Assignment 3: Cost-Sensitive Data Analysis Using Regression Models

Kummitha Jhanavi CS21BTECH11032
Potta Vennela CS21BTECH11046
Vinjam Aswitha MA21BTECH11018
Pundi Bindusree CS21BTECH11048
K Vivek Kumar CS21BTECH11026

May 2, 2024

Abstract

Keywords: Cost-sensitive analysis, Data analysis, Fraud Analytics

This paper presents a solution for conducting cost-sensitive data analysis on a dataset provided for Assignment 3 in the Fraud Analytics course.

---

1 Introduction to Cost-Sensitive Regression

The problem with traditional regression models is their inability to account for varying costs of prediction errors. In real-world scenarios like medical diagnostics or fraud detection, the impact and associated costs of false positives and false negatives can differ significantly.

1.1 What's Cost-Sensitive Regression?

Cost-sensitive regression addresses the challenge of varying costs associated with different types of prediction errors (e.g., false positives, false negatives) in predictive modeling. It aims to optimize models considering the specific costs of prediction outcomes.

Equation for Cost-sensitive loss function:

$$\text{Loss} = \sum_{i=1}^{n} (\text{Cost}(y_i, \hat{y}_i)) \tag{1}$$

Bahnsen's Approach

Bahnsen's approach adjusts the learning process by incorporating example-specific costs directly into the model training. It aims to minimize the expected cost of predictions using a cost-sensitive loss function.

Nikou and Gunnemann's Approach

Nikou and Gunnemann's approach focuses on minimizing the overall expected cost of predictions by adapting the learning process to account for varying costs of false positives and false negatives.

2 Problem Statement

Given a dataset `costsensitive.csv`, we have to do the cost-sensitive analysis using two approaches, Bahnsen's Approach and the Nikou and Gunnemann's Approach. Provide proper results by also testing the models on test datasets.

3 Description of the Dataset

Independent Variables

NotCount, YesCount, ATPM, PFD, PFG, SFD, SFG, WP, WS, AH, AN are the columns associated with the independent variables.

Dependent Variable

Status: Categorical variable representing the status (1 or 0) is the dependent variable.

Cost Information

False Negative Cost (FNC): Column FNC specifies the false negative cost associated with each observation, which varies from row to row based on risk parameter details.

Constant Costs for Prediction Outcomes

- **True Positive Cost**: Constant cost of 6 for predicting a positive instance (true positive).
- **False Positive Cost**: Constant cost of 6 for predicting a negative instance as positive (false positive).
- **True Negative Cost**: Constant cost of 0 for correctly predicting a negative instance (true negative).

4 Approach (Algorithm Used)

4.1 Bahnsen's Approach

1. **Dataset Preparation:** The dataset is loaded into a pandas DataFrame (`data`), where columns `NotCount` to `AN` are designated as independent variables (`X`), and `Status` is set as the dependent variable (`y`). Additionally, the false negative costs (`FNC`) are extracted as `false_negative_cost`.

2. **Data Splitting:** The dataset is split into training (`X_train`, `y_train`, `fnc_train`) and testing (`X_test`, `y_test`, `fnc_test`) sets using `train_test_split()`, allocating 20% of the data for testing purposes.

3. **Class Weight Calculation:** Class weights are computed using `class_weight.compute_class_weight()` to account for class imbalance in the training data, ensuring the model learns effectively under cost-sensitive conditions.

4. **Logistic Regression Model Setup:** A logistic regression model (`bahnsen_model`) is instantiated with the following configurations:

   - `class_weight={0: class_weights[0], 1: class_weights[1]}`: Assigns computed class weights to balance the impact of different classes during training.
   - `solver='lbfgs'`: Optimization solver for logistic regression.
   - `max_iter=1000`: Maximum number of iterations for the optimization process.

5. **Model Training and Prediction:** The logistic regression model (`bahnsen_model`) is trained on the training data (`X_train`, `y_train`) using `fit()` and subsequently used to predict the test data (`X_test`) with `predict()`.

4.2 Nikou and Gunnemann's Approach

Implementation of Nikou and Gunnemann's Cost-Sensitive Logistic Regression

1. **Dataset Setup**: The provided dataset (`costsensitiveregression.csv`) is loaded into a pandas DataFrame (`data`). The independent variables (`X`) are defined using columns `NotCount` to `AN`, while the dependent variable (`y`) is set as `Status`. The false negative costs (`false_negative_cost`) are extracted from the dataset's `FNC` column.

2. **Data Splitting**: The dataset is split into training (`X_train`, `y_train`, `fnc_train`) and testing (`X_test`, `y_test`, `fnc_test`) sets using `train_test_split()`. A test size of 20% is specified with a random seed (`random_state=42`) for reproducibility.

3. **Class Weight Calculation**: Class weights are computed using `class_weight.compute_class_weight()` to address class imbalance in the training labels (`y_train`), ensuring the model learns effectively under cost-sensitive conditions.

4. **Logistic Regression Model Initialization**: A logistic regression model (`nikou_gunnemann_model`) is instantiated with specific configurations:

   - `class_weight={0: class_weights[0], 1: class_weights[1]}`: Assigns computed class weights to balance the impact of different classes during training.
   - `solver='lbfgs'`: Specifies the optimization solver used for logistic regression.
   - `max_iter=1000`: Sets the maximum number of iterations for the optimization process.

5. **Model Training and Prediction**: The logistic regression model (`nikou_gunnemann_model`) is trained on the training data (`X_train`, `y_train`) using `fit()`. Predictions (`nikou_gunnemann_y_pred`) are then generated on the test data (`X_test`) using `predict()`.

5 Results

5.1 Model Evaluation for Bahnsen's Approach

The results of Bahnsen's Approach are as follows:

Bahnsen's Approach Confusion Matrix

The confusion matrix for Bahnsen's Approach is shown below:

$$\begin{bmatrix} 18145 & 2562 \\ 1636 & 7185 \end{bmatrix}$$

The confusion matrix reveals the following:

- True Negatives (TN): 18145
- False Positives (FP): 2562
- False Negatives (FN): 1636
- True Positives (TP): 7185

Bahnsen's Approach Classification Report

The classification report for Bahnsen's Approach is presented below:

	Precision	Recall	F1-score	Support
0	0.92	0.88	0.90	20707
1	0.74	0.81	0.77	8821
Accuracy			0.86	29528
Macro Avg	0.83	0.85	0.84	29528
Weighted Avg	0.86	0.86	0.86	29528

Table 1: Classification Report for Bahnsen's Approach

The classification report provides insights into the precision, recall, and F1-score for each class (0 and 1), along with the overall accuracy, macro average, and weighted average metrics. These metrics help evaluate the performance of Bahnsen's Approach in predicting the target variable.

5.2 Model Evaluation for Nikou-Gunnemann's Approach

The results of Nikou-Gunnemann's Approach are as follows:

Nikou-Gunnemann's Approach Confusion Matrix

The confusion matrix for Nikou-Gunnemann's Approach is shown below:

$$\begin{bmatrix} 18145 & 2562 \\ 1636 & 7185 \end{bmatrix}$$

The confusion matrix reveals the following:

- True Negatives (TN): 18145
- False Positives (FP): 2562
- False Negatives (FN): 1636
- True Positives (TP): 7185

Nikou-Gunnemann's Approach Classification Report

The classification report for Nikou-Gunnemann's Approach is presented below:

	Precision	Recall	F1-score	Support
0	0.92	0.88	0.90	20707
1	0.74	0.81	0.77	8821
Accuracy			0.86	29528
Macro Avg	0.83	0.85	0.84	29528
Weighted Avg	0.86	0.86	0.86	29528

Table 2: Classification Report for Nikou-Gunnemann's Approach

The classification report provides insights into the precision, recall, and F1-score for each class (0 and 1), along with the overall accuracy, macro average, and weighted average metrics. These metrics help evaluate the performance of Nikou-Gunnemann's Approach in predicting the target variable.