Towards Deep Representation Learning for Fraud Classification

Author: Julian van Meggelen (2675511)

1st supervisor:	Dr. Kevin Sebastian Luck
daily supervisor:	Steven van der Veer (Perseuss)
2nd reader:	Prof. Dr. Sandjai Bhulai

A thesis submitted in fulfillment of the requirements for the VU Master of Science degree in Business Analytics

November 22, 2024

PERSEUSS

VU VRIJE UNIVERSITEIT AMSTERDAM

Abstract

Fraud detection is a critical task for financial institutions, traditionally dominated by tree-based models such as XGBoost. This work investigates the potential of deep learning methods for fraud detection by conducting a comprehensive benchmark study and exploring novel self-supervised and supervised representation learning techniques. The study evaluates two datasets: ULB300K, a publicly available dataset, and PER40M, a proprietary, large-scale dataset representative of real-world scenarios. First, we benchmark a selection of common methods for fraud classification, highlighting XGBoost's dominance due to its strong performance and computational efficiency. Next, self-supervised learning methods based on autoencoder architectures are tested to assess whether meaningful representations can be learned without labeled data. While these methods demonstrate potential, particularly on ULB300K, their performance on PER40M is less competitive. In our second experiment, we show that supervised representation learning using metric learning and the triplet loss yields significant improvements in downstream classification performance. Transformer-based encoders, in particular, outperform traditional and self-supervised methods on PER40M, demonstrating the capacity of deep learning models to capture intricate patterns in large, high-dimensional datasets. This performance comes at a cost, as the best performing transformer-based encoder requires two OOM more FLOPs for inference than an XgBoost classifier.

Contents

List of Figures

Page

List of Tables

Chapter 1

Introduction

Recent advancements in computing and machine learning (ML) have led to significant advancements in the performance of artificial neural networks (ANN) in a broad range of applications. In particular, deep learning methods in computer vision and natural language processing have achieved super-human performance on a variety of benchmarks. The superiority of deep learning models in these areas is mostly attributed to their capability of representation learning. This means that ANN models learn informative features from raw input data without human intervention, whereas traditional ML methods often require human-crafted features. These feature representations form the basis of their strong performance on downstream tasks such as classification and regression. However, the application of deep learning to domains beyond structured data, such as tabular data, remains less explored due to inherent challenges.

One area where the usage of DL is less widespread is tabular data. Tabular data consists of rows and columns, where each row represents an instance, and columns correspond to potentially unrelated features. This structure is fundamentally different from images or text, which exhibit significant local structure (pixels/words that are structurally close are often related). Specialized neural network architectures exploit this local structure, contributing to their superior performance. Another difference between tabular data and other structures is the relatively low dimensionality typically found in tabular data. Commonly, tabular data has an order of magnitude lower dimensionality than, for example, image data, resulting in a potentially lower signal-to-noise ratio. Combined, these factors often make traditional machine learning methods, such as tree methods, favorable over DL methods (Ye et al., 2024). More recently, interest in DL for tabular data has resurrected. A combination of the ever-increasing size of datasets used in industry and the discovery of new methods for supervised tabular learning have revived the potential of DL for tabular data as they achieve superior performance on some benchmarks (Gorishniy et al., 2023).

Credit card fraud detection, a domain where tabular data is prevalent, is the focus of this work. This is an extremely relevant application as fraud-induced chargebacks cause hundreds of millions of losses yearly. Per illustration, credit-card fraud caused £574 million of losses in 2021 in just the UK alone (Worobec, 2021). Industry-wide, ML methods are applied to detect fraudulent transactions and avoid the associated losses. The detection of fraudulent transactions is essentially a binary classification problem. However, several factors make this a challenging problem. Firstly, the data is heavily imbalanced as fraudulent transactions occur substantially less frequently than valid transactions. This means that fraud detection systems need to be trained using heavily imbalanced datasets. Secondly, fraudulent transactions may go unnoticed or could be mislabeled, which requires ML methods to be robust against data contamination.

In this work, we take a significant step in the direction of deep representation learning for credit card fraud detection by conducting thorough experiments using a supervised and self-supervised representation learning method on real-world transaction data. In the self-supervised setting, we train an autoencoder ANN and utilize the obtained encoder in a downstream classifier. In the supervised training setting, we train an ANN encoder in a triplet loss training framework, explicitly guiding the embedding space towards a structure where the embeddings of fraudulent and valid transactions are well-separated. For both training paradigms, an Isolation Forest and K-nearest neighbors classifier are used to obtain final predictions.

This work makes several key contributions. Firstly, we conduct an in-depth benchmark study comparing common methods for fraud classification. We obtain benchmarks on both a public fraud dataset and a proprietary dataset used in industry by a multinational fraud-detection service provider. This provides insights into the relevance of public datasets compared to proprietary industry datasets. Secondly, we show how self-supervised learning can be used to obtain efficient feature encoders and how the performance behaves with respect to the scale of the encoder model. Lastly, we demonstrate how a triplet loss-based supervised learning framework structures the latent space to separate fraudulent and valid transactions effectively. We also analyze its scaling properties and performance relative to self-supervised learning.

1.1 Related work

This work lies on the intersection of a stream of applied research on credit card fraud detection and the fundamental study of neural networks and representation learning. On the applied research side, we can roughly divide current work in two categories. One body of research uses open source datasets, which is almost exclu-

sively a dataset published by Dal Pozzolo et al. (2014) in 2014. This dataset forms the basis of a large body of literature on fraud detection. This dataset is used by Yu et al. (2024) who show that a transformer architecture enhances classification performance compared to a vanilla MLP architecture. Zhu et al. (2024) show that, on this dataset, a SMOTE sampling technique improves the performance of a NN classifier. Talukder et al. (2024) use a grid search method to find an ensemble of classifiers that outperforms traditional methods such as KNN and Random Forest. de Souza and Jr (2021) use an ensemble between a self-supervised learning method (K-means clustering) and a supervised classifier and show that this reduces inference complexity while maintaining accuracy. Porwal and Mukund (2019) approach the problem from an anomaly detection setting and show the efficacy of an Isolation Forest method on this fraud dataset. All the aforementioned work uses this same dataset introduced by Dal Pozzolo et al. (2014). A downside is the small number of records in this dataset and the even smaller number of fraudulent transactions in this dataset (which is less than 500). First of all, this means that comparing many different methods becomes infeasible from a statistical standpoint, as a low number of fraudulent transactions in the test set poses a substantial issue with regard to statistical significance of the results. Furthermore, it cannot be assessed how a method scales to a larger (real-world) dataset.

There is little public work on ML for fraud detection on large industry datasets. Duan et al. (2024) use a private dataset consisting of 2M records from a Chinese bank but focus on graph learning and do not include benchmarks for classifier models typical for tabular data. de la Bourdonnaye and Daniel (2021) use a proprietary dataset containing 10M records but focus solely on gradient boosting methods and methods for categorical feature encoding. Melo et al. (2023) use a large-scale (150M) proprietary dataset but focus exclusively on attack propagation through adversarial training. The work by Thimonier et al. (2023) is one of the few providing a comparative benchmark study between different classification methods on a large (192M) private industry dataset and show that the LightGBM gradient boosting algorithm outperforms methods such as KNN and Isolation Forest. S. et al. (2023) is the only work that uses a supervised representation learning approach on a large-scale dataset, making it methodologically the most similar to the method proposed in this thesis. However, their work focuses on malicious software session detection and is based on a sequential model architecture.

All in all, there is little to no work that contains a complete benchmark study on a large-scale fraud dataset. This is one gap that this work aims to solve. There is, besides S. et al. (2023), also no work on (self-)supervised representation learning on a large-scale fraud dataset. This thesis aims to fill this gap by exploring both supervised and self-supervised methods on both a public and a large-scale proprietary dataset.

Chapter 2

Background

This section aims to provide the reader with the required background knowledge to comprehend this work's methodology. The section starts with a theoretical framework for supervised learning and progresses to some applied methods for fraud classification and representation learning.

2.1 Supervised learning: the fundamentals

2.1.1 Risk Minimization

The ultimate purpose of supervised learning is to find a functional representation for the unknown conditional distribution

$$p(y|x)$$

where y is some response variable and x are some features. In the case of binary classification, y is binary s.t $y \in \{0, 1\}$. Now, consider some universe of functions $\{f(x, \theta),\ \theta \in \Theta\}$ parametrised by θ that we can choose from to model $p(y|x)$. Our goal then becomes to choose $f(x, \theta)$ that behaves the most like $p(y|x)$. We define a loss $L\left(y, f(x, \theta)\right)$ to quantify how well $f(x, \theta))$ mimics $P(y|x)$. If we then take the expectation

$$\mathbb{E}\Big[L\big(y, f(x, \theta)\big)|\theta\Big] = \int L\big(y, f(x, \theta)\big)dp(x, y)$$

we obtain an expression for the total loss of the function $f(x, \theta)$ over the whole data distribution $p(x, y) = p(x)p(y|x)$. This expectation is often referred to as the risk functional $R(\theta)$ (Vapnik, 1999). We could simply use this risk functional to choose an θ that minimizes the risk functional:

$$\theta_0 = \underset{\theta \in \Theta}{\operatorname{argmin}} R(\theta)$$

2.1.2 Empirical Risk Minimization

However, in practice, the exact specification of $p(x, y)$ is unknown, making the evaluation of $R(\theta)$ impossible. We therefore need to revert to evaluating $R(\theta)$ based on sampled data. That is, we collect a set of samples $\{(x_i, y_i)\}_{i=1}^{N} \overset{\text{i.i.d.}}{\sim} p(x, y)$ and construct the empirical risk functional

$$\hat{R}(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i, \theta))$$

Now we can obtain an estimate $\hat{\theta}_0$ of θ_0 by minimizing the empirical risk functional:

$$\hat{\theta}_0 = \underset{\theta \in \Theta}{\operatorname{argmin}} \hat{R}(\theta)$$

2.1.3 Consistency of ERM

Now, one fundamental assumption underlying empirical risk minimization is that of uniform convergence of the empirical risk function to the (non-empirical) risk function. Formally this convergence is defined as

$$\lim_{N \to \infty} P\left(\sup_{\theta \in \Theta} \left(R(\theta) - \hat{R}(\theta) \right) > \varepsilon \right) = 0$$

If this convergence holds, $\hat{R}(\theta)$ approached $R(\theta)$ and $\hat{\theta}_0$ approaches θ_0 as the size of the dataset grows. This implies that, given a large enough dataset, our learned model will adequately generalize to the whole data distribution.

This convergence mostly relies on the Uniform Law of Large Numbers (ULLN). Some other import conditions for this convergence to hold are stated in Vapnik and Chervonenkis (1971). One noteworthy condition is that of the Vapnik and Chervonenkis (VC) dimension. The VC dimension d_{VC} is a measure of the complexity of a set of classification functions, defined as the maximum number of input vectors that the set of functions can shatter. Here, shattering means that for a set of n input vectors, the hypothesis class can realize all 2^n possible binary labelings of those vectors. In order for the above convergence to hold, the VC dimension of our function family $\{f(x, \theta) \colon \theta \in \Theta\}$ must be finite. One can verify intuitively that a family of functions with an infinite VC dimension (infinite complexity) can realize every possible labeling of an infinitely large dataset, allowing it to overfit on any arbitrarily large dataset, violating the convergence property.

2.1.4 VC-dimension and the Bias-Variance trade-off

The role of the VC dimension in the consistency of ERM discussed above may suggest a model family with a low VC dimension is always favorable over those with high VC dimension due to improved generalization capabilities. This is true

to some extent, but the Bias-variance decomposition sheds a different light on this. This trade-off, first introduced by Geman et al. (1992) states that we can decompose a model's expected loss into three parts; bias, variance and noise. They show that the expected Mean Squared Error loss, as defined by $L\left(y, f(x,\theta)\right) = (y - f(x,\theta))^2$, can be decomposed into

$$\mathrm{MSE}(\hat{f}) = \mathbb{E}_{\mathbf{x}}\left[\left(\hat{f}(\mathbf{x}) - f(\mathbf{x})\right)^2\right] = \mathrm{Bias}(\hat{f}(\mathbf{x}))^2 + \mathrm{Var}(\hat{f}(\mathbf{x})) + \sigma^2$$

The decomposition is less obvious for other loss functions, but several efforts have been made over the years (Belkin et al., 2019; Domingos, 2000; Kohavi et al., 1996). Bias is induced by the models incapability to correctly model the data. Variance is induced by a models sensitivity to the training data, and the noise is an irreducible part of the loss induced by noise in the data. Thus, models with a high VC-dimension typically have low bias and models with a low VC-dimension typically have a high bias.

2.1.5 Practical implications

The theoretical framework of statistical learning forms the basis for many experimental methods used by machine learning (ML) practitioners. Firstly, it is standard practice to use out-of-sample testing (such as cross-validation) to assess the generalizability of models and diagnose overfitting, particularly for model families with high complexity (i.e., high VC dimension). This ensures that the model performs well on unseen data, not just the training set.

Secondly, statistical learning theory informs us that high-complexity models (with higher VC dimension) should only be used when sufficient data is available. Without enough data, these models are prone to overfitting. On the other hand, the bias-variance trade-off suggests that more complex models typically have lower bias than simpler models, making them more flexible in capturing underlying patterns in data. However, this flexibility comes at the cost of higher variance, which increases the risk of overfitting if data is limited.

To mitigate these challenges, data augmentation is commonly used to artificially expand the dataset, helping to improve the generalization of complex models when data is scarce. Additionally, regularization techniques (such as L2 or L1 regularization) are employed to reduce the effective VC dimension of models by constraining the model parameters, thereby reducing variance and helping prevent overfitting.

2.2 Common model families

Many function families (models) for $f(x, \theta)$ have been proposed and used over time. This section will cover the most significant ones with respect to the fraud detection problem.

2.2.1 Tree based models

Tree-based models are popular for both classification and regression tasks due to their flexibility in handling different types of data (continuous and categorical) and their ability to capture non-linear relationships in the feature space without the need for extensive data preprocessing.

Decision tree

The simplest form is the decision tree (Quinlan, 1986; Salzberg, 1994), which iteratively partitions the feature space X by choosing, at each step, at node m the feature i and threshold t_m that maximize some measure of quality of the split $H(\cdot)$. For $H(\cdot)$, typically, the gini-coefficient or Information gain is used for classification tasks and variance reduction for regression tasks. For a continuous feature X_i, the decision tree searches over all possible threshold values and selects the one that best separates the data into two subsets with lower impurity (or variance). For a categorical feature, it examines different groupings of categories. Node splitting continues until all possible splits have been made or some stopping criteria is reached. One can verify that a decision tree model on continuous features can continue splitting the feature space until all datapoints in the dataset are isolated. This implies that decision trees without adequate stopping criteria have infinite VC dimension. Stopping criteria are needed to ensure generalization accuracy over the data distribution. Common stopping criteria put a limit on the depth of the tree or a minimum on the number of samples per leaf of the tree. A decision tree's ability to fit to the training data imposes a significant risk of overfitting, which is the main reason other variants were developed.

Random forests

The Random Forest aims to overcome the risks of generalization inaccuracies of the decision tree. Pioneered by Ho (1995) and formalized by Breiman (2001), this is an ensemble method that combines multiple decision trees trained on randomly sampled subsets of the datasets. Furthermore, at each node, a random subset of features is considered. These tricks act as a de facto regularization technique that prevents the model from becoming too complex and overfitting on the training set. Final predictions are obtained by either averaging the result of all the trees (regression) or using a majority vote to choose the predicted class (classification). A downside is that Random Forests lose interpretability. Where decision trees can easily be

visualized and interpreted, random forest make this more difficult, especially as the number of trees in the ensemble grows.

Gradient boosted trees

Gradient boosting is an ensemble technique introduced by Friedman (2001) with foundations laid by Freund et al. (1996, 1999); Breiman (1997), where multiple decision trees work together in an additive manner. Intuitively, the idea is to have a sequence of learners (trees) that correct the mistakes made by previous learners. The gradient of the selected loss function with respect to the prediction is used to decide the direction in which to correct the error. This is also referred to as gradient descent in function space. Formally, the idea is to have M trees $F_m(x)$, $m = 1, ..., M$ where

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

Here, $h_m(x)$ is a tree fitted to the pseudo-residuals

$$\tilde{y}_{im} = -\left[\frac{\partial L\left(y_i, F_{m-1}(x_i)\right)}{\partial F_{m-1}(x_i)}\right]$$

and γ_m is a step size obtained by solving the minimization

$$\gamma_m = \underset{\gamma}{argmin} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(x_i) + h_m(x_i)\right)$$

The outcome of the last tree $F_M(x)$ is used as the final estimate $\hat{y}$.

The above specification represents the most essential form of the gradient-boosted tree algorithm. Many different variations to the algorithm have been proposed over the years (Ke et al., 2017; Prokhorenkova et al., 2019). Most noteworthy is the XGBoost algorithm (Chen and Guestrin, 2016). These modern variations combine a set of features to improve generalization accuracy. Typically, a form of regularization is used to penalize tree complexity and constrain the VC dimension of the model. In particular, Chen and Guestrin (2016) introduces a regularization term in the loss function which is defined as

$$\Omega\left(F_m(x)\right) = \gamma T + \frac{1}{2}\lambda ||w||^2$$

where T is the number of leaves in the tree and w represents the parameters in the leaves of the tree. γ and λ are hyperparameters that control the degree of regularization. The L2 norm $||w||^2$ can be replaced by the L1 norm $|w|$.

Furthermore, the second-order derivative of the loss with respect to the predictions is used for a more accurate estimation of the gradient of the loss. We must note that none of these features was fundamentally novel about the XGBoost algorithm. For example, both regularization and the use of second-order derivatives

were mentioned as early as by Friedman (2001). XGBoost mainly gained popularity due to its efficient software implementation, leading it to be the main tree-boosting algorithm currently used.

2.2.2 Artificial Neural networks

Artificial neural networks have become the standard method for many modeling tasks in a wide range of applications. The Multi-Layer Perceptron (MLP), as first defined by Rosenblatt (1958), is the core of many different architectures. An ANN is a highly complex mathematical function composed of linear and nonlinear functions chained together in a layer-like structure. In particular, we define the output of hidden layer l as a vector $\boldsymbol{h}^{(l)}$, such that

$$\boldsymbol{h}^{(l)} = g^{(l)}\left(\boldsymbol{W}^{l}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right) \tag{2.1}$$

where $g^{(l)}$ is the activation function used in layer l and $\boldsymbol{W}^{(h)}$ and $\boldsymbol{b}^{(l)}$ are the weight matrix and bias for the linear transformation in layer l. $h^{(0)}$ is defined to be the input of the function. Common choices for $g(\cdot)$ are the ReLU function and the sigmoid function.

ANN's are commonly trained through error backpropagation (Rumelhart et al., 1986). The update equation for the weights and bias at layer l are given by

$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \eta\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} \qquad\qquad \boldsymbol{b}^{(l)} \leftarrow \boldsymbol{b}^{(l)} - \eta\frac{\partial L}{\partial \boldsymbol{b}^{(l)}}$$

where η is the learning rate and $\frac{\partial L}{\partial \boldsymbol{W}^{(l)}}$ and $\frac{\partial L}{\partial \boldsymbol{b}^{(l)}}$ are the gradient of the loss with respect to the weight matrix and bias vector at layer l, respectively. In practice, these are evaluated numerically using the chain rule.

An important result in the context of ANN's for function approximation is that of Cybenko (1989) and Hornik (1991), who provided a first proof for the Universal Approximation Theorem. Many versions of the theorem exist, but in general, it tells us that for any function $f \in \mathcal{C}(\mathcal{X}, \mathbb{R}^D)$ there exists $\hat{f} \in \mathcal{N}^g_{d,D}$ such that $\sup_{x\in\mathcal{X}} ||\hat{f}(x) - f(x)|| < \varepsilon$, under the right conditions for $g(\cdot)$ and a sufficient number of layers. Here, $\mathcal{C}(\mathcal{X}, \mathbb{R}^D)$ is the function space spanned by all continuous functions that map $\mathcal{X}$ to $\mathbb{R}^D$, $\mathcal{N}^g_{d,D}$ is the function space spanned by the ANN's with d inputs and D outputs, and $\varepsilon > 0$ is a constant. Theoretically, this result implies that any function $P(y|x)$ can be represented by an ANN function $f(x, \theta)$ of sufficient complexity. That is, this representation exists but we still have to find the parameter set that creates it.

The universal approximation theory also warns about generalization inaccuracies that may arise when a complexly structured ANN is used to model $P(y|x)$. The theory tells us that we can build ANN functions $f(x, \theta)$ that can map any finite dataset exactly to its target label (sometimes referred as "remembering" the dataset). This is easier to deal with than for tree-based models, since we need to fix the ANN structure a-priori (the ANN structure does not grow during the training procedure). This means we need to be careful to choose an ANN architecture which complexity suits the dataset size. Another technique commonly used to tackle generalization inaccuraies is early-stopping, which entails using an evaluation set to determine whether overfitting is ocurring and stop the training procedure.

2.3 Evaluating binary classifiers

We previously discussed the role of the loss $L(y, f(x, \theta))$ that measures the performance of $f(x, \theta)$ with respect to $P(y|x)$. A common choice for the loss function in binary classification setting is the classification accuracy

$$L(y; f(x, \theta)) = \begin{cases} 0, & \text{if } y = \mathbb{1}_{f(x,\theta)>t} \\ 1, & \text{if } y \neq \mathbb{1}_{f(x,\theta)>t} \end{cases}$$

Where $t \in [0, 1]$ is some threshold value for our probability (commonly 0.5). Even though this metric serves as a good proxy of classification performance, practical applications often require a more sophisticated analysis of the classifiers behaviour. This is because of two main reasons. Firstly, the dataset may be imbalanced. Imbalanced data refers to an imbalance in the data-distribution, where the unconditional probability of observing a particular class $P(y = c_1)$ is substantially higher than the probability of observing the other class $P(y = c_2)$, such that $P(y = c_1) \gg P(y = c_2)$ for some class labels $c_1,\ c_2$. This means that a good classification performance based on the accuracy can be fully attributed to good performance on one class, instead of on both classes. Secondly, the application at hand might not equally weigh the importance of the classes. A false positive may be valued differently from a false negative. For these reasons, a range of other classification metrics have been proposed over the years.

2.3.1 Confusion-matrix based metrics

The confusion matrix, a term coined as early as by Pearson (1904) is the matrix composed of the number of true positives (P), false negatives (FN), false positives (FP) and true negatives (TN) (from top-left to bottom right). Based on these elements, a series of metrics can be defined that offer different perspectives to classification performance. Table A.1 shows a selection of such metrics.

As for the application of fraud detection, with an inherently imbalanced data distribution (fraudulent transactions occur substantially less frequently than non-fraudulent transactions) these metrics are essential for a complete picture of the behavior of a classifier. However, as noted before, we typically use a threshold t that determines the decision boundary of our classifier. This means that the confusion-matrix-based metrics are, as a matter of fact, functions of t. The next section covers ROC analysis, a method to quantify classification performance independent of t.

2.3.2 ROC analysis

We consider the FPR and TPR as a function of threshold t. Furthermore, we define two probability density functions $P(f(x,\theta)|y=1)$ and $P(f(x,\theta)|y=0)$, which represent the unconditional distribution of the estimator $f(x,\theta)$ given that the true class is positive or negative. We can express the TPR function as

$$TPR(t) = \int_t^{\infty} p(f(x,\theta)|y=1)df(x,\theta)$$

(which is the blue area in figure 2.1) and the FPR function as

$$FPR(t) = \int_t^{\infty} p(f(x,\theta)|y=0)df(x,\theta)$$

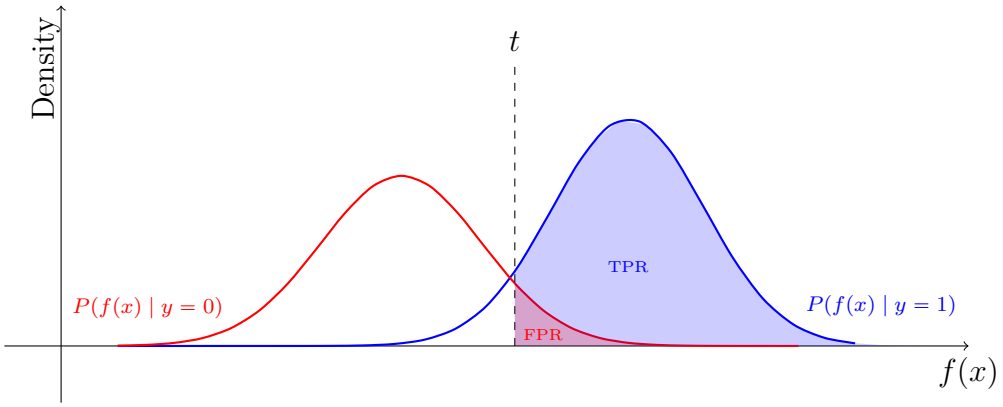


Figure 2.1: Visual representation of the PDF $p(f(x,\theta)|y=1)$ and $p(f(x,\theta)|y=0)$. The red and blue shaded are show the FPR and TPR respectively.

The ROC curve (Hanley and McNeil, 1982) is defined as the curve that plots the TPR as a function of FPR, which we can formalize as

$$ROC(t) = TPR\left(FPR^{-1}(t)\right)$$

We can verify that in the case where there is no meaningful difference between $P(f(x,\theta)|y=1)$ and $P(f(x,\theta)|y=0)$ (e.g if a random prediction label is assigned) the two integrals for $FPR(t)$ and $TPR(t)$ will reduce to be equal and the ROC

curve will the line where $FPR(t) = TPR(t)$. Now, for any classifier of decent quality (that is, better than random), $FPR(t)$ will be larger than $TPR(t)$ for all t.

We now have a way to quantify the trade-off between the FPR and TPR for a given value of t, as the ROC function gives us the TPR associated with any given FPR for t. We can capture the total trade-off regardless of t by integrating the $ROC(t)$ over t, giving us the ROC Area under the curve (AUC) (Fawcett, 2006):

$$AUC = \int_0^1 TPR\left(FPR^{-1}(t)\right) dt$$

it can furthermore be shown (Hanley and McNeil, 1982; Calì and Longobardi, 2015) that the AUC is equal to the probability that a randomly chosen positive instance will have a higher predicted score $f(x, \theta)$ than a randomly chosen negative instance, more formally defined as

$$AUC = P\left(f(x^+, \theta) > f(x^-, \theta)\right) \text{ where } x^+ \overset{\text{iid}}{\sim} p(x|y = 1),\ x^- \overset{\text{iid}}{\sim} p(x|y = 0)$$

It can be observed that $AUC = 0.5$ for a classifier that outputs random class labels and that $AUC = 1$ for classifiers that can fully separate all positive from negative cases. All in all, the AUC metric proves to be an efficient metric to capture the ability of a classifier to discriminate between positive and negative classes.

Some other variants of the AUC metric are used. For example, we could integrate the ROC curve over any arbitrary range of f, resulting in the partial AUC (pAUC) (McClish, 1989). This is useful in applications where it is known that some areas in the ROC space are certainly irrelevant because of high FPR values.

The derivation of the ROC AUC metric above leans on a few assumptions that are not feasible in practice. Firstly, it assumes we know the distribution $p(f(x, \theta)|y)$ and secondly it assumes this distribution is continuous. We must therefore rely on an estimate of the AUC based on the empirical data distribution. This could be done parametrically by fitting a distribution function to the empirical distribution and computing the AUC analytically, but is commonly done in a nonparametric way. One nonparametric way is to simply evaluate the integral numerically using a trapezoid method. This means that the FPR and TPR are evaluated for a range of values for t after which the area under the ROC curve is estimated by interpolating linearly between points.

Another nonparametric method is closely related to the Mann-Whitney U test (Mann and Whitney, 1947). This is a nonparametric test that tests wether two samples originate from the same distribution. It uses the test statistic

$$U_1 = \sum_{x^+ \in\ \mathcal{D}^+} \sum_{x^- \in\ \mathcal{D}^-} \mathbb{1}_{f(x^-,\theta) < f(x^+,\theta)}$$

where $\mathcal{D}^+$ and $\mathcal{D}^-$ and are our data samples of positive and negative instances respectively. This test statistic can be transformed to the AUC metric according to $\hat{AUC} = U_1(|\mathcal{D}^+||\mathcal{D}^-|)^{-1}$, which is an unbiased estimator for the AUC.

In practice, estimated AUC values are often compared in order to compare model performance. In some fields, such as the medical sciences, where datasets typically have a low number of samples, inference methods are used to obtain significance of conclusions (Qin and Zhou, 2006; Ekström et al., 2023). In the machine learning world, where datasets typically have a large number of samples, this is often neglected. It is simply assumed that the sample size is large enough for the estimate of the AUC to have a sufficiently low variance.

2.4 Representation learning

We previously covered Artificial Neural Networks in the context of function estimation, a paradigm in which they prove to be extremely powerful. One of the main factors that this is commonly attributed to is the ability of an ANN to learn "implicit feature representations" of the input data (Bengio et al., 2014). This means that no explicit (manual) feature engineering is required for the model to achieve good performance. This phenomenon can be exploited to obtain efficient efficient feature encoders that can be used as standalone models. We denote an encoder

$$G : \mathbb{R}^d \to \mathbb{R}^r, \quad x \mapsto G(x), \quad d \gg r$$

that maps a raw feature vector from the data space $\mathbb{R}^d$ to a low dimensional representation in $\mathbb{R}^r$, through some (nonlinear) transformation parametrized by an ANN. The idea is that the encoder creates compact (low-dimensional) and informative features that can be used in downstream learning tasks. Learned features are universal and can be used in a set of different applications. This section discusses some methods for estimation of the encoder $G(\cdot)$.

2.4.1 Self-supervised Representation Learning

In unsupervised learning, the goal is to learn informative features without any data label. One method proposed to achieve this is self-associative learning (Rumelhart et al., 1986), a paradigm more commonly referred to as auto-encoders (Hinton and Salakhutdinov, 2006). The core idea is to train an encoder $G(\cdot)$ together with a decoder $D(\cdot)$. The encoder maps the input vector x to some compressed representation z, which is then decompressed by the encoder, s.t $\hat{x} = D(G(x))$. Intuitively, the auto-encoder learns to compress input data, forced by an information bottleneck in the network. Auto-encoders are trained by minimizing a reconstruction loss between the original data x and the reconstructed data $\hat{x}$.

There is a noteworthy relationship between the auto-encoder ANN and Principal Component Analysis (PCA). PCA, first proposed by Pearson (1901) is a dimensionality-reduction technique where the principal component matrix $\mathbf{\Lambda}$ is the result of the optimization problem

$$\min_{\mathbf{\Lambda}} \sum_{i=0}^{N} ||x_i - \mathbf{\Lambda}(\mathbf{\Lambda}' x_i)||^2 \tag{2.2}$$

subject to

$$\mathbf{\Lambda}\mathbf{\Lambda}' = I_r$$

which is the minimization of the mean squared error (MSE) under orthogonality constraints. It is easy to see that, in the case of using an MSE loss in the auto-encoder ANN paradigm, the optimization is essentially the same. Now, if we also take $G(\cdot)$ and $D(\cdot)$ to be linear mappings, the space spanned by the learned representations z spans the same space as the principal component latent space. A particularly interesting result in this regard is presented by Bourlard and Kamp (1988), who show that an autoencoder network, with one hidden layer and one nonlinear activation in both the encoder and decoder, does not learn more informative representations $\boldsymbol{z}_i$ than PCA. We can consider autoencoders a form of non-linear PCA and this result tells us that it is useless to use an autoencoder with one hidden layer for nonlinear dimensionality reduction, as the obtained representations are not better than the principal components.

Autoencoders have proven to be efficient feature learners, with applications spanning multiple domains from computer vision (CV) (He et al., 2021) to natural language processing (NLP) (Oshri, 2015) and anomaly detection (Sakurada and Yairi, 2014). Several extensions have been proposed and used over the years. One extension is the denoising autoencoder. Proposed by Vincent et al. (2008), this method adds some i.i.d noise to the input and trains the autoencoder to reconstruct the original instance of the input. Another noteworthy example is the masked autoencoder (He et al., 2021), where a part of the input is masked and reconstructed by the autoencoder. Both these techniques are de facto regularization techniques, aiming to increase robustness in the encoder and obtaining better generalization accuracy.

2.4.2 Supervised representation learning

Supervised representation learning is an umbrella term for all those learning methods where data labels are used to in some way to obtain informative feature encoders. In its simplest form, this encoder could consist of the first set of hidden layers from a multilayer perceptron (MLP) that was trained for a binary classification task. The hidden state of the network can be considered a vector of implicitly learned features of the input data. In more general terms, supervised representation learning aims to obtain a well-structured embedding space of the input data where the structure is

somehow guided by the labels associated with the input data. The resulting structure in the embedding space should then be more informative of the target label than without supervision.

We focus the rest of this section on metric learning (Xing et al., 2002). A fundamental idea in metric learning is to train an ANN by optimizing some distance metric that provides a measure of similarity between two (or multiple) instance embeddings. In its simplest form, we could have a distance metric $d(z_i, z_j)$ that accepts two embeddings z_i, z_j and outputs a single scalar indicating whether these two instances are nearby or far apart in the embedding space. The data labels can be used to determine whether we want them closeby or far apart in this space. We define the loss function

$$l_{i,j}(x_i, x_j) = \begin{cases} d\big(G(x_i), G(x_j)\big), & \text{if } y_i = y_j \\ -d\big(G(x_i), G(x_j)\big), & \text{if } y_i \neq y_j \end{cases}$$

where $G(\cdot)$ is the encoder parameterized by an ANN and $d(\cdot)$ is some distance metric. This loss aims to nudge the encoder to push similar samples toward each other and dissimilar samples away from each other. This technique is mostly used in the CV domain, with the Siamese network (Chopra et al., 2005) as the most common example. This is a metric-learning technique that learns image embeddings that can be utilized to decide whether two images are similar, with applications such as face recognition.

One common choice for the distance $d(z_i, z_j)$ is the cosine similarity (Hinton and Salakhutdinov, 2006)

$$d(z_i, z_j) = \frac{z_i \cdot z_j}{\|z_i\|\|z_j\|}$$

which has range $[-1, 1]$, where a similar pair of vectors will result in a similarity close to 1 and an extremely dissimilar (that is, pointing in the opposite direction) pair of vectors will be close to -1.

Schroff et al. (2015) propose a loss function that uses three sampled instances from the training set instead of two, yielding the triplet-loss function. We define a triplet of data instances $(x_a,\ x_p,\ x_n)$. x_a is some anchor instance. x_p is an instance that, together with x_a forms a positive pair (that is, they belong to the same class). x_n is an instance that forms a negative pair with x_a. For each data instance in the triplet, the embedding is obtained using $G(\cdot)$ to obtain $z_a,\ z_p,\ z_n$ which is a representation of the triplet in embedding space, as displayed in figure 2.2 . Evidently, we desire z_a and z_p to be close in embedding space and z_a and z_n to be distant. We can define the loss function

$$\ell_i(z_i, z_p, z_n) = d(z_i, z_p) - d(z_i, z_n)$$

to capture how well structured the embedding space considering the triplet $(x_a,\ x_p,\ x_n)$. Schroff et al. (2015) furthermore add a margin α to this loss and define the triplet loss function

$$\ell_i(z_i, z_p, z_n) = \left[d(z_i, z_p) - d(z_i, z_n) + \alpha\right]_+$$

where $[\cdot]_+$ is the hinge function that clips values below 0 to 0. This loss function ensures that if $d(z_i, z_p) + \alpha < d(z_i, z_n)$ holds, the loss is clipped to 0 and effectively neglected. This effect ensures that only those triplets that are not well distanced are considered in the loss function.

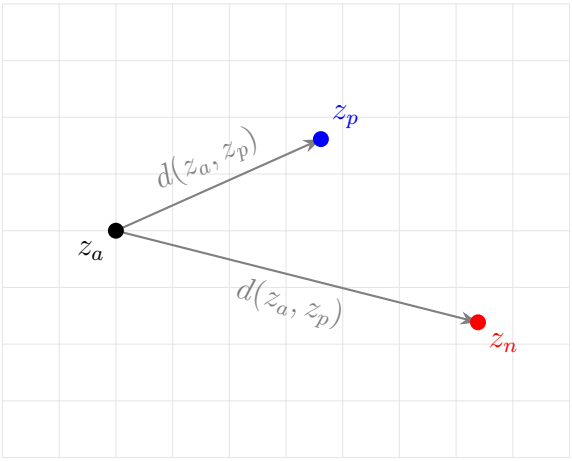


Figure 2.2: Visual representation of the triplet loss function in embedding space. $d(z_a, z_p)$ indicates the distance between anchor and positive and $d(z_a, z_n)$ indicates the distance between anchor and negative. In this visualization the desired effect of $d(z_a, z_p) < d(z_a, z_n)$ holds. In practice, the embedding space is high-dimensional rather than two-dimensional.

As distance metric, the triplet loss as proposed by Schroff et al. (2015) uses the squared L2 norm $d(z_i, z_j) = \|z_i z_j\|_2^2$ but other norms could be used as well. The loss function over the whole triplet sample is defined as

$$\sum_{(x_a,\ x_p,\ x_n)\in\mathcal{T}} \Big[d\big(G(x_a), G(x_p)\big) - d\big(G(x_a), G(x_n)\big) + \alpha\Big]_+ \tag{2.3}$$

where $\mathcal{T}$ is some sample of triplets. The notion of a sample of triplets encompasses a range of sampling strategies. The final goal is to construct a set of triplets $\mathcal{T} = \{(x_i^a, x_i^p, x_i^n) \mid i = 1, 2, \ldots, N\}$ that, when used within the triplet loss learning framework, results in a well structured embedding space. Several sampling strategies for constructing $\mathcal{T}$ have been proposed over the years, which can be categorized in *offline* and *online* sampling strategies.

Offline sampling strategies include all sampling strategies where $\mathcal{T}$ is constructed without knowledge on the state of the encoder $G(\cdot)$. This means the sampling procedure does not depend on the training procedure and $\mathcal{T}$ can be constructed before

training starts. An example of such a sampling procedure is a procedure where for each instance $x \in \mathcal{D}$ we take $x^a = x$ and sample $x_p \sim \mathcal{D}^+$ and $x_n \sim \mathcal{D}^-$ to form the triplet. A simple extension to this would be to not take every $x \in \mathcal{D}$ but sample $N/2$ instances from $\mathcal{D}^+$ and $N/2$ from $\mathcal{D}^-$. This procedure would be suitable for imbalanced class distributions.

The offline sampling procedures are inherently simple sampling strategies. Online sampling allows for much more complex procedures, sampling triplets during the training procedure using the state of the encoder $G(\cdot)$ at each training step. One such procedure is *Batch Hard* mining (Hermans et al., 2017). The fundamental idea underlying this strategy is to select positives for an anchor that are, for the current state of the encoder $G(\cdot)$, far apart in embedding space and negatives that are close in embedding space. This means that at each training step, we select the triplets that would produce a high loss $\ell_i(z_i, z_p, z_n)$. Mining the set of triplets requires inference of $G(\cdot)$ and evaluation of the loss function, which is computationally complex. To overcome this, the strategy is executed on mini-batches of data. We can formally define the Batch Hard loss for a binary-class setting on a mini-batch $\mathcal{B} \subseteq \mathcal{D}$ of size $|\mathcal{B}| = N_\mathcal{B}$ as

$$L_{BH}(\mathcal{B}) = \sum_{x \in \mathcal{B}^+} \left[\max_{x_p \in \mathcal{B}^+} d(x_a, x_p) - \max_{x_n \in \mathcal{B}^-} d(x_a, x_n) + \alpha \right]_+ + \sum_{x \in \mathcal{B}^-} \left[\max_{x_p \in \mathcal{B}^-} d(x_a, x_p) - \max_{x_n \in \mathcal{B}^+} d(x_a, x_n) + \alpha \right]_+$$

Other noteworthy online triplet mining strategies include *Distance Weighted* mining (Wu et al., 2018), *Easy Batch* mining (Xuan et al., 2020).

2.5 Interpreting the embedding space

In the previous section, we discussed representation learning and the *embedding* space. The embedding space offers a projection of feature instances $x \in \mathbb{R}^d$ to an embedding $z \in \mathbb{R}^r$. The exact dimensionality r is typically regarded as a hyper-parameter and is commonly selected relative to the dimensionality of the input data. A typical value of r lies in the range $[8, 256]$. Per illustration Hermans et al. (2017) and Wu et al. (2018) use $r = 128$ and Xuan et al. (2020) use $r = 64$. It is often valuable to inspect an embedding space in order to confirm and understand the learned representations. Unfortunately, high-dimensional spaces are ungraspable for human cognition. We commonly try to overcome this by visualizing the r-dimensional embedding space in two dimensions. Two mainstream methods are typically used to do this.

One obvious method to reduce a high-dimensional space to a two-dimensional one is PCA. PCA identifies the directions of maximum variance in the high-dimensional space (called principal components). It then projects the data points onto a lower-dimensional subspace (or plane) defined by the top principal components, thereby

reducing the dimensionality while preserving the most significant variation in the data. The exact optimization performed is defined in equation 2.2. A downside of PCA is that it is a fundamentally linear projection, and can fail to capture nonlinear high-dimensional local structures.

A solution to the linearity issue in PCA is found in the t-SNE method. t-SNE (Van der Maaten and Hinton, 2008) is a stochastic neighbor embedding (SNE) method that is useful for capturing the structure of non-linear high-dimensional manifolds. The method, an extension to vanilla SNE (Hinton and Roweis, 2002) defines the pairwise similarity $p_{j|i}$ to be the similarity between z_i and z_j in the embedding space. A Gaussian distribution is chosen to model $p_{j|i}$ resulting in

$$p_{j|i} = \frac{\exp\left(-\frac{\|z_i - z_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|z_i - z_k\|^2}{2\sigma_i^2}\right)}$$

where σ_i is set by specifying a target complexity (we refer the reader to Van der Maaten and Hinton (2008) for details on this). We observe that this is in fact the probability that point j would be selected if we were at point i if the probabilities were proportional to the distance from point i to j and followed a Gaussian distribution. It is obvious to set $p_{i|i} = 0$. Furthermore, we denote the symmetrized conditional probability

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

where n is the number of data points. Next, we define q_{ij} to be the symmetric conditional probability of z_i choosing z_j as its neighbor (and the other way around) in the low-dimensional representation (typically two- or three-dimensional). One fundamental principle underlying t-SNE is that q_{ij} and p_{ij} should be similar between the high-dimensional and low-dimensional space for all data points because the local structure should not change (points close to each other in high dimensional space should be close in low dimensional space). In t-SNE, q_{ij} is parametrized by a Student-t distribution with one degree of freedom (in the original SNE method by Hinton and Roweis (2002) this was also a Gaussian), such that

$$q_{ij} = \frac{(1 + \|v_i - v_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|v_k - v_l\|^2)^{-1}}$$

where v_i is the low-dimensional representation of point z_i. We now have a formulation for two probability distributions p_{ij} and q_{ij} that we desire to be similar. We can trivially define the difference between the two distributions using the Kullback-Leibler divergence to obtain

$$KL(P\|Q) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

where P and Q are the joint probability distributions related to p_{ij} and q_{ij}. The goal now is to find the low-dimensional vectors v_i $i = 1, ..., n$ that minimize $KL(P\|Q)$. Van der Maaten and Hinton (2008) derive the following gradient of the $KL(P\|Q)$ with respect to y_i:

$$\frac{\partial KL(P\|Q)}{\partial v_i} = 4 \sum_j (p_{ij} - q_{ij}) (v_i - v_j) \left(1 + \|v_i - v_j\|^2\right)^{-1}$$

This expression can be used to perform gradient descent to find the low dimensional representations v_i $i = 1, ..., n$ that are produced through a non-linear transformation with respect to the embedding space. The t-SNE projection of an embedding space has proven to be extremely insightfull and is used in most work on representation learning, amongst others Hermans et al. (2017),

Chapter 3

Benchmark Study

Based on existing literature, we select a set of methods to serve as a benchmark for the two datasets used in this work. This selection is based on a broad review of the literature and includes the methods typically used in research and industry. We note that, in the field of fraud detection, many industry leaders keep their methodologies proprietary. However, we believe the methods in this benchmark serve as an adequate representation of what is typically used in real-world systems.

3.1 Datasets

For a multitude of reasons, many payment processors across the industry do not provide access to datasets for scientific research on fraud detection. On the one hand, this payment data contains personally identifiable information (PII), which is legally stipulated to be kept confidential and secure. Secondly, this data gives competitors undesired insights into a company's internals, diminishing the company's competitive advantage. Lastly, from a fraud-prevention point of view, data may be used by harmful actors to analyze and circumvent fraud detection systems.

For these reasons, high-quality public fraud datasets are practically non-existent. The best option currently available in the public domain is a dataset released by Dal Pozzolo et al. (2014). This is a dataset (further denoted ULB300k), consisting of 284,807 transactions including 492 fraudulent transactions, collected by a French multinational payment processor in September 2013. The dataset counts 31 features, for which the exact names and meanings are undisclosed. One row in this dataset would look like $[x_1, x_2 \dots x_{30}, x_{31}, y]$ where $y \in \{0, 1\}$

This work further uses a proprietary dataset collected by a Dutch multinational fraud-prevention company that offers a fraud detection system to a wide range of merchants. The dataset consists of a subset of transactions spanning from 2015 to 2023. In total, the dataset (further denoted as PER40M) contains 40 million records and 150 features. Among the features, we find both numerical and categorical

features. The dataset has a fraud rate of 0.7% (This is not necessarily representative of the real-world class distribution).

3.2 Benchmark methods

This section introduced the methods used in the benchmark study. These methods are selected based on a broad literature review and should serve as an accurate representation of the current use in the industry. All these methods are used in combination with the SMOTE (Chawla et al., 2002) technique for dealing with class imbalance. We put a strong emphasis on the avoidance of hyperparameter selection bias, where the baseline hyperparameters would be suboptimal due to a poor selection effort, while the newly proposed method is heavily optimized. We achieve this by selecting relevant hyperparameters to tune in each of the selected benchmark methods. Hyperparameters are tuned using a 5-fold cross-validation grid-search strategy. Table B.1 shows each benchmark method and the selected parameters to tune, as well as a reference for the implementation details. In this benchmark, the MLP classifier concerns shallow architectures (less than 2 hidden layers). We do report a full set of results for larger MLP architectures. In our results, we further report the values for each hyperparameter used in the final model.

Table 3.1: Benchmark Models and Hyperparameter Tuning Details

Model	**Tuned Variables**	**Implementation Reference**
XGBoost	max_depth n_estimators alpha lambda	Chen and Guestrin (2016)
Logistic Regression	C L1_ratio	Pedregosa et al. (2011)
Random Forest	max_depth n_estimators	Pedregosa et al. (2011)
MLP	hidden_layer_sizes activation	Pedregosa et al. (2011)
Decision Tree	max_depth	Pedregosa et al. (2011)

3.3 Evaluation procedure

In section 2.1.5 we highlighted some practical implications arising from the theoretical ERM framework. One of these implications is the need for dealing with generalization inaccuracies. In our experimental studies, we evaluate performance based on a classical train-test split, which means a subset of the complete dataset

is hidden from the training procedure and only used for model testing. One consideration here is that, because of the unbalanced nature of the class distribution, we need the test set to contain enough positive (fraud samples) to compute the desired performance metrics.

For the ULB300K dataset, which contains only 492 fraudulent transactions, the class imbalance forms an issue in the construction of the test set. For this dataset, we therefore construct the training- and test datasets such that each dataset has a fraction $\frac{N_{neg}}{N}$ of negative samples (this can arbitrarily be achieved by sampling the positive and negative samples separately and concatenating the result sets). This ensures that the class balance is equal between both sets.

For the PER40M dataset, this problem is less prevalent due to an abundance of negative data samples. For this dataset, we choose a time-based test-split. This means that we use the last $r_{test} * N$ records in the test set, where r_{test} is the fraction of records used in the test set. This strategy ensures that methods are not only judged based on their generalization accuracy with respect to sample size but also with respect to time. Per illustration, a model could overfit to a certain period where some pattern is overly present in the data. An alternative evaluation procedure would be to use a rolling-window framework, where models are estimated sequentially on an expanding training set through time. For simplicity and computational complexity reasons, we opted not to use such a method.

We report the evaluation metrics ROC AUC, accuracy, precision and recall. Precision and recall are reported for the positive label. We a focus on the ROC AUC, as mentioned in 2.3.2, because is not dependent on the specific tuning of the decision threshold (which precision and recall are). This metric is computed using the trapezoidal method. The described evaluation procedure remains fixed through the rest of this work.

3.4 Model complexity estimation

In order to reveal the trade-off between inference complexity and model performance we make an estimate for each method inference complexity. We use two metrics for this. Firstly, we consider the number of floating point operations (FLOPs) a model requires to produce a prediction. This metric (not to be confused with floating point operations per second) counts the number of elementary floating point operations required to evaluate some computation. For each benchmark method, we report the number of inference FLOPs. This number depends on the selected hyperparameter configuration.

3.5 Results

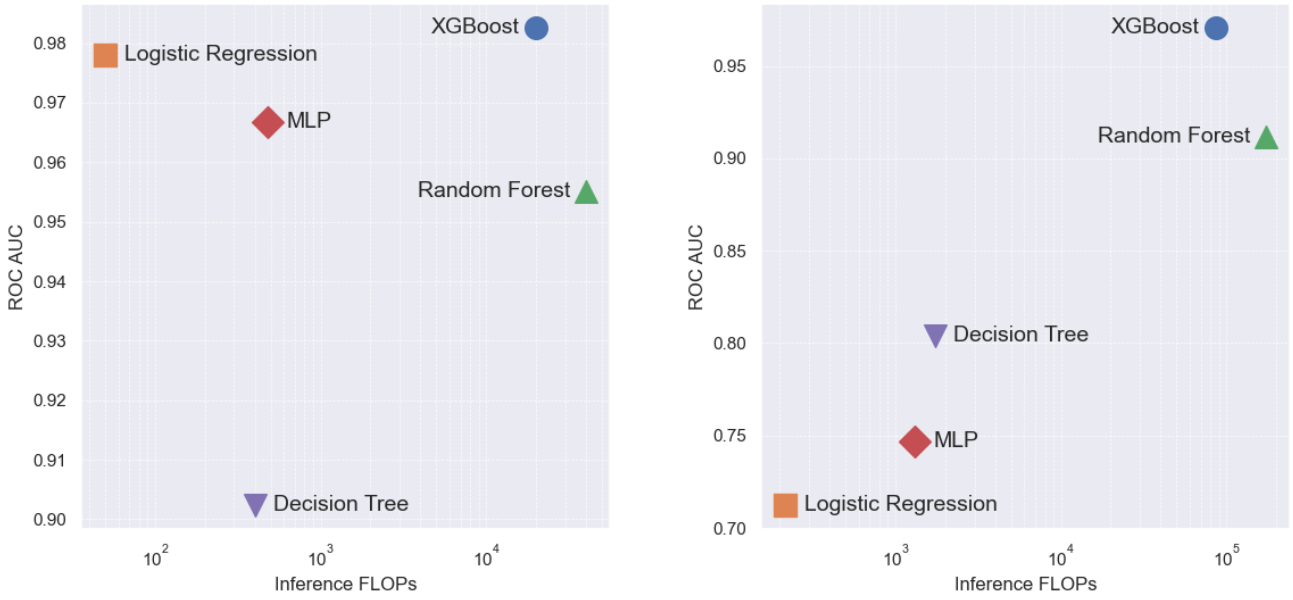
The resulting values for the hyperparameters for the benchmark models are shown in table B.1. Table 3.2 shows the benchmark results for the ULB300K dataset. Table 3.2 shows the benchmark results for the PER40M dataset. We observe that, from these baselines, the XGBoost method outperforms the other benchmarks for both baselines. Logistic regression performs surprisingly well on the ULB300K dataset and rather poorly on the PER40B dataset. Appendix C gives a brief overview of how the inference FLOPs for each method are determined, while Appendix B.3 provides an additional set of results for different MLP configurations. Figure 3.1 displays the FLOPs-performance trade-off. These figures reveal that only Logistic Regression, Decision Tree, and XGBoost can be considered efficient models. That is, there is no model with better performance at lower inference complexity.

Table 3.2: Benchmark results on the ULB300k dataset.

Method	ROC AUC	Accuracy	Precision	Recall
XGBoost	**0.9826**	0.9996	**0.9379**	0.8120
Logistic Regression	0.9781	0.9809	0.8436	0.9060
MLP	0.9667	0.9993	0.9035	0.6912
Random Forest	0.9550	0.9996	**0.9379**	**0.8120**
Decision Tree	0.9023	**0.9990**	0.6818	0.8053

Table 3.3: Benchmark results on the PER40M dataset.

Method	ROC AUC	Accuracy	Precision	Recall
XGBoost	**0.9703**	**0.9830**	**0.9216**	**0.7533**
Random Forest	0.9114	0.9825	0.7926	0.4047
Decision Tree	0.8038	0.9899	0.5443	0.3906
MLP	0.7468	0.9897	0.6881	0.1829
Logistic Regression	0.7123	0.9892	0.4600	0.3462



(a) ULB300K

(b) PER40M

Figure 3.1: Inference FLOPs vs. ROC AUC for the benchmark methods on the ULB300K and PER40M datasets. This plot compares the computational inference cost (FLOPs) and predictive performance (ROC AUC) of different machine learning models. The x-axis represents the inference FLOPs on a logarithmic scale, indicating the computational effort required for each model to process a single input. The y-axis shows the ROC AUC score, a measure of classification performance. Any method that has both a lower ROC AUC and higher inference FLOPs can be considered inefficient, as better performance can be achieved with lower inference complexity.

Chapter 4

Self-supervised Representation Learning

This chapter focuses on the application of self-supervised learning to fraud detection. In this chapter we aim to answer the following questions:

1. Do feature encoders obtained through self-supervised result in useful features for downstream fraud classification?

2. How does the performance of the encoder depend on the network size?

The chapter is structured as follows. Section 4.1 introduces the model framework. Section 4.1.1 describes a method for dealing with class imbalance in the learning setting. Section 4.2 discusses the experimental setup followed by 4.3 which contains a detailed analysis of the results and findings.

4.1 Self-associative learning for fraud detection

As discussed in section 2.4.1, the main goal of self-associative learning is to train a feature encoder $G(x)$ that efficiently encodes feature vector x_i. The encoder is trained in an autoencoder setting, which means that the embedding z_i is obtained by $z = G(x_i)$ and $\hat{x}_i$, and the reconstructed feature vector is obtained by $\hat{x}_i = D(G(x_i))$. The system is trained on the reconstruction loss

$$l(x_i) = (x_i - D(G(x_i))^2$$

for feature vector x_i. In this work, we exclusively use symmetric autoencoder structures, where the encoder and decoder are a vanilla MLP structure. The hidden layer sizes of the decoder are equal to the encoder but in reversed order. Subsequently, the embeddings produced by feature encoder $G(\cdot)$ are used to fit a classifier $f(z)$. Several model options are common for $f(\cdot)$. These will be discussed in more detail in 4.2.

4.1.1 Dealing with class imbalance

The problem of fraud classification is characterized by the inherent imbalanced class distribution. A strategy to mitigate this issue is required. Several options have been proposed and used over the years, most notable are SMOTE (Chawla et al., 2002), a synthetic oversampling technique, and weighted loss functions (Fernando and Tsokos, 2022). In this chapter, we use a basic oversampling procedure where the infrequent class is oversampled so that each batch $\mathcal{B} \in \mathbb{R}^{N_\mathcal{B} \times d}$ of size $N_\mathcal{B}$ has a fixed ratio between both classes, denoted $r_\mathcal{B} = \frac{N_\mathcal{B}^+}{N_\mathcal{B}}$, where $N_\mathcal{B}^+$ is the number of positive instances in the batch. Throughout the experiments in this chapter, we set $r_\mathcal{B} = 0.5$, such that there is an equal number of positive and negative samples in the batch. The full training procedure, including this sampling strategy, is shown in algorithm 1.

Algorithm 1 The training procedure for training encoder G_{θ_G}, parametrized by θ_G in an autoencoder setting.

Initialize θ_F, θ_G
for each epoch **do**
  for batch $\mathcal{B}^-$ in dataset X^- **do**
    sample $\mathcal{B}^+$ from X^+ and construct $\mathcal{B}$
    compute batch loss $\mathcal{L}(\mathcal{B}) = \sum_{x_i \in \mathcal{B}} (x_i - D(G(x_i))^2$
    execute gradient step $\theta_G \leftarrow \theta_G - \eta \frac{\partial \mathcal{L}}{\partial \theta_G}$
    execute gradient step $\theta_D \leftarrow \theta_D - \eta \frac{\partial \mathcal{L}}{\partial \theta_D}$
  end for
end for

Table 4.1: Experiment configurations evaluated for the auto-encoder setting. The code is a unique identifier for the model configuration. The number of parameters # Paramaters is the total number of weights in the autoencoder setup (both encoder and decoder). The Encoder structure column indicates the sizes of the hidden layers used in the encoder (and in the decoder but reversed.

Code	Encoder structure	#Parameters	Classifier	ULB300K	PER40M
AE1K-ILF	[16]	1.5K	IF	✓	
AE1K-KNN	[16]	1.5K	KNN	✓	
AE30K-ILF	[128, 64, 32]	29.2K	IF	✓	✓
AE30K0-KNN	[128, 64, 32]	29.2K	KNN	✓	✓
AE100K-ILF	[256, 128, 64, 32]	102.4K	IF	✓	✓
AE100K-KNN	[256, 128, 64, 32]	102.4K	KNN	✓	✓
AE400K-ILF	[512, 256, 128, 64, 32]	379.9K	IF	✓	✓
AE400K-KNN	[512, 256, 128, 64, 32]	379.9K	KNN	✓	✓
AE1.5M-ILF	[1024, 512, 256, 128, 64, 32]	1.5M	IF	✓	✓
AE1.5M-KNN	[1024, 512, 256, 128, 64, 32]	1.5M	KNN	✓	✓

4.2 Experimental setup

The experimental setup is largely equal to the setup described in section 3.3. This means that the train and test set are equal for both experiments, such that we can safely compare the newly proposed methods and the benchmarks. From the training set, a validation set is sampled at random which is used for diagnostic purposes.

As we are interested in the scaling properties of the auto-encoder learning methods, several model configurations are tested, each with increasing model size. The ReLU nonlinearity is used in all configurations. We scale up the model architecture by adding a hidden layer twice the size of the smaller model's largest hidden layer. For each configuration, a validation loss is tracked such that the loss curves for several model sizes can be compared. We report the validation loss as a function of training FLOPs, a quantification of the total training resources used by a model. The FLOP count is obtained using the *fvcore* python package published by Facebook Research.

We select K-nearest neighbors (KNN) (Fix and Hodges, 1989) and Isolation Forest (IF) (Liu et al., 2008) as the two downstream classifier models. These are both non-parametric methods that work well in continuous dense feature spaces. The KNN method has one significant downside, which is that it cannot efficiently handle large datasets. We therefore need to limit the number of records used for training the KNN classifier for the PER40M dataset to 200.000 (for the ULB300K dataset

this is not needed as the feature dimensionality is smaller).

Table 4.1 shows the experiment configurations used for both datasets. As the table indicates, not all configurations are used for both models. In particular, the smaller model is not used for the PER40M dataset. In that regard, we further note that an embedding dimension $r = 8$ for the ULB300K dataset and $r = 16$ for the PER40M dataset.

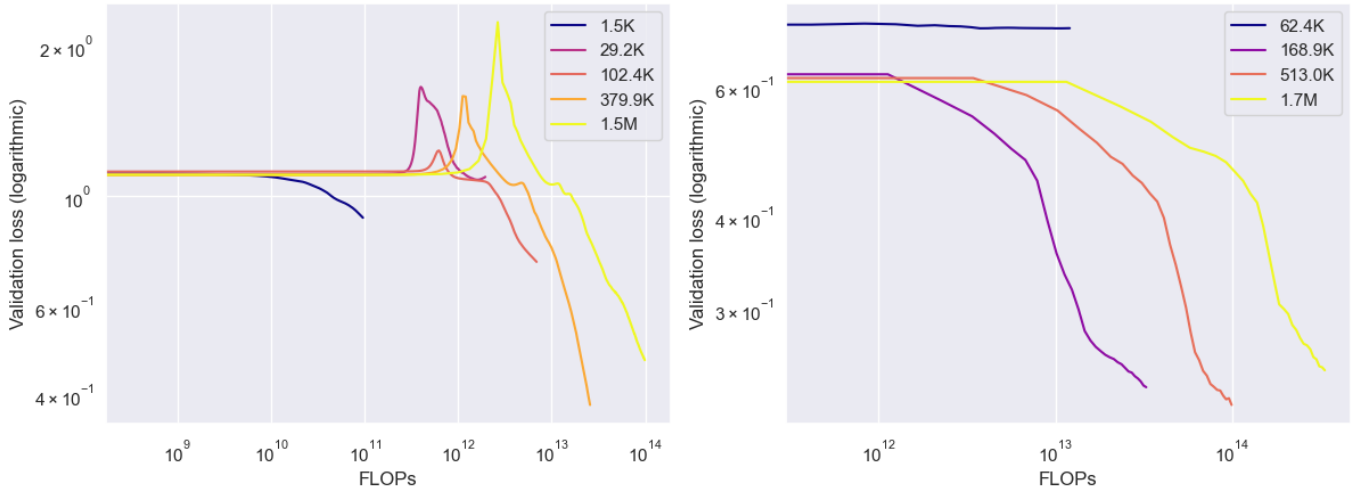
In the results, for both datasets, we also report a visual representation of a two-dimensional t-SNE (see section 2.5) representation of the embedding space produced by the encoder associated with the highest AUC score on the test set. This visualization, which includes 1000 randomly sampled positive and 1000 sampled negative instances from the validation set, allows us to interpret the learned embedding space. In a well-structured, informative embedding space, we expect to see a noticeable difference between a cluster including the positive and a cluster including the negative samples.

4.3 Results

4.3.1 Scaling properties

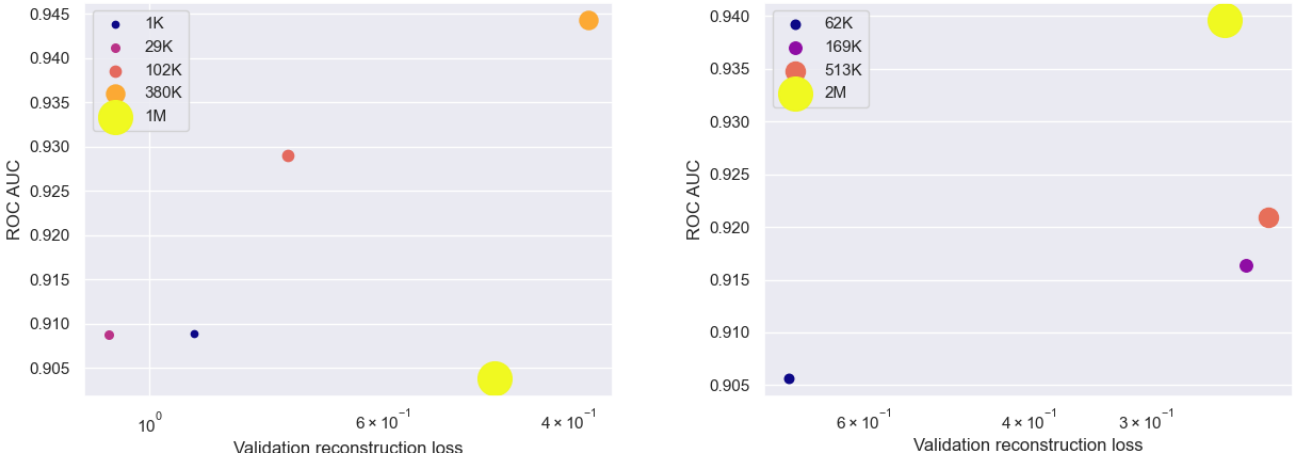
Figure 4.1 shows the validation reconstruction loss vs the number of training FLOPs for the training procedures for both datasets. In general, we observe that the final validation loss is lower for larger model sizes. However this scaling property does not hold for the last model size, which, for both datasets shows a higher final validation loss for both datasets. For the ULB300K dataset, we observe the presence of the double-descent phenomenon (Nakkiran et al., 2019). This occurs when the validation loss increases during a critical period where the training procedure transitions from the *traditional regime* to the *modern regime*.

A low reconstruction loss is only meaningful if it leads to a higher final classification performance. Figure 4.2 shows the relation between the KNN test ROC AUC and the auto-encoder validation loss. The size of each point represents the model size. For the ULB300K dataset, we observe that the encoder with the lowest validation loss also obtains the highest ROC AUC, however this does not generally hold. For the PER40M dataset, we see that the 2M model obtains the highest test ROC AUC, while it does not achieve the lowest validation loss. This is easily explained when one considers that the reconstruction loss is an average over both classes and does not directly quantify the usefulness of the resulting embedding space.



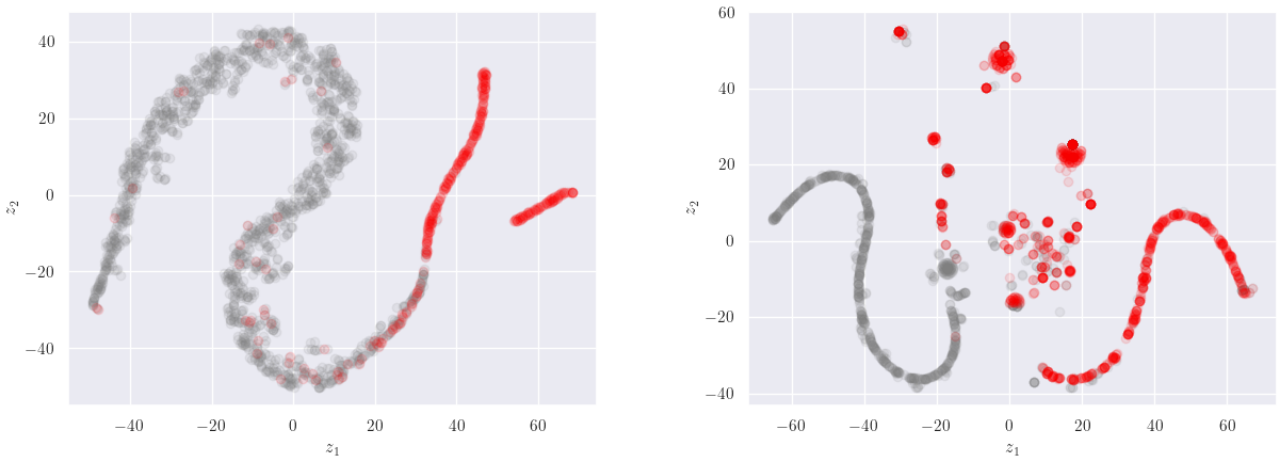
(a) ULB300K (b) PER40M

Figure 4.1: Validation loss vs. training FLOPs for both the autoencoder training method for both datasets. The ULB300K dataset shows a particular form of double-descent, where the validation loss first increases and then decreases again, whereas the PER40M shows rather stable convergence. These figures show that for both datasets, encoding performance does not scale linearly with model size, as the best validation losses are obtained for the second-to-largest models and not the largest.



(a) ULB300K (b) PER40M

Figure 4.2: Test ROC AUC obtained by KNN classifications vs. validation loss for the autoencoder training method for both datasets. These figures show that a lower validation reconstruction loss does not always indicate a higher performance on the downstream classification task.



(a) ULB300K

(b) PER40M

Figure 4.3: Two-dimensional t-SNE representations of the embedding spaces for 1000 positive (red) and 1000 negative (grey) sampled instances for both datasets. The ULB300K dataset shows an embedding space where positive and negative instances are semi-separated, with the two clusters having significant overlap. As for the PER40M dataset, the embedding space shows some degree of separation between the two classes but has areas with a large overlap.

4.3.2 Embedding visualizations

Figure 4.3 shows a two-dimensional t-SNE visualization of the embedding space for a random sample of instances from the training set. The encoder is taken from the best-performing model based on the ROC AUC. We observe a clear structure in the ULB300K embedding space, where positive embeddings are visibly separated from positive ones. However, there is still a large area where the negative and positive clusters overlap. For the PER40M dataset, the embedding space is visibly different from the ULB300K embedding space. In this embedding space, there is separation to some extent, but the space also contains many small clusters of positive instances cluttered around. Overall, these figures indicate that the learned features should be relevant, at least to some extent, for the downstream classification performance.

4.3.3 Classification performance

Tables 4.2 and 4.3 report all the relevant classification metrics on the test set for the configurations as introduced in 4.1. Overall, the KNN classifier works remarkably well on the PER40M dataset, even though its training was limited by a smaller training set. On the other hand, the ILF method works poorly on the PER40M dataset, as it does not beat KNN on any of the model sizes. Overall, the AE1.5M-KNN method performs best based on the ROC AUC. It is noteworthy that, as displayed in figure 4.2, this configuration did not achieve the lowest reconstruction loss.

For the ULB300K dataset, this behavior is completely different. KNN performs poorly and ILF is the better classifier across the board, with the AE400K-ILF configuration obtaining the highest ROC AUC score. The AE30K configuration is a remarkable outlier, as it performs significantly worse than its smaller variant AE1K for both KNN (AE1K-KNN) and IF (AE1K-IF). This can be explained by a poor convergence of the reconstruction loss as displayed in figure 4.1a.

Code	ROC AUC	Accuracy	Precision	Recall
AE1K-ILF	0.9743	0.9321	0.8882	0.9191
AE1K-KNN	0.9139	0.8995	0.9995	0.8080
AE30K-ILF	0.9119	0.8830	0.8979	0.8989
AE30K-KNN	0.8885	0.9992	0.9091	0.6969
AE100K-ILF	0.9623	0.9158	0.8981	**0.9290**
AE100K-KNN	0.9442	**0.9994**	0.9194	0.7373
AE400K-ILF	**0.9876**	0.8825	**0.9983**	0.9191
AE400K-KNN	0.9240	0.9993	0.9293	0.7870
AE1.5M-ILF	0.9821	0.9134	0.8982	0.9090
AE1.5M-KNN	0.8835	0.9992	0.9191	0.7777

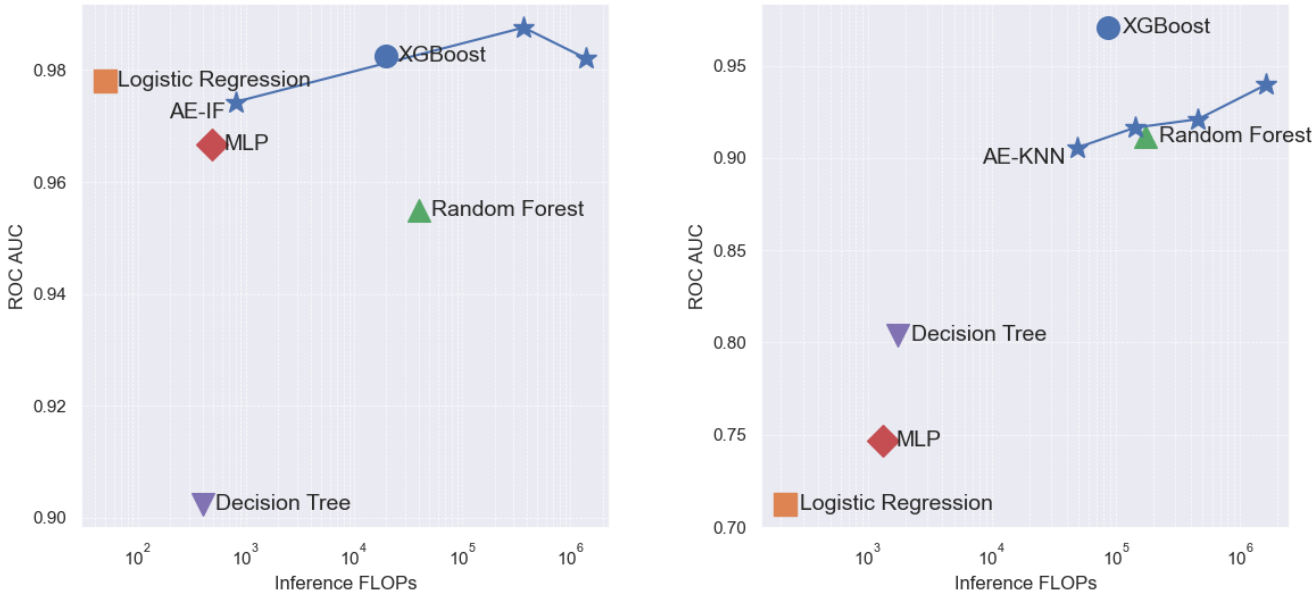
Table 4.2: Results for the auto-encoder representation learning experiments for the experiments as described in table 4.1, for the ULB300K dataset.

Code	ROC AUC	Accuracy	Precision	Recall
AE30K-ILF	0.8594	0.9504	0.3789	**0.8277**
AE30K-KNN	0.9056	0.9904	**0.9588**	0.8038
AE100K-ILF	0.8617	0.9613	0.5816	0.8227
AE100K-KNN	0.9164	0.9914	0.9502	0.8043
AE400K-ILF	0.8562	0.9546	0.5522	0.8217
AE400K-KNN	0.9210	0.9935	0.9535	0.8060
AE1.5M-ILF	0.8637	0.8876	0.2536	0.8273
AE1.5M-KNN	**0.9396**	**0.9934**	0.9350	0.7957

Table 4.3: Results for the auto encoder representation learning experiments for the experiments as described in table 4.1, for the PER40M dataset.

Figure 4.4 shows the inference FLOPs vs ROC AUC tradeoff as introduced in figure 3.1 including the models introduced in this section. Overall, none of the configurations can be considered a breakthrough in efficiency. Nonetheless, for the ULB300K dataset, a configuration of AE-IF obtains better performance than the

highest benchmark (XGBoost). For the PER40M dataset, none of the models scores better than the highest benchmark (XGBoost).



(a) ULB300K

(b) PER40M

Figure 4.4: Inference FLOPs vs ROC AUC comparison between the self-supervised autoencoder models and the baselines as established in table B.1. For the ULB300K dataset, the AE-IF model family is shown (AE30K-ILF and AE100K-ILF omitted). On this spectrum, AE400K-ILF can be considered an efficient method, as it obtains higher inference performance at higher inference complexity compared to the baselines. For the PER40M dataset, the AE-KNN model family is shown. The smallest configuration (the AE30K-KNN) can be considered efficient, although the XGBoost model performs significantly better at slightly higher inference complexity.

Chapter 5

Supervised representation learning

In chapter 4 we made use of the auto-encoder training paradigm to train a feature encoder $G(\cdot)$ that maps the data features to an informative feature space without the utilization of the true labels of the data. The effectiveness of this method is remarkable, as one would expect the target labels to be extremely helpful in learning efficient feature representations. In this chapter, we explore a supervised learning method to train the encoder $G(\cdot)$. We hypothesize that, guided by the target labels, an encoder can be trained that results in an embedding space with a better structure for downstream classification tasks. To that end, this chapter focuses on the questions:

1. Do feature encoders obtained through supervised learning result in more useful features for downstream fraud classification then self-supervised learning?
2. How does the performance of the encoder depend on the network size?

This chapter is structured as follows. Section 5.1 introduces the training framework used in this chapter. 5.2 discusses the choice of encoder architecture. 5.3 introduces the data sampling strategy. 5.4 defines the setup of the experiments that will be evaluated and section 5.5 covers the results for all these experiments.

5.1 Deep metric learning for fraud detection

In chapter 4, we trained a feature encoder $G(x_i)$ by minimizing the self-reconstruction loss in an auto-encoder setting. As a result, the embedding space is structured implicitly, according to whichever structure is optimal for reconstructing x_i from the embedding. The concept of metric learning, as described in section 2.4.2 involves explicit structuring of the latent space. This means that the structure of the embedding space is a direct result of the loss function used in the training procedure. The loss function guides the embedding space to some desired structure. Inevitably, the quality of the resulting structure is a result of the specification of the loss function as well as the convergence of the learning process. We hypothesize that an

encoder $G(x_i)$ trained through a supervised metric learning procedure, under the right choice of the loss function, results in informative features for downstream classification tasks.

In the case of fraud detection, a binary classification problem, an obvious choice for the loss function is a function that quantifies the distance between positive and negative samples in the latent space. In the experiments conducted in this work, we use the triplet-loss as defined in 2.3. A simplified description of the training procedure is described in algorithm 2. The Adam optimizer Kingma and Ba (2017) is used for weight updates.

Algorithm 2 The training procedure for training encoder G_{θ_G}, parametrized by θ_G in a supervised metric learning setting.

Initialize θ_F, θ_G
for each epoch **do**
  for batch $\mathcal{B}^-$ in dataset X^- **do**
    1. sample $\mathcal{B}^+$ from X^+ and construct $\mathcal{B}$
    2. Evaluate $z_i = G(x_i)$ for each $x_i \in \mathcal{B}$
    3. Construct batch $\mathcal{T}$ containing triplets (x_a, x_p, x_n) through some triplet mining strategy
    4. compute batch loss
    $\mathcal{L}(\mathcal{B}) = |\mathcal{T}|^{-1} \sum_{(x_a,x_p,x_n)\in\mathcal{T}} \left[d(G(x_a), G(x_p)) - d(G(x_a), G(x_n)) + \alpha \right]^+$
    5. execute gradient step $\theta_G \leftarrow \theta_G - \eta \frac{\partial \mathcal{L}}{\partial \theta_G}$
  end for
end for

5.2 Encoder architecture

Conceptually, $G(\cdot)$ is flexible and can be embodied by any differentiable transformation. In the setting of tabular data, there are a few obvious choices for this encoder architecture. Firstly, a vanilla Multi-layer Perceptron is a sensible choice. In this case, the encoder would effectively have the same specification as the encoder used in the auto-encoder method in chapter 4. Another idea is the use of self-attention in the encoder. In particular the transformer (Vaswani et al., 2023) architecture has found an increased use in encoder architectures for tabular data in recent years, some examples being Sreekar et al. (2023) and Huang et al. (2020). We include this architecture in our experiments to analyze its performance compared to the regular MLP encoder. In general, the transformer encoder works as follows. The input X (batch of instances x_i) is first projected to dimension d_1 through a linear mapping $X' = XW_{in} + b_{in}$. Subsequently, the batch flows through a series of encoder layers.

In each layer, the *query* Q, *keys* K and *values* V are computed by

$$Q = X'W^Q, \quad K = X'W^K, \quad V = X'W^V$$

next, scaled-dot-product attention is applied:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{\text{embedding\_dim}}}\right)V$$

The result is a weighted sum of the values V, where the weights are given by the attention scores. In practice, we use multi-head attention. This means that the above self-attention mechanism is applied in parallel for different query, key, and value weight matrices. The output is concatenated to one single vector. The self-attention module is followed by a feed-forward network with ReLU nonlinearity. The final output of the encoder layer is a normalized sum of the input to the block and the output of the block (residual connection). For a full specification of the architecture, we refer to Vaswani et al. (2023). The encoder block with *num_heads* is repeated *num_layers* times before the output is linearly mapped to the embedding dimension r.

5.3 Triplet selection

The triplet selection strategy determines which triplets are used to evaluate the loss for each batch. An intelligent sampling strategy is needed since it is impossible to evaluate all triplets and random sampling leaves the usefulness of the selected triplets to chance. Some triplet selection strategies are described in section 2.4.2. We exclusively use the Batch Hard triplet mining strategy in this section. This means that for each batch in the data, the embeddings for the batch are obtained by passing the data through the encoder. Next, the pairwise distances are calculated between the instances in the batch, after which each instance in the sample is matched with its furthest positive and closest negative pair. From a sampling perspective, the quality of the select triples increases as the batch size $|\mathcal{B}|$ increases (because there are more instances to construct the triples with). However, triplet selection is a fundamentally expensive $\mathcal{O}(|\mathcal{B}|^2)$ calculation, due to the distance matrix calculation in the r-dimensional embedding space. Thus, in practice, the choice of batch size is a trade-off between more training steps (epochs) or better triplet selection. After initial manual tuning, we set the batch size to $|\mathcal{B}| = 4.096$.

5.4 Experimental setup

5.4.1 Down-stream classifier models

The experimental setup is largely equivalent to the one described in section 4.2 and 3.3. Furthermore, we select the same classifier model families for the supervised

learning experiments as for the self-supervised learning experiments, as described in 4.2.

5.4.2 Experiment configurations

Table 5.1 shows the experiment configurations that will be evaluated. Similarly to earlier experiments, the smallest configuration of the MLP encoder is only evaluated for the ULB300K dataset. The three transformer-based encoder configurations are evaluated for the PER40M dataset. In line with previous experiments, we are interested in scaling properties of the model. However, for this training regime, keeping track of a validation loss is less straightforward than for the autoencoder method, since the model is trained on sampled data and it is computationally infeasible to evaluate the loss for all triplets. We revert to tracking the training loss over each sampled batch and use this to diagnose the quality of convergence of the training method.

5.4.3 Training details

The Adam Kingma and Ba (2017) optimizer is used with learning rate 4e−5 for the MLP encoder and 3e−4 for the transformer-based encoder.

Table 5.1: Experiment configurations evaluated for the auto-encoder setting. The code is a unique identifier for the model configuration. The number of parameters # Parameters is the total number of weights in the autoencoder setup (both encoder and decoder). The Encoder structure column indicates the sizes of the hidden layers used in the encoder (and in the decoder but reversed.

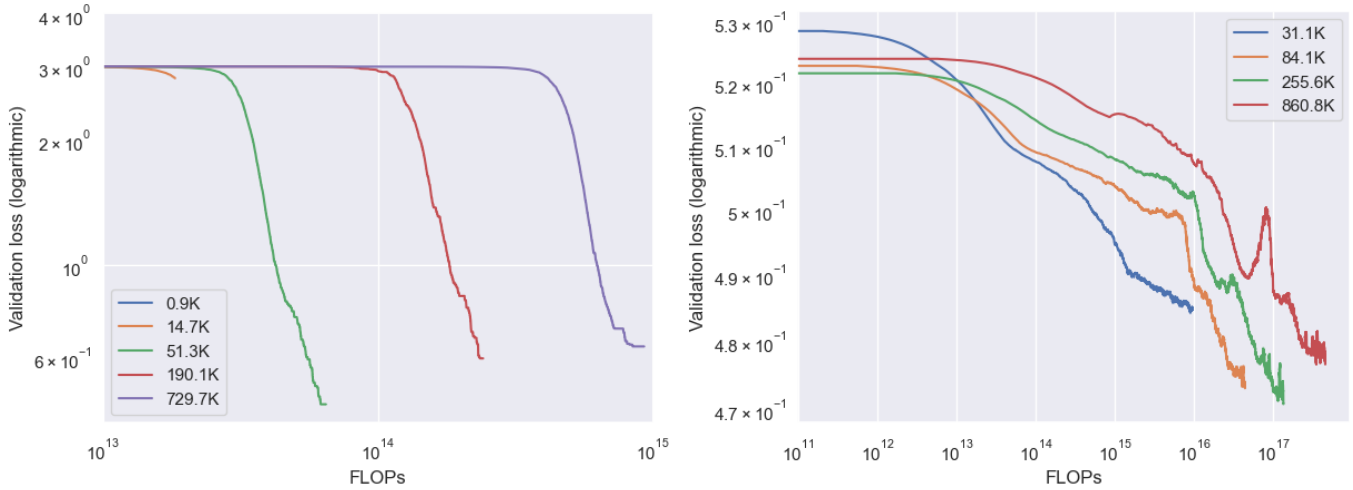
Code	Encoder structure	#Parameters	Classifier	ULB300K	PER40M
MLP1K-ILF	[16]	1.5K	IF	✓	
MLP1K-KNN	[16]	1.5K	KNN	✓	
MLP30K-ILF	[128, 64, 32]	29.2K	IF	✓	✓
MLP30K0-KNN	[128, 64, 32]	29.2K	KNN	✓	✓
MLP100K-ILF	[256, 128, 64, 32]	102.4K	IF	✓	✓
MLP100K-KNN	[256, 128, 64, 32]	102.4K	KNN	✓	✓
MLP400K-ILF	[512, 256, 128, 64, 32]	379.9K	IF	✓	✓
MLP400K-KNN	[512, 256, 128, 64, 32]	379.9K	KNN	✓	✓
MLP1.5M-ILF	[1024, 512, 256, 128, 64, 32]	1.5M	IF	✓	✓
MLP1.5M-KNN	[1024, 512, 256, 128, 64, 32]	1.5M	KNN	✓	✓

Code	num_heads	num_layers	#Parameters	Classifier	ULB300K	PER40M
TF75K-ILF	1	1	75K	IF		✓
TF75K-KNN	1	1	75K	KNN		✓
TF150K-ILF	2	2	150K	IF		✓
TF150K-KNN	2	2	150K	KNN		✓
TF150K-ILF	4	4	300K	IF		✓
TF150K-KNN	4	4	300K	KNN		✓

5.5 Results

5.5.1 Scaling properties

Figure 5.1 shows the training loss trajectory for the experiments for both datasets. These plots reveal that the larger model sizes do not necessarily achieve a lower training loss. This is a remarkable artifact that may indicate that these larger model sizes require an alternated training procedure in order to obtain better loss convergence. Figure 5.2 shows the test ROC AUC metric vs. the inference FLOPs of each model configuration. This figure shows a clear scaling benefit for the ULB300K dataset, whereas the PER40M performance does not scale beyond the 134M FLOPs model. All in all, this is a sign that there is a complex interaction between the dataset, model scale, the structure of the resulting embedding space, and the downstream classification performance. Depending on the dataset it does not necessarily hold that a lower representation loss leads to better classification performance. Figure 5.4a shows the test ROC AUC for the Transformer based encoder architecture, using the



(a) ULB300K (b) PER40M

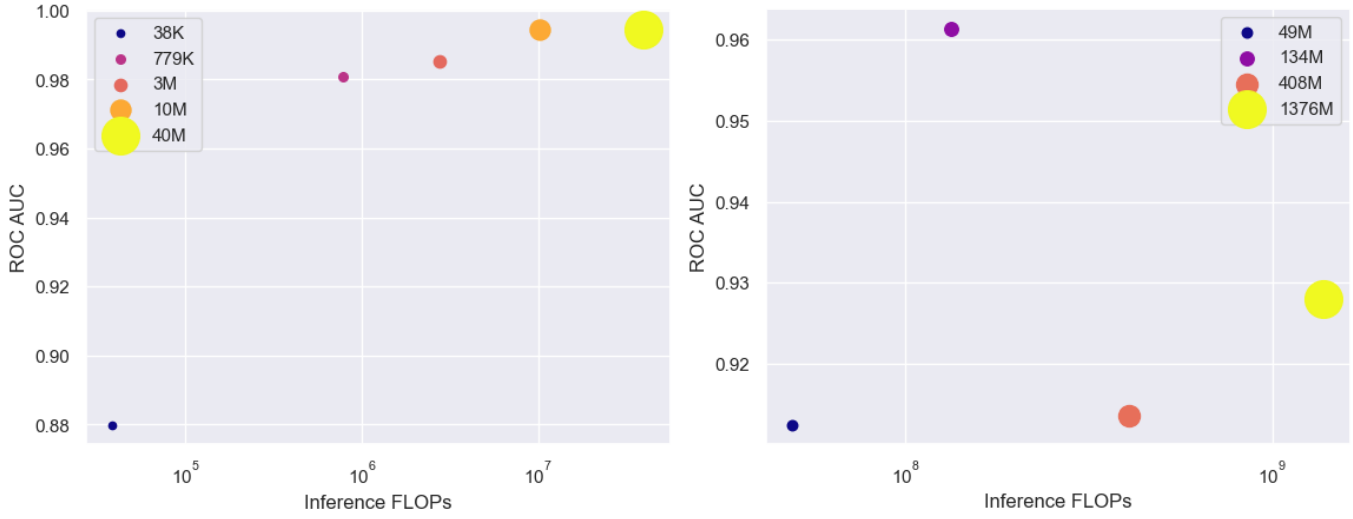
Figure 5.1: Training loss trajectory of the triplet loss method for both datasets and different model sizes. In both cases, it can be observed that a larger model does not necessarily always lead to lower loss values.

Isolation Forest downstream classifier. A clear scaling benefit is observed, as scaling the model size leads to significantly higher downstream classification performance.

5.5.2 Embedding visualizations

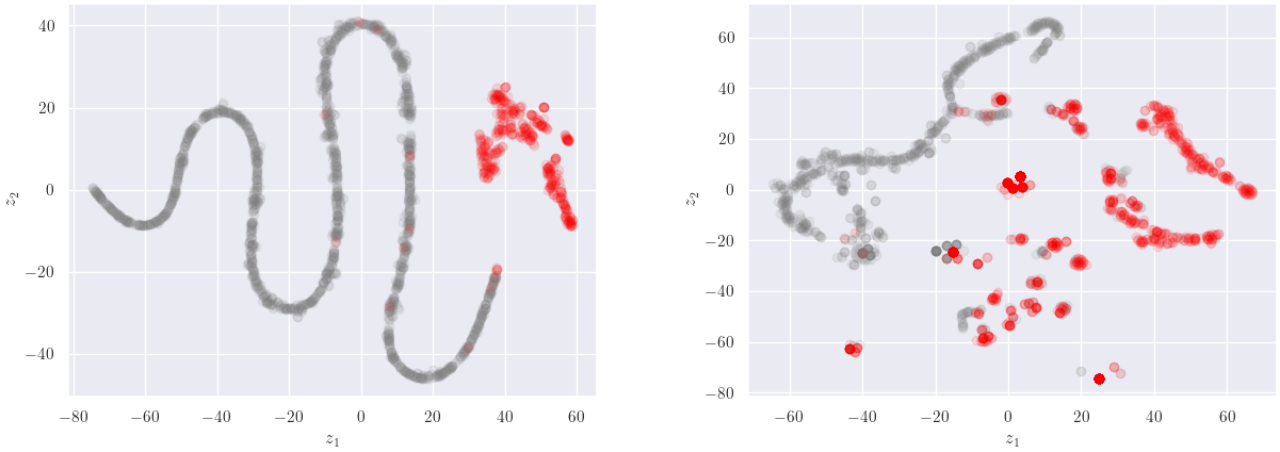
Figure 5.3b shows a visualization of the embedding spaces of the encoder models taken from the experiment with the best downstream classification performance. For the ULB300K dataset, a clear difference is visible in the way the positive and negative embeddings are positioned in the embedding space, compared to the self-supervised experiments. The cluster containing the positive instances is now well-separated from the negative cluster. As for the PER40M dataset, some visual differences are notable but it is hard to further reason about the quality of the embedding space.

The embedding visualization for the Transformer-based encoder is visible in 5.4b. This plot shows significant differences in the way the latent space is structured compared to the MLP encoders used in chapter 4 and the experiments in this chapter. The first notable difference is the variation between data points of the same class. Whereas the points in the MLP embedding spaces seem to lie on a single manifold, for the transformer-based embedding space they seem to form a point cloud roughly following some manifold. Another difference is that there is a clear presence of one cluster per class, instead of multiple smaller clusters as visible in the MLP embedding spaces.



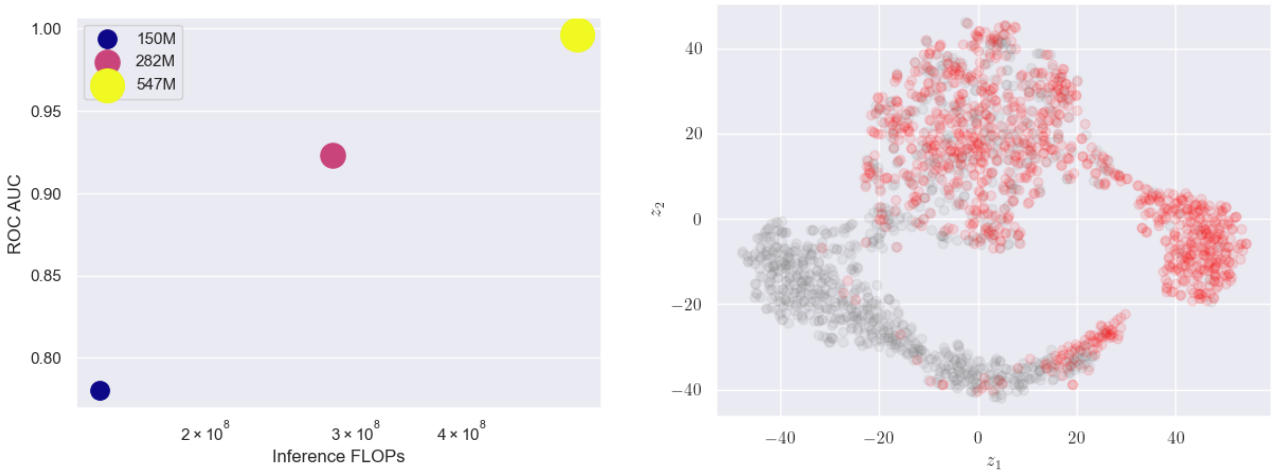
(a) ULB300K (b) PER40M

Figure 5.2: Test ROC AUC vs. Inference FLOPs for the triplet mining training experiments using the KNN classifier. The ULB300K dataset shows that larger model sizes directly improve downstream classification performance. For the PER40M dataset, this behavior is not present, as the best downstream classification test ROC AUC is obtained at the second-smallest model size.



(a) ULB300K (b) PER40M

Figure 5.3: Two-dimensional t-SNE representations of the embedding spaces obtained from the best-performing encoder in the triplet mining experiments, for 1000 positive (red) and 1000 negative (grey) sampled instances for both datasets. The ULB300K dataset shows an embedding space where positive and negative instances are well separated, with the two clusters having no significant overlap. As for the PER40M dataset, the embedding space shows some degree of separation between the two classes but has areas with a large overlap.



(a) Classifier performance vs. encoder size

(b) Embedding space visualization

Figure 5.4: Transformer complexity vs. downstream classification performance using the Isolation Forest classifier (left). Two-dimensional t-SNE representations of the embedding spaces for 1000 positive (red) and 1000 negative (grey) sampled instances for the PER40M using the transformer-based encoder (right).

5.5.3 Classification performance

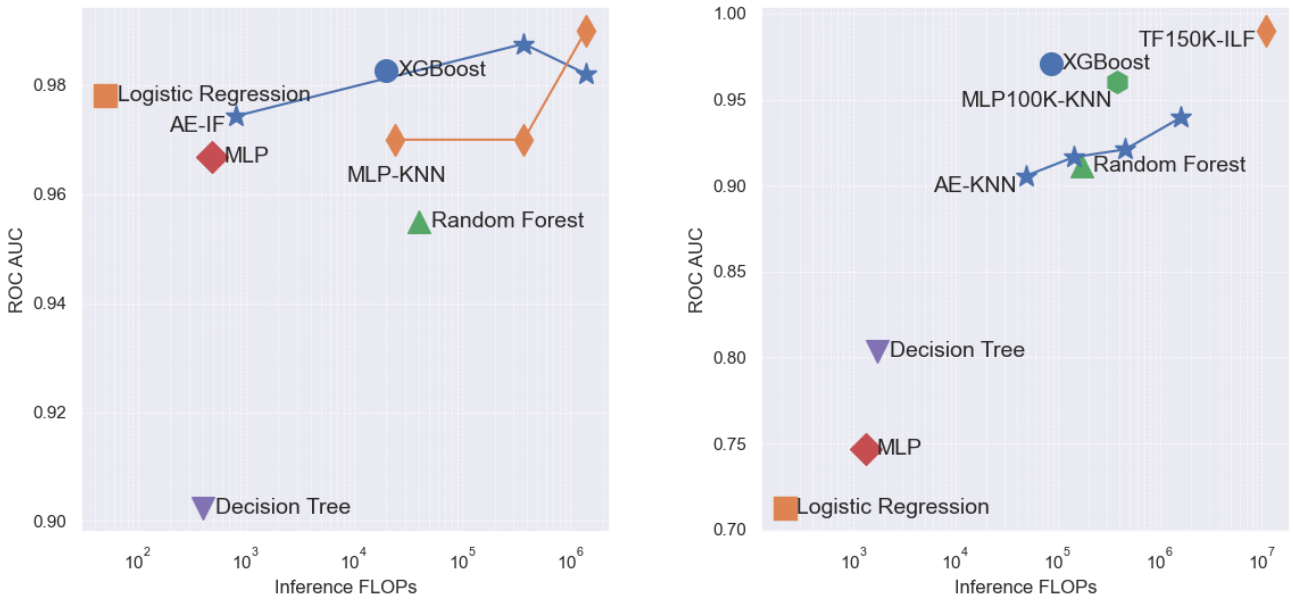
Table 5.2 shows the classification performance of the model configurations on the ULB300K dataset. The MLP1.5M-KNN model outperforms both the self-supervised methods and the best baseline. Table 5.3 shows the classification performance for the experiment configurations on the PER40M dataset. The transformer-based encoder combined with an Isolation Forest significantly outperforms the MLP-based encoder with KNN classifier, based on test ROC AUC. It also outperforms the self-supervised methods and the XGBoost baseline. A visualization of the complexity-performance trade-off compared to the baselines is shown in figure 5.5.

Table 5.2: Results for the supervised representation learning experiments for the experiments as described in table 5.1, for the ULB300K dataset.

Code	ROC AUC	Accuracy	Precision	Recall
MLP1K-ILF	0.8799	0.8857	0.8977	0.5850
MLP1K-KNN	0.9087	0.9994	0.8933	0.6760
MLP30K-ILF	0.9808	0.9968	0.8854	0.9696
MLP30K-KNN	0.9798	0.9998	0.9393	0.9393
MLP100K-ILF	0.9851	0.9965	0.9187	0.9393
MLP100K-KNN	0.9697	**0.9999**	**0.9484**	0.9292
MLP400K-ILF	0.9945	0.9987	0.9293	0.9595
MLP400K-KNN	0.9848	**0.9999**	0.9320	**0.9696**
MLP1.5M-ILF	0.9946	0.9979	0.9110	0.9595
MLP1.5M-KNN	**0.9999**	0.9998	0.9100	0.9191

Table 5.3: Results for the supervised representation learning experiments for the experiments as described in table 5.1, for the PER40M dataset.

Code	ROC AUC	Accuracy	Precision	Recall
MLP30K-ILF	0.7839	0.9716	0.8198	0.8824
MLP30K-KNN	0.9125	0.9868	0.9171	0.8180
MLP100K-ILF	0.7265	0.9584	0.8959	0.8795
MLP100K-KNN	0.9614	0.9875	0.9397	0.8206
MLP400K-ILF	0.7139	0.9318	0.8918	0.8867
MLP400K-KNN	0.9136	0.9869	0.9349	0.8204
MLP1.5M-ILF	0.7412	0.9672	0.9197	0.8797
MLP1.5M-KNN	0.9280	0.9867	0.9263	0.8254
TF75K-ILF	0.7801	0.9179	0.7194	0.8998
TF75K-KNN	0.9428	0.9877	**0.9504**	0.8186
TF150K-ILF	0.9229	0.9139	0.7913	0.9079
TF150K-KNN	0.7613	0.9601	0.8524	0.7956
TF150K-ILF	**0.9858**	**0.9907**	0.8958	**0.9288**
TF150K-KNN	0.8640	0.9871	0.9378	0.7902



(a) ULB300K

(b) PER40M

Figure 5.5: Inference FLOPs vs ROC AUC comparison between the supervised method and the autoencoder methods and baselines as established in table B.1. For the ULB300K dataset, the MLP-KNN model family is shown. On this spectrum, MLP1.5M-KNN can be considered an efficient method, as it obtains higher inference performance at higher inference complexity compared to the baselines. For the PER40M dataset, the TF150K-ILF and MLP100K-KNN configuration is shown. TF150K-ILF is an efficient method as it obtains better ROC AUC than the XGBoost baseline, even though it's inference FLOPs is two orders of magnitude larger.

Chapter 6

Conclusion

This work has explored the potential of deep learning methods for fraud detection, a domain traditionally dominated by tree-based models, through two primary objectives. First, we established a benchmark comparing common fraud detection methods on a public (ULB300K) and proprietary (PER40M) dataset. Second, we evaluated self-supervised and supervised deep learning approaches to assess their viability and scalability in fraud detection tasks. Here, we summarize the key findings and their implications.

The benchmark results highlight the dominance of XGBoost as the most effective method across both datasets, achieving superior performance in terms of classification metrics and inference efficiency among the baselines. The analysis of dataset representativeness revealed that while the public dataset (ULB300K) provides a reasonable proxy for academic evaluations, its low dimensionality and scale limit its applicability for real-world fraud detection tasks. The proprietary dataset (PER40M) offers a greater number of features and a much greater number of fraudulent transactions. This contrast underscores the need for publicly available datasets that better emulate industrial challenges to drive impactful research.

The self-supervised experiments demonstrated that autoencoder-based feature encoders yield informative embeddings for downstream classification. On the ULB300K dataset, self-supervised models like AE400K-ILF achieved competitive performance, occasionally surpassing traditional baselines like XGBoost. However, on the PER40M dataset, these methods struggled to outperform baselines, indicating limitations in representational power. The embedding visualizations showed that the embedding space created by autoencoders provided some degree of separation between fraudulent and non-fraudulent transactions, as observed in the t-SNE visualizations. Nonetheless, the observed overlap in clusters suggested that these embeddings lacked the precision needed for robust downstream classification in the PER40M dataset.

The triplet loss-based supervised learning method demonstrated clear advantages

in both performance and embedding quality. On the ULB300K dataset, supervised models such as MLP1.5M-KNN outperformed the best self-supervised configurations and the XGBoost baseline. On the PER40M dataset, the transformer-based encoder paired with an Isolation Forest classifier achieved the highest ROC AUC score, outperforming traditional and self-supervised methods. However, this comes at a large cost, with the number of inference FLOPs being two orders of magnitude larger. All in all, these results highlight the potential of supervised learning, particularly when leveraging architectures like transformers, to capture complex relationships in high-dimensional and imbalanced datasets. The structured separation observed in t-SNE embeddings from supervised models further affirmed their ability to encode meaningful representations for fraud classification.

Both self-supervised and supervised models demonstrated complex scaling behaviors. While increasing model size often improved performance, diminishing returns were observed in larger configurations, particularly on the PER40M dataset. This suggests that model design and training strategies need further optimization to fully leverage the potential of large-scale models.

6.1 Limitations

This work faced several clear limitations, some of which include:

1. Limited availability of public data: The public dataset used in this work has proven to be insufficient to properly benchmark fraud detection methods. Some methods obtain near-perfect scores, suggesting highly informative features are available in the dataset. Yet, the feature names and meanings are undisclosed, leaving high classification accuracy unexplained.

2. Incomplete set of benchmark methods: This work uses a set of benchmark methods manually selected based on current literature. It cannot be ruled out that there exists some method that outperforms the selected benchmarks on the basis of inference complexity or classification performance.

3. Model selection bias: A large number of experiment configurations have been evaluated in this work. It can be expected that, as the number of experiments grows, favorable results could be obtained as a result of *chance*. However, we believe, especially for the PER40M dataset, the test dataset has sufficient size to prevent this bias.

6.2 Future Directions

The findings of this work underscore the growing potential of deep learning in fraud detection. While tree-based models remain strong contenders, particularly in the low-complexity domain, deep learning methods have the potential to be competitive in the high-complexity domain. The success of supervised representation learning methods, especially transformer-based encoders, suggests that further research into hybrid models and pretraining strategies could yield interesting results. Some ideas for future work include:

1. Training strategies: Incorporating techniques like auxiliary loss functions to stabilize training and improve generalization. This might lead to a solution to the poor loss convergence in the larger MLP encoders in the triplet learning setting.

2. Triplet sampling methods: Many triplet sampling strategies have been proposed in the literature. Some of these may lead to better latent spaces or more efficient training in the triplet loss training setting.

3. Dataset Augmentation: In the self-supervised setting, data augmentation methods may yield more efficient feature encoders. One idea would be to use a denoising auto-encoder framework.

4. Model explainability: This work put little emphasis on the explainability of model decisions. Research on explainable methods for ANN encoders in combination with the downstream classifier would be highly valuable.

In conclusion, this study has demonstrated the viability of deep learning methods as an emerging paradigm in fraud detection. While traditional methods continue to excel in certain contexts, the promising results of self-supervised and supervised learning approaches pave the way for further research in this domain.

Bibliography

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Bengio, Y., Courville, A., and Vincent, P. (2014). Representation learning: A review and new perspectives.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. 59(4-5):291–294.

Breiman, L. (1997). Arcing the edge. Technical report, Citeseer.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Calì, C. and Longobardi, M. (2015). Some mathematical properties of the roc curve and their applications. *Ricerche di Matematica*, 64:391–402.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 11 of *KDD '16*, page 785–794. ACM.

Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 539–546. IEEE.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. 2(4):303–314.

Dal Pozzolo, A., Caelen, O., Le Borgne, Y.-A., Waterschoot, S., and Bontempi, G. (2014). Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, 41(10):4915–4928.

de la Bourdonnaye, F. and Daniel, F. (2021). Evaluating categorical encoding methods on a real credit card fraud detection database.

de Souza, D. H. M. and Jr, C. J. B. (2021). Ensemble and mixed learning techniques for credit card fraud detection.

Domingos, P. (2000). A unified bias-variance decomposition for zero-one and squared loss. *AAAI/IAAI*, 2000:564–569.

Duan, Y., Zhang, G., Wang, S., Peng, X., Ziqi, W., Mao, J., Wu, H., Jiang, X., and Wang, K. (2024). Cat-gnn: Enhancing credit card fraud detection via causal temporal graph neural networks.

Ekström, J., Åkerrén Ögren, J., and Sjöblom, T. (2023). Exact probability distribution for the roc area under curve. *Cancers*, 15(6):1788.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

Fernando, K. R. M. and Tsokos, C. P. (2022). Dynamically weighted balanced loss: Class imbalanced learning and confidence calibration of deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2940–2951.

Fix, E. and Hodges, J. L. (1989). Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238–247.

Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612.

Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. (2023). Revisiting deep learning models for tabular data.

Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2021). Masked autoencoders are scalable vision learners.

Hermans, A., Beyer, L., and Leibe, B. (2017). In defense of the triplet loss for person re-identification.

Hinton, G. E. and Roweis, S. (2002). Stochastic neighbor embedding. *Advances in neural information processing systems*, 15.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Huang, X., Khetan, A., Cvitkovic, M., and Karnin, Z. (2020). Tabtransformer: Tabular data modeling using contextual embeddings.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA. Curran Associates Inc.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Kohavi, R., Wolpert, D. H., et al. (1996). Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–283. Citeseer.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422.

Mann, H. B. and Whitney, D. R. (1947). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50 – 60.

McClish, D. K. (1989). Analyzing a portion of the roc curve. *Medical decision making*, 9(3):190–195.

Melo, T. L., Bravo, J., Sampaio, M. O. P., Romano, P., Ferreira, H., Ascensão, J. T., and Bizarro, P. (2023). Adversarial training for tabular data with attack propagation.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. (2019). Deep double descent: Where bigger models and more data hurt.

Oshri, B. (2015). There and back again : Autoencoders for textual reconstruction.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572.

Pearson, K. (1904). Mathematical contributions to the theory of evolution. xii.—on a generalised theory of alternative inheritance, with special reference to mendel's laws. *Proceedings of the Royal Society of London*, 72(477-486):505–509.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Porwal, U. and Mukund, S. (2019). Credit card fraud detection in e-commerce: An outlier detection approach.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2019). Catboost: unbiased boosting with categorical features.

Qin, G. and Zhou, X.-H. (2006). Empirical likelihood inference for the area under the roc curve. *Biometrics*, 62(2):613–622.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1:81–106.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

S., V. M., Yuan, S., and Wu, X. (2023). Robust fraud detection via supervised contrastive learning.

Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pages 4–11.

Salzberg, S. L. (1994). C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 815–823. IEEE.

Sreekar, P. A., Verma, S., Madhavan, V., and Persad, A. (2023). Unveiling the power of self-attention for shipping cost prediction: The rate card transformer.

Talukder, M. A., Hossen, R., Uddin, M. A., Uddin, M. N., and Acharjee, U. K. (2024). Securing transactions: A hybrid dependable ensemble machine learning model using iht-lr and grid search.

Thimonier, H., Popineau, F., Rimmel, A., Doan, B.-L., and Daniel, F. (2023). Comparative evaluation of anomaly detection methods for fraud detection in online credit card payments.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999.

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103.

Worobec, K. (2021). Fraud -the facts 2021 the definitive overview of payment industry fraud.

Wu, C.-Y., Manmatha, R., Smola, A. J., and Krähenbühl, P. (2018). Sampling matters in deep embedding learning.

Xing, E., Jordan, M., Russell, S. J., and Ng, A. (2002). Distance metric learning with application to clustering with side-information. *Advances in neural information processing systems*, 15.

Xuan, H., Stylianou, A., and Pless, R. (2020). Improved embeddings with easy positive triplet mining.

Ye, H.-J., Liu, S.-Y., Cai, H.-R., Zhou, Q.-L., and Zhan, D.-C. (2024). A closer look at deep learning on tabular data.

Yu, C., Xu, Y., Cao, J., Zhang, Y., Jin, Y., and Zhu, M. (2024). Credit card fraud detection using advanced transformer model.

Zhu, M., Zhang, Y., Gong, Y., Xu, C., and Xiang, Y. (2024). Enhancing credit card fraud detection a neural network and smote integrated approach.

Appendix A

Confusion-matrix based classification metrics

Table A.1: A selection of confusion-matrix based metrics

Metric	**Definition**	**Description**
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	Proportion of correctly classified instances.
Precision	$\frac{TP}{TP+FP}$	Proportion of predicted positives that are actually positive.
Recall (Sensitivity)	$\frac{TP}{TP+FN}$	Proportion of actual positives correctly identified.
Specificity	$\frac{TN}{TN+FP}$	Proportion of actual negatives correctly identified.
F1-Score	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$	Harmonic mean of precision and recall.
False Positive Rate	$\frac{FP}{FP+TN}$	Proportion of actual negatives incorrectly classified as positive.
False Negative Rate	$\frac{FN}{FN+TP}$	Proportion of actual positives incorrectly classified as negative.
Positive Predictive Value	$\frac{TP}{TP+FP}$	Probability that a positive prediction is correct.
Negative Predictive Value	$\frac{TN}{TN+FN}$	Probability that a negative prediction is correct.

Appendix B

Benchmark Study

B.1 Hyperparameter results

Table B.1: Benchmark Models and Hyperparameter Tuning Details for the benchmark study. Indicated parameters are the hyperparameters tuned during the cross-validation procedure.

Model	**ULB300k**	**PER40M**
XGBoost	max_depth: 16	max_depth: 6
	n_estimators: 50	n_estimators: 50
	alpha: 0	alpha: 0.1
	lambda: 0	lambda: 0
Logistic Regression	C: 1	C: 1
	L1_ratio: 0	L1_ratio: 0
Random Forest	max_depth: 16	max_depth: 8
	n_estimators: 100	n_estimators: 50
MLP	hidden_layer_sizes: [20, 10]	hidden_layer_sizes: [20, 10]
	activation: ReLU	activation: ReLU
Decision Tree	max_depth: 20	max_depth: 20

B.2 Additional results for MLP classifiers

Table B.2: Additional results for regular MLP classification models on the PER40M dataset. These are experiments where a regular MLP is used to classify the class label. The binary cross entropy is used as the loss function.

MLP structure	ROC AUC	Accuracy	Precision	Recall
[64,32,16,8]	0.8888	0.9911	0.7456	0.2660
[128,64,32,16,8]	0.8965	0.9914	0.7435	0.2999
[256,128,64,32,16,8]	0.9118	0.9916	0.7517	0.3205
[512,256,128,64,32,16,8]	0.9166	**0.9916**	0.7579	0.3201
[1024,512,256,128,64,32,16,8]	**0.9171**	**0.9916**	**0.7648**	**0.3205**

Table B.3: Additional results for regular MLP classification models on the ULB300K dataset.

MLP structure	ROC AUC	Accuracy	Precision	Recall
[64,32,16,8]	0.8936	0.9984	0.7910	0.5217
[128,64,32,16,8]	0.9097	0.9984	0.8104	0.6512
[256,128,64,32,16,8]	0.9748	0.9992	0.870	0.5838
[512,256,128,64,32,16,8]	0.9779	**0.9994**	**0.8571**	**0.7651**
[1024,512,256,128,64,32,16,8]	**0.983**	0.9991	0.8431	0.577

Appendix C

FLOPs estimation of benchmark models

This appendix provides an overview of how the floating point operations (FLOPs) were estimated for each machine learning model during inference. FLOPs are an approximate measure of computational cost, and they are calculated based on the model's architecture, number of parameters, and the input data size. The following describes the FLOP calculations for each model.

XGBoost

For XGBoost, the FLOPs are primarily determined by the number of trees, the maximum depth of each tree, and the number of features in the input. Each tree performs a series of comparisons as it traverses from the root to a leaf node.

- **Assumptions:**
  - Maximum depth $d = 16$, resulting in approximately $2^d = 65,536$ nodes per tree.
  - Number of trees = 50.
  - Each feature comparison in a tree node requires 1 FLOP.
- **Calculation:**
  - FLOPs per tree $\approx d \times$ number of features.
  - Total FLOPs $\approx d \times$ number of features $\times$ number of trees.

For 108 features, the total FLOPs per instance is approximately $16 \times 108 \times 50 = 86,400$.

Logistic Regression

In logistic regression, inference involves a single matrix-vector multiplication between the input features and the weight vector, followed by an addition with the bias term.

- **Assumptions:**
  - Each feature-weight multiplication and addition (dot product) requires 2 FLOPs.
- **Calculation:**
  - FLOPs $\approx 2 \times$ number of features.

For 108 features, the total FLOPs per instance is $2 \times 108 = 216$.

Random Forest

Random Forest operates similarly to XGBoost in terms of FLOP calculations, but typically has more trees, and each tree can be shallower or deeper depending on the hyperparameters.

- **Assumptions:**
  - Maximum depth $d = 16$, resulting in approximately 2^d nodes per tree.
  - Number of trees = 100.
  - Each feature comparison in a tree node requires 1 FLOP.
- **Calculation:**
  - FLOPs per tree $\approx d \times$ number of features.
  - Total FLOPs $\approx d \times$ number of features $\times$ number of trees.

For 108 features, the total FLOPs per instance is approximately $16 \times 108 \times 100 = 172,800$.

Multi-Layer Perceptron (MLP)

The MLP used in this experiment has two hidden layers with ReLU activation. The FLOPs are based on the number of neurons in each layer and the number of input features.

- **Assumptions:**
  - Layer 1 has 10 neurons, and Layer 2 has 20 neurons.

  - Each neuron computation involves a matrix-vector multiplication followed by a ReLU operation.

- **Calculation:**
  - FLOPs for Layer 1 $\approx$ number of features $\times$ 10.
  - FLOPs for ReLU activations in Layer 1 $\approx$ 10.
  - FLOPs for Layer 2 $\approx 10 \times 20$.
  - FLOPs for ReLU activations in Layer 2 $\approx$ 20.
  - Total FLOPs $\approx$ FLOPs for Layer 1 + FLOPs for Layer 2 + ReLU FLOPs.

For 108 features, the total FLOPs per instance is approximately $108 \times 10 + 10 + 10 \times 20 + 20 = 1,310$.

Decision Tree

The Decision Tree model performs inference by traversing from the root to a leaf, making comparisons at each node. The FLOPs depend on the depth of the tree and the number of features.

- **Assumptions:**
  - Maximum depth $d = 20$, resulting in approximately 2^d nodes per tree.
  - Each feature comparison requires 1 FLOP.

- **Calculation:**
  - FLOPs $\approx d \times$ number of features.

For 108 features, the total FLOPs per instance is approximately $20 \times 108 = 2,160$.