Mini-Project 3: Report

-Sainath Chettupally, Srikar Gowrishetty, Sai Satwik Yarapothini

Introduction and Overview

In this project, we set out to build a predictive model for a binary classification problem using the dataset from `new_train_EGN5442.csv`. It wasn't just a walk in the park — the dataset had a mix of both categorical and numerical features, each needing a good bit of wrangling and love to make them usable. We dove into tasks like data cleaning, feature engineering, and even some dimensionality reduction. After all that, we trained and tested our models using Logistic Regression and Random Forest to see which one would come out on top. Our goal? To figure out which approach worked best for our classification challenge.

Data Cleaning and Preparation

Examination of Data

- **Balanced or Imbalanced Classes:** The first thing we did was check if the target variable (`y`) was balanced. It turned out to be slightly imbalanced, which could potentially mess with our model's performance if left unchecked.
- **Constant Features:** We searched for any features that were constant (essentially no variance) and removed them since they don't add any valuable information.
- **Handling Missing Data:** Missing data was present in several columns. Specifically, we imputed the missing values in `x24` with the median, while `x30` was filled with a constant value of zero. For the categorical columns, we went with the mode for each one to handle missing data.
- **Data Types and Scaling:** We used OneHotEncoder to convert categorical features and scaled the numerical ones with StandardScaler to make sure everything was on the same page, so to speak.

Visualizations and Correlations

- **Missing Data Heatmap:** We used a heatmap to get a visual idea of just how much data was missing in each feature. It's always good to see the problem before tackling it.
- **Feature Correlations:** We took a look at correlations between our numerical features and the target to understand what relationships might exist. We encoded our categorical data and scaled the numerical stuff before doing this.

Data Scaling and Splitting

- **Feature Transformation:** The numerical features were scaled using StandardScaler, and OneHotEncoder took care of the categorical features. Because of the dataset's high dimensionality, we employed TruncatedSVD to reduce it, keeping the key information while making it a little easier to handle.

- **Train-Test Split:** We split the dataset into training and testing sets, using an 80:20 ratio. This split helped us evaluate how well our model would perform with new, unseen data.

Part A - Logistic Regression Model

Logistic Regression Overview

Logistic Regression is a fairly straightforward, linear model used for binary classification problems. It's kind of like the simplest tool in your toolbox for this type of job — easy to use, easy to understand, but not necessarily the best fit for every scenario.

- **Advantages:** It's simple, easy to implement, and pretty computationally cheap.
- **Disadvantages:** It's only good for data with linear relationships, so if the data's complex or non-linear, it struggles.

Feature Selection and Cross-Validation

- **Feature Engineering:** We used TruncatedSVD to knock down our high-dimensional data to just 10 components.
- **Cross-Validation:** We ran a 5-fold cross-validation to see how robust our model was, achieving an average AUC score of about 0.91.
- **Hyperparameter Tuning:** We also used L1 regularization to try and prevent the model from overfitting.

Logistic Regression Results

- **Confusion Matrix:** Our Logistic Regression model showed a pretty good balance between True Positives and True Negatives.
- **ROC Curve and AUC Score:** We got an AUC score of 0.92 on the test set, which means it did a decent job of telling apart the two classes.

Part B - Non-Logistic Model: Random Forest Classifier

Random Forest Overview

Random Forest is kind of like having a whole bunch of decision trees that all pitch in their votes to make a final decision — it's an ensemble method, which often works wonders for non-linear data.

- **Advantages:** Handles complex, non-linear relationships really well. It's also less prone to overfitting compared to a single decision tree and usually provides better accuracy.
- **Disadvantages:** It's not quite as simple or easy to interpret as Logistic Regression and can be computationally expensive.

Hyperparameter Tuning and Cross-Validation

- **Cross-Validation:** We performed 5-fold cross-validation and got an average AUC score of 0.96, which was pretty great.
- **Hyperparameter Tuning:** We tweaked a couple of parameters — specifically, the number of estimators (`n_estimators=50`) and maximum depth (`max_depth=10`) to optimize performance.

Random Forest Results

- **Confusion Matrix:** The Random Forest model outperformed Logistic Regression, showing better recall and precision.
- **ROC Curve and AUC Score:** The AUC score here was 0.96 on the test set, making it clear it did a better job than Logistic Regression.

Discussions

Model Comparison

- **Performance:** Overall, Random Forest came out ahead, with an AUC score of 0.96 compared to Logistic Regression's 0.92. It was just better at identifying the right patterns, which showed in its higher recall and precision scores.
- **Complexity and Interpretability:** While Logistic Regression is simple and very interpretable — useful if we need to explain things to non-technical folks — Random Forest captured the data's complexities better. That said, it comes at the cost of added complexity.

Explanation to Business Partner

So, why did Random Forest do better? Well, it just picked up more subtle and complex patterns compared to Logistic Regression. This means it's going to make more accurate predictions overall, which translates to better decision-making. Logistic Regression might be easier to explain but isn't quite as effective at finding those nuanced relationships.

Model	Logistic Regression	Random Forest
AUC for Training	~0.91	~0.96
AUC for Cross-Validation	~0.91	~0.96
AUC for Testing	0.92	0.96

Challenges Faced

- **Data Imbalance:** Since the dataset was slightly imbalanced, it could've led to biased predictions. We tackled this with cross-validation and by using balanced metrics.
- **High Dimensionality:** The dataset had a lot of features after OneHotEncoding. TruncatedSVD helped reduce this while preserving the important parts.

Conclusion

Both models were effective in their own right, but Random Forest came out on top because of its ability to capture the complex relationships in the data. However, Logistic Regression remains a perfectly reasonable choice if simplicity and interpretability are more important.