Final Case Study, Units 14/15
Daniel Davieau, Lei Jiang, Kristen Rollins
14 April 2020

Introduction

In the real world, it is often the case that data is dirty, missing documentation, or anonymized. In this case study we experienced this to a great extent, working with a completely "unknown" dataset. We were not given any metadata, and all the feature names were unidentifiable. Even without context, however, much information can still be extracted from datasets through exploratory data analysis, and models can be built to classify even an unknown target column. Even though we could not interpret our models for this mysterious dataset, we could still achieve the desired goal of minimizing cost for a stakeholder.

Background

For this study, we were given a completely unknown dataset that had 50 features labeled x0, x1, ... x49, from which we could classify a binary target column labeled 'y.' Our stated goal was to minimize the dollar cost on this dataset, given the cost matrix shown in Table 1. In this matrix we see that correctly predicting the y label does not net any cost or gain. However, the dollar penalty for false positives is $10, and the cost of false negatives is very heavy at $500. The total cost can be written out explicitly as:

Total Cost = $500*Count of False Negatives + $10*Count of False Positives

Such an extreme penalty might be in place for a situation like predicting cases of COVID-19, where it would be very bad to wrongly tell someone they do not have the virus, when in fact they do, as they will not get treated quickly and they will continue to spread the virus. On the other hand, it is unfavorable but not detrimental to tell someone they have the virus when they don't, as it will just scare them a little and make them self-quarantine. While we do not know the actual context of this dataset, we can still relate the outcomes to other similar situations we can identify.

Table 1. Cost Matrix for Unknown Dataset

		Predicted Label	
		0	1
True Label	0	$0	$10
	1	$500	$0

Data Cleaning and EDA

Before we could start modeling, it was important to explore the data and its characteristics and get an idea of what was influencing the target column. The first thing we noted was that the full dataset had 160,000 rows. The next thing we observed was that the 50 feature columns had between 0.01% and 0.03% missing values. Since this was such a small amount, we decided to remove the rows with any missing values, which resulted in 158,392 remaining rows for us to work with. The next thing we looked for was the distribution of the target variable y. We found that class 0 (negative) had 94,846 observations, while class 1 (positive) had 63,546 observations, as visualized in Figure 1. This is approximately a 60/40% split, which is relatively balanced. Based on this observation, when we began modeling and had to choose between sampling methods, we did not use any sort of stratification because the dataset was not highly imbalanced.

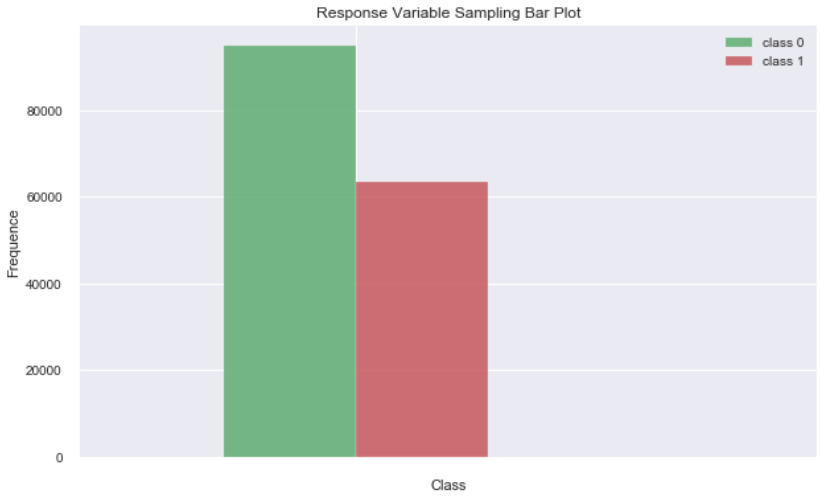

Figure 1. Target Class Distribution

Next we dove into examining relationships between the variables. We created a correlation matrix containing all of the columns, found in Figure 2. Because there are so many variables, it is difficult to distinguish them, but we highlighted the column showing correlations of the features with the target column y. From this we noticed that x20, x23, x40, and x49 correlated to y relatively higher than other variables, and x12, x38, and x42 are moderately correlated to the target, so these could be important features as we create our models. We later confirmed these features to be important in one of our Random Forest models, shown in the feature importance results in Figure 6 in the Appendix. In addition, we found that two sets of variables were perfectly correlated to each other, those being x2 and x6, as well as x38 and x41, as seen in Figure 5 in the Appendix. These redundant features should be removed for models that are sensitive to multicollinearity, such as logistic regression. While we observed this obvious case of multicollinearity, we did not investigate other possible issues with feature correlation because there are so many possible combinations of the 50+ features (i.e. 67 after one-hot encoding the categorical variables). Therefore we decided to pursue tree-based models

such as Random Forest or XGBoost, which can automatically perform feature selection and do not have such assumptions about multicollinearity.

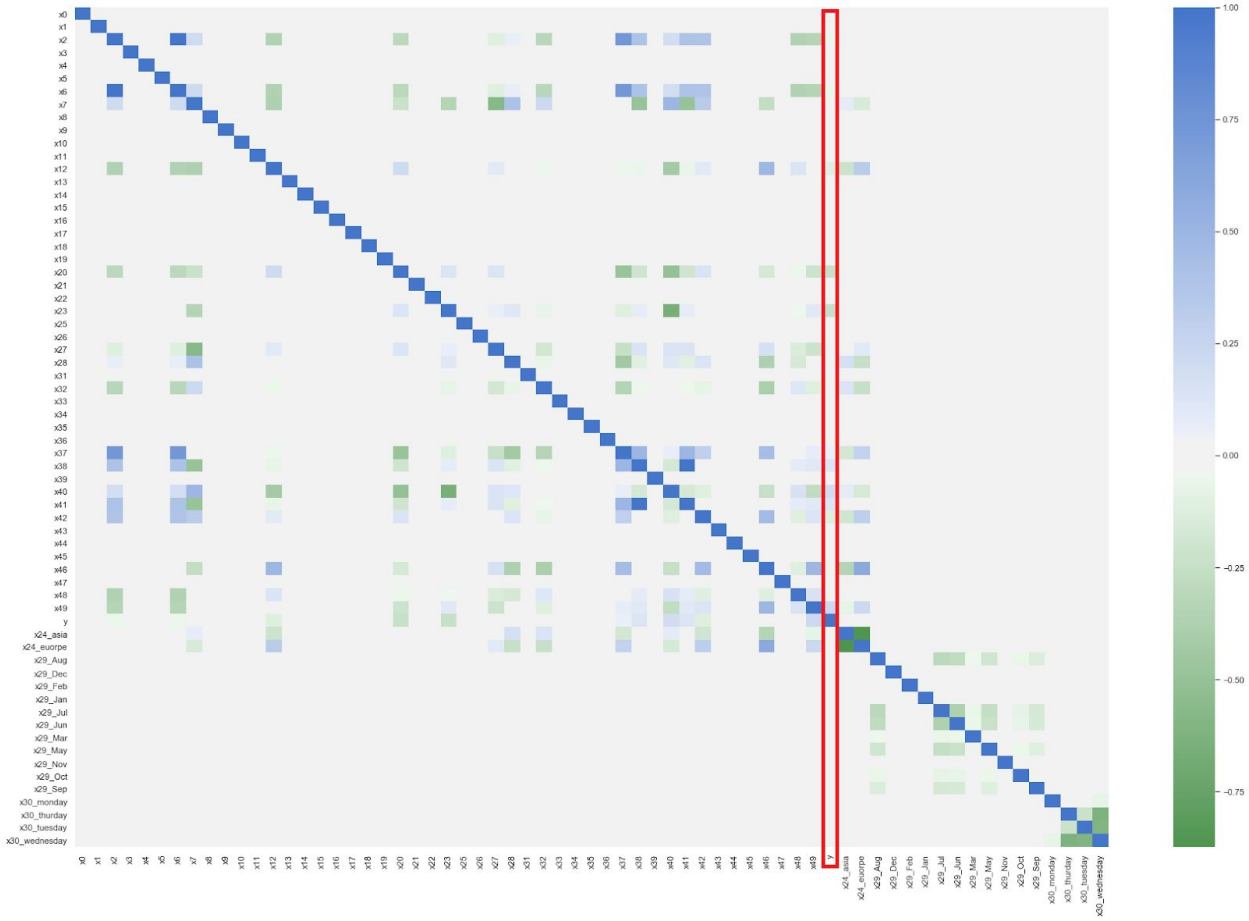

Figure 2. Pairwise Correlation Matrix

Modeling

After performing this exploratory data analysis, we could begin modeling with the goal of minimizing cost. Even though we removed missing values from this dataset, we tried to choose models that would handle missing data, since we do not know what future data will look like. Also, as discussed above, because there are so many features and thus we could not thoroughly investigate multicollinearity, we wanted to choose models that are able to automatically perform feature selection. Therefore, we decided to go with tree-based models such as Random Forest or XGBoost, as they are able to handle missing data as well as perform feature selection.

We went down multiple avenues for model creation. In our initial approach, due to computing resources we utilized only a 20% sample of the full dataset (just under 32,000 points) for training and testing, to see how models performed. In this case we tried three different models: XGBoost, Random Forest, and a GBM (from the LightGBM framework). For each of these types of models, we performed a grid search on parameters, while optimizing for recall on

a 20% test set. We chose to optimize recall for this situation because we are primarily needing to minimize false negatives since the penalty for false negatives is so high. The formula for recall is given below. In this formula, if False Negatives are very low, then the recall rate will be very close to 1.

Recall = True Positives / (True Positives + False Negatives)

The next approach we took was to utilize the full dataset, to try to tune just a Random Forest model more thoroughly. On this path, we also performed a grid search to try to optimize the parameters of the RF algorithm itself. However, after arriving at the best RF parameters, we also tried optimizing the threshold by which classes are assigned. For binary classification, algorithms normally do not just output discrete 0 or 1 values, but rather they provide continuous values ranging from 0 to 1. The decision threshold decides where the cutoff is, i.e. when to assign a 0 versus a 1. The threshold is often chosen to be 0.5 as the midpoint between 0 and 1, but it can theoretically be assigned any value. This is important for us because we can treat this threshold as another parameter to tune in order to minimize cost according to the cost matrix given. For example, in Figure 3 we plotted the precision and recall rates for our final RF model, for decision thresholds ranging from 0 to 1. We noted that if we were to make the decision threshold be 0, then we would actually have perfect recall (i.e. we would always predict the positive class). However, this is probably not desirable because there is still a penalty for false positives ($10), that may outweigh the money saved from predicting the positive class every time.

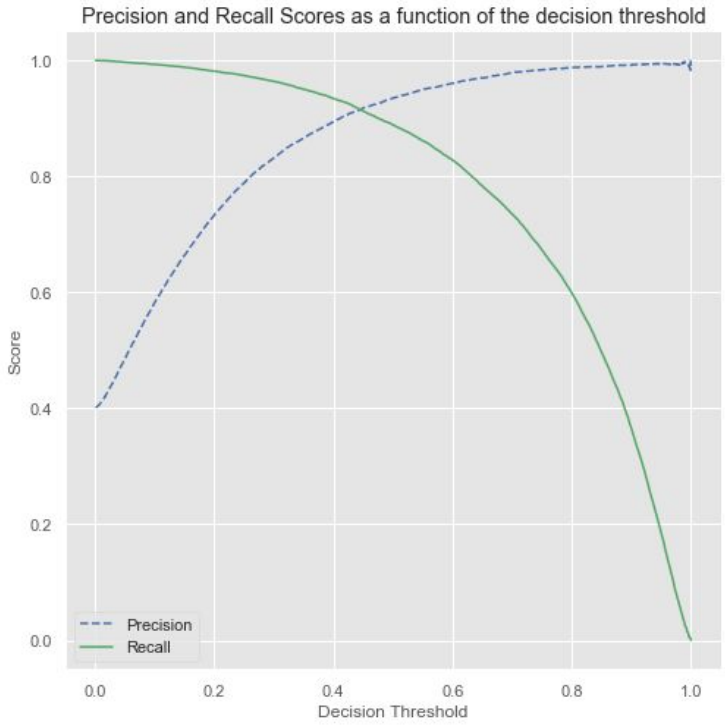

Figure 3. Precision and Recall based on Decision Threshold

To balance between the penalties of false negatives and false positives, we actually chose the decision threshold for this RF classifier based on algebra. We knew from previous experience that in this cost matrix situation, the cost is minimized when the dollar cost from the false negatives are equal to the dollar cost of the false positives. That is, we wanted to solve for $500*FN = $10*FP. Therefore, since there is a relationship between false negatives and false positives, we used trial and error of different thresholds until we found FN and FP quantities that got the closest to satisfying that equation. For this RF model we found that the lowest cost occurred when the decision threshold was 0.14. There were then 172 false negatives and 8602 false positives, which resulted in costs of $86,000 and $86,020 respectively.

Finally, we re-attempted the XGBoost and GBM models using the full dataset as well. After performing grid searches we did not optimize the decision threshold as precisely as in the RF model, but tried out several decision thresholds and chose the one resulting in the lowest cost. After doing this we saw significant performance gains compared to the models only using 20% of the data.

Results

To be able to compare our models on the same scale, we computed the average cost per observation for our models. We did this because in some cases, we tested our models on test sets that were different sizes or used different sampling schemes. In this situation, even if a model is worse it could have lower total cost because it was tested on a smaller dataset. Therefore we had to normalize these costs to the same scale. We chose to normalize to the average cost per single observation because this value is easily interpretable, and it can also be used to estimate the cost of future datasets. We would just multiply the average cost by the number of observations in the new dataset.

To get this average cost we first had to calculate the full cost of the test set predictions. As an example, Table 2 depicts the confusion matrix for our RF model. To compute the full cost, we can do element-wise multiplication between this confusion matrix and the cost matrix in Table 1, and sum those values. In this case the cost comes out to be 172*$500 + 8602*$10 = $172,020. To get the total number of observations that were predicted, we can simply sum up all the numbers in the confusion matrix in Table 2. For this model there were 15349+8602+172+15877 = 40,000 predictions made. Therefore the average cost per observation for this model is $172020/40000 = $4.30.

Table 2. Confusion Matrix for Random Forest Model

		Predicted Label	
		0	1
True Label	0	15349	8602
	1	172	15877

Figure 4 depicts the average cost per observation, calculated in a similar manner as above, for the 6 models we created. These were XGBoost, Random Forest, and GBM, each utilizing either 20% or 100% of the dataset. Our first finding was that utilizing the full dataset caused significant cost reduction for most models (e.g. $53.74 reduced to $7.24 for the XGBoost model). We expected this result, but not to the extent that we observed. Therefore for this dataset we would recommend training on as much data as possible in order to optimize performance. We next compared the three models that utilized all of the data. We found that our XGBoost optimized with recall produced an average cost of $7.24 per prediction, our GBM model incurred an average cost of $4.57 per observation, and our RF with an optimized threshold costs an average of $4.30 per observation. This $4.30 cost from the RF model was the smallest amount we achieved for this anonymized dataset. We did not expect any models to perform perfectly, so some cost is expected to be incurred when making many predictions on a dataset like this. We believe the best result was achieved with this RF model because of our precise choice of a decision threshold. Being able to automate this decision process could improve the performance of the other models as well.

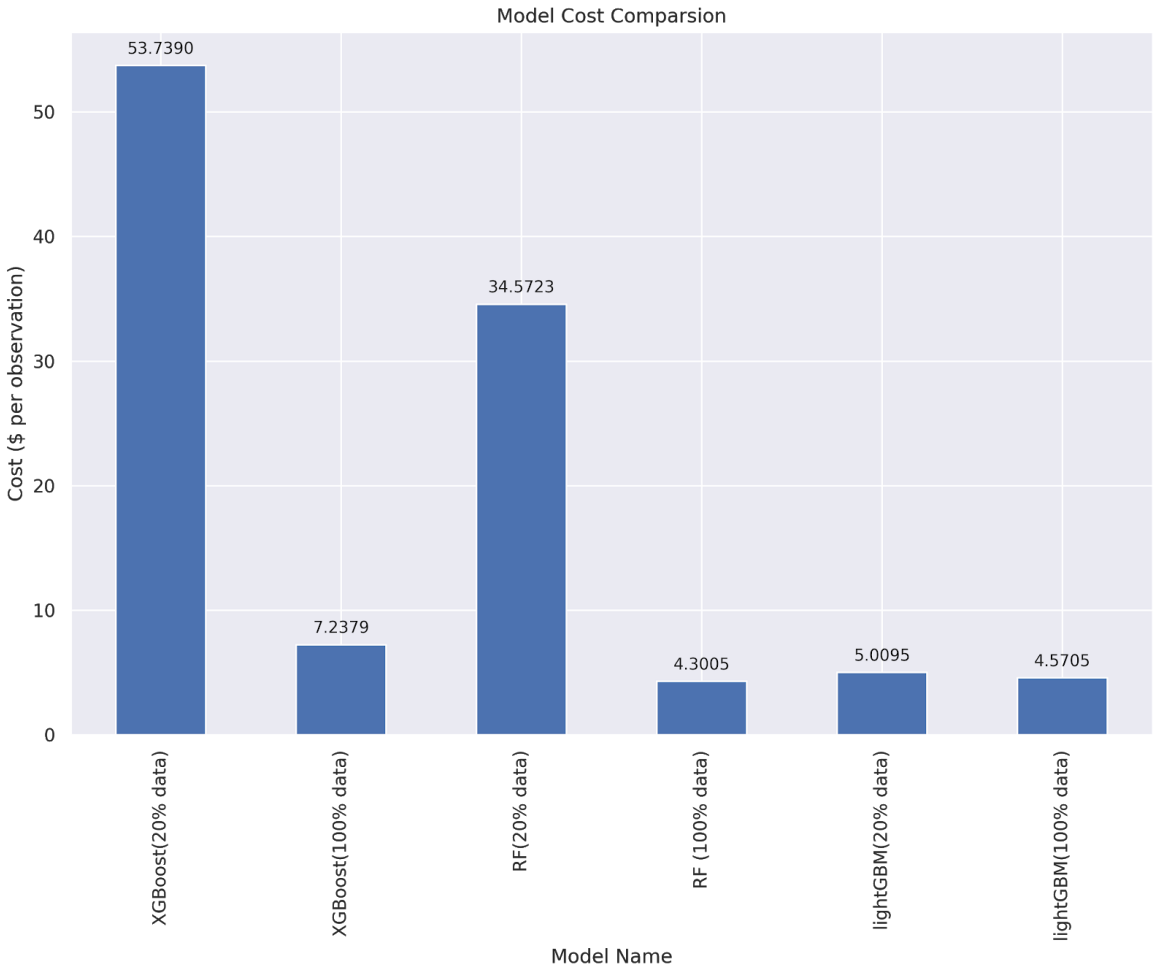

Figure 4. Model Cost Comparison

Conclusions

While it is obviously helpful to have context for a dataset, we found that it is not impossible to achieve desired outcomes on an unknown dataset. We chose to focus our creation of models to tree-based models, because they are able to handle missing data as well as automate feature selection, which are helpful characteristics on this large, unknown dataset. However, with some extra effort other types of models could be tested out as well. We could consider other classifiers such as logistic regression, SVM, or an MLP, but care should be taken to make sure that missing values are handled appropriately for a future dataset, or that assumptions of the model are satisfied. For example, feature engineering may need to be done manually before running the data through a logistic classifier, since it is sensitive to multicollinearity. Whichever model is used, we found that the decision threshold used to assign the 0 and 1 classes is very important when minimizing cost. If we can find a threshold where the cost due to false negatives and the cost caused by false positives are approximately equal, in this way we have balanced the penalties from both types of misclassifications and minimized the cost.

References

- https://towardsdatascience.com/fine-tuning-a-classifier-in-scikit-learn-66e048c21e65
- https://lightgbm.readthedocs.io/en/latest/
- https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html#sphx-glr-auto-examples-compose-plot-column-transformer-mixed-types-py

Appendix

Supplemental Figures

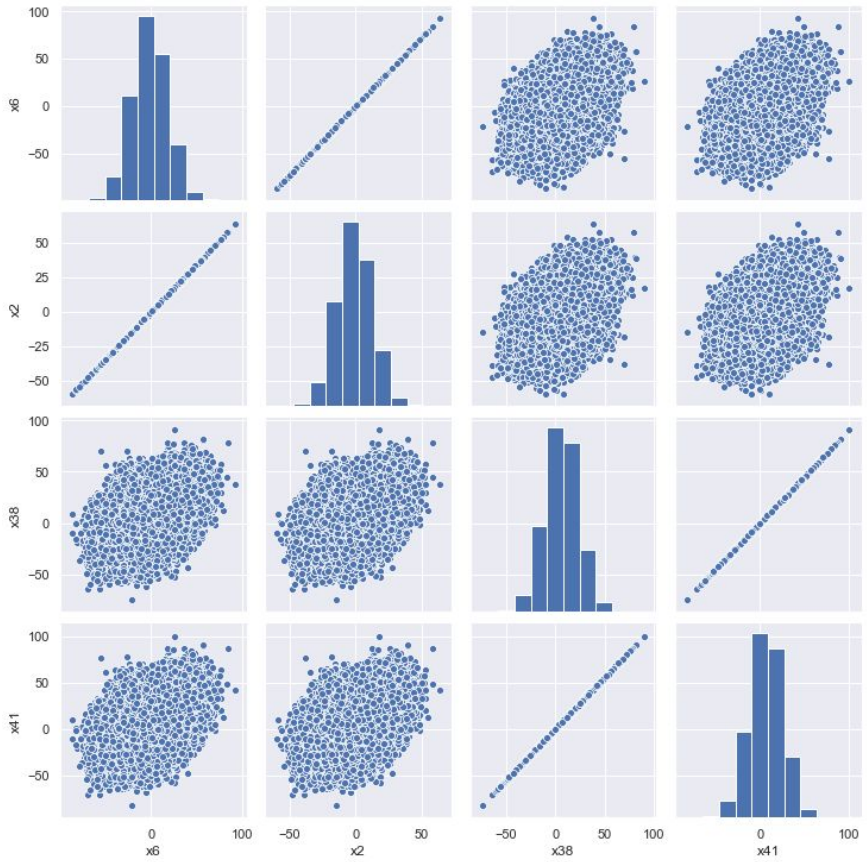

Figure 5. Plots showing perfect correlation between two sets of features (x2/x6 and x38/x41)

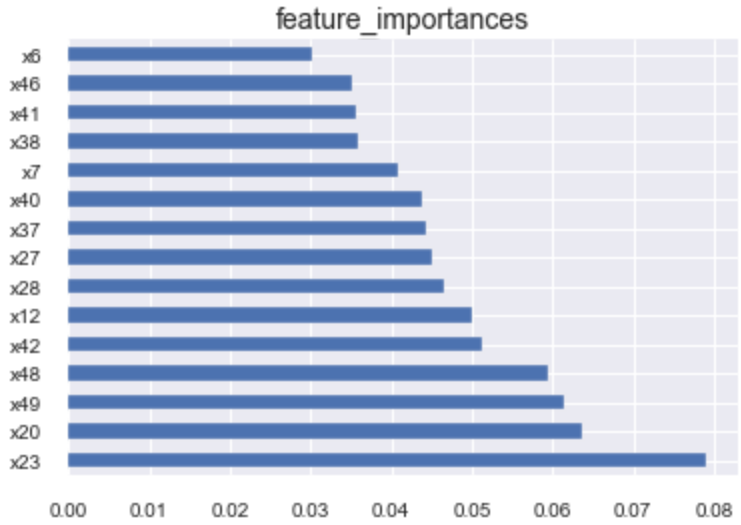

Figure 6. Feature importances from Random Forest model

Codebase

Our codebase has been included in a file called lJiang_kRollins_dDavieauQTWFinal_Code.zip. The files in this zip folder include:

- lJiang_kRollins_dDavieauQTWFinal.ipynb - notebook with primary RF model
- Colab_lJiang_kRollins_dDavieauQTWFinalUpdate2.ipynb - notebook exploring many other models

Assignment

- Get the data from https://smu.box.com/s/k9x192jxm39enjw2wx8ouw2kopx33l32
- Classify the Column Labeled ‘y’
- Your features are labeled x0 through x49
- Deliverable:
    - I want you to minimize my dollar cost on an UNKNOWN dataset
    - Each False Positive costs me $10
    - Each False Negative costs me $500
    - True Positives and True Negatives cost me $0