**Milestone 3: Capstone Final Report**

**Eowyn Baughman**

**Seattle Police Department 2017 Call Logs and Response Time**

**1). Problem and Key Insight: high-level question and answer from the modeling analysis**

* **Describe the business problem and background**

The goal of this project is to determine how (and how well) Seattle Police Department response times are predicted by characteristics of the incident report including day/time, location (neighborhood, precinct), assigned priority, content of calls, and type of report (visible due to officer-on-scene or other modes). The target variable is “Fast Response Time” which is defined as a less than five-minute difference between an initial call or report being made and an officer arriving on the scene of the incident. This binning was arrived at following exploratory data analysis.

* **Summarize the key findings/conclusions (from after performing model evaluation)**

**Summary:**

Fast police response times can be accurately predicted using a combination of dispatcher prioritization and call type (visible = officer on scene being most positively predictive). Considering the hypothetical in which a dispatcher is not available to interpret the circumstances to set a priority, we are still able to accurate predict fast responses using a slightly larger number of features related to call type, call content, and neighborhood. The most important factor is how officers are distributed across the city. This appears to be set fairly well according to time/day/month. Dispatchers do a great job setting priorities. Downtown really does have a crime problem. North Precinct really does have slower response times, though what it needs to improve this is more cops on the beat.

**Detail**:

Strong linear correlations were observed between a fast response and call type visible, which were also visualized in histograms. The other call modes are all negatively correlated with fast response. There is a positive correlation with priority, which makes sense. There is a negative correlation with the north precinct, which helps support the idea that they need more resources to effectively respond to crime. The time of day, day of week, and month of year do not have a strong correlation, suggesting that staffing levels are already well-matched to "demand" as it were.

The decision tree does quite well at predicting fast responses. It can achieve >90% accuracacy and >0.9 AUC with the features 'Call Type\_visible', 'Scaled\_Priority','Call Type\_building-alarm', 'Call Type\_non-emergency-call', 'Call Type\_emergency-call'. The choice of variables closely resembles the linear correlations. Precinct\_North did well on linear correlations, but is absent from the decision tree feature selection -- indeed, the best set of features is entirely related to the prioritization and the call type.

As suspected, Call Type\_visible -- officer already on the scene -- is a very important predictor. Call type visible is positively correlated with prioritization, but negatively correlated with the north precinct. This was explored by dropping the prioritization, selecting a different subset of features, and training models on that subset.

If the assigned priority is not available, we can still train a decision tree to produce >90% accuracy and >0.9 AUC -- it just takes more features to do so. The call type is still most predictive, but now location info (precinct north, the downtown business area, SODO), the call hour, and the token\_prob (related to the call content) are all contributing to the model prediction. This can be thought of as what would drive response time if we didn't have a dispatcher setting prioritization. It suggests that the North precinct has a point about resources, and Downtown/SODO are right to have reputations as crime hot-spots. And by comparing the two models, we can say the dispatchers are doing a pretty good job setting priorities. That was essentially the reason for abolishing the non-emergency phone line in 2018 -- and it seems to bear out.

Random Forests, and their improved versions (Gradient Boosted Random Forest, Ada Boosted Random Forests) perform somewhat better than the decision trees. In the discussion on model performance, I will explain why the Ada Boosted Random Forest is the best choice for this problem.

* **Suggest next steps to take action or improve the models.**

Following the selection of the Ada Boosted Random Forest, several improvements could be made to improve the performance of the model:

* Perform grid search to optimize all hyper-parameters (slow!)
* Use a larger forest (more trees); sqrt(p) would be too slow on my laptop but a server could handle it
* Use techniques such as SMOTE to rebalance the training dataset, which currently has many 0.0 response times and fewer > 30min response times
  + Which is a good thing, considering police speediness, but a bad thing if you want to classify all classes equally well

Additional machine learning models could also be tested on the problem. Support Vector Machines and Neural Networks are both strong choices for classification problems, particular if there are non-linear relationships and/or combinations of features that provide predictive capacity that individual features do not.

**2). Data: quick overview of the data including 1 or 2 visualizations**

* **Report the source(s) of the data**

There are three data sources for this project:

1. City of Seattle Initial Police Call dataset, limited to 2017 (.csv file from https://data.seattle.gov/Public-Safety/Call-Data/33kz-ixgy) – initially limited to 2017 using vim as file was very, very large

2. Text file (.csv) of neighborhood central location (latitude, longitiude) and neighborhood name (https://gis-kingcounty.opendata.arcgis.com/datasets/neighborhood-centers-in-king-county--neighborhood-centers-point/data) – converted to geospatial form (.kml) via online tool (http://www.earthpoint.us/ExcelToKml.aspx )

3. Geospatial file (.kml) of the SPD beat polygon boundaries valid in 2017 (https://data.seattle.gov/Public-Safety/Seattle-Police-Department-Beats/nnxn-434b)

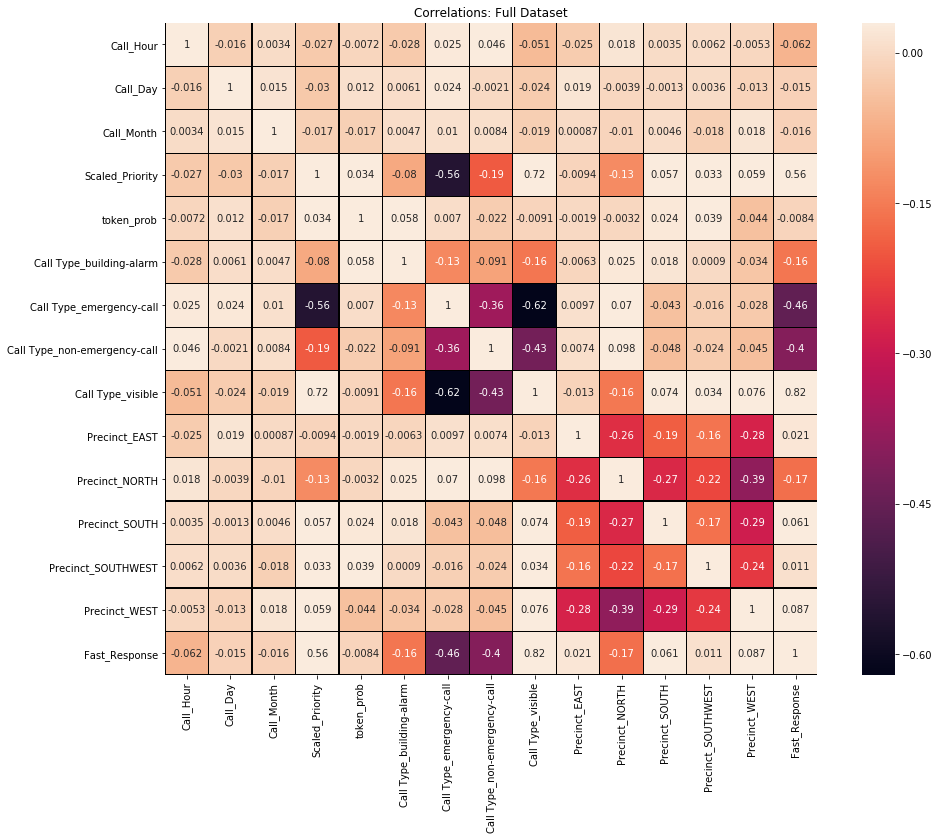
Datasets (2) and (3) were mutually mapped and a lookup-table of neighborhood centers most closely associated with beat boundaries was manually generated. Additional neighborhoods were added (e.g. Belltown, SODO) that were missing, and neighborhoods with too much overlap (e.g. Greenlake, Crown Hill) were consolidated. The resulting file is named neighborhoods\_to\_beats.csv.

* **Include profile report**

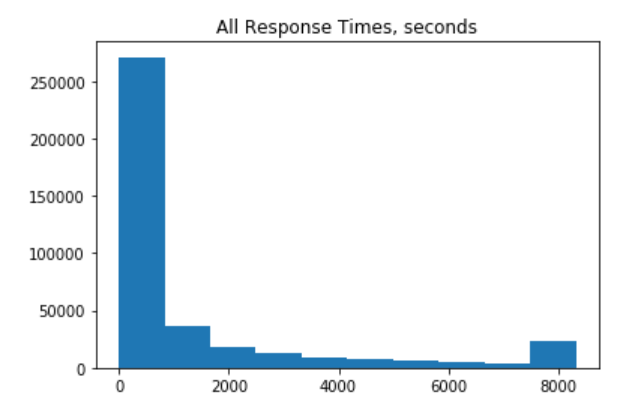
Profile report is submitted as separate file. This is the profile report following the data cleaning and merging processes.

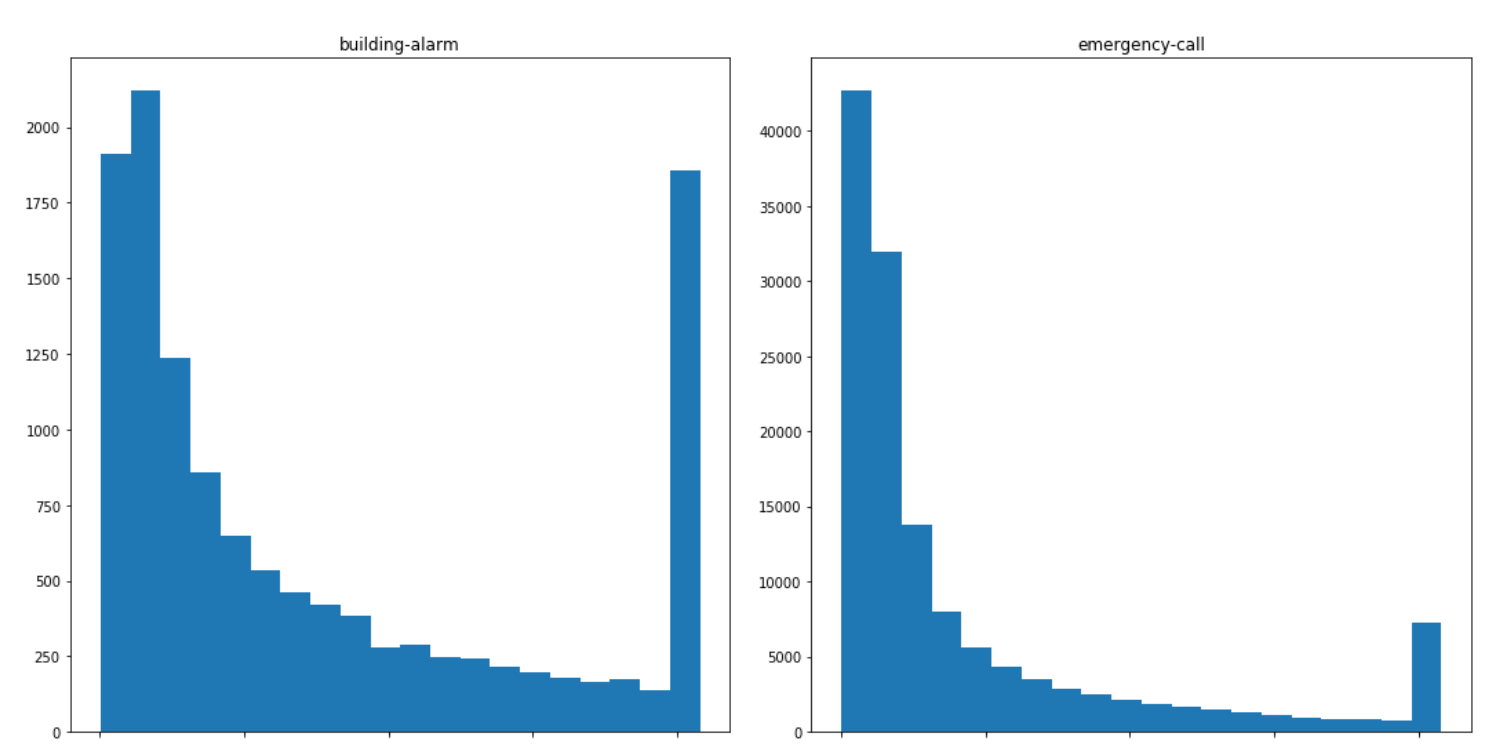
* **Include 1 or 2 key visualizations**

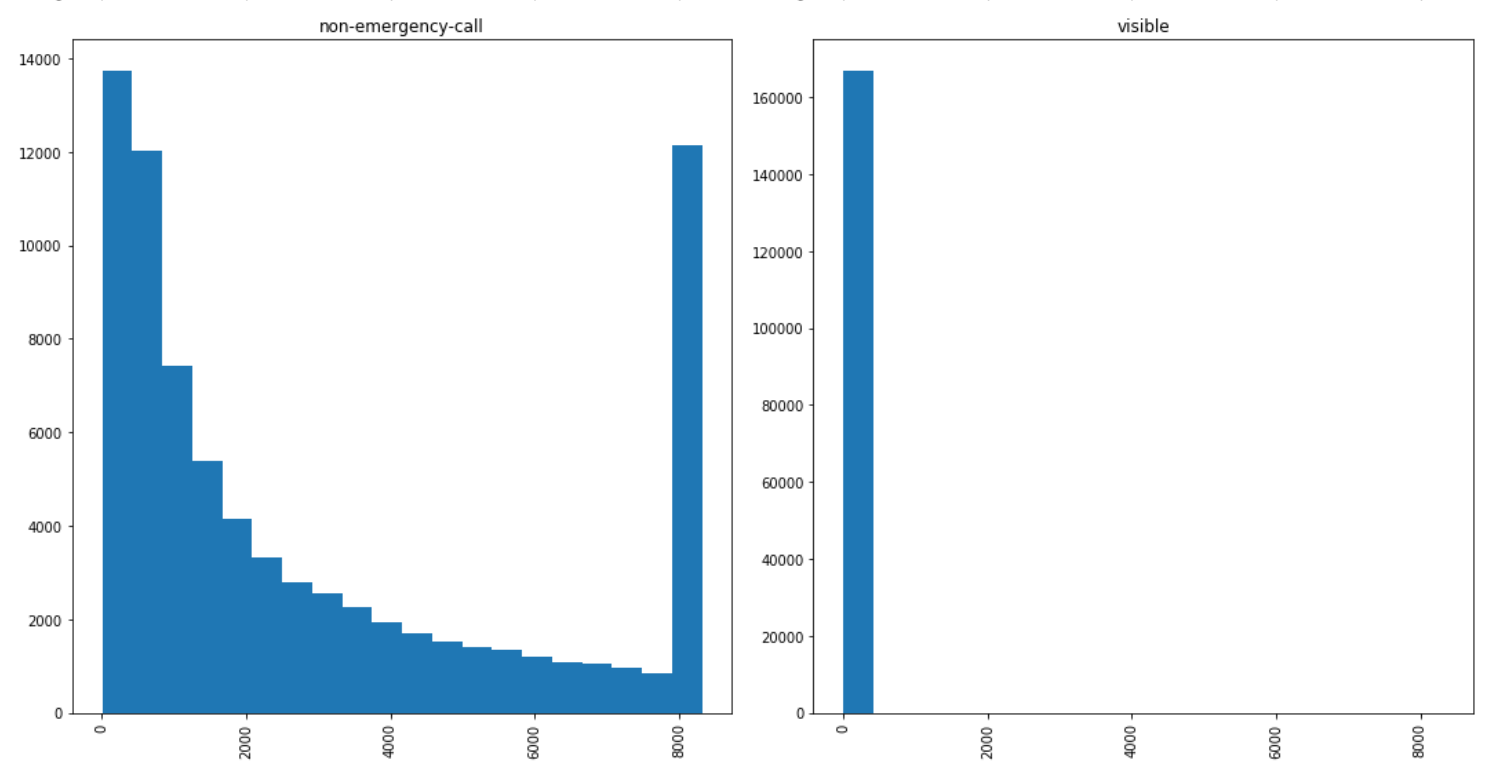
Linear correlations among the cleaned, scaled dataset numerics:



Histograms of response time for all calls and then grouped by call type:







* **Summarize the data exploration and preparation**
  + Data Preparation:
    - Load call\_data\_2017.csv
      * Replace ‘nan’ and ‘unknown’ with np.nan
      * Format two time stamp columns as Pandas DateTime objects
      * Create a calculated column of response time in seconds
      * Create columns for the hour, month, and day of week of calls
      * Filter out any remaining non-2017 rows
      * Drop 2 rows where arrived time preceded original call time
      * Windsorize call time on the right-hand side to consolidate very long response time outliers into the distribution
      * There are no outliers on the left hand side, as 0.0s is a valid and indeed common response time
    - Load neighborhood\_to\_beats.csv data:
      * Explore invalid neighborhood entries in call data
      * Some are clearly tests, miscoded or obviously erroneous
      * Many are missing Sector information, suggesting invalid location data
      * Others have unclear problems
      * The total count is a fairly small percentage of rows, so allow these columns to be effectively dropped during the inner-join step
    - Create key field “beats” common to both call log and neighborhood data sets
    - Perform inner join on “beats” key field
    - The inner join automatically drops the rows with missing or bad neighborhood data. There are no missing data at this stage.
    - Min-max scale numeric columns to 0..1
  + Data Exploration:
    - Examine distributions via histograms and quantile statistics
    - Explore mean, median, other quantiles
    - Use .info() and spot-check printing to examine instances of missing data
    - Calculate the frequency of word-tokens in call-logs and plot the most frequent 10% of word-tokens (figure above)
    - Plot correlation matrix with pearson-correlation values for all numeric variables
      * These results were not strong, which lead to more feature engineering
    - Histograms of response time, for all calls, and then grouped by call-type (figure above)
    - Plot correlation matrix/values again relative to “fast response” binary target (figure above)
    - Use forward selection to fit an ordinary least squares linear model, which provided coefficients, p-values, and confidence intervals (on coefficients)
      * Very useful for understanding key relationships
    - Calculate mutual information to examine non-linear relationships
      * Not very useful due to algorithm’s inability to handle many data
    - Training, fitting, testings models with different subsets of features to explore the combined explanatory power with and without the assigned Call Priority

**3). Features: overview on feature selection and transformations**

* **Describe the needed features**

Five features were needed to achieve >90% AUC and Accuracy: 'Call Type\_visible','Scaled\_Priority','Call Type\_building-alarm','Call Type\_non-emergency-call','Call Type\_emergency-call'. The selection of these features indicates the great importance of the call type (especially on-view, or when an officer is already present) and the priority assigned by the 911-operator or officer him/herself

However one should note that the assigned priority encapsulates quite a lot of the other information. If we do not permit this variable (considering the hypothetical situation where the other context is not going through a human filter), the selected variables are:

'Call Type\_visible','Call Type\_emergency-call','Call Type\_non-emergency-call', 'Precinct\_NORTH','Neighborhood\_Central Business District','Call\_Hour','token\_prob','Neighborhood\_SODO'

Wherein the call type is still very important, but also location takes on greater importance, as does the content of the calls (represented by token\_prob, the result of my NLP feature engineering).

* **Perform the feature engineering**

Following data cleaning (which was described in detail in a prior capstone) the feature engineering and selection process proceeded as follows:

1. Feature Engineering:
   1. Perform natural language processing & feature engineering on the call log column
      1. Tokenize Call Content
         1. Lower case each word
         2. Split logs into tokens
         3. Remove punctuation around and within words
         4. Re-tokenize
         5. Remove English language stop-words
         6. Store the “stem” or root of each token
      2. Calculate numeric “risk score” based on call log tokens
         1. Convert string of tokens into list of strings of tokens for each row
         2. Calculate frequency distribution of all 440 unique tokens
         3. Calculate the probability of occurrence of each unique token
         4. Set probability = 0 for the 3 most common words because on/off are meaningless, and onview is duplicative with the call-type “visible”
         5. Create a new column of the calculated sum of probabilities of each token, the “risk score”
         6. Normalize the new score on min-max 0..1 scale
   2. All Other Features
      1. Drop columns whose information is captured by calculated fields
      2. Encode call-type to more human-readable categories in order to generate exploratory figures of response-time
         1. Plot distributions of response time overall and as a function of call-type
         2. Select as sensible threshold between fast and not-fast calls
      3. One-hot encode all categorical variables except neighborhood (due to size, may need to do later for certain machine learning models should the need arise)
      4. Create a binary response variable, “Fast\_Response” where response time < 5min (300 seconds) is set to 1 and slower response times are set to 0
2. Feature Selection:
   1. Calculate and visualize linear correlation matrix
   2. Calculate mutual information (non-linear relationships) among Fast\_Response and the features
   3. Run forward-random-selection algorithm to select features

Consider all the above to construct a list of selected features. Construct a second set excluding Priority due to its co-linearities with response time.

**4). Model Building: overview of algorithms and tools**

* Describe at least 2 models selected and provide justification for their selection
  + Decision Tree
    - Appropriate for binary classification problems (such as our “Fast Response”)
    - Suited to labeled data (supervised learning)
    - Ability to visualize and interpret results
    - Able to handle categorical data (such as our “Precincts”)
    - Runs quickly, so easy to test different key hyper-parameters without doing an exhaustive grid-search
      * Particularly max-depth
      * Key to preventing over-fitting
    - Performs well with fairly little tuning required
  + Random Forest
    - An ensemble version of decision tree, this is a natural model framework to test alongside decision tree
    - Similar strengths as Decision Tree:
      * Binary classifier problems
      * Labeled data/supervised learning
      * Handles some categorical data
    - Employs “bagging” (individual trees use random subset of features) which functions as a built-in feature selection
      * The trade-off: less interpretable than decision trees
    - Often perform better than decision trees
    - Less risk of over-fitting vs decision trees
  + Gradient-Boosted Random Forest
    - Gradient boosting improves on the random forest
    - Sequential iterations fit to residuals (errors) of prior iteration
    - Improves on error with each iteration
    - Trade-off: High accuracy but prone to overfitting
  + Ada-Boosted Random Forest
    - Ada-Boosting is an alternate approach to improving on the random forest
    - Sequential iterations selectively sample from incorrectly predicted observations
    - Trains on a random subset of the original data (not the residuals)
    - Less prone to overfitting vs gradient-boosting
* **Split, Train, Run Models**
  + 20% of data were set aside as test data
  + 80% of data were used as training data
  + Data were then split into features and the binary target
  + Models were all from sklearn so similar interfaces were available to train, fit, and test the models

**5). Model Evaluation: highlight key achievements and main metrics**

* Report on Accuracy and other key metrics (MSE, R^2, ROC/AUC, Precision, Recall?) for each model

To keep this readable, I will display the Accuracy and ROC/AUC metrics in a pair of tables. The columns correspond to different sets of features:

* All features: the model was allowed to use all available features
* Selected: the model was restricted to a subset of 6 features chosen during feature engineering
* Selected/NoPriority: similar to Selected, but the feature selection process did not permit “Priority” to be among the features, on the basis that it was a “composite” feature arising from the dispatcher or officer’s assessment of all the other features. 7 features total.

The models had the following properties:

* Entropy was found to be best measure of gain
* Max\_depth = 6 was found to be best with decision tree, and used throughout to aid comparison
* Ntrees=100 used throughout to balance robustness with compute time (the random forest models were all slow compute, and the boosted versions were *very* slow)

|  |  |  |  |
| --- | --- | --- | --- |
| **ACCURACY** | Features |  |  |
| Model | All | Selected | Selected/NoPriority |
| Decision Tree | -- | 90.20% | 90.20% |
| Random Forest | 90.18% | 90.20% | 90.20% |
| Gradient-Boosted RF | 90.35% | 90.35% | 90.35% |
| Ada-Boosted RF | 90.65% | 90.20% | 90.80% |
|  |  |  |  |
| **ROC/AUC** | Features |  |  |
| Model | All | Selected | Selected/NoPriority |
| Decision Tree | -- | 0.905 | 0.905 |
| Random Forest | 0.906 | 0.906 | 0.906 |
| Gradient-Boosted RF | 0.915 | 0.906 | 0.912 |
| Ada-Boosted RF | 0.908 | 0.906 | 0.911 |

* **Summarize model performance comparisons**

The most positive outcome of each of these models was that they were all able to achieve better than 90% accuracy and better than 0.90 ROC/AUC in the task of correctly predicting fast response times. Within a class of model (e.g. Random Forest, Decision Tree), the models were able to achieve similar performance metrics with all features, a selected subset including priority, and a selected subset excluding priority – which demonstrates the efficacy of tree-based algorithms at doing their own feature selection (in the “all features” case) and their ability to do accurate classification even when the subset of features varies (which is the basis for the ensemble/forest approach, and a boon to avoiding overfitting).

As expected, the Gradient Boosted and Ada Boosted Random Forests had the strongest performance metrics. These models also took considerably longer to run, making them more costly. With accuracy as the deciding metric, the Ada Boosted RF using a subset of features excluding priority had the best performance. Using ROC/AUC as the deciding metric, the Gradient Boosted RF with all features had the best performance, with GBRF/selected-nopriority and ABRF/selected-nopriority as close seconds.

In the interest of lowering cost, minimizing overfitting, and considering both metrics, I declare the best model for this problem to be the Ada Boosted Random Forest using a selected set of features excluding priority. This model has the best accuracy, very close to the best AUC, uses only 7 features, and in construction is less prone to overfitting than the GBRF.

**6). Time on Task: summary of time on each phase. (If you were going to bill the client for the work involved, report on the time per task.)**

Approximately 100 hours were spent on the project in its entirety, split into the following tasks:

| **Task** | **Time (hrs)** |
| --- | --- |
| Data Retrieval | 5 |
| Data Preparation & Exploration | 40 |
| Feature Selection & Engineering | 35 |
| Data Modeling & Evaluation | 15 |
| Deriving Insight | 5 |

Team members: Eowyn Baughman, Roshni Ganesan, Krantee Bhangalay

(they will be submitting their own documents as we did not work super closely together)