Real-life Problem #1

1. Task

We are given a dataset with number of cash withdrawals from 47 different ATMs of a bank using the information given about each ATM and the withdrawal date. A data entry has these information in order:

(1) ATM's identification number *id*
(2) ATM's geographical region $r \in \{1, 2, ..., 25\}$
(3) Transaction day d as a number.
(4) Transaction month m as a number.
(5) Transaction year y as a number.
(6) Transaction type $t \in \{1, 2\}$:
   - 1: Card present
   - 2: No card present

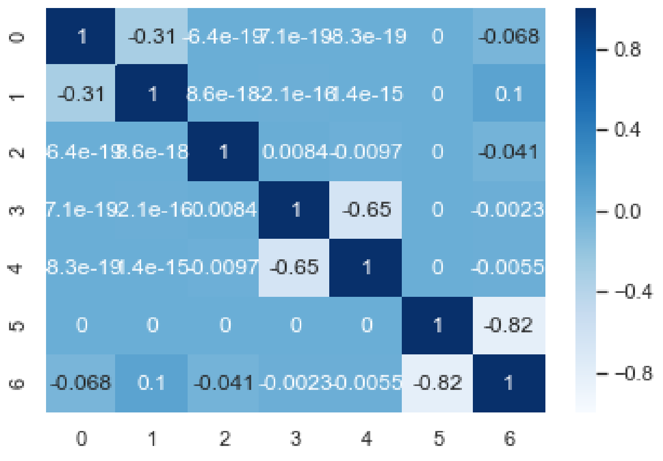
(7) Transaction count c which is the number of cash withdrawals performed from the ATM with the given information. This is the output, and this is what we will try to predict.

We are given the freedom of trying any method, as long as it works of course. Root mean squared error and mean absolute error will be our focus on evaluating our methods. Given the fact that our output variable is numeric, this is a regression problem with more than one feature.

2. Implementation

2.1. **Feature Selection.** First of all, on a problem like this it is important to check the features and correlations among eachother, most importantly the correlation between the feature and target output. In figure 1 we have

FIGURE 1. Pearson Correlation Matrix



the Pearson Correlation Matrix. In that figure, 0 is *id*, 1 is r, 2 is d, 3 is m, 4 is y, 5 is t and 6 is c.

2.1.1. *Removing month and year features.* We can observe that month and year has very low correlation to the output variable. I have tried each method after removing month and year from the feature set, however, it performed worse. Thinking about the problem at hand, one may think that year does not have too much effect on the cash withdrawal unless the country is going through some ecomonic event at that year, but month would make a difference. I would like to say the low correlation surprised me at this. In the end, I have decided not to remove them from the feature set.

2.1.2. *Converting year, month and day to hours overall.* As another approach, I have tried to combine these 3 features into one by converting them to date and then calculating the hour difference between that date and epoch which is 1.1.1970. The results had more error, so I decided not to use this approach.

2.1.3. *Removing id feature.* Though id seems to have a correlation with the output variable, logically I found this to be slightly out of place. People dont choose a specific ATM usually, they choose whatever ATM is available and that is mostly defined by the region itself. Motivated by this, I have removed the id field and got better results immediately.

2.1.4. *Making region parameter dichotomous.* A known approach in categorical variables for regression is to convert the variable to many dichotomous (binary) variables. In the case of region, we have 31 different regions, so we will create 31 variables instead, where for example if a data row i has region $r = 4$ then the corresponding variable will be 1, and every other region variable will be 0. In another words, we will do One-Hot-Encoding here. By itself, this performed worse, but combined with removing the id form features, this worked better overall.

In conclusion, I have converted region parameters to their respective dichotomous variables and removed the id feature. In the following subsections, I will describe my results with several methods, note that I am showing the results with the features removed. In my results, I will do 5-fold cross-validation and show the average errors, minimum and maximum errors from that cross-validation. I will also include the model training time (in seconds) on average.

2.2. **Decision Tree Regressor.** Due to its categorical independent variable nature, the first thing I tried was to try Decision Tree for regression. I have used *sklearn.tree.DecisionTreeRegressor* from the *sklearn* package. The results are great! Good training time with kind of good error rates. Immediately, this gave me the motivation to research into decision tree regression techniques, and as a result I have 3 decision tree based methods in this task: Random Forest, XGBoost and LightGBM.

Decision Tree	AVG	MIN	MAX
RMSE	24.92479	17.49900	32.08451
MAE	14.43111	10.42612	18.28205
Time (sec): 0.27629			

Table 1. Decision Tree Regressor

Multilayer Perceptron	AVG	MIN	MAX
RMSE	26.69785	21.82431	34.53084
MAE	15.80408	12.68853	19.84406
Time (sec): 8.82511			

Table 2. Multilayer Perceptron Regressor

XGBoost	AVG	MIN	MAX
RMSE	26.12721	21.64978	32.44767
MAE	15.59602	12.76651	19.27959
Time (sec): 3.92292			

Table 3. XGBoost Regressor

2.3. **Multilayer Perceptron Regressor.** I still wanted to try how a multilayer perceptron would perform in this task. My hyper parameters are:

- Learning rate 0.01
- 2 Hidden Layers, each of size 50.
- Alpha 0.001

I have used *sklearn.neural_network.MLPRegressor* from *sklearn* package. The first thing that catches the eye is the huge training time, which is expected given the nature of this algorithm. It also has the highest RMSE among the methods I have tried. So this is definetly not a good choice.

2.4. **XGBoost Regressor.** XGBoost stands for eXtreme Gradient Boosting. In general, Gradient Boosting is the practice of using an ensemble of trees (each having depth usually between 8 and 32) to predict the output variable. New models are created to deal with errors of existing models, and in the end you have an ensemble of models which you use together. XGBoost is one of the algorithms in this category of "Boosted Gradients". It is known for being fast and efficient. I have used *xgboost* package. Though it is known for being fast, we have a training time of 3.92292 on average, which we will is the slowest among the 4 methods I have tried, excluding multilayer perceptrons.

2.5. **Random Forest Regressor.** Random Forest is also an ensemble technique, but unlike boosters this one uses bagging. Bagging is when we use small samples of data to train and create many models (trees) and aggregate them in the end, hence the name: Forest. I have used *sklearn.ensemble.RandomForestRegressor*

Random Forest	AVG	MIN	MAX
RMSE	25.52983	23.22682	29.51543
MAE	15.74735	13.79042	17.89368
Time (sec): 0.16158			

LightGBM	AVG	MIN	MAX
RMSE	24.64834	18.68125	32.22125
MAE	14.45761	10.89574	18.37656
Time (sec): 0.23766			

by *sklearn* package. Random Forest achieved the fastest training time, and a moderatly good error rate. However, we value the error rate more in this task, so this is still not the best choice.

2.6. **LightGBM Regressor.** Finally we have LightGBM, another Gradient Boosting framework, made by Microsoft. It is fast and accurate! I have used *lightgbm* package. Decision Tree Regressor has a smaller MAE average error, but LightGBM has an even smaller RMSE error compared to Decision Tree Regressor, as well as slightly faster training time. As a result, I have decided to use LightGBM for this regression problem. Some hyper parameters I have are:

- $num\_leaves = 34$.
- $learning\_rate = 0.2$
- *boosting_type* as “gbdt” which stands for Gradient Boosting Decision Tree.

Note that one boosting type parameter was “rf” which stands for Random Forest, however it did not work so I used the Random Forest described in the previous subsection to test. The predictions stored in “test_predictions.csv” are the results of lightGBM trained on the whole dataset.