STAT 536 - Case Study 5

Credit Card Fraud

Sam Lee, Paul Crowley

Introduction

Credit card fraud represents a significant financial burden, with estimated global losses reaching approximately $22 billion annually. In response, credit card companies employ advanced machine learning techniques to identify fraudulent transactions accurately. This report examines a dataset containing around 300,000 credit card transactions, of which only 492 are fraudulent, translating to a prevalence of approximately 0.1%. Given the rare nature of fraudulent events, our objective is to develop a high-performing model that can detect fraud with precision, minimizing both financial losses and false positives, which could otherwise disrupt legitimate users.

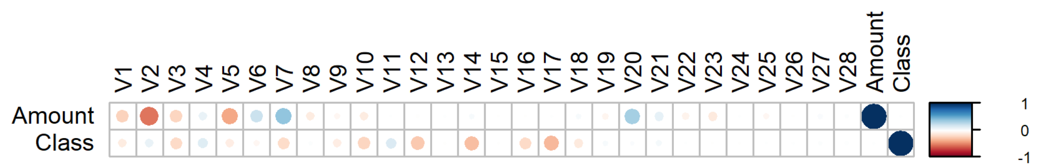

Figure 1: Correlation matrix between factors used in model specification. Note that all partial components (V1-V28) are, by construction, uncorrelated with each other.

Several potential challenges in the data may impact our analysis. Given the nature of the data, we will estimate a series of non-parametric binary classification models to predict whether a transcaction is fraudulent. The dataset's extreme class imbalance raises concerns about model performance, especially with a tendency to misclassify rare fraudulent transactions. Additionally, while principal components enable dimensionality reduction, they also reduce interpretability since they lack direct transactional meaning—while we ignore this caveat in this analysis to prioritize prediction, the lack of interpretability means we are unable to come up with an *a priori* non-linear specification for a parametric model to account for any non-linear trends in the data. Since these trends are unknown, we rely on non-parametric specifications that will better be able to capture unique interactions and non-linear effects. Each Pricinpal component, by construction, is uncorrelated with each other; however, this doesn't negate the possibility of *Amount* being correlated with any of the partial components (See Figure 1). In our analysis, we will evoke methods that are robust to multicolinearity as well as methods that are resilient to factors that aren't significantly meaningful.

A preliminary analysis reveals that as *Amount* increases, fraud is more likely (See Figure 2). However, we caution as interpreting this result as causal as there may be confounding effects unadjusted for. We also acknowledge the potential non-linearity in *Amount* with respect to the likelihood of fraud (as captured by the uncertainty bounds in Figure 2).

Methodology

To identify fraudulent credit card transactions effectively, we consider two robust ensemble methods: Random Forest and Gradient Boosting. Both methods have demonstrated high predictive performance in classification

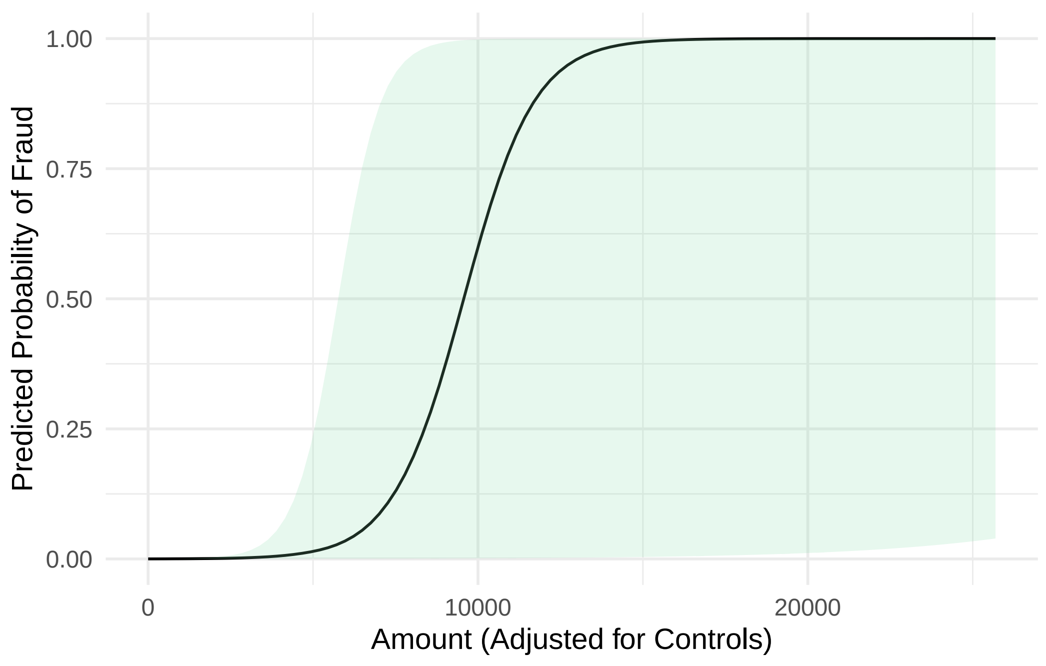

Figure 2: Relationship between transaction amount and likelihood of fraud. Estimates given by a logistic regression model with $n = 284,802$. Uncertainty estimates computed as the 95% quantile interval of $B = 100$ bootstrapped estimations.

tasks, particularly with imbalanced datasets, making them suitable for our analysis. We also consider a k-Nearest Neighbors (KNN) model as a baseline model to compare against the ensemble models.

1. A k-Nearest Neighbors (KNN) model classifies new observations based on the "k" most similar data points in the training set (See Algorithm in 2). Since the data contain uncorrelated principal components, similarity is evaluated based on Euclidean distance between observations. KNN is non-parametric, so no distributional assumptions are necessary to model fraud. KNN is sensitive to localities, allowing the model to capture complex, non-linear patterns. However, such sensitivity can cause the model to overfit the data. KNN can also struggle with the class imbalance present in the data, as the majority class tends to dominate nearest neighbors. To address these, we can tune hyperparameters such as the number of neighbors and weighting schemes of these neighbors to improve predictive performance. The KNN model cannot assess a variable's significance or relative importance, but this is of little concern given the inherent barrier to interpretability from dimension reduction.

2. Using a Random Forest (RF) model will help us eliminate potentially unnecessary factors that would otherwise fit noise in the data through *bagging* (we describe the bagging method, as implemented through a RF model in the Appendex; see Algorithm 1). The RF model, by construction, will reduce the variance in our predictions. Additionally, since the RF model is an aggregation of decision trees, it will also help model the non-linear trends and complex interactions within our data. However, the extreme class imbalance may lead the RF model to underperform in identifying fraudulent transactions. In order to minimize this bias, we will tune fitting hyperparameters.

3. Similar to the Random Forest Model, Gradient Boosting (GB) uses a series of decision trees—by construction, this will help us model the complex and non-linear relationships in our data. However, in contrast from RF, rather than an *aggregation* of indenpent decision trees fit on a random selection of the data, GB builds trees sequentially, each one focused on correcting the errors of its predecessor. In each iteration, the model minimizes a differentiable loss function by adding a new tree that fits the residuals of the combined ensemble from previous iterations. For binary classification, we specify the

Gradient Boosting model as[1],

$$\hat{f}_{GB}(x) = \sum_{m=1}^{M} \alpha_m h^{(m)}(x) \tag{1}$$

Where each new tree, $h^{(m)}(x)$, is trained[2] on the residuals (error) of the current prediction ($\hat{f}_{m-1}$). Similar to the RF model, the GB model yields high accuracy, models complex patters, and through hyperparameter tuning, allows for control of model complexity. The ability to "learn" from it's mistakes allows it to reduce bias iteratively, although we caution against overfitting a model like GB due to its inherent decision tree structure and sensitivity to model hyperparameters.

[1]Here we define M as the total number of trees, $h^{(m)}(x)$ is the m-th decision tree in the sequence, and α_m is the learning rate to control the contribution of each tree.

[2]We evaluate how well the new tree fits on the residuals of the current predictions by computing the negative gradient of the loss function, $L(y, \hat{f}(x)) = -[y \log \hat{p}(x) + (1-y)\log(1-\hat{p}(x))]$ (that is using the binary cross-entropy loss for binary classification),

$$g_i(m) = -\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\Bigg|_{\hat{f}(x)=\hat{f}_{m-1}(x)}$$

These often-called "pseudo-residuals" (the vector of $g_i(m)$) is what the subsequent model is trained on.

Appendix

Algorithm 1 Random Forest Algorithm

```
1: **Input:** Training data with $n$ samples and $p$ features
2: **Parameters:** Number of trees $B$, number of features to consider at each split $m$ (where $m < p$)
3: for $b = 1, \dots, B$ do
4:     Draw a bootstrapped sample of size $n$ from the training data
5:     Grow a decision tree $\mathcal{T}_b$ on this sample:
         1. At each node, randomly select $m$ features from the $p$ available features
         2. Split on the best feature among the $m$ chosen features (based on some criterion, e.g., Gini impurity for classification)
         3. Repeat until the stopping criterion is met (e.g., maximum depth or minimum node size)
6: end for
7: **Prediction:** For a new observation $x_0$
         1. For each tree $b = 1, \dots, B$, obtain a prediction $\hat{y}^b(x_0)$ by passing $x_0$ down tree $\mathcal{T}_b$
         2. For regression: average the predictions:
                $\hat{y}(x_0) = \frac{1}{B}\sum_{b=1}^{B}\hat{y}^b(x_0)$
         3. For classification: take the majority vote:
                $\hat{y}(x_0) = \text{mode}(\hat{y}^1(x_0), \dots, \hat{y}^B(x_0))$
```

Algorithm 2 Weighted k-Nearest Neighbors Algorithm

```
1: Input: Training data with $n$ samples and $p$ features
2: Parameters: Number of neighbors $k$, Distance metric $d$ (e.g., Euclidean, Mahalanobis), Weighting
   scheme $w$ (e.g., uniform, distance-based)
3: for a new observation $x_0$ do
4:     Calculate the distance $d(x_0, x_i)$ between $x_0$ and each training sample $x_i$ in the dataset
5:     Identify the $k$ nearest neighbors to $x_0$ by selecting the $k$ samples with the smallest distances to $x_0$
6:     Apply the weighting scheme $w_i$ to each of the $k$ neighbors:
         1. For uniform weights, $w_i = 1$
         2. For distance-based weights, $w_i = \frac{1}{d(x_0,x_i)}$
7: end for
8: Prediction: For a new observation $x_0$
     1. For classification:
          a) Calculate weighted class frequencies:
               $$F_c = \sum_{i=1}^{k} (w_i \cdot \mathbb{I}(y_i = c))$$
             where $\mathbb{I}(y_i = c)$ is 1 if $y_i = c$, and 0 otherwise.
          b) Predict the class for $x_0$ as:
               $$\hat{y}(x_0) = \arg\max_c F_c$$
     2. For regression:
          a) Calculate the weighted average of the target values of the $k$ neighbors:
               $$\hat{y}(x_0) = \frac{\sum_{i=1}^{k} w_i \cdot y_i}{\sum_{i=1}^{k} w_i}$$
```