

A Semi-Supervised Approach to Anomaly Detection for Tax Compliance

Leo Corman¹, Jason Bono¹, Catherine Acton¹,
Danielle Gewurz¹, Evan Schulz²

¹Deloitte Consulting LLP, 1919 N. Lynn Street, Arlington, VA 22209

²Internal Revenue Service Research, Applied Analytics, and Statistics, 1111 Constitution Avenue NW, Washington, DC 20224

Abstract

A semi-supervised autoencoder model is applied to predict anomalies in tax form data consisting of hundreds of sparse, irregularly distributed features. Historical tax data generally contains a small number of labeled examples (i.e., returns with known compliance outcomes) and a much larger number of unlabeled examples (i.e., returns with unknown compliance status). Compliance outcomes can be measured by various success metrics; these metrics tend to have right-tailed distributions, with higher values indicating outcomes that are more beneficial to the IRS. Building on recent literature related to semi-supervised anomaly detection and our previous work applying unsupervised autoencoders to tax data, we introduce new techniques for incorporating labeled data into model training and leveraging both binary (i.e., compliant vs non-compliant) and continuous (i.e., success metrics) outcomes to update the model. These techniques provide flexibility regarding the level of influence that the labeled data has on the training process – ranging from no influence at all to near total influence – while addressing the sparsity of the data. We also apply novel ensemble methods to improve detection of anomalies. Testing across different datasets and population segments shows favorable performance for the semi-supervised autoencoder compared to existing operational models.

Key Words: Sparse data, semi-supervised, deep learning, autoencoder, deep anomaly detection, ensemble

1. Introduction

The Internal Revenue Service (IRS) dedicates extensive resources to manual detection of data errors in form submissions. The manual review process, where employees identify anomalous form values by hand, is both time-consuming and labor intensive. Replicating the manual review process with a model-based approach frees up scarce resources, ensures consistency, and potentially reduces certain types of biases. The goal is to detect anomalous data across over 100 million tax returns with accuracy through a model-based approach.

Challenges exist due to both the volume and characteristics of the data. Evaluating over 100 million tax returns means that manual methods are impractical at scale. Because tax filers populate only lines that apply to them, most forms can be represented as a sparse matrix. The natural sparsity of features can complicate analysis and must be handled appropriately with a specialized approach. Additionally, measures of error on a return tend to be skewed, since errors tend to be biased in the direction that favors the tax filer. An effective anomaly detection method should balance finding the most severe errors with achieving high levels of accuracy in terms of binary compliance outcomes (i.e., compliant vs non-compliant).

Fully supervised approaches are less desirable for several reasons. First, available labeled data is limited, and creating additional labeled data comes at a prohibitive cost to the IRS. Furthermore,

data patterns and signatures of anomalies may change over time for the population as tax filer data varies from year to year. Unsupervised covariance methods have previously been studied by Parker *et al.*^{1,2} to detect anomalies on sparse form data, and unsupervised autoencoder methods for tax data have been studied by Acton *et al.*³. We propose a *semi-supervised* autoencoder model as an alternative approach to this anomaly detection problem.

2. Autoencoder Approach

2.1 Background

An autoencoder is a deep learning technique that uses neural networks to learn compressed data encodings by capturing the underlying relationships in a population of data. It forces the reduction of input data down to a much smaller size representation of the data, and then attempts to reconstruct the data input with minimal loss.

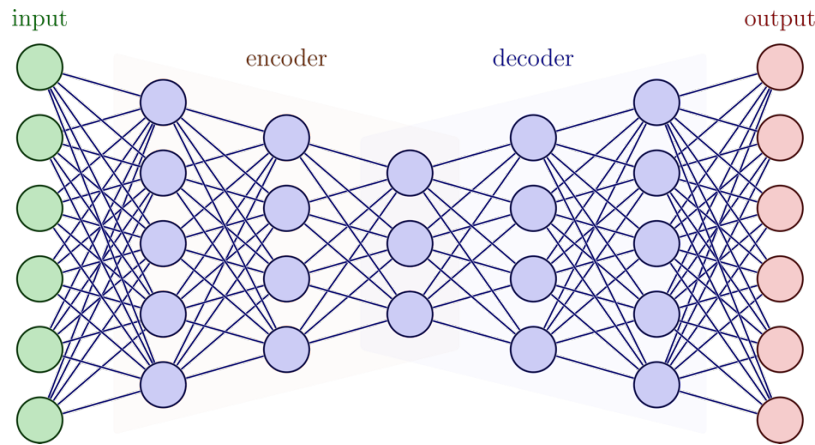


Figure 1: High-Level Depiction of an Autoencoder

For an autoencoder to learn the reconstruction of inputs, the training data is passed through the network many times; each time, the model assesses its ability to reconstruct the input data (loss function) and makes corresponding adjustments to improve reconstruction.

Autoencoders have the potential to consider unspecified and non-linear relationships and require no a priori knowledge of which line items are critical or the “preferred” direction of specific lines (i.e., whether errors in the positive or negative direction for a given line are more favorable for the tax filer). Additionally, the initial layers of an autoencoder model can effectively execute much of the basic data preprocessing, saving additional time and experimentation in research.

When designed for anomaly detection in the tax space, autoencoders can learn from a theoretically “normal” set of data (i.e., compliant tax forms) to identify deviations from “normal” behavior (i.e., non-compliance). The autoencoder learns low-dimensional, condensed representations of forms based on the patterns and relationships in the input data. The autoencoder then reconstructs those forms from the condensed representation by first providing predicted values for each line item. The differences between line item predictions and actual line item values are evaluated across all line items to derive a return-level score representing the overall anomalousness of a form.

2.2 Previous Autoencoder Research in the Problem Space

Acton *et al.*³ demonstrated the potential for applying unsupervised autoencoder methods for tax form anomaly detection, focusing on methods drawn from Merrill and Eskandarian⁴ for addressing the “contamination” of training data. Contamination refers to the fact that unlabeled training data consists of mostly “normal” forms but is also contaminated with some unknown amount of “not normal” forms; if the model also learns to reconstruct the anomalous forms effectively, then the

model is not helpful for anomaly detection. The main modifications tested were the use of a Percentile Loss Threshold (PLT) during training to prevent anomalous returns from disproportionately contributing to parameter updates and Cumulative Error Scoring (CES) to account for the full history of reconstruction errors for each return (with the assumption that the model will take longer to learn anomalous patterns).

The research described in the remaining sections builds on the work in Acton *et al.*³ in several ways, including the following:

- **Use of semi-supervised learning:** we incorporate labeled data into the model training process, rather than modeling as a fully unsupervised problem.
- **Use of ensemble methods:** we combine scores from multiple autoencoders to maximize performance.
- **Performance generalization:** we evaluate autoencoder performance on a wider variety of tax data.
- **Additional data elements and data processing:** we include additional line items and test new data processing steps (e.g., cube root).

Many aspects of the previous research were carried forward, including sparse mean squared error (MSE) (defined below), model architecture, and use of multiple “trials” for each parameter combination (also discussed below). Other aspects were de-emphasized (PLT) or removed (CES) in the more recent research. Note that while PLT has a straightforward translation to the semi-supervised setting (i.e., only applying PLT to the losses of unlabeled returns), CES cannot easily be applied to new data (i.e., data that has not been part of the model training process).

2.3 Semi-Supervised Training

Semi-supervised approaches are useful when dealing with a large amount of unlabeled data and a small amount of labeled data, helping the model learn the overall distribution of the data (i.e., not overfitting to a small sample of labeled data) while still gaining additional information from the labels. In a semi-supervised anomaly detection problem, there are three sets of data points: positive labels (i.e., known anomalies), negative labels (i.e., known “normal” examples), and unlabeled data; the unlabeled data is expected to contain mostly normal examples, with a small number of (unknown) anomalous examples mixed in. Many semi-supervised learning methods only use one or two of these sets – e.g., only training on unlabeled data plus “normal” examples. Ruff *et al.*⁵ propose a new loss function that leverages all three sets and allows them to influence training in different ways. We adapted this loss function to match our modeling problem, using as the core measure of anomalousness the sparse MSE between the original input and the autoencoder’s reconstructed output instead of the distance from the model output to the center of the data:

$$Batch\ Loss = \sum_{i=1}^n MSE(x_i) + \eta \sum_{j=1}^m MSE(\tilde{x}_j)^{\tilde{y}_j}$$

Where

x_i are unlabeled returns, and \tilde{x}_j are labeled returns

\tilde{y}_j are labels for \tilde{x}_j (-1 for non-compliant and 1 for compliant)

$$MSE = \frac{1}{k_{filled}} \sum_{l=1}^{k_{filled}} (populated\ line_l - predicted\ line_l)^2$$

$\eta \geq 0$ (Note that $\eta = 0$ would be equivalent to unsupervised learning)

In this loss function, η allows for weighting the labeled data relative to the unlabeled data, varying the influence that the labeled data has on the parameter updates for each batch. The use of \tilde{y}_j as the exponent on the labeled losses means that for non-compliant returns (i.e., $\tilde{y}_j = -1$), lower reconstruction error will cause a *higher* contribution to the overall batch loss, incentivizing the model to have larger reconstruction errors for those returns (even though non-compliant returns

are coded as -1 in the loss function, we will still refer to them as “positive” labels for clearer discussions around anomaly detection). The customized MSE calculation helps account for the sparsity of the data, so errors corresponding to missing values do not skew relationships learned during training and missing values do not need to be imputed. Note that the sparse MSE is also used to calculate the score for each return during model evaluation.

We also introduced modifications to this loss function by incorporating a continuous success metric, which we will call S , that is available for all labeled returns (S is always 0 for compliant returns). First, we allowed for treating only labeled returns that have a value for S higher than some threshold as the positive labels during training – i.e., $\tilde{y}_j = \{-1 \text{ if } S(\tilde{x}_j) \geq \text{thresh}; 1 \text{ otherwise}\}$. Second, we allowed the magnitude of S to affect η directly – i.e., $\sum_{j=1}^m (\eta + \hat{S}(\tilde{x}_j)) \text{MSE}(\tilde{x}_j)^{\tilde{y}_j}$ for the second term in the loss function, where \hat{S} is S scaled to have roughly proportional magnitude to the chosen value for η . This means labeled returns with higher values of S are weighted more heavily in the batch loss calculation. Note that both modifications were parameterized so that they could be easily turned “on” or “off” as needed – we will call the first parameter *train_with_thresh* and the second parameter *add_to_eta*.

2.4 Ensemble Methods

Our approach involves multiple levels of model ensembling. We run several “trials” for each set of parameters, then combine the scores from each trial to evaluate the model(s), thereby increasing the stability of model performance. Because scores in one trial could be higher on average than scores in another trial, the ensemble across trials was implemented by scaling the scores within each trial to between 0 and 1, then summing the scaled scores across trials for each return. Generally, 3-5 trials were used for each experiment. Note that all discussions of model performance that follow are based on combined scores from multiple trials unless stated otherwise.

Using these combined scores from across trials for each model, we implemented another layer of ensembling, which we will call the “**two-model**” approach. As the name suggests, the two-model approach involves ensembling scores from two independent autoencoders, one with positive performance and one with negative performance (i.e., area under the ROC curve (AUC) below 0.5) – we will refer to these as M^+ and M^- , respectively. M^+ and M^- are semi-supervised models with the same data, parameters, loss function, etc.; the difference between the models is how they were trained: M^- reverses the labels used during training – i.e., $\tilde{y}_j = -1$ for compliant returns. This change incentivizes M^- to have low reconstruction errors for non-compliant returns and high reconstruction errors for compliant returns – ideally leading to negative performance when evaluated on the true labels. Intuitively, this means training one model (M^+) that flags non-compliance as anomalous and another model (M^-) that flags *compliance* as anomalous – meaning a higher M^- score should be associated with a higher likelihood of compliance. Flipping the labels creates a non-trivial difference between M^+ and M^- because positive and negative labels affect the loss function differently and because the unlabeled data still has an influence on model training. The two models are not perfectly anti-correlated, and in general, $AUC(M^-) \neq 1 - AUC(M^+)$, so there is complementary information in each model that can be combined via an ensemble score. The two-model score for return \tilde{x}_j is calculated as:

$$\text{Two Model Score}_j = \frac{M^+(\tilde{x}_j) - M^-(\tilde{x}_j)}{M^+(\tilde{x}_j) + M^-(\tilde{x}_j)}$$

Where $M^+(\tilde{x}_j)$ is the score for return \tilde{x}_j using the model M^+ , calculated by taking the sparse MSE between \tilde{x}_j and the reconstructed output obtained by passing \tilde{x}_j as an input to M^+ .

This setup means the highest-scoring returns under the two-model approach will be those that have a large relative difference between their M^+ and M^- scores. Note that the two-model approach only affects scoring – M^+ and M^- are trained independently.

3. Implementation and Numerical Results

3.1 Data

We incorporated a variety of different tax data into our research to ensure robust performance and generalizability, including data generated by different labeling mechanisms, data from different tax filer population segments (i.e., *strata*), and data from different tax years.

Labels available for model training can be divided into two categories: National Research Program (NRP) and operational. NRP labels represent a random sample from the population, while operational labels represent returns that the IRS selected due to suspected errors. NRP and operational labels can capture different aspects of a model’s performance: performance on NRP labels can be interpreted as the model’s ability to identify non-compliance in general (i.e., across the whole population), while performance on operational labels can be interpreted as the model’s ability to prioritize among returns that are deemed higher risk on average – similar to looking only at the bottom corner of a ROC curve generated on NRP labels.

To mirror operational practices, data can be further split into 11 different strata. Most experiments focused on a few representative strata to speed up runtime, but performance was periodically evaluated across all strata. The strata ranged from ~200 labeled / ~50K unlabeled returns to ~8K labeled / 1.5M unlabeled, depending on the dataset used (unlabeled data was randomly sampled from the full population, again for computational efficiency). Each stratum was trained separately, but some experiments involved combining multiple strata into a single training set. Both the labeled and unlabeled sets can also include one or more different tax years. More recent data is generally preferred, but labeled data for the most recent years may not be complete.

Features include 250+ continuous line items from F1040 and associated schedules; these line items are filtered to remove those that are only populated on a small number of returns. This filtering helps the model capture meaningful patterns in the data and avoid overfitting based on a small amount of data. Further data processing includes applying a cube root transformation for all line items, then applying Maximum Absolute Value scaling to make each line item range from 0 to 1 (or -1 to 0).

3.2 Model Architecture

We used a relatively simple model architecture with fully connected linear layers, as illustrated in Figure 2. Most experiments used a model with 5 hidden linear layers containing [100, 80, 64, 32, 8] neurons per layer for the encoder step and the reverse for the decoder step, meaning the bottleneck layer had $32 \times 8 + 8 = 264$ parameters. Other structures were also evaluated, including use of dropout layers, varied numbers of layers, and varied numbers of neurons per layer (more detail in sections below). SeLu was used for each hidden layer activation function and Tanh for the output activation function.

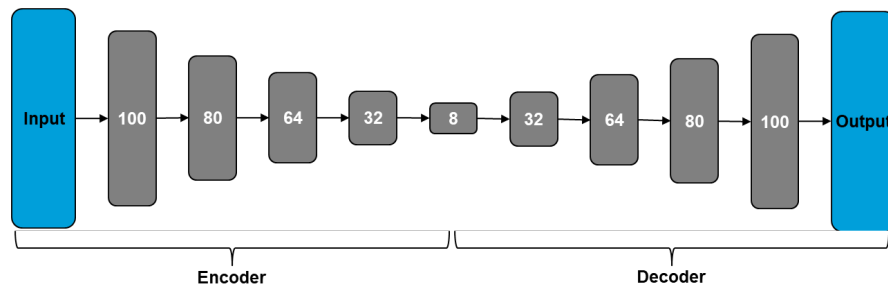


Figure 2: Illustration of Default Network Structure

All model training and evaluation was executed in Python using the PyTorch library as the primary tool for model building. An NVIDIA DGX GPU server was leveraged to improve computational efficiency of modeling such a large, high dimensional dataset.

3.3 Experimentation

Labeled data included distinct “train” and “test” sets; these could be drawn from the same overall set (e.g., 2016 operational labels) using a random 50/50 split or from entirely separate sets (e.g., NRP vs operational labels). Performance for each experiment (i.e., combination of parameters) was evaluated primarily based on ROC curve AUC on the test set. Other outputs for each experiment included ROC curves for the training set, Precision-Recall curves, and AUC vs epochs plots (to analyze trends in performance over the course of model training). When evaluating experiments, performance was considered across strata to emphasize parameter changes that improved performance for most or all strata. Performance for both the M^+ and two-model scores was also considered, as two-model AUC could improve even if M^+ AUC did not, and vice versa. AUC for an existing operational model was used as a benchmark.

The following key findings emerged from our research:

- **Semi-supervision improves performance:** the semi-supervised autoencoder consistently outperforms the unsupervised autoencoder with the same parameters.
- **Performance is favorable relative to benchmark:** the semi-supervised autoencoder consistently outperforms the existing operational model.
- **Performance generalizes:** the semi-supervised autoencoder performs well across strata and across different datasets.
- **Two-model ensemble improves performance:** the two-model score often provides a significant boost in AUC over M^+ , although the boost is more consistent for certain strata than others, and the two-model AUC is not always better than the M^+ AUC.
- **Cube root transformation improves performance:** taking the cube root of each line item as an intermediate pre-processing step, combined with corresponding adjustments to the learning rate and weight decay parameters, provides a solid boost in AUC.

We focused our exploration of the parameter space on parameters that control the use of labeled data during model training – namely η (i.e., the weight applied to labeled data in the loss function), *train_with_thresh* (i.e., whether to only consider higher values for the continuous success metric S as positive labels during training), and *add_to_eta* (i.e., whether to scale η based on the magnitude of S for each labeled return). Varying η did not change performance as much as expected, even when ranging from .001 to 1000. Strong performance was still achieved with smaller values for η , but for most experiments, we defaulted to $\eta = 1$. Setting *train_with_thresh* = True was often found to increase AUC for M^+ but not the two-model score. Setting *add_to_eta* = True did not significantly change performance when $\eta > 0$, although the combination of *add_to_eta* = True and $\eta = 0$ achieved similar performance to *add_to_eta* = False and $\eta > 0$.

We also focused on the impact of varying the number of epochs and the neural network structure. We varied the number of epochs from 1 to 100 and found that strong performance could be achieved with as few as 5-10 epochs. We varied the network structure by testing a smaller bottleneck layer (i.e., 4 neurons instead of 8), addition of dropout layers, and smaller number of hidden layers (i.e., 4 layers instead of 5 for both the encoder and decoder). These modifications to the network structure did not significantly change model performance, suggesting that future versions could utilize even simpler models and still achieve similar performance.

Table 1 shows the default parameter values used for most experiments. Since experiments were evaluated based on a combination of factors, including results across different strata and datasets, it is possible that a different set of parameters may perform better for a specific use case (e.g., single stratum).

Table 1: Parameter Definitions and Default Values

<i>Parameter</i>	<i>Definition</i>	<i>Value</i>
eta (η)	Weight on labeled data in loss function	1

train_with_thresh	Whether to only consider higher values for a continuous success metric as positive labels during training	False
add_to_eta	Whether to scale η based on a continuous success metric for each labeled return	False
epochs	Number of times all data will be passed through the network and parameters will be updated through backpropagation	20
batch_size	Number of returns to include in each training batch	10,000
learning_rate	Magnitude by which to adjust model parameters during each training iteration	1e-3
weight_decay	Regularization method to penalize larger weights in the network	1e-5
plt_threshold	Used to prevent learning patterns from unlabeled returns that have reconstruction error above the threshold (if value is between 0 and 1)	1
neurons_list	Controls number of hidden layers, number of neurons in each layer, and presence of dropout layers	[100, 80, 64, 32, 8]

3.4 Results

Unless stated otherwise, results shown in this section are based on models that were trained using unlabeled tax filings and operational labels from 2016 and evaluated on 2018-2020 operational labels. Note that where the labels “flipped” and “usual” are applied in the plots, they refer to M^+ and M^- , respectively.

Figure 3 compares performance for a semi-supervised autoencoder with a corresponding unsupervised autoencoder (i.e., same parameters, just setting $\eta = 0$) and with the existing operational benchmark model. Note that the semi-supervised model here is M^+ , not the two-model score. The semi-supervised model achieves AUC = 0.66, compared to 0.61 for the unsupervised model and 0.58 for the benchmark.

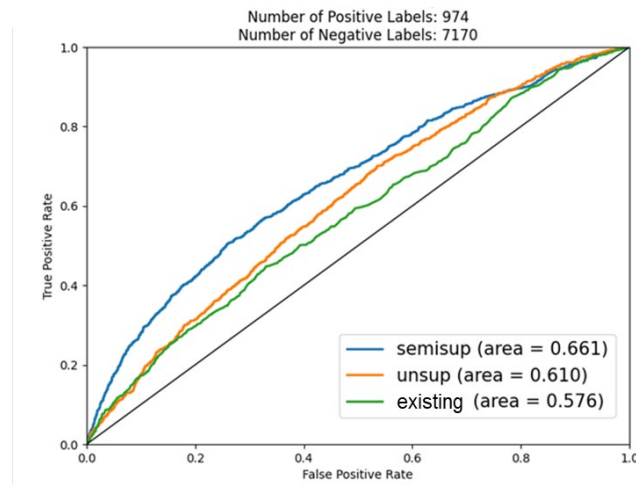


Figure 3: ROC Curve – Semi-Supervised vs Unsupervised Autoencoder

Figure 4 provides a representative example of the boost in performance from the two-model approach: the two-model approach achieves AUC = 0.67, well above the M^+ AUC of 0.51. Note that it is possible for both M^+ and M^- to have AUC below 0.5 and still achieve two-model AUC well above 0.5. The two-model AUC tends to be higher when the difference between the M^+ and M^- AUCs is larger, although this relationship is not always consistent.

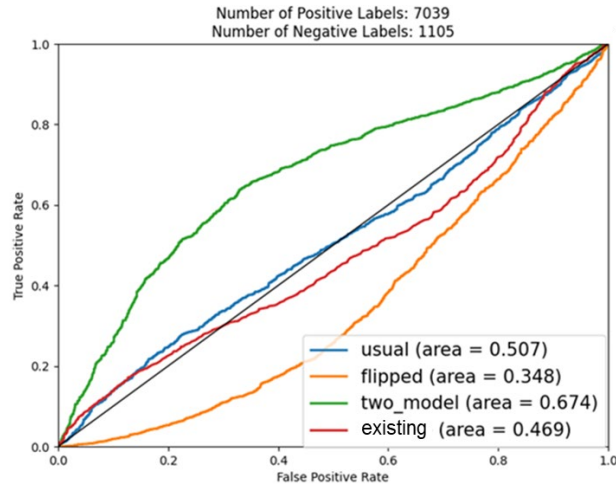


Figure 4: ROC Curve – Two-Model Scoring Approach

Figure 5 shows two plots, one where a cube root transformation has been applied as part of data processing, and one without the cube root transformation (without changing any other parameters). The two-model approach with cube root achieves AUC = 0.58, compared to 0.51 without the cube root.

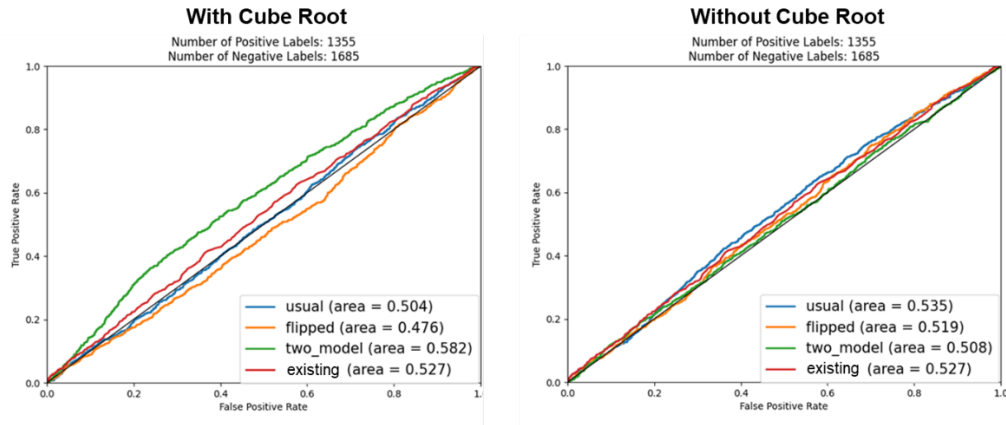


Figure 5: ROC Curve – Cube Root Transformation

Table 2 shows how performance varies with η across multiple strata – bold values indicate the highest AUC within each stratum (in the case of ties, the lowest value for η is bolded). Note that best or near-best performance can generally be achieved with small η values (i.e., 0.01 or 0.1). All else equal, we prefer smaller values for η to reduce the likelihood of overfitting the model to the labeled data.

Table 2: AUC by Stratum and η

Stratum	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$	$\eta = 10$
E	0.59	0.64	0.65	0.65	0.65
I	0.53	0.60	0.61	0.65	0.61
J	0.61	0.63	0.63	0.62	0.61
K	0.55	0.57	0.58	0.58	0.55

Table 3 shows results across all strata comparing the semi-supervised autoencoder with the existing benchmark model – both the M^+ and two-model AUCs are shown, and bold values indicate the highest AUC (if the difference in AUCs is less than 0.02, then no values are bolded). The M^+ and/or two-model AUC is higher than the benchmark for all but one stratum, while the two-model score provides a boost over the M^+ score for most but not all strata.

Table 3: AUC by Stratum and Model Type

<i>Stratum</i>	<i>M^+ Score</i>	<i>Two-Model Score</i>	<i>Benchmark</i>
A	0.60	0.54	0.49
B	0.51	0.59	0.53
C	0.59	0.58	0.58
D	0.47	0.56	0.52
E	0.51	0.68	0.47
F	0.52	0.56	0.54
G	0.62	0.59	0.56
H	0.54	0.57	0.51
I	0.54	0.62	0.50
J	0.52	0.62	0.51
K	0.50	0.59	0.53

Figure 6 shows two plots – the left shows models trained on NRP labels, while the right shows models trained on TY16 operational labels; both sets of models are also trained on TY16 unlabeled data and evaluated on NRP labels. Not surprisingly, the AUC is higher (0.61) when training and evaluating on NRP labels (i.e., using a 50/50 train-test split) compared to training on TY16 operational labels, then evaluating on NRP labels (0.56); this is still close to the benchmark model AUC (0.58) and shows how the autoencoder performance can generalize across different datasets and tax years. Note that performance relative to the benchmark model is more mixed for NRP data, but this makes sense given how the benchmark model is trained.

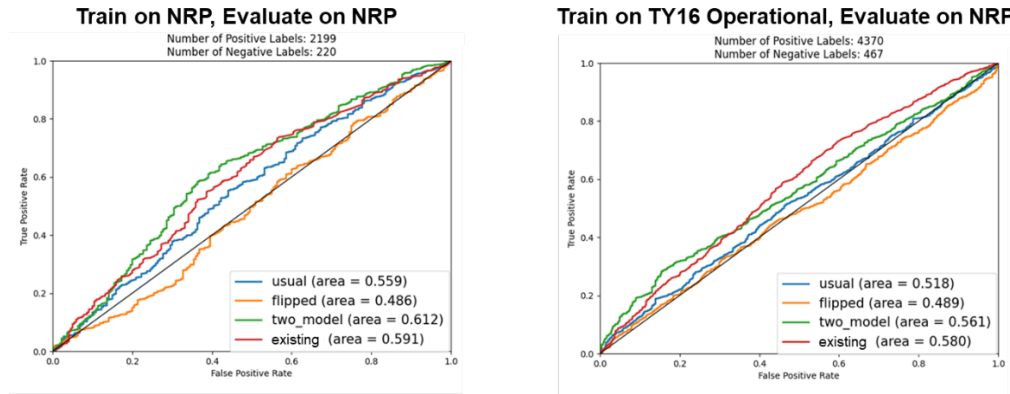


Figure 6: ROC Curve – Generalization to Different Datasets

Table 4 shows how AUC changes with the number of epochs for which the model is trained – bold values indicate the highest AUC within each stratum (in the case of ties, the lowest number of epochs is bolded). While the optimal number of epochs varies across strata, the best or near-best performance can be achieved with a relatively small number of epochs (i.e., between 5 and 20 epochs). Note that all else equal, we prefer fewer epochs to reduce the risk of overfitting and the runtime needed to train the model.

Table 4: AUC by Stratum and Epochs

<i>Stratum</i>	<i>Epochs = 5</i>	<i>Epochs = 10</i>	<i>Epochs = 20</i>	<i>Epochs = 40</i>	<i>Epochs = 100</i>
E	0.67	0.66	0.65	0.64	0.64
I	0.60	0.62	0.62	0.61	0.60
J	0.60	0.61	0.63	0.63	0.63
K	0.53	0.57	0.57	0.57	0.55

3.5 Additional Exploration

We tested additional techniques that ultimately were not included in the final approach. One was “early stopping”: instead of stopping after a fixed number of epochs, model training would stop if performance on a separate validation set decreases for several consecutive epochs. We hypothesized that early stopping could both optimize performance and reduce the likelihood of overfitting by stopping model training at a point that was “just right.” However, after testing early stopping using a validation set based on a 20% sample of the training labels (i.e., using 80% of the training labels for direct model training and 20% for validation after each epoch) and with 3 or 5 consecutive epochs of decreasing AUC on the validation set as the criteria, we found that early stopping did not provide a boost in performance. One potential improvement for future consideration could be to retrieve an earlier version of the model if the early stopping criteria is met, rather than the latest model parameters at the time training stops – e.g., if model training stops early at epoch n , use the parameters from the model as of epoch $n - 3$.

We also tested a “pseudo-label” approach that involved training an initial (semi-supervised) model and using it to score the unlabeled data, then training the “final” model on labeled data plus only the unlabeled data that had scores above or below a certain threshold, using the scores from the initial model. This approach can be viewed as a modified version of PLT: filtering the unlabeled data that the model “sees” during training. The use of pseudo-labels was implemented separately for M^+ and M^- using a fixed threshold of 0.7: after the initial M^- was trained and used to score the unlabeled data, only the unlabeled returns in the top 30% of initial M^- scores would be used to train the final M^+ , and vice versa for the final M^- (if the initial M^- is performing as intended, then the returns in that top 30% should be those with the highest probability of compliance). This approach provided a small boost in performance for some experiments and had no impact in others. We decided not to include pseudo-labels in the final approach due to the added complexity and uncertain benefit.

Lastly, we tested using different parameters for M^+ and M^- (i.e., beyond flipping the labels – all the two-model results presented so far are based on M^+ and M^- with the same parameters). Since this change vastly increases the potential parameter space, we limited the scope of parameter differences to η (e.g., small η for M^+ , large η for M^-) and *train_with_thresh* (e.g., True for M^+ , False for M^-). These changes generally yielded performance that was the same or worse than simply using the same parameters for both models. Given the added complexity, we decided to focus on improving performance while keeping the parameters for each model the same.

4. Discussion

The results presented here suggest that semi-supervised autoencoder methods can consistently match or exceed existing processes for detecting anomalies in tax form data. This provides a promising path toward incorporating limited labeled data into model training while maintaining the ability to detect novel anomalous examples. The results also show the capacity for novel ensemble methods with multiple autoencoder models (i.e., two-model approach) to significantly enhance performance, as well as the flexibility of this semi-supervised autoencoder framework – e.g., varying how much the labeled data affects the training process, incorporating both binary and continuous success metrics into model training, achieving robust performance across different datasets and strata, and more – further suggesting the potential benefits of this approach.

References

- [1] A. S. Parker, D. Gewurz, and W. J. J. Roberts. "Quality and Validity Testing of Sparse Form Data using Gaussian Mixture Models," JSM Proceedings, Social Statistics Section, 2018.
- [2] A. S. Parker, D. Gewurz, and W. J. J. Roberts. "Recommender Algorithms for Form Anomaly Detection," JSM Proceedings, Government Statistics Section, 2020.
- [3] C. Acton, L. Corman, J. Bono, D. Gewurz, C. Walsh, and E. Schulz. "Anomaly Detection on Sparse Data with Autoencoders," JSM Proceedings, 2023, DOI: 10.5281/zenodo.10001050.
- [4] N. Merrill and A. Eskandarian. "Modified Autoencoder Training and Scoring for Robust Unsupervised Anomaly Detection in Deep Learning," IEEE Access, vol. 8, pp. 101824-101833, DOI: 10.1109/ACCESS.2020.2997327.
- [5] L. Ruff, R. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K-R. Müller, and M. Kloft. "Deep Semi-Supervised Anomaly Detection," International Conference on Learning Representations, 2020, URL: <https://doi.org/10.48550/arXiv.1906.02694>.