Detecting Heavy Drinking with Smartphone Accelerometers

Grace Tang
Springboard DSC - Capstone 2

The Problem with Alcohol

The Solution: Smartphones and Algorithms

- Can we predict incidences of heavy drinking using smartphone accelerometer data?
- Many applications:
  - Just-In-Time Adaptive Interventions (JITAIs)
  - Law Enforcement
  - Medical Intervention

The Data: Accelerometers and TAC

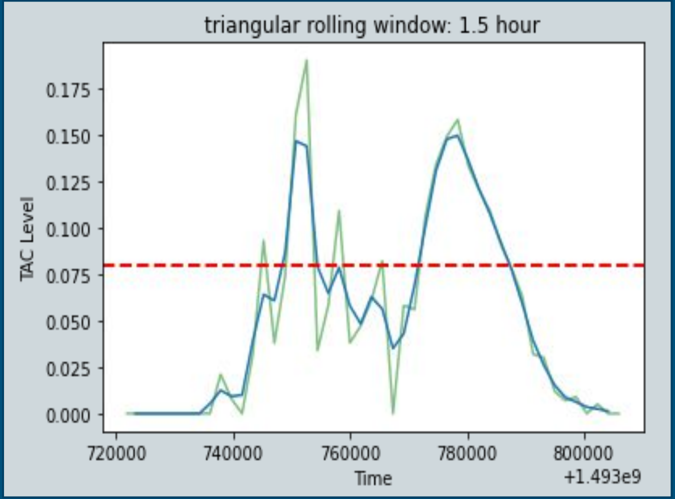
- 13 anonymous persons
  - 11 iPhones
  - 2 Android phones
- Accelerometer data
  - Sampling rate: 40 Hz
  - 14 million rows
- Transdermal Alcohol Content (TAC)
  - Sampling rate: per 30 min
  - 600 rows

	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

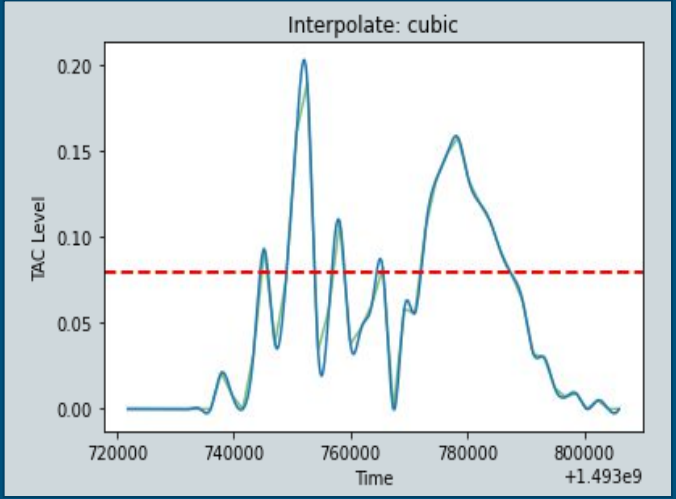
	TAC Level	Time
0	0.0	2017-05-02 10:36:54
1	0.0	2017-05-02 11:09:57
2	0.0	2017-05-02 11:15:27
3	0.0	2017-05-02 11:20:57
4	0.0	2017-05-02 11:26:26

Bifurcating the Dataset: 2 Ways to Impute TAC

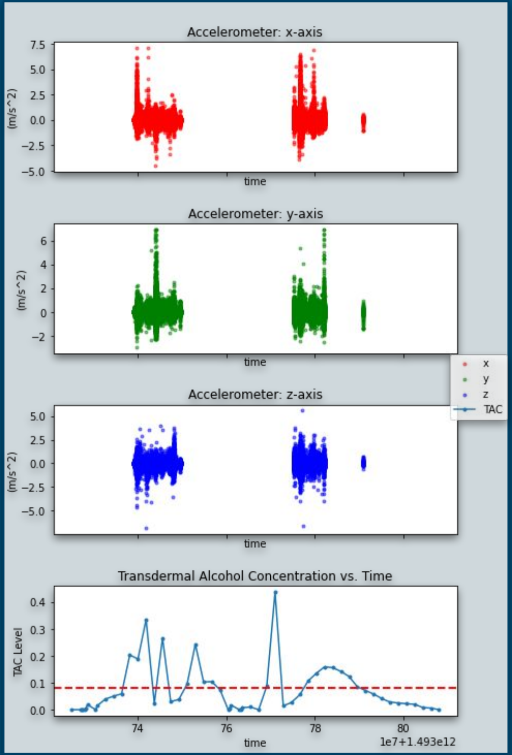
- Smooth with TMA
- Then scipy interpolate

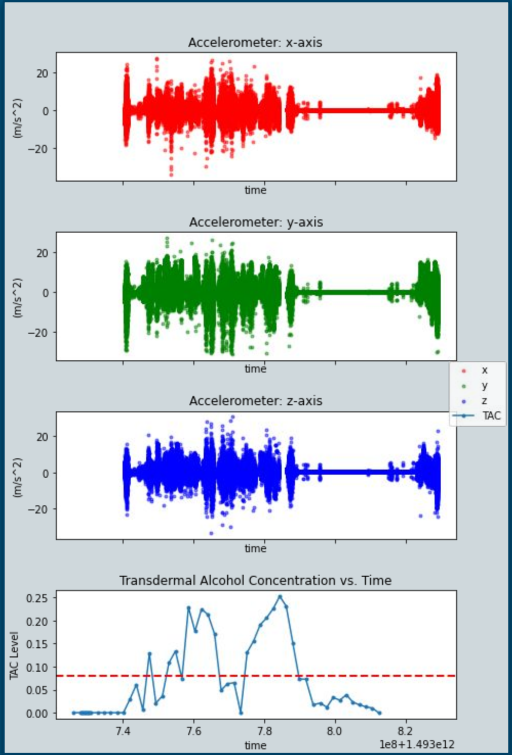


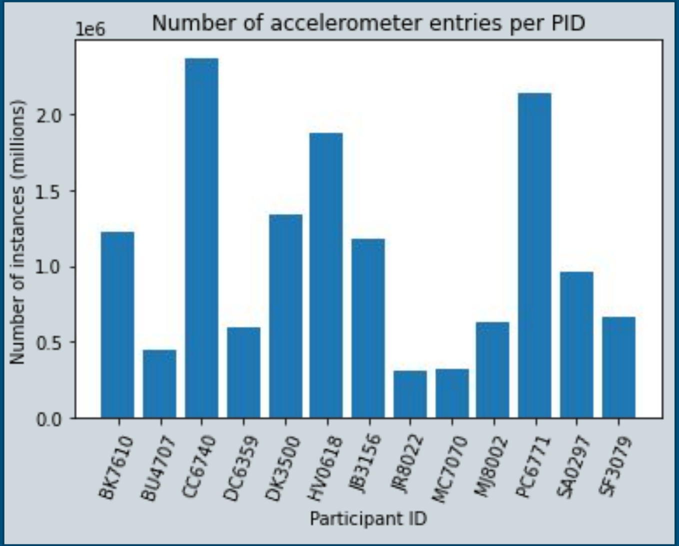
- Preserve the raw datapoints
- Then scipy interpolate



Accelerometers: Noisy Data and Missing Data



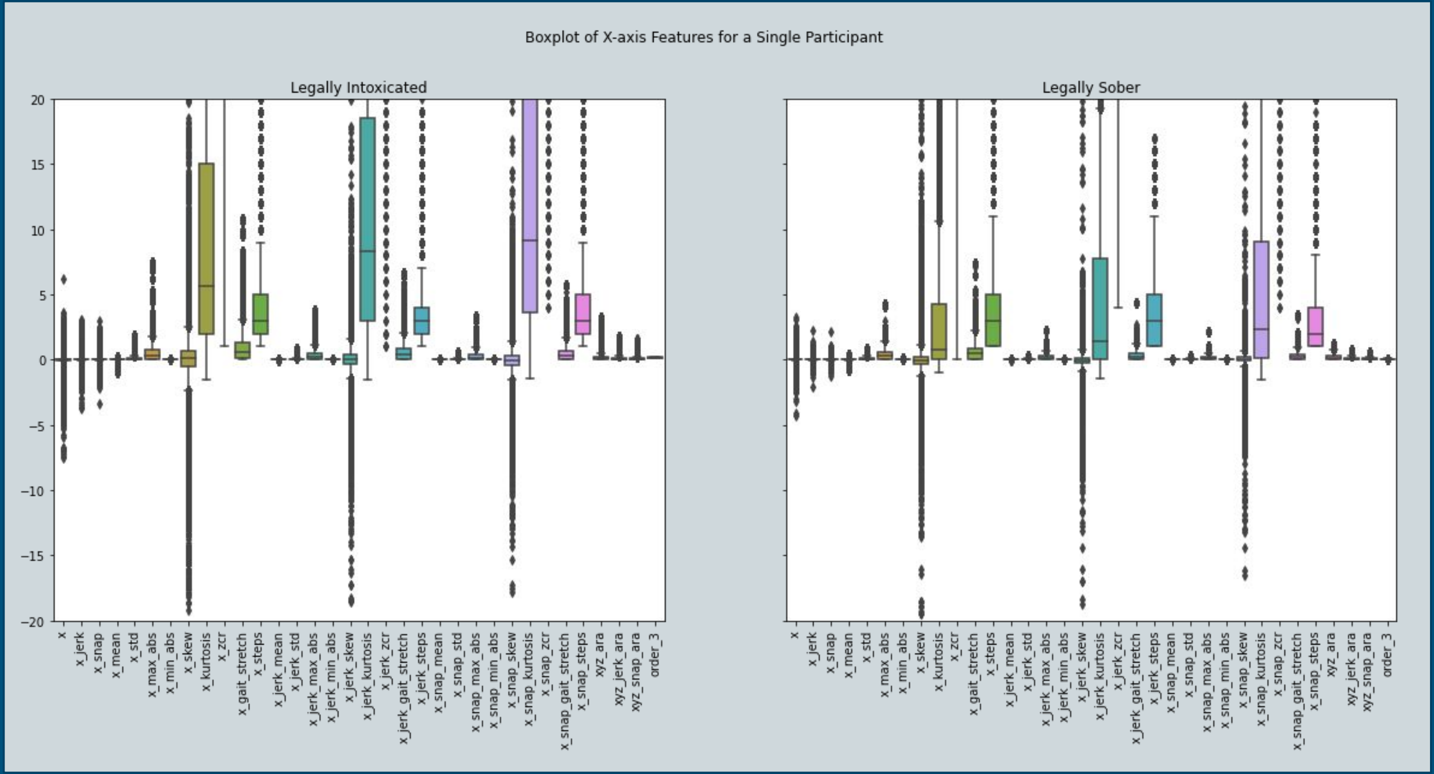




Featurization: Making Meaning out of Noise

- Before featurization: 3 features (x, y, z) & 1 target (TAC)
- Pandas rolling() method, 10-second windows
- After featurization: 306 new features
    - Mean, Standard Deviation, Variance, Median, Max & Min (of raw & absolute signal), Skew, Kurtosis, Zero Crossing Rate, Gait stretch, Number of steps, Step Time, Root Mean Squared, Average Resultant Acceleration, partial derivatives of each axes
    - Jerk (the derivative of acceleration) + above
    - Snap (the derivative of jerk) + above
    - The standard deviation of all of the above features

No Immediate Patterns



Sequential Processing is slow...

- 23 million rows to featurize
  - 13 dataframes (1 per person)
  - 2 datasets (raw vs smoothed)

Dataframe Size	Sequential Processing	Parallelized Processing	Speed Increase
10,000 rows	126.9 seconds	17.2 seconds	7.4x faster
447,423 rows	3414 seconds	--	--
22,756,812 rows	174k seconds (48.2 hours)	48732 seconds (13.5 hours)	3.6x faster

Using Multiprocessing to Parallelize

- Python’s multiprocessing package
- Challenges with Windows OS

- Result: 48732 seconds (13.5 hours)
    - 3.6x faster
    - Reduce from 2-day to overnight!

```
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’)
```

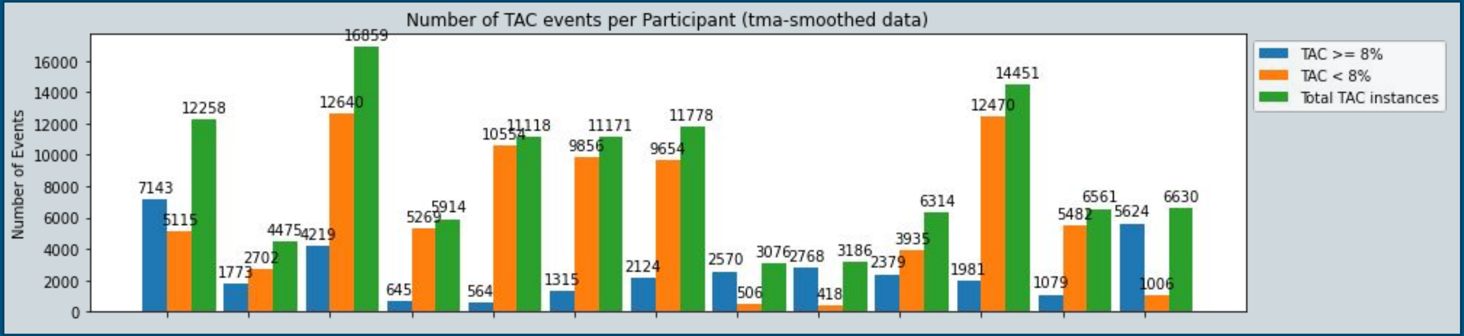
Preprocessing Data

Reduce from 53.0 GB to 42.4 MB

- Keep only every 100th row
- Random select n = 400/combo
- Drop columns with > 100 NaNs
- Drop rows with NaNs

Result

- Raw/interpolate dataframe
  - 9107 rows, 293 columns
- TMA-smoothed dataframe
  - 9486 rows, 287 columns
- 70/30 train/test split



Lots and Lots of Decision Trees!

Binary Classification Models:

- Random Forest
- AdaBoost
- Gradient Boost
- XGBoost

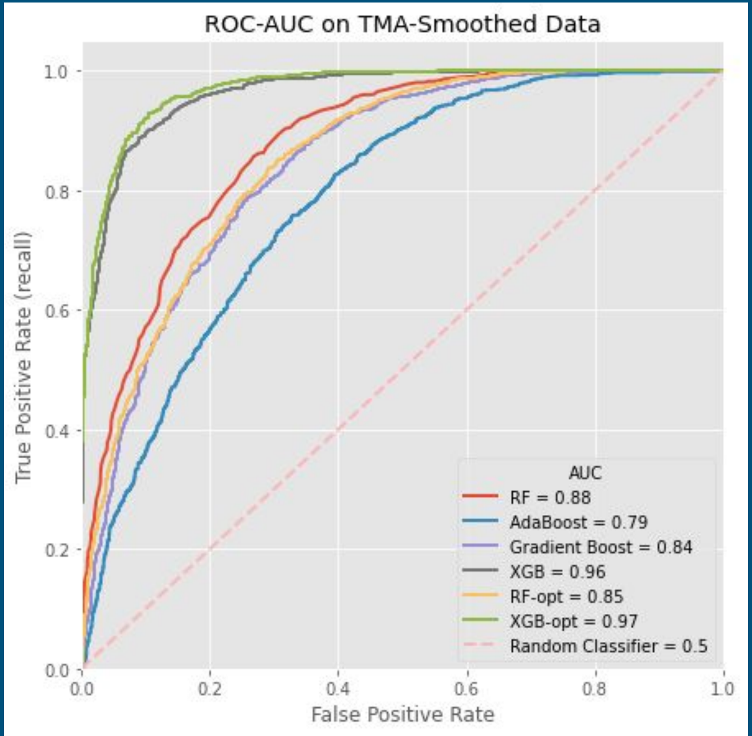
Hyperparameter Tuning: Random Search Cross Validation

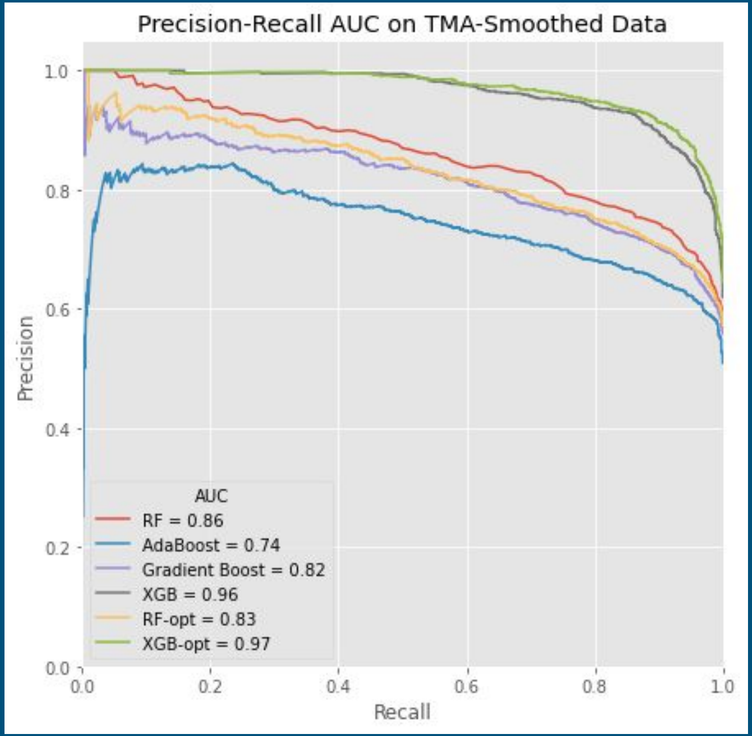
- XGBoost Best Params: {'n_estimators': 1000, 'max_depth': 15, 'learning_rate': 0.1}

Table of Results

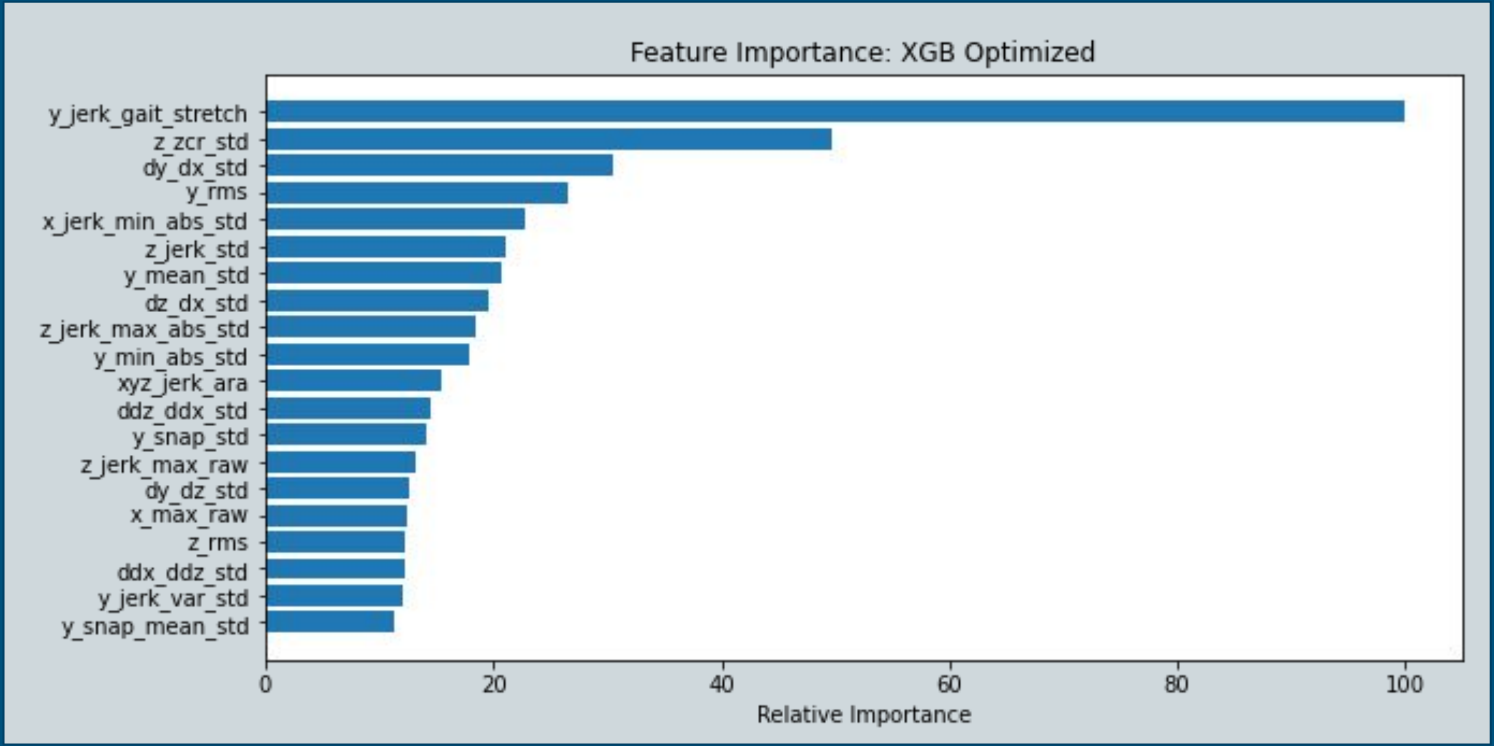
Model	Data	Accuracy	F1 -Score	Precision	Recall	P-R AUC	ROC AUC	Runtime
RF (paper)	-	0.775	-	0.666	0.698	-	-	-
RF	raw	0.753	0.748	0.718	0.873	0.82	0.69	2.526
RF	tma	0.793	0.791	0.742	0.903	0.88	0.72	**2.692**
AdaBoost	raw	0.680	0.679	0.680	0.741	0.75	0.64	34.953
AdaBoost	tma	0.711	0.710	0.694	0.759	0.79	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 - tuned	tma	0.768	0.765	0.722	0.874	0.85	0.69	**0.528**
XGBoost - tuned	tma	**0.909**	**0.909**	**0.891**	**0.931**	**0.97**	0.86	32.856

Model Comparison: ROC and Precision-Recall





Feature Importance:
Standard Deviations of Other Features



Takeaways: XGB is Best & Features are Good

- XGBoost: best model, but heavy.
  - Out-of-the-box XGB: 9x faster than tuned XGB
  - RF: much faster, but needs additional tuning
- Features:
  - Adding Jerk, Snap, and Standard Deviation of other features improves model prediction

Future Directions: Make it Lighter

- Rewrite featurization step - feature reduction
- RF Classifier - additional tuning
- XGB Classifier - additional tuning, cap n_estimators ≤ 100
- Alternative models:
  - ARIMA
  - Facebook Prophet

Thank you

Chris Esposo for his guidance and being an awesome Springboard mentor!

Ryan Langman for giving me feedback on generating features.

Patrick Au for answering my questions about iPhones.

Jackson A Killian (Harvard University), Danielle R Madden (University of Southern California), and John Clapp (University of Southern California) for uploading this valuable dataset to the UCI Machine Learning Repository.

Citations

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). [http://ceur-ws.org/Vol-2429/paper6.pdf]

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

Questions?