CUNY MSDS – DATA622

Homework 2

**Introduction**

For this assignment, I chose a dataset of simulated online payments, labelled to identify fraudulent transactions. I sourced the data from Kaggle [1], but the data ultimately comes from a 2016 paper from Lopez-Rojas et al [2] that introduced a payment simulator named PaySim.

The dataset provides ~6.35 million observations and includes nine features and one label:

1. step: represents a unit of time where 1 step equals 1 hour
2. type: type of online transaction
3. amount: the amount of the transaction
4. nameOrig: customer starting the transaction
5. oldbalanceOrg: balance before the transaction
6. newbalanceOrig: balance after the transaction
7. nameDest: recipient of the transaction
8. oldbalanceDest: initial balance of recipient before the transaction
9. newbalanceDest: the new balance of recipient after the transaction
10. isFraud: fraud transaction

The target variable (“isFraud”) is a binary label indicating whether a transaction is fraudulent (1) or not (0). The dataset is heavily imbalanced, with over 99% of the transactions being non-fraudulent.

The overall approach for this assignment is as follows:

* Exploratory Data Analysis
* Preprocessing
* Baseline Training
* Data Adjustment and Re-Training
* Hyperparameter Tuning
* Final Training
* Conclusions

The goal of this analysis is to (i) identify the differences between various tree-based methods for handling classification problems, with particular attention on the bias-variance trade-off; (ii) highlight impacts of changes to the underlying data, focusing on sensitivity versus robustness; and (iii) identify best-fit models for this data, in terms of predictive accuracy.

Code for all the following analyses is saved here: https://github.com/kac624/cuny/tree/main/D622.

**Exploratory Data Analysis and Feature Selection**

For each column, I analyzed the datatype and the number of unique values, summarized in Figure 1.

*Figure 1: Feature Data Types and Unique Values*

*A screenshot of a computer

Description automatically generated*

Based on this, I concluded that “amount”, “oldbalanceOrg”, “newbalanceOrig”, “oldbalanceDest”, and “newbalanceDest” are continuous, numerical features. While tree-based methods are typically robust to varied scales, I typically apply some kind of standardization with machine learning, and these features appeared to be reasonable candidates for such a transformation. Regarding the “step” feature, however, I noted that this was *not* continuous, instead representing discrete intervals of time. As such, I decided it was best left as is, using its original scale.

This left the categorical features: “type”, “nameOrig” and “nameDest”. While “type” had a reasonable number of categories (only five), the two name-related features exhibited very high cardinality. This condition was expected, given the large number of potential payers / payees who might be involved in an online transaction. So, I decided to keep only “type.” The two name-related features may hold some valuable information, but they would likely require significant analysis and natural language processing.

Finally, the “isFlaggedFraud” feature appeared to be a duplicate of the target label, and the dataset’s documentation provided no details on it. So, I decided to drop it.

I examined the distributions of each variable through histograms (or bar charts for categorical features). Figure 2 shows the distribution for variables across the entire dataset, whereas Figure 3 shows the distribution for observations associated with the positive class (i.e. fraudulent transactions).

*Figure 2: Distribution of Variables – All Data*

*A screenshot of a graph

Description automatically generated*

*Figure 3: Distribution of Variables – Positive Class Observations Only*

*A screenshot of a graph

Description automatically generated*

These plots reveal a few key observations. First, the data is heavily imbalanced, with only 0.13% of the transactions related to the positive class. This imbalance highlighted the need for stratified sampling when splitting training and validation data. Second, most of the numerical variables are heavily right skewed. As noted previously, tree-based methods are typically robust to such conditions, but this reinforces the potential value-add of scaling. Third, the positive class is associated with only two “type” categories: CASH\_OUT and TRANSFER. We can therefore expect the “type” feature to be important for modeling.

**Preprocessing**

Based on EDA, I applied the following procedure to process data.

* One-hot encode the only categorical variable (“type”)
* Apply sklearn’s StandardScaler to continuous numerical variables (“amount”, “oldbalanceOrg”, “newbalanceOrig”, “oldbalanceDest” and “newbalanceDest”)
* Apply stratified sampling to split the data into training, validation and testing subsets using a 80% / 10% / 10% split.

On this final point, such a large dataset might allow for an even larger proportion of data under the training subset (e.g. 90%). However, given the heavy imbalance, I wanted to ensure that both validation and testing subsets had a sufficient number of positive class observations. So, I kept the training subset at 80%.

**Baseline Training**

In terms of modeling, I chose three algorithms:

* Decision Tree Classifier from sklearn (“DT”);
* Random Forest Classifier from sklearn (“RF”); and
* eXtreme Gradient Boosted Trees through the XGBoost implementation (“XGB”).

I trained a baseline model using all default hyperparameters. Below is a summary of the results across training and validation subsets.

*Figure 4: Baseline Model Results*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | DT | RF | XGB |
| **Train** | Accuracy | 1.00 | 1.00 | 1.00 |
| Precision | 1.00 | 1.00 | 0.97 |
| Recall | 1.00 | 1.00 | 0.80 |
| F-1 Score | 1.00 | 1.00 | 0.88 |
| **Validation** | Accuracy | 1.00 | 1.00 | 1.00 |
| Precision | 0.91 | 0.96 | 0.95 |
| Recall | 0.91 | 0.79 | 0.77 |
| F-1 Score | 0.91 | 0.86 | 0.85 |

The baseline models have relatively strong results, providing a near-perfect fit (allowing for some rounding) to training data (with the exception of XGBoost). It is important to note that accuracy is not a very helpful metric in this case, because of the extreme imbalance in the dataset. Precision, Recall and F-1 are much more insightful in terms of evaluating how well the model performs in identifying cases of fraud.

We do see some evidence of over-fitting (i.e. high variance), as the models perform less well on the validation subset. The exception is again XGBoost, which has very similar performance across both subsets, indicating slightly higher bias, but lower variance.

Surprisingly, the single decision tree appears to provide the best fit. One like driver of this situation is the size of the data. When visualizing the fitted tree (see Figure 4), it becomes clear that the amount of training data enables a very complex tree with repeated, granular splits upon the same features. So, the decision tree comes to resemble the ensemble approaches, in that it provides limited explainability, but strong predictive accuracy.

*Figure 5: Baseline Decision Tree Visualization*

***A group of people with different colored lines

Description automatically generated with medium confidence***

**Data Adjustment and Re-Training**

To assess the relative sensitivity / robustness of these modeling frameworks, I cut down the size of the training data and re-trained the models. Specifically, I took only 40% of the original training data, and kept the validation subset the same (for a more apples-to-apples comparison). Figure 6 shows the performance of these re-trained models.

*Figure 6: Re-Trained Model Results*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | DT | RF | XGB |
| **Train** | Accuracy | 1.00 | 1.00 | 1.00 |
| Precision | 1.00 | 1.00 | 0.98 |
| Recall | 1.00 | 1.00 | 0.81 |
| F-1 Score | 1.00 | 1.00 | 0.86 |
| **Validation** | Accuracy | 1.00 | 1.00 | 1.00 |
| Precision | 0.88 | 0.96 | 0.94 |
| Recall | 0.86 | 0.77 | 0.77 |
| F-1 Score | 0.87 | 0.86 | 0.84 |

With these re-trained models, we can highlight a few takeaways. First, we see that DT performance declined slightly, whereas RF and XGB performance remained mostly the same. This sheds some light on the central question of this assignment. That is, single decision trees are more sensitive to changes in underlying data, whereas ensemble methods are more robust. I expect that, if we were to shrink the size of the training data even further, the performance of the DT model would continue to decline, with less marked changes on the ensemble methods.

Another key takeaway from these re-trained models is the increasing difference in precision. While the DT model maintains the highest recall and F-1 score, the ensemble methods produce fewer false positive predictions.

Finally, even with only 40% of the training data, the DT model remains quite complex and difficult to interpret, as shown by the refit tree in Figure 7. It is important to note that, even with only 40% of the training data, the model has over 3mm observations to reference. So, with a smaller dataset, I might expect the tree to (i) be more interpretable, and (ii) be even more sensitive to changes in the data.

*Figure 7: Re-Trained Decision Tree Visualization*

*A network of small objects

Description automatically generated with low confidence*

**Hyperparameter Tuning**

Now that we’ve substantiated the DT model’s sensitivity and the ensemble models’ robustness, we can work to produce a final series of models. To do this, we need to find optimal hyperparameters. Given the volume of our data, I decided to skip cross-validation, and instead assess hyperparameters by comparing performance between training and validation subsets. The hyperparameters considered included:

* the loss criterion and max features considered for splits for DT and RF;
* the number of estimators for RF and XGB;
* various regularization hyperparameters for XGB; and
* the max depth for all models.

For full details on the grid of hyperparameters considered, refer to the tuning.py script on the abovementioned GitHub repository. Results for the top 10 combinations of hyperparameters (ranked by F-1) are also available in the Appendix of this document.

The tuning process revealed two key takeaways. First, the DT model appears less dependent on optimal hyperparameters. After hyperparameter tuning, performance remained largely the same for the DT model. The ensemble methods, however, improved considerably. While precision remained roughly the same, recall and F-1 improved by as much as 10%. This indicates that the ensemble methods require more fine-tuning to reach optimal performance.

Second, while the performance difference between RF and XGB remained small, the time it took to train became much more impactful. In particular, XGB proved much faster to train, and therefore more efficient to tune. I have noticed this trend more and more. Given the faster training time and comparable (and sometimes superior) performance of XGBoost over Random Forest, I’m finding fewer situations where Random Forest proves most useful.

**Final Training**

With more optimal hyperparameters, I re-trained the models one final time. This time, however, I used both the combined training and validation subsets for model fitting, then evaluated performance with the holdout test subset. Figure 8 summarizes the results.

*Figure 8: Final Model Performance on Holdout Test Data*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | DT | RF | XGB |
| **Train + Valid** | Accuracy | 1.00 | 1.00 | 1.00 |
| Precision | 0.99 | 1.00 | 0.97 |
| Recall | 0.97 | 1.00 | 0.80 |
| F-1 Score | 0.98 | 1.00 | 0.88 |
| **Test** | Accuracy | 1.00 | 1.00 | 1.00 |
| Precision | 0.92 | 0.93 | 0.92 |
| Recall | 0.89 | 0.91 | 0.81 |
| F-1 Score | 0.91 | 0.92 | 0.86 |

Final performance appears very close for all three models. Surprisingly, XGB appears to perform the worst on the holdout, and DT and RF have very similar performance. Given the similar performance of both models, and the similar levels of interpretability, the final decision may come down to efficiency, in which case DT might prove best.

**Conclusions**

This assignment highlights a few important conclusions.

* Single decision trees do indeed appear more sensitive to changes in training data, whereas ensemble methods appear more robust.
  + However, this difference can be mitigated with a sufficiently large training set.
  + In such cases, it appears single decision trees can rival the performance of ensemble methods, though I suspect this is largely dependent on the underlying data.
* In this particular case, the ensemble methods appear to provide higher precision, whereas the DT model provides better recall.
  + The optimal trade-off between precision and recall is not something we can determine quantitatively. Instead, it relates to the models’ usage and the business’s risk appetite.
  + Specifically, model users must determine if they prefer (a) to be more confident that predicted fraud is truly fraudulent (i.e. minimize false positives), or (b) to be more confident that no potential fraud is missed (i.e. minimize false negatives)
  + In the case of (a), users might choose the ensemble methods. In the case of (b), they may be better off with the DT model.
* In final training, however, DT and RF models appeared to provide the best performance, outperforming XGBoost.

**Appendix – Top 10 Hyperparameter Results for Each Model**

*Decision Tree*



*Random Forest*



*XGBoost*



**Sources**

[1] Roy, Rupak (2022). "Online Payments Fraud Detection Dataset." Sourced from https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset/discussion/330548.

[2] Lopez-Rojas, Edger Alonso; Axelsson, Stefan; and Elmir, Ahmed (2017). “PAYSIM: A financial mobile money simulator for fraud detection.” Sourced from https://www.researchgate.net/publication/313138956\_PAYSIM\_A\_FINANCIAL\_MOBILE\_MONEY\_SIMULATOR\_FOR\_FRAUD\_DETECTION.