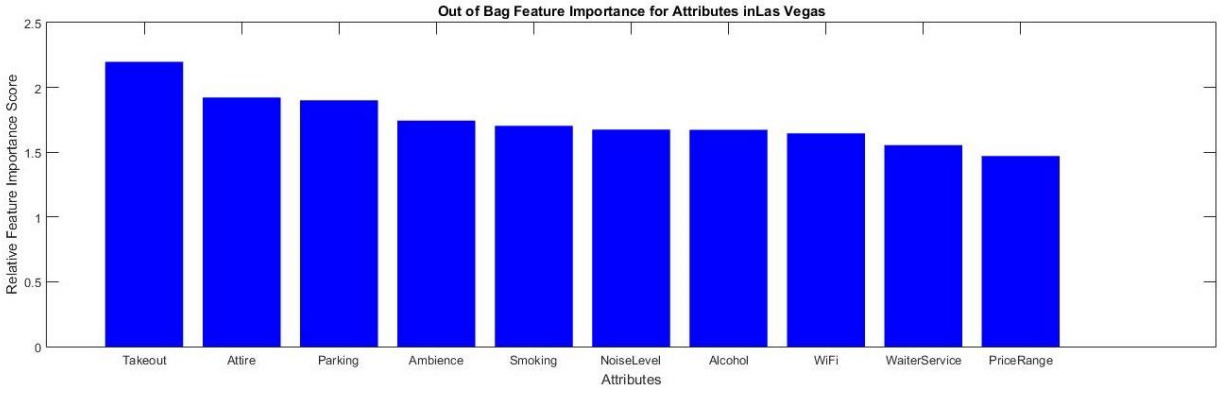


True Negative Rate – 35%

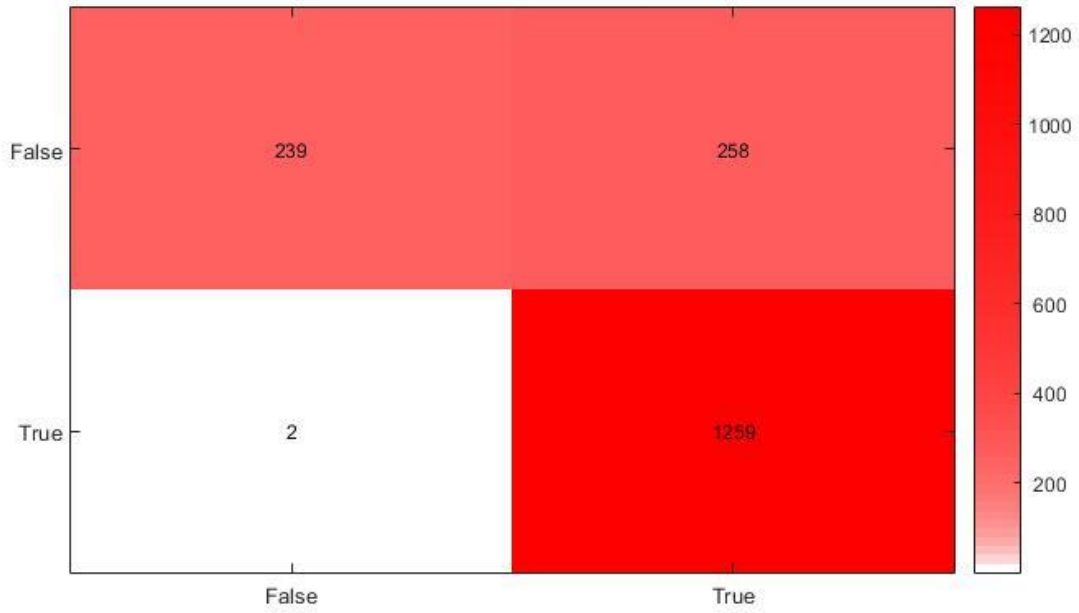
False Positive Rate – 65%

False Negative Rate – 1.5%

True Positive Rate – 98.5%



Madison

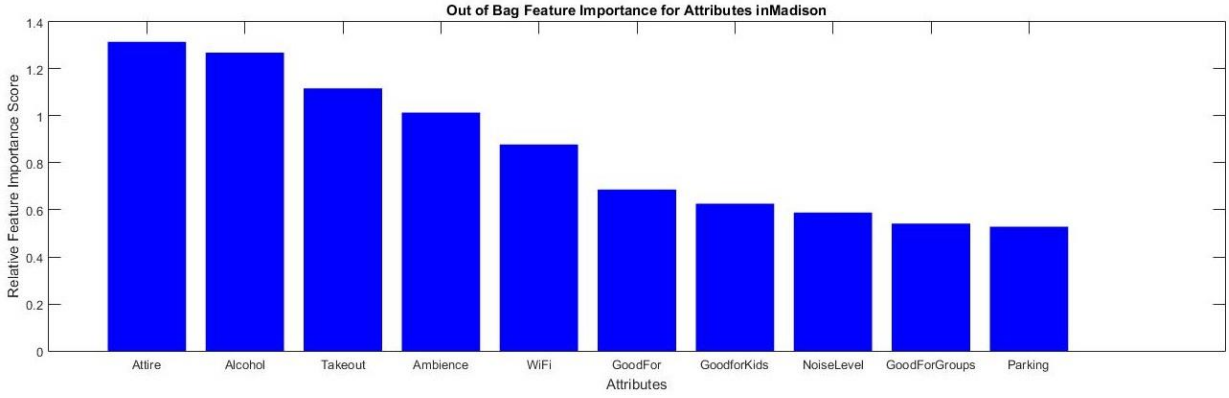


True Negative Rate – 48%

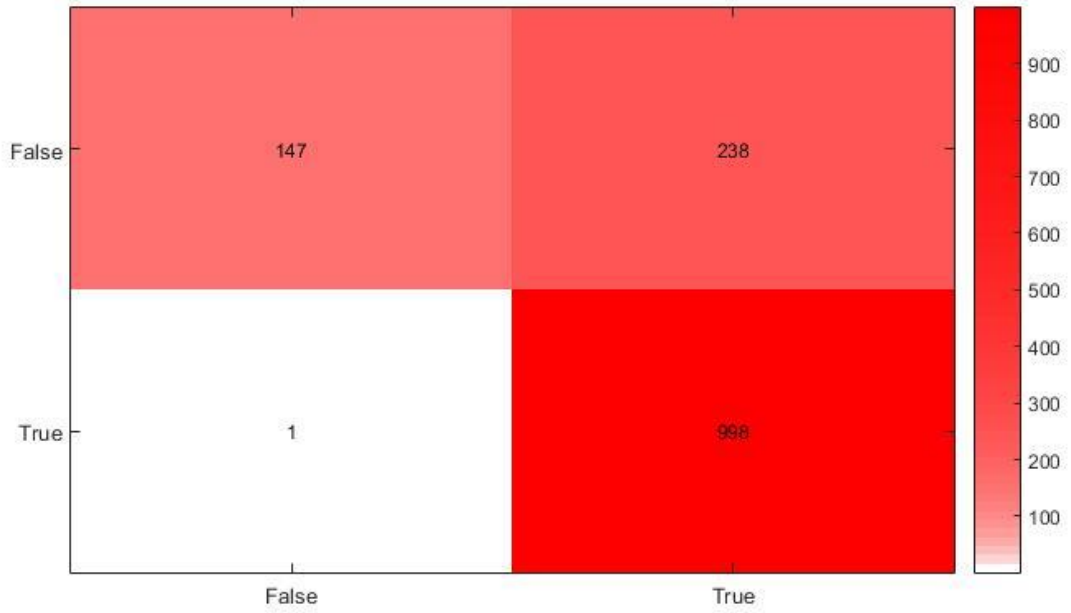
False Positive Rate – 52%

False Negative Rate - <1%

True Positive Rate – 99%



Montreal

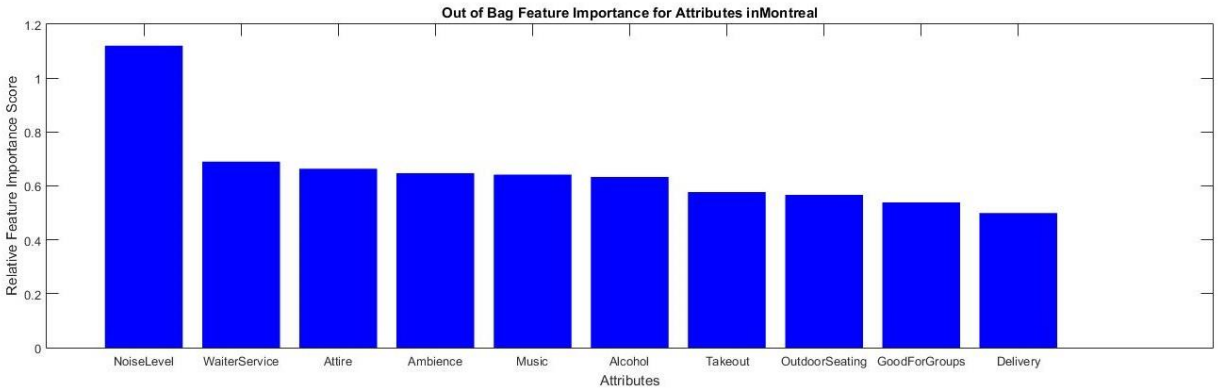


True Negative Rate – 38%

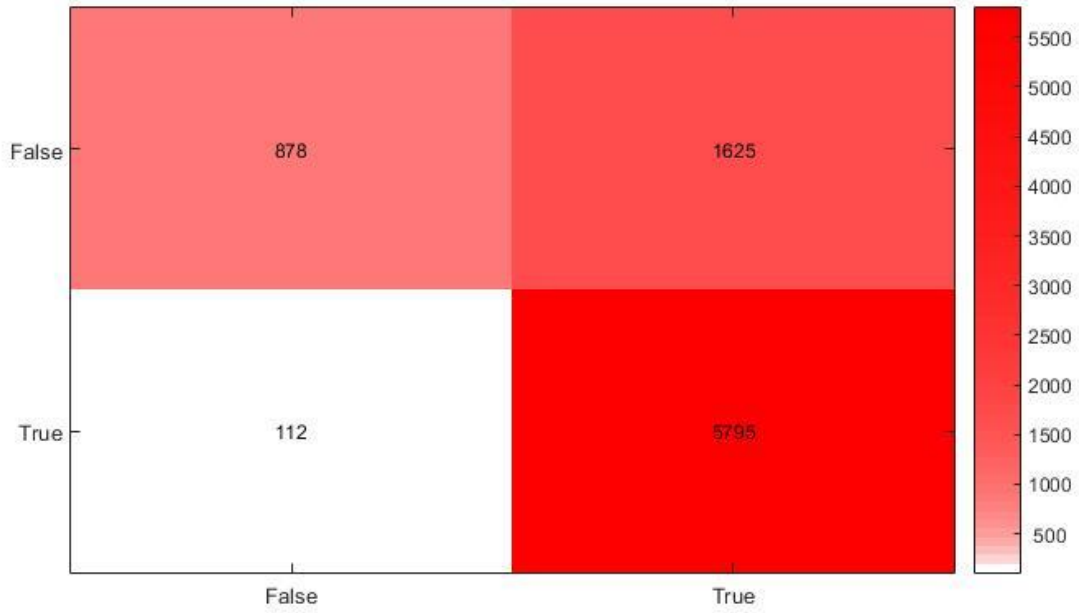
False Positive Rate - 62%

False Negative Rate - < 1%

True Positive Rate – 99%



Phoenix

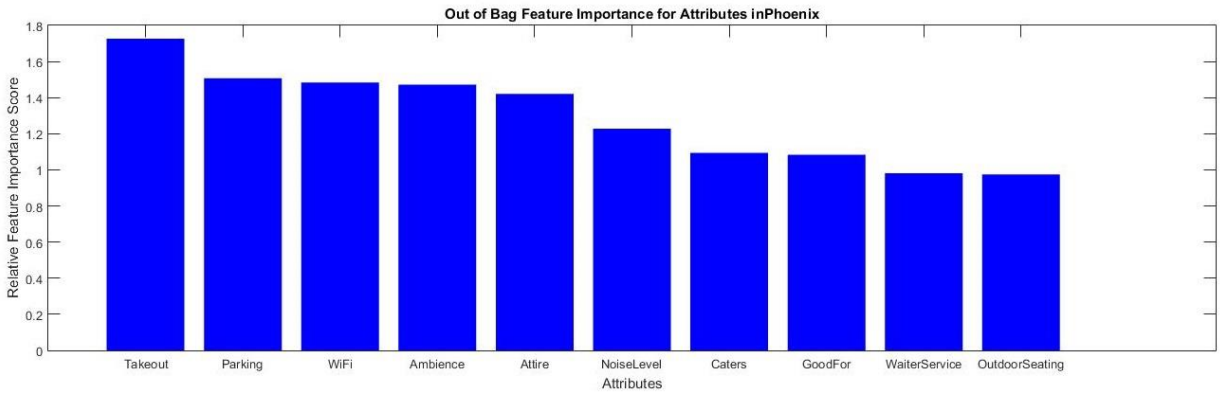


True Negative Rate – 35%

False Positive Rate – 65%

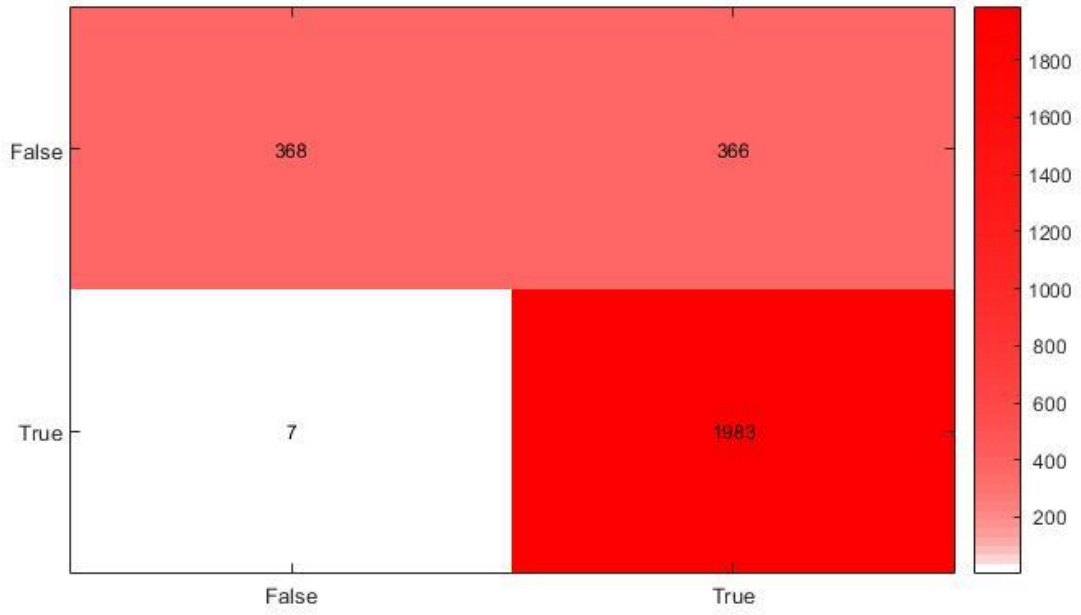
False Negative Rate – 2%

True Positive Rate – 98%



This is the only city in which Alcohol does not reach the top 10 results.

Pittsburgh

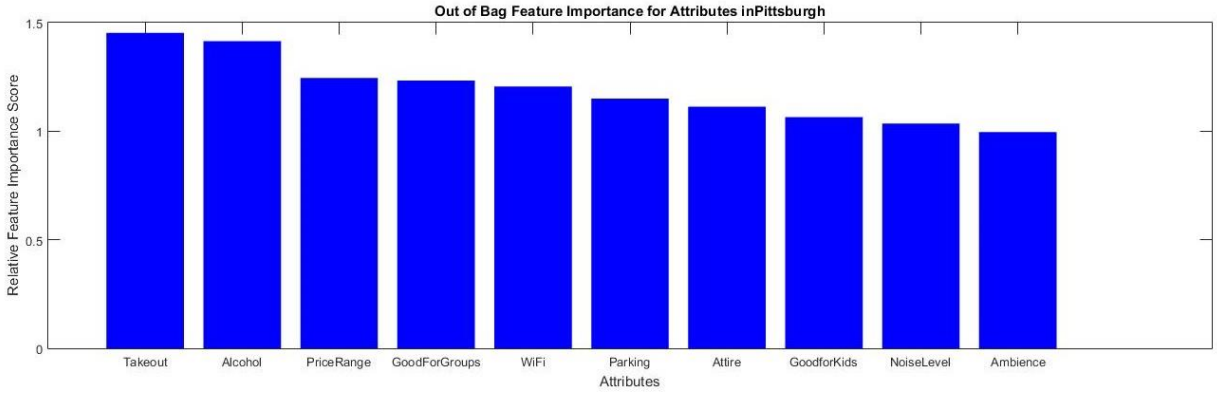


True Negative Rate – 50%

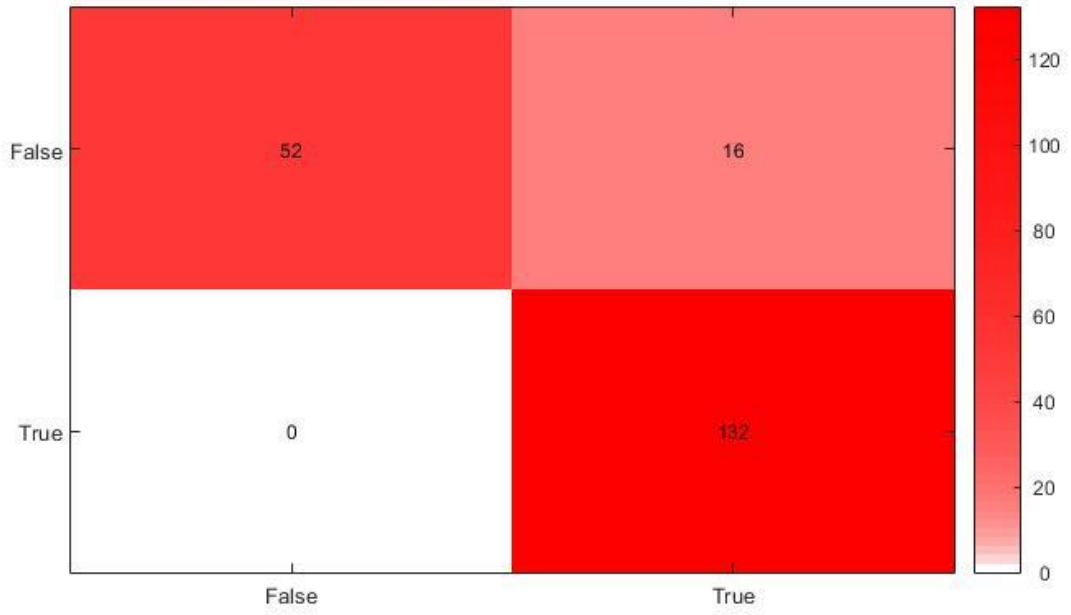
False Positive Rate – 50%

False Negative Rate - <1%

True Positive Rate – 99%



Waterloo

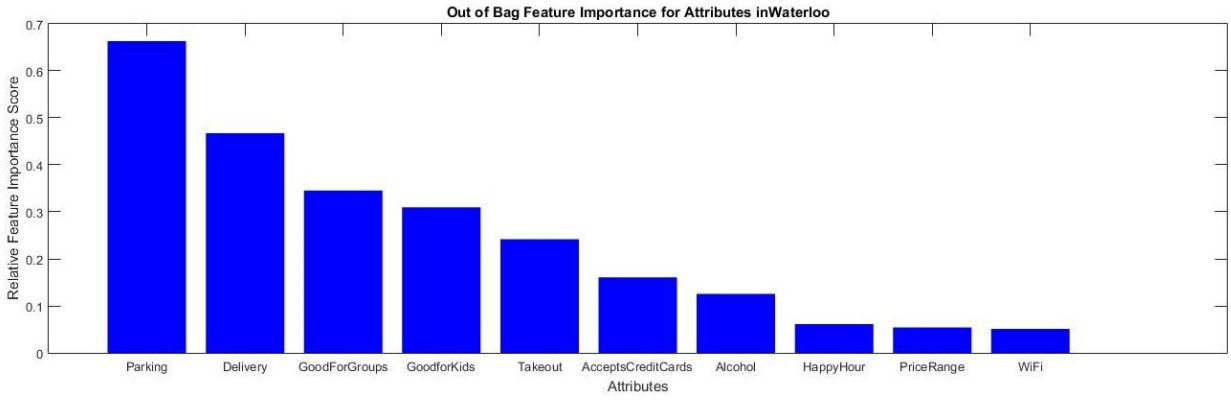


True Negative Rate – 76%

False Positive Rate – 24%

False Negative Rate – 0%

True Positive Rate – 100%

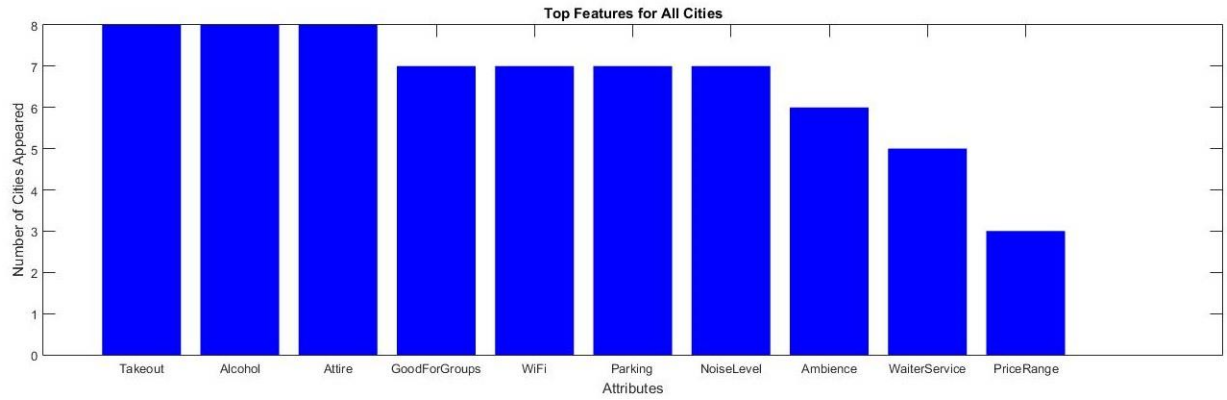


This is the most accurate classifier of the project. However the sample size for this city is much smaller than the other cities, and I believe the tree may have been overfitted, considering the

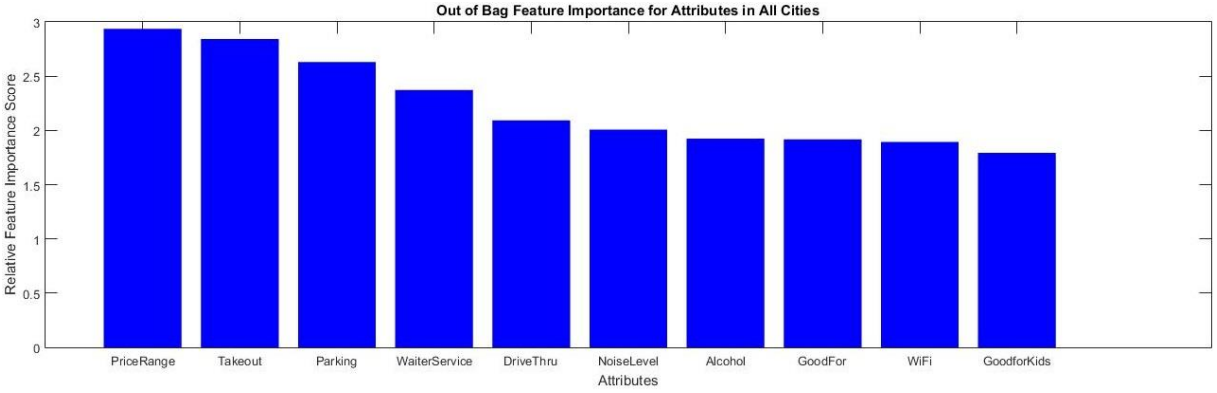
same size trees were created for every city. This is the only city which doesn't have Attire in the top 10 features

All Cities

This graph is a count of every feature in the 9 major cities' results.



When looking at the individual cities, usually the higher ranked categories in this graph correspond to higher ranked categories in the individual city graphs. When we compare this graph to the graph created earlier from a classifier…



We can see similar results in the top 10. The most major rating difference is that Price Range goes from 10th to 1st when changing evaluation methods.

Evaluation

The results for the attribute classifier make sense in general. For example, the Smoking attribute should not be popular in the US because many states have smoking bans in restaurants. But this does not apply to cities outside of the US or Las Vegas. Restaurants seems to be the most numerous businesses which the Yelp dataset covers, and all of the top valued attributes are important things that people may consider either consciously or unconsciously.

However it is difficult to evaluate the efficacy of this project because of the classification rate in every classifier used. Although the results make sense, it is because of the classification rate that I cannot say it is completely accurate. Excluding Waterloo, there is a very high false positive rate and well as a very low false negative rate. This implies that most businesses are rated over the threshold of 3 stars, and it would be difficult to evaluate which businesses are bad because they share some attributes of good businesses. Rather than saying it is because of these attributes that businesses value can be calculated, I believe it is more of a 'correlation without causation' scenario. I believe that businesses with good food or service will already know that these attribute points are important for customer service, and so there will be a correlation between these positive business attributes and a good restaurant or business.

Website

http://andrewtoscano.github.io/6339Project/