Using KDD Methodology for Credit Card Fraud Detection

Steven Chang

College of Engineering, San Jose State University

Abstract

KDD is a methodology for performing data mining. It involves 5 phases: selection, pre-processing, data transformation, modeling, and evaluation. In this paper, we will explore the KDD process by performing principled, step-by-step, phase-by-phase data science with the methodology. To do so, we will demonstrate its usage in a project. Due to the limitation of not having practical, business data, this project will pertain to detecting credit card fraud based on a dataset retrieved from Kaggle. The dataset selected already had some data transformation steps performed that would typically be performed in the data transformation phase. This transformation was made primarily for confidentiality reasons, however, this will simplify the data transformation phase of this project.

Selection

All of the data for this process was selected from Kaggle. The dataset selected contains data on credit card transactions in Europe over the course of two days. Many of the data points have already undergone PCA transformation to maintain data anonymity and confidentiality.

Data Set Overview

The Time column represents the seconds elapsed between each transaction and the first transaction in the dataset. Features V1 to V28 are the principal components obtained via PCA transformation. These components are already in a numerical format and have been transformed to maintain data confidentiality. The Amount column shows the transaction amount. The Class column is our target variable, where 1 indicates a fraudulent transaction and 0 indicates a non-fraudulent transaction.

Pre-Processing

This phase involved further exploring our dataset and making minor changes to the data set for data cleaning. This phase involves checking for missing values, developing a plan for data imbalance, and obtaining descriptive statistics on the dataset.

Missing Values: Fortunately, there are no missing values in our dataset. Each feature is complete.

Class Distribution: As anticipated, there's a significant class imbalance. Approximately 99.83% of the transactions are non-fraudulent (Class = 0), and only about 0.17% are fraudulent (Class = 1). This imbalance confirms the need to use metrics like AUPRC rather than simple accuracy when evaluating our model's performance.

Descriptive Statistics:

Amount: Transactions range from \$0 to about \$25,691.16, with a mean transaction amount of approximately \$88.35.

Time: The recorded time ranges from 0 seconds to approximately 172,792 seconds (or about 48 hours, which matches the two-day data collection period). The median time is around 84,692 seconds.

Data Transformation

Much of our dataset had already been through the data transformation phase prior to receiving it so the sensitive information would be anonymized. The remaining features, 'Time' and 'Amount' must still be transformed. Additionally, in this phase, we will split the dataset into a training and validation set.

Feature Scaling: While the PCA components (V1 to V28) are likely already scaled, the features 'Time' and 'Amount' are not. To ensure that our model does not unduly prioritize one feature over others because of its scale, we need to standardize these features.

Dataset Splitting: We need to split the dataset into training and testing sets. This will allow us to train our model on one subset of the data and then evaluate its performance on a separate subset that it hasn't seen before. We used an 80/20 split for training and validation.

Training features (X_train): 227,845 samples with 30 features each.
Testing features (X_test): 56,962 samples with 30 features each.
Training target (y_train): 227,845 samples.
Testing target (y_test): 56,962 samples.

After this phase, the dataset is now ready for modeling. The features were appropriately scaled, and we had distinct training and testing sets to build and evaluate our models.

Modeling

In this step, we will focus on building a baseline classifier to detect fraudulent transactions. Given the class imbalance and the nature of the data, I'll start with a logistic regression model. Logistic regression is simple, interpret-able, and often serves as a good baseline. Because of its simplicity I expect that it will not have the best performance. As such this model will serve as a baseline to compare against more complicated models.

Because of the extreme imbalance in this dataset, we will be using Area-Under Precision-Recall Curve for evaluation.

Logistic Regression
AUPRC = 0.7423

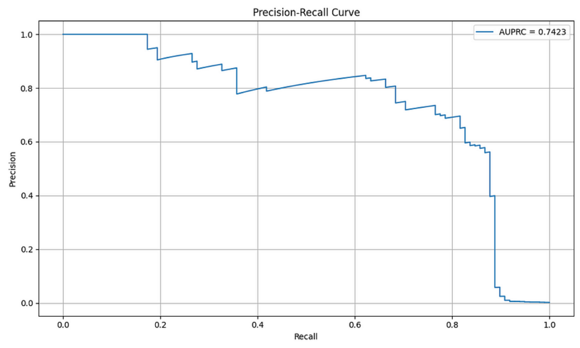

Figure 1. PRC Curve for Logistic Regression Model

The Precision-Recall curve has been visualized, and the Area Under the Precision-Recall Curve (AUPRC) for our logistic regression model is approximately 0.7423. This is a reasonable starting point, especially for a baseline model on such an imbalanced dataset.

Random Forest Machines
AUPRC = 0.8774

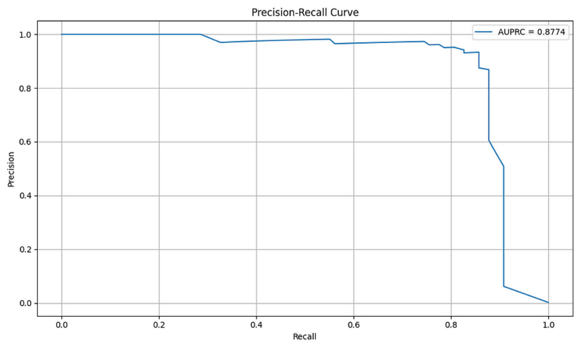

Figure 2. PRC Curve for Random Forest Machines Model

The Area Under the Precision-Recall Curve (AUPRC) for our logistic regression model is approximately 0.8774. An AUPRC of approximately 0.8774 for the Random Forest classifier is quite impressive, especially given the challenge of the class imbalance. It demonstrates that the Random Forest model is significantly better than our initial logistic regression baseline, which had an AUPRC of about 0.7423.

Gradient Boosted Trees (XGBoost)
AUPRC = 0.8804

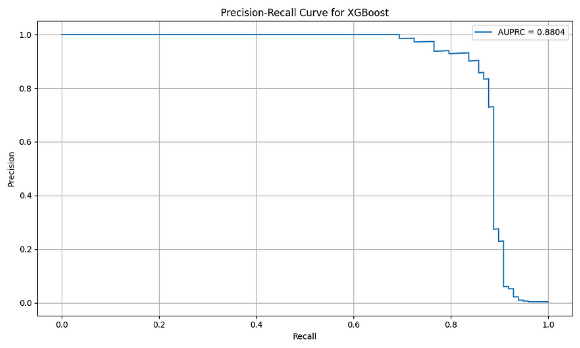

Figure 3. PRC Curve for XGBoost Model

The Area Under the Precision-Recall Curve (AUPRC) for our logistic regression model is approximately 0.88042. An AUPRC of approximately 0.88042 for the XGBoost classifier indicates that it's performing slightly better than the Random Forest model, which had an AUPRC of 0.87740. Both models are significantly better than our initial logistic regression baseline.

Evaluation

Deployment Strategies

Batch Predictions:
Use Case: When you need to score or classify large amounts of data at once (e.g., end-of-day processing).
Implementation: Model predictions are made on batches of data, often stored in databases or data lakes.

Real-Time Predictions:
Use Case: For real-time fraud detection where a transaction needs to be classified instantly.
Implementation: Deploy the model using a service that can handle API requests. Services like AWS Lambda, Google Cloud Functions, or a dedicated server with Flask or FastAPI can be used.

Edge Deployments:
Use Case: If the fraud detection needs to happen on a device (e.g., a credit card terminal) without reliable internet access.
Implementation: Lightweight models or quantized models can be deployed directly onto devices.

Summary

Logistic Regression: AUPRC = 0.7423
Random Forest: AUPRC = 0.87740
XGBoost: AUPRC = 0.8804

Both the Random Forest and XGBoost models outperform the Logistic Regression baseline, with XGBoost slightly edging out the Random Forest. The small difference in AUPRC between Random Forest and XGBoost suggests that both models capture similar patterns in the data, although XGBoost might be capturing some additional nuances. The Logistic Regression model, while simpler, still provides a decent baseline and can be useful for understanding linear relationships in the data.