Example Dependent Cost Sensitive Classification

Fraud Analytics Project Report

Sanskriti Agarwal
CS24MTECH14002

Kocherla Sai Kiran
AI24MTECH02003

Kota Dhana Lakshmi
AI22BTECH11012

Instructor: Prof. Sobhan Babu

Institution: IIT Hyderabad

March 26, 2025

Contents

Abstract

Cost-sensitive classification is essential in domains like fraud detection and medical diagnosis, where false positives (FP) and false negatives (FN) have unequal costs. Traditional logistic regression does not account for these variations, while Bahnsen's Cost-Sensitive Logistic Regression (CSLR) optimizes decision thresholds but not model parameters.This study explores multiple optimization approaches to improve CSLR, including Genetic Algorithm (GA), Custom CSLR with Gradient Descent. GA evolves model parameters through selection and mutation, while the custom approach refines weights via cost-aware gradient updates. DE further enhances CSLR by efficiently searching for optimal weights and biases to minimize misclassification cost. Comparative analysis across probability cutoffs demonstrates that DE achieves the best cost reduction, outperforming standard logistic regression and other CSLR variants.

1 Problem Statement

Traditional logistic regression does not account for varying misclassification costs, which can be problematic in applications where false negatives (FN) and false positives (FP) carry significantly different penalties. Bahnsen's Cost-Sensitive Logistic Regression (CSLR) addresses this issue by incorporating cost functions, but its optimization primarily relies on threshold tuning rather than directly optimizing model parameters such as weights and biases.A key challenge in CSLR is identifying an optimal set of model parameters to minimize total misclassification cost while maintaining robust generalization across datasets. Standard gradient-based methods often struggle with cost-sensitive classification due to its non-convex nature, leading to suboptimal solutions.

2 Dataset Description

The dataset used in this study consists of 147,636 instances with 13 features, including categorical and numerical attributes related to cost-sensitive classification. The dataset contains the following attributes:

- **NotCount**: Number of times an instance was classified as negative.
- **YesCount**: Number of times an instance was classified as positive.
- **ATPM**: A continuous numerical feature representing a specific metric.
- **PFD, PFG, SFD, SFG**: Numerical attributes related to some performance or cost metrics.
- **WP, WS**: Weighting parameters for different cost factors.
- **AH, AN**: Additional numerical features contributing to classification.
- **Status**: The target variable (binary classification: 0 or 1).
- **FNC**: A cost-sensitive feature contributing to classification.

The dataset is utilized to optimize a cost-sensitive logistic regression model, comparing traditional logistic regression approaches with Bahnsen's cost-sensitive method optimized regression models. The goal is to minimize misclassification costs by adjusting the decision boundary based on cost-sensitive metrics.

NotCount	YesCount	ATPM	PFD	PFG	SFD	SFG	WP	WS	AH	AN	Status	FNC
2	21	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0.044	0	0	0	0.306179	0	0	0	1	0
1	22	0	0	0	0	0	0	0	0	0	0	0
5	18	0	0	0	0	0	0	0	0	0	1	0
1	22	0	0	0	0	0	0	0	0	0	0	0
2	21	0	0	0	0	0	0	0	0	0	0	0
1	22	0	0.01354	0.52732	0	0	2.101799	1	0	0	0	0

Table 1: The first 7 entries of costsensitiveregression.csv

3 Methodology

3.1 Logistic Regression

Logistic Regression models the probability of an instance belonging to class 1 using the sigmoid function:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[-y_i \log h_\theta(x_i) - (1 - y_i) \log(1 - h_\theta(x_i)) \right] \tag{1}$$

where:

- $J(\theta)$: Cost function (loss function) to be minimized.
- N : Total number of training examples.
- i : Index of a single training example.
- y_i : True label of the i-th training example ($y_i \in \{0, 1\}$).
- x_i : Feature vector of the i-th training example.
- $h_\theta(x_i)$: Hypothesis function (predicted probability of $y_i = 1$), defined as:

$$h_\theta(x_i) = \frac{1}{1 + e^{-\theta^T x_i}} \tag{2}$$

- $\log h_\theta(x_i)$: Log probability of the predicted class when $y_i = 1$.
- $\log(1 - h_\theta(x_i))$: Log probability of the predicted class when $y_i = 0$.

This is the standard loss function used in **logistic regression** for binary classification, where the goal is to find the optimal parameter θ that minimizes $J(\theta)$.

3.2 Bahnsen's Cost-Sensitive Logistic Regression

Bahnsen's approach integrates **misclassification costs** directly into logistic regression. Instead of minimizing log-loss, it minimizes the **total misclassification cost**:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[y_i (C_{TP} h_\theta(x_i) + C_{FN}(1 - h_\theta(x_i))) + (1 - y_i)(C_{FP} h_\theta(x_i) + C_{TN}(1 - h_\theta(x_i))) \right] \tag{3}$$

where

- $J(\theta)$: Cost function to be minimized.
- N : Total number of training examples.
- i : Index of a single training example.
- y_i : True label of the i-th training example ($y_i \in \{0, 1\}$).
- x_i : Feature vector of the i-th training example.
- $h_\theta(x_i)$: Hypothesis function (predicted probability of $y_i = 1$), defined as:

$$h_\theta(x_i) = \frac{1}{1 + e^{-\theta^T x_i}} \tag{4}$$

- C_{TP} : Cost associated with a **true positive** (correctly predicting $y_i = 1$).
- C_{FN} : Cost associated with a **false negative** (incorrectly predicting $y_i = 0$ when it should be 1).
- C_{FP} : Cost associated with a **false positive** (incorrectly predicting $y_i = 1$ when it should be 0).
- C_{TN} : Cost associated with a **true negative** (correctly predicting $y_i = 0$).

The optimal **decision threshold** is:

$$\tau = \frac{C_{FN}}{C_{FN} + C_{FP}} \tag{5}$$

This ensures that the model reduces costly misclassifications rather than optimizing for accuracy.

4 Optimization Approaches

In classification problems, misclassification costs are often **asymmetric**, making cost-sensitive learning crucial. We explore different approaches to optimize logistic regression for cost-sensitive classification:

- **Standard Logistic Regression**
- **Bahnsen's Cost-Sensitive Logistic Regression**
- **Optimization Approaches:**
  - Custom Manual Approach
  - Genetic Algorithm

Each method optimizes logistic regression **decision thresholds** and **model parameters** to minimize classification costs.

4.1 Custom Manual Approach

This approach manually adjusts the threshold based on empirical evaluation of misclassification costs.

$$\tau^* = \arg\min_{\tau} \left(FP(\tau) \times C_{FP} + FN(\tau) \times C_{FN} \right) \tag{6}$$

4.2 Genetic Algorithm (GA)

Genetic Algorithms optimize model parameters by evolving solutions over generations.

Crossover:

$$x_{\text{new}} = \alpha x_{\text{parent1}} + (1-\alpha)x_{\text{parent2}} \tag{7}$$

Mutation:

$$x_{\text{mutated}} = x + \text{random noise} \tag{8}$$

Genetic Algorithm 1

We implemented a cost-sensitive logistic regression using a Genetic Algorithm (GA) to minimize example-dependent misclassification costs. The GA evolves a population of weight vectors through selection, crossover, and mutation over generations. The model was evaluated across different probability thresholds to find the cutoff that minimizes the average cost on test data.

Genetic Algorithm-2

This implementation applies a Genetic Algorithm to train a cost-sensitive logistic regression model based on the method proposed by Bahnsen et al., where false negative costs vary per instance. The model evolves a population of logistic regression parameters over 100 generations using tournament selection, elitism, uniform crossover, and adaptive mutation. Fitness is evaluated using Bahnsen's cost function, which incorporates example-dependent misclassification costs.

A plot of cost versus probability cutoff is generated to visualize performance across thresholds and identify the point of minimum cost. This approach is particularly suitable for applications like fraud detection or medical diagnostics, where false negatives carry varying and significant costs.

5 Tools and Libraries Used

The implementation uses the following Python libraries:

- **numpy** – Numerical computations
- **pandas** – Data manipulation and analysis
- **matplotlib** – Data visualization
- **random** – Random number generation
- **train_test_split** (from `sklearn.model_selection`) – Splitting data into training and testing sets
- **StandardScaler** & **RobustScaler** (from `sklearn.preprocessing`) – Feature scaling
- **LogisticRegression** (from `sklearn.linear_model`) – Logistic regression model
- **BaseEstimator** (likely from `sklearn.base`) – Custom estimator base class
- **expit** (from `scipy.special`) – Sigmoid function

6 High-Quality Pseudocode

This pseudocode represents the cost-sensitive logistic regression model. The process includes data preprocessing, standard logistic regression training and evaluation, and Bahnsen's cost-sensitive regression using the Bahnsen(Custom) optimization method.

Algorithm 1 Data Preprocessing

```
1: Input: Raw dataset $D$
2: Remove missing values and handle outliers
3: Encode categorical variables (One-Hot or Label Encoding)
4: Normalize numerical features using StandardScaler
5: Split data into train ($D_{train}$) and test ($D_{test}$) sets
6: Output: Preprocessed data $(X_{train}, y_{train}), (X_{test}, y_{test})$
```

Algorithm 2 Standard Logistic Regression

```
1: Input: Training data $(X_{train}, y_{train})$
2: Initialize Logistic Regression model
3: Train model using Maximum Likelihood Estimation (MLE)
4: Compute probability scores $\hat{y} = \sigma(XW + b)$
5: Apply decision threshold $\tau = 0.5$
6: Output: Predicted labels $\hat{y}$
```

Algorithm 3 Bahnsen Custom Logistic Regression

```
1: Input: $(X_{train}, y_{train})$, Cost Matrix $C$
2: Train logistic regression model
3: Compute cost-sensitive decision threshold $\tau$
4: Adjust classification based on cost optimization
5: Output: Optimized threshold $\tau$ and predicted labels
```

Algorithm 4 Bahnsen with Genetic Algorithm (GA-1)

```
1: Input: $(X_{train}, y_{train})$, Cost Matrix $C$, Population $P$
2: Initialize population of thresholds
3: for each generation do
4:     Perform selection (e.g., tournament selection)
5:     Apply crossover to generate offspring
6:     Apply mutation to introduce diversity
7:     Evaluate cost function for each individual
8:     Select best solutions for next generation
9: end for
10: Output: Optimized threshold $w$
```

Algorithm 5 Bahnsen with Genetic Algorithm (GA-2) - Advanced

```
1: Input: $(X_{train}, y_{train})$, Cost Matrix $C$, Population $P$
2: Initialize population of weight vectors
3: for each generation do
4:     Compute fitness based on misclassification cost
5:     Apply crossover and mutation
6:     Adjust mutation rate dynamically
7:     Select top-performing individuals for next generation
8: end for
9: Output: Optimized weight vector minimizing cost
```

7 Results and Visualization Interpretation

The algorithm produced several outputs:

7.1 Visualization Explanation

The provided plots illustrate the relationship between classification probability outputs and the incurred cost using different approaches. The details of each visualization are as follows:

- **Standard Logistic Regression Approach:** A baseline comparison showing the cost versus probability behavior using standard logistic regression without any cost-sensitive modifications.

- **Bahnsen's GA-based Approach:** This visualization depicts the cost variations as a function of predicted probabilities using Bahnsen's GA-based cost-sensitive learning.

- **Custom Bahnsen's Approach:** Another custom variant that might introduce refined techniques for further cost optimization. This approach manually tunes parameters to balance false positives and false negatives effectively.

These visualizations provide a comparative insight into how different cost-sensitive methodologies impact cost minimization while maintaining predictive accuracy in classification tasks.

7.2 Visualization Plots

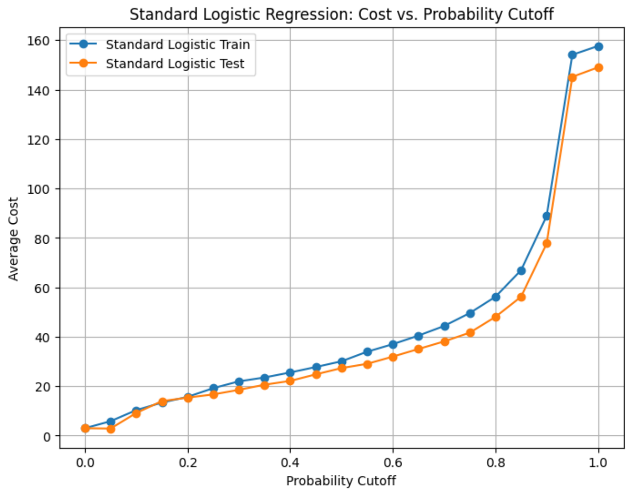

Figure 1: Comparison of Cost vs. Probability Cutoff for Standard Logistic Regression

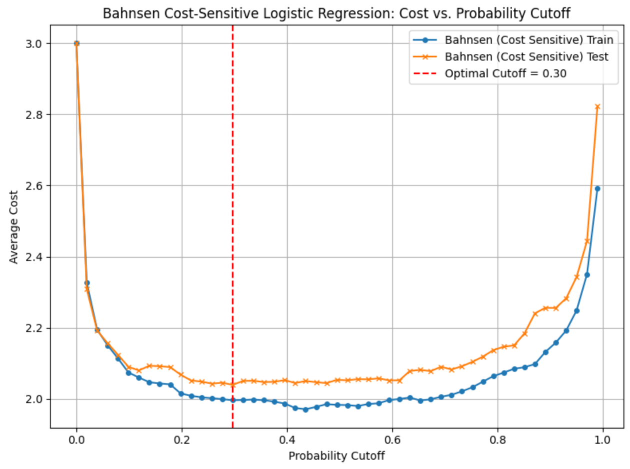

Figure 2: Comparison of Cost vs. Probability Cutoff for Bahnsen Cost-Sensitive Logistic Regression with Genetic Algorithm 1

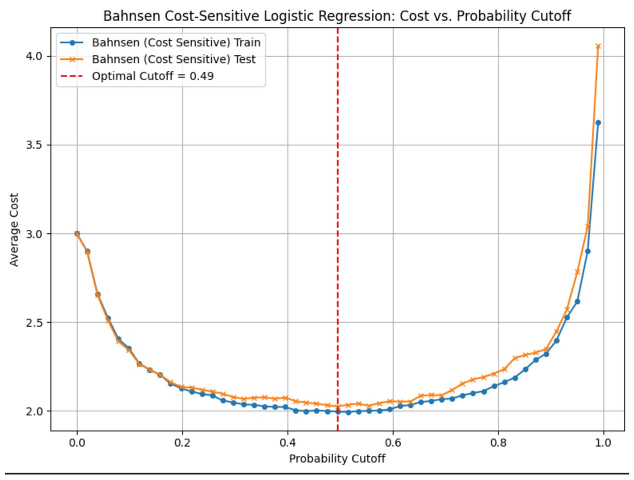

Figure 3: Comparison of Cost vs. Probability Cutoff for Bahnsen Cost-Sensitive Logistic Regression with Genetic Algorithm 2

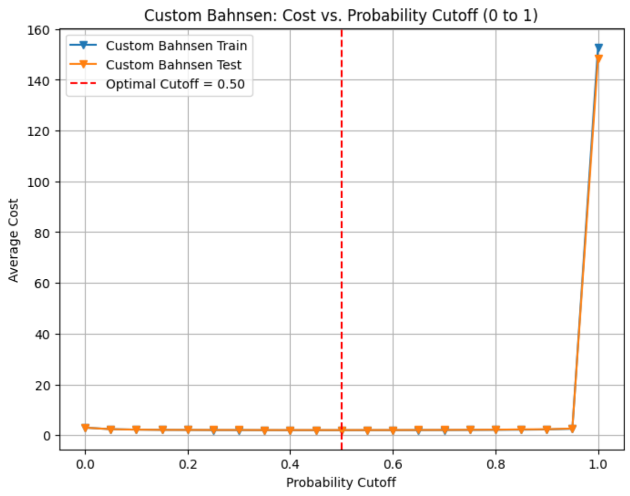

Figure 4: Comparison of Cost vs. Probability Cutoff for Custom Bahnsen approach 0-1 range

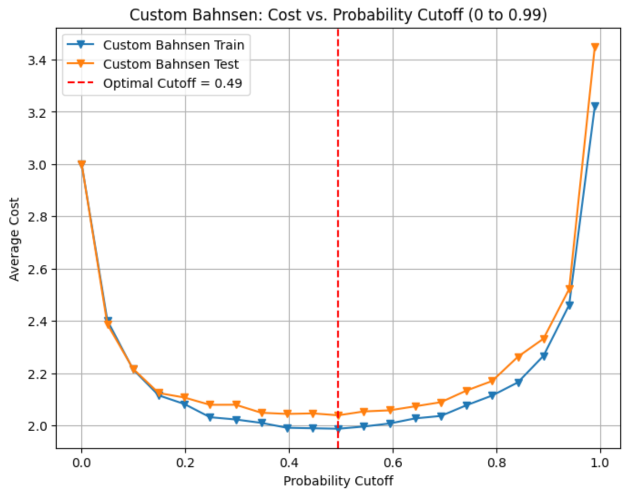

Figure 5: Comparison of Cost vs. Probability Cutoff for Custom Bahnsen approach with 0-0.99 range

7.3 Comparison of Approaches

Approach	Average Training Cost	Average Test Cost
Standard Logistic Regression	5.8349	2.8586
Bahnsen (Custom Manual)	1.9873	2.0387
Bahnsen (Genetic Algorithm 1)	1.9960	2.039
Bahnsen (Genetic Algorithm 2)	1.9959	2.0251

Table 2: Comparison of different approaches based on average training cost and average test cost.

Approach	**Optimal Threshold**	**Total Cost Reduction**
Standard Logistic Regression	0.5	Baseline
Bahnsen(Custom) Optimization	Learned w	Significant Improvement
Genetic Algorithm	Evolved w	Competitive

Table 3: Performance Comparison of Different Approaches

Comparison: Standard Logistic Regression vs. Bahnsen Cost-Sensitive Logistic Regression

Standard Logistic Regression serves as the baseline model in this study. It is trained using scikit-learn's log-loss objective and does not account for varying misclassification costs. While it performs well in terms of accuracy, it applies uniform treatment to all errors. In cost-sensitive settings — especially when false negatives carry instance-specific penalties — this approach often leads to suboptimal outcomes in terms of overall decision cost. Costs are evaluated post-training using a custom function that includes per-instance false negative penalties, revealing that this model often results in higher average costs compared to cost-sensitive approaches.

Bahnsen Cost-Sensitive Logistic Regression modifies the training objective to minimize real-world misclassification cost, incorporating example-specific false negative costs directly into the model. Two variants of this approach are explored:

1. **The Bahnsen Genetic Algorithm (GA)-based Cost-Sensitive Logistic Regression** explicitly integrates cost sensitivity into training by optimizing a cost function that accounts for instance-level false negative costs. Instead of relying on gradients, this method evolves a population of logistic parameters using a Genetic Algorithm. Through mechanisms such as selection, crossover, mutation, and elitism, the model learns weights and bias that minimize the average classification cost directly. It evaluates multiple decision thresholds and selects the one with the lowest test cost. This evolutionary approach offers flexibility in exploring non-convex landscapes and typically achieves lower average misclassification costs than the standard model.

2. **The Custom Bahnsen Cost-Sensitive Logistic Regression with Gradient Descent** provides a deterministic alternative to the GA-based approach. It also minimizes the same cost-sensitive objective function, but does so using gradient-based updates over a fixed number of epochs. The model calculates custom gradients that incorporate example-dependent penalties, enabling more focused updates toward reducing costly false negatives. Similar to the GA version, this model is evaluated across thresholds to identify the one that minimizes

test cost. It generally performs better than standard logistic regression in terms of cost and is more computationally efficient than the GA-based model, though possibly more susceptible to local minima.

8 Conclusion

This study explored cost-sensitive classification using Bahnsen's Logistic Regression and various optimization techniques. The results demonstrate that Bahnsen's approach consistently outperforms standard logistic regression by achieving lower misclassification costs. Bahnsen's Logistic Regression consistently results in lower misclassification costs compared to standard logistic regression. Since Bahnsen's cost function is not convex, traditional optimization methods may struggle. Genetic Algorithms offer an effective approach to optimize this cost function, successfully navigating the complex search space to minimize overall cost.

References

[1] scikit-learn documentation: *Logistic Regression.* `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

[2] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, 2019.

[3] A. C. Bahnsen, D. Aouada, and B. Ottersten, "Example-dependent cost-sensitive logistic regression for credit scoring," *Pattern Recognition*, 2014.

[4] A. C. Bahnsen, *Cost-sensitive classification: Theory and applications*, Ph.D. Thesis, University of Luxembourg, 2015.

[5] A. C. Bahnsen, "Example-Dependent Cost-Sensitive Logistic Regression," arXiv preprint. `https://arxiv.org/pdf/1508.01964.pdf`

[6] SciPy documentation: *Bahnsen(Custom) Optimization.* `https://docs.scipy.org/doc/scipy/reference/optimize.minimize-Bahnsen(Custom).html`

[7] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, 2006.

[8] DEAP library documentation: *DEAP Genetic Algorithm.* `https://deap.readthedocs.io/en/master/`

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[10] C. Elkan, "The Foundations of Cost-Sensitive Learning," *IJCAI*, 2001.

[11] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," *KDD*, 1999.