COMP9417 Trachack Report

Team RE&CLA a.k.a. Emotional Support Vector Machine

Dezhao Chen z5302273, Guohuan Li z5315754,
Zijin Shen z5286962, Yuanzhao Chen z5041686,
Kevin Zhu z5206576

1 Introduction

1.1 Problem Statement

In this project, we were tasked with predicting whether customers of TracFone Wireless, an American Prepaid Wireless Provider, would like to upgrade their device according to the user data spanning across 10 tables. The 10 tables contained the following information:

1. `upgrades` - Stores a list of individuals, as well as whether they have upgraded recently or not.
2. `customer_info` - Stores the carrier, plan and date a customer joined.
3. `phone_info` - Stores the type and specifications of phones owned by each customer
4. `redemptions` - Stores each time a customer has paid money to start or renew a plan.
5. `deactivations` - Stores each time a customer has been deactivated.
6. `reactivations` - Stores each time a customer has reactivated after being deactivated.
7. `suspensions` - Stores whether a customer has been suspended or not.
8. `network_usage_domestic` - Stores each usage of SMS, MMS, cellular data, or voice calls.
9. `lrp_points` Stores each time a customer has earned LRP points. LRP stands for the loyalty rewards program, where customers can earn points by completing surveys or playing games, which can then be used to buy things of monetary value.
10. `lrp_enrollment` Stores information on customers who are enrolled in the LRP program.

For the competition, we were provided two sets of 10 tables. The first was a 'developer' dataset containing information on 55,868 individuals, as well as whether they upgraded or not. The second was the 'evaluation' dataset, containing information on 37,155 individuals, but missing the information as to whether they upgraded. The idea was to train our models on the developer set, and then use them to predict whether users in the evaluation set wanted to upgrade their devices or not. The main evaluation metric used was the F_1 score, and every day, teams could submit a prediction that would show up on a leaderboard, comparing their F_1 scores on the evaluation data with all the other teams. In addition, we were provided a data dictionary, which provided very brief explanations of what each column in the datasets meant.

1.2 Issues faced

In order to obtain predictions with high accuracy, there are two main tasks our team had to deal with: feature engineering and machine learning. Feature engineering is a crucial step for all machine learning tasks, as good feature engineering can make even sub optimal models produce good results. To help with this, we did a thorough data exploration, keeping in mind issues such as how we would deal with missing values, outliers, and redundant features. We also wanted to make sure that we were not simply blindly extracting features, so we took care to try to emphasise with the user, and generate new features that would likely indicate that a user would like to upgrade. As a result of our work, we ultimately generated two different feature extracted datasets, which we will refer to as D_1 and D_2. D_1 was generated early in development, and was used to inform the decisions made in D_2. These formed the basis for the two models included in our report.

After getting good feature extractions, the second task our team faced was in machine learning. Our team experimented with a variety of different classifiers, running hyper-parameter tuning on each one in order to maximise our results. We also experimented with majority voting and stacking algorithms with multiple high scoring classifiers, but the F_1 score were not satisfactory, so we scrapped the idea. We ultimately chose the best performing model as the one for our final submission.

2 Exploratory Data Analysis

Exploratory data analysis is an important part for understanding an overview of the data before doing data cleaning and feature engineering. In this project, we are given 10 different datasets, each of which reveal some information about the an individual, and our goal is to explore this dataset and try to get an idea of how we can handle and manipulate this data so as to best expose the important features for feature extraction.

There are many different ways to explore data, and each feature should be treated individually depending on its type, and what it represents. However, there are also some general strategies for data exploration. One of the easiest ways to explore data is to look at the data distributions. For example, looking at the `upgrades` dataset we see that the ratio of people who upgraded to people who didn't is 15,174 to 40,694, which indicates that an imbalance of target distribution exists. This gives us a baseline, since we know that predicting 'no' for every person will at least result in an accuracy of about 72%.

2.1 Categorical Data Distributions

For observing the distribution of categorical features, we use bar plots to show how the count of each category. For example, we can see that in `customer_info`, we can easily see that `carrier 1` dominates the other carriers as the most popular choice, whereas for plans, they are more evenly distributed, as show in Figure 1. This is useful to us, as it indicates that the `carrier` column is likely not going to be as useful in helping us to determine whether someone upgrades or not, based on the fact that the vast majority of people use the same carrier. On the other hand, the plan distribution is a bit more even, with the exception of plan 4, which only 145 people used. Once again, categories that are extremely rare may not give much useful information in differentiating the target values.

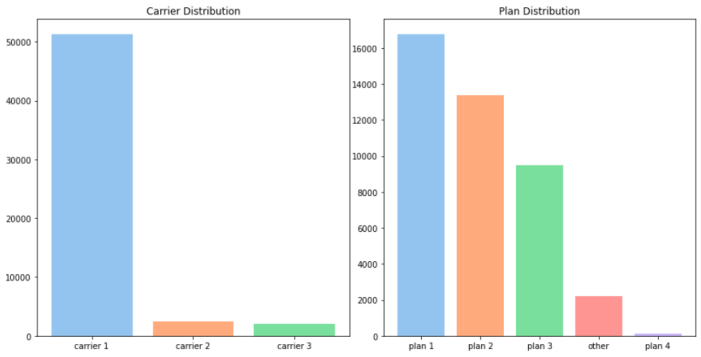

Figure 1: Carrier Distributions

2.2 Numerical Data Distributions

For numerical data, we can use histograms to show us the approximate spread across data. For example, for the `redemptions` dataset, one potential feature is to count the number of times each person has redeemed a plan. We then plot out the frequencies of each number of redemptions in the following histogram 2. Here we can see that everyone person has redeemed at least once, and the vast majority of people also only redeem once. Furthermore, this histogram also tells us that there are outliers, in that a couple of customers have redeemed 120 times or more. This tells us that during feature extraction, we will need some method to handle outliers for this feature.

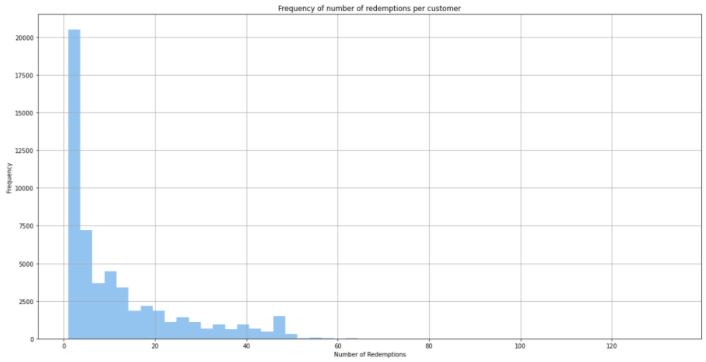

Figure 2: Frequency of Number of Redemptions Histogram

2.3 Distribution of Target Value

Another method of working out how important a category is is to plot out what percentage of people who have that category actually chose to upgrade. For example every single person who has used `BP_PAID` as their most frequent redemption channel has also chosen to upgrade, from a sample size of 75 people. Thus this shows that perhaps that this is an important feature to use.

2.4 Interesting Examples

Finally, we share some specific examples of interesting things found through data exploration.

- For the `redemptions` dataset, we noticed that some people had redeemed over a hundred times. We made a plot showing the amount they spent per redemption, as well as the time and date that they did it. Figure 3 shows the plots for one of these users. We notice that the user makes a consistent payment of about $45 per month, and then makes a series of sporadic payments, which cost about $10 or less. We suspect that these 'one-off' calls are pay-per-call models, where the user pays a fixed price for each call. We found that monthly payments were above $15, and one-off calls were below, so we used this threshold to separate out these two bits of data for feature extraction.

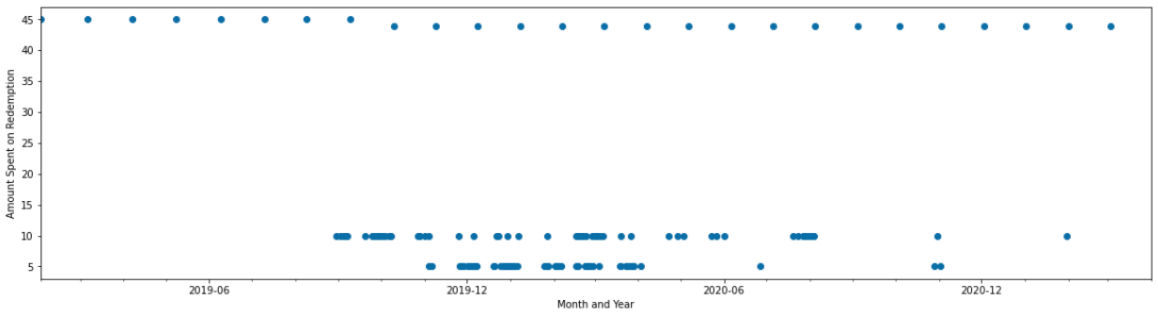

Figure 3: Plot of amount spent over time

- The `lrp_points` dataset contains 166 out of 8860 entries where the total quantity does not equal to their quantity earned. This indicates that they have earned and spent points in the past before, which also indicates that they are engaging with the LRP points system.
- For the `lte_category` feature in the `phone_info` dataset, a Google search told us that each category had certain upload and download times [2]. Hence, instead of using the category as is as a feature, we instead replaced the category with the respective download and upload times as features.

3 Methodology

Over the course of the competition, we experimented with multiple different methods of feature extractions and machine learning algorithms. Our first preliminary model was one discovered earlier on in the exploration phase, where we naively gathered features together to put into a classifier. Our second model improved on the feature extraction process of our initial model, and thus was able to perform much better. We submitted model 2 as our final submission to the TracHack competition.

3.1 Model 1

3.1.1 Data Cleaning

To deal with missing values in our dataset, we fill them with meaningful values. For numeric features, we can fill the missing values by the mean. For categorical values, they can be represented by the most frequent value of that column, or by treating them as a new category

call `is_missing`. Other times, for certain numeric features, we would like to use a sensible default value instead of the mean or median. For example, after merging the suspension dataset with the target datasets and generating a feature called `suspension_count`, people who have never been suspended before will have a missing value in that column. In this case, we should use 0 as a default value, instead of defaulting to the mean or median.

3.1.2 Feature Engineering

For feature engineering, we wanted to figure out how to merge the datasets together so that our final output would be one big table, with exactly one row for each user and one column for each feature. Hence, we needed to work out how we would merge the given datasets together, namely those that contained multiple entries per individual line ID. In these cases, we combined these multiple entries together using either their size, mean, sum, or most frequent variable of the features associated with the same line ID. For dealing with outliers, we use capping. We consider the difference between the 1st and 3rd quartile to be the inter quartile range, or IQR. Anything that lies outside of 1.5 IQR of the median is then considered to be an outlier, and is then capped to be within 1.5 IQR. This effectively removes outliers from numerical features, which can improperly weight certain features during machine learning.

For categorical features, some variables may appear extremely infrequently. For example, in the `gmsa_operating_system` row, 54,491 out of 55,868 people either have an iPhone, Android, or an unknown phone, whereas only a single person owns a Blackberry. If we choose to directly feed the information about Blackberry phones into our model, it will add minimal information due to how rare it is, and add more noise that can cause overfitting. Therefore, we group rare values together under a single category called 'other'. This also has the added effect of reducing the dimensionality after applying one hot encoding to categorical data.

Finally to for scaling, we considered three different popular scaling methods: Min-max scaling, standardised scaling and robust scaling. All the above methods can scale the values but the first two methods are sensitive more sensitive to outliers compared to robust scaler. Therefore, we chose to use robust scaling as our final method of processing. This helps to ensure our data has similar ranges and distributions, which can aid in increasing accuracy and speed of machine learning techniques.

3.1.3 Categorical Feature Encoding

In this project, we have implemented two different encoding methods: one-hot encoding and ordinal encoding. One-hot encoding is a very popular encoding method for nominal features since it does not introduce a misleading order of the features, making it good for distance-based models. However, it also suffers from the curse of dimensionality which will lower the performance of tree-based models. Ordinal encoding is for encoding the features with a natural order. This technique can also encode features that have no natural order, but are only useful in tree-based model such as decision trees or random forests. We used one-hot by default, but ordinal encoding for tree classifiers.

3.1.4 Feature selection

In our model, we use only the top 25 features as suggested by random forest classifiers. During the competition run time, we experimented with multiple statistical measures of feature selection such as chi-test, t-test, and correlation. These measures were aimed at working out which features would be more relevant and removing redundant features. However, these tests didn't perform well for non parametric models, so we ultimately decided to use the results from the RFC instead. This had the effect of decreasing the complexity of our model, whilst still maintaining a similar degree of accuracy.

3.1.5 Model training and tuning

Model training and tuning was the final part in the machine learning pipeline and gives us feedback on how we can improve the above methods.For evaluation, we used 80% of the dev data to train our model, and 20% to evaluate it. Here we tried different models like classic ML models Logistic Regression, SVM, Multilayer Perceptrons, Decisions Trees, and KNN. For ensemble learning models such as random forests, XGBoost, and Adaboost. For training non-tree-based models, we used binary or one-hot encoding, before running a grid search. For training boosting models, first we feed all the features with ordinal encoding or one-hot encoding in categorical features to the RFC, then extract only the top 25 features for use in other models.

3.2 Model 2

For model 2, we more or less started from scratch and tried to use different choices of feature engineering from the original. The biggest change in philosophy was that we would

try to understand the problem domain and choose features that would logically correspond to customers desiring upgrades, instead of blindly gathering features and doing feature selection after the fact. This resulted in a better model, which used for our final submission.

3.2.1 Data cleaning

We deal with missing values the same way as before, either using a default value, using the median, or creating a new category. We note that we prefer median this time over mean, as it is less sensitive to outliers. Other changes that we make were that for `cpu_cores`, some of the input was given as '2+4' instead of just '6'. To clean this, we simply evaluated any sums given in that table to get a final number of cores. Furthermore, `internal_storage_capacity`, many times the capacity was not given in terms of a single number, but multiple options the customer could have chosen from, for example: '32/128/256'. To handle these, we simply split up the string and took the median out of all the values.

3.2.2 Feature engineering

As before, we generate new features by simply counting or summing up the number of times data has appeared for each line ID, such as summing up the total number of kilobytes used by a user.

To deal with numerical outliers, our method of choice was using log transforms, since all the numerical values were positive. Log transforms are useful as they capture the notion of 'diminishing importance'. For example, we take the `year_released` feature and transform that feature that to how old the phone is on the current date in years. Theoretically, the difference between two people, one with a 2 year old phone and the other with a 3 year old phone is the same as the difference between two people, one with a 20 year old phone, and the other with a 21 year old phone. However, in terms of whether they will upgrade or not, the difference between 2 or 3 years could greatly affect whether a person wants to upgrade, whereas if two people have already kept their phone for over 20 years, both of them are likely to behave the same. Hence, taking the log captures this discrepancy in differences, and also helps to deal with outliers.

Afterwards, we then used either min-max scaling or standard scaling to normalise our data. For most data, we used min-max scaling, but data that would have a normal distribution such as the `internal_storage_capacity` feature would be standardised using the normal distribution. Normalising our data is important for distance based algorithms, as it helps with not prioritising one feature much more than the other. It can also lead to faster training

times for gradient based learning algorithms. Hence, we ended up using log transform and min-max or standard scaling on almost all numerical data.

Another useful feature type is to calculate the days between two dates. This is often important, as it can be used by the ML model to determine how important other features are as well. For example, a deactivation that happened over 10 years ago is far less meaningful than a deactivation that only happened last month. Each date difference was also log scaled and standard scaled, for reasons we've described before.

Finally, a very useful feature type was the use of indicator variables. Indicator variables are boolean features that draw attention to certain features. For example, we can use the `has_been_deactivated` to indicate whether a person has ever been deactivated before. Indicator features are useful for 'drawing attention' to certain important features, which saves your ML algorithm from having to learn the importance itself. According to Sci-Kit Learn's Random Forest Classifier, one of the most important indicators features was the `is_active` feature, which compares the last time a user has deactivated with the last time a user has reactivated to determine if the user is currently active.

In total, we obtain 77 features from this extraction. A full list of all the features can be found in the appendix.

3.2.3 Model training and tuning

Our process for model training and tuning was very similar to model 1, with the main change being that we used grid search to tune our models, and we also introduced a new classifier, LGBM. We considered LGBM because of it's superior training time, which would allow us to iterate parameters quickly, its comparatively better performance compared to other boosting algorithms, and the fact that it works well for large datasets. The specifics of which models we tested as well as the parameters used in grid search can be found in the results section. This ended up being the best model, and so was the one that we used for our final submission.

4 Results

Our first model was a preliminary model, so we simply wanted an idea of the performance of different classifiers. We achieved the following scores.

Model	$F_1 Score$
SVM	0.67
KNN	0.66
Decision Tree	0.69
Random Forests	0.78
AdaBoost	0.78
XGBoost	0.79

For our second attempt, we wanted to used hyperparameter tuning to increase the performance of our model as much as possible. Here are the parameters used in hyperparameter tuning:

Model	Grid Search Range
SVM	kernel=[rbf, linear], gamma=[10^{-3}, 10^{-4}]
KNN	neighbors=[1,...,500]
Random Forests	estimators=[100, 150, 200], max depth=[10, 20, 30] max feature=[auto, log2, sqrt], min samples leaf=[1, 2, 4] min samples split=[2, 5, 10], criterion=[gini, entropy]
Bagging	estimators=[10,...,60]
AdaBoost	learning rate=[10^{-4},10^{-2},0.1,1.0], base criterion=[gini, entropy] base estimator splitter=[best, random]
XGBoost	min child weight=[1,10], gamma=[0.5,1,2,5],max depth=[3, 5, None] colsample bytree=[0.6, 1.0], min samples leaf=[30,...,81]
GradientBoost	estimators=[391,...1000], max depth=[5,...,16] min sample split=[200,...,1000]
LightGBM	max bin=[633, 255], max depth=[20, 40, 50], iterations=[100, 150, ..., 500] learning rate=[0.05, 0.1], leaves=[100, 150, 200]

After hyperparameter tuning, these are the F_1 scores achieved for the each model, as well as the best parameters for each. The final stacking classifier consists of combining the best four models we found to perform a majority vote for prediction. It's also worth noting that model 2 received a score of 0.87 using RFC even without hyperparameter tuning.

Model	Parameters	F_1 score
SVM	kernel=rbf	0.591
KNN	neighbours=1	0.664
Bagging	estimator=50	0.879
Random Forests	estimators=200, max depth=30 max feature=auto, min samples leaf=2 min samples split=5, criterion=entropy	0.881
AdaBoost	learning rate=0.001, criterion=entropy, splitter=best, estimator=50,	0.882
XGBoost	base score=0.5, booster = gbtree, learning rate = 0.3 , max depth=6	0.884
GradientBoost	max depth=13, min samples split=600	0.897
LightGBM	learning rate=0.1, max bin=255 max depth=40 iterations=150, leaves=150	0.902
Stacking	classifier1=random forest, classifier2=bagging classifier3=AdaBoost, classifier4=GradientBoost	0.874

We submited LightGBM because it had the highest F_1 score on testing data of 0.902. However, on the final leaderboard, the performance dropped down to 0.868, which was lower than some of our previous submissions using Random Forest Classifier.

5 Discussion

Firstly, in terms of evaluation, the TracHack competition uses solely the F_1 score of the submitted evaluation predictions in order to compare submissions of teams. Since ultimately, the main goal of the project was to perform as well as possible in TracHack, we think that it is reasonable to treat to the F_1 score as the only metric used to evaluate our models.

Next, we compare the results of our two models. Our second model achieved a score of 0.87 on RFC without hyperpameter tuning, which was already better than any result obtained using our first model. This indicates that the feature extraction alone for model 2 was superior to the feature extraction from model 1. We believe that this was due to the fact that this is because we extracted features for model 2 with the problem domain in mind, whereas for

model 1, we blindly extracted as many features as possible, and then tried to narrow them down afterwards.

Based on the results, we see that distance based models such as SVM and KNN do not perform well. Both of these models rely on distance, and hence suffer from the curse of dimensionality due to the large size of our feature set. Hence other models based on decision trees and ensemble learning tended to outperform them.

Furthermore, out of all of these, Gradient Boost and Light GBM performed the best out of all other models. Boosting is known to produce high accuracy results, and also we effectively deal with it's weaknesses. The disadvantages of boosting is that it does not respond to outliers, and that they are difficult to interpret. However, we remove outliers during the feature extraction stage, and also interpretability is not important to us, only the results.

In terms of future improvement, one aspect is to increase the informativeness of the extracted features. We can achieve this by exploring more methods to handle missing values such as single imputation by expectation maximization, as described in [1] or multiple imputation [3], which uses multiple imputations in order to determine how to fill in missing values. Another possible way to improve prediction accuracy is to consider more complex models such as deep neural networks in model selection phase. Finally, we note the discrepancy between our evaluated result (0.902) and the actual result on the final evaluation data (0.868). It's certainly possible that we over-fitted on our training data without realising, so limiting the maximum depth or iterations of our LightGBM might have prevented this from occurring.

6 Conclusion

In conclusion, our team was given the task of predicting whether users would want to upgrade their device based on data collected by TracFone. Our final F_1 score achieved during testing was 0.902 and our final score achieved in competition was 0.868. During the course of the competition, we wrote two different models. The first one was a preliminary model aimed at understanding the dataset and the classifiers required, and the second one built on the previous by taking into account the problem domain, which improved our results.

For this problem, we observed that good feature selection was vital in improving the performance of our model, and that boosting algorithms were superior in working with this type of problem. In the future, it would perhaps be wise to work with the problem domain in mind right from the beginning, and to also take into account computation time, as that was one of the limiting factors in our work.

7 References

[1] Alan C Acock. "Working with missing values". In: *Journal of Marriage and family* 67.4 (2005), pp. 1012–1028.

[2] CableFree. "LTE UE Category & Class Definitions". In: (). URL: `https://www.cablefree.net/wirelesstechnology/4glte/lte-ue-category-class-definitions/`.

[3] Patrick Royston. "Multiple imputation of missing values". In: *The Stata Journal* 4.3 (2004), pp. 227–241.

8 Appendix

Here is a list of the 77 features used in our final submission. We add additional notes if necessary.

8.1 Customer Info

- `carrier_1-3` - the carrier used, one hot encoded.
- `days_to_redemption` - the number of days between `first_activation_date` and `remdemption_date`, logged and min-max scaled.
- `plan_1-4`, `plan_other` - the plan, one hot encoded

8.2 Phone Info

- `cpu_cores` - number of cpu cores, minmax scaled.
- `expandable_storage` - 1 or 0 if it has storage
- `internal_storage_capacity` - logged and standard scaled
- `lte`
- `lte_advanced`
- `total_ram` - logged and standard scaled
- `touch_screen`
- `wi_fi`

- `has_phone` - whether the persons phone info is filled in or missing
- `device_type` - one hot encoded based on the type of device
- `os_Android`, `os_Linux`, `os_iOS` one hot encoding for the 3 most popular operating systems
- `lte_downlink`, `lte_uplink` - downlink and uplink based on the lte category [2]
- `years_since_release` - difference in years between `date_observed` and `year_released`, logged and minmax scaled.

8.3 Redemptions

- `num_redemptions` - number of redemptions a user has made, logged and minmax scaled
- `num_one_offs` - number of redemptions below $15, logged and minmax scaled
- `total_spent` - total amount paid in redemptions, logged and minmax scaled
- `average_monthly_spend` - total paid divided the first and last month redemptions have been made, logged and minmax scaled
- `monthly_plan_cost` - median on redemptions made above, $15, logged and minmax scaled

8.4 Deactivations

- `has_deactivated` - whether a user has deactivated before
- `reason_PASTDUE`, `reason_UPGRADE`, ... - reasons why the person has deactivated. A person may have deactivated multiple times with different reasons. Reasons observed are: [”PASTDUE”, ”UPGRADE”, ”DEVICE CHANGE INQUIRY”, ”STOLEN”, ”DEFECTIVE”, ”ACTIVE UPGRADE”, ”REFURBISHED”, ”NO NEED OF PHONE”, ”DEVICERETURN”]
- `num_deactivations` - number of times a user has deactivated, logged, minmax scaled.

8.5 Reactivations

- `is_active` - compares the last date of reactivation and last date of deactivation, and checks to see if the user is currently active. If no deactivations at all, then the user is also active
- `num_reactivations`
- `days_since_reactivation` - number of days that have passed since the most recent reactivation, logged and minmax scaled.

8.6 Suspensions

- `num_suspensions` - logged and minmax scaled
- `has_been_suspended`
- `months_since_reactivation`
- `average_suspension_length` - the average time a person is suspended. 0 if they have never been suspended before.

8.7 Network Usage Domestic

- `sms_in_count`, `sms_out_count`, `sms_total_count` - combines SMS and MMS data together. Logged and minmax scaled.
- `over_15_SMS` - over 15 total SMS messages
- `hotspot_gb` - hotspot data converted to gigabytes
- `gb_5g` - 5g data converted to gigabytes
- `total_gb` - total data converted to gigabytes
- `voice_in_count`, `voice_out_count`, `voice_total_count`
- `voice_min_in`, `voice_min_out`, `voice_min_total`
- `over_5_voice` - over 5 calls made
- `average_call_length` - total minutes of voice calls divided by total number of calls

- num_network_usages - number of times a person used the network, logged and minmax scaled.

8.8 LRP Points

- total_quantity
- has_earned_points - a measure of whether the person is engaging with the system or not
- total_spent_lrp - difference between the total spent and the points already earned
- has_spent - whether the person has spent
- days_since_update - stores how recently the person has engaged with the LRP system.

8.9 LRP Enrollment

- num_enrols
- has_enrolled
- has_enrolled_over_twice
- months_since_enrolled