A Predictive Machine Learning Pipeline for Large-Scale Fitness Data

Julian McAuley, Ph.D.
David Doerner, Jason Gilberg, Masashi Omori, Patrick Mulrooney

Abstract:

A three-part machine learning pipeline was created to determine the level of accuracy attainable in predicting large-scale consumer fitness data on a commercial application, which tracked workouts using wearable devices and smartphones. The pipeline was built using a sequence of distributed unsupervised and supervised learning algorithms. A two-step sequence of bisecting K-Means clustering separated the dataset first into route clusters based solely on route information to identify targets for regression. Then, those route clusters were split into performance clusters based on the temporal and biometric data from the user, which allowed for the generation of new features that capture how well the user performs on particular routes. The route information and the user's historical performance were fit with four different regression models using the workout duration as the target value. The resulting regression models were assembled into an ensemble to make predictions for a specific user on a given route input. Despite inconsistencies due to malfunctions of the devices providing sensor data and the dependency on inherently variable user data, the pipeline was successful in creating predictions using a very limited number of features with accuracy levels that could be used for a novel fitness application.

Introduction and Question Formulation:

The overall goal of the Large-Scale Fitness Data Analysis project is to create high level predictive models and route recommendations for a general end user of a wearable fitness tracker. The data used for this project is largely time series data from the Endomondo fitness app with metadata for each workout and some metadata for each user. The size of the data is in the tens of gigabytes in JSON format. This project attempts to use information derived from the data about the difficulty of routes, ability of the user, and similarities between different workouts across users to find patterns in the behavior of users. The end goal of the project is to provide a reasonable prediction for a given user on a particular route based on their previous performance coupled with the performance of other users on similar routes. Although the predictor system will be built using a static dataset, it would be possible to adapt streaming data into the predictor system, so a user would be able to see how their performance on more recent routes has affected the system's estimates for their future performance.

This data science problem begins at the initial data wrangling of scraped data in JSON format, which needs to be aggregated and organized into a usable form, presumably a database. Then, the data needs to be explored and evaluated to verify its quality to determine the level of cleaning required to provide an adequate input for machine learning. The machine learning pipeline itself has multiple clustering steps that are fed into a regression. Using ensemble methods, the results from the regression are stacked to create a final prediction. Finally, throughout various phases data visualizations are necessary for exploratory analysis to aid in the data cleaning phases and to verify distinct clustering.

To build a prediction and recommender system, underlying assumptions have been made about what should be possible to achieve with the different aspects of the data. In terms of routes, there should be a limited number of types of geographic features that compose a route. These features can be aggregated into summary statistics that represent the entirety of the route in terms of a few fundamental values. Thus, routes with similar, summarized geographic features can be clustered to minimize the variance of the geographic features used for clustering. Users may not have a defined pace or tendency that persists through all their workouts; however, there should also be a limited number of different tendencies, which would allow the performance of users in one route cluster to be clustered into different performance clusters to minimize variance of the total time of the workout. Each performance profile cluster should lend itself to a general predictive model for duration and pace. With a distribution of the user's behavior coupled with predictive models for each performance profile as well as route information, the amalgamated result is a predictive machine learning pipeline for the user's probable performance of a given route.

The Data Science Team:

Julian McAuley, PhD
Advisor
McAuley received a Ph.D. from the Australian National University in Computer Science. He was a well-regarded postdoctoral researcher at Stanford University from 2011 to 2014 before joining the UC San Diego Department of Computer Science and Engineering. McAuley's expertise is data mining, machine learning, and recommender systems. He has applied these techniques to large volumes of social network data (Amazon, Reddit, Yelp, Facebook, BeerAdvocate) and understanding users' opinions.

David Doerner
Chief Analytics Officer
Doerner received a Bachelor's degree in Biochemistry and Management Sciences from UC San Diego with a strong background in life sciences and statistics. At Ajinomoto Althea, he is a resident subject matter expert in crystallization scale up, automation, and analytics in

the Crystalomics Department. With strengths in predictive modeling and machine learning, the team is excited to have him on board as Chief Analytics Officer. Doerner will oversee the development of the regression algorithms and other machine learning techniques that predict performance on routes.

Jason Gilberg
Chief Business Development Officer
Gilberg received a Bachelor's degree in Mechanical Engineering with a focus in Biomechanics from Brown University with training in physical systems and statistical analysis. With experience in sports and wearable technology, the team has called upon Gilberg to conduct market research and develop a business strategy, as well as coordinate product management with the Analytics team. Currently, he works as a Baseball Research and Development Analyst with a focus on performance science for the Los Angeles Dodgers.

Patrick Mulrooney
Chief Data Architect
Mulrooney received a Bachelor's degree in Mathematics from University of Illinois at Urbana-Champaign. The native Midwesterner has settled on the West Coast, utilizing his expertise in data storage. He recently transitioned from a Senior Storage Engineer at the San Diego Supercomputer Center to Rady Children's Institute for Genomic Medicine as a High Performance Computing Engineer. His experience in Systems Engineering and achievements as a long-distance runner for the past decade make Mulrooney the ideal addition to the team as Chief Data Architect.

Masashi Omori
Chief Financial Officer
Omori received a Bachelor's degree in Mathematics and Computer Science from UC San Diego. His background in 3D modeling, software engineering and passion for CrossFit and tennis make him an integral part of the team. With his current work as a software operations engineer at Intuit, creator of applications such as TurboTax and Quickbooks, Omori has been tapped to be the team's CFO for his ability to maintain the system's integrity while minimizing both financial and computational expenses.

Data Acquisition:

Endomondo is a fitness tracking application that allows users to track workouts using variety of sensor technology. Endomondo included data access controls that allow users to make their exercise data private, friends only, or public. Endomondo provided a RESTful

API for programmatically interacting with user workout data. Endomondo had a large diverse community of millions of users that record workouts all over the world with the majority of its user base in Europe. Using the metadata from each individual workout with the time series that describes the course of the workout facilitated analysis to be performed fluidly, allowing new features to be created on the fly from the time series in case the workout summary was missing important details about the workout.

Endomondo used sequentially incrementing numeric ids to identify both users and individual workouts. When using the RESTful API to query a user, the numeric user identifiers could be used to pull their associated data. The data returned a list of the numeric identifiers of their workouts in addition to a variety of metadata describing the user and overall workout information in JSON format. With the user and workout identifier, detailed information could be obtained about each workout in a JSON format. By incrementing through user identifiers, a large amount of workout data was collected from between 2010 and late 2014.

```
//pseudo code
for user_id in (1..1000)
    workouts = get  https://www.endomondo.com/rest/v1/users/$user_id
    for workout_id in workouts
        get http://www.endomondo.com/rest/v1/users/$user_id/workouts/$workout_id
```

Using this technique, approximately 40GB of workout data was collected. There were approximately 1364 users who recorded 960,000 unique workouts containing a total of 370,000,000 time series records. Collecting updated information would require crawling all users a second time and looking for new workout records. Due to time constraints and the speed at which the interface can be queried, it would not be practical to collect updated workout information within the scope of the project.

There may be additional value added from the inclusion of additional information from public APIs to provide weather, air quality, and geographic information. Due to the time range of the data and the quantity of data, it has proved difficult to find affordable APIs that provide an appropriate applicable size of useful information for historic information.

Data Preparation:

The starting point of the data was a per workout JSON file. A Jupyter notebook was used to facilitate the movement of data into PostgreSQL and InfluxDB databases. Two tables were created per workout category (run, bike, workout, etc): one containing per workout metadata and the other the time series data for each workout. The JSON data was structured in a similar format. Each workout iterated over one at a time and inserted into the databases. The JSON data only included fields for which there was data. Prior to insertion into the databases several changes needed to be made to provide a level of consistency in terms of number of fields.

The data contained several fields that cannot be translated due to a lack of reference, and are of minimal benefit to the analysis so they were removed. The data contained multiple fields with numeric identifiers. Lookup tables were constructed to map the identifiers to useful categorical values. The data contains UTC timestamps in ISO 8601 format which were translated to Unix time. Due to database field type range limitations some extremely implausible numeric data was removed, e.g. altitude values of over a hundred million kilometers.

Histogram visualizations were used for basic data exploratory analysis during the initial stages of data loading. Minimal steps were taken to clean the data until it was loaded into the database, but the exploration revealed that data contained a variety of improbable values. After data was loaded into the databases, it was found that the in-memory indexing method utilized by InfluxDB would be problematic and was abandoned.

Instead, Postgres databases were utilized to load the data. The Multiversion Concurrency Control feature of PostgreSQL and relatively limited disk space required dividing updates into multiple parts and routinely vacuuming old records. Additional indexes were created to improve performance for certain updates, but they were later removed to preserve space.

After removing a large amount of duplicate time series records, data cleaning efforts were focused on the five time series variables: heart rate, speed, altitude, latitude/longitude, and time. Each variable presented some unique and some common problems. Obvious incorrect values such as altitudes lower than possible and times dating back over twenty years were easy to flag and resolve. For other inconsistencies such as improbable heart rates, the data was smoothed to remove outliers within each workout. Altitude values presented the greatest amount of problematic data. With most latitude and longitude pairs being represented more than once in the dataset, a new field was generated containing the median altitude value for each location and the records updated with the appropriate value for their location. Additionally, standard deviations were calculated per variable in each workout and values above or below two standard deviations from the mean were flagged and reviewed.

Several new fields were generated such as first and second derivatives of the altitude, heart rate, etc. These fields were used to programmatically find inconsistent or incorrect data that was otherwise difficult to distinguish. To find inconsistent latitude and longitude information, a euclidean distance was calculated between sequential time series records. This combined with the time delta to create a new speed field with a standard unit. Ultimately, this fields were intended to be incorporated into the predictive modeling aspect of the project; however, there was not enough time to evaluate the colinearity of these terms with the existing terms to justify their utility before spending additional resources cleaning out values that could potentially derail a regression model.

Additional derived fields are unlikely to drastically improve the current machine learning pipeline, but additional metadata, such as device information, might aid in future data cleaning. However, the inclusion of the right feature into the right type of regression model in a similar pipeline could prove useful even though the implementation may not be trivial. Some preliminary work was done to incorporate additional data from external APIs, such as altitude lookup by geographic location, which showed initial promise, but time constraints prevented further incorporation within the prediction pipeline. For sanity checking, an API was used to determine the timezone and start time of the workout by combining the timestamp and first latitude and longitude pair for each unique workout. A combination of these different additional strategies could provide enough additional predicting power to cross the 10% error threshold that was not crossed with the current pipeline.

Analysis Methods:

The first step to becoming familiar with this dataset to be able to perform meaningful analysis was to look at specific cases to materialize what was contained in the JSON. One user was chosen out of a list of most active users for further analysis. The routes were mapped on a geographical map as seen in Figure 1:

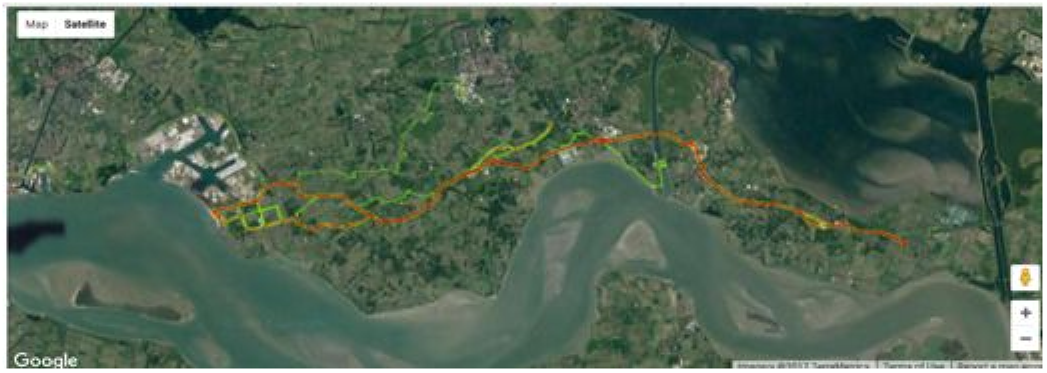


Figure 1: A geospatial heatmap of frequency of workout routes by a single user with red as most frequent and green as least frequent.

The geospatial and temporal aspects of the data identified this user's residence and workplace, which was an interesting (and unintended) result of deeply analyzing information for this one specific user. This demonstrates the power that location data in elucidating seemingly orthogonal personal information.

To further understand how the data collected from a smartphone was converted into the data contained in the dataset, a run along a known route at a known rate was performed, and the json output from the app was analyzed to determine units for many pieces of metadata. The pivotal finding at this stage was determining that the 500 points in the time series were equidistant. This explained why each segment in the time series was labelled with a different number of seconds to describe how long it took the user to traverse that particular section of the route in the time series. At this point, elapsed time throughout the workout was identified as a target for prediction that could be used for supervised learning to create models whose accuracy could be measured. Thus, the main objective for a large-scale data science problem surfaced.

With knowledge of the great variation in how individuals may approach workouts, or more specifically runs, attempting to create a general regression algorithm to predict the performance of any user on any route would be futile. Thus, the machine learning pipeline that emerged was to first use clustering to break out specific groups of cases from the general pool of workouts. Due to the continuous numeric nature of the data, K-Means clustering appeared as a straightforward choice to start partitioning. After attempting to simultaneously cluster information about how a user performed with information about the route itself failed, a two-step clustering methodology was devised. The route information would be used first to separate the different types of routes that a user could run, which would ultimately be the input of the user choosing a route or a recommender system suggesting one. Then, the workout performance summary in each cluster of routes was clustered in a second stage. This information would not be accessible to the predictor system since it is generated during the workout that the predictor is estimating. The performance clustering created a sublevel under the route clusters that would a proxy to performance of the user in the corresponding route cluster since the user could simply be walking a route, jogging a route, or trying to achieve a personal best on the route. This would allow the predictor to give the user an idea of what times to expect across a grid of different levels of workout intensity.

Findings:

The analysis of the existing data shows that future effort should be undertaken to improve the data quality at the source level. Several different techniques could be useful, but the first steps in identifying the sources of inaccuracy are finding trends between the different sources of data and the frequency of inaccuracy in related data. The various derived fields are extremely effective as a tool in identifying inaccuracy. Removing the restriction of representing the time series over 500 points should improve the finding of errant data. Incorporating external data sources, and building up an on-prem database of the information would provide a cost-effective lookup for data validation. Using these additional derived fields in the workout summaries should also enhance the accuracy of models produced with this data; although, it can be shown that a reasonable level of accuracy is achievable with a small number of fields by using a sophisticated, well-tuned machine learning pipeline.

The machine learning pipeline uses seven features to predict the duration of approximately 350,000 workouts within 13.2% using Mean Absolute Error or 19.4% using Root Mean Squared Error. The Mean Absolute Error is 23.4% of the standard deviation of the run workouts, and the Root Mean Squared Error is 34.4% of the standard deviation of the run workouts. The ensemble explained 88.2% of the total sum of squared, ($R^2 = 0.882$). This level of accuracy was achieved despite substantial amounts of missing data for altitude, speed, and heart rate, which made up ⅗ of the features used for clustering and ½ of the features used for regression. With more diligent data collection and curation as well as use of the additional features generated with the time series data, these predictions can be made even more accurately across the other types of workouts in the dataset that utilize geospatial data.

Several visualizations were made in the performance and evaluation stages to evaluate the performance of the machine learning pipeline, these visualizations lended themselves to the communication of the results of the project. For clustering, interactive parallel coordinate plots were initially used since the individual clusters could be easily identified by highlighting the cluster of interest when hovered over; however, radar charts eventually replaced them to visualize the different attributes of a given cluster as seen in the comparison in Figure 2. The radar charts create a polygonal shape that a person can remember better than comparing a large number of lines crossing over several axes, which generally makes for a more memorable and ultimately better visualization. Additionally, without the interactivity, parallel coordinate plots can become difficult to identify individual clusters when the total number of clusters grows.

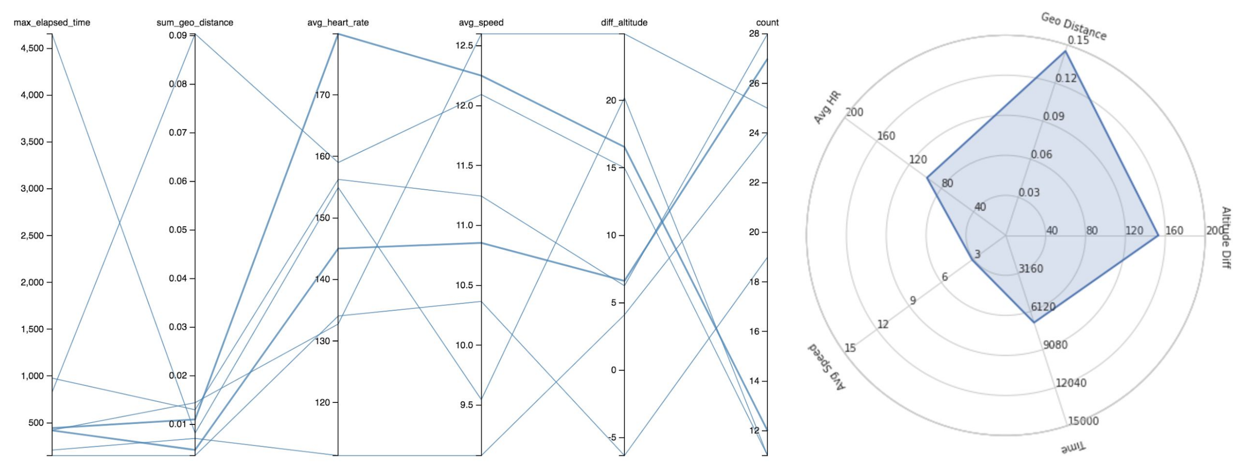


Figure 2: A comparison of two styles of visualizations for representing the attributes of a given cluster. On the left, there is a parallel coordinates plot, which uses the magnitude of the y channel of several different axes on a single segmented line to represent cluster attributes. On the right, there is a radar or spider plot, which illustrates the attributes by their distance from the center point with each direction as an individual axis.

For regression analysis, different error metrics were effective in comparing the fit of the model to the standard deviation, which represents the natural error or spread of the distribution itself. This was coupled with plots comparing the predicted values to actual values and the error in each of these predictions, the residual value, to the actual value to evaluate fit. When over 60 models were generated, visualization of the models was more elegantly communicated in a bar chart with each model type as a different color (see Performance and Evaluation section). These tools were used to illustrate the degree of fit and overall accuracy of the models generated to create the final predicted values.

For the end user, the overall goal is to convey a prediction of how they could possibly perform on a route. With this in mind, the value of each prediction is best conveyed simply as a numeric value.

Performance and Evaluation:

The performance of algorithms used in the project was measured by timing the task at completion. If suboptimal performance caused an operation to take longer than expected, generally the operation would be cancelled, and the algorithm was reevaluated and optimized for speed. Detailed investigation into the scalability of training the clustering and regression models was not performed since the actual prediction would be able to be performed almost instantaneously once the models were trained, and the focus of the project was to prove the feasibility of the pipeline itself. If a particular parameter of a model, type of model, or database operation seemed to be prohibitively expensive in terms of computational load, it was excluded or replaced.

Initially, data cleaning operations were performed using Python to query, analyze, derive, and update data. This method was extremely slow so in database queries were used to query, manipulate, and update the data without needing to pull down data out of its relational database. Many of the visualizations, namely the histograms, used for data engineering were also generated by PostgreSQL functions to reduce the need for large pulls from the database. To optimize and test performance of queries that would be run on the entire database, the query was first run on 10% of the database, at which point its relative performance was evaluated. Unfortunately, the sample was not always representative of edge cases found in the entire dataset, so some of these operations needed to be augmented after visualizing the entire database after the operation to catch additional outliers.

For the machine learning pipeline, a dataset of 10,000 run workouts were randomly sampled from the 10% subset of the database for early model evaluation on local machines. This was eventually scaled to a larger dataset on the master node of the future cluster on AWS. Once three slaves were introduced to the cluster, all of the run workouts in the 10% subset of the database were clustered and fit with regression models. At this point, it was clear that three of the four regression models, linear regression, decision tree regression, and random forest regression, were computationally inexpensive even with grid searches across parameter maps. Conversely, gradient-boosted trees required substantially longer training times and at times were removed from the regression algorithm. Meanwhile, smaller aspects such as result capture and evaluation were performed in the interest of time and resources. However, they were included in the final prediction ensemble because training only needed to be performed once, and afterwards, the models could be saved for future use.

The budget was monitored daily when computationally expensive operations were performed to determine the immediate cost of such an operation in case it needed to be repeated or expanded in future use. Generally, the budget was monitored weekly to get an idea of the average daily cost of maintaining the database that housed the raw data and additional features that were generated throughout the project. Of the original $2000 allotted for use of AWS, $63.26 was left unused at the end of the project.

To cluster the dataset, two clustering algorithms from the pyspark.ml package[4], a general clustering algorithm (K-Means) and a hierarchical clustering algorithm (Bisecting K-Means), were compared. Several articles[1,2,3] explain the difference between the two algorithms. K-Means is known to collapse into a local minima when clustering due to the random initialization of the centroids, which can be a big issue when considering reproducibility and optimal clusterings. Bisecting K-Means overcomes this matter by initializing one centroid for all the data points, then dividing it into two clusters using K-Means. The two clusters are determined as the clusters with the lowest total sum of squared error (SSE) among other possible bisection of the original cluster. This step is repeated until the desired number of clusters is achieved. Since the initialization step in bisecting K-Means is always the same, the creation of the following clusters are also the same. Another key point on taking up bisecting K-Means is that in the pyspark implementation of the algorithm, the bisecting step for each cluster increases parallelism which cannot be achieved in regular K-Means, making the bisecting K-Means often times quicker. For small data, the overhead of initializing multiple K-Means during bisecting K-Means can make it slower than regular K-Means. However, with large datasets, bisecting K-Means was found to scale better than K-Means. Ultimately, bisecting K-Means was chosen due to faster performance and lower clustering error when the dataset was scaled from the 10% subset to the full 350,000 workouts.

Another key factor which had to be considered when clustering was the normalization of data. Because bisecting K-Means works off of euclidean distance to calculate SSE, different normalizations could skew the cluster centroids. Pyspark offers couple of different normalization options: MaxAbsScaler, MinMaxScaler, and StandardScaler.

StandardScaler subtracts the mean from the dataset, and creates a unit variance using samples to estimate the variance of the dataset. MaxAbsScaler is a normalization which divides all the data points with the absolute value of the maximum value for that attribute. MinMaxScaler is a normalization which creates a linear scaling of the data between the specified minimum and maximum values. MaxAbsScaler and MinMaxScaler both have the same issue where an outlier can bias the normalized data, which impacts the clusters. After different normalizations were put to the test, StandardScaler was used as the approach because it produced the least biased outputs. The normalized values were also used to clean the data further by excluding data that was more than three standard deviations from the mean.

Using the bisecting K-Means along with Standard Scaler normalization, the first step of clustering was done using route descriptors (distance and net altitude difference). For interpretability, the number of route clusters were limited to 5. Once the route clustering was finished, each cluster was further clustered into three clusters by using performance descriptors (average speed, average heart rate, and duration of the run). This totals 15 clusters, which the centroids are presented in a series of radar charts in Figure 3. Then, two new features were generated to capture an individual user's performance in each cluster by averaging the speed and distance of their workouts. These features were intended to act as proxies for a user's typical performance as inputs for regression models.

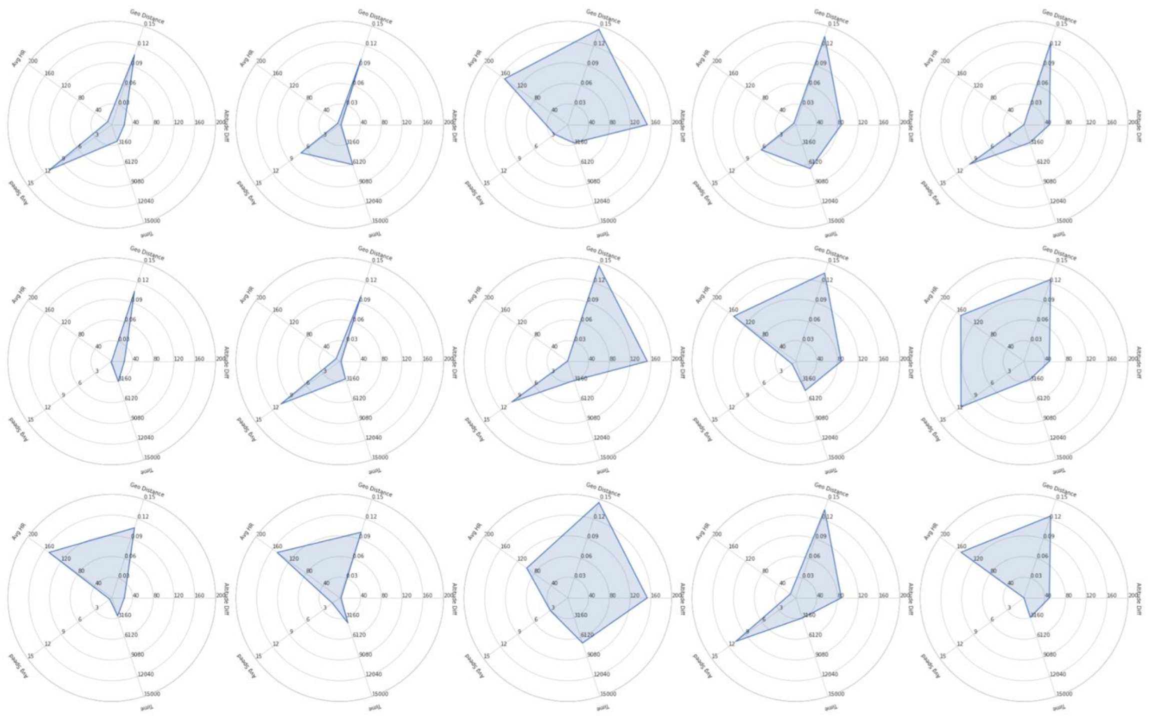


Figure 3: Radar chart representations of each of the clusters produced by two-step clustering. The columns of charts represent each of the five route clusters with each of the three rows representing the different performance clusters.

The regression models tested to evaluate predictive modeling capabilities against the cleaned dataset were all contained in the regression module of the pyspark.ml package. Regression models were chosen since both the target, elapsed_time (workout duration), and the features, geo_distance (distance), diff_altitude (altitude change), user_avg_dist (user's average distance), and user_avg_speed (user's average speed), are continuous variables. LinearRegression, DecisionTreeRegressor, RandomForestRegressor, and GBTRegressor (Gradient-Boosted Trees) were chosen to evaluate a basic linear regression model against a decision tree regression model along with two ensembles of decision tree regressors. This provided a variety of appropriate models to evaluate the performance of different algorithms on the limited number of features, which could be combined in a secondary ensemble model to further increase the prediction performance.

Each of the four models was fit with hypertuning in an attempt to optimize the ability of the algorithm to fit an appropriate model to the data regardless of the default settings. The hypertuning parameter map for each model was built to balance a number of models trained with a wide span of the model's versatility.

For linear regression, the max number of iterations varied between 5 and 10 to test different levels of granularity and repetition when fitting the model. The regularization parameter had four different levels, 0, 0.1, 1, and 10, to test different orders of magnitude to tease out potentially large hidden effects in the sparse data, as well as a simple least squares model with no regularization when the parameter was set to 0. The elastic net parameter, which designates the type of regularization to use, was set at 0 (L2 penalty for ridge regression), 0.5 (L1 + L2 for elastic net regression), and 1 (L1 for lasso regression) to test the three possible types of regularization against the three levels of regularization parameters.

For decision tree regression, the max depth of the trees was varied between 3 and 5 to test different number of splits in each tree. The minimum information gain to make a split varied between 0, 0.1, and 1 to test different levels of purity at each node.

For random forest regression, the max depth of the trees varied between 3 and 5 to test different number of splits in each tree. The maximum number of trees in the forest varied between 10, 20, and 40 trees to test how varying the averaging of multiple trees will affect the accuracy of the ensemble.

For gradient-boosted trees, the max depth of the trees varied between 3 and 5 to test different number of splits in each tree. The maximum number of trees in the forest varied between 10, 20, and 40 trees to test how a different number of trees with the default learning rate will affect the accuracy of the ensemble.

Each model was evaluated with 10-fold cross validation to reduce bias in a potential test set and to average the models over different training sets to utilize as much data as possible in the process. The performance of the regression models was evaluated through a combination of several metrics from the evaluation module of the pyspark.ml package using the RegressionEvaluation object; however, Mean Absolute Error (MAE) was chosen as the cross validation evaluator to reduce the impact of large outliers since they were common across all the factors. It would be better for a widespread fitness predictor to reserve better predictions for the widespread population rather than attempt to accurately predict outliers. After the cross validator determined the best instance of each model type against their respective parameter maps, MAE, Root Mean Squared Error (RMSE), and R-Squared (R^2) were generated for the cluster of each model. From these values, the difference between the standard deviation of the cluster and both the MAE and RMSE were calculated to find the models that best outperformed a prediction of the mean of the cluster as a predictor of cluster performance. These metrics were also normalized against the mean of the cluster that they predicted against to compare the error against the total length of workouts in that particular cluster. The relative percent error helps us generalize the error despite the length of the workout overall. A dataframe with these metrics is included for reference as appendix D**.** The performance of each model against its respective cluster is visualized in figure 4, which highlights a higher accuracy for gradient-boosted trees over the other models across the clusters.

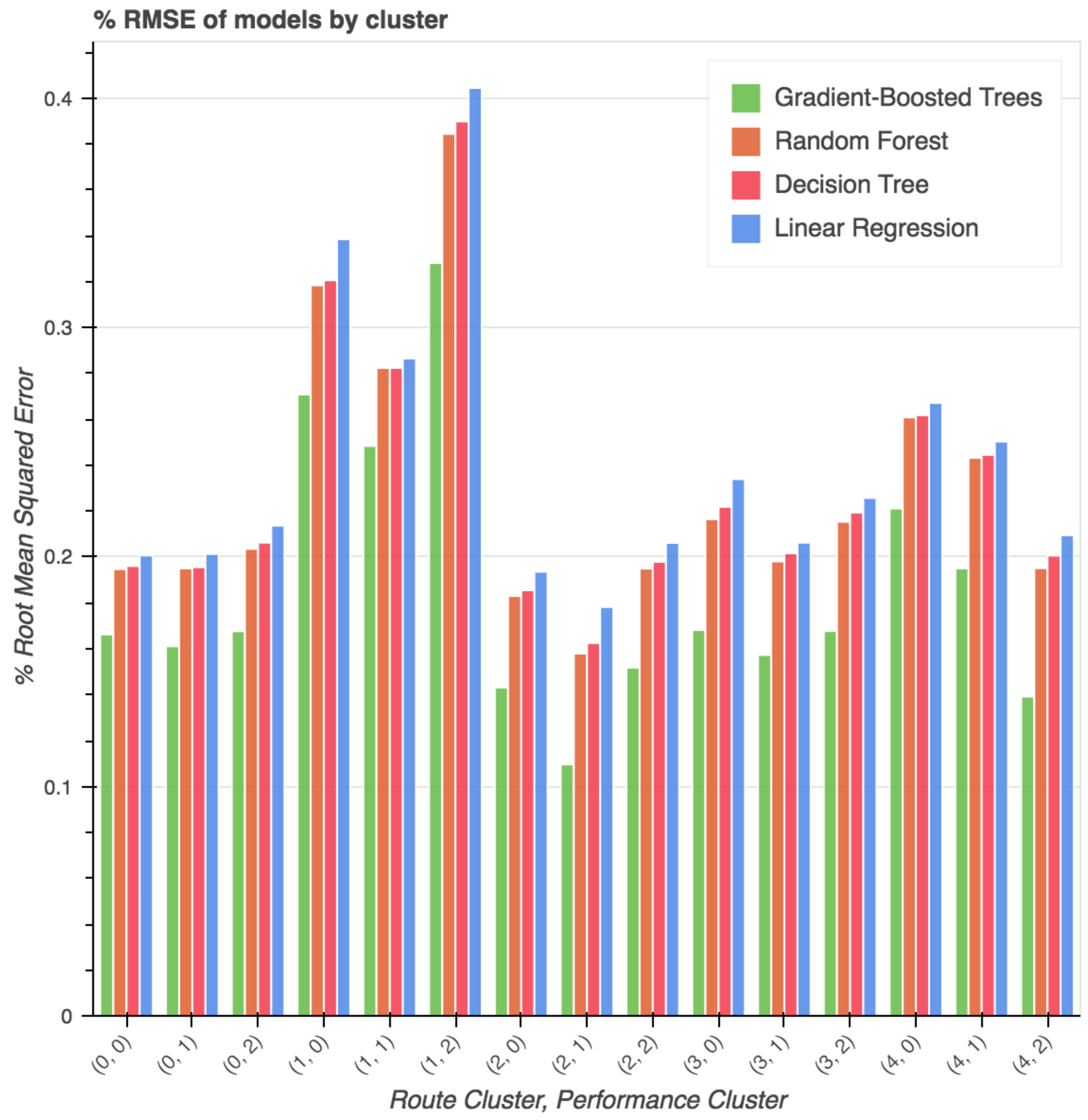


Figure 4: Relative Root Mean Squared Error as a percentage of total elapsed workout duration for each model type across each of the 15 clusters.

To better understand the fit of the models against the data, predicted values and residual values were plotted against the actual values to look for patterns in figure 5. The strong fit of the gradient-boosted tree model can be seen by the positive correlation between the predicted values and the actual values. All of the models show different magnitudes of positive correlation between the residuals of the model and the actual values that the model is trying to predict. This implies that all of the models underpredict against values lower than the mean and overpredict values greater than the mean, which shows that room for improvement of the models exist.

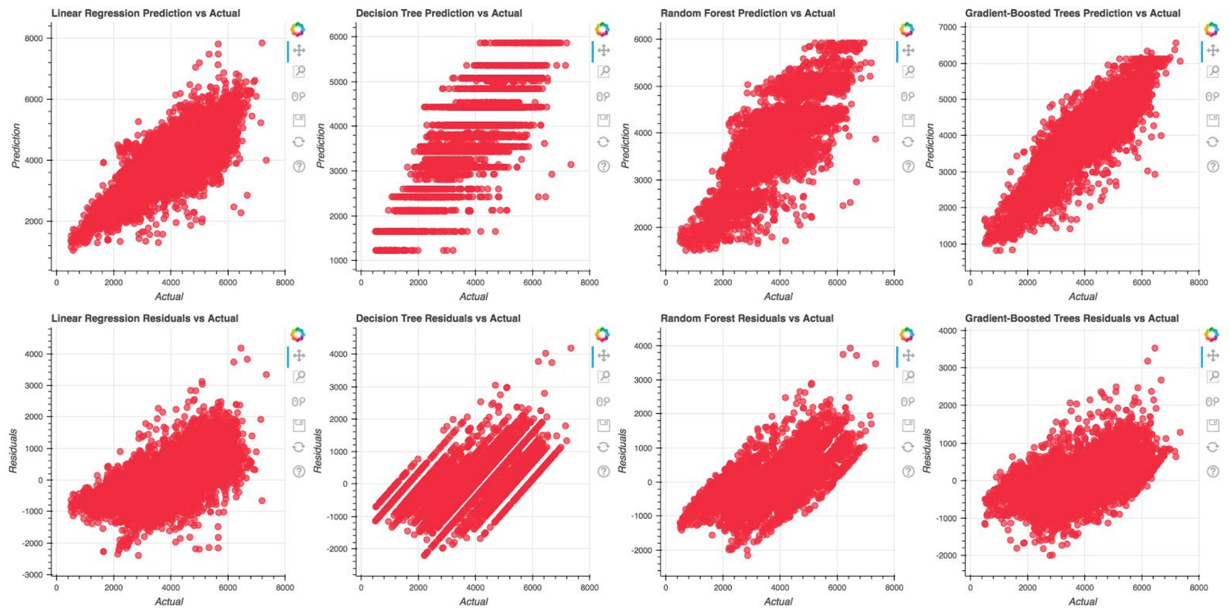


Figure 5: Predictions and residuals against actual values of workout duration for Route Cluster 2 and Performance Cluster 0, which had weaker correlation in the residual plots compared to most of the other clusters. There is a higher correlation between predicted and actual values for gradient-boosted trees than the other models.

To further improve overall prediction accuracy, an ensemble was created with a combination of all the regression models. For simplicity, a linear regression model was trained on the predictions for each workout from each of the four model types. This ensemble reduced the RMSE to 595 seconds from 789 seconds, which was the average RMSE of the models used in stacking. The MAE was reduced to 406 seconds from 591 seconds, which was the average MAE of the models used in stacking. Figure 6 shows the good correlation between predicted values and actual values, which is reflected in the 0.882 value of R^2 for the model.

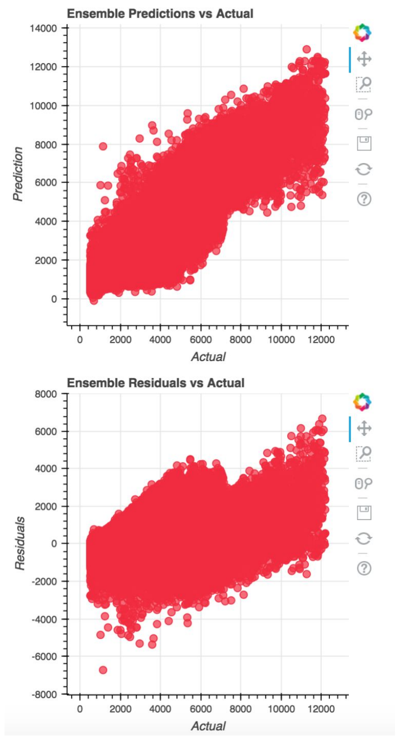


Figure 6: Predictions and residuals against actual values of workout duration for the ensemble across entire dataset. There appears to be a split around 8000 seconds, which could indicate different degrees of fit for shorter runs than longer workouts.

Conclusions:

The data contained in the Endomondo dataset is sparse and vastly inaccurate in one or more fields for a significant proportion of entries. Two main problems surfaced in the data: inaccuracies of the gps and heartbeat sensors in consumer wearable devices and the inconsistencies in depending on a user to properly use an application for its intended purpose. Despite the, a small number of features can be used to create reasonably accurate predictions across a diverse dataset with a single machine learning pipeline. This indicates that even better predictions can be made with better curation of the dataset as well as the improvement of the wearable sensors themselves. Additionally, greater prediction power can be made from increasing the number of features used in the prediction. These features could be generated from the time series of each workout or pulled from outside sources to incorporate seemingly extraneous information that could prove extremely useful when paired with the existing data. The consolidation of these outside sources with the large-scale fitness data could also elucidate physiological and psychological tendencies involved in modern day fitness. These findings have a potentially large impact on the growth of health care in the age of data science.

References

[1] K. Abirami, Dr. P.Mayilvahanan “Performance Analysis of K-Means and Bisecting K-Means Algorithms in Weblog Data”

[2] Seth Marshall, M. Emre Celebi “Comparison of Conventional and Bisecting K-Means Algorithms on Color Quantization”

[3] Dave Blodgett “Bisecting K-Means Algorithm for Clustering”

[4] “Spark Programming Guide,” https://spark.apache.org/docs/latest/programming-guide.html.

Appendices:

A. Field dictionary

Provided Time Series Fields:			
time	integer	performance	UTC Unix timestamp
altitude	numeric(20,10)	route	In meters
heart_rate	numeric(10,5)	performance	In BPM
latitude	numeric(20,10)	route	Latitude
longitude	numeric(20,10)	route	Longitude
speed	numeric(20,10)	performance	Speed at time point in kph
workoutid	integer	meta	By workout ID, links to XX_by_workout workoutid

Derived Time Series Fields:			
altitude_second	numeric(10,5)	route	First derivative of altitude. First entry by workout blank
altitude_first	numeric(10,5)	route	Second derivative of altitude. First and second by workout blank
speed_first	numeric(10,5)	performance	First derivative of speed.
geo_distance	numeric(20,10)	route	Euclidean distance (sqrt((X2-X1)^2 + (Y2 - Y1)^2)) between points. First entry by workout blank
elapsed_time	integer	route	Time since start of workout.
elapsed_distance	numeric(15,10)	route	Distance since start of workout. Equal to: (row_number - 1) * (distance / series length)
speed_ma_50	numeric(8,5)	performance	Moving average of 50 preceeding speed data points.
speed_ma_100	numeric(8,5)	performance	Moving average of 100 preceeding speed data points.
heart_rate_ma_25	numeric(8,5)	performance	Moving average of 25 preceeding heart_rate data points.
speed_by_geo	numeric(20,10)	performance	geo_distance / time delta.

altitude2	numeric(20,10)	route	median of altitude for all entries at given latitude / longitude, meters

Provided Workout Metadata Fields:		
gender	character varying	"male" or "female"
workoutid	integer	Unique ID per workout, links workoutid across tables
userid	integer	User identifying ID
start_time	integer	Unix stamp, first timestamp for each workout.
id	integer	Arbitrary ID
altitude_max	numeric(20,10)	
altitude_min	numeric(20,10)	
calories	numeric(20,10)	
distance	numeric(20,10)	Kilometers
duration	numeric(20,10)	Seconds
hydration	numeric(20,10)	Calculated using weather information and other factors
speed_avg	numeric(20,10)	Kilometers / hour
speed_max	numeric(20,10)	Kilometers / hour
humidity	integer	Percent
temperature	integer	Celsius
wind_speed	integer	Kilometers / hour

Derived Workout Metadata Fields:		
start_altitude	numeric(20,10)	Meters

start_latitude	numeric(20,10)	
start_longitude	numeric(20,10)	
series_length	integer	Count of corresponding timestamps
series_time_delta	integer[]	Delta between timestamps
series_time_delta_average	numeric(20,10)	Average delta of timestamps
timezone	character varying	timezone of start point
heart_rate_avg	numeric(20,10)	Beats per minute
heart_rate_max	numeric(20,10)	Beats per minute
elapsed_time	integer	Time since the workout
geo_distance	numeric(15,10)	Sum of geo_distance field.
altitude_max2	numeric(20,10)	Maximum altitude, meters
altitude_min2	numeric(20,10)	Minimum altitude, meters
speed_avg2	numeric(20,10)	Kilometers / hour
speed_max2	numeric(20,10)	Kilometers / hour

B. DSE MAS Knowledge Applied to the Project [5 points]

The following combination of programmatic languages, packages, database management systems, and data structures will be used in this project. All of the work will be made publically available through the following github repository: https://github.com/mas-dse-pjmulroo/fitness_capstone.

Tool	**Use**
Data collection	
Curl	Used to collect data from the Endomondo API
Python	Used automate the data scraping process
Perl	Initial manipulation of JSON data

Data ingest	
Python / Jupyter	Used to translate JSON to database format
JSON	Data format returned by Endomondo API
Storing data	
InfluxDB	Time series database contains copy of all data
InfluxDB-Python	Python library for interacting with InfluxDB
PostgreSQL	RDBMS contains copy of all data
psycopg2	Python library for interacting with PostgreSQL
Data processing	
numpy	Intermediate format
pandas	Intermediate format
scipy	Summary statistic generation
Pytz / timezonefinder	Given location and timestamp produces timezone
Data exploration	
matplotlib	Summary statistic charts
gmplot	Heatmap of workouts by location
basemap	Maps with location
bokeh	Advanced Interactive Plotting
Data analysis	
Spark	Parallel analysis
PySpark (ml package)	Python library for interacting with Spark (Machine Learning)
Infrastructure	
AWS	Database host

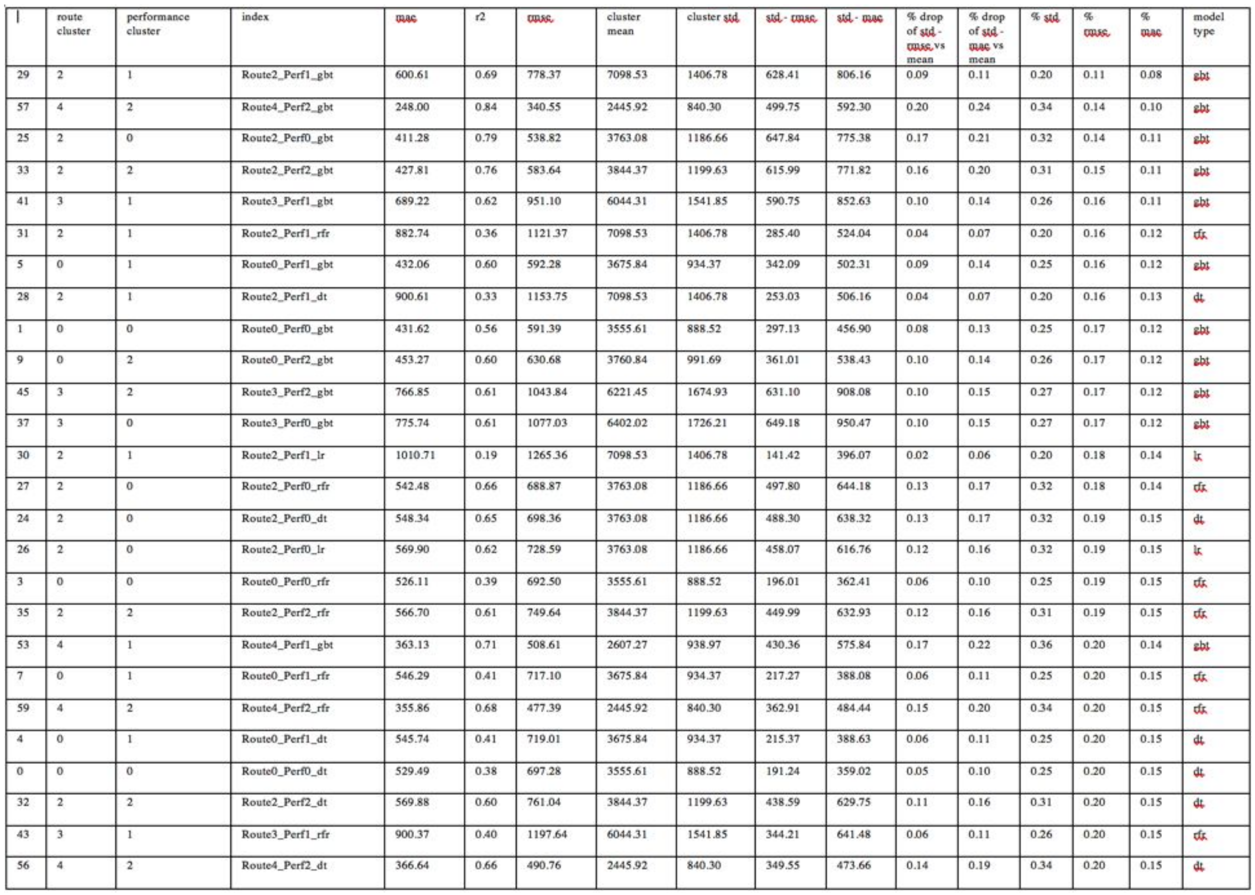
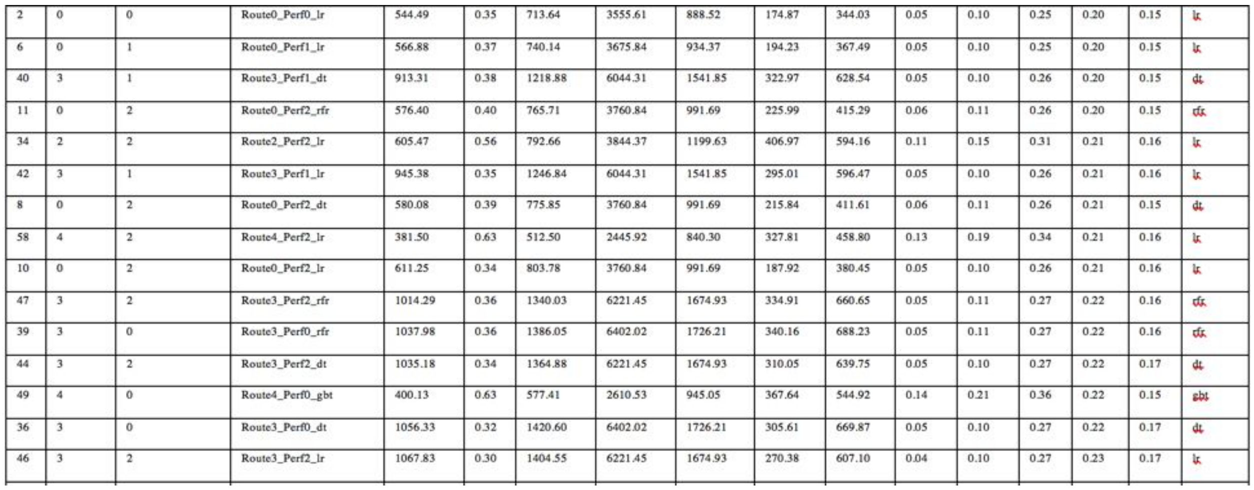
C. Data and Software Archive for Reproducibility [10 points]

- Project Links
  - Git: https://github.com/mas-dse-pjmulroo/fitness_capstone

- Please check the README for detailed information on the repository.

- Tools Set Up
  - Jupyter Notebook
    - https://jupyter.org/install.html
  - Amazon S3
    - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonS3.html
  - Spark
    - https://mas-dse.github.io/DSE230/installation/
  - Databricks setup
    - https://docs.databricks.com/user-guide/cloud-configurations/aws/index.html
  - Python
    - https://www.tutorialspoint.com/python/python_environment.html
  - Bokeh
    - http://bokeh.pydata.org/en/latest/docs/user_guide/setup.html
  - PostGreSql
    - https://wiki.postgresql.org/wiki/Detailed_installation_guides

D. Regression Results

	route cluster	performance cluster	index	mae	r2	rmse	cluster mean	cluster std	std - rmse	std - mae	% drop of std - rmse vs mean	% drop of std - mae vs mean	% std	% rmse	% mae	model type
29	2	1	Route2_Perf1_gbt	600.61	0.69	778.37	7098.53	1406.78	628.41	806.16	0.09	0.11	0.20	0.11	0.08	gbt
57	4	2	Route4_Perf2_gbt	248.00	0.84	340.55	2445.92	840.30	499.75	592.30	0.20	0.24	0.34	0.14	0.10	gbt
25	2	0	Route2_Perf0_gbt	411.28	0.79	538.82	3763.08	1186.66	647.84	775.38	0.17	0.21	0.32	0.14	0.11	gbt
33	2	2	Route2_Perf2_gbt	427.81	0.76	583.64	3844.37	1199.63	615.99	771.82	0.16	0.20	0.31	0.15	0.11	gbt
41	3	1	Route3_Perf1_gbt	689.22	0.62	951.10	6044.31	1541.85	590.75	852.63	0.10	0.14	0.26	0.16	0.11	gbt
31	2	1	Route2_Perf1_rfr	882.74	0.36	1121.37	7098.53	1406.78	285.40	524.04	0.04	0.07	0.20	0.16	0.12	rfr
5	0	1	Route0_Perf1_gbt	432.06	0.60	592.28	3675.84	934.37	342.09	502.31	0.09	0.14	0.25	0.16	0.12	gbt
28	2	1	Route2_Perf1_dt	900.61	0.33	1153.75	7098.53	1406.78	253.03	506.16	0.04	0.07	0.20	0.16	0.13	dt
1	0	0	Route0_Perf0_gbt	431.62	0.56	591.39	3555.61	888.52	297.13	456.90	0.08	0.13	0.25	0.17	0.12	gbt
9	0	2	Route0_Perf2_gbt	453.27	0.60	630.68	3760.84	991.69	361.01	538.43	0.10	0.14	0.26	0.17	0.12	gbt
45	3	2	Route3_Perf2_gbt	766.85	0.61	1043.84	6221.45	1674.93	631.10	908.08	0.10	0.15	0.27	0.17	0.12	gbt
37	3	0	Route3_Perf0_gbt	775.74	0.61	1077.03	6402.02	1726.21	649.18	950.47	0.10	0.15	0.27	0.17	0.12	gbt
30	2	1	Route2_Perf1_lr	1010.71	0.19	1265.36	7098.53	1406.78	141.42	396.07	0.02	0.06	0.20	0.18	0.14	lr
27	2	0	Route2_Perf0_rfr	542.48	0.66	688.87	3763.08	1186.66	497.80	644.18	0.13	0.17	0.32	0.18	0.14	rfr
24	2	0	Route2_Perf0_dt	548.34	0.65	698.36	3763.08	1186.66	488.30	638.32	0.13	0.17	0.32	0.19	0.15	dt
26	2	0	Route2_Perf0_lr	569.90	0.62	728.59	3763.08	1186.66	458.07	616.76	0.12	0.16	0.32	0.19	0.15	lr
3	0	0	Route0_Perf0_rfr	526.11	0.39	692.50	3555.61	888.52	196.01	362.41	0.06	0.10	0.25	0.19	0.15	rfr
35	2	2	Route2_Perf2_rfr	566.70	0.61	749.64	3844.37	1199.63	449.99	632.93	0.12	0.16	0.31	0.19	0.15	rfr
53	4	1	Route4_Perf1_gbt	363.13	0.71	508.61	2607.27	938.97	430.36	575.84	0.17	0.22	0.36	0.20	0.14	gbt
7	0	1	Route0_Perf1_rfr	546.29	0.41	717.10	3675.84	934.37	217.27	388.08	0.06	0.11	0.25	0.20	0.15	rfr
59	4	2	Route4_Perf2_rfr	355.86	0.68	477.39	2445.92	840.30	362.91	484.44	0.15	0.20	0.34	0.20	0.15	rfr
4	0	1	Route0_Perf1_dt	545.74	0.41	719.01	3675.84	934.37	215.37	388.63	0.06	0.11	0.25	0.20	0.15	dt
0	0	0	Route0_Perf0_dt	529.49	0.38	697.28	3555.61	888.52	191.24	359.02	0.05	0.10	0.25	0.20	0.15	dt
32	2	2	Route2_Perf2_dt	569.88	0.60	761.04	3844.37	1199.63	438.59	629.75	0.11	0.16	0.31	0.20	0.15	dt
43	3	1	Route3_Perf1_rfr	900.37	0.40	1197.64	6044.31	1541.85	344.21	641.48	0.06	0.11	0.26	0.20	0.15	rfr
56	4	2	Route4_Perf2_dt	366.64	0.66	490.76	2445.92	840.30	349.55	473.66	0.14	0.19	0.34	0.20	0.15	dt
2	0	0	Route0_Perf0_lr	544.49	0.35	713.64	3555.61	888.52	174.87	344.03	0.05	0.10	0.25	0.20	0.15	lr
6	0	1	Route0_Perf1_lr	566.88	0.37	740.14	3675.84	934.37	194.23	367.49	0.05	0.10	0.25	0.20	0.15	lr
40	3	1	Route3_Perf1_dt	913.31	0.38	1218.88	6044.31	1541.85	322.97	628.54	0.05	0.10	0.26	0.20	0.15	dt
11	0	2	Route0_Perf2_rfr	576.40	0.40	765.71	3760.84	991.69	225.99	415.29	0.06	0.11	0.26	0.20	0.15	rfr
34	2	2	Route2_Perf2_lr	605.47	0.56	792.66	3844.37	1199.63	406.97	594.16	0.11	0.15	0.31	0.21	0.16	lr
42	3	1	Route3_Perf1_lr	945.38	0.35	1246.84	6044.31	1541.85	295.01	596.47	0.05	0.10	0.26	0.21	0.16	lr
8	0	2	Route0_Perf2_dt	580.08	0.39	775.85	3760.84	991.69	215.84	411.61	0.06	0.11	0.26	0.21	0.15	dt
58	4	2	Route4_Perf2_lr	381.50	0.63	512.50	2445.92	840.30	327.81	458.80	0.13	0.19	0.34	0.21	0.16	lr
10	0	2	Route0_Perf2_lr	611.25	0.34	803.78	3760.84	991.69	187.92	380.45	0.05	0.10	0.26	0.21	0.16	lr
47	3	2	Route3_Perf2_rfr	1014.29	0.36	1340.03	6221.45	1674.93	334.91	660.65	0.05	0.11	0.27	0.22	0.16	rfr
39	3	0	Route3_Perf0_rfr	1037.98	0.36	1386.05	6402.02	1726.21	340.16	688.23	0.05	0.11	0.27	0.22	0.16	rfr
44	3	2	Route3_Perf2_dt	1035.18	0.34	1364.88	6221.45	1674.93	310.05	639.75	0.05	0.10	0.27	0.22	0.17	dt
49	4	0	Route4_Perf0_gbt	400.13	0.63	577.41	2610.53	945.05	367.64	544.92	0.14	0.21	0.36	0.22	0.15	gbt
36	3	0	Route3_Perf0_dt	1056.33	0.32	1420.60	6402.02	1726.21	305.61	669.87	0.05	0.10	0.27	0.22	0.17	dt
46	3	2	Route3_Perf2_lr	1067.83	0.30	1404.55	6221.45	1674.93	270.38	607.10	0.04	0.10	0.27	0.23	0.17	lr

38	3	0	Route3_Perf0_lr	1120.80	0.25	1497.96	6402.02	1726.21	228.25	605.41	0.04	0.09	0.27	0.23	0.18	lr
55	4	1	Route4_Perf1_rfr	478.46	0.54	634.28	2607.27	938.97	304.69	460.50	0.12	0.18	0.36	0.24	0.18	rfr
52	4	1	Route4_Perf1_dt	479.57	0.54	637.62	2607.27	938.97	301.35	459.39	0.12	0.18	0.36	0.24	0.18	dt
17	1	1	Route1_Perf1_gbt	284.09	0.67	406.47	1636.21	709.33	302.86	425.24	0.19	0.26	0.43	0.25	0.17	gbt
54	4	1	Route4_Perf1_lr	492.24	0.52	652.73	2607.27	938.97	286.24	446.72	0.11	0.17	0.36	0.25	0.19	lr
51	4	0	Route4_Perf0_rfr	494.13	0.48	681.06	2610.53	945.05	263.99	450.91	0.10	0.17	0.36	0.26	0.19	rfr
48	4	0	Route4_Perf0_dt	495.91	0.48	683.54	2610.53	945.05	261.51	449.13	0.10	0.17	0.36	0.26	0.19	dt
50	4	0	Route4_Perf0_lr	507.79	0.46	697.49	2610.53	945.05	247.56	437.25	0.09	0.17	0.36	0.27	0.19	lr
13	1	0	Route1_Perf0_gbt	319.17	0.66	470.75	1738.01	804.12	333.38	484.96	0.19	0.28	0.46	0.27	0.18	gbt
16	1	1	Route1_Perf1_dt	333.55	0.58	462.10	1636.21	709.33	247.23	375.78	0.15	0.23	0.43	0.28	0.20	dt
19	1	1	Route1_Perf1_rfr	334.57	0.58	462.12	1636.21	709.33	247.21	374.76	0.15	0.23	0.43	0.28	0.20	rfr
18	1	1	Route1_Perf1_lr	340.72	0.56	468.84	1636.21	709.33	240.49	368.61	0.15	0.23	0.43	0.29	0.21	lr
15	1	0	Route1_Perf0_rfr	394.39	0.53	553.45	1738.01	804.12	250.68	409.74	0.14	0.24	0.46	0.32	0.23	rfr
12	1	0	Route1_Perf0_dt	396.46	0.52	557.32	1738.01	804.12	246.80	407.66	0.14	0.23	0.46	0.32	0.23	dt
21	1	2	Route1_Perf2_gbt	472.96	0.52	662.69	2019.53	957.86	295.17	484.90	0.15	0.24	0.47	0.33	0.23	gbt
14	1	0	Route1_Perf0_lr	414.62	0.46	588.37	1738.01	804.12	215.76	389.50	0.12	0.22	0.46	0.34	0.24	lr
23	1	2	Route1_Perf2_rfr	584.97	0.34	776.30	2019.53	957.86	181.56	372.90	0.09	0.18	0.47	0.38	0.29	rfr
20	1	2	Route1_Perf2_dt	589.03	0.32	787.31	2019.53	957.86	170.56	368.84	0.08	0.18	0.47	0.39	0.29	dt
22	1	2	Route1_Perf2_lr	614.83	0.27	816.74	2019.53	957.86	141.12	343.03	0.07	0.17	0.47	0.40	0.30	lr