Grace Tang
Springboard DSC - Capstone 2
March 2021

Capstone 2 Final Report:
Detecting Heavy Drinking with Smartphone Accelerometer Data

Problem Statement:

Alcohol has been found to be the third leading preventable cause of death in the US, preceded only by tobacco (first leading preventable cause) and poor diet and physical inactivity (second leading preventable cause). As smartphones become increasingly commonplace in the US, and heavy drinking remains an ongoing problem, it is desirable to be able to detect and prevent incidences of heavy drinking using an easily accessible metric, such as a mobile phone's accelerometer data.

One particular method used to curb heavy drinking behavior is just-in-time adaptive interventions (JITAIs). This method sends prompts or cues in a strategically timed manner to the user (sent "just-in-time") to alter user behavior and change habits. Can we use people's smartphone devices to detect heavy drinking, and thus increase the effectiveness of JITAIs? The results of this project would have a variety of applications, from better implementation of JITAIs for curbing heavy drinking, to law enforcement applications, to medical interventions in severe drinking circumstances.

By using a dataset developed by Killian et al. (2019), I created a supervised binary classification model that classifies whether a personal is considered legally impaired from drinking alcohol--this is defined as when a person's blood alcohol content (BAC) is 8% or higher. The model is an XGBoost Classifier, and achieved an Accuracy of **90.9%** and True Positive Rate of **93.1%** (exceeding the authors' Accuracy and TPR of 77.5% and 69.8%, respectively).

Data Wrangling:

The raw dataset by Killian et al. (2019) contains smartphone accelerometer data from 13 anonymized Ohio State University college students and corresponding Transdermal Alcohol Concentration (TAC) data using SCRAM ankle bracelets, during a bar crawl. The SCRAM bracelets provided 24/7 transdermal alcohol testing, and functions like a breathalyzer for the ankle. SCRAM bracelets show better TAC accuracy at higher alcohol concentrations, making the dataset unsuitable for determining all instances of drinking, but suitable for detecting heavy drinking. The raw dataset contains 15 CSV files of interest:

- 13 files containing the raw TAC data of all 13 participants, one file per one participant.
    - The file contains 5 columns: TAC Level, IR Voltage, Temperature, Timestamp, and Date. Only the TAC Level and timestamp are of interest.
    - The TAC was measured at 30 minute intervals.
- 1 file containing the phone types of each participant: 11 iPhones and 2 Android phones.
- 1 file containing the accelerometer data of all 13 participants.
    - The file contains 5 columns: a timestamp, a participant ID, and a sample from each axis of the accelerometer.
    - The accelerometer data was collected at 40 Hz sampling rate.
    - The file contains 14 million (14057567) rows , and there are no missing values. The data is a concatenation of the 13 individuals' cell phone accelerometer data. Some individuals have as few as 300k instances while others have more than 2 million.

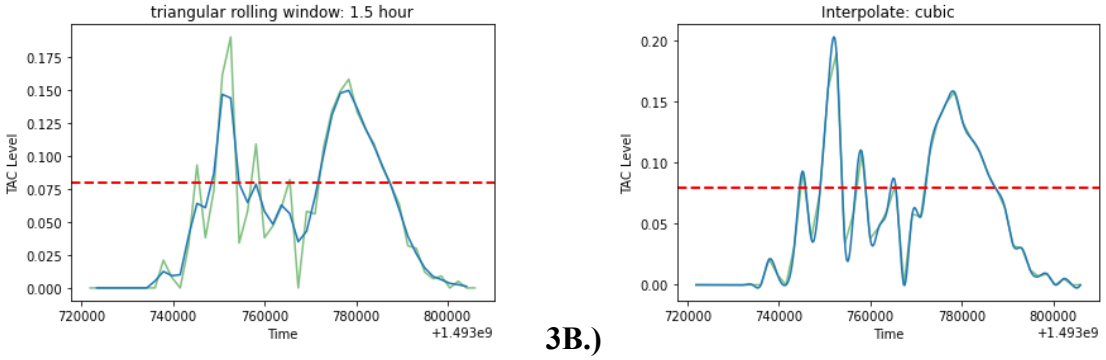
	time	pid	x	y	z
0	0	JB3156	0.0000	0.0000	0.0000
1	0	CC6740	0.0000	0.0000	0.0000
2	1493733882409	SA0297	0.0758	0.0273	-0.0102
3	1493733882455	SA0297	-0.0359	0.0794	0.0037
4	1493733882500	SA0297	-0.2427	-0.0861	-0.0163

Figure 1. First 5 rows of the raw accelerometer data. Time is recorded as a Unix timestamp, with millisecond granularity.

	TAC Level	Time
0	0.0	2017-05-02 10:36:54
1	0.0	2017-05-02 11:09:57
2	0.0	2017-05-02 11:15:27
3	0.0	2017-05-02 11:20:57
4	0.0	2017-05-02 11:26:26

Figure 2. First 5 rows of the raw TAC data for participant BK7610. Time is recorded as a datetime object.

Immediately we see a problem. There is much higher granularity in the accelerometer data (sampled at 40 Hz) than in the TAC data (sampled every 30 minutes). Furthermore, the data is also very noisy, and needs to be smoothed. This resulted in a bifurcation of our data: one dataset which used a triangular-moving-average of window size 3 to smooth all data points (for both the accelerometer data and TAC data), and one dataset which used the scipy interpolation function to fill in the missing TAC data for each row in the accelerometer data.



3A.) **3B.)**

Figure 3. Example of smoothing the TAC data of participant DC6359, using **A.)** Triangular Moving Average with rolling window of size 3, and **B.)** interpolation of the 3rd order. The red-dotted line is the threshold for drunkenness: TAC Level = 0.08

Another problem we discovered was that much of the accelerometer data was missing for certain individuals (Figure 4), leading to an imbalance dataset where some participants had as many as 2 million rows of accelerometer data while others had as few as 300 thousand. I inspected the phone types and found that only the participants with iPhones had swaths of missing data. After contacting a former Apple employee, performing my own online search, and parsing through the Killian et al. (2019) paper, we concluded that it was likely that the program that used to collect the accelerometer data may have been more power-hungry or buggy on the iPhone operating system than on the Android phones. To mitigate this problem of data imbalance, in the train/test split step, I randomly sampled an equal number of instances from each participant so that the final dataset used on our models was balanced.

4A.) Participant SF3079 (iPhone) **4B.) Participant CC6740 (Android)**

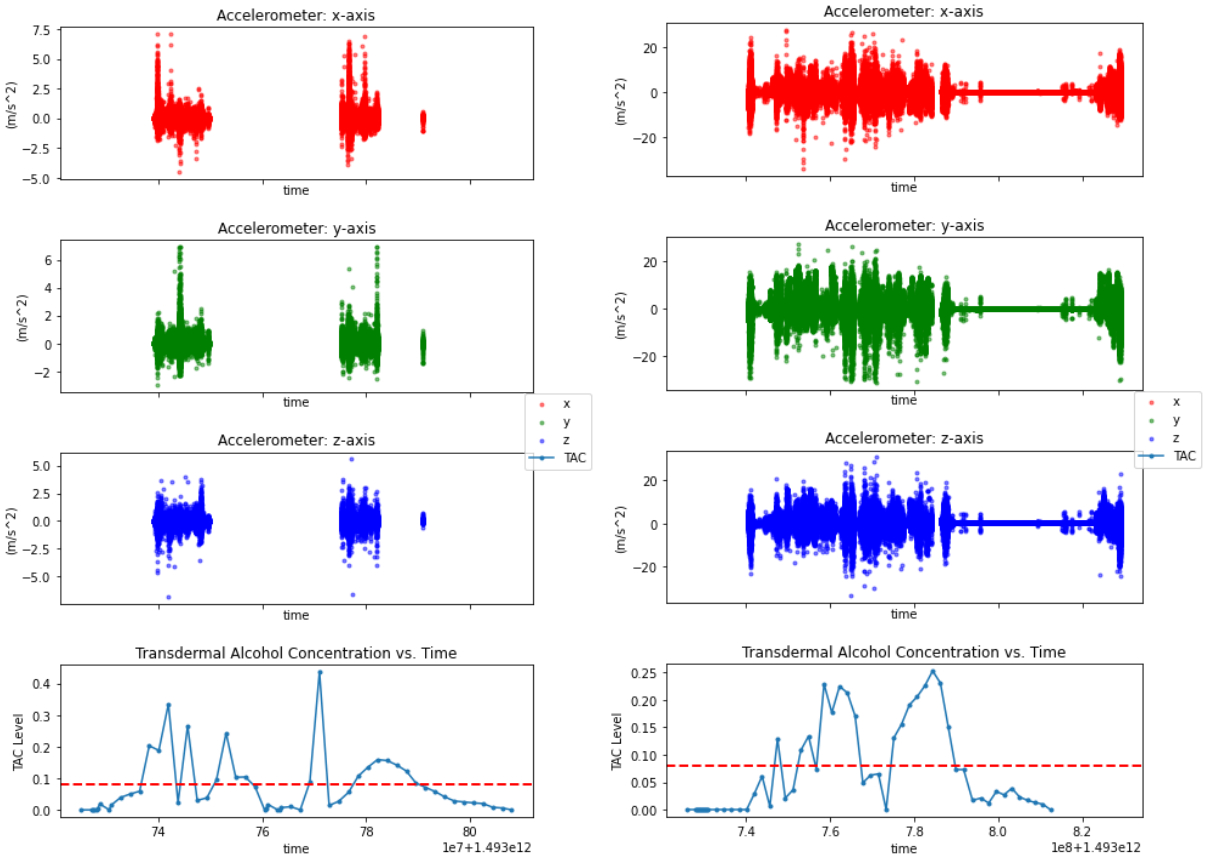


Figure 4. Missing accelerometer data for participant with iPhone (**A**) versus participant with Android (**B**).

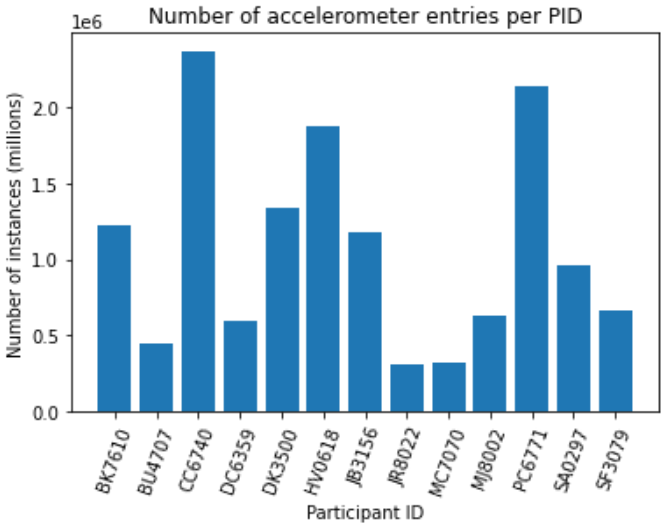


Figure 5. Imbalanced Dataset. Some participants have over 2 million rows of accelerometer data while others have as few as 300 thousand.

Exploratory Data Analysis and Featurization:

Since there is a slight lag between a person's blood alcohol content and detection of transdermal alcohol content, I subtracted 45-minutes from the TAC data to align the accelerometer data with changes in the TAC. I then imputed the missing TAC data with the 2 methods I determined in the Data Wrangling step:

1. Using an interpolation function on the raw data to impute TAC values for every accelerometer row; and
2. Using a triangular-moving-average of window size 3 to smooth all data points (for both the accelerometer data and TAC data), followed by a 3rd order interpolation on the smoothed TAC data to impute TAC values for every accelerometer row.

The dataset had only 3 independent variables (x, y, and z accelerometer data) to predict our 1 dependent variable (TAC Level). Feeding these 3 columns of data into a machine learning model is essentially meaningless. Features have to be derived from the accelerometer data. I made the following features in 6-fold sets (x, y, and z axis, plus their smoothed counterparts x_tma, y_tma, and z_tma) using the pandas rolling() method over 10-second windows:

- Mean
- Standard Deviation
- Variance
- Median
- Max (of raw and of absolute signal)
- Min (of raw and of absolute signal)
- Skew
- Kurtosis
- Zero Crossing Rate - the number of times the signal changes signs.
- Gait stretch - the difference between max and min of one stride (of raw signal).
- Number of steps - the total number of peaks.
- Step Time - the average time between steps.
- RMS - Root-mean-square of accelerations, i.e. the average power.
- Average Resultant Acceleration - Average of the square roots of the sum of the values of each axis squared: $\sqrt{x^2 + y^2 + z^2}$
- Jerk (the gradient of the accelerometer data) and the above features for Jerk.
- Snap (the gradient of jerk) and the above features for Snap.
- The standard deviation of all of the above features.

I would like to note that jerk and snap were not features present in the original paper by Killian et al. (2019), and neither were the standard deviations of all the features. The decision to calculate the standard deviations of all features was made after plotting heatmaps, scatterplots, histograms, and boxplots of the former features. I found that there was no obvious relationship

between any of the features and the participant's TAC. However, there visually appears in the boxplots to be less variation in the features for the participant when they are sober compared to when they are intoxicated (Figure 6).

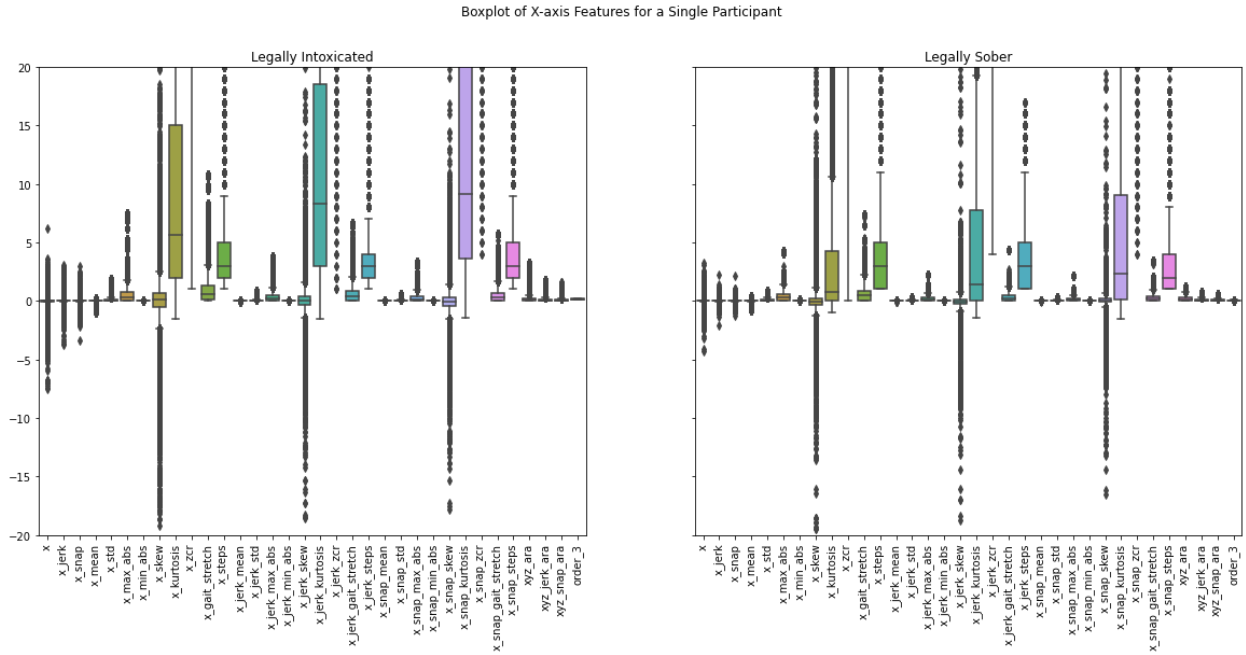


Figure 6. Boxplots of X-axis features for a single participant. TAC > 8% (left) , TAC < 8% (right). Note that the interquartile range appears larger when the participant is intoxicated compared to when they are non-intoxicated.

One possible hypothesis to explain this pattern may be that intoxicated persons may have more erratic or varied movement, resulting in a greater variance in features. I used a 10-second rolling window to calculate the standard deviation of every feature, effectively doubling my features, for a final count of 306 features.

Using Multiprocessing to Parallelize:

As I began featurizing, I quickly realized that adding 306 new features to my dataset was no trivial matter; it was going to take an exorbitant amount of time. I had 13 dataframes (one for each participant) with hundreds-of-thousands to millions of rows in each dataframe. In total, I had approximately 23 million rows to featurize (about 11.4 million for the raw/interpolated dataset, and 11.4 million for the TMA-smoothed dataset).

On my system (Dell Laptop with 12 processors and 16 GB RAM), it took me approximately 127 seconds to featurize 10k rows, and 3413.8 seconds for 447423 rows. Extrapolating from these two data points, I deduced that it would have taken me roughly 87k seconds to featurize all 13 df's of a given dataset (11.4 millions rows), or on the order of 24.1 hours.

Since I had both the raw x, y, z axes as well as their TMA-smoothed counterparts (x_tma, y_tma, z_tma) the total time would have been closer to 48 hours. Therefore, being able to parallelize my processes would be a huge gamechanger!

Python comes with the multiprocessing package, so no additional installation was needed. But the challenge came when implementing it. Because I was using a windows computer (as opposed to a Mac or Linux), multiprocessing would not run unless it called a function from a separate py file. This is something unique to Windows operating system, as Mac and Linux could run a function in multiprocessing directly from the file itself. As a result, I wrote a file called defs.py that contained all of my functions, imported it back into my jupyter notebook, and called it with multiprocessing. The resulting code is formatted as follows:

```
import defs
if __name__ == '__main__':
    num_processors = 10
    p=Pool(processes = num_processors)
    output = p.map(defs.featurize_df,[i for i in df_list])
    for i in range(len(df_list)):
        output[i].to_csv('pid'+str(i+1)+'_featurized.csv')
        print('pid'+str(i+1)+'_featurized.csv')
```

This allows us to put all of our dataframes into a list, map that list to our featurization function, and export each featurized dataframe into a CSV file.

The total time it took to featurize everything was 48732 seconds, or 13.5 hours. Compared to our previous estimate of 48 hours, this was a 3.5x increase in speed! While this number might not seem like much, it importantly allowed me to reduce my processing time from 2 straight days to a simple overnight run.

Preprocessing and Training Data Development:

After featurization, my two datasets totaled to 53.0 GB. Since I did not need such high granularity in the data, I kept only a fraction of each df to use in my train/test split. I used the multiprocessing package to load each dataframe, select every 100th row, dropped the rest, and wrote the selected data into a smaller CSV file.

Earlier when I wrangled the data, I mentioned the issue of imbalance data. This was a two-fold problem: there were far more instances of TAC < 8% than TAC ≥ 8%, and each participant contributed a different percentage to the total number of rows (for example, the participant with the most data had 6 times more rows than the participant with the least data). There was unequal representation from each person and from each binary class.

I created a dictionary of values and plotted the number of TAC instances for each participant in each dataset (Figure 7). We can see that some participants had more sober instances than drunk instances, as previously stated, while others have the opposite. The smallest number we see is 418 rows of TAC < 8% for participant 9. I used n=400 number as the number of rows I randomly selected from each participant/class combination. This resulted in 2 dataframes with 10400 rows and 310 columns, where all rows were evenly distributed between the 13 participants and 2 classes.

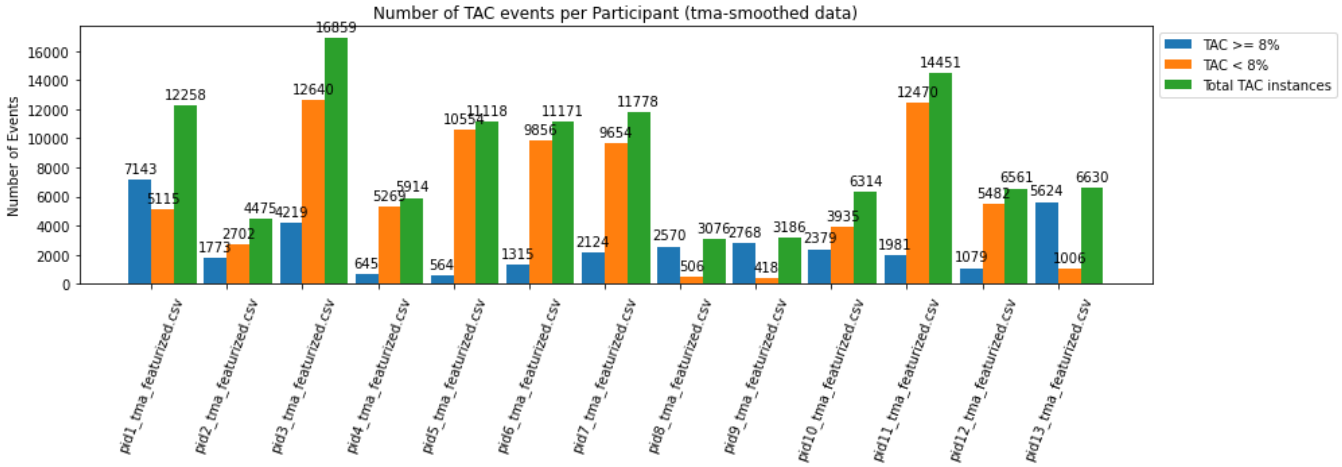


Figure 7. The number of TAC events per participant. Blue = number of rows where TAC ≥ 8%. Orange = number of rows where TAC < 8%. Green = total number of rows. Note that no single person has an approximate 50/50 split between the binary classes.

I then dropped any columns containing 100 or more NaN values, and then dropped any rows containing NaN or np.inf values. This left me with 2 dataframes completely clean from any NaN or infinite values. My raw/interpolated dataset contained 9107 rows and 293 columns, and my TMA-smoothed dataset contained 9486 rows and 287 columns.

I conducted a 70/30 train/test split, and used sklearn's StandardScaler to scale the independent variables.

Model Evaluation:

For each data set (raw/interpolated, tma-smoothed), I tested 4 different Decision Tree Classifiers:

- Random Forest Classifier
- AdaBoost Classifier
- Gradient Boosting Classifier
- XGBoost Classifier

I evaluated each classifier by measuring their accuracy, precision, recall, F1-score, ROC Area Under Curve, Precision-Recall Area Under Curve, and runtime (the time it takes to train the model and make a prediction). I found that RF and XGB using TMA-smoothed data performed best in all evaluation metrics compared to AdaBoost and GradientBoosting. These models also outperformed the RF classifier in the Killian et al. (2019) paper, the original authors of the dataset. The authors achieved an accuracy of 77.5%, and both our out-of-the-box RF and XGB models surpassed that when run on the TMA-smoothed data. This means that the smoothing and featurization conducted on the raw data provided useful new features that improved the predictive power of even the simple RF model.

I then conducted hyperparameter tuning with 5-fold cross validation on the RF and XGB models. The tuned RF model ("RF-optimized") performed worse than prior to tuning in almost every metric, except for runtime, where it was the fastest of all models. The tuned XGB model ("XGB-optimized") performed best in all metrics, except for runtime, where it was 9 times slower than out-of-the-box XGB. XGB-optimized had the following best parameters: {'n_estimators': 1000, 'max_depth': 15, 'learning_rate': 0.1} In summary, while XGB performed better than RF in all measured evaluation metrics, RF had the advantage of being faster. These results can be seen in Figures 8, 9, and 10.

Table of Results

Model	Data	Accuracy	F1 -Score	Precision	Recall	P-R AUC	ROC AUC	Runtime
RF (paper)	-	0.775	-	0.666	0.698	-	-	-
RF	raw	0.753	0.748	0.718	0.873	0.82	0.69	2.526
RF	tma	0.793	0.791	0.742	0.903	0.88	0.72	**2.692**
AdaBoost	raw	0.680	0.679	0.680	0.741	0.75	0.64	34.953
AdaBoost	tma	0.711	0.710	0.694	0.759	0.79	0.65	35.277
GradientBoost	raw	0.739	0.737	0.719	0.830	0.81	0.69	85.575
GradientBoost	tma	0.762	0.760	0.723	0.853	0.84	0.69	83.256
XGBoost	raw	0.864	0.863	0.836	0.922	0.94	0.81	4.655
XGBoost	tma	**0.894**	**0.893**	**0.875**	**0.920**	**0.96**	0.84	3.641
RF - optimized	tma	0.768	0.765	0.722	0.874	0.85	0.69	**0.528**
XGBoost - optimized	tma	**0.909**	**0.909**	**0.891**	**0.931**	**0.97**	0.86	32.856

Figure 8. Table of results of model evaluation metrics. Note that out-of-the-box RF and XGBoost perform better than the RF model from the paper. Also note that while XGBoost-optimized performed the best, it also took significantly longer to process than RF.

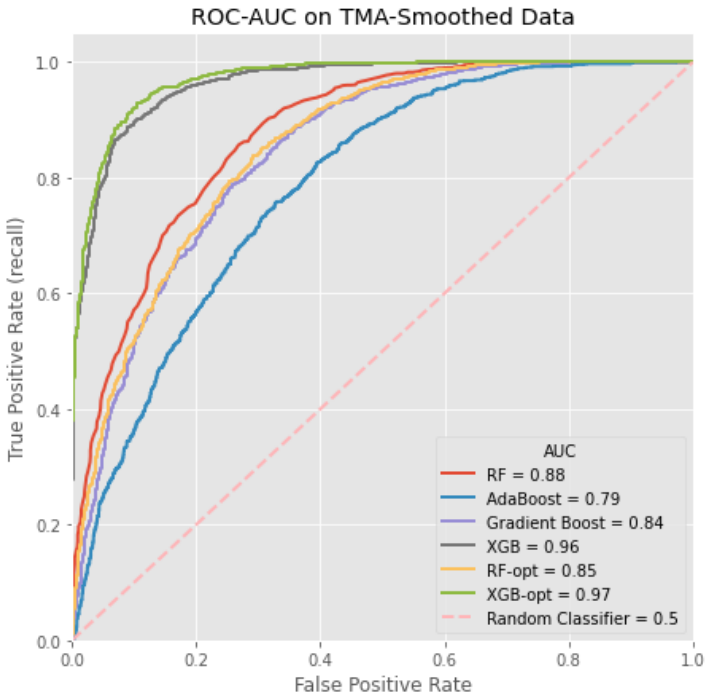


Figure 9. ROC Area Under Curve model comparison. Note that XGB-optimized is only slightly better than the out-of-the-box XGB.

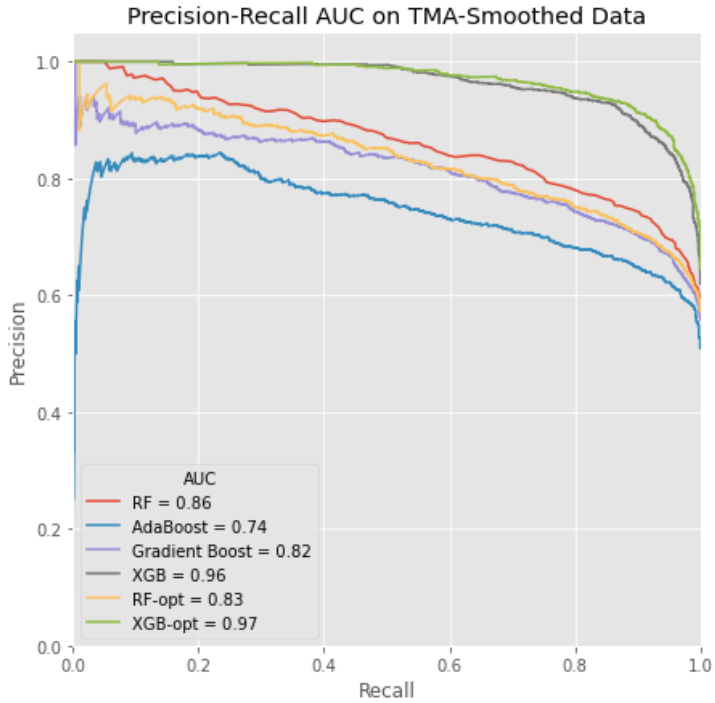


Figure 10. Precision-Recall Area Under Curve model comparison. Note that XGB and XGB-optimized outperform all of the other models.

One more thing I wanted to look at was feature importance. Decision Trees are generally interpretable, but Random Forests and Boosted trees can become more like a black box, where it is difficult to understand what is driving the model predictions. Fortunately we can take a look at what features play a large role in prediction, and which do not. The ones which contribute little to predictive power can be removed from the featurization process in the future, which could further save on resources.

I plotted the top 20 most important features for our highest performing model (XGBoost-optimized), second-best performing model (XGBoost), and our fastest performing model (RF-optimized). All models were trained on the TMA-smoothed data. Figure 11 shows the plot for the relative feature importance of the top 20 features of XGB-optimized.

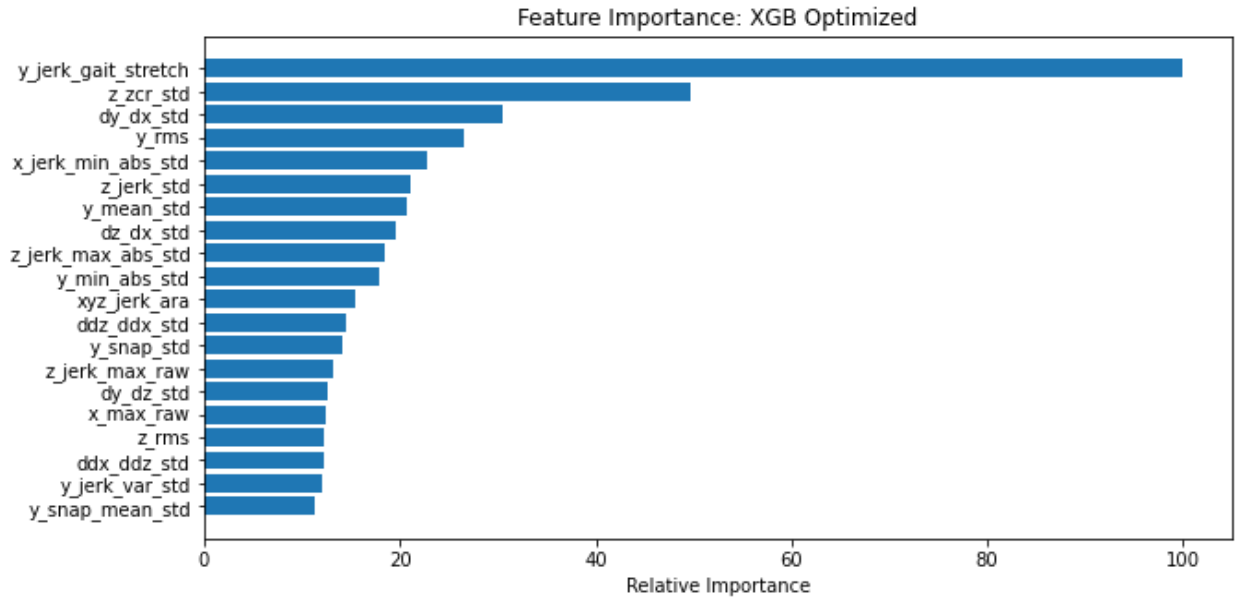


Figure 11. Relative feature importance of the top 20 most important features for XGB-optimized. Note that 14 of the 20 features are standard deviations of another feature.

For the top 20 features of all models, a significant portion of the top 20 features pertain to the standard deviation of another feature (14 of 20 for XGB-optimized, 16 of 20 for XGB, and 9 of 20 for RF-optimized). I similarly looked into the bottom 10 least important features of these 3 models, nad none of them included the standard deviation of another feature. As suspected in the exploratory data analysis and data preprocessing steps, there appeared to be more spread/variation in the features when the participant was intoxicated compared to when they were not intoxicated. Computing the standard deviation of every feature has indeed provided more predictive power to the model.

Model Selection:

The most accurate model was XGB-optimized with the following parameters: {'n_estimators': 1000, 'max_depth': 15, 'learning_rate': 0.1}. If resources were unlimited, this model could be used to develop an app that would predict when the user is no longer sober and can prompt them with a JITAI to curb heavy drinking behavior. If, however, we want to develop something lighter, I recommend instead the out-of-the-box XGB (where 'n_estimators':100), which also performs well. And again, if we want something even lighter than that, then I would recommend Random Forest Classifiers as an option. In summary:

- XGBoost-optimized: best performance, but heavy.
- XGBoost: second-best performance, and 9x faster than the best.
- Random Forest: third-best performance, and much faster. Needs additional tuning.

Future Directions:

If this model were to be implemented into a just-in-time adaptive intervention app, there is still room for improvement. Had there been more time, I would have liked to do more hyperparameter tuning on the RF classifiers to improve its predictive power while keeping it light. I also would have liked to do the same with XGB Classifier, tuning it while keeping the n_estimators capped at 100 to avoid making the model heavier.

Alternative models that were not discussed here could also be looked into. Instead of randomly selecting data from each participant when preparing the training data, I could opt to preserve the datetime index and use it in combination with the features to predict blood alcohol content ahead of time, using forecasting methods like ARIMA or Facebook Prophet.

Machine learning is ever evolving--people are always developing new and better models. While XGBoost is an incredibly powerful classifier, it is important to remember to never become complacent, and keep up a curiosity for where the field of data science can go next.

Credit:

I would like to thank Chris Esposo for being an awesome Springboard mentor: for giving me guidance, for teaching me about multiprocessing when I got stuck on featurization, and for all the encouragement throughout the course of the bootcamp!

I would like to thank Jackson A Killian (jkillian '@' g.harvard.edu, Harvard University); Danielle R Madden (University of Southern California); John Clapp (University of Southern California) for uploading this valuable dataset to the UCI Machine Learning Repository.

Sources:

Killian, J.A., Passino, K.M., Nandi, A., Madden, D.R. and Clapp, J., Learning to Detect Heavy Drinking Episodes Using Smartphone Accelerometer Data. In Proceedings of the 4th International Workshop on Knowledge Discovery in Healthcare Data co-located with the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019) (pp. 35-42). [Web Link]

Dataset: http://archive.ics.uci.edu/ml/datasets/Bar+Crawl%3A+Detecting+Heavy+Drinking