Detecting Credit Card Fraud

Team 5: Tony Lin Lucas Embrey Beatrice Lee Gabriel Montes Simon Grishin

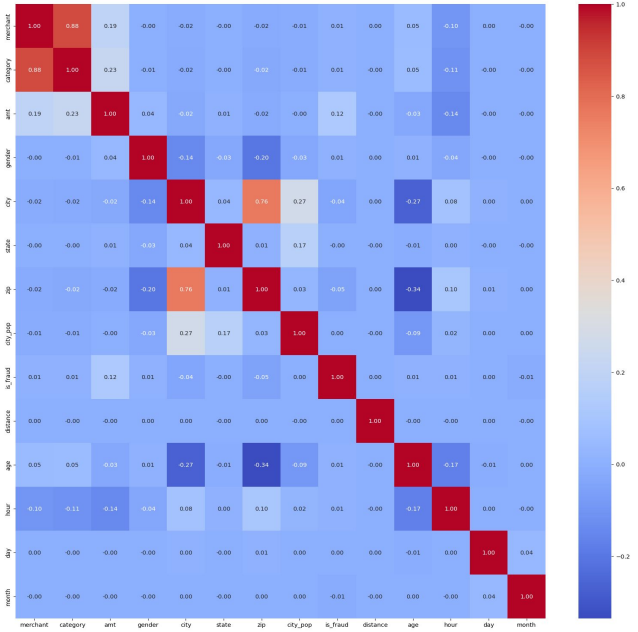
Objective

- Detect fraudulent transactions from a credit card transaction dataset containing both legitimate and fraudulent transactions
- Minimize false positives and false negatives
- Various models will be trained on a training set, assessed with the validation dataset and then evaluated on the test data
- Use a utility function to penalize false predictions and reward correctly identified fraud

Utility Function

- The utility function is ∑#TP (S - L) - ∑#FN C - ∑#FP P
  - S is the savings from preventing fraudulent transactions.
  - L is the cost of blocking or investigating a flagged transaction.
  - C is the cost incurred due to undetected fraud.
  - P is the penalty for incorrectly flagging a legitimate transaction
- (S-L)=$50, C = $100, P = $5

Data Description



1296675 rows of training data, with 23 predictors and a binary response

1289169 non-frauds, 7506 frauds

Removed non-predictive explanatory variables, and created new distance variable to measure how far from purchase a person's address was (used latitudes and longitudes to compute this)

<u>Encoding</u>

- Frequency: Measures the normalized count of categorical values (ex. Merchant, State, Category)
- Label: Generic ordinal transformation (ex. Gender)
- Log: Linear transformation for numerical features with large distributions (ex. amt, city_pop)

Encodings transform categorical features to numerical values while maintaining variance and semantic meaning for greater predictability.

See explanatory variables included in analysis, and their pairwise correlations in the correlation matrix.

Data Splitting

Models were trained using 80% of the available training data, with the remaining 20% reserved for validation to assess performance before testing on unseen data.

The training dataset exhibited significant class imbalance, with approximately 1 fraudulent transaction per 172 non-fraudulent ones. To address this, a class weight of 172 was applied to fraudulent cases in models such as XGBoost, Random Forest, SVM ensuring the models paid appropriate attention to the minority class. A bisection method was used to search for the best class weight for logistic regression, which was 84.

Baseline model results

Based on our initial model testing on validation set, these utility scores were achieved by using 80%/20% split on training data:

	Validation Utility Score	TP	TN	FP	FN
Logistic Regression	-13025	1410	252879	8489	735
Gaussian Naive Bayes	-52675	811	252989	4845	690
Random Forest	-1325	990	257743	91	511
SVM	-68860	585	256893	922	935

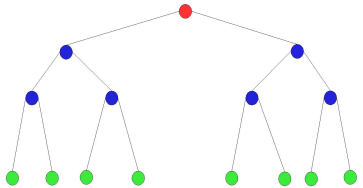
XGBoost Model

- Builds an ensemble of decision trees to make predictions, improving accuracy by iteratively correcting errors from previous trees.
- Objective is binary:logistic, which outputs probabilities between 0 and 1, indicating the likelihood of a sample belonging to the positive class.
- Model parameters were determined after running 5 fold cross validation, selecting the ensemble of parameter values with the highest AUC score.
    - n_estimators: Number of trees in the forest
    - learning_rate: Learning rate for boosting
    - max_depth: Maximum depth of a tree
    - subsample: Fraction of samples used per tree
    - colsample_bytree: Fraction of features used per tree

XGBoost Results

- When running the model utilizing XGBoost on a validation set created from our training data, it performed best by a large margin.
  - Utility was 60,070.
- When running the model on the test data as a final step, the utility was 63,785.
- Allowed for a very small number of false negatives, which is particularly advantageous given the utility function.

Why XGBoost Works Well



- Efficiency and Scalability: renowned for its execution speed and model performance.
- Handling of Imbalanced Data: addresses class imbalance by scaling positive weights in the data.
- Improved Accuracy: optimizes both bias and variance.
- Flexibility: hyperparameters can be finely tuned.
- Regularization: It includes L1 and L2 regularization, which help prevent overfitting. Helpful for noisy data with lots of potentially unimportant features.

Conclusion

After trying a range of models, XGBoost performed best with a utility of 60070 on the validation set. Out of the baseline models, Random Forest also performed well with a utility close to 0 unlike the other models.

The test data had a utility of 63785, which shows that the model was not overfitted to the training data and generalised well to unseen data.

Some of the features were useful in predictions whilst others were often discarded during model building. A calculated distance feature proved useful.