24th International Conference on Business Information Systems (BIS 2021)

Applications

https://doi.org/10.52825/bis.v1i.61

Published: 23 June 2021

A Novel Example-Dependent Cost-Sensitive Stacking Classifier to Identify Tax Return Defaulters

Sanat Bhargava[1], M. Ravi Kumar[2], Priya Mehta[2], Jithin Mathews[2], Sandeep Kumar[2], and Ch. Sobhan Babu[2]

[1]Indian Institute of Technology Roorkee, Roorkee, India

[2]Indian Institute of Technology Hyderabad, Hyderabad, India

Abstract. Tax evasion refers to an entity indulging in illegal activities to avoid paying their actual tax liability. A tax return statement is a periodic report comprising information about income, expenditure, etc. One of the most basic tax evasion methods is failing to file tax returns or delay filing tax return statements. The taxpayers who do not file their returns, or fail to do so within the stipulated period are called tax return defaulters. As a result, the Government has to bear the financial losses due to a taxpayer defaulting, which varies for each taxpayer. Therefore, while designing any statistical model to predict potential return defaulters, we have to consider the real financial loss associated with the misclassification of each individual. This paper proposes a framework for an example-dependent cost-sensitive stacking classifier that uses cost-insensitive classifiers as base generalizers to make predictions on the input space. These predictions are used to train an example-dependent cost-sensitive meta generalizer. Based on the meta-generalizer choice, we propose four variant models used to predict potential return defaulters for the upcoming tax-filing period. These models have been developed for the Commercial Taxes Department, Government of Telangana, India. Applying our proposed variant models to GST data, we observe a significant increase in savings compared to conventional classifiers. Additionally, we develop an empirical study showing that our approach is more adept at identifying potential tax return defaulters than existing example-dependent cost-sensitive classification algorithms.

Keywords: goods and services tax, tax evasion, example-dependent cost-sensitive stacking classifier, example-dependent cost-sensitive ANNs,Benford's analysis, social network analysis, cosine similarity.

1 Introduction

Taxes can be classified into direct taxes, which are payable directly to the government (Eg. Income tax). These taxes cannot be transferred to any other third party, and indirect taxes, which can be shifted to a third party by the entity that is levied the tax (Eg. VAT, excise duty). The Goods and Services Tax (GST) system is an indirect taxation system introduced in India in July 2017. This paper proposes a methodology to predict potential tax return defaulters for the GST system [1].

1.1 Working of the GST system

For demonstration purposes, we take a fictitious ornament manufacturer as an example, and 10% as the GST rate levied at every step (See Figure 1). Note that throughout the paper, we

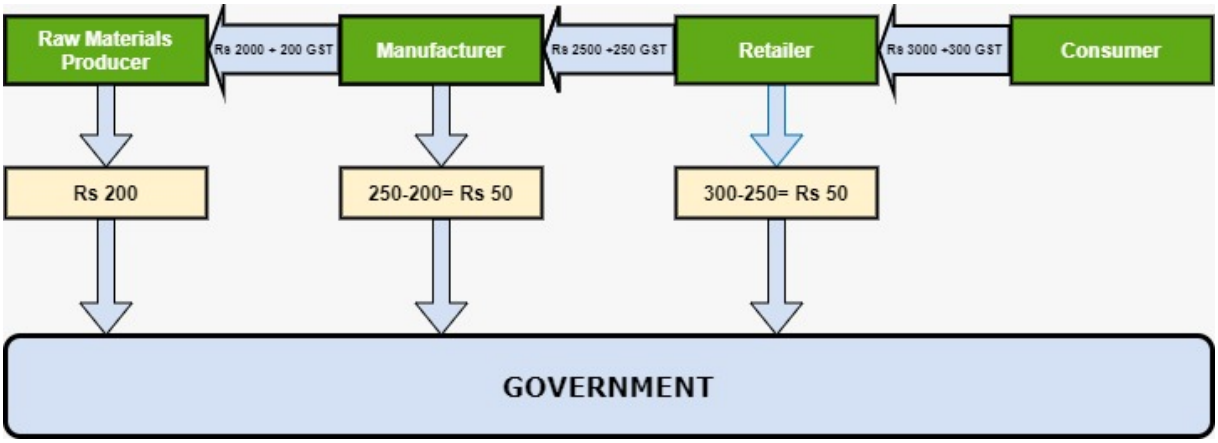

Figure 1. Flow of Tax in GST.

represent currencies in "Indian National *Rupees (INR)*," denoted henceforth as "Rs.". Assume the manufacturer purchases raw material worth Rs. 2000 and is hence levied a GST of Rs. 200 (10% of 2000). Suppose, the manufacturing process adds a value of Rs. 500 to the ornament. Hence, the value of the ornament is now Rs. 2500. Now, the total tax levied on the sales of this ornament to the retailer is Rs. 250 (10% of 2500). By setting off the tax he had already paid at the time of purchasing the raw material, the GST payable to the manufacturer is Rs. 50 (tax collected-tax already paid), $i.e.$, Rs. 50 (250-200). The retailer adds his margin of Rs. 500 increasing the total value to Rs. 3000 and sells it to the consumer for Rs. 3000, and the consumer is levied Rs. 300 as tax for the purchase. Similarly, the retailer is liable to pay a GST of Rs. 50 (tax collected - tax already paid), $i.e.$, Rs. 50 (300-250) at the time of purchasing the ornament from the manufacturer. Finally, the GST received by the Government is Rs. 300, which is completely borne by the end consumer.

1.2 Motivation for this work

In the GST system, taxpayers are required to file their tax returns once a month. Defaulting on filing tax returns has the following consequences: First, defaulters have enough time at their disposal to manipulate their records; second, the penalty imposed by the Government is negligible compared to the going interest rates of the market, and therefore not an effective deterrent. Lastly, having movable assets is always beneficial to businesses involving large monetary transactions. This work's motivation is to construct a classification model to identify potential return defaulters and implement preventive measures such as sending emails, SMS, or physically visiting their business premises. We are working with the Government of Telangana, India, and are using their data for analysis and building models to increase tax returns compliance.

To attack a classification problem such as the one presented here, one would be inclined to use conventional cost-insensitive classification algorithms such as Logistic Regression, K-Nearest Neighbors, $etc.$, to design the classifier. However, this presents us with a significant problem. A conventional classifier assigns equal misclassification costs for every example. In practice, however, the misclassification cost associated with classifying a genuine taxpayer as a return defaulter might vary significantly from classifying a return defaulter as an honest taxpayer. Similarly, the misclassification cost associated with misclassifying a return defaulter as a genuine taxpayer would vary for individual taxpayers based on their respective turnovers. Hence, there is a trade-off between choosing a model with better cost savings and choosing a model with better performance. To deal with this trade-off better, we propose four variants of an *example dependent cost-sensitive stacking classifier*. In a later section, we show that our Proposed Approach (PA) is adept at identifying potential tax return defaulters for the upcoming month with high accuracy. The approach that we have adopted in this paper can be generalized

to any indirect taxation system used globally.

Our paper is structured as follows. In Section 2, we brief on existing works that are related to ours. Section 3 describes the data set and the feature extraction techniques used for designing the model. In Section 4, we describe the framework for the proposed variants of the example-dependent cost-sensitive stacking classifier. Section 5 discusses the performance of the PA on the data set and compares it with some example-dependent cost-sensitive and conventional cost-insensitive classifiers commonly in use. Finally, in Section 6, we provide concluding remarks for our work.

2 Related Work

In [2], Jasmien Lismont et al. used social network analysis concepts to develop a model to predict tax avoidance by including a wider variety of network features. In [3], Bianchi et al. use network measures of centrality to show that the taxpayers who collaborate with better-connected auditors are likely to have lower effective tax rates. In [4], Veronique Van Vlasselaer et al. worked on identifying entities that indulge in social security fraud by assigning a time-dependent exposure score to each business entity based on its involvement with known fraud business entities in the social network. In [5], Yusuf Sahin and Ekrem Duman have built classification models for detecting credit card fraud using Logistic Regression and Artificial Neural Networks, one of the first studies to compare the performance of Logistic Regression and ANNs for this use case. In [6], Charles X. Ling and Victor S. Sheng showed that cost-sensitive learning is a common approach to solve data imbalance problems. In [7], A. C. Bahnsen et al. proposed an example dependent cost matrix for credit scoring. They proposed a cost function that introduces the example dependent costs into logistic regression. In [8], A.C Bahnsen et al. propose a framework for cost-sensitive classifiers, including Cost-Sensitive Decision Trees, Cost-Sensitive Random Forests, and ensembles of cost-sensitive models based on techniques such as majority voting and stacking Cost-Sensitive Logistic Regression generalizers. In [9], David H. Wolpert introduces Stacking or Stacked Generalization, an ensemble learning technique that aims to deduce generalizers' biases for the training set provided. In [10], Matjaz Kukar and Igor Kononenko designed a cost-sensitive analog for ANNs, with their study being the first to do so.

3 Data Description and Feature Extraction

3.1 Benford's law

Benford's law is a mathematical method for identifying fraud [11], [12],[13] in naturally-occurring numbers, considering that these numbers are neither highly constrained nor purely random. This law posits that the percentage of numbers with the first digit as $k \in \{1, 2, ..., 9\}$ follows the formula $log_{10}(1 + 1/k)$.

3.2 Description of the data set

We now proceed to briefly describe the data used to design our models. We were provided two types of data sets to prepare our models, namely: GSTR-1 data and month-wise GST returns data.

3.2.1 GSTR-1 Data

GSTR-1 is a financial statement that every taxpayer is required to submit monthly. This statement consists of details of all outward supplies, $i.e.$, all sales done during the month corresponding to this statement. A fictitious sample of this statement is given in Table 1. Every row in Table 1 corresponds to one transaction. The data set contains several millions of such rows. The actual statement contains more information, such as the tax rate, the number of goods sold

etc.

S.No.	Month	Seller	Buyer	Invoice Number	Amount (Rs)
1	Jul 2017	A	D	AB323	13000
2	Aug 2017	B	C	ZX362	16000
3	Sep 2017	B	A	BC9414	14490

Table 1. GSTR-1 Data

3.2.2 Monthly GST Returns Data

Table 2 contains a few select fields of GST returns data. Each row in this table corresponds to the monthly returns filed by a taxpayer. ITC (Input tax credit) is the amount of tax paid by the taxpayer during purchases of services and goods. The output tax is the amount of tax collected by the taxpayer during the sales of services and products. The taxpayer has to pay the Government the difference between the output tax and ITC, $i.e.$ (output tax – ITC). The actual dataset consists of much more information like tax payment method, return filing data, international exports, exempted sales, and sales on RCM (Reverse Charge Mechanism).

S.No.	Firm	Month	Purchases	Sales	ITC	Output Tax
1	D	Jul 2017	170000	250000	17000	25000
2	C	Oct 2017	230000	300000	11500	30000
3	F	Dec 2017	350000	450000	17500	45000

Table 2. Monthly GST Returns Data

3.3 Creation of Network of taxpayers

In this model, we have attempted to quantify the amount of interaction between taxpayers. To compute this independent variable, we created a weighted, directed graph (social network) in which each vertex (node) corresponds to a taxpayer. The weight assigned to the vertices is the average tax paid per month [$ATPM$] associated with the corresponding taxpayer from July 2017 to November 2019. Vertex weights have been normalized using min-max normalization. We have utilized the month-wise GST Returns Data explained in Table 2 to compute each taxpayer's vertex weights. We have placed a weighted, directed edge from taxpayer a to taxpayer b, where the weight of the edge is the amount of sales done by taxpayer a to taxpayer b during the period July 2017 to November 2019. Similar to the vertex weights, the edge weights have been normalized using min-max normalization. For the same, we have used the GSTR-1 data explained in Table 1. This graph captures the scale of interaction between taxpayers.

3.4 Feature Extraction

3.4.1 Ratio

This is the variable extracted from the weighted, directed graph defined in subsection 3.3. This graph captures the degree of interaction and the monetary transactions between taxpayer b and other taxpayers. This variable captures the influence of other taxpayers on b. If b has close ties with taxpayers who are known tax return defaulters, they will influence b not to file GST returns and vice-versa [2]. Let B be the set of all vertices corresponding to defaulters (who have filed at most $1/4^{th}$ of their returns) and Y be the set of all vertices corresponding to taxpayers who have filed their returns in time (who have filed more than $3/4^{th}$ of their returns in time).

- $b_{11} = \sum_{v \in B} \frac{\omega(v) * \omega(vb)}{\omega(v) + \omega(vb)}$, where $\omega(v)$ is the weight of vertex v and $\omega(vb)$ is the weight of directed edge vb
- $b_{12} = \sum_{v \in B} \frac{\omega(v) * \omega(bv)}{\omega(v) + \omega(bv)}$.
- $b_{21} = \sum_{v \in Y} \frac{\omega(v) * \omega(vb)}{\omega(v) + \omega(vb)}$, where $\omega(v)$ is the weight of vertex v and $\omega(vb)$ is the weight of

directed edge vb

- $b_{22} = \sum_{v \in Y} \frac{\omega(v) * \omega(bv)}{\omega(v) + \omega(bv)}$.
- $Ratio = \frac{b11+b12}{b21+b22}$.

3.4.2 Filed

This is the *dependent variable* in the model with a binary outcome. This variable gives the GST return filing status (whether the taxpayer has filed returns in-time or not) of the taxpayer b for December 2019. $Zero$ denotes returns were filed in-time (negative class) and, one denotes returns were not filed in-time (positive class).

3.4.3 Not Filed Count

This is the number of GST returns not filed in-time before the due date of the corresponding month by b from July 2017 to November 2019.

3.4.4 Division-Name

The state of Telangana is divided into 12 geographic divisions for simplification of administration works. This independent variable gives the geographic location in which b is located.

3.4.5 ATPM

This is the average tax per month paid by b. We included square, cube, log and the square root terms of the $ATPM$ in the model as the relation between $ATPM$ and *Log of Odds* of the Filed variable is a polynomial.

3.4.6 MAD Value

It is the Mean absolute deviation value of the first digit Benford's Law (Section 3.1) on sales transactions of b.

3.4.7 Seasonality

Case A: Retailers selling a single commodity:
In an actual market, the annual revenue of some businesses may show a seasonal trend. For example, for a taxpayer involved in the yogurt business, one might observe higher revenues in the peak summer (May-June in India) and lower revenues in the winter (November-February in India). To quantify this seasonality, we have calculated the cosine similarity between the *output tax* of each taxpayer selling a particular type of commodity and the mean of the *output tax* of all taxpayers selling that commodity.

$$similarity(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

Here A denotes a vector of the *output tax* for every month for each taxpayer selling a particular type of commodity, and B denotes the vector of the mean of the *output tax* of all taxpayers selling that commodity for every month.

Case B: Retailers selling multiple commodities:
In a more general case, a retailer might generate revenue by selling multiple commodities, and each commodity might have its own seasonal trend. Consider a retailer sells commodities from a set $I = A, B, C, D$. For this retailer, we calculate the seasonality parameter as follows

$$Seasonality = \sum \omega_i s_i, \forall i \in I$$

Here , ω_i weight associated with each commodity i, defined as
$\omega_i = \frac{\textit{Total revenue generated by sales of commodity i (for that retailer)}}{\textit{Total revenue generated by that retailer}}$,
$s_i =$ *Similarity of commodity i for that retailer*. Here, similarity is calculated as in case A.

3.5 Class Imbalance

The ratio of genuine taxpayers to defaulters was noted to be 0.22, hence, the use of sampling techniques was not deemed necessary.

4 Framework for Example Dependent Stacking Classifier

In problems such as the identification of tax return defaulters, it is of paramount importance to minimize the government's losses on account of the defaulters. For this task, an *example-dependent cost-sensitive classifier* would be the most prudent choice, as opposed to cost-insensitive classifiers [7]. Intuitively, one can deduce that a cost-sensitive classifier would minimize the total cost (or increase the total savings), compromising overall model performance. On the other hand, a cost-insensitive classifier would aim for optimal model performance while leading to higher losses to the government. It follows that there is a trade-off between higher cost savings and better model performance. To alleviate this problem, we propose a novel framework for example-dependent cost-sensitive stacked classifiers that give a competitive model performance and increased savings compared to cost-insensitive classifiers.

4.1 Stacked Generalizers and General Framework

Stacked Generalization or stacking is an ensemble learning technique that aims to improve upon the performance of its constituent generalizers by deducing the biases of each of the individual generalizers. However, stacked generalizers do not always perform better than individual generalizers, and their efficacy depends on the choice of generalizers. While there is no defined architecture for a stacked generalizer, it is observed that stacking is most effective when the choice of individual generalizers is as diverse as possible.
We propose a framework for a two-level stacked generalizer constructed as follows: The first level G_1 consists of conventional cost-insensitive classifiers to deduce the biases of classifiers on the input space, such that,

G_1 = {K-Nearest Neighbors Classifier, XGBoost Classifier, Random Forest Classifier, Logistic Regression, Artificial Neural Network, AdaBoostClassifier}.

The second level G_2, consists of the meta-generalizer, which generalizes on the second space, consisting of the predictions of G_1. We consider four choices of generalizers for G_2, which gives rise to the following four variants:

- **Variant A** (G_2=Cost Sensitive Decision Tree Classifier[8]),
- **Variant B** (G_2=Cost-Sensitive Bagging Classifier[8]),
- **Variant C** (G_2=Cost-Sensitive Random Forest Classifier[8]),
- **Variant D** (G_2=Cost-Sensitive ANN [10]).

The choice of the meta-generalizers was dictated by the savings score and the AUC-ROC score (Section 5.3.1). The models with the highest savings score and highest AUC-ROC score were chosen as meta-generalizers.

4.2 Meta Learners

4.2.1 Cost function

Let S be a set of N examples $\mathbf{x}_i$, where each example is represented by the augmented feature vector with associated costs $\mathbf{x}_i^* = [\mathbf{x}_i, C_{TP_i}, C_{FP_i}, C_{FN_i}, C_{TN_i}]$ and labelled using the class label y_i. A classifier f, which generates the predicted label c_i for each example i is trained using the set S. Then the cost of using f on $\mathbf{x}_i^*$ is calculated by

$$\begin{aligned} Cost(f(\mathbf{x}_i^*)) =& y_i(c_i C_{TP_i} + (1-c_i)C_{FN_i}) \\ &+ (1-y_i)(c_i C_{FP_i} + (1-c_i)C_{TN_i}), \end{aligned} \tag{1}$$

and the total cost defined as

$$Cost(f(S)) = \sum_{i=1}^{N} Cost(f(\mathbf{x}_i^*)). \tag{2}$$

C_{TP_i}, C_{FN_i}, C_{FP_i}, C_{TN_i} are defined in Table 3.

	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	C_{TP_i}	C_{FP_i}
Predicted Negative $c_i = 0$	C_{FN_i}	C_{TN_i}

Table 3. Cost Matrix

4.2.2 Variant A

For variant A, we have G_1 as defined above, and we use G_2= Cost-Sensitive Decision Tree Classifier (CSDT) [8]. In CSDTs, instead of using traditional splitting criteria such as Gini, entropy, or misclassification, the cost as defined in (1) is calculated for each node, and the gain of using each split is evaluated as the decrease in the total cost of the algorithm. The cost-based impurity measure is defined by comparing the costs when all the examples in a leaf are classified as negative and as positive,

$$I_c(S) = min\left\{Cost\left(f_0\left(S\right)\right), Cost\left(f_1\left(S\right)\right)\right\}.$$

Then, using the cost-based impurity, the gain of using the splitting rule $(\mathbf{x}^j, l^j)$, that is the rule defined as splitting the set S on feature $\mathbf{x}^j$ on value l^j, is calculated as

$$Gain(\mathbf{x}^j, l^j) = I_c(S) - \frac{|S^l|}{|S|} I_c(S^l) - \frac{|S^r|}{|S|} I_c(S^r),$$

where $S^l = \{\mathbf{x}_i^* | \mathbf{x}_i^* \in S \wedge x_i^j \leq l^j\}$,$S^r = \{\mathbf{x}_i^* | \mathbf{x}_i^* \in S \wedge x_i^j > l^j\}$, and $|\cdot|$ denotes the cardinality. Afterward, a decision tree is grown using the cost-based gain measure until no further splits can be made. After the tree is constructed, it is pruned by using a cost-based pruning criteria

$$PC_c = Cost(f(S)) - Cost(f^*(S)),$$

where f^* is the classifier of the tree without the pruned node.

4.2.3 Variant B

For variant B, we have G_1 as defined above, and we use G_2= Cost-Sensitive Bagging Classifier (CSB). Bagging or Bootstrap Aggregating is an ensemble technique that involves fitting base estimator(s) to random samples of the data set. The individual predictions are then aggregated using *majority voting* or *weighted average* to form a final prediction. To build our CSB, we have used the CSDTs mentioned above as base estimators and aggregated the individual predictions using majority voting [8].

4.2.4 Variant C

For variant C, we have G_1 as defined above, and we use G_2= Cost-Sensitive Random Forest Classifier (CSRF). Cost-Sensitive Random Forest Classifiers are ensemble classifiers that work by creating multiple CSDTs and outputting the mode of the predictions made by the CSDTs as the final prediction of the ensemble classifier [8].

4.2.5 Variant D

Finally, we have implemented a Cost-Sensitive analog for an Artificial Neural Network [10]. To design our Cost-Sensitive ANN Classifier (CSANN), we have used the ReLU function as the activation function for the hidden layers and the logistic (sigmoid) function for the output layer. We have used equation (1) as the loss function for the neural network to incorporate the example-dependent cost-sensitive losses.

5 Experimental Results

5.1 Software Used

All the models in this work have been designed using **Python** as it is a high-level, open-source language with an extensive library ecosystem. Python can also handle large amounts of data very well.

5.2 Cost Matrix

Table 3 gives different miss-classification costs of a given taxpayer.

- **True-negative cost ($\mathbf{C_{TN}}$)** is zero. We would not incur any cost for classifying an in-time return filer (actual class zero) as an in-time return filer (predicted class zero).
- **True-positive cost ($\mathbf{C_{TP}}$)** is the expenditure towards sending SMS, calling the taxpayer and other preventive measures and the cost of associated manpower. This cost is the same for all taxpayers whose actual class is one and predicted class is one. This cost is Rs. 150.
- **False-positive cost ($\mathbf{C_{FP}}$)** is the expenditure towards sending SMS, calling the taxpayer and other preventive measures and the cost of associated manpower. This cost is the same for all taxpayers whose actual class is zero and predicted class is one. This cost is also Rs. 150.
- **False-negative cost ($\mathbf{C_{FN}}$)** depends on the $ATPM$ of each taxpayer and the expected number of days of delay in filing return by a taxpayer. This is given by $\frac{ATPM * \textit{expected number of days of delay} * 18}{36500} * 3 + 100$.

Here $\frac{ATPM * \textit{expected number of days of delay} * 18}{36500}$ is the loss incurred due to late filing of return, where interest rate is 18%. This cost is different for every taxpayer as the $ATPM$ and expected number of days of delay in filing return may vary for each individual taxpayer. We have multiplied this loss by three times and added 100 to it, in order to deter a defaulter from becoming a chronic defaulter.

5.3 Performance of Proposed Variants

In this section, we have compared the four proposed variants (variants A, B, C, and D) on tax return data vis-à-vis each other. The models have been compared on the following metrics:

5.3.1 Savings score

The savings score is defined as the relative improvement in cost using a classifier $f(S)$, compared to the cost of classifying all entries as class one or class zero, whichever is lesser [7].

$$Savings(f(S)) = \frac{Cost(f(S)) - Cost_l(S)}{Cost_l(S)}.$$

5.3.2 Balanced Accuracy Score

The balanced accuracy score is a metric for models trained on imbalanced data sets, which avoids inflated performance metrics due to the abundance of one class (in a binary classification

problem). It is defined as follows:

$$\textit{Balanced accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right).$$

5.3.3 Recall

Recall or Recall score refers to the fraction of relevant records correctly classified by the models. It is defined as follows:

$$Recall = \frac{TP}{TP+FN}.$$

In the context of this paper, the recall score is the fraction of tax return defaulters correctly identified by the model.

5.3.4 F1-Score

The F1-Score is defined as the harmonic mean of the precision and recall of a model. Thus,

$$\textit{F1-Score} = 2 * \left(\frac{precision * recall}{precision + recall}\right).$$

The comparative performance of the four variants is summarized in the Table 4. From the four variants, we propose Variant D (G_2=Cost-sensitive ANN) to be our proposed approach (PA) for this data set as it is the most adept at correctly predicting tax return defaulters, with the highest savings score and the highest AUC-ROC predicted on the train and test data set among the four variants.

Proposed Models	Savings Score		Balanced Accuracy		F1-Score		Recall		AUC-ROC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Variant A	0.536	0.174	83.36%	82.23%	0.76	0.75	0.95	0.94	0.92	0.92
Variant B	0.535	0.530	83.90%	83.82%	0.77	0.77	0.91	0.90	0.93	0.94
Variant C	0.583	0.572	85.58%	85.94%	0.83	0.83	0.90	0.91	0.94	0.94
Variant D	0.520	0.582	85.14%	85.21%	0.79	0.79	0.94	0.95	0.94	0.93

Table 4. Performance of variants

Proposed Models	Savings Score		Balanced Accuracy		F1-Score		Recall		AUC-ROC	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
ANN	0.272	0.242	84.11%	83.40%	0.69	0.66	0.76	0.74	0.93	0.93
CSANN	0.393	0.440	84.31%	84.29%	0.84	0.84	0.84	0.84	0.93	0.93
CSDT	0.610	0.600	81.52%	79.00%	0.84	0.84	0.78	0.71	0.93	0.94
CSB	0.557	0.633	82.00%	78.50%	0.83	0.83	0.83	0.83	0.94	0.94
CSRF	0.495	0.517	79.15%	83.34%	0.52	0.62	0.93	0.91	0.91	0.92
Proposed Approach	0.520	0.582	85.14%	85.21%	0.79	0.79	0.94	0.95	0.94	0.93

Table 5. Performance of PA compared to existing algorithms

5.4 Performance of Proposed Approach (PA) with existing algorithms

In this section, we have compared our proposed approach's performance with some cost-sensitive algorithms mentioned in [8]. Additionally, we compare the performance of the PA with a cost-sensitive ANN [10]. We have also compared the performance of our PA with a cost-insensitive ANN. We have chosen a cost-insensitive ANN as it gave the most promising results

among various cost-insensitive algorithms we experimented with, including, KNNs, Random Forests, XGBoost Classifier, AdaBoostClassifier, and Logistic Regression. The performance has been compared using the same metrics described in section 5.3. The results have been summarized in Table 5.

5.5 Model Validation for PA

5.5.1 Confusion and Cost Matrices

Tables 6 and 7 are the training and the testing confusion matrices for the PA. Tables 8 and 9 are the training matrix and the testing cost matrix for the PA. These give the true-positive cost, false-negative cost, true- negative cost, and false-positive cost of both the training and testing data sets.

	Predicted 0	Predicted 1
Actual 0	9842	2946
Actual 1	196	2742

Table 6. PA Train Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	3320	1006
Actual 1	48	930

Table 7. PA Test Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	0	441900
Actual 1	276796	411300

Table 8. PA Train Cost Matrix

	Predicted 0	Predicted 1
Actual 0	0	150900
Actual 1	68217	130200

Table 9. PA Test Cost Matrix

5.5.2 Training and Testing ROC Curves

Training and testing ROC curves for the PA are given in Figure 2 and 3. The AUC value of training ROC curve is 0.94 and AUC value of testing ROC curves is also 0.93. From these values, one can conclude that the model is neither under-fitting nor over-fitting.

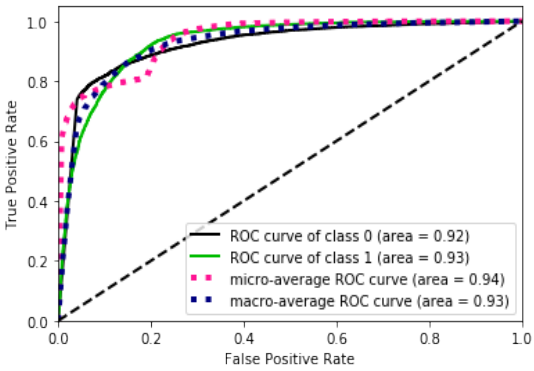

Figure 2. PA ROC on Train.

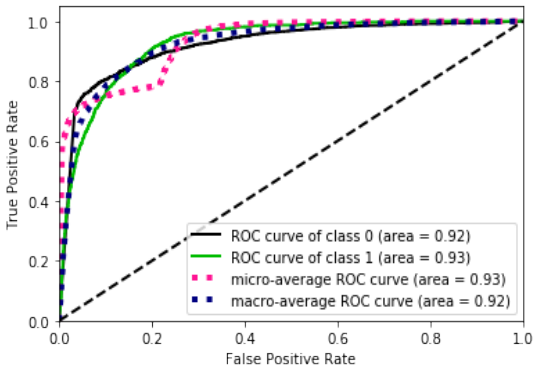

Figure 3. PA ROC on Test.

5.5.3 Savings score

To measure an example-dependent cost-sensitive algorithm's performance, we use the savings score (Section 4.3). As observed in Table 5, the savings score for the PA is 0.520 and 0.582 for the training and test sets, respectively. Since the values of the savings score for the training and testing set are reasonably high and almost similar, we can conclude that our PA is performing well.

6 Conclusion

In this paper, We propose a framework for example-dependent cost-sensitive stacked generalization comprising four variant models. We show that our Proposed Approach (PA) outperforms commonly used example-dependent cost-sensitive classifiers. We use our PA to predict whether a given taxpayer is a potential tax return defaulter or not for the upcoming month. While this framework was designed on the GST returns data for Telangana, it can be generalized to predict potential tax return defaulters using any of the four proposed variants depending on their performance, for any indirect taxation system around the world.

References

[1] S. Dani, "A research paper on an impact of goods and service tax (gst) on indian economy," *Business and Economics Journal*, vol. 07, Jan. 2016. DOI: 10.4172/2151-6219.1000264.

[2] J. Lismont, E. Cardinaels, L. Bruynseels, S. D. Groote, W. Lemahieu, and J. Vanthienen, "Predicting tax avoidance by means of social network analytics," *Decision Support Systems*, vol. 108, pp. 13–24, 2018.

[3] P. A. Bianchi and M. Minutti-Meza, "Professional networks and client tax avoidance: Evidence from the italian statutory audit regime," *SSRN Electronic Journal*, Jan. 2016. DOI: 10.2139/ssrn.2601570.

[4] V. V. Vlasselaer, L. Akoglu, T. Eliassi-Rad, M. Snoeck, and B. Baesens, "Guilt-by-constellation: Fraud detection by suspicious clique memberships," in *2015 48th Hawaii International Conference on System Sciences*, 2015, pp. 918–927.

[5] Y. Sahin and E. Duman, "Detecting credit card fraud by ann and logistic regression," in *2011 International Symposium on Innovations in Intelligent Systems and Applications*, 2011, pp. 315–319.

[6] C. Ling and V. Sheng, "Cost-sensitive learning and the class imbalance problem," *Encyclopedia of Machine Learning*, Jan. 2010.

[7] A. C. Bahnsen, D. Aouada, and B. Ottersten, "Example-dependent cost-sensitive logistic regression for credit scoring," in *2014 13th International Conference on Machine Learning and Applications*, 2014, pp. 263–269.

[8] A. C. Bahnsen, D. Aouada, and B. Ottersten, *Ensemble of example-dependent cost-sensitive decision trees*, 2015. arXiv: 1505.04637 [cs.LG].

[9] D. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241–259, Dec. 1992. DOI: 10.1016/S0893-6080(05)80023-1.

[10] M. Kukar and I. Kononenko, "Cost-sensitive learning with neural networks," in *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98*, John Wiley & Sons, 1998, pp. 445–449.

[11] M. L. J. Nigrini Mark J., "The Use of Benford's Law as an Aid in Analytical Procedures," *Auditing: A journal of practice & theory*, vol. 41, p. 52, 1997.

[12] A. Asllani and M. Naco, "Using Benford¡¯s Law for Fraud Detection in Accounting Practices," *Journal of Social Science Studies*, vol. 2, no. 1, pp. 129–143, Jan. 2015. [Online]. Available: https://ideas.repec.org/a/mth/jsss88/v2y2015i1p129-143.html.

[13] C. Durtschi, W. Hillison, and C. Pacini, "The effective use of benford's law to assist in detecting fraud in accounting data," *J. Forensic Account*, vol. 5, Jan. 2004.