Human Activity Recognition & Live City IoT Sensor Network

By: Tom Bresee and Michael Phillips

Introduction

GitHub Code Repositories:

Human Activity Recognition	(supervised machine learning repo)
Sensor Analysis Splash Page	(introduction and background on this complicated effort)
Sensor Analysis	(unsupervised machine learning repo)

Supplementary Items:

Heroku	(unsupervised machine learning add-on for some visualization and plotting)

Supervised Learning

Motivation

In recent years, human activity recognition (HAR) has become a hot topic inside the scientific community, as more and more wearable devices enter the market. We will use realistic-focused data (extracted directly from the Android app and the sensors of each smartphone involved) to predict user association with one of the four possible registered activities whether a user is inactive, active, walking, or driving, and perform association with defined movements, in addition to general information extraction. We both wear fitness trackers, so learning more about HAR, seemed like a great way to test our skills.

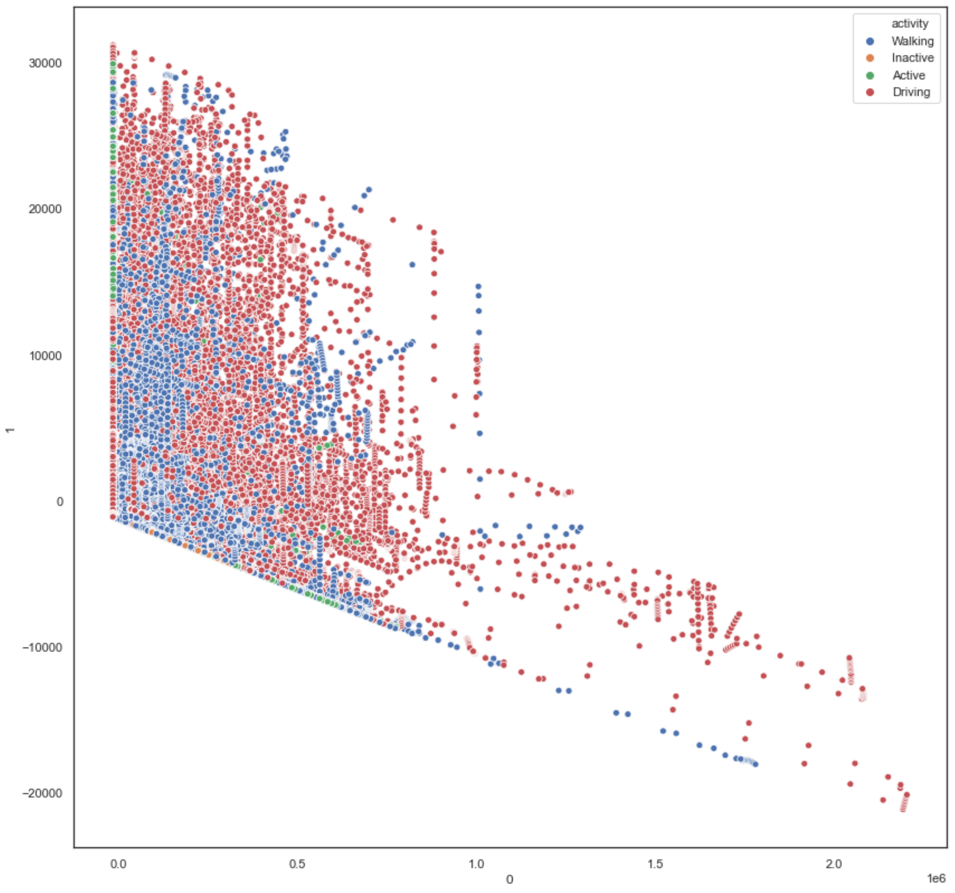
With this dataset, we hoped to identify key features that can help distinguish activities from one another. However, with there being 90+ features, we decided to pursue dimensionality reduction.

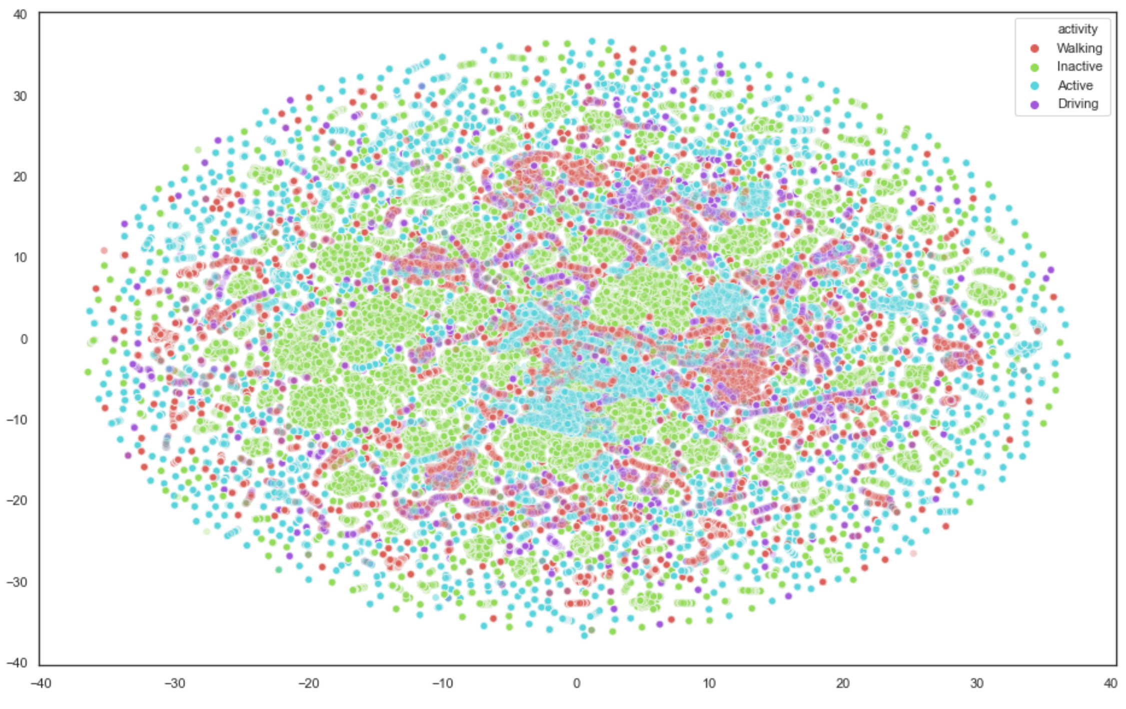
Data Source

We got the data for our supervised learning from https://lbd.udc.es/research/real-life-HAR-dataset/data_cleaned_adapted_features_full.zip . The data came in a csv format and included accelerometer, gyroscope, magnetometer, and gps readings. The data also included what activity the user was doing at the time of the reading. The activities were broken up into 4 categories: Active, Inactive, Driving, and Walking. In total, there were 499,276 records from 11 users which covered October 8, 2019 to December 20, 2019.

Methods & Evaluation

To start this analysis, we loaded the data into a pandas dataframe. From there, we dropped the columns: ‘Activity_ID’, ‘User’, ‘ID’, and ‘Timestamp’ from the dataframe. We took the remaining ‘Activity’ column and turned that into its own Series for labels. The next step we took was to perform dimensionality reduction to get the number of features from 91 down to 2. To do this, we did PCA and T-SNE (images below in order of reference) to see which would perform better in the models. Once that was done, we created two sets of train and test data, one for the PCA dataset and the other for the TSNE data set:

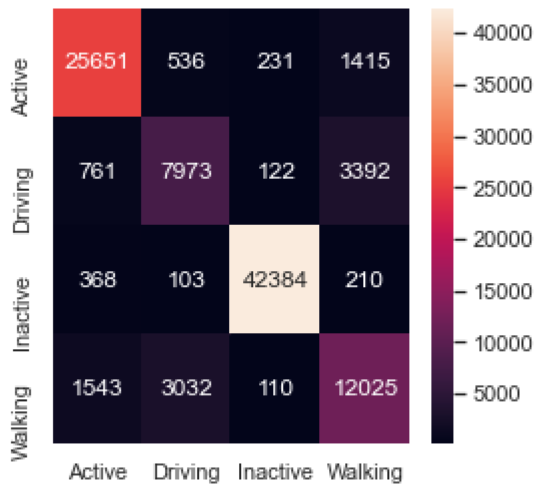

activity
Walking
Inactive
Active
Driving
30000
20000
10000
0
−10000
−20000
1
0.0
0.5
1.0
1.5
2.0
0
1e6


activity
Walking
Inactive
Active
Driving
40
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40

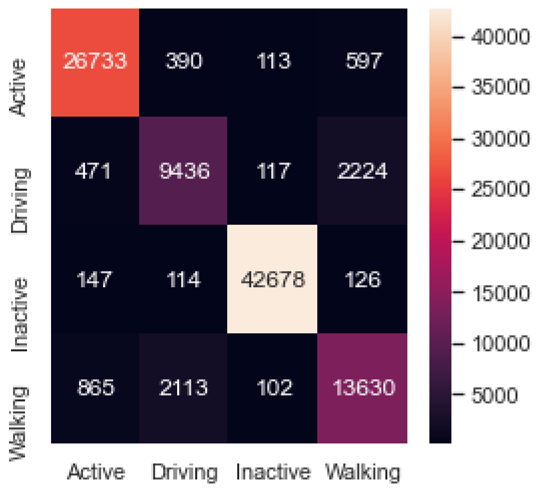
Random Forest

The first model that we decided to use was Random Forest Classifier. For all of the applicable models, we used a random_seed of 42 and default parameters. The Random Forest Classifier performed very well on both sets of data. It had an accuracy score of 88.2% for the PCA dataset and 92.6% for TSNE. We also took the confusion matrices of the data to see where the most misclassification was.

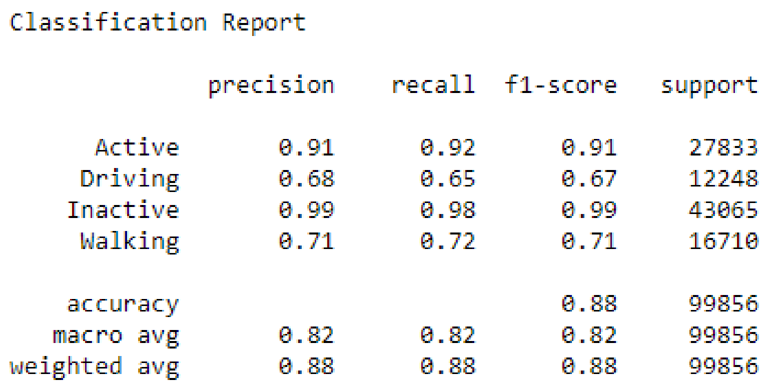
PCA



TSNE

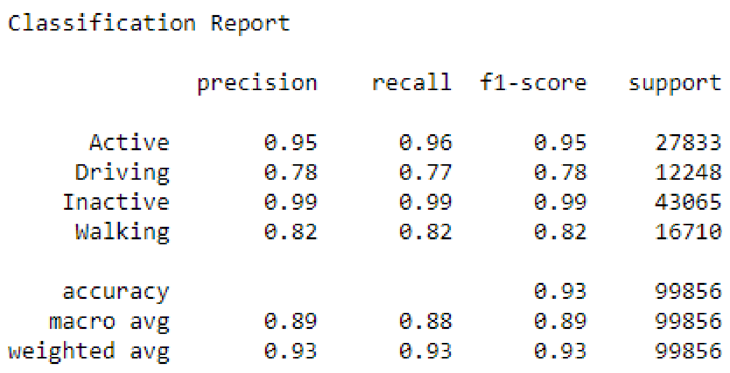


We see that most of the misclassifications were between driving and walking. We determined that this is likely due to slow driving having similar readings as walking. More details, such as precision, recall, and f1-score were generated using a classification report:



```
Classification Report
               precision    recall  f1-score   support

      Active       0.91      0.92      0.91     27833
     Driving       0.68      0.65      0.67     12248
    Inactive       0.99      0.98      0.99     43065
     Walking       0.71      0.72      0.71     16710

    accuracy                           0.88     99856
   macro avg       0.82      0.82      0.82     99856
weighted avg       0.88      0.88      0.88     99856
```

PCA



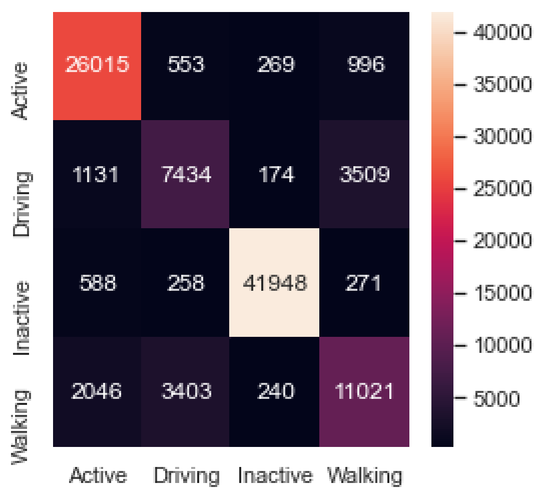
```
Classification Report
               precision    recall  f1-score   support

      Active       0.95      0.96      0.95     27833
     Driving       0.78      0.77      0.78     12248
    Inactive       0.99      0.99      0.99     43065
     Walking       0.82      0.82      0.82     16710

    accuracy                           0.93     99856
   macro avg       0.89      0.88      0.89     99856
weighted avg       0.93      0.93      0.93     99856
```

TSNE

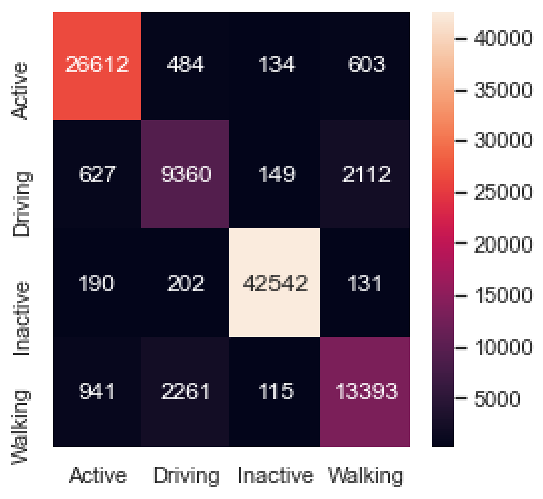
K-Nearest Neighbors

The next model that we tried was K-Nearest Neighbors with default parameters. This model also performed well, generating accuracy scores of 86.5% and 92.0% for PCA and TSNE, respectively. We also created more confusion matrices for this model.

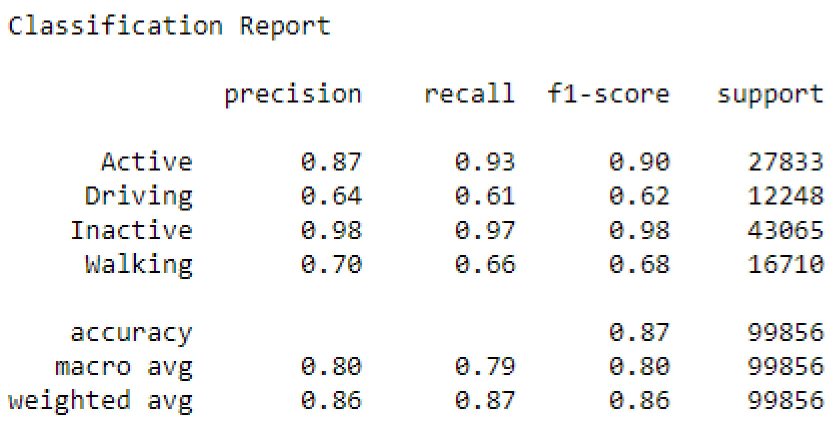
PCA



TSNE

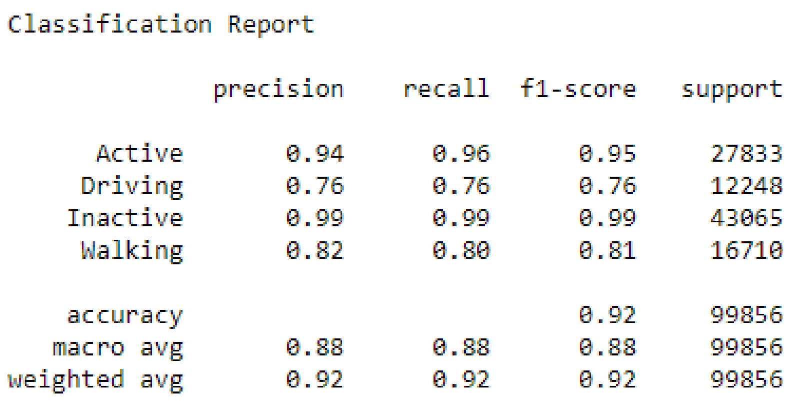


Again, we see that the most overlap is between driving and walking. The classification reports are as follows.



```
Classification Report

              precision    recall  f1-score   support

      Active       0.87      0.93      0.90     27833
     Driving       0.64      0.61      0.62     12248
    Inactive       0.98      0.97      0.98     43065
     Walking       0.70      0.66      0.68     16710

    accuracy                           0.87     99856
   macro avg       0.80      0.79      0.80     99856
weighted avg       0.86      0.87      0.86     99856
```

PCA



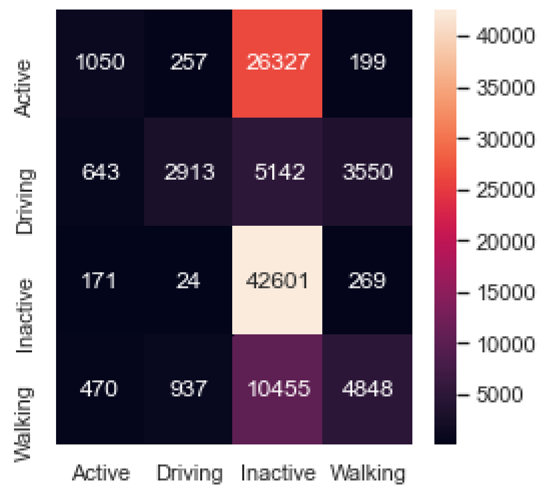
```
Classification Report

              precision    recall  f1-score   support

      Active       0.94      0.96      0.95     27833
     Driving       0.76      0.76      0.76     12248
    Inactive       0.99      0.99      0.99     43065
     Walking       0.82      0.80      0.81     16710

    accuracy                           0.92     99856
   macro avg       0.88      0.88      0.88     99856
weighted avg       0.92      0.92      0.92     99856
```

TSNE

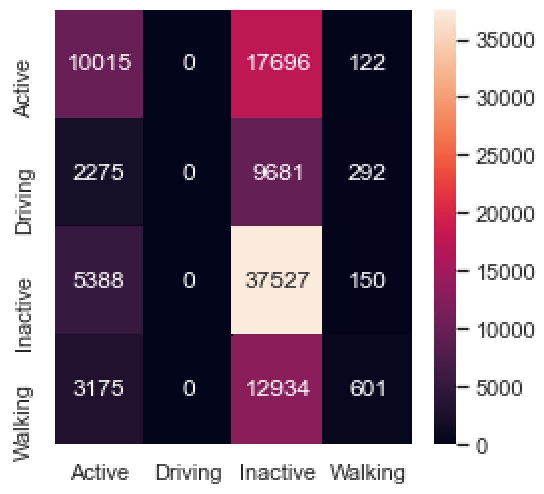
Support Vector Classifier

The final model that we used for Supervised Learning was SVC with default parameters. This did not perform as well as the others with accuracy scores of 51.5% and 48.2% for PCA and TSNE, respectively. The confusion matrices also tell a different story.

PCA



TSNE



The confusion matrices tell us that the model classified nearly everything as inactive. The classification reports show us just how inaccurate the model was across the board.

```
Classification Report

              precision    recall  f1-score   support

      Active       0.45      0.04      0.07     27833
     Driving       0.71      0.24      0.36     12248
    Inactive       0.50      0.99      0.67     43065
     Walking       0.55      0.29      0.38     16710

    accuracy                           0.51     99856
   macro avg       0.55      0.39      0.37     99856
weighted avg       0.52      0.51      0.41     99856
```

PCA

```
Classification Report

              precision    recall  f1-score   support

      Active       0.48      0.36      0.41     27833
     Driving       0.00      0.00      0.00     12248
    Inactive       0.48      0.87      0.62     43065
     Walking       0.52      0.04      0.07     16710

    accuracy                           0.48     99856
   macro avg       0.37      0.32      0.27     99856
weighted avg       0.43      0.48      0.39     99856
```

TSNE

Some important trade offs that we came across was that TSNE took much longer to do dimensionality reduction than PCA. However, the TSNE dataset performed better than the PCA dataset in the Random Forest and KNN models, so the pay off was worth it.

The results were very reactive to what model was run. Random Forest and K-Nearest Neighbors had very similar results, while Support Vector Classifier was drastically more inaccurate compared to those previously mentioned models.

Failure analysis

SVC took ages to run and then the performance was horrid. This likely took so long due to the time complexity difference. SVM's time complexity is $O(n^2p + n^3)$ while Random Forest is $O(n^2pntrees)$. By this logic, SVM should take 2500x longer than Random Forest based on the size of our dataset, which seemed accurate when running these models.

Unsupervised Learning

Motivation

As a continuing investigation into the realm of real-world sensor analytics, we focus on a massive Internet of Things (IoT) cluster known as the Array of Things, hosted in the city of Chicago and part of a continuing effort for smart city enablement. It is expected that smart city initiatives will progress, as Chicago takes the lead in deploying it's AoT cluster. In recent years, the smart city concept has gone from theoretical to actual live production networks. We leverage unsupervised machine learning tactics to dive deeper into a large set of this extensive AoT sensor data. We used Folium (python geo-mapper) to plot all 126 current sensor node locations. The AoT project leverages open source hardware/software developed by Argonne National Laboratory. The supervised machine learning techniques were initially associated with our first dataset (the human activity recognition) HAR dataset. This section details mainly the second part, which was to identify unsupervised machine learning techniques for an **additional** dataset we identified. The IoT sector is expected to be a billion dollar market as advances in computing, lower cost sensors, and 5G cellular technology progress. **We anticipate more and more IoT-related AI research is done into subjects such as sensor parameter key performance indicator (KPI) anomaly detection**, sensor predictive maintenance, sensor malfunction estimation, and sensor status. Our investigation revolved around clustering sensor data KPI values into 'normal' vs 'anomalous', as well as comparing individual cluster node data with other nodes. Specific unsupervised learning techniques involved are Isolation Forest, DBSCAN, k-means, and generalized clustering. Part of the motivation for this project is the fact that this dataset represents a huge real-world non-synthetic view of sensor IoT programs. Many datasets from the UCI repo are synthetic or run under laboratory-like conditions, however this dataset represents insight into how IoT sensors really perform, the good, the bad, and the **practical**. One of our goals is to create code to cluster streams of time-series data into normal, non-normal, and anomalous. We created a list of reference links as well about the complex architecture for clarity, and we streamlined our code to be very viewable, with code/result links kept here. [Splash Page]

Data Source

The main dataset is consolidated into one massive (tar) compressed file and hosted here.

- **filename** - AoT_Chicago.complete.latest.tar
- **uncompressed file size - 320 GB (compressed file size - 33.73 GB)**
- **uncompressed file type** - .csv
- **granularity** - Every sensor parameter records parameter data every **25 seconds**
- **raw rows of data** - Over four billion (4,195,104,977 unique observations)
- **time period of data** - 2018-03-06 22:14:56 through 2021-03-22 05:18:42; representing 1,112 days, i.e. 3 years, 16 days
- **granularity of data** - 25s increments for multiple sensors, multiple sensor substems, and multiple parameters
- **total number of unique nodes** - 126. Many nodes include chemical sensors, Alphasense OPN-N2 air quality sensors, and/or Plantower PMS7003 air quality sensors

- Important variables included: timestamp (UTC timestamp of when the measurement was done), node_id (ID of node which did the measurement), subsystem (Subsystem of node containing sensor), sensor (sensor that was measured), parameter (sensor parameter that was measured), value_raw (raw measurement value from sensor), and value_hrf (the critically important converted "human readable" measurement value from the sensor). We have documented a deep dive into the data and its explanation here. Additions: the data source also included multiple metadata files.

Unsupervised Learning Methods

In this section we will describe the workflow of our source code, the learning methods and feature representations we used, how we tuned parameters, and the challenges we encountered (and how we overcame them).

Our main data file was in compressed tar format. This main file was downloaded, and the archive was untared, which produced a directory. We unpacked this data archive, and the archive decompressed to a huge CSV file about 10 times its original size (ending up being over 320 GB in total size). We leveraged Dask for the initial assimilation of the data, due to the massive size of the data and inability to fit into conventional RAM memory. Initially, we created a single massive dataframe which encompassed all sensor data over all the years (with 25s granularity) associated with one (of the 126) node locations, and also cleaned that data and exported it to parquet format, which we had excellent results with for speed of calculations and IO functionality. This served as our initial examination, until we expanded to multiple sets of nodes, comparing and contrasting parameters. We could extensively clean the data, remove any missing values, and then depending on the algorithm employed, use scikit-learn's StandardScaler function to normalize the data. We focused our investigation around unsupervised machine learning algorithms such as Isolation Forest, DBSCAN, HDBSCAN, and k-means clustering. Many of our plots were uploaded to our heroku page, where we would store them in the 'Unsupervised Learning' tab.

For DBSCAN, we specifically started by creating a version of gridsearch (using different combinations of eps and min_samples), running the algorithm, and classifying time-series data into anomalous and normal clusters, determining how many points would land in each cluster, the number of clusters discovered, and overall how the algorithm would work. This turned out to be enormously helpful understanding how the algorithm worked at a base level. At that point, we switched to conventional tuning methods such as nearest neighbor (since the eps value is proportional to the expected number of neighbors discovered, we could use the nearest neighbours to reach a fair estimation for eps (using a form of the knee-method). This is how we discovered our optimal eps (0.003). For **Isolation Forest**, we were able to see fairly accurate results. We initially used plots of the datapoint anomaly scores that were calculated by the algorithm (see below), and then formulated contamination values that made sense. We noted that contamination assumptions could vary (assuming 1, 2, 3, or 4% of the data was 'contaminated' for instance), and we could thus plot multiple outputs to compare and contrast results. For our k-means clustering analysis, we started off with examining clusters of k=4 and k=6, but eventually found via the elbow plot that the optimal number of clusters for many windows was actually k=3. K-means does not inherently require 'tuning', but it did provide insightful clusters that added to the insight we derived from DBSCAN and Isolation Forest.

Some challenges (also listed below) included the fact that we did not have truly labeled data, it was ALL unlabelled, so we had to truly rely on unsupervised machine learning techniques. However, we did know that there was a period of extremely low temperatures (2019 storm that hit Chicago), so we used this as a baseline to determine if our algorithms could find those windows, and they were able to successfully do so.

File Size - The massive size of the core csv datafile (over 310 GB) was extremely difficult to parse through due to the memory size requirements, including the limitation of pandas. We made all efforts to process elements in parallel, and had to learn a considerable amount of Dask in order to process the file effectively and smoothly. We categorized certain columns as categorical which sped things up as well. Indexing and persisting the filtered data in memory alleviated some time constraint issues as well.

Complexity - Time-series anomaly detection is not always easy for anomaly detection analysis, and the size of the actual window under examination makes a substantial difference in results sometimes. When we used very large time windows (a year), we would get different results than when we used smaller time windows. We chose to try to search particular window sizes (usually 3 month windows), due to higher granularity in our anomaly detection.

UML - Tuning parameters for DBSCAN is sometimes limited, due to the need for subject-matter expertise of the actual dataset itself. High computational costs for algorithms such as DBSCAN - High computational expense of average O(n log(n)) coming from a need to execute a neighbourhood query for each point; killing my time...also, the quality of the clustering results strongly depends on the measure you choose to compare the time-series (the standard euclidean distance measure may not be ideal for time-series, maybe).

Real-World Issues - Filtering: There were some node sensors that flat-lined for a window of time (conventional malfunction or loss of data for a certain time period), these windows were documented and sometimes avoided.

Plotting - It should be understood that given the granular nature of the streaming data (every 25-30s), a single dimension single variate parameter results in **over 1.2 M observations per year, and plotting time ranges was challenging. We leveraged** high level plotting libraries such as Matplotlib/Seaborn initially which helped, but for shorter timeframes where real-time granularity was necessary, Altair and Plotly were used exclusively. Plotly was also used in some cases so that the Dash version of the plots could be plotted online (heroku). We also experimented with Datashader, which was enormously powerful in plotting millions of data points.

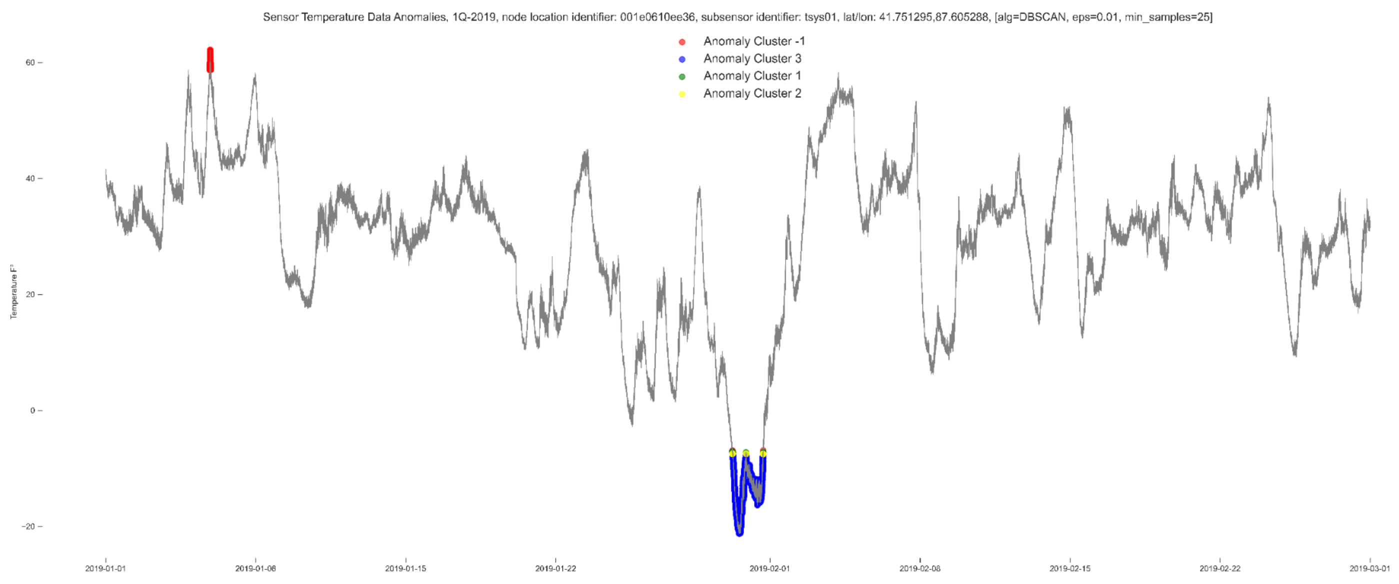
Unsupervised Evaluation

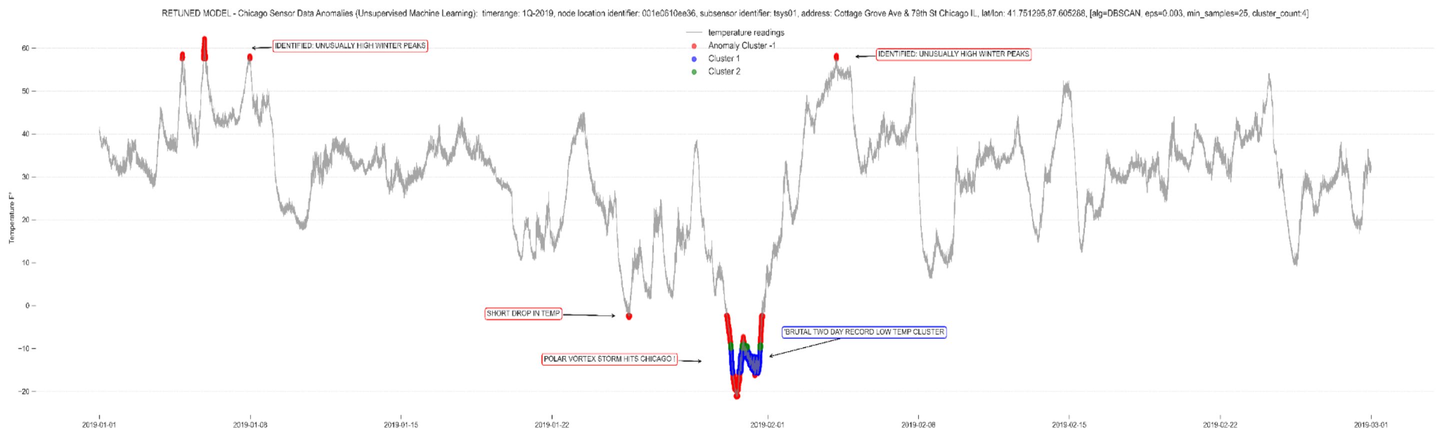
With DBSCAN we were surprised to find that our eps hyperparameter that we used after extensive tuning analysis was actually quite small. This was attributed to the large numbers of datapoints (and their close proximity to one another in the time space). We did also find that arbitrarily setting the hyperparameters for DBSCAN (eps and min_pts values) did **not** work in any form, tuning was necessary to even get us in the right ballpark. Without using the right hyperparameters, we saw (as expected) enormously different clustering results, or sometimes even having more cluster -1 anomalies than actual non-anomalies (a sure indicator that hyperparameters were massively wrong).

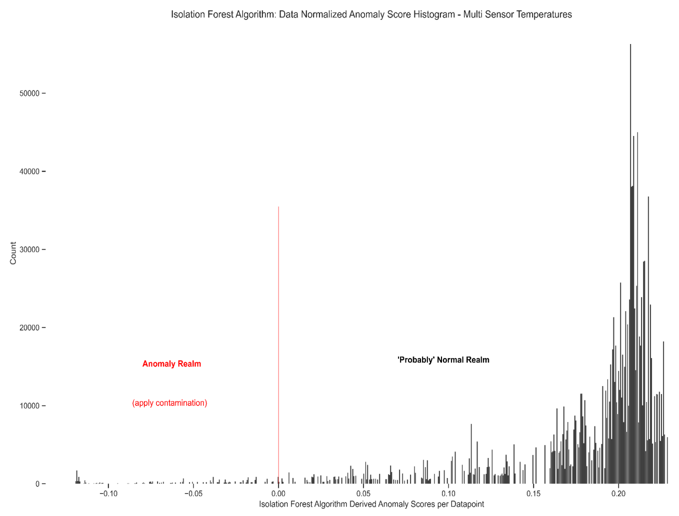
With k-means clustering, we were able to demonstrate that with relative accuracy, we could cluster a year of temperature data into four clusters and actually segment sensor data into clusters of seasons (with the goal of determining if we could align with Summer/Fall/Winter/Spring). We were also able to derive incredibly insightful anomaly detection of temperature readings in Chicago during what we would later find was the Polar Vortex storm of 2019. We were able to successfully create highly accurate anomaly detector code to gather insight into unusual behavior of sensors (variations, broken or malfunctioning zones, etc). This also allowed us to create interactive plots where users could change the time window size to get closeup visualization of the anomalies. We found that isolation forest also offered similar insights, but that it was mainly tailored towards determining max and min clusters (and not usually identifying anomalous points that were near the 'middle' of the data.

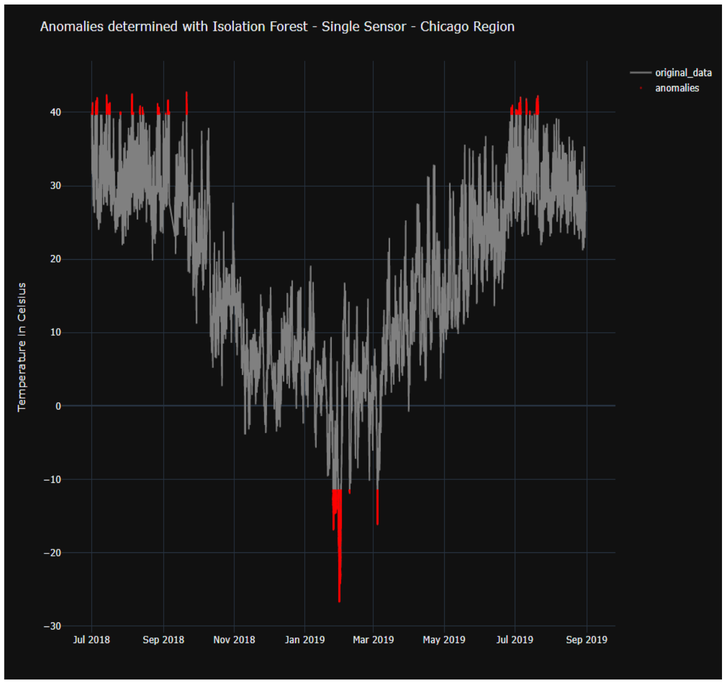
Some insight we gained on the data is that we were effectively measuring the 'weather' for many of the sensors, and that can be relatively random (although there are trends in temperature). We believe this was actually a good thing, given that most anomaly detection on sensors is on 'sinusoidal' trends that are relatively easy to do anomaly detection on, while the goal of our analysis was to eventually extend the algorithm to detect unusual malfunctioning windows. By using real-world data, we ran into real-world problems, which is precisely what we wanted experience with. What did not seem to work well was using too high of a contamination parameter for Isolation Forest, as it resulted in higher and higher amounts of anomalies detected, sometimes in areas that didn't make sense (as expected).

What worked effectively was running the anomaly detection algorithm (once tuned) in a loop on windows of two-three months each !


Sensor Temperature Data Anomalies, 1Q-2019, node location identifier: 001e0610ee36, subsensor identifier: tsys01, lat/lon: 41.751295,87.605288, [alg=DBSCAN, eps=0.01, min_samples=25]
Anomaly Cluster -1
Anomaly Cluster 3
Anomaly Cluster 1
Anomaly Cluster 2
Temperature F°
60
40
20
0
−20
2019-01-01
2019-01-08
2019-01-15
2019-01-22
2019-02-01
2019-02-08
2019-02-15
2019-02-22
2019-03-01


RETUNED MODEL - Chicago Sensor Data Anomalies (Unsupervised Machine Learning): timerange: 1Q-2019, node location identifier: 001e0610ee36, subsensor identifier: tsys01, address: Cottage Grove Ave & 79th St Chicago IL, lat/lon: 41.751295,87.605288, [alg=DBSCAN, eps=0.003, min_samples=25, cluster_count:4]
temperature readings
Anomaly Cluster -1
Cluster 1
Cluster 2
IDENTIFIED: UNUSUALLY HIGH WINTER PEAKS
IDENTIFIED: UNUSUALLY HIGH WINTER PEAKS
SHORT DROP IN TEMP
POLAR VORTEX STORM HITS CHICAGO !
'BRUTAL TWO DAY RECORD LOW TEMP CLUSTER
Temperature F°
2019-01-01
2019-01-08
2019-01-15
2019-01-22
2019-02-01
2019-02-08
2019-02-15
2019-02-22
2019-03-01


Isolation Forest Algorithm: Data Normalized Anomaly Score Histogram - Multi Sensor Temperatures
Count
Anomaly Realm
(apply contamination)
'Probably' Normal Realm
Isolation Forest Algorithm Derived Anomaly Scores per Datapoint
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20


Anomalies determined with Isolation Forest - Single Sensor - Chicago Region
original_data
anomalies
Temperature in Celsius
Jul 2018
Sep 2018
Nov 2018
Jan 2019
Mar 2019
May 2019
Jul 2019
Sep 2019

Discussion

What we learned from doing Part A, what surprised us, and how we would extend our solution with more time/resources: Part A really helped develop our understanding that different models can have drastically different results. It also helped us with learning how useful dimensionality reduction can be. The results difference between using PCA data and TSNE data was also significant, so in the future we will definitely continue to test our data using different forms of dimensionality reduction and models. Given more time, we would focus more on hypertuning our parameters to try and get even more accurate results.

What we learned from doing Part B, what surprised us, and how we would extend our solution with more time/resources: When dealing with very large dataset, we feel it is critical to know Dask (and also fundamental concepts like partition sizes, block sizes, persist vs compute commands, etc), parquet-based dataframe forms, multiprocessing on pandas, and any edge on speeding up dataframe processing. Considerable time can be lost without this. We feel grouping time-series data into clusters is sometimes quite subjective; 'subject-matter expertise' and judgement calls become a key component of the process. Time-series dataset machine learning examination is still really being developed, completely new methods (such as Matrix-Profile) are being created on a yearly basis. This dataset allows one to create whole new approaches on how to deal with the challenging cases of longitudinal data and time-series for sensors. Time-series output results depend on many factors (window size, base clustering algorithm, pre-filtering assumptions, contamination percentage assumptions, etc.) part of the exciting part is developing whole new ways of approaching this problem. We also believe that when doing true unsupervised machine learning (zero labels), subject matter expertise of the system under investigation (and its properties) is a key component of analysis. The anomalous datapoint count varied *dramatically* based on input hyperparameters (contamination parameter, eps and min_samples hyperparameters, etc). We found it startling how well DBSCAN worked in terms of identifying anomalies, but also how accurate it seemed to be in identifying clusters of unusual data values that were not necessarily defined as anomalies, but also were not part of the main 'group' (cluster 0). With more time/resources, investigation into brand-new approaches such as Matrix-Profile would have been fascinating to leverage.

Ethical Issues that could arise in providing a solution to Par A, and how we would address them:
If someone had a live feed of the data we used in part A and it fell into the wrong hands, there could be issues of knowing when a user is driving and away from home. To address this, we would need to either never have live data or we could ensure that the data is encrypted, so it is more difficult to intercept.

Ethical Issues that could arise in providing a solution to Part B, and how we would address them:
According to the original AoT sensor network design, the sensors do **not** have the capability to measure or identify individuals, and that microphones and cameras in public spaces do not collect sensitive personally identifiable information (PII). However, the AoT original team did expect some pushback at some point, so they created governance policies documented here. The newer generations of sensors have the ability to take pictures, which could give away personal information like walking patterns, GPS coordinate locations of individuals, proximity detection, etc. To address these, it would be necessary to use deep learning methods conceivably to ‘anonymize’ any person found in an image (similar to what Google does in Google Maps application where it makes faces ‘fuzzy’).

Statement of Work

Michael primarily focused on the supervised learning section and the Heroku website while Tom focused on the unsupervised learning section. We cross collaborated to feed the other ideas and tips to progress the work and take it to the next level. We feel that both our code, our documentation, our analysis, and going above and beyond by creating the heroku online application demonstrated our ability as data scientists.