Master in Informatics Engineering
Internship
Final Report

Multi-Domain and Unsupervised Machine Learning for eCommerce Fraud Detection

Tiago Sapata
sapata@student.dei.uc.pt

Advisors:
Bernardete Ribeiro
António Alegria
Miguel Almeida

July 1st, 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Master in Informatics Engineering
Internship
Final Report

Multi-Domain and Unsupervised Machine Learning for eCommerce Fraud Detection

Tiago Sapata
sapata@student.dei.uc.pt

Advisors:
Bernardete Ribeiro
António Alegria
Miguel Almeida

Jury:
João P. Vilela
Alexandre Miguel Pinto

July 1st, 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

A globalização e o progresso tecnológico impulsionaram o aumento exponencial do acesso à internet nos últimos anos. O comércio eletrónico (eCommerce) seguiu a tendência, constantemente acompanhado por pagamentos fraudulentos, levando a perdas consideráveis nas receitas destes comerciantes. A Feedzai é uma empresa especializada em Machine Learning e Big Data que oferece soluções de prevenção de fraude em pagamentos. Uma destas soluções é um serviço baseado na Cloud especificamente para comércio eletrónico. Neste trabalho, utilizando dados históricos de comerciantes já existentes (chamado o domínio Source), são implementados dois algoritmos de Domain Adaptation, com o objectivo de melhorar a performance na deteção de fraude em pequenos ou novos comerciantes que não possuem pagamentos suficientes para treinar devidamente um modelo de ML. Adicionalmente, e com o mesmo objectivo, é explorado um algoritmo de deteção de outliers não supervisionado, não sendo necessário assim o recurso a dados históricos (do domínio Source) ou a dados anotados no domínio Target. Após teste com múltiplos e distintos datasets, conclui-se que ambos os algoritmos de Domain Adaptation não melhoram a performance na deteção de fraude. Por outro lado, considerando que apenas utiliza dados não anotados no domínio Target, o algoritmo não supervisionado apresenta uma performance considerável e apresenta-se como uma solução promissora para novos comerciantes no caso de uso de eCommerce apresentado.

Abstract

Globalization and technological progress led to exponential growth of internet access in the last decade. eCommerce followed the trend, constantly accompanied by payment fraud, costing merchants a substantial percentage of their revenue. Feedzai is a company specializing in Machine Learning (ML) and Big Data, providing solutions for fraud prevention in payments. One of Feedzai's solutions is a Cloud-based SaaS for eCommerce merchants. In this work, using historical data from existing merchants (called Source domain), we implement two Domain Adaptation algorithms with the goal of improving fraud detection for small or new merchants (called Target domain), who typically have insufficient payment data to properly train a ML model. Furthermore, we explore an Unsupervised Learning outlier detection algorithm, which does not rely on historical or labeled data, with the same goal. Upon testing on multiple distinct datasets, obtained results demonstrate both Domain Adaptation algorithms fail to improve performance and actually decrease it. On the other hand, considering it only uses unlabeled data from the Target domain, the Unsupervised Learning method provides quite good performance and proves to be a promising solution for new merchants in the presented eCommerce use case.

Contents

List of Tables

List of Figures

List of Acronyms

AeMMF All merchants except Merchant with Most Fraudulent transactions

AeMMT All merchants except Merchant with Most Transactions

ARFF Attribute-Relation File Format

ATM Automated Teller Machine

AUC Area-Under-Curve

CEP Complex Event Processing

CNP Card Not Present

CORAL Correlation Alignment (algorithm)

CP Card Present

DA Domain Adaptation

eCommerce Electronic Commerce

EMV Europay, MasterCard, and Visa

FEDA Frustratingly Easy Domain Adaptation (algorithm)

FN False Negative

FP False Positive

FPR False Positive Rate

JSON JavaScript Object Notation

LOF Local Outlier Factor (algorithm)

LRD Local Reachability Density

ML Machine Learning

MMF Merchant with Most Fraudulent transactions

MMT Merchant with Most Transactions

pAUC Partial Area-Under-Curve

PCA Principal Component Analysis

PQL Pulse Query Language

ROC Receiver Operating Characteristic

SaaS Software as a Service

SQL Structured Query Language

TN True Negative

TP True Positive

Chapter 1

Introduction

1.1 Context

Globalization and technological progress led to the massification of internet access in the last decade. From 2005 to 2015, internet users tripled to almost half of the world population. Electronic Commerce (eCommerce) followed this trend, with worldwide sales expected to reach over $2 trillion in 2017. [1, 2]

One of the greatest challenges eCommerce faces is fraudulent payments. These commonly originate from stolen credit cards or copied magnetic stripes, which are then sold or used to buy easily resellable products.

Feedzai is a company that specializes in providing state-of-the-art Machine Learning tools and Big Data analysis for fraud prevention in payments. In 2014 alone, Feedzai processed 18 billion transactions with a volume worth of $760 billion. [3] Feedzai's technology helps companies from all spectrum of the payment ecosystem, helping large financial institutions such as banks and processors as well as the merchants doing eCommerce. Of these, the eCommerce merchants are one of the most affected by fraud, as they are generally liable for it. Additionally, event though eCommerce is only a small part of the whole commerce, it has the greatest share of fraud. [4]

After fraudulent payment occurs, when the legitimate card owner notices a transaction he didn't make, the bank is contacted and a dispute is filled. This triggers a complex and costly process which usually results in a chargeback in favor of the legitimate card owner, and the merchant being liable. [5] In the case of in-store physical payments using Europay, MasterCard, and Visa (EMV) authentication methods, fraud liability is shifted to the cardholder's bank.

One of the main Feedzai's products is a Cloud-based Software as a Service (SaaS) offering for Fraud Prevention in eCommerce use cases - Feedzai Fraud Prevention for eCommerce - aiming to detect fraudulent payments before they occur, saving time and costs mainly for merchants, but also for all other entities involved in the process. This solution is based on machine learning models, allowing merchants to obtain a risk score for customer payments, representing the likelihood of a payment being fraudulent.

Clients of this product include new and small merchants, typically with a reduced or non-existent fraud prevention team. This means they don't have a substantial amount of labeled historical data from which to build good machine learning models right from the start.

1.2 Motivation and Scope

The main motivation for this work is the eCommerce merchant use case, as these are the most vulnerable targets of fraud. Training a robust machine learning model capable of scoring and

classifying payments for a merchant requires a considerable amount of payments data from that particular merchant. This leads to two main problems: first, merchants need to provide labeled data, which may not be readily available or can have prohibitive labeling costs; second, available data may not be enough to properly train a model. One of the reasons for not having enough data is merchants starting a new business. These are at even higher risk as they are more exposed to new fraud patterns.

Since Feedzai Fraud Prevention serves many clients, historical data from existing merchants could be used together, benefiting new merchants by providing an out-of-the-box global model. Additionally, as new payment data becomes available, existing models should be re-trained and gradually shift from a global model (trained with data from many merchants) to a merchant-specific model, with the goal of continually improving model's performance. Even when sufficient data for particular merchant is achieved, using data from many other can be beneficial, by searching for new fraud patterns appearing only in some merchants.

The main setback for a global model comes from the difference between the domains each merchant belongs to. In particular, merchants can operate in distinct areas (for example, an online e-book store versus an online shoes store) or even merchants operating in the same business can have unique characteristics, which might result in different payment data attributes and data distributions. This fact can cause a global model trained with data from several merchants perform rather poorly and ultimately useless. Thus, this work will not focus on particular machine learning algorithms but, instead, will focus on methods for transferring knowledge from one domain (existing merchants) to a different domain (new merchants) with the goal of improving fraud detection by using existing labeled transaction data. This concept of applying knowledge from a Source domain to improve performance on a Target domain is known as Transfer Learning. Furthermore, an Unsupervised Learning algorithm will also be explored, applying it directly to new merchant data, since it can provide a solution to the current eCommerce use case as well.

Being able to harness the full potential of heterogeneous data sources from multiple merchants, Feedzai could take advantage of network effects to improve fraud detection performance for all its clients.

1.3 Goals

This work aims to develop a method capable of dealing with data from merchants of different domains for model training, improving fraud detection for current merchants as well as providing an out-of-the-box global solution for new clients. The developed method should also be implemented in Feedzai tools.

In particular, the following goals are set:

(a) A well defined method that can learn from existing data from different merchants, allowing them to mutually benefit each other, as well as applying this knowledge for new merchants with no available data.

(b) A well defined method to continually improve machine learning models for each particular merchant. As new data from that merchant becomes available, the global model trained with data from many merchants should periodically be re-trained, with higher weights for the data of that specific merchant.

(c) The developed method should also support heterogeneous merchants with distinct source fields.

(d) Ideally, the methods mentioned should be developed on a layer above machine learning algorithms, that is, they should be independent from the machine learning models used. However, methods that depend on a particular machine learning algorithm can also be explored.

The developed methods should be tested and validated with real data from the eCommerce use case. Finally, a prototype of the method that exhibits the best results for the proposed goals should be implemented and integrated with Feedzai's Machine Learning tools.

1.4 Structure

This document is structured in the following chapters:

In Chapter 2 the Background Knowledge of the developed work is presented, with Transfer Learning concepts being introduced and the most common terminology defined. Following, two Domain Adaptation (DA) algorithms are introduced. Ending the Chapter, we present an Unsupervised Learning outlier detection algorithm that can also be applied for fraud detection in the current eCommerce use case.

In Chapter 3, we introduce practical methodologies related to this work, namely, two Machine Learning algorithms: Decision Trees and Random Forests. Closing the chapter, performance metrics for model evaluation and comparison are defined.

Chapter 4 contains the process used in the developed work and the obtained results. It starts with a detailed description of the datasets and tools used as well as the complete simulation process. Following, we present the preliminary experiments and results obtained in the First Semester. The last section of the chapter presents all final experiments completed during the Second Semester and discusses the achieved results.

Closing the document, Chapter 5 discusses the main conclusions drawn from this thesis and suggest several ideas for future work.

Chapter 2

Background Knowledge

Traditional machine learning methods assume training and testing data originate from the same domains, thus containing the same feature space and probability distributions. While this notion is true for many applications, for many other problems data can belong to different domains.

In this scenario, the knowledge acquired from one domain, known as Source domain, could help the learning task on the other domain, known as Target domain, as long as they have some similarity.

The same happens in the context of this work: by looking at historical payment data from existing merchants (the Source domain), we want to improve the learning task of fraud detection for new merchants (the Target domain), where both domains can differ in feature space, data distribution, or both.

The process of applying knowledge from a Source domain to improve the learning process on a Target domain is known as Transfer Learning. In the next section we will discuss several Transfer Learning techniques as well as traditional Machine Learning methodologies.

2.1 Transfer Learning

Transfer Learning is a Machine Learning field that aims to improve learning tasks for a particular domain (Target domain) by reusing knowledge acquired from a different domain (Source domain). This approach of applying knowledge transfer to machine learning is inspired by human learning - we regularly use previous knowledge to improve learning processes. For example, the more similar a new task is to previous learning experiences, the faster we learn it. [6]

Domain definition

A domain consists of a feature space $\mathcal{X}$ and a probability distribution $P(X)$ where $X = \{x_1, ..., x_n\} \in \mathcal{X}$, x_i being the vector of all samples for the ith feature and X a particular instance. A domain $\mathcal{D}$ can be defined as $\mathcal{D} = \{\mathcal{X}, P(X)\}$, consequently, two domains are considered distinct if they differ in feature space $\mathcal{X}$ or probability distributions $P(X)$. [7]

Task definition

Given feature space $\mathcal{X}$, a probability distribution $P(X)$ of a certain domain $\mathcal{D}$ and a label space $\mathcal{Y}$, the goal of a task is to find a predictive function $f()$ that can predict the label of a future sample. This function is obtained from observing current instances and the corresponding class label, that is, the pairs $(X, \mathcal{Y})$. In summary, task $\mathcal{T}$ aims to find $f()$, which goal is to determine the label $y = f(x) = P(y|x)$ of a new instance x. Using formal notation, a task can be defined as $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. [7]

Given the definitions above, Transfer Learning can be applied in two scenarios: first, when Source domain $\mathcal{D}_S$ and Target domain $\mathcal{D}_T$ are different, that is, $\mathcal{X}_S \neq \mathcal{X}_T$ or $P_S(X) \neq P_T(X)$; second, when tasks $\mathcal{T}_S$ and $\mathcal{T}_T$ are different, that is, when $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$. [7]

When both domains and both tasks are the same, then we have a traditional machine learning problem, and Transfer Learning no longer applies.

2.1.1 Transfer Learning settings

In this section we will describe each of three settings of Transfer Learning based on the different scenarios for Source and Target domains and tasks. A summary of the three settings is presented in Table 2.1.

Inductive Transfer Learning

In this setting, Source and Target tasks are always distinct, that is, $\mathcal{T}_S \neq \mathcal{T}_T$ regardless of Source and Target domains similarity. This setting requires some labeled data on the Target domain in order to induce a predictive function $f_T()$ to be applied in the Target domain, while the Source domain can have labeled data or not.

When the Source domain has labeled data available, we have a scenario related to Multi-task Learning. On the other hand, when the Source domain has no labeled data, the setting is similar to Self-Taught Learning. In Self-Taught Learning it is assumed that having more data will lead to better performance, thus the main idea is to acquire large amounts of unlabeled data instances from similar domains, which is significantly easier than getting labeled data. Using the unlabeled examples, the method generates new features and represents the labeled data in the new feature space, then applying classification methods as usual. [8]

Formally, the goal of Inductive Transfer Learning is to improve the predictive function $f_T()$ in the Target domain, using the knowledge acquired from the Source domain $\mathcal{D}_S$ and the Source task $\mathcal{T}_S$, where $\mathcal{T}_S \neq \mathcal{T}_T$. [7]

Transductive Transfer Learning

In Transductive Transfer Learning, Source domain and Target domain are different but the tasks are the same. Generally, the Source domain has plenty of labeled data while the Target domain has very little labeled data or none at all.

There are two distinct cases in this setting: first, the feature spaces of the Source and Target domains are different, that is, $\mathcal{X}_S \neq \mathcal{X}_T$. In other words, one of the domains can have more, fewer or different features. Second, the features spaces are the same but the probability distributions of the data are different. In this case we have a scenario related to Domain Adaptation, Sample Selection Bias or Co-variate Shift.

In Sample Selection Bias, the common assumption that training examples are sampled from the same distribution of testing data does not verify, since training data is usually sampled in a biased way and testing instances are from a more diverse population. Algorithms deal with this scenario by trying to correct the distribution bias between training and testing. [9, 10] Similar to this concept, Co-variate shift occurs when data training data, from the Source domain, and testing data, from the Target domain, differ in probability distribution. Algorithms try to solve Co-variate shift by approximating Source and Target data distributions, effectively minimizing domain differences. [11]

It should be mentioned that this setting, where Source and Target domain are different but the tasks are the same, includes our use case, where we have transactions from multiple origins (different domains) and the goal is to classify them as fraud or legitimate (same task).

Formally, the goal of Transductive Transfer Learning is to improve the learning of the predictive function $f_T()$ for the Target domain, using the knowledge acquired from the Source domain $\mathcal{D}_S$ and the Source task $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$. [7]

Unsupervised Transfer Learning

In Unsupervised Transfer Learning, Source and Target tasks are different but related, and no labeled data exists both in Source and Target domains. As such, this setting focuses on unsupervised learning tasks such as Clustering and dimensionality reduction.

Settings	Related Areas	Source Domain labels	Target Domain labels	Tasks
Inductive Transfer Learning	Multi-task Learning	Available	Available	Regression, Classification
	Self-Taught Learning	Unavailable	Available	Regression, Classification
Transductive Transfer Learning	Domain Adaptation, Selection Bias Co-variate Shift	Available	Unavailable	Regression, Classification
Unsupervised Transfer Learning		Unavailable	Unavailable	Clustering, Dimensionality Reduction

Table 2.1: Description of Transfer Learning Settings
Taken from [7].

An important aspect of Transfer Learning relates to how to transfer knowledge between domains or tasks. Namely, three important issues arise: what to transfer, how to transfer, when to transfer.

'What to transfer' relates to what knowledge should be transferred across domains or tasks. For example, two domains might share important knowledge that could improve the learning task. When 'What to transfer' is determined, a solution to transfer that knowledge must be implemented, and that refers to 'how to transfer'. Finally 'When to transfer' asks in which occasions we should really transfer knowledge. This is a pressing question since it is possible that knowledge transfer can have a negative impact in the performance of learning in the Target domain. This is referred to as negative transfer. [7]

Current approaches to Transfer Learning are based on 'What to transfer' and consist of the following four cases:

- **Instance-transfer** - the basic assumption of this approach is that Source and Target domains are somewhat related and some data from the Source domain can be used to improve the learning task on the Target domain by re-weighting instances.
- **Feature-representation-transfer** - this method tries to find a good feature representation of the Target domain with which knowledge transfer from the Source domain is encoded in, minimizing differences between both domains.
- **Parameter-transfer** - the main idea of this approach is to discover common parameters between Source and Target domains, and transferring knowledge from the Source domain by encoding it in these shared parameters.

- **Relational-knowledge-transfer** - this method assumes data from Source domain and Target domain share some kind of relationship. The main goal is to map these relations between Source and Target domains, allowing knowledge to be transferred accordingly.

In the next section, a concrete algorithm for Transfer Learning is presented. The algorithm implements Transductive Transfer Learning, specifically, Domain Adaptation.

2.2 Supervised Domain Adaptation by feature duplication

Hal Daumé III [12] proposed a straightforward algorithm for Domain Adaptation that is easily implemented by pre-processing datasets. The algorithm, which will be referred to as FEDA, acronym taken directly from the paper name (**F**rustratingly **E**asy **D**omain **A**daptation), consists of a simple transformation of feature augmentation: for each feature of a given dataset, three versions of that feature are created - a general version, a Source-specific version and a Target-specific version. After the transformation, instances from the Source domain will contain only the general and the Source feature versions while the instances from the Target domain will have the general and the Target versions. As such, if $\Phi^s(x)$ and $\Phi^t(x)$ are the the feature sets for the Source and Target domains, respectively, the transformation can be defined as:

$$\Phi^s(x) \rightarrow \langle \Phi^s(x), \Phi^s(x), 0 \rangle \text{ (for Source domain instances)}$$
$$\Phi^t(x) \rightarrow \langle \Phi^t(x), 0, \Phi^t(x) \rangle \text{ (for Target domain instances)}$$

As a concrete example, considering three features, let's suppose we have the two following instances:

$$\text{Source instance: } \langle a_s, b_s, c_s \rangle$$
$$\text{Target instance: } \langle a_t, b_t, c_t \rangle$$

After the transformation they would become:

$$\text{Source instance: } \langle a_s, b_s, c_s, a_s, b_s, c_s, 0, 0, 0 \rangle$$
$$\text{Target instance: } \langle a_t, b_t, c_t, 0, 0, 0, a_t, b_t, c_t \rangle$$

Using formal notation, the algorithm pseudo-code is defined as follows.

Algorithm 1: Domain Adaptation FEDA algorithm

Data: Dataset Φ, $m \times n$, composed of m instances, each with n features
Result: Dataset Φ_{DA}, $m \times 3n$ composed of m instances, each with $3n$ features

```
initialize Φ_DA empty;
define 0 as 1 × n zero vector;
if Φ from Source domain then
    foreach instance Φ[i] in Φ do
        newInstance ← ⟨Φ[i], Φ[i], 0 ⟩
        append newInstance to Φ_DA
else
    foreach instance Φ[i] in Φ do
        newInstance ← ⟨Φ[i], 0 , Φ[i]⟩
        append newInstance to Φ_DA
```

When dealing with categorical attributes, we binarize them using One Hot Encoding before applying the algorithm. With One Hot Encoding, features with n possible values will be encoded in n new features with a binary value. As an example, a feature F with possible values a, b and c will be encoded in three (number of distinct possible values of F) new features: F_a, F_b and F_c. One of these features will then have the value 1 and all the others 0, according to the original value of F. As a concrete example , instances where $F = a$ will have three new features $F_a = 1$, $F_b = 0$ and $F_c = 0$, and the original feature F is removed.

After the pre-processing algorithm, the generated feature-augmented dataset is used as the original would for training and testing. By augmenting the feature space, we are essentially letting the model learn to recognize the different feature patterns introduced by the algorithm and identifying which domain each instance belongs to. Additionally, since it can be implemented in a pre-processing stage, this algorithm can be applied independently of the machine learning model used. On the other hand, this method has poor scalability since the number of additional features grows linearly to the number of different domains.

2.3 Unsupervised Domain Adaptation

Baochen Sun et al. proposed an unsupervised algorithm for domain adaptation that aims to minimize domain shift of Source and Target by aligning data distribution from both domains [13]. This method, called Correlation Alignment (CORAL), computes the covariance matrix of the Target domain and applies a transformation to the Source domain such that both covariance matrices are identical.

One of the main advantages of this approach is that it doesn't need labels on the Target domain, making it ideal for the current eCommerce use case, where typically new merchants have some unlabeled data. Another benefit is that after processing, no additional data is generated on the Source dataset, as opposed to the FEDA algorithm, presented in 2.2, resulting in no overhead to both training and testing.

The algorithm starts by calculating the covariance matrices for Source data D_s, with dimensions $m \times n$, and for Target data D_t with dimensions $m' \times n$, where m and m' represents the respective number of instances and n the number of features. Then, an identity matrix $I_{n \times n}$ is added to each covariance matrix, resulting in matrices C_s, $n \times n$, for Source and C_t, $n \times n$ for Target, with the purpose of making each matrix full rank, meaning all rows and columns are linearly independent and C_s and C_t are invertible. After this step, Source data D_s is multiplied by the inverse square root of C_s. This process is called whitening, or sphering, and transforms D_s such that its features are uncorrelated and have variance 1. Finally, the whitened Source data matrix D_s is multiplied by the square root of C_t, an operation the authors call "re-coloring" with the Target data covariance (opposed to the whitening transformation).

A formal definition of the algorithm is presented next.

Algorithm 2: Domain Adaptation CORAL algorithm

Data: Source data D_s, $m \times n$ and Target data D_t, $m' \times n$, composed of m and m' instances, respectively, and n features

Result: Transformed Source data $D_s{}^*$, $m \times n$, with similar data distribution to D_t, $m' \times n$

define $I_{n \times n}$ as a $n \times n$ identity matrix

$C_s = cov(D_s) + I_{n \times n}$

$C_t = cov(D_t) + I_{n \times n}$

$D_s = D_s \times {C_s}^{-\frac{1}{2}}$

$D_s{}^* = D_s \times {C_t}^{\frac{1}{2}}$

After the algorithm is applied, model training resumes with the transformed Source data, making this method easily implementable with different machine learning models. An important aspect to note is that data must be numerical so the algorithm can be applied. Although the authors implemented the algorithm in Matlab, we implemented it in Python using Numpy, a free and open source Python package designed for scientific computing [14].

2.3.1 2D Visualization

To demonstrate how this algorithm operates, two datasets were randomly generated with different data distributions, acting as Source and Target. Figure 2.1 shows a 2D visualization of how the algorithm works, specifically how each transformation affects the data.

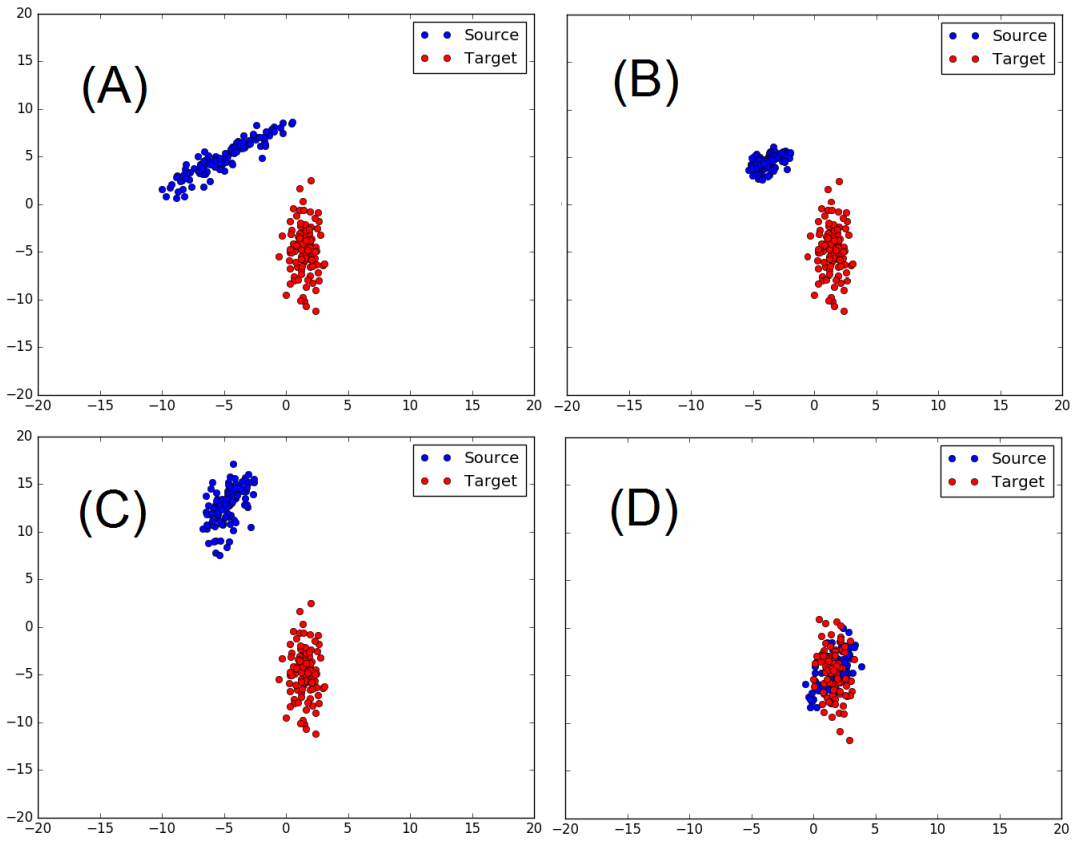

Figure 2.1: CORAL 2D visualization

Frame (A) shows the original Source (blue) and Target (red) data, with both sets having different data distribution. On frame (B) we can see the effect of the whitening transformation decorrelating the data, now resembling a sphere (or a circle, in 2D), hence the alternative name for this operation, sphering. On Frame (C) the uncorrelated Source data is re-colored with the Target covariance, seen by the identical distribution shape of both data sets. Finally, on Frame (D) the mean values of each dimension (each feature) of the transformed Source data is corrected so it is aligned with the Target's mean feature values.

2.4 Unsupervised fraud detection with Local Outlier Factor

Domain adaptation algorithms were introduced as possible tools for solving the current eCommerce use case, in which the goal is to transfer knowledge from existing data from a Source domain (current merchants) to a Target domain (new merchants), where limited amount or unlabeled data exists. Taking into account this scenario, and knowing fraudulent payments are instances deviating from the typical transaction patterns, outlier detection techniques could also be used to tackle the problem.

With outlier detection, fraudulent payments could be identified for new merchants with unlabeled data and without relying on historical data from other merchants. On the other hand, the new merchant would need a considerable number of transactions so the typical payment behavior is accurately represented and, consequently, identified outliers are actual fraud instances and not just isolated legitimate transactions.

In this work, we apply Local Outlier Factor (LOF), an outlier detection algorithm introduced in 2000 by Breunig et al. [15]. The core concept of this method is to estimate how isolated an instance is by comparing densities of each point with the density of their neighborhood. Points with significant lower densities than neighbor points are considered outliers. To understand how this algorithm works, the following metrics need to be defined:

- **N_k(p)** - the set of k points closest to point p;
- **k-distance(p)** - distance to the k^{th} nearest point of p, for k positive integer. As an example, considering $k = 2$, if point p has p_1 and p_2 as nearest neighbors at distances 3.4 and 2.1, respectively, the $k\text{-}distance$ is 3.4;
- **Reachability-distance$_k$(p,o)** - the reachability distance of point p from o represents the euclidean distance between points p and o but, at minimum, the $k\text{-}distance(o)$. The formal definition is:

  $$reachability\text{-}distance_k(p, o) = max\{k\text{-}distance(o), d(p, o)\}$$

  where $d(p, o)$ is the distance between points p and o;
- **Local Reachability Density (LRD) of** p - local reachability density of point p is defined as:

$$LRD(p) = 1/\frac{\sum_{q \in N_k(p)} reachability\text{-}distance_k(p, q)}{k}$$

- **Local Outlier Factor (LOF)** - represents the final score each point will be awarded. Points with score values near 1 indicate points belonging to a cluster while score values significantly superior than 1 are classified as outliers. The formal definition of LOF is:

$$LOF_k(p) = \frac{\sum_{q \in N_k(p)} LRD(q)}{k} / LRD(p)$$

The algorithm starts by calculating the k-nearest neighbors for all data points. Then, for each point p' and its neighborhood $N_k(p')$ the reachability-distance is determined. Finally, using the reachability-distance, LRD and LOF are calculated. The formal definition of the algorithm is as follows:

Algorithm 3: Local Outlier Factor algorithm

Data: Parameter k; Dataset D
Result: LOF score for each point

Determine the $N_k(p')$ for each $p' \in D$
Calculate the reachability-distance for each point $p' \in D$ and its neighborhood $N_k(p')$
Calculate the Local Reachability Density for all points $p' \in D$
Calculate the Local Outlier Score for all points $p' \in D$

An important aspect about the algorithm is that its performance is heavily dependent on the pre-selected parameter k. The original paper addresses this issue by studying the impact of the parameter within datasets with known distributions. To determine the lower bound for k, the authors use a uniform distribution dataset and select a k value such that there are no outliers, that is, no LOF values significantly greater than 1. This lowerbound was set at $k = 10$. For an upperbound, k was determined empirically with different datasets and found to be highly dependent of data distribution itself. In our experiments, we determine the optimal k by parameter sweep on a training dataset, starting at $k = 8$ and multiplying by 2 on each iteration.

Advantages of this algorithm include being able to identify outliers taking into account not only distances but also local neighbors and the density of the cluster these points belong or are near to. Figure 2.2 illustrates this concept.

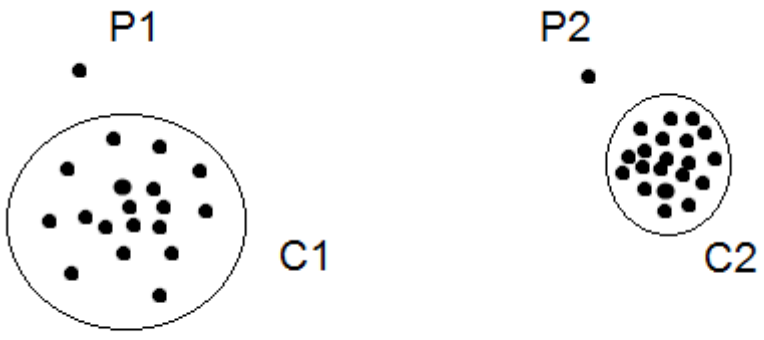

Figure 2.2: LOF score illustration based on proximity to sparse or dense clusters.
P1 may not be an outlier due to its proximity to the sparse cluster C1. P2 might be considered an outlier since C2 is a dense cluster. Adapted from [16].

Figure 2.2 illustrates how LOF score points when dealing with data clusters with different densities. Although points P1 and P2 are at the same distance of clusters C1 and C2, respectively, P1 might not be considered an outlier due to C1 being a sparse cluster. Oppositely, point P2 is likely to be considered an outlier due to the higher density of cluster C2.

Another important aspect about LOF is that, instead of providing a binary classification (outlier/regular), it assigns a score to each point, where scores near 1 represent regular points and scores with significant high values are considered outliers. The threshold at which a point is considered an outlier can be quite ambiguous and might depend on data itself. In our experiments, after applying LOF, scores are normalized between 1 and 1000 indicating lower likelihood of being an outlier (fraud) and higher likelihood, respectively. This scoring method is identical to the process of using a model to score transaction instances, which is addressed and further explained in section 4.2.

Chapter 3

Methodologies

In this chapter we introduce some algorithms used in Machine Learning, specifically Decision Trees and Random Forests. Although this work does not focus particularly on Machine Learning algorithms it is critical to understand how these algorithms operate. Additionally, we chose to present Decision Trees and Random Forests as they represent the main algorithms used by Feedzai and have also been proposed in the literature as capable tools for fraud detection. [17, 18]

Closing the chapter, we introduce and define performance metrics for model evaluation and comparison.

3.1 Decision Trees

A decision tree is a structure composed of nodes, branches and leaves, used for classification in supervised Machine Learning. Branches connect two nodes together and each node can have multiple output nodes. Starting at the root node of the tree, data attributes are checked against a certain condition and routed to another node according to the result of that test. This process continues, until the current instance reaches the end of the tree, a leaf. Leaves have an associated class that will label all instances reaching that leaf. [19]

As an example, consider the classic case of a bank deciding if it should hand out a loan to a client. A possible decision tree for this scenario is presented in Figure 3.1.

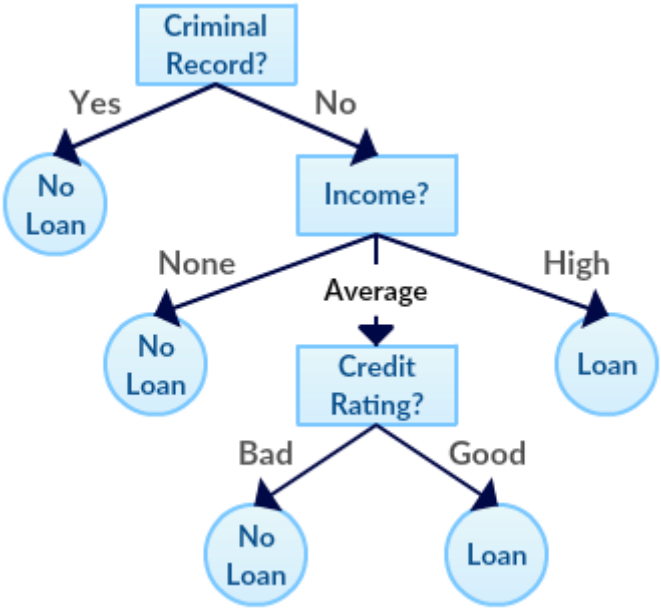

Figure 3.1: Decision Tree example for deciding if a bank client is eligible for a loan. Internal nodes (rectangles) represent a test on a particular attribute. Leaves (circles) represent all the possible classes instances will be classified as.

In this example, the first attribute being checked is Criminal Record - if a client has committed any crime, the loan application is immediately rejected. If there is no criminal record, the income attribute is checked, with three possible outcomes. First, if the client has no income, the loan is rejected; second, if the income is high, the loan is accepted. Finally, for an average income, Credit Rating attribute is analyzed. In the case of a good credit rating (good debt paying reputation) the loan application is accepted, otherwise it is rejected.

In summary, each internal node represents a test on a particular attribute, each branch represents a possible value of the attribute being tested and leafs contains the possible classes instances will be labeled as. An important aspect about decision trees is that nodes closer to the root node should test the attributes that provide the best discrimination of data in the different classes. A solution to find these attributes will be presented in the next section, as we describe the ID3 algorithm, an algorithm for building decision trees.

3.1.1 ID3

ID3 is an algorithm for building decision trees based on Information Theory, and was first introduced by J. R. Quinlan in 1986. [20] It relies on the concept of information gain to select the 'best' attributes for splitting data. Intuitively, if a particular attribute can separate all instances into different classes, it can be placed in the root node of a decision tree, and no further attributes need to be tested. This would result in a single node decision tree where all instances could be correctly identified (at least from the training set) by looking solely at one attribute.

One particular issue with this algorithm is that it is defined only for problems with two classes, usually referred as positive and negative class. The pseudo-code for ID3 is described next. [21]

Algorithm 4: ID3

Data:

- Dataset S with positive and negative training instances
- Set A of attributes and the corresponding possible values
- An attribute selection method T

Result: A decision tree.

if *All instances in S are from the same class C* **then**
    Create node N with label C;
    Return N;
if *A is empty* **then**
    Create node N with the majority class in S ;
    Return N;
Select an attribute a from A according to T;
Divide S into subsets according to the possible values of a;
Call the algorithm recursively;
Build a decision tree with attribute a as root node, and branches for each possible value of a connected to the corresponding subtree of each S subset;

The most important aspect of the ID3 algorithm is how do we chose the 'best' attribute. For this task, the concept of information gain is applied. Information gain translates to how well a particular attributes divides the corresponding instances into the different classes. To calculate this parameter, we first need to introduce the concept of entropy.

Given a set S with a positive (P) and a negative (N) class, entropy can be calculated by:

$$I(S) = -p_p \log_2(p_p) - p_n \log_2(p_n)$$

with p_p being the probability of a particular instance belonging to the positive class and the same for p_n for the negative class. For example, considering a set S of 17 instances with 6 positive and 11 negative, the entropy for this set would be:

$$I(S) = I(6, 11) = -\tfrac{6}{17} \log_2(\tfrac{6}{17}) - (\tfrac{11}{17}) \log_2(\tfrac{11}{17})$$

Now considering attribute A with v possible values, set S will be partitioned into sets $\{S_1, ..., S_v\}$. Each set S_i contains p_i instances of the positive class and n_i instances of the negative class. The entropy for selecting this attribute is:

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I(p_i, n_i)$$

Finally, the information gain of a particular attribute is defined as:

$$Gain(A) = I(p, n) - E(A)$$

After calculating the information gain for all attributes, we chose the one with the greatest information gain or, equivalently, the one that minimizes $E(A)$. As a concrete example, ID3 is applied to simple dataset with only two attributes, shown in Table 3.1.

Income	Credit Rating	Class
None	Good	-
None	Bad	-
None	Bad	-
Average	Good	+
Average	Good	+
Average	Good	+
Average	Bad	-
High	Good	+
High	Bad	+
High	Bad	+

Table 3.1: Example training dataset for creating a decision tree with ID3.
Positive class represents clients that paid back a loan; Negative class represents clients that failed to pay back a loan.

The first step is to calculate the entropy of the dataset:

$$\begin{aligned} I(S) &= I(6,4) \\ &= -(6/10) * log_2(6/10) - (4/10) * log_2(4/10) \\ &= 0.971 \end{aligned}$$

The second step is to calculate the information gain for each attribute:

$$\begin{aligned} E(Income) = &(3/10) \times (-(3/3) \times log_2(3/3) - (0/3) \times log_2(0/3)) \\ &+(4/10) \times (-(3/4) \times log_2(3/4) - (1/4) \times log_2(1/4)) \\ &+(3/10) \times (-(0/3) \times log_2(0/3) - (3/3) \times log_2(3/3)) \end{aligned}$$

$$\begin{aligned} G(Income) &= I(6,4) - E(Income) \\ &= 0.971 - 0.325 \\ &= 0.646 \end{aligned}$$

$$\begin{aligned} E(CreditRating) = &(5/10) \times (-(2/5) \times log_2(2/5) - (3/5) \times log_2(3/5)) \\ &+(5/10) \times (-(4/5) \times log_2(4/5) - (1/5) \times log_2(1/5)) \end{aligned}$$

$$\begin{aligned} G(CreditRating) &= I(6,4) - E(CreditRating) \\ &= 0.971 - 0.846 \\ &= 0.125 \end{aligned}$$

Finally, the attribute with the greatest information gain, Income, should be selected as the root node, creating a branch for each possible value of this attribute: 'High', 'Average' and 'None'. Next, for each of these branches we apply the algorithm recursively until there are no more attributes or we end up on a leaf with instances all from the same class. For example,

value 'High' of attribute Income corresponds only to positive instances, as such, there is no need to use the Credit Rating attribute to further split the data. The same happens to value 'None', but for value 'Average' we need to select another attribute to correctly classify all instances. In this example, there is only one more attribute (Credit Rate) and its possible values (Good, Bad) can separate all instances belonging to 'Average' Income into different classes. Completing the algorithm, the decision tree presented in Figure 3.2 is obtained.

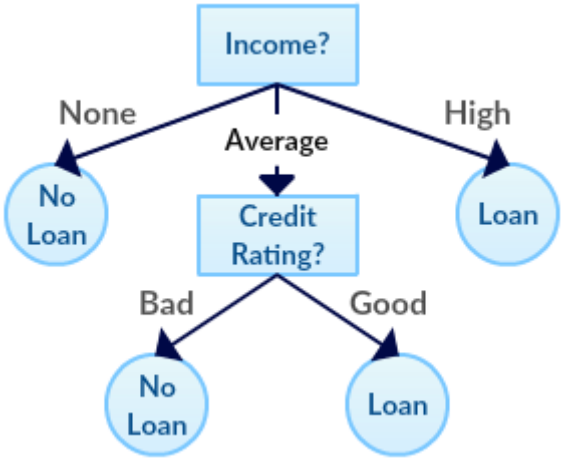

Figure 3.2: Decision Tree obtained from applying ID3 algorithm to the dataset from Table 3.1.

Despite being straightforward and easy to implement, ID3 has a few downsides, namely, it does not guarantee an optimal solution and often falls on local optima since the information gain method used is a greedy approach. Furthermore, it is vulnerable to overfitting and can only be applied to categorical attributes. Although discretization of numerical attributes is a possibility, it is often impractical. Finally, ID3 also suffers from bias since the information gain is biased for attributes with many possible outcome values.

3.1.2 C4.5

C4.5 was introduced in 1993 by the same author of ID3, J. R. Quinlan. [22] This algorithm was developed with the goal of overcoming the limitations of ID3, namely the issues regarding continuous attributes, overfitting and the bias of information gain towards attributes with many possible values.

Continuous Attributes

For continuous attributes, training instances are first sorted according to the values of those attributes, then a threshold value Z is chosen such that the possible values for the attribute are split among two subsets, one with values less or equal to Z and the other with values greater than Z. For example, considering an attribute with $\{a_1, ..., a_v\}$ possible values, there are $v - 1$ possible values for Z. Usually, split value Z is chosen as the mid point of consecutive values:

$$Z = \frac{a_i + a_{i+1}}{2}$$

Z is then calculated for all possible $v - 1$ values, resulting in two subsets of A, D_1 and D_2, where $D_1 <= Z$ and $D_2 > Z$. The entropy of the attribute, $E(A)$ is then determined as in ID3, but for

all $v-1$ split values of Z. The value of Z that maximizes information gain is finally chosen as the optimal split value.

Overfitting

Decision trees can occasionally suffer from overfitting. Overfitting means the algorithm adapts to training data so well that it becomes useless for classifying future testing data. C4.5 tries to overcome this problem with a method of postprunning (after the decision tree is complete), called 'pessimistic pruning', by replacing entire branches and the corresponding sub-tree with a leaf. [23]

Sub-tree replacement starts by estimating classification errors with a validation set separated from training data. If the combined error for a set of leaves is greater than the parent node, that sub-tree is pruned.

Bias

ID3 tends to be biased towards attributes with many different values. To illustrate this issue, consider an attribute a with distinct values for each instance. The entropy for this attribute would be minimal and information gain would be maximized. As such, this attribute would be selected for a splitting node with many output branches, one for each possible value of a. Although this attribute could correctly classify training data, it would be useless for future data.

To overcome this limitation, C4.5 introduces the gain ratio. The gain ratio relies on split information, which represents the potential information for splitting data into v subsets by choosing attribute A with v possible values. Split Information and Gain Ratio are defined as:

$$SplitInfo(A) = -\sum_{i=1}^{v} \frac{p_i + n_i}{p+n} \times log_2(\frac{p_i + n_i}{p+n})$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

Similarly to information gain in ID3, the attribute that maximizes Gain Ratio is chosen as the splitting node.

3.2 Random Forests

Random Forest algorithm was first introduced by Leo Breiman in 2001 [24] and rely on the concept of bootstrap aggregating (also referred as bagging). Bootstrap aggregating is an ensemble method which combines multiple individual models into a global model with the goal of improving the overall classification performance.

Ensemble methods tend to perform better than the individual classifiers they are composed of, as the global classification is determined by average or majority voting. In this case, for the ensemble method to provide an incorrect classification, then more than half of the individual classifiers would need to provide an incorrect result as well. To minimize this, considerable diversity between the individual models should exist.

Random Forests rely on random instance sampling and attribute selection to guarantee distinct decision trees.

The algorithm starts by creating an in-bag set of instances by sampling with replacement from the training set. Sampling with replacement means the same training instance can be selected multiple times or, in other words, the in-bag set can be composed of duplicate instances. Usually, the number of instances of in-bag set is the same as in the training set. The remaining samples of the training set that are not selected for the in-bag set are referred to as out-of-bag data and are used for validation or testing. This process is repeated for the total number of desired decision trees. For example, if a particular Random Forest has k decision trees, an in-bag set is generated k times, once for each tree.

Finally, for every node of each individual tree, a subset of the total m attributes is randomly selected and the best splitting attribute is chosen. Examples of splitting criteria to select the best attribute were discussed for algorithms ID3 and C4.5 before, in sections 3.1.1 and 3.1.2. A frequent approach to the size of the subset of attributes is to randomly select $\sqrt{m}$ attributes.

In summary, the complete process to build a random forest is, for each tree:

1. Create an in-bag set by sampling the training set with replacement. Instances that are not selected are known as out-of-bag data and is used for testing of the individual tree or the complete Random Forest;

2. For each node, select the best splitting attribute from a random subset of the total m attributes. The size of the subset is considerably smaller than m, usually $\sqrt{m}$;

3. Let the tree grow without pruning.

Once the Random Forest is complete, classification is done by taking all of the individual decision tree outputs, which can be weighted, and selecting the majority or the average, depending on the context. Weights can be determined by using out-of-bag data to validate and test the performance of each individual decision tree or the complete forest.

An important aspect about the performance of Random Forests is that it is directly related to the variability between individual trees. As such, ensuring low correlation between trees is critical.

3.3 Performance Metrics

After model training and testing, multiple metrics are calculated in order to evaluate the performance of the model. During testing, each transaction is awarded a score between 1 (less likely to be fraud) and 1000 (most likely to be fraud) and, if the score is greater than or equal to a certain Fraud Threshold, the transaction is classified as fraud (positive class); if the score is less than the Fraud Threshold, the transaction is classified as not-fraud (negative class).

After classification, by comparing the actual class of each transaction with the model's classification the following metrics are calculated: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). These metrics are defined as:

- **TP** - number of transactions **classified as fraud** that are **actually fraud**, that is, the number of transactions correctly classified as fraudulent;

- **TN** - number of transactions **classified as not-fraud** that are **actually legitimate**, that is, the number of transactions correctly classified as not fraudulent;

- **FP** - number of transactions **classified as fraud** that are **actually legitimate**, that is, the number of transactions incorrectly classified as fraudulent;

- **FN** - number of transactions **classified as not-fraud** that are **actually fraud**, that is, the number of transactions incorrectly classified as legitimate;

These values are then used to calculate various performance metrics for model evaluation such as Recall, Precision, False-Positive Rate, Area-Under-Curve (AUC) and Partial Area-Under-Curve (pAUC). Each of these performance metrics is defined next.

3.3.1 Recall

Recall is a performance metric that tells the fraction of relevant instances that are identified, that is, in the context of fraud detection, the percentage of fraudulent instances identified. The formal definition of Recall is:

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

Recall, in itself, is not very useful since it cannot accurately evaluate a model. For instance, let's consider a model that classifies all instances of a certain dataset as fraud. If so, then all fraudulent transactions are correctly identified. Thus, $FN = 0$ and, by definition, $Recall = 100\%$. This result might give the idea that all fraudulent transactions were identified (and rightly so), however, all legitimate transactions were also flagged as fraudulent, making the model ultimately useless.

To accurately evaluate a model with Recall, for the reason described above, Precision or False-Positive Rate also need to be taken into account. These performance metrics are defined next.

3.3.2 Precision

Precision translates to the fraction of instances that were selected that are actually relevant. In other words, in the context of fraud detection, the percentage of transactions classified as fraud that are actually fraudulent. Precision is formally defined as:

- Precision = $\frac{TP}{TP+FP}$

Similarly to the reasoning that was described for Recall, Precision, by itself, can also be considered of limited use.

3.3.3 False-Positive Rate

The FPR represents the fraction of instances classified as positive that are actually negative. In other words, it represents the percentage of transactions classified as fraud that are actually legitimate. The formal definition of FPR is:

- FPR = $\frac{FP}{FP+TN}$

In the context of fraud detection, this metric is extremely important. It should be noted that each positive (transaction flagged as fraudulent) might require human intervention, so a high FPR will increase costs to prohibitive levels. As such, while Recall and Precision should be maximized, keeping a low FPR is of utmost importance.

3.3.4 Money-Recall

Money-Recall, usually denoted as Recall$, is a money-based metric similar to Recall, but instead of relying directly on the number of TP and FN, it is calculated using the monetary value associated with the corresponding transactions.

Using the Recall definition, Money-Recall is easily determined by finding the money amount of each term, that is, the money associated will all TP and FN, commonly denoted as $TP and $FN, respectively. $TP represents the sum of money of all detected fraudulent transactions and, equivalently, $FN represents the sum of money of all fraudulent transactions that were not detected. In summary, Money-Recall represents the percentage of money from fraudulent transactions that is effectively detected, and the formal definition is the following:

- Money-Recall = $\frac{\$TP}{\$TP+\$FN}$

Money-Recall is very useful in a business perspective since it represents all detected money involved in fraudulent transactions. Normally this would be the ideal metric to present to potential clients as it effectively shows how much money the system could possibly save. However, since the average transaction monetary amount can vary considerably, Money-Recall is not appropriate to compare models, as few transactions can represent large amounts of money and the opposite can also happen. As such, a model m can exhibit a higher Money-Recall than model m' for the same FPR, but actually have a lower Recall, making it difficult to compare performance.

3.3.5 Area-Under-Curve

To define Area-Under-Curve (AUC) we must first introduce the Receiver Operating Characteristic (ROC) curve: the ROC curve plots Recall (vertical axis) versus FPR (horizontal axis), and, in our context, is obtained by calculating said metrics for all possible values of Fraud Threshold - from 1 to 1000. This operation will generate 1000 pairs of Recall - FPR values that will be plotted against each other. An example of a ROC curve is displayed in Figure 3.3.

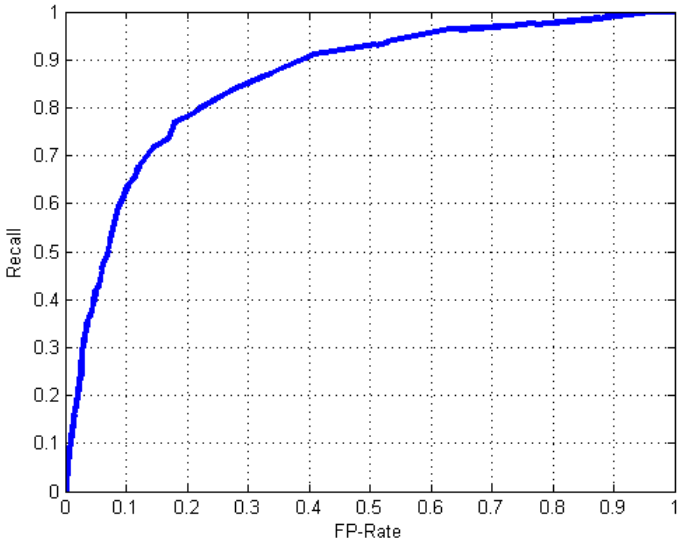

Figure 3.3: ROC curve example.

AUC is usually a good metric to compare models and is determined by joining point (0,0) with the classifier's Recall-FPR points, joining these with point (1,1) and finally by calculating the area under the obtained curve.

In an ideal scenario, AUC would be 1, meaning a FPR of 0% and 100% Recall for all threshold levels. These values would represent a model that could detect all fraudulent transactions while committing no errors.

3.3.6 Partial Area-Under-Curve

In fraud detection, limiting FPR to a small value is usually a priority in order to reduce costly human intervention dealing with false alarms. As such, we may want to calculate what is referred to as Partial Area-Under-Curve (pAUC). pAUC is calculated exactly as AUC except instead of considering the complete ROC curve, we limit the FPR (horizontal axis) to a certain maximum value. To illustrate this concept, Figure 3.4 shows the pAUC area for a maximum FPR of 10%, commonly denoted as pAUC(0.1).

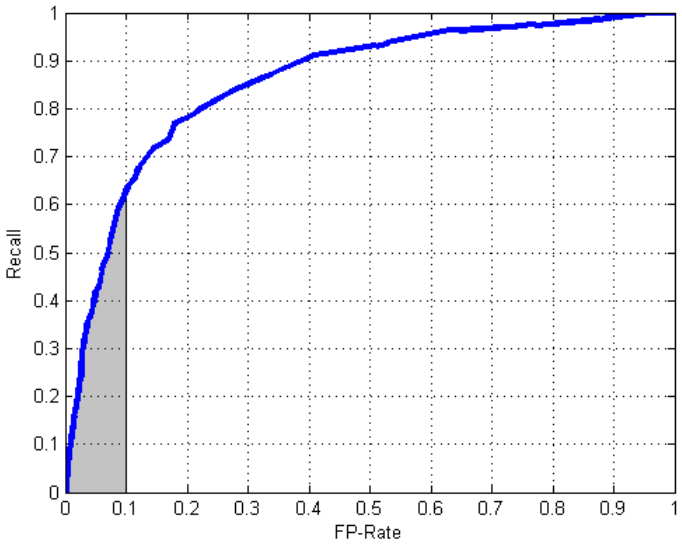

Figure 3.4: ROC curve showing, in grey, pAUC for maximum FPR of 10%.

pAUC is generally a better performance metric than AUC when there is a maximum allowed FPR. For example, a client might restrict FPR to a certain maximum value p. As such, when building and evaluating models we should consider the model with the highest pAUC(p), even if there are other models with greater AUC.

What happens in this scenario is that the Fraud Threshold is set to a certain value to which the model stays below the maximum allowed FPR p, thus, pAUC(p) is a superior metric than AUC to compare models, since they will operate at FPR lower than or equal to p.

Chapter 4

Experimental Results

In this chapter we explore two algorithms for Domain Adaptation, introduced in sections 2.2 and 2.3, and an unsupervised ML algorithm, presented in section 2.4. We evaluate both approaches by comparing their performance with traditional ML methods on different scenarios.

We start the chapter by describing the datasets used in all experiments, followed by the preliminary results obtained in the first semester. Finally, the experimental results achieved during the second semester are presented and discussed.

4.1 Datasets

All experiments relied on four datasets from different domains, described next and anonymized for confidentiality reasons:

- Dataset A - contains eCommerce transactions obtained from financial institutions. This dataset contains a large number of transactions but with a low number of features.
- Dataset B - contains transactions from a payment service provider serving multiple eCommerce merchants. It must be noted that these merchants can operate in different types of business. This dataset contains only a fraction of the number of transactions of A but a higher number of features.
- Dataset C - this dataset is from an eCommerce merchant aggregating multiple sub-merchants. These sub-merchants operate all in the same business, that is, all of them sell similar types of goods.
- Dataset D - similar to A, this dataset contains transactions from a financial entity operating as a bank.

A complete characterization of all four datasets is presented in Table 4.1.

Dataset	Time window (months)	Total Transactions	Fraud (%)	Number of Features
A	36.0	5414507	0.53%	44
B	15.0	148228	1.21%	151
C	15.5	1324505	2.02%	151
D	6.0	500300	0.15%	469

Table 4.1: Datasets description.

4.2 Tools and Simulation

In this section, we describe the main tools and the simulation process used in the developed work.

Tools

Training and performance evaluation of the developed models were accomplished using Feedzai's proprietary data processing, feature engineering and machine learning tools, which include:

- **PKernel** - a Complex Event Processing (CEP) Engine that uses a stream processing language, Pulse Query Language, defined next, to manipulate streams of events. Event processing refers to the concept of processing real-time data, meaning that events can arrive continuously at any point in time, commonly referred to as stream. CEP can be defined as the processing of these streams with the goal of identifying complex patterns by analyzing low complexity events. [25]

- **Pulse Query Language (PQL)** - a stream processing language, with a query syntax similar to Structured Query Language (SQL), used to define the transformations PKernel applies to the incoming streams of events. Using PQL, we can configure PKernel to apply feature engineering and define output features. After processing, data instances can be outputted in Attribute-Relation File Format (ARFF) format;

- **Model builder** - with the output ARFF file defined above, Model builder is used for model training. This tool supports several algorithms, such as Random Forests or Naive Bayes, among others, and multiple parameters for each model. A diagram for model training is displayed in Figure 4.1;

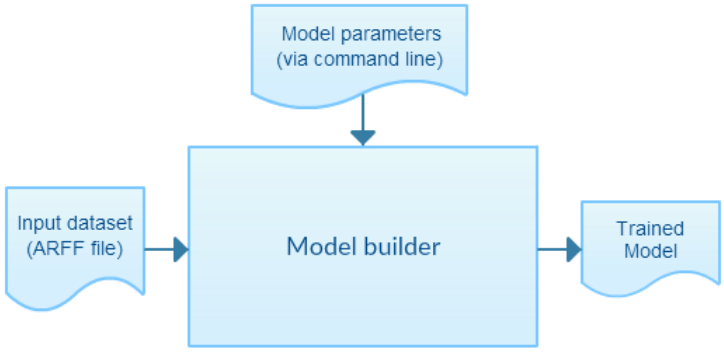

Figure 4.1: Model builder process for model training.

- **Simulator** - this tool simulates, in batch mode, a live production system where local datasets are treated as a continuous stream of events, which, in this context, are transactions composed of several fields such as timestamp, amount of money transferred, cardholder's name and so on.

  Before going into detail on how the Simulator works, the concept of profile must first be introduced - a profile is a set of data from a particular card owner, containing information regarding all transactions made by that card over a certain time frame. As an example, a profile can contain data such as the number of transactions made in the last two weeks or the average value of the last five transactions. Every time a new transaction occurs, the profile of the respective card number is updated. This profiling step is done by the PKernel engine.

The Simulator ties together the tools defined above with two methods of operation: instance-generation mode and model-testing mode.

In instance-generation mode two input files must be supplied: a dataset in JavaScript Object Notation (JSON) format and a PQL file. Using the transformations defined in the PQL file, PKernel manipulates the events in the JSON dataset and outputs an ARFF file.

Model-testing mode implements the instance-generation mode with an additional step: instance scoring and performance metrics. For this step, a trained model (trained previously with Model builder) must be inputted. After the instance-generation step, the Simulator uses the input model to classify data from the generated ARFF file by scoring each transaction from 1 (less likely to be fraud) to 1000 (most likely to be fraud). Following, for a given threshold, all the metrics described in section 3.3 are calculated.

Figures 4.2 and 4.3 show a diagram for the simulator process in instance-generation and model-testing modes, respectively.

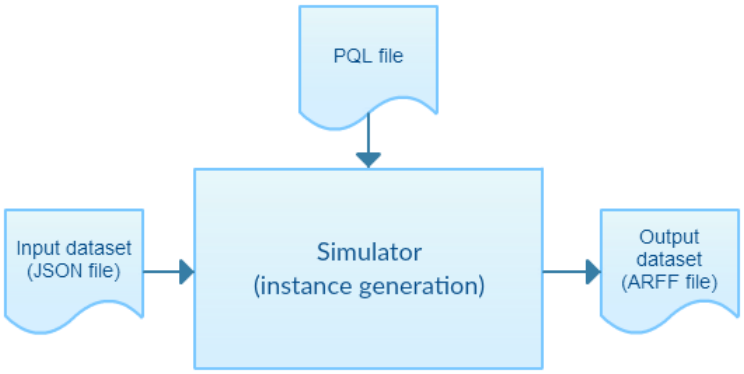

Figure 4.2: Simulator process in instance-generation mode.

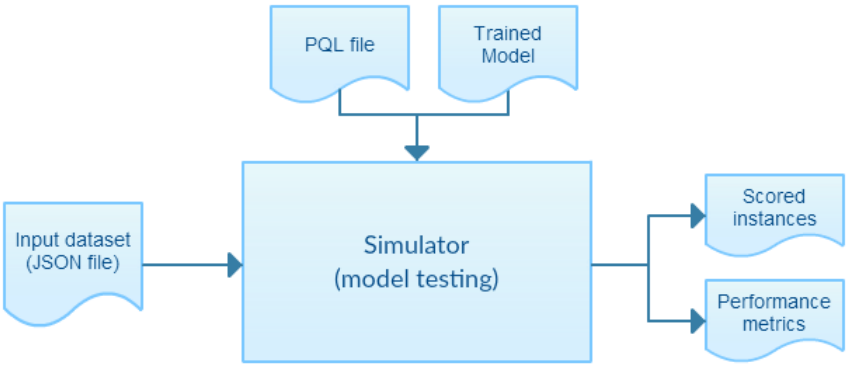

Figure 4.3: Simulator process in model-testing mode.

In summary, the purpose of the Simulator tool is to test models offline and make sure they generate the same results when deployed in production.

With PQL, we can easily define new profiles, effectively implementing feature engineering, by manipulating streams of transactions and transforming them into instances. These instances are then converted to ARFF data files for model training or directly passed to an already trained model for transaction scoring and classification with the Simulator tool.

Simulation

The simulation process starts with raw payments data in a JSON file, whose format can be found in [26]. The file is then split in two sets: the first 70% transactions as the training set and the last 30% as the testing set.

Before training a model with Model builder, the training set must first be generated with Simulator in instance-generation mode, as described before and shown in Figure 4.2. This profiling stage will apply the transformations defined in a PQL file to calculate and extract from the raw data the final features that will be used in training and classification, and finally outputting these instances in ARFF format.

With the generated ARFF file, a machine learning model is trained with Model builder. As mentioned before, this tool supports various algorithms and multiple parameters for each.

Finally, the testing set, in JSON format, is inputted to Simulator along with the trained model and the same PQL file used to profile the training set. The Simulator will then score each transaction of the testing set from 1 to 1000 and calculate the performance metrics for each score value, as described before.

For the implementation of the DA algorithms presented in 2.2 and 2.3, a python script was developed. Before training or testing, this script is applied to the ARFF file defined before, executing the respective algorithm transformations and outputting the final ARFF file that will then be used for model training or testing. Figure 4.4 shows a diagram of this process and at what step is DA implemented.

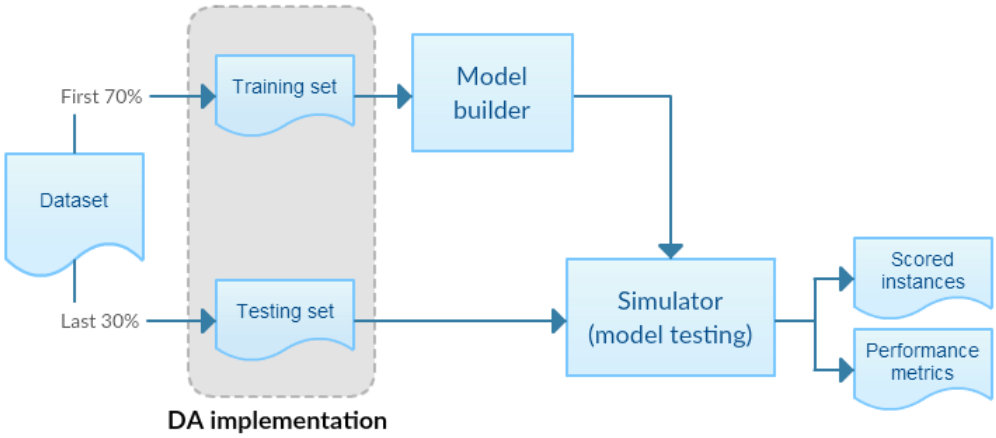

Figure 4.4: Overview of the complete simulation process.

In order to explore the LOF algorithm, we chose to use ELKI, an open source data mining framework focusing on unsupervised methods such as clustering and outlier detection. [27]

In the next section all experiments and the corresponding results are presented.

4.3 Results and Discussion

In this section, we present and compare the results between the baseline experiments and the DA algorithms implemented, as well as the unsupervised method, LOF, applied with ELKI software. This section starts with the preliminary experiments completed during the first semester, where the Frustratingly Easy Domain Adaptation (FEDA) algorithm is applied to datasets A (Source domain) and B (Target domain). The tests to compare both approaches use multiple combinations of dataset A and B and were designed in order to simulate the following scenarios: the Target domain has no labeled data, thus only data from the Source domain can be used for model training;

Target domain has some data that can be used in model training. A more detailed definition of these tests is presented next.

4.3.1 First Semester Preliminary Results

4.3.1.1 Baselines

The first test considered was calculating baselines with the datasets described in 4.1 which will then serve as reference for the implementation of the DA algorithm presented in 2.2. The first baseline, called **JustB**, was determined using only dataset B both for training (first 70% instances) and testing (last 30% instances). The second baseline, called **JustA**, used dataset A for training and dataset B (last 30% instances) for testing. For the third baseline, called **A+B**, dataset A and the first 70% instances of B were used for training and the last 30% instances of B for testing. In summary, the following baselines were calculated:

- **JustB** - Train (B [first 70% instances]) $\rightarrow$ Test(B [last 30% instances])
- **JustA** - Train (A) $\rightarrow$ Test(B [last 30% instances])
- **A+B** - Train (A + B [first 70% instances]) $\rightarrow$ Test(B [last 30% instances])

The following Table 4.2 show the performance metrics Recall, Precision, FPR, AUC and pAUC(0.05) obtained for each baseline for maximum FPR below 1%. The algorithm used for model training was a particular implementation of Random Forests by Feedzai, using the same parameters for all baselines: in bag percent - 50%; trees - 100; undersampling - 65%.

Baseline	Recall	Precision	AUC	pAUC(0.05)
JustB	55.70%	34.13%	94.89%	3.20%
JustA	12,05%	10,28%	85,87%	1.22%
A+B	30,62%	22,27%	93,40%	2.42%

Table 4.2: Evaluation metrics for Baselines JustB, JustA and A+B.

The best performance is shown by Baseline **JustB**, as expected, since the model was trained and tested with data from the same domain. Similarly, Baseline **A+B** shows better performance than Baseline **JustA** as it used data from the Target domain for model training, although most of the training instances were from a different domain. A visual perspective of the performance is shown in Figure 4.5 displaying the ROC curve of the three Baselines, for FPR below 10%.

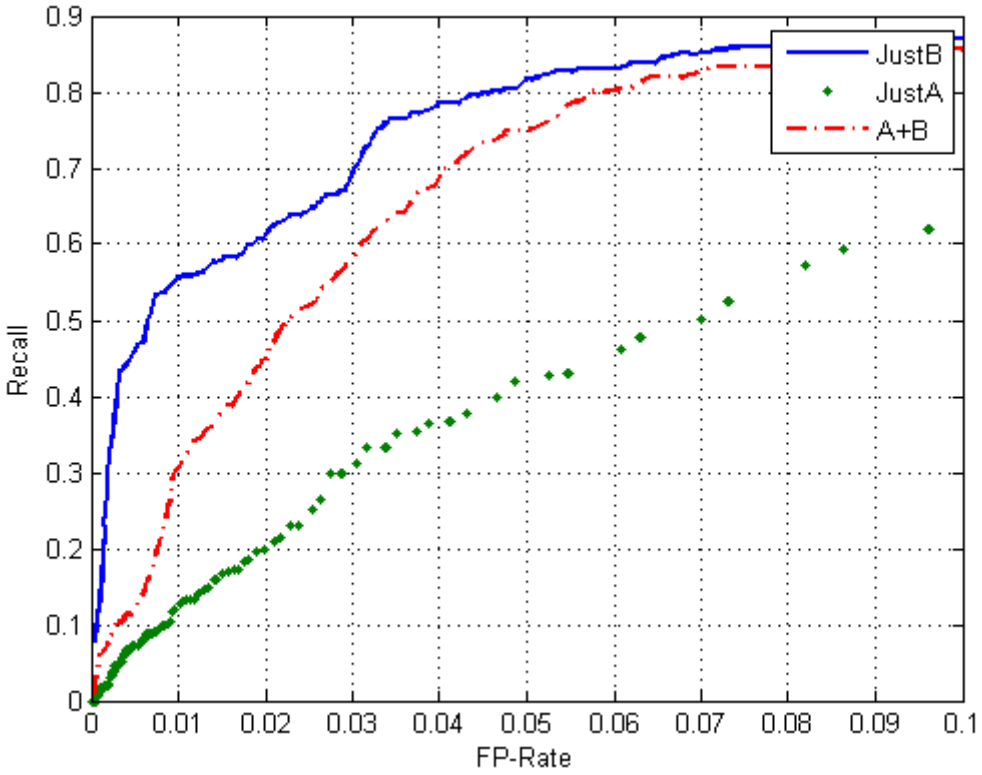

Figure 4.5: ROC curves for the three Baselines.

4.3.1.2 Domain Adaptation implementation

For this task, the DA algorithm described in 2.2 was implemented for Baseline **JustA** and Baseline **A+B**. It should be noted that for Baseline **JustB** the algorithm does not apply since both training and testing data are from the same domain.

The implementation consisted on pre-processing datasets A and B by feature-augmentation and then using them for training and testing exactly the same way as for Baseline **JustA** and Baseline **A+B**, as described in 4.3.1.1. One important aspect of the implementation is that categorical features are encoded with One-Hot Encoding, as explained in section 2.2. Table 4.3 shows the performance results for Baseline **JustA** and Baseline **A+B** with DA and Figure 4.6 displays the ROC curves for all Baselines, with and without DA, for FPR below 10%.

Baseline	Recall	Precision	FPR	AUC	pAUC(0.05)
DA(JustA)	7.49%	8.68%	0.72%	73.54%	0.93%
DA(A+B)	46.25%	31.00%	0.94%	94.96%	3.10%

Table 4.3: Evaluation metrics for Baselines **JustA** and **A+B**, with Domain Adaptation.

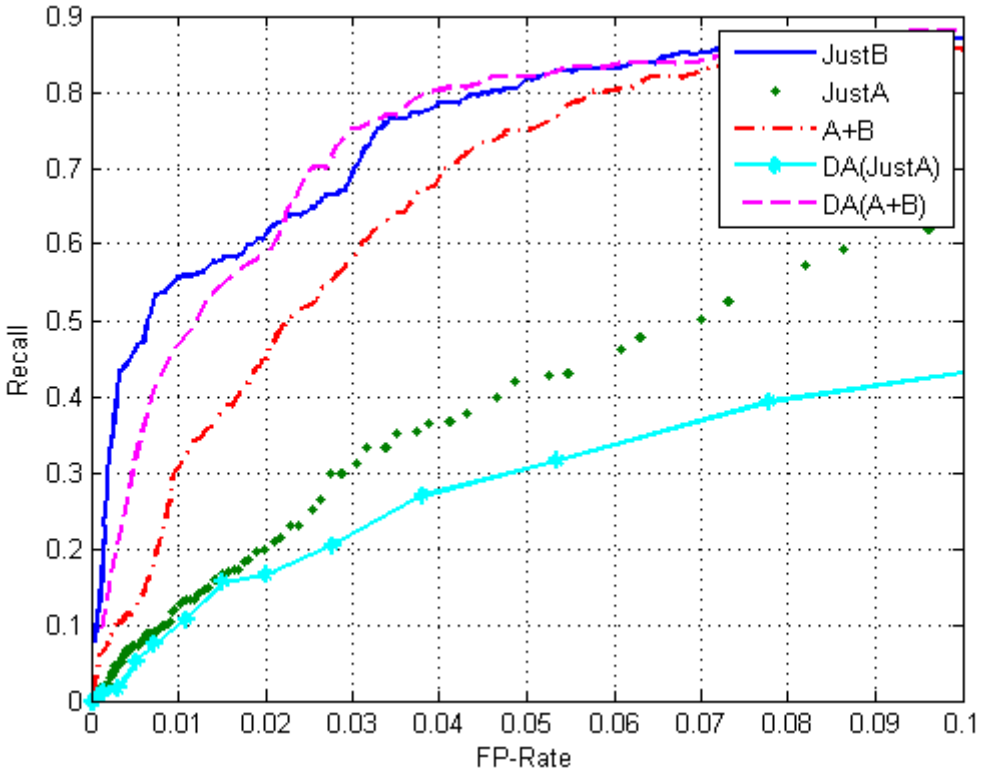

Figure 4.6: ROC curves for the Baselines, with and without Domain Adaptation.

By analyzing the performance metrics for Baseline **JustA**, we can see a decrease in performance when using FEDA. Intuitively it makes sense, since the model was trained with instances belonging only to the Source domain and, while classifying instances from the Target domain, the model has to deal with features that were always zero during training. On the other hand, evaluation metrics for Baseline **A+B** show a significant increase in performance with the DA algorithm, especially in pAUC(0.05). This result can be explained by the training dataset containing some instances from the Target domain and the pattern of testing instances being recognized by the model. The performance of this model is similar to Baseline **JustB**, with a minor increase in AUC: 94.89% in Baseline **JustB** versus 94.96%; but a slight decrease in pAUC(0.05): 3.20% in Baseline **JustB** versus 3.10%.

In order to understand the performance difference between using FEDA or just the Target data as new data becomes available, a new set of tests were considered:

- **B1m** - Train(B [last month of first 70% instances]) → Test(B [last 30% instances])
- **B3m** - Train(B [last 3 months of first 70% instances]) → Test(B [last 30% instances])
- **B6m** - Train(B [last 6 months of first 70% instances]) → Test(B [last 30% instances])
- **AB1m** - Train(A + B [last month of first 70% instances]) with DA → Test(B [last 30% instances])
- **AB3m** - Train(A + B [last 3 months of first 70% instances]) with DA → Test(B [last 30% instances])

- **AB6m** - Train(A + B [last 6 months of first 70% instances]) with DA → Test(B [last 30% instances])

These tests were designed to answer the question ”How much data from the Target domain (i.e. the new merchant client) is needed before FEDA with existing historical data is no longer better than simply using that Target data for training?”. The AUC and pAUC(0.05) of the previous tests are shown in Table 4.4.

Test	AUC	pAUC(0.05)
B1m	91.53%	1.57%
B3m	93.73%	2.89%
B6m	94.29%	3.13%
AB1m	93.14%	2.77%
AB3m	92.90%	2.36%
AB6m	94.54%	2.93%

Table 4.4: Performance metrics for tests using multiple amounts of Target data.

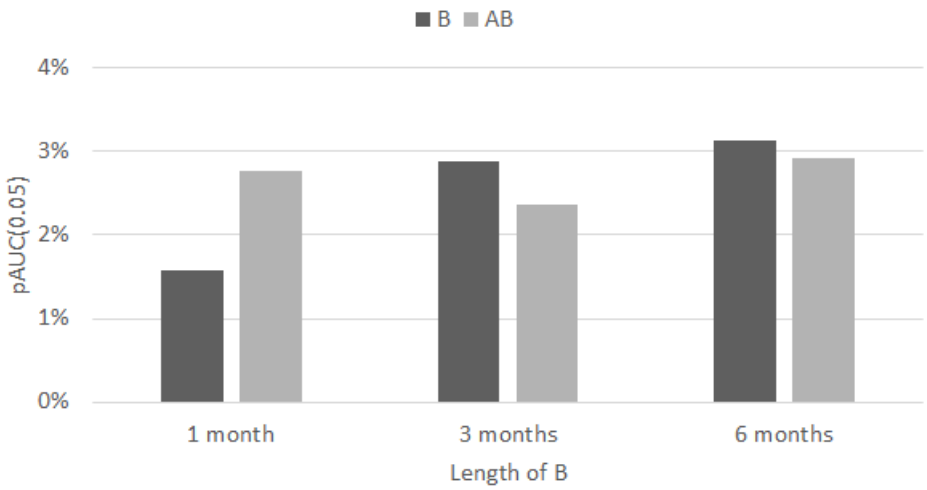

Figure 4.7: pAUC(0.05) for tests from Table 4.4.

The results from test **B1m**, **B3m** and **B6m** are as anticipated, lower but gradually approaching the performance values of Baseline **JustB**, which contained approximately 8 months of training data. Test **AB3m** is somewhat surprising, as both AUC and pAUC(0.05) are lower than test **AB1m** when precisely the opposite was expected. Nevertheless, these results suggest that using existing data and FEDA for at least one month can lead to better results than using only that month of merchant data for model training. On the other hand, comparing tests **B3m** and **AB3m**, they hint at the possibility that 3 months of data of a particular merchant is enough to achieve better performance than resorting to existing data and using FEDA.

4.3.1.3 Discussion

The preliminary results suggest that the implemented DA algorithm has potential to improve results by using existing historical data for new merchants with small amounts of transactions data. Specifically, results hint that one month of data of the Target domain is enough to achieve

a better performance with FEDA. However, as more transactions of the new merchant become available, there is a point where the merchant achieves enough data to actually provide better results than resorting to a vast amount of transactions from other merchants and applying DA.

Additional experiments were carried out in the Second Semester to better understand how FEDA could improve fraud detection. Furthermore, a new DA algorithm was implemented (CORAL), and an unsupervised outlier detection algorithm (LOF) was applied using existing frameworks. The corresponding experiments and results are presented in the following section 4.3.2.

4.3.2 Second Semester Results

This section contains the main experiments completed during the Second Semester and the corresponding results.

Starting, we repeat and expand the experiments from the First Semester, using dataset A as Source domain and dataset B as Target domain, and considering different time windows for dataset B from 1 week to 6 months. Following, a new scenario is simulated with dataset B being separated into the different sub-merchants it is composed of. Specifically, the merchant with most fraud is used as Target and all remaining merchants as the Source domain. In both experiments we apply the FEDA and LOF algorithms. Due to time constraints, the CORAL algorithm was not applied to these datasets.

The next experiments use dataset C split into several stores, similar to the previous scenario for dataset B. Once again, the store with most fraud is used as Target domain and all remaining as the Source domain. For these experiments, all algorithms (FEDA, CORAL and LOF) were tested.

Finally, dataset D is explored by separating all Card Present (CP) (Source domain) from Card Not Present (CNP) (Target domain) transactions and again testing all algorithms. A CP transaction takes place when a card and the card owner are present when making the payment. On the other hand, a CNP occurs when the payment is completed by providing card information over a remote channel such as the internet, by phone or over mail. A plausible reason to separate transactions by card presence could be a merchant with physical stores only, thus with high amounts of CP transactions, wanting to expand business to the eCommerce space. As such, CP transactions were considered as Source domain, and CNP as Target domain.

In all experiments, unless stated otherwise, ”training” implies using the first 70% of the corresponding dataset and ”testing” implies the last 30%. Another crucial point is that experiments where a stochastic component exists, such as in training models, presented results are averages of ten runs.

4.3.2.1 Testing FEDA on datasets A and B

For this test we apply the FEDA algorithm to datasets A and B, used as Source and Target domains, respectively. In this scenario, dataset A acts as existing historical data from several merchants and dataset B simulates a new merchant with different amounts of available data, aiming to expand and confirm the results obtained in the First Semester and reported on section 4.3.1. Since only up to 6 months of data from B are used for model training, instead of the typical 70/30 split for training/testing, dataset B is split 40/60, leaving 9 months for testing. This decision was made also to improve the number of fraudulent instances in the testing set, thus increasing the reliability of the results. Additionally, when training with A, the complete dataset is used, since it belongs to the Source domain and it is not used for testing. Another important aspect is that dataset A has a subset of features from B, thus, all experiments use only those common features.

Tables 4.5, 4.6 and 4.7 show the evaluation metrics for those experiments, for maximum FPR of 1%. The first column, **'Length of Target (B)'**, displays the amount of data from the Target domain B used in each test, starting at 1 week of data up until to 6 months. The second column **'Train'** indicates which dataset was used for model training: **JustB** means only data from dataset B was used; **DA(A+B)** indicates data from A and B, and also applying the DA algorithm FEDA; finally, **A+B** is similar to the previous case, where data from A and B is used for training but this time no DA algorithm is applied. Table 4.8 contains the evaluation results of using just dataset A for model training. Figures 4.8, 4.9, 4.10 and , 4.11 display a visual representation of the performance metrics from the previous Tables.

In all experiments data from B, the Target domain, was used for testing.

Length of Target (B)	Train	Recall$	Recall	AUC	pAUC(0.05)
1 week	JustB	2.20%	5.75%	87.40%	0.90%
2 weeks		10.23%	21.18%	87.31%	1.65%
3 weeks		29.68%	26.63%	90.91%	2.35%
1 month		40.03%	39.18%	90.84%	2.63%
2 months		23.92%	36.16%	91.36%	2.57%
3 months		36.54%	38.73%	92.26%	2.79%
6 months		42.55%	38.58%	93.23%	2.86%

Table 4.5: Performance metrics for model training and testing with data from dataset B, for maximum FPR of 1%.

Length of Target (B)	Train	Recall$	Recall	AUC	pAUC(0.05)
1 week	DA(A+B)	5.61%	8.02%	88.14%	1.09%
2 weeks		8.67%	13.46%	90.25%	1.85%
3 weeks		16.81%	25.42%	89.62%	1.92%
1 month		6.94%	16.64%	90.74%	1.69%
2 months		23.44%	27.84%	91.92%	2.47%
3 months		45.39%	40.54%	92.83%	2.87%
6 months		52.58%	43.27%	93.58%	2.95%

Table 4.6: Performance metrics for model training with data from datasets A and B, using FEDA algorithm, and tested with data from B, for maximum FPR of 1%.

Length of Target (B)	Train	Recall$	Recall	AUC	pAUC(0.05)
1 week	A+B	51.75%	19.67%	79.41%	1.37%
2 weeks		37.55%	16.94%	81.43%	1.28%
3 weeks		31.65%	16.49%	85.88%	1.41%
1 month		35.66%	16.79%	86.85%	1.30%
2 months		70.26%	30.26%	88.66%	1.85%
3 months		36.99%	24.05%	90.77%	2.06%
6 months		61.05%	26.78%	92.41%	2.27%

Table 4.7: Performance metrics for model training with data from datasets A and B, and tested with data from B, for maximum FPR of 1%.

Train	Recall$	Recall	AUC	pAUC(0.05)
JustA	51.53%	17.40%	85.05%	1.55%

Table 4.8: Performance metrics for model training with data from dataset A, and tested with data from B, for maximum FPR of 1%.

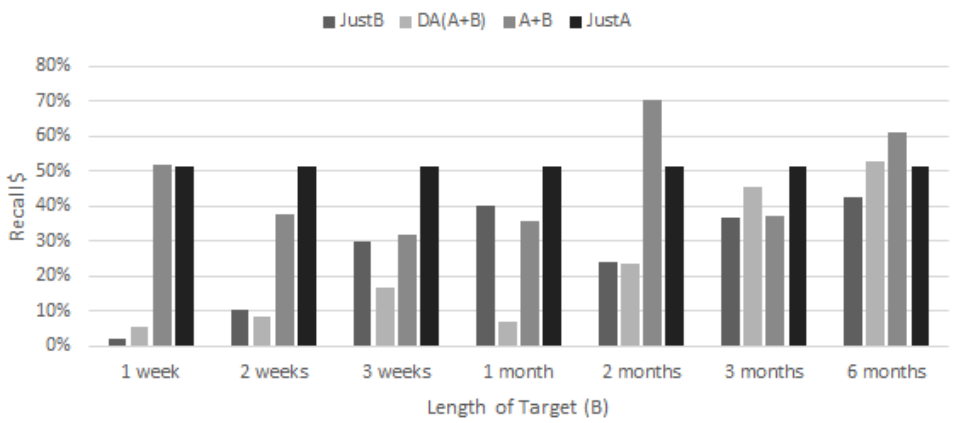

Figure 4.8: Money-Recall for all Train options, for maximum FPR of 1%.

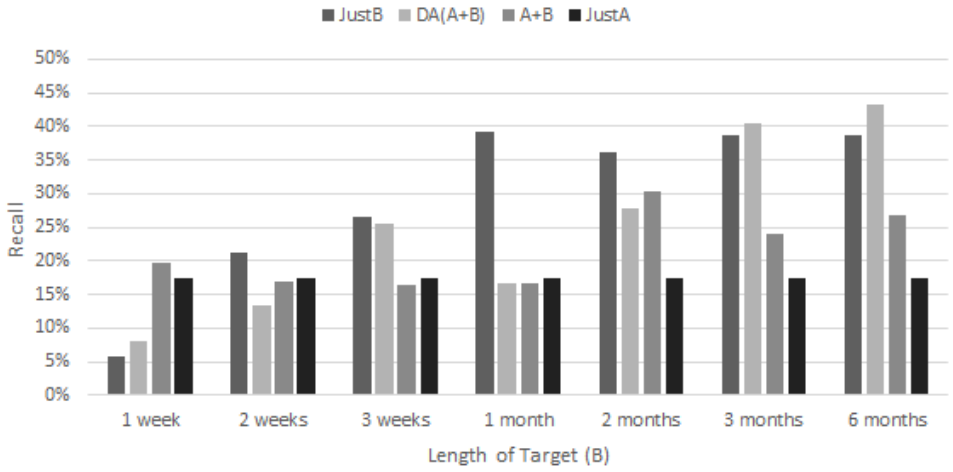

Figure 4.9: Recall for all Train options, for maximum FPR of 1%.

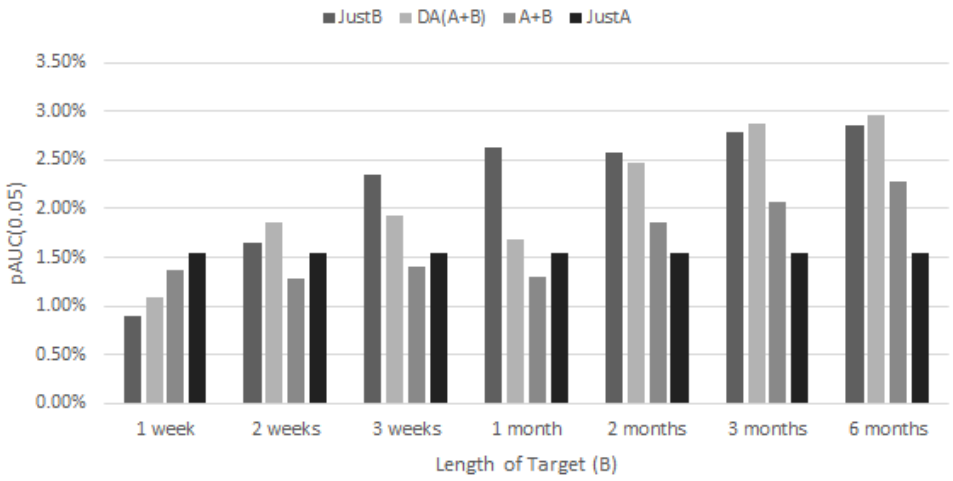

Figure 4.10: pAUC(0.05) for all Train options, for maximum FPR of 1%.

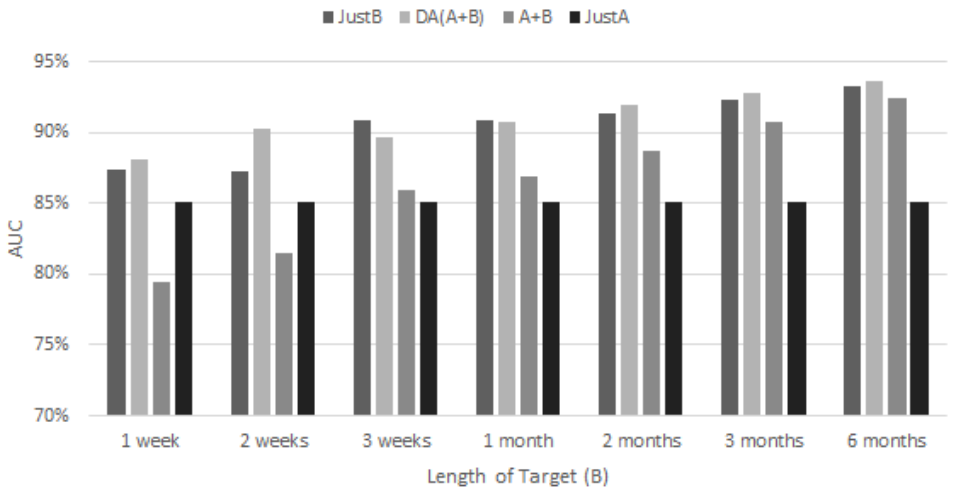

Figure 4.11: AUC for all Train options, for maximum FPR of 1%.

Looking at pAUC(0.05), the best option for the first week is using just data from A for training. On the second week, using DA with both datasets A and B seems to yield the best results. In the timeframe between the third week and the second month, the best performance is obtained using only data from the Target domain B. Finally, between three and six months, using DA provides the best performance in all metrics except for Money-Recall.

These results are quite inconclusive and do not provide a definite answer. On the hand, analyzing Money-Recall (column Recall$), training with just dataset A (Table 4.8) or with A and B without DA (Table 4.7) obtains the best performance for all time windows. Although not the best metric to compare performance, it should always be considered, as it effectively represents money that could potentially be saved.

An important aspect to notice is that when training with data from A, Money-Recall seems to be higher when analyzed proportionally to Recall. This is caused by transactions from A having an average amount higher than transactions from B, which also happens for fraudulent transactions, essentially making the trained model to learn to look for higher value transactions.

Taking into account metric Money-Recall, applying DA does not seem to provide additional

benefits rather than just training models on historical data and Target data as it becomes available, although further experimentation is needed.

4.3.2.2 Testing FEDA on datasets A and B, with feature union

This experiment follows the previous scenario, presented in 4.3.2.1, but instead of using the common features between both datasets, we use the feature union. What this means is that on dataset A we introduce zeros for all features that previously only existed on B.

The following Tables 4.9, 4.10 and 4.11 show the evaluation metrics for maximum FPR of 1%. Figures 4.12, 4.13, 4.14 and , 4.15 display a visual representation of the performance metrics from those Tables.

Length of Target (B)	Train	Recall$	Recall	AUC	pAUC(0.05)
1 week	JustB	48.15%	51.13%	94.66%	2.98%
2 weeks		46.50%	50.38%	94.57%	2.95%
3 weeks		61.26%	63.39%	96.01%	3.42%
1 month		61.65%	65.36%	96.18%	3.50%
2 months		69.20%	67.93%	97.10%	3.74%
3 months		74.29%	72.01%	97.73%	3.93%
6 months		67.19%	57.94%	97.82%	3.75%

Table 4.9: Performance metrics for model training and testing with data from dataset B, with feature union, for maximum FPR of 1%.

Length of Target (B)	Train	Recall$	Recall	AUC	pAUC(0.05)
1 week	DA(A+B)	21.89%	17.55%	88.96%	1.65%
2 weeks		32.60%	36.31%	93.84%	2.30%
3 weeks		43.16%	46.44%	95.57%	2.89%
1 month		59.44%	59.46%	95.94%	3.28%
2 months		60.30%	61.42%	96.27%	3.36%
3 months		74.10%	68.23%	97.70%	3.82%
6 months		62.84%	56.13%	97.47%	3.66%

Table 4.10: Performance metrics for model training with data from datasets A and B, using FEDA algorithm with feature union, and tested with data from B, for maximum FPR of 1%.

Length of Target (B)	Train	Recall$	Recall	AUC	pAUC(0.05)
1 week	A+B	68.88%	20.88%	82.43%	1.81%
2 weeks		68.49%	35.85%	93.72%	2.47%
3 weeks		65.95%	45.84%	94.69%	2.85%
1 month		58.79%	52.34%	94.79%	3.05%
2 months		60.43%	62.63%	95.97%	3.51%
3 months		79.01%	70.20%	97.46%	3.82%
6 months		80.53%	65.96%	97.48%	3.77%

Table 4.11: Performance metrics for model training with data from datasets A and B, with feature union, and tested with data from B, for maximum FPR of 1%.

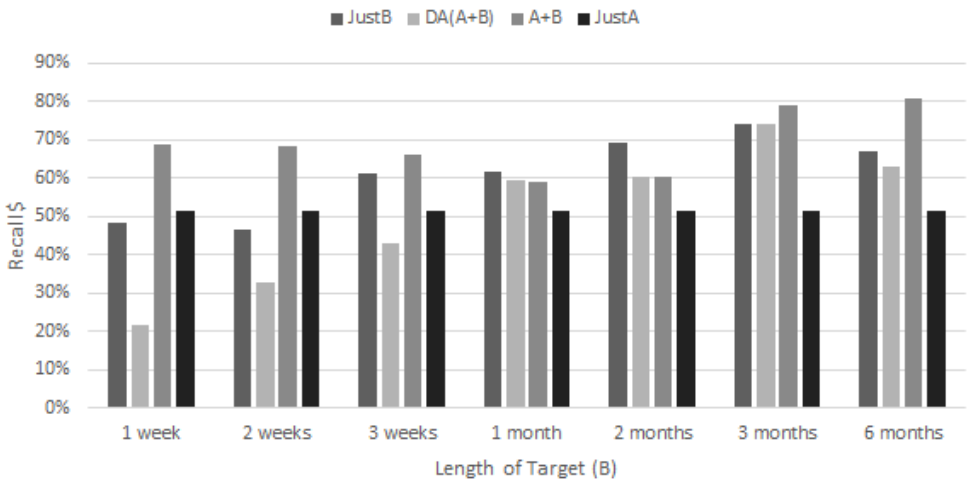

Figure 4.12: Money-Recall for all Train options, for maximum FPR of 1% and feature union.

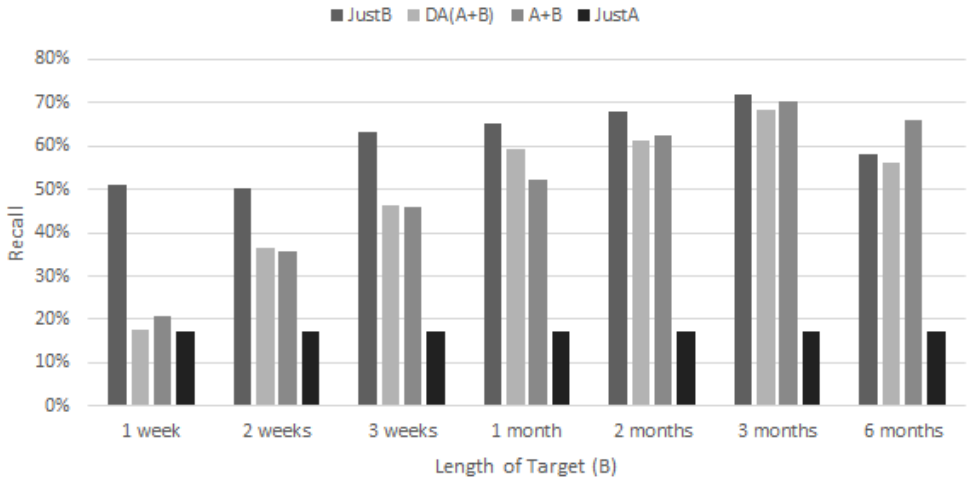

Figure 4.13: Recall for all Train options, for maximum FPR of 1% and using feature union.

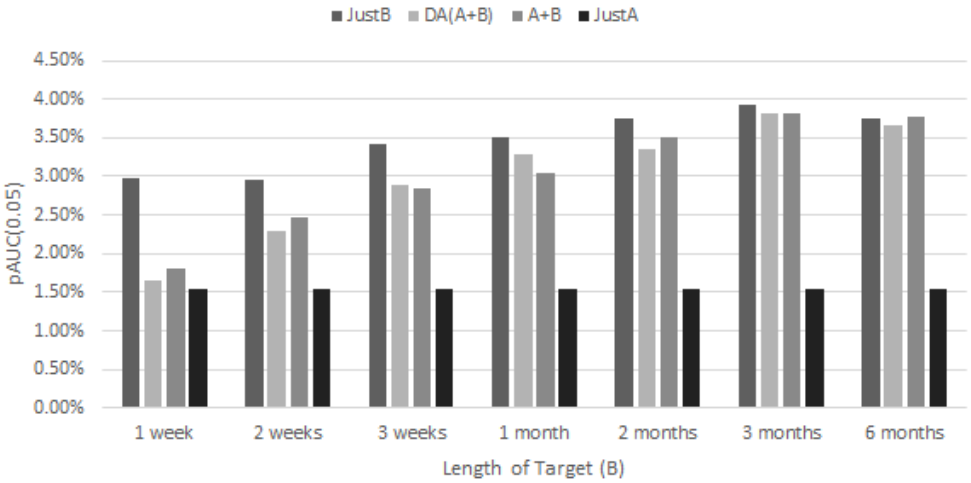

Figure 4.14: pAUC(0.05) for all Train options, for maximum FPR of 1% and using feature union.

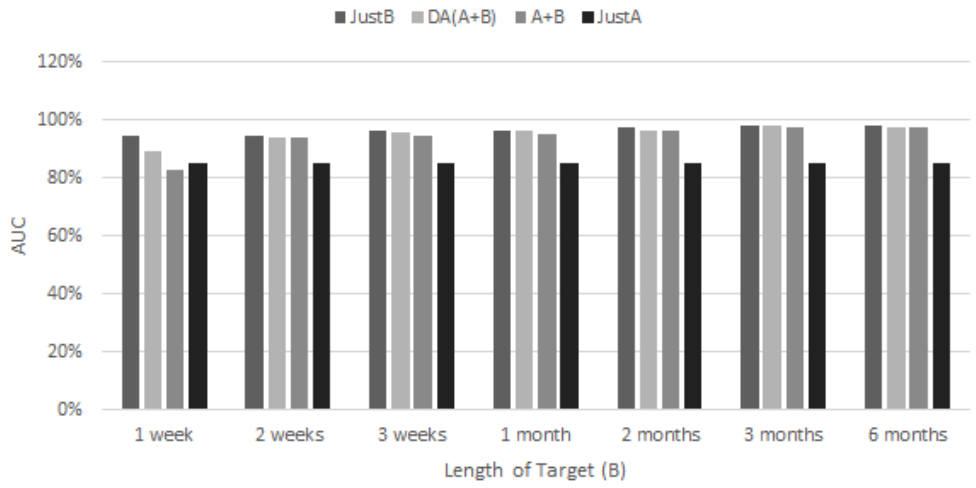

Figure 4.15: AUC for all Train options, for maximum FPR of 1% and using feature union.

In this scenario, results show DA does not bring additional benefit, since for all time windows using only data from B (Table 4.9) or data from A and B without DA (Table 4.11) better results are achieved for all metrics.

4.3.2.3 Splitting dataset B into sub-merchants

After the previous experiments, presented in 4.3.2.1 and 4.3.2.2, it was hypothesized that datasets A and B are from two very distinct domains, essentially making DA methods to fail. Consequently, this experiment aimed to split dataset B into several sub-merchants in order to generate closer Source and Target domains.

Since dataset B contains payment data from several different merchants, it was decided to split them according to the total number of transactions. In this experiment, the merchant with most transactions, referred to as MMT, is considered as the Target domain and all remaining merchants, designated as AeMMT, as the Source domain. To achieve this, dataset B is first split with the common 70/30 percentage for training/testing, and only then, for each set, the merchant with most transaction is separated from the remaining merchants, ensuring the temporal order between training and testing.

Table 4.12 shows the obtained results for each test for maximum FPR of 1%, where column **'Train'** indicates what data was used for training ("All" represents all merchants), and the **'Test'** column is omitted since all models are tested on the Target data, MMT. For the test corresponding to the last row, DA(All), the FEDA algorithm is applied. Since pAUC(0.05) provides a good representation of the performance of each model, Figure 4.16 show this metric for each test of Table 4.12.

Train	Recall$	Recall	Precision	AUC	pAUC(0.05)
MMT	4.66%	4.94%	1.52%	83.58%	0.36%
AeMMT	49.43%	35.80%	10.03%	91.70%	2.69%
All	26.42%	33.33%	9.75%	90.28%	2.64%
DA(All)	5.62%	7.41%	2.35%	86.96%	1.25%

Table 4.12: Performance metrics for model training with dataset B split into sub-merchants using MMT as Target and AeMMT as Source.

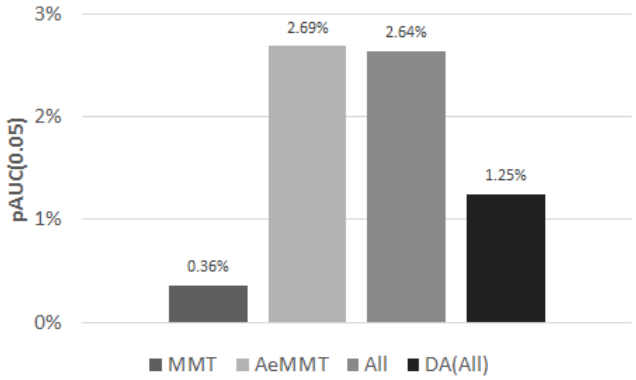

Figure 4.16: pAUC(0.05) of tests from Table 4.12

Results are somewhat surprising since the theoretical best-case, training with MMT, yields the worst performance. Another surprising fact is that using AeMMT for training, which contains no data from the Target domain, provides the best results for all metrics. Finally, using all merchants in the train set, the performance is significantly inferior to the previous case, AeMMT. Applying DA, namely the FEDA algorithm, not only does it offer no performance boost but also massively deteriorates it. This outcome could be explained by the low amount of fraudulent transactions for MMT (only 230 of a total 81944 transactions, or 0.28%), which might be too few for training as well as for testing.

For this reason, the experiment was repeated but, this time, using for Target domain the merchant with most fraudulent transactions, referred as MMF (with 1337 fraud transactions in 23713 total, or 5.64% fraud rate), and all other merchants as the Source domain, designated as AeMMF. As before, the testing set for every scenario is MMF, the Target domain. Table 4.13 contains the performance results for this experiment and Figure 4.17 shows the pAUC(0.05) for each test.

Train	Recall$	Recall	Precision	AUC	pAUC(0.05)
MMF	56.04%	42.04%	48.89%	90.90%	2.53%
AeMMF	29.79%	9.55%	21.74%	85.07%	1.39%
All	54.52%	40.76%	47.76%	90.99%	2.38%
DA(All)	42.72%	35.67%	44.80%	91.52%	2.23%

Table 4.13: Performance metrics for model training with dataset B split into sub-merchants using MMF as Target and AeMMF as Source.

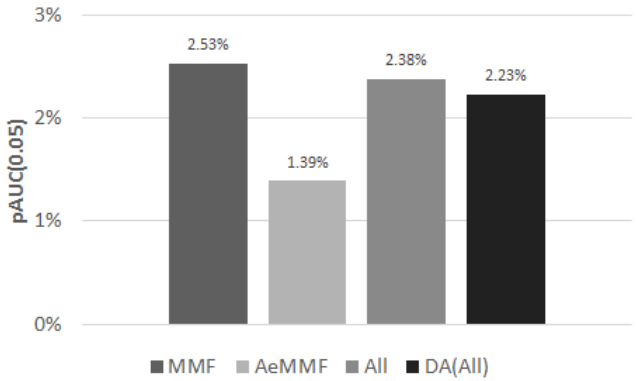

Figure 4.17: pAUC(0.05) of tests from Table 4.13

Results for the first two rows are as expected, with the best performance achieved when using the Target domain for training, MMF, and worst performance when training with a different domain, AeMMF. When using all merchants ('All' row), performance approximates the best-case but, as before, using the FEDA algorithm decreases performance for all metrics except for AUC metric ('DA(All)' row).

It is interesting that FEDA provides the best AUC value over all other training options, meaning that overall performance is improved considering all values of FPR up to 100%. However, since limiting FPR to a low value (usually 1% or 5%) in fraud detection systems is of vital importance, as explained in 3.3.6, that particular result is not significant and, in fact, pAUC(0.05) for FEDA is actually only the third highest value.

In the next section, a new approach distinct from DA is studied, as the unsupervised algorithm presented in 2.4 is applied to dataset B.

4.3.2.4 Applying LOF to dataset B

This section presents a new approach, alternative to DA, that could potentially solve the current eCommerce use case. Introduced in section 2.4, LOF is an unsupervised learning algorithm for outlier detection. In this context, fraudulent transactions can be seen as deviations from legitimate payment patterns, making them detectable by outlier detection algorithms.

Opposite to DA, this approach does not need historical data since it only uses data from the Target domain. Additionally, Target data can be unlabeled, a huge benefit for small merchants for which labeling costs can be prohibitive. On the other hand, enough data on the Target domain should exist in order to create a good representation of the dataset.

LOF was tested on three subsets of dataset B, with all 151 features, each containing the last 100 000, 50 000, 20 000 and 10 000 instances of the complete dataset. The different amount of transaction was chosen to evaluate how LOF performance would evolve according to the number of transactions considered.

Before applying LOF, the number of feature dimensions is reduced with Principal Component Analysis (PCA), a technique that converts a set of possibly correlated features into a smaller subset of uncorrelated features, referred to as principal components. The reason to use PCA was based on empirical observation, namely, the fact that, without PCA, LOF performance was atrocious. After trying PCA, results were significantly improved and, on top of that, computational time required to apply LOF was decreased.

Total instances	PCA dimensions	LOF parameter k	Recall$	Recall	Precision
100000	100	256	64.46%	52.93%	39.43%
50000	50	128	74.31%	48.73%	26.09%
20000	50	32	29.93%	15.85%	6.16%
10000	25	32	13.23%	9.84%	5.77%

Table 4.14: Performance metrics and optimal parameters for LOF applied to dataset B, for maximum FPR 1%.

Recall$	Recall	Precision
84.44%	77.20%	41.73%

Table 4.15: Performance metrics for training and testing with dataset B, using all 151 features and maximum FPR of 1%.

Table 4.14 contains the results for dataset B applying LOF to different amounts of transactions, showing the optimal PCA and k parameters as well as performance metrics Money-Recall, Recall and Precision for maximum FPR of 1%. Table 4.15 shows the performance results for training and testing with dataset B using all 151 features, acting as a baseline to compare LOF results. It should be noted that AUC and pAUC is not shown since LOF generates arbitrary scores which are then normalized between 1 and 1000, similar to the threshold value described in 3.3.5, however, even after normalization, given the erratic sequence of LOF scores (low and very high values) AUC and pAUC might not be an accurate performance metric.

LOF exhibits very good performance when comparing with the baseline shown in Table 4.15 for 100 000 and 50 000 data instances. Although it presents a lower Money-Recall and Recall it should be noted that LOF does not need labels on the data, but, on the other hand, there is a significant performance drop when decreasing the number of available instances, which might be a problem for new merchants with low number of transactions.

In summary, LOF shows great potential when dealing with unlabeled data, however, a considerable number of transactions (over 20 000) is needed to achieve good performance.

4.3.2.5 Splitting dataset C into sub-merchants

As mentioned before, dataset C is composed of multiple merchants operating in the same type of business, consequently, data from these merchants is considered from closer domains, opposed to what happens in dataset B. Similarly to experiments reported in section 4.3.2.3, dataset C is also split between the merchant with most transactions (MMT) which, coincidentally, contains the most fraud, and all other merchants (AeMMT). As before, MMT is considered as the Target domain and AeMMT the Source domain, then, each DA algorithm is applied, namely FEDA and CORAL and, for all tests the training set is, again, MMT, the Target domain.

Table 4.16 shows the performance results for all tests, training models with data from Source, Target or both ('All'), and applying FEDA and CORAL algorithms, for maximum FPR of 1%. Since pAUC(0.05) provides a good representation for model performance, Figure 4.18 shows this metric for tests from Table 4.16.

Train	Recall$	Recall	Precision	AUC	pAUC(0.05)
MMT	18.26%	7.41%	11.83%	77.93%	0.76%
AeMMT	23.27%	11.55%	17.38%	77.97%	0.83%
All	27.20%	12.74%	18.67%	79.67%	0.99%
FEDA(All)	25.61%	10.47%	15.92%	79.39%	0.93%
CORAL(AeMMT)	19.49%	11.42%	17.17%	79.29%	0.96%

Table 4.16: Performance metrics for model training with dataset C split into sub-merchants using MMT as Target and AeMMT as Source, for maximum FPR of 1%.

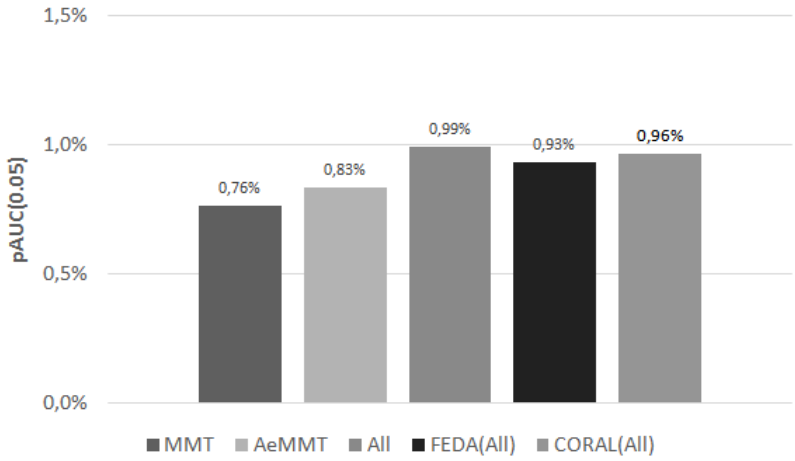

Figure 4.18: pAUC(0.05) of tests from Table 4.16

Even using Target and Source from closer domains than dataset B, results are similar in the sense that DA algorithms do not improve performance of other alternatives. In this case, the best option is to train a model with data from all merchants ('All' row), which exhibits the best performance for all metrics.

An important observation about CORAL is that, after applying this algorithm, values from binary or integer features are transformed to float values, which might deteriorate data quality and have a severe impact on model performance. A possible solution to this issue could be applying this algorithm only to features containing float values, however, this improvement was not implemented due to time constraints.

4.3.2.6 Applying LOF to dataset C

As for dataset B, we apply LOF to dataset C using all 151 features and for three different sets containing the last 100 000, 50 000 and 10 000 instances. Just like before, PCA is also applied in order to reduce data dimensionality, and several LOF parameter k are tested. Table 4.17 contains the results of LOF applied to dataset C testing different amounts of data, also showing the optimal PCA and k parameters as well as metrics Recall, Money-Recall and Precision for maximum FPR of 1%.

Total instances	PCA dimensions	LOF parameter k	Recall$	Recall	Precision
100000	75	32	5.37%	4.04%	7.13%
50000	75	64	5.09%	3.70%	7.13%
10000	75	16	4.06%	4.94%	11.82%

Table 4.17: Performance metrics and optimal parameters for LOF applied to dataset C, for maximum FPR 1%.

Obtained results show poor LOF performance when observing LOF performance for dataset B, however, comparing with the best result from dataset C from Table 4.16, LOF achieves almost 40% of the best Recall value (4.94% against 12.74%) and over 63% of the best Precision value (11.82% against 18.67%) with only 10 000 instances but, on the other hand, less than 20% of the best Money-Recall (5.37% against 27.20%) with 100 000 instances.

Overall, comparing with the supervised training/testing scenario shown in Table 4.16 LOF achieves relatively good performance considering it uses only unlabeled data from the Target domain. Furthermore, the best Recall and Precision values for this algorithm are achieved for a dataset sample of only 10 000 instances.

4.3.2.7 Splitting dataset D into CP/CNP transactions

Following the same idea applied to datasets B and C, dataset D is split into CP transactions (the Source domain) and CNP transactions (the Target domain). The reason for this was, given the data format, it would be impossible to identify groups of merchants and split them like previous experiments. As such, CP and CNP transactions were considered suitable for Source and Target domains, respectively, and, as stated before, could simulate the plausible scenario where a merchant with physical stores decides to expand to the eCommerce space.

Train	Recall$	Recall	Precision
CNP	41.03%	28.11%	11.46%
CP	35.03%	20.38%	8.61%
All	37.67%	25.85%	10.73%
FEDA(All)	34.85%	25.85%	10.75%
CORAL(CP)	23.92%	18.02%	8.01%

Table 4.18: Performance metrics for model training with dataset D split into CP and CNP transactions, used as Source and Target domain, respectively, for maximum FPR of 1%.

Total instances	Recall$	Recall	Precision
23 336	29.88%	8.26%	3.14%

Table 4.19: Performance metrics and optimal parameters for LOF applied to dataset D, CNP transactions only, for maximum FPR 1%.

Table 4.18 contains the performance results for tests using training data from Target, Source or both domains ('All') and applying FEDA and CORAL algorithms, for maximum FPR of 1%. In all tests, like previous experiments, models were tested on the Target data, that is, CNP transaction instances. Table 4.19 presents the results for LOF applied to Target data, containing 23 336 CNP transaction instances.

As in previous experiments, DA algorithms do not provide increased model performance and training a model with Target data (CNP) achieves the best results in all metrics, as expected. If there are no labels on the Target data, then the only available options are training with Source data (CP), applying CORAL to Source data (CORAL(CP)), or applying LOF. Of those, the best option is to use only Source data, with no transformation. Furthermore, if no historical Source data is available, LOF provides considerably good results: event though Recall and Precision are significantly lower than the best case (CNP row in Table 4.18), LOF manages to achieve 29.88% Money-recall, almost 3/4 of the best case Money-Recall (41.03%).

4.3.2.8 Sanity check for FEDA and CORAL algorithms

After observing both DA algorithms did not provide any performance improvement, a new experiment was designed in order to apply a sanity check to both algorithms. The experiment consisted in using data from dataset C, which is the most uniform in terms of merchant similarity (all merchants belonging to this dataset operate in the same business), and randomly selecting 50% of the training set transactions as the Target domain and the remaining 50% as the Source domain. This method ensures data distribution is roughly the same between Target and Source and, by applying DA to different combinations of data, performance should be stable.

FEDA

Since FEDA algorithm transforms data such that each instance contain one third of the features with zeros, when building a Random Forest, all N features are considered for selecting the best node split, instead of the typical $sqrt(N)$. For each test, the number of transactions was kept constant in order to fairly compare results using the same amount of data. With the goal of evaluating how FEDA affects model performance, a total of seven tests were completed, with the following data configurations:

- Training with 100% transactions from Source, applying no DA, serving as baseline
- Training with 100% transactions from Source, applying FEDA
- Training with 80% transactions from Source and 20% from Target, applying FEDA
- Training with 60% transactions from Source and 40% from Target, applying FEDA
- Training with 40% transactions from Source and 60% from Target, applying FEDA
- Training with 20% transactions from Source and 80% from Target, applying FEDA
- Training with 100% transactions from Target, applying FEDA

Considering data from Source and Target are identical, the expected result was similar performance for all tests. Table 4.20 contains the results using Target data for testing. For conciseness, Source is denoted as S and Target as T.

Train	DA algorithm	Recall$	Recall	Precision	AUC	pAUC(0.05)
100% S	-	16.37%	11.28%	18.41%	79.94%	0.95%
100% S	FEDA	15.00%	9.87%	17.15%	78.42%	0.85%
80% S + 20% T	FEDA	15.60%	10.11%	16.81%	79.19%	0.87%
60% S + 40% T	FEDA	15.22%	9.95%	16.60%	79.22%	0.87%
40% S + 60% T	FEDA	15.08%	9.82%	16.42%	79.61%	0.87%
20% S + 80% T	FEDA	14.61%	10.75%	17.67%	79.49%	0.92%
100% T	FEDA	16.20%	12.12%	19.51%	80.36%	1.02%

Table 4.20: Performance metrics for FEDA sanity check for dataset C, testing with Target data, for maximum FPR of 1%. S represents Source data and T represents Target data.

Although we expected similar performance for all tests, results show otherwise, with FEDA exhibiting a performance drop when comparing with the baseline (first row). Looking closer at the obtained metrics for FEDA algorithm, despite Money-Recall, Recall and Precision values oscillating slightly, we observe a steady increase in AUC and pAUC as the percentage of Target data increases as well. The explanation for this is that testing data is treated as from the Target domain and applied the corresponding transformation, described in 2.2. As such, when a higher percentage of Source data exists, there is a greater probability of a Source-specific feature being selected as the best split when building the Random Forest model. What happens next is, during testing, that particular feature will contain no information since all testing instances are from the Target domain.

Before running these tests we did not anticipate this problem and incorrectly hypothesized that performance should be stable. However, for the last row (100%), the issue does not occur since all training data is from Target, and performance metrics are somewhat similar to the baseline (first row), although, surprisingly, slightly higher except for Money-Recall.

CORAL

For CORAL algorithm, the previous problem does not apply since no additional features are introduced. As explained in 2.3, CORAL observes Target data and tries to transform Source such that data distribution from both domains are as close as possible. Like before, multiple tests were designed in order to evaluate how CORAL would affect data with the same distribution. The tests considered were as follows:

- Training with Source data, applying no DA
- Training with Source data, transformed using CORAL and 25% of Target data
- Training with Source data, transformed using CORAL and 50% of Target data
- Training with Source data, transformed using CORAL and 75% of Target data
- Training with Source data, transformed using CORAL and 100% of Target data

Train	DA algorithm	Recall$	Recall	Precision	AUC	pAUC(0.05)
S	-	19.10%	12.82%	20.46%	80.42%	1.06%
S	CORAL (25% T)	12.97%	10.97%	19.54%	76.21%	0.82%
S	CORAL (50% T)	12.80%	10.69%	18.17%	76.71%	0.79%
S	CORAL (75% T)	11.09%	9.43%	16.37%	75.47%	0.81%
S	CORAL (100% T)	11.52%	9.87%	17.17%	74.78%	0.80%

Table 4.21: Performance metrics for CORAL sanity check for dataset C, for maximum FPR of 1%.
S represents Source data and T represents Target data.

Table 4.21 shows the performance for CORAL for the described tests. Although results were expected to be similar, applying the algorithm causes a significant performance drop in all metrics, just like in all previous experiments. As mentioned before, CORAL transforms data such that binary values become float values, which can have a severe impact in model performance. On top of that, most of the datasets used in this work are composed by binary values, aggravating this problem. As suggested in 4.3.2.5, a possible solution would be filtering all binary features from Source and Target and applying the algorithm to all remaining features, finally joining the binary features again. This improvement was not tested and, consequently, left for future work.

Chapter 5

Conclusion

In this Chapter, a summary overview of the tasks completed during this work and the main conclusion drawn from it are presented. Closing the Chapter, we discuss possible steps for future work.

5.1 Summary

This work focuses on the fraud detection for the eCommerce use case, where losses due to fraudulent payments is steadily increasing every year. Although fraud affects the whole spectrum of commerce and financial institutions, eCommerce merchants are the most penalized, since fraud is most prevalent in this space and legitimate card owner reimbursement is the merchant's responsibility. Typical approaches to fraud detection include Machine Learning models trained with transactions data that learn common fraud patterns. For some eCommerce merchants this might pose a problem if they contain few data to properly train a model.

This use case is the main motivation of this work, which aims to solve the problem by using payment data from other merchants (other domains). In particular, we implement two Domain Adaptation algorithms that harness the information contents of existing merchants data with the goal of improving fraud detection for new merchants. Additionally, an Unsupervised Learning outlier detection method was tested, in order to explore the possibility of fraud detection in unlabeled data.

Although the initial plan was to explore the FEDA algorithm and implement it in Feedzai's Machine Learning Tools, experiments during the Second Semester revealed the algorithm did not improve performance. As such, a new DA algorithm, CORAL, was also tested, revealing no performance boost as well. Finally, an Unsupervised Learning algorithm, LOF, was applied to the same datasets, exhibiting good performance considering it uses no Source data or Target labels. A summary of all datasets and algorithms tested is shown in Table 5.1.

Dataset		Algorithm					
Source (S)	**Target (T)**	**FEDA**	**CORAL**	**LOF**	**S only**	**T only**	**S+T**
A	B	X	-	X	X	X	X
B(AeMMT)	B(MMT)	X	-	-	X	X	X
B(AeMMF)	B(MMF)	X	-	-	X	X	X
C(AeMMT)	C(MMT)	X	X	X	X	X	X
D(CP)	D(CNP)	X	X	X	X	X	X

Table 5.1: Multiple datasets and all tested algorithms.
S represents Source data and T represents Target data.

In all tests FEDA and CORAL cause performance to drop moderately, and the best option to maximize performance is to train with historical Source data, Target data, both, or apply LOF, subject to Source and Target data availability.

In order to explore the multiple scenarios regarding Source and Target data availability and assess the best decision to make in order to optimize fraud detection for new merchants (Target) with few data, we summarize some of the main conclusions of this work in Table 5.2, assuming a small amount of Target data is always available.

Source data available	**Target data labeled**	**Best option**
Yes	Yes	Train model with Source and Target data
Yes	No	Train model with Source data only OR apply LOF
No	Yes	Train model with Target data only OR apply LOF
No	No	Apply LOF

Table 5.2: Summary of the best options to maximize performance for new merchants.
This conclusion assumes there is always some amount of Target data available.

If historical data is available (Source domain) and Target data is labeled, optimal performance should be obtained by training a model with all (Source and Target) data. If Target data is unlabeled there are two option: either train a model with existing Source data or apply LOF to Target data. The amount of Target data here is crucial to decide which method to follow, although difficult to quantify a specific value. Another important factor is the different number of Target features: if Target data has a significantly higher number of features than Source data, applying LOF should yield better performance since Target instances contain much more information than Source instances.

If Source data is not available and Target data is labeled, a model should be trained with Target data or LOF applied, considering the previous conditions regarding amount and number of features of Target data. Finally, if no Source data and no Target labels are available, the only option is to apply LOF.

The main contributions of this works are the study of two Domain Adaptation algorithms applied to fraud detection in the eCommerce use case, with results showing these algorithms are not effective in this scenario when considering two Source and Target domains. Furthermore, we explore how an Unsupervised Learning outlier detection method performs when applied to payment data, effectively proving that fraudulent payments can be detected as outliers from common payment patterns.

5.2 Future Work

Although numerous scenarios with multiple datasets were extensively explored, we only considered two domains in all experiments, when, in reality, datasets were composed of many submerchants and, consequently, multiple domains. This scenario with numerous merchants should be explored with FEDA algorithm, which supports data from multiple domains. If successful, the algorithm should be optimized for scalability since the number of features when using FEDA increases linearly with the number of domains, making it impractical in a real scenario with possibly hundreds or even thousands of merchants. One possible optimization could be assigning a number to each domain and encoding that number in binary features while keeping only the general version (see 2.2) features. Using this method, it would be possible to consider 1024 distinct domains with only 10 additional features.

LOF proved to be a good solution for the presented use case, however, it was only tested in batch mode. As it stands, it needs to be implemented in Feedzai's Machine Learning Tools to be able to work in a production environment. Furthermore, performance issues need to be addressed since the algorithm has a quadratic time complexity, which might pose a problem considering the typical low latency requirements of transaction classification.

Bibliography

[1] Statista. http://www.statista.com/statistics/273018/number-of-internet-users-worldwide/, 2015. [Online; accessed 22nd January 2016].

[2] eMarketer. http://www.emarketer.com/Article/Worldwide-Ecommerce-Sales-Increase-Near1011039, 2014. [Online; accessed 19th January 2016].

[3] P. Startups. http://portugalstartups.com/2015/05/feedzai-raised-175m-in-series-b/, 2015. [Online; accessed 17th January 2016].

[4] L. Nexis, "True cost of fraud," 2014.

[5] Dalpay. https://www.dalpay.com/en/support/chargebacks.html, 2015. [Online; accessed 17th January 2016].

[6] L. Torrey and J. Shavlik, "Transfer learning," *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 2009.

[7] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. on Knowl. and Data Eng.*, vol. 22, pp. 1345–1359, Oct. 2010.

[8] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: Transfer learning from unlabeled data," in *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, (New York, NY, USA), pp. 759–766, ACM, 2007.

[9] B. Zadrozny, "Learning and evaluating classifiers under sample selection bias," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, (New York, NY, USA), pp. 114–, ACM, 2004.

[10] J. Huang, A. Smola, A. Gretton, K. Borgwardt, and B. Schölkopf, "Correcting sample selection bias by unlabeled data," in *Advances in Neural Information Processing Systems*, vol. 19, The MIT Press, Cambridge, MA, 2007. Pre-proceedings version.

[11] M. Sugiyama, S. Nakajima, H. Kashima, P. V. Buenau, and M. Kawanabe, "Direct importance estimation with model selection and its application to covariate shift adaptation," in *Advances in Neural Information Processing Systems 20* (J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, eds.), pp. 1433–1440, Curran Associates, Inc., 2008.

[12] H. Daumé III, "Frustratingly easy domain adaptation," in *Conference of the Association for Computational Linguistics (ACL)*, (Prague, Czech Republic), 2007.

[13] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. [13], pp. 2058–2065.

[14] "Numpy." http://www.numpy.org/, 2016. [Online; accessed 2nd May 2016].

[15] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, pp. 93–104, May 2000.

[16] P. Perner, ed., *Machine Learning and Data Mining in Pattern Recognition, 6th International Conference, MLDM 2009, Leipzig, Germany, July 23-25, 2009. Proceedings*, vol. 5632 of *Lecture Notes in Computer Science*, Springer, 2009.

[17] E. Altendorf, P. Brende, J. Daniel, and L. Lessard, "Fraud detection for online retail using random forests," tech. rep., Citeseer.

[18] C. Liu, Y. Chan, S. Kazmi, and H. Fu, "Financial fraud detection model based on random forest," *International Journal of Economics and Finance*, vol. 7, no. 7, pp. 178–188, 2015.

[19] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 3rd ed., 2005.

[20] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, Mar. 1986.

[21] E. Costa and A. Simões, *Inteligência Artificial, Fundamentos e Aplicações*. FCA, 2004.

[22] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[23] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, pp. 1–37, Dec. 2007.

[24] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.

[25] D. Wang, E. A. Rundensteiner, and R. T. Ellison, III, "Active complex event processing over event streams," *Proc. VLDB Endow.*, vol. 4, pp. 634–645, July 2011.

[26] Feedzai, "Fraud prevention api." http://dev.feedzai.com/rest-api/, 2016. [Online; accessed 15th January 2016].

[27] "Elki." http://elki.dbs.ifi.lmu.de/, 2016. [Online; accessed 27th May 2016].

Appendices

Appendix A

Work Planning

A.1 First Semester

Activities during the First Semester took place in Coimbra, at the Informatics Engineering Department and at Feedzai's office in Instituto Pedro Nunes. Since Feedzai's advisors were located in Lisbon, weekly online meetings where held in order to discuss the student's progress and define new goals. Developed work followed an exploratory research process with the goal of determining if Domain Adaptation could provide improved results in the presented context. The following tasks were planned:

- **Learn about payments and fraud** - this task provided valuable context for this work, particularly how payments are processed, the involved entities and fees as well as how does fraud occur and how merchants are affected by it.
- **Participation in Feedzai's Machine Learning Hackaton** - this activity was organized by Feedzai and consisted of a brief introduction to Machine Learning, followed by an implementation and application of a particular algorithm to a real dataset.
- **Read Feedzai's documentation** - this task consisted of understanding the overall Machine Learning methodology used at Feedzai, technical definitions and performance metrics used to compare results.
- **Learn Feedzai's Data Science tools** - learn how Feedzai's Data Science tools work. Tools include the Model builder, Simulator and PQL, described in section 4.2.
- **Define and calculate baseline results** - this task aimed to determine the results reported in section 4.3.1.1.
- **Implementation and testing of FEDA algorithm** - for this task the DA algorithm presented in section 2.2 was implemented and the corresponding results obtained.

A detailed schedule plan of the tasks completed for the 1st semester is shown in Figure A.1.

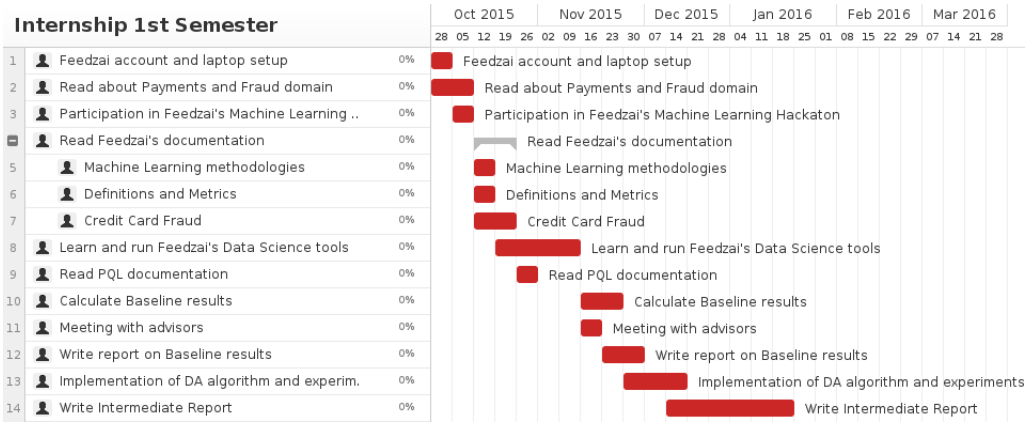

Figure A.1: Planning for the 1st Semester.

A.2 Second Semester

Work during the Second semester took place in Feedzai's office in Lisbon. Initial planning for the Second Semester aimed to validate the FEDA algorithm, extending it to a multi-merchant scenario with multiple source domains and one or more target domains, and finally implementing it with Feedzai's Machine Learning tools. After concluding the algorithm did not actually provide performance improvements, two new methods (CORAL and LOF) were explored and extensively tested with multiple datasets.

During this period, the student was integrated with Feedzai's Research Group, where daily meetings were held in order to review the previous day progress and the current day work plan. Furthermore, weekly presentations took place at the end of the week, aiming to present and discuss obtained results and planning work for the following week. A summary of the main tasks completed throughout the semester is described next:

- **Validation and tuning of implemented FEDA algorithm** - this task aimed to validate the current FEDA implementation with new datasets and other variations such as different features between source and target domains, and splitting datasets into sub merchants.
- **Implement CORAL algorithm** - after FEDA was implemented and observed it did not provide improved results, CORAL algorithm was explored and applied to multiple datasets.
- **Explore an Unsupervised Learning algorithm** - since both DA algorithms failed to perform, LOF was applied to all datasets, using the ELKI framework, as mentioned before.
- **Final Report** - although the final report was written continuously throughout the second semester, one month was allocated for its conclusion.

A detailed chart of the scheduled tasks is shown in Figure A.2.

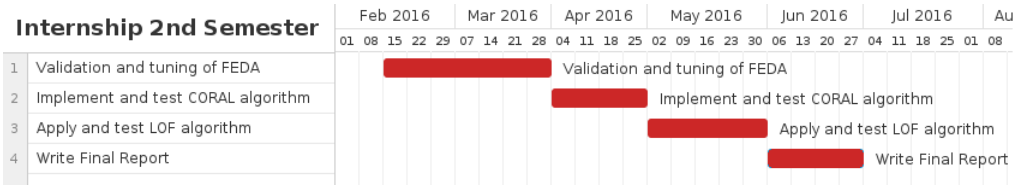

Figure A.2: Planning for the 2nd Semester.