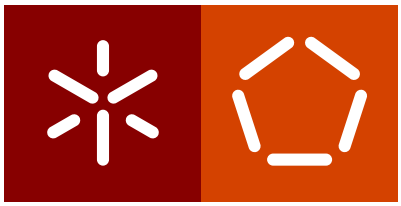
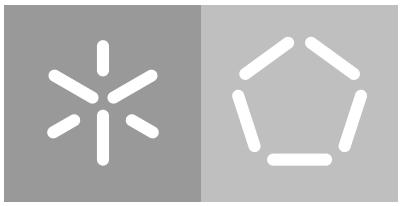


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Miguel Lobo Pinto Leite

Active Learning for Fraud Detection

November 2020



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Miguel Lobo Pinto Leite

Active Learning for Fraud Detection

Master dissertation
Integrated Master in Informatics Engineering

Dissertation supervised by
Paulo Azevedo (DI, Universidade do Minho)

November 2020

COPYRIGHT NOTICE

ACKNOWLEDGEMENTS

I want to express my gratitude for those who were part of or contributed to this work. My supervisor and mentor, Marco Sampaio, who managed our project remarkably, provided me with knowledge and guidance throughout the whole process and performed a pervasive review of my dissertation. My other mentor, Ricardo Barata, who brought exceptional work and ideas to our project and was always available to help me. And our past colleague, Ricardo Pacheco, for his contributions and how he revolutionized the project with high standards for how we organized our work. I learned a lot and received much friendship from them, but also from all the others who are part of Feedzai's Research department. I must thank the company as a whole, which made it possible for me to work at such an exciting and challenging project. And of course, my supervisor from the University, Professor Paulo Azevedo, who was always committed to keeping up with our work, providing his ideas, and ensuring I deliver a well-written document.

This work represents the end of my Masters. Therefore, I would appreciate offering my special thanks to the many who had an impact on my education. I would rather not say all the names, but I must emphasize my parents. They are the main reason for this to be possible by providing me with an education, teaching me to persist through challenges, and always guiding me in my decisions.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of Universidade do Minho.

Assinado por : **MIGUEL LOBO PINTO LEITE**
Num. de Identificação: BI15184822
Data: 2020.11.13 11:25:40 +0000

RESUMO

Um obstáculo comum em vários domínios no processo de preparação de um modelo de *Machine Learning* (ML) é a escassez de *labels* (i.e., etiquetas dos dados). Em aplicações reais, algures no processo de construção de um *dataset* existe um especialista a fazer anotação manual de cada instância dos dados para identificar a respetiva *label*. Dentro do domínio de deteção de fraude, que é normalmente tratado como um problema de ML supervisionado, a existência de analistas de fraude a reverem todas as transações que ocorrem representaria um nível de custos em recursos humanos inexequível. Isto leva a que apenas uma fração dos dados possam ser manualmente analisados. O sub-campo de ML conhecido como *Active Learning* (AL) surgiu em resposta a este problema. Em AL são implementados algoritmos que selecionam de forma eficiente quais as instâncias dos dados que devem ser analisadas de forma a otimizarem-se os custos de anotação dos dados. O objetivo principal deste processo é a criação de um modelo de previsão eficaz treinado com a menor quantidade de dados possível. Neste trabalho, apresentamos um estudo detalhado de diversas estratégias de AL em que realizamos experiências com dados de aplicações reais. Focamo-nos principalmente no cenário em que a anotação dos dados é iniciada a partir do primeiro dia de geração dos mesmos, não tendo à partida dados prévios para a construção de perfis dos utilizadores nem quaisquer *labels*. Apresentamos avaliações de novos algoritmos e configurações de AL, assim como métodos pré-existentes, através de múltiplas experiências. Estas experiências são realizadas num ambiente em *streaming* (tal como nos sistemas de produção em causa), em que as transações são processadas em tempo real.

Para além da escolha do algoritmo de AL existem outros parâmetros a definir na configuração geral. Realizamos estudos que nos permitem compreender quais os valores mais favoráveis de vários destes parâmetros, incluindo o impacto da escolha do método de pré-processamento de dados e do modelo de ML usado em avaliação.

A maioria dos algoritmos de AL existentes na literatura exigem um conjunto de dados já com *labels* que tenha elementos de todas as classes existentes (e.g., transações legítimas e fraudulentas). Dado que no domínio da deteção de fraude é comum a ocorrência de transações fraudulentas ser rara, isto pode limitar quão rápido um algoritmo de AL totalmente supervisionado pode começar a ser utilizado nas primeiras iterações do processo. Em resposta a este problema nós apresentamos uma *framework* de AL em três fases que utiliza, num período intermédio, um algoritmo de AL que recorre à estrutura dos dados com *labels* sem utilizar as mesmas. Isto resulta num aumento da eficácia do sistema de AL.

Dada a hipótese de que dois algoritmos de AL podem ser combinados de forma a produzir um que seja melhor que as suas partes, também desenvolvemos e estudamos vários métodos de combinação destes algoritmos. Realizamos uma comparação com uma grande quantidade de combinações que nos levam à conclusão de que tais combinações não aumentam a eficácia relativamente aos algoritmos individuais numa *framework* de três fases.

Finalmente, realizamos um conjunto de experiências em larga escala que cobrem os diversos casos de uso da deteção de fraude. Os resultados indicam que AL é uma solução adequada para os casos de *banking* e *merchant*, principalmente quando utilizados algoritmos de AL baseados em incerteza. Contudo, o nosso estudo não demonstrou resultados positivos para um *dataset* de *banking* com ocorrências de fraude extremamente raras nem para o *dataset* de *merchant acquirer*.

***Keywords*—** active learning, data science, fraud detection, machine learning

ABSTRACT

A problem that arises in many domains when preparing a machine learning (ML) model is label scarcity. In various real world applications, somewhere in the loop of building a dataset, there is a human expert manually annotating each dataset entry with the class label it belongs to. In fraud detection, which is usually addressed as a supervised machine learning problem, having fraud experts carefully reviewing every single transaction is often too expensive, so only a subset of them can be manually annotated. The sub-field of ML known as active learning (AL) has emerged to address this problem. AL implements policies that intelligently choose which instances should be labeled by a human annotator in order to optimize the data labelling costs. The ultimate goal of this procedure is to create a robust predictive model with as little data as possible [Settles (2009)].

In this work, we present a detailed study of various proposed AL strategies by performing experiments with real world data. We focus, primarily, on the scenario where the annotation starts from day-one with no previous data to build historical user profiles and, hence, no labeled data. We present evaluations of several new and already existing types of AL policies and AL configurations through various sets of experiments. The analysis is performed in a streaming setup (as required by the production systems under study) where transactions are processed in real-time.

Besides the choice of a policy, there are other parameters that must be chosen in our AL setup. We conduct dedicated studies to assess the most suitable choices for several such parameters. These studies include the understanding of the impact on the choice of the data pre-processing methods and the ML model to use in evaluations.

Since most AL policies proposed in the literature require that the pool of labeled instances contains labels from all classes, the extreme class imbalance in the fraud detection domain can limit how fast a fully supervised AL policy can start being used in the first iterations of an AL process. To address this issue, we introduce a three-phase AL framework, which uses an intermediate stage policy that does not resort to the label values but can still exploit the labeled pool. This improves the overall performance of all policies used.

Based on the hypothesis that two AL policies can be combined to produce one that outperforms each part, we also develop and study several policy combination methods. We perform a comparison on a large set of combinations that leads us to the conclusion that these do not increase performance when compared to the individual policies in a three-phase setup.

Finally, we perform a set of large-scale experiments that cover several business cases for fraud detection. The results support that AL is an appropriate solution for the banking and merchant business cases, especially when using uncertainty sampling as final policy. However, our study did not demonstrate good results for a banking dataset with an extremely small fraud prevalence nor for a merchant acquirer dataset.

Keywords— active learning, data science, fraud detection, machine learning

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

The field of Supervised Machine Learning (ML) has been the focus of a lot of research in recent years and is becoming and increasingly better solution to address challenges such as automatic fraud detection. The performance of ML models depends crucially on the quality of the training data, in particular the correctness of its labels. In many domains, labeled data is expensive to collect, often requiring human annotation. In such scenarios, it is common for the system to collect large amounts of data for labeling, subject to a limited budget of human annotations. Thus, to maximize the utility of the collected labels, it becomes essential to make sure that the most informative data instances are labeled. The core hypothesis of Active Learning (AL) is that, for every supervised learning problem, there is a minimum amount of intelligently chosen data to fit a predictive model that results in optimal model performance. Hence, in cases where labeling is costly, this sub-field of ML can optimize the costs of deploying a predictive system. More specifically, in an AL framework, a *policy* is used to iteratively select the instances to be queried by an *oracle* (e.g., a human analyst) for labeling [Settles (2009)].

In this study, we work with data from Feedzai. This company implements fraud detection systems for several fraud business cases, including banks, online merchants, and merchant acquirers (explained in section 3.1) some of which process millions of transactions daily. In this domain, there are typically two ways of assigning a fraud label to transactions:

- *Chargebacks* occur when the card or account holder successfully disputes a transaction because the card or account information was compromised. However, the chargeback information may be received several days/weeks after the transaction has occurred or not at all (e.g., if the user does not notice the fraudulent transaction).
- *Analysts' feedback* from experts who analyze and label transactions manually and mark them as fraudulent or legitimate.

It is important to avoid reviewing redundant transactions as manual labeling is expensive (though it provides labels more immediately and it is important to complement chargebacks, e.g., fraudulent transactions may be found for which a chargeback did not occur).

Potential applications of AL in the fraud detection domain include:

1. AL *to build a new model* — this is the scenario of building a predictive model from scratch at the start of a new project. In this case, the client might also be able to provide a historical dataset of unlabeled transactions, enabling the computation of historical card or account holder profiles. Otherwise, if no previous data is available, the system can only use the data that it will start collecting from that moment on.

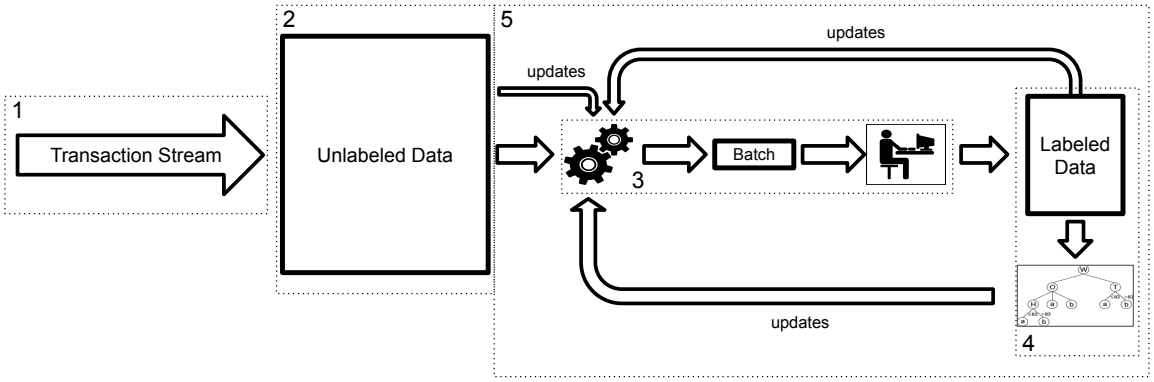


Figure 1: The flow of events of active learning in fraud detection.

2. AL *to prevent model degradation* — fraud detection is an adversarial problem, prone to concept drift [Gama et al. (2014)]. When fraudsters realize their strategies are no longer working, they try new ones to bypass the current system. Hence, to prevent performance degradation, the model must be regularly updated. In this scenario, the system could leverage AL to select which of the new incoming transactions will be most informative to label and then use to update the model.

In Figure 1, we present a diagram that illustrates the expected flow of events in an application of AL to fraud detection:

1. The transactions arrive in real-time (represented by the *transaction stream*) and enter the system sequentially, ordered by time;
2. The transactions accumulate in the *unlabeled data pool*, where all the unlabeled transactions are stored. The size of this pool grows over time (which is a niche of the literature, AL for streaming data);
3. The gears in the middle represent an *AL policy* (the core component of this project), which selects a batch of unlabeled transactions considered to be the most relevant, to be labeled;
4. The *labeled data pool* includes all labeled transactions, including the newly labeled ones, and it is used to train a supervised ML model;
5. On every iteration, we can use the labeled pool, the unlabeled pool, and the current model to *update the AL policy* so that the next batch is selected using the most recent information available. The labeled pool grows iteratively until a stopping criterion is met.

In sum, this framework continuously collects unlabeled data that streams into the system, while selecting small batches of that data to be labeled by human analysts using an AL policy. On each iteration, both the model and the AL policy can be updated with the newly collected data and labels.

In this work, we propose several new AL strategies and evaluate them against other methods in the literature. Unlike mainstream AL literature, we perform our experiments in a streaming scenario. To achieve this we implement a system that simulates the streaming setup in Figure 1, where AL is used to build a new ML model. Using this system, we attempt to address the following questions:

- What is a sufficient budget, in terms of how many data instances we are able to label, to consistently produce efficient ML models?
- Which ML models are most adequate for the specificities of AL?
- Which data pre-processing methods benefit AL the most?
- How can we the address extreme class imbalance in AL?
- Which AL policies tend to obtain best results?
- Can the combination of AL policies work better than its parts?
- How big is the influence of label noise?
- How much do results vary across several fraud business cases?

The software tools and techniques developed in this dissertation can be more widely applied to other use cases. In particular, they have been used recently in Lorenz et al. (2020) to address the money laundering sub-domain of fraud detection.

The technical contributions of this dissertation are as follows:

- We propose a novel discriminative AL policy that is computationally efficient for very large datasets. It uses the labeled pool without resorting to the labels, which is especially useful at early iterations of AL for imbalanced problems. This is introduced in Section 3.6.1.
- A framework for AL that splits the process into three phases, and that is particularly suited to deal with the problem of extreme class imbalance (section 3.7).
- Two measures of similarity to compare AL policies (Section 3.8.1), namely:
  - The similarity between the feature space data distribution of two data subsets. This is based on comparing the distributions of instances in the data clusters between the two pools. We apply this to compare pairs of labeled pools, each generated by a different AL policy.
  - The similarity between the predictions of pairs of ML models. We apply this to compare two ML models trained each with a different labeled pool. The two labeled pools are produced by two different AL policy (Section 3.8.1).
- Two methods to combine different AL policies (section 3.8.2). One is based on weighted scoring and the other is a pipeline of policies that filter instances.
- Techniques for the evaluation of AL configurations, such as our learning curves visualization for the analysis of a single run (Section 4.4.1) and Key Performance Indicators (KPIs) that allow to quantify the performance of AL configurations and carry out large-scale comparisons (Section 4.4.2).
- A large-scale study where several AL policies are benchmarked for multiple fraud business cases (banks, merchants and merchant acquirers – Section 5.6).

The structure of the dissertation is the following. In Chapter 2, we review related work and state of the art in the field of AL. In Chapter 3, we present the financial fraud domain, including our main challenges and potential approaches. It includes system architecture, experimental setup, and AL

policies. Chapter 4 includes datasets and evaluation criteria for the different AL configurations. In Chapter 5 we discuss experimental results and determine the best performing AL configurations. Finally, in Chapter 6, we summarize the main results and conclusions of our experiments as well as future research avenues.

2

STATE OF THE ART

AL is a sub-field of ML with already a few decades of development and several techniques proposed. Given that our goal is to find or develop new policies which best fit our use cases, we now provide an extensive review of the current state of the art [Settles (2009)].

2.1 SAMPLING APPROACHES

When performing AL, it is important to define the data querying procedure, i.e., how the data is selected before an AL method determines which data instances to send to the oracle/labeler. There are several proposed approaches in the literature [Settles (2009)]:

- Pool-based sampling — an unlabeled dataset is available and the policy must iteratively select the samples that it finds most relevant to query.
- Stream-based selective sampling — the data is incoming through a stream and the policy must decide for each entry, individually, if it should be queried or not.
- Query synthesis — instead of selecting samples of the data, the policy generates synthetic samples that represent relevant clusters of the data.

The querying method must be chosen considering the use-case/context of the system and also the performance level that it provides. In a fraud detection scenario, it is common that large amounts of transactions stream into the system every second. Even though it is a streaming scenario, the transactions are coming at a such a high rate that no team of analysts could ever review them all. Therefore a pool of unlabeled data will inevitably be formed. For this reason, we prefer the pool-based sampling option, since it is naturally adapted to this data collection context. Query synthesis is not very suitable when the oracle is a human [Lang and Baum (1992)] (which is our target scenario), since it can generate non-interpretable data (in fraud detection analysts often look at a complex graph of transactions to find out which ones are fraudulent).

In a pool-based approach there is a specific parameter that must be decided: the batch size, i.e. the number of transactions sent to the oracle on each iteration. In theory, a smaller batch size should improve AL because then the policy is updated more frequently. For example, if the batch size is 100, the last instance will be selected by the system without information on the labels of the other 99. On the other hand, if the batch size is 10, at most only the labels of the previous 9 instances will be unknown. However, the smaller the batch size, the higher the computational cost. This can be significant if a heavy AL policy is used or if a ML model is trained after each iteration. Therefore, a careful choice must be made to achieve a good trade off between policy performance and computational performance.

2.2 Active learning techniques

In this section we review the AL techniques most frequently discussed in the literature.

2.2.1 *Uncertainty Sampling*

Uncertainty sampling is a classic, and one of the simplest, AL techniques. In this method, one starts by training a ML model using the labeled pool instances. Then, the model is used to score the unlabeled pool and, based on those scores, we select the instances for which the model is most uncertain about the label. In our scenario, a binary classification problem, the selected transactions are those for which the model predicts scores closer to 0.5. For binary classification, this is equivalent to selecting transactions with a maximum entropy score (if the score is interpreted as a probability). On each iteration, the model is updated and, as a consequence, its decision boundary is refined. Therefore, in this approach, it is essential to use an effective probabilistic model or a ML model with decision boundaries clearly defined (such as support vector machines) [Lewis and Catlett (1994)]. The main advantage of this method is that it is extremely light, given that it only re-trains the model on each iteration with a labeled pool that should be relatively small. However it incurs into the risk of only refining the decision boundary, which could result in the method ending up trapped near a sub-optimal solution due to lack of exploration [Huang et al. (2014)]. Even though it is the simplest method, uncertainty sampling has demonstrated to usually achieve the best results [Yang and Loog (2016)].

2.2.2 *Query by committee*

Query by committee is a simple, but slightly heavier method, whose goal is to explore regions of the data that are more ambiguous and difficult for various models. This method uses an ensemble of machine learning models, selected by the user, which are again trained on the already labeled data and then provide scores on the unlabeled data. Subsequently, a measure of model disagreement is computed for each instance (usually the entropy of the prediction outputs). The instances on which the committee of models disagree the most are the ones selected for querying. This is called the *principle of maximum disagreement* [Seung et al. (1992)]. Even though this method appears to be interesting, it has the major disadvantage that there is no "ideal" committee, which means that the choice of committee members is extremely arbitrary.

2.2.3 *Expected Model Change*

Based on the principle that a useful instance is one that will impact the model, the expected model change method aims to identify the unlabeled instances that will most impact the parameters of the model if their labels are revealed. To achieve this, first a gradient-based classifier is trained on the labeled data pool. Then, for each unlabeled instance, the contribution of the instance to the gradient of the loss function is computed for each label assignment. Finally a weighted sum of the L2 norm of the two possible gradients is computed, which corresponds to the expected gradient norm (see

equation in section 3.3 of Settles (2009)). Instances with the highest expected gradient norm are the ones that should be queried [Settles (2009)]. This technique is quite simple, light and displays good results in most cases [Yang and Loog (2016)].

2.2.4 *Estimated Error Reduction*

Estimated error reduction is a technique that attempts to estimate which of the unlabeled instances would result in a greater reduction of the error produced when making predictions with the model. Provided that in an AL context there is not enough labeled data to obtain the actual model error, it is estimated through the level of confidence on its predictions of the unlabeled pool (see equation 1 in section 2 of Roy and McCallum (2001)). For each unlabeled instance, this method fits an updated model for each of its possible label assignments (in the case of binary classification, 0 or 1). These models are then used to estimate the new error after each of the possible label assignments. Finally, an estimated error reduction value is computed for the instance by combining the errors of each of the possible label assignments [Roy and McCallum (2001)]. This approach is demonstrated in algorithm 1. The whole procedure is repeated for all unlabeled instances. In summary, in order to select the instances to query on each iteration the following steps are preformed:

1. a ML model is fitted on the labeled pool;
2. for each instance in the unlabeled pool:
   a) a copy of the model is retrained with the instance, assuming its label is *negative*.
   b) another copy of the model is retrained with the instance, assuming its label is *positive*.
   c) the estimated error reduction is computed for each copy of the model and they are combined to obtain the total estimated error reduction for the instance.
3. the instances with the greatest total estimated error reductions are the most relevant ones.

Algorithm 1 Estimated Error Reduction.

```
model = MODEL.TRAIN(labeled_data)
error_reductions = []
for instance ∈ unlabeled_data do
    model_0 = COPY(model).RETRAIN(instance, 0)
    model_1 = COPY(model).RETRAIN(instance, 1)
    estimated_error_0 = model_0.ESTIMATED_ERROR(unlabeled \ instance)
    estimated_error_1 = model_1.ESTIMATED_ERROR(unlabeled \ instance)
    error_reductions.APPEND(estimated_error_0 + estimated_error_1)
return MAX_INDEX(error_reductions)
```

This method is expected to work well with the disadvantage that it is computationally extremely heavy. This makes it impractical for our application, due to the amount of transactions we have to handle. Nevertheless, in Yang and Loog (2016) we can observe that uncertainty sampling and expected model change sometimes outperform it. In its original form this algorithm might not satisfy computational efficiency constraints. Other versions attempt to estimate the error reduction in a lighter way. For example, the method in Fu et al. (2018) uses a sampling technique to select a small

number of instances to estimate the estimated error reduction (as an alternative to using all the unlabeled data).

2.2.5 *Density-weighted methods*

Uncertainty sampling assumes that the most relevant instances are those closer to the decision boundary. However, instances can simultaneously be outliers and close to the decision boundary. Such instances may not be of interest in AL, since a data point that is similar to no others may bring less useful information for a ML model. Density-weighted methods aim to select instances that are most representative and that cover all of the data distribution. This can be achieved through, e.g., density computation methods or clustering algorithms [Settles (2009)]. Therefore, even though these approaches can provide good data representations, they are usually computationally heavy. A tree-based clustering approach, proposed in Wang et al. (2017), seems to obtain a good performance, however it comes with a relatively high computational cost.

2.2.6 *Discriminative active learning*

In Gissin and Shalev-Shwartz (2019) a method is proposed that aims to create a labeled pool that is indistinguishable, in a distributional sense, from the unlabeled pool. The hypothesis is that a small but representative labeled pool will provide a good ML model. This method is implemented by i) training a binary classification model that attempts to detect to which pool an instance belongs, and ii) then it queries the unlabeled pool instances for which the auxiliary model is more confident they do not belong to the labeled pool. The rationale is that these instances are not well represented in the labeled pool so they will bring new information if labeled.

This can be interpreted as an outlier detection algorithm which scores the instances of the unlabeled pool that are the greatest outliers relative to the labeled pool. Therefore, we can extend this type of approach by using any outlier detection algorithm that allows scoring instances not used in training, such as Liu et al. (2008), Breunig et al. (2000), Rousseeuw and Driessen (1999) or Schölkopf et al. (1999). In this dissertation, we will propose experiments with this new approach, named Outlier detection Discriminative Active Learning (ODAL), further explained in section 3.6.1.

2.2.7 *Adaptive active learning*

In active learning, different methods typically have different advantages or shortcomings, and follow different principles. Therefore, the question arises whether it is useful to make combinations of different policies. Following this reasoning, adaptive active learning was proposed in Li and Guo (2013), with the goal of combining uncertainty sampling with a density-weighted method to reduce the risk of lack of representativeness in uncertainty sampling. Furthermore, the policy also uses the estimated error reduction method, but only on a small selection of instances with high ranking (as given by the first combination of uncertainty sampling and density-weighted methods) to avoid its high computational costs. This method was named *adaptive* because it adaptively selects the weights assigned to each of the two policies when scoring instances with the uncertainty sampling

and density-weighted policies. The performance of this policy has previously shown good results – see also Yang and Loog (2016).

2.3 Active learning for streaming data

Some studies have appeared in the literature discussing AL methods in a streaming data scenario [Žliobaitė et al. (2011); Zhu et al. (2010b); Zhang et al. (2019); Carcillo et al. (2018)]. Notably, in Carcillo et al. (2018), several AL methods were investigated from a perspective of data visualization, for a credit card fraud dataset. In that simulation study, a budget of instances were selected with AL, to be labeled by analysts, once a day. For some of their methods, a budget was also reserved for semi-supervised labeling using a model trained on the labeled data. In contrast we consider scenarios where several small batches of instances are processed during the day to exploit the collected labels more frequently. Furthermore, we will present a detailed analysis of AL curves in the fraud domain, to give a more complete understanding of the effectiveness of AL for fraud, which is not available in Carcillo et al. (2018). Finally, most of the other studies we found in our literature review are either:

- focused on applying AL to address concept drift, which is a challenge we will not address at this stage.
- not focused on highly imbalanced problems.
- not focused on designing an AL method that deals with the cold start.

For these reasons, we did not find any of the techniques presented in these studies to be of use within our own.

2.4 Stopping criteria

An important question when performing AL is when to stop querying data. When performing AL in a real life scenario, usually there is no test dataset. Furthermore, in a streaming scenario, creating one in parallel doubles the costs of AL and might not provide a good representation of the real data distribution of the future streaming data. In Settles (2009), it is stated that the motivations for choosing specific stopping criteria in AL are usually grounded by economic or other external factors and that its objective can be perceived in two ways:

- The costs of obtaining more labels is higher than the gains that those labels might bring;
- The active learner performance has achieved a plateau and more queries will not bring any significant improvements.

Vlachos (2008) proposed an approach that uses the uncertainty of the scores of predictions on the unlabeled data pool to evaluate the confidence of the model. Above a predetermined confidence threshold the querying process can then be stopped. This method has the caveat of requiring that the machine learning model is a strongly probabilistic one.

One problem that might occur with this method is, as explained in the section 2.2.1, that the exploration focuses solely on exploiting the decision boundary, resulting in lack of exploration. The model might become too confident on the instances that it has already seen while not ever selecting

important "unseen" instances. A possible solution to this, would be to combine this stopping criterion with one that would analyze the existence of outliers in the unlabeled data pool compared to the labeled pool, in an identical fashion to the method explained in the section 2.2.6.

Several metrics for representing uncertainty as a stopping criteria were proposed in Zhu et al. (2010a), such as:

- *Maximum uncertainty* is the uncertainty score for the instance the model is most uncertain about.
- *Overall uncertainty* is the mean of the uncertainty scores for all instances.
- *Selected accuracy*, first queries the top-*m* instances on which the model is most uncertain. Once they are labeled, it computes the accuracy, i.e., how many were correctly predicted by the model;

The thresholds for these metrics to trigger the stopping must be user-predefined. However, Zhu et al. (2010a) also proposes a *threshold update strategy* to avoid the issue of defining an ideal threshold for each AL application. In short, every time the stopping criterion is met with the current threshold this strategy verifies if the ML model predictions on the unlabeled pool have changed since the previous threshold update. If that is the case then the threshold is updated to a more restrictive one. Otherwise, if the predictions have not changed, the process stops.

This dissertation, however, is mostly focused on evaluating AL policies through simulations using historical data. Therefore the stopping criteria will not be a concern in our experiments (we can run a simulation experiment for any streaming period and observe the results by evaluating on a test set). This brief review is included for completeness and as reference for future work.

3

PROBLEM STATEMENT & PROPOSED APPROACHES

In this chapter, we present some the different business use cases where AL can be applied within fraud detection as well as particular challenges for AL in this domain. We also go through the several proposed methods that we will experiment towards the goal of finding an ideal AL system. Specifically, we present:

- The overall architecture of our deployable AL system.
- The overall architecture of the experimental framework used to benchmark several configurations of the system.
- The various AL policies to be evaluated in our experiments.
- How we decompose the AL process into phases.
- More complex methods to combine different AL policies, as well as methods to assess which policies could be worth combining.

3.1 FRAUD BUSINESS CASES

The central goal of this dissertation is to investigate a set of AL policies that can be used to build a predictive model in a situation where labels are scarce in the fraud domain. However, within this field there are several diverse sub-domains, which are victims of very different fraud patterns. Since there is never a single ideal AL policy for all cases [Yang and Loog (2016)], our goal is to investigate an effective AL policy for each business case.

There are three main business cases for transaction data, each with several specific fraud types, which we now discuss in the following sub-sections.

3.1.1 *Financial business case*

The financial business case involves the provision of products and services to banks and other financial institutions. As such, many of the risk patterns found in this context are related with movement of funds that can occur using multiple channels such as ATM, transfers, credit and debit cards. Additionally, in the financial context it is important not only to look at each transaction by itself, but also to analyze the profile of each individual/entity, considering account behavior.

One particular and quite big problem within this business case is money laundering. This is the act (or attempted act) to conceal or disguise the proceeds of illegal activities so that they appear to

come from legitimate sources of activities. The tools and techniques developed in our work have been used to tackle that problem in Lorenz et al. (2020).

3.1.2 *Merchant business case*

A merchant is any business engaged in the sale of goods or services. But, only merchants that accept bankcards as a form of payment are pertinent to our explanation. So with that disclaimer, a merchant is any business that maintains a merchant account that enables them to accept credit or debit cards as payment from customers (cardholders) for goods or services provided.

3.1.3 *Acquirer business case*

An acquiring bank is a registered member of the card associations (Visa and MasterCard). An acquiring bank is often referred to as a merchant bank because they contract with merchants to create and maintain accounts that allow the business to accept credit and debit cards. Acquiring banks provide merchants with equipment and software to accept cards, promotional materials, customer service and other necessary aspects involved in card acceptance. The acquiring bank also deposits funds from credit card sales into a merchant's account.

3.2 THE CHALLENGES OF ACTIVE LEARNING IN FRAUD DETECTION

AL may be extremely useful in the fraud domain because obtaining a high quality labeled dataset is expensive – it requires hiring fraud analysts to spend a substantial amount of time labeling transactions. However, there are some challenges specific to this domain, namely:

- The data is streaming. The existence of a growing pool of unlabeled data is not standard in AL contexts. In fact, many state of the art methods require a fixed sized unlabeled data pool, which makes them unsuitable for our problem (e.g., most density-weighted methods are not directly applicable, as explained in section 2.2.5).
- In the domain of financial transactions, it is common to have to deal with a magnitude of thousands of data instances incoming per minute.
- Given that analyzing transactions in order to detect fraud can be a complex and difficult task, a lot of data noise at the label level is to be expected.

Regarding the computational challenges, all solutions proposed in this work must comply with the following pre-requisites:

- Must be computationally efficient enough to work with large amounts of data. The complexity of fitting the policy must not grow with the unlabeled pool size.
- Cannot rely on a fixed unlabeled pool to be efficient, i.e., some methods (such as some density weighted method, in section 2.2.5), are slow because they must fit on the available unlabeled pool (which is a growing problem if the unlabeled pool is continuously expanding).

Also, it is expected that label noise makes it harder to build efficient ML models with small quantities of data. We will address this by using highly regularized ML models (section 4.3).

3.3 PROBLEM STATEMENT

The central goal of this project is to find the sets of configurations that perform well within the fraud detection domain. This leads us to ask several questions that we will attempt to address:

- What is a sufficient budget (i.e., labeled pool size and/or workdays of analysts) to consistently produce efficient ML models?
- What ML algorithms, and respective hyper-parameters, are most suited to AL?
- Which is the way of pre-processing the data that benefits AL the most (i.e., feature engineering and feature selection)?
- Provided the extreme class imbalance of some datasets in the fraud domain, how can we address the absence of positive labels in labeled pools at early stages in the AL process?
- Which are the AL policies with the best model and computational performance?
- Can we combine AL policies in order to get improved ones?
- How much does the noise in the labels (possibly from analysts' mistakes) influence the performance of AL?
- How much do these results vary across the different fraud business cases?

In section 5 we will present results that will help us draw conclusions regarding some of these questions.

3.4 SYSTEM ARCHITECTURE

This dissertation aims to provide a study of different AL methods on various real datasets in a fraud detection setting. In this chapter, the architecture of our experimental framework is presented. This system can be deployed in a streaming scenario, as illustrated earlier in the diagram of figure 1, to output suitably trained predictive models. The system architecture is represented in the diagram of figure 2. The process starts in the *data stream*. The data is then passed through the *pre-processing pipeline* and then into the *data manager*, where it is stored (two indexes are also created in the data manager to define data views for the labeled and unlabeled pools). The *policy manager* is responsible from accessing the *data manager* in order to signal the *labeling manager* which transactions should be classified by a labeler. The *model train manager* requests the labeled pool from the *data manager*, in order to train ML models. The aim is to deploy this system in a production environment, to output trained predictive models to classify incoming transactions in real-time.

In this section we explain in detail each component of this system and how they connect to each other.

3.4.1 *Pre-processing*

The pre-processing happens in two phases: the autoML tool in Marques et al. (2020) followed by our pre-processing pipeline (where additional operations, such as feature selection and encoding, are applied).

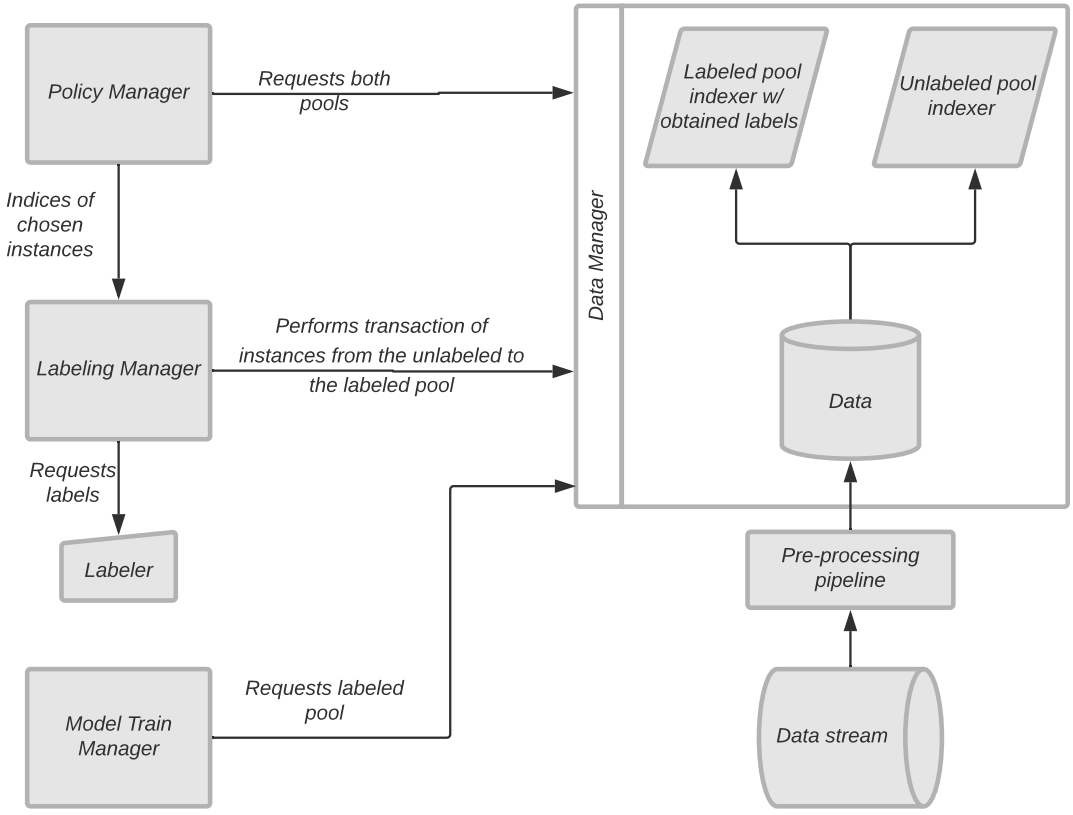


Figure 2: The necessary components to deploy the proposed active learning tool.

For our experiments it turned out to be computationally inefficient to use complete datasets (the daily volume of data varies greatly according to the use case). Therefore, in preparing our datasets, we applied an additional stratified sampling step to normalise all datasets to a similar transactions volume to be able to run our experiments in a practical amount of time (this step may or may not happen in a real production system). This stratification occurs separately for each label group (cards victim of fraud and cards not victim fraud) with the same sampling fraction. For the purpose of our experiments, in figure 2, the data arriving at the *data stream* component has already gone through this sampling process.

AutoML

Financial fraud cannot be detected by observing a single transaction. In order to detect fraud, a transaction (or a group of transactions) must be compared with the typical behavior of the user. When using machine learning to detect fraud, one of most common and effective methods is to perform feature engineering by computing behavioral profiles to characterize users [Kou et al. (2004)].

As a fundamental step, *Feedzai*'s auto ML solution [Marques et al. (2020)] already incorporates the feature engineering process. We use it in this dissertation for data preparation. Features are generated to build user profiles using sliding windows (varying in size from seconds to several weeks). The aggregations to characterize a user can be done by several relevant grouping entities such as user id, card id or email (e.g., average amount spent in a 10 minutes time interval for a given card).

Active learning pre-processing pipeline

Before entering the AL process, the data still needs to be further transformed for several reasons:

- The number of numerical features generated by the autoML tool is usually too large to be used in a production system, and it may introduce unnecessary noise in the ML model training. This makes the application of dimensionality reduction techniques necessary;
- Categorical features are not suitably encoded, which is a limitation for most machine learning models;
- The data source created by autoML contains raw features that were in the original data source and are not useful after feature engineering (e.g., the user ids used to compute user profiles).

The *pre-processing pipeline* addresses these issues by applying the following procedures:

- Ordinal encoding of categorical features [Potdar et al. (2017)];
- Frequency encoding of categorical features [Uyar et al. (2009)];
- (Optional) Unsupervised dimensionality reduction of the set of numerical features;
- Dropping of raw features that are not to be used.

For the unsupervised dimensionality reduction step, we consider the following methods (depending on the study):

- Principal component analysis (PCA) [Wold et al. (1987)] — this method turned out to be extremely efficient at optimizing the features generated by the user profiles, mainly because features based on user profiles tend to be highly correlated. However, this method produces complex linear combinations of the original features making it harder for the interpretability of model decisions, which can represent a concern in the fraud detection business case. We use the implementation in the *scikit-learn* library [Pedregosa et al. (2011)].
- Pairwise correlation filter [Yu and Liu (2003)] — we implemented this method that, provided a maximum correlation threshold, computes the correlation between every pair of two features and removes one of the features if the correlation exceeds the threshold.
- Domain knowledge selection — another alternative is to perform manual selection of the features to be computed, simply using the domain knowledge of an expert in fraud detection.

Observe that both *pairwise correlation* and *dimensional reduction* require a sample of unlabeled data. In real applications this is not an issue because unlabeled data is easy to collect through an initial waiting period (e.g., one day). The feature selection pipeline can, in principle, be periodically updated by re-fitting to the latest available data.

In order to decide which of the presented methods we should use we conducted some experiments that will be presented in section 5.2.

3.4.2 *Data manager*

In an AL problem there are always two main data components: the labeled pool and the unlabeled pool. The unlabeled pool, in our case, grows overtime as data arrives at the system via the data stream. As for the labeled pool it also grows, as more labels are collected for the selected instances from the unlabeled pool. We assume that the data comes from the stream at a much higher rate than the rate at which instances are labeled, so the unlabeled pool is always growing.

Our method to simulate a realistic scenario and managing the data consists of storing all of it, sorted by its timestamps in ascending order, and then each of the pools, as described above, is defined as a set of indices (or views) pointing to the original data store. In the case of the labeled pool, in addition, the obtained labels are also stored in the corresponding data indexer.

3.4.3 *Policy manager*

The policy manager (2) is the component in charge of signaling which instances should be labeled and sending them to the Labeling manager (3.4.4). Some AL policies are not viable in every condition, e.g., when the labeled pool is still empty, only policies which do not resort to the labeled pool can be used, such as a random policy. For this reason, in our framework we do not use a single policy throughout the whole AL process. The policy being used is swapped overtime when given conditions are met (we call these conditions the *switching criteria*). The policy manager is responsible to verify these conditions and determine which policy should be used at each time.

Algorithm 2 Policy manager.

Require: P_n is the list of AL policies, sorted by intended usage order; SC_p is the list of switching criteria, where its i^{th} element is the switching criterion between policies i^{th} and $(i+1)^{\text{th}}$; X_u, X_l and y_l are the unlabeled pool, the labeled pool and the labels of the labeled pool, respectively. LM is the labeling manager object.

Ensure: $\text{LENGTH}(P_n) = \text{LENGTH}(SC_p) - 1$

$idx \leftarrow 0$
while $idx < \text{LENGTH}(SC_p)$ & $SC_{idx}.\text{SWITCH}(X_u, X_l, y_l)$ **do**
    $idx += 1$
$chosen\_unlabeled\_pool\_indices \leftarrow P_{idx}.\text{CHOOSE}(X_u, X_l, y_l)$
$LM.\text{QUERY}(chosen\_unlabeled\_pool\_indices)$

3.4.4 *Labeling manager*

This component has two entities: the labeler and the labeler scheduler. Labelers are entities that, upon request, retrieve labels for data instances. In a real life streaming scenario, these labelers would likely be fraud analysts. The labeling manager can have several active labelers. The labeler scheduler is what manages the distribution of instances to be reviewed by the labelers. We implement this component in anticipation of future work to develop a tool that assigns transactions to be reviewed to the analysts in an intelligent manner (e.g., by learning which analysts are better with the different types of transactions). In this work, we only use the labeling manager with a single labeler (explained in further detail in section 3.5.2).

3.4.5 *Model train manager*

From the moment we have a labeled pool a ML model can be trained, even if, initially, it might not be an accurate one. The goal of the *Model Train Manager* is to train a model, on each iteration, following any given training procedure – e.g., a fixed set of hyper-parameters or using hyper-parameter tuning, with or without (cross)validation. This model may in principle be used for two purposes: evaluation of the current labeled pool (if a test set is available or cross validation is used) or deployment.

3.5 EXPERIMENTAL FRAMEWORK

This dissertation aims to investigate which AL configurations (e.g., which pre-processing pipeline, AL policy, ML model) we should use to obtain consistently good results in a production AL system. To perform such investigation, we developed an enlarged version of our framework with the purpose of experimentation. The added/modified components are the following:

- Simulated data stream — despite the fact that the data we use in the experiments was collected in a streaming environment, it is historical data. Therefore we simulate the stream.
- Data manager — the only difference in this component, is that the data already contains labels (even though the ML model only has access to them when the instances are in the labeled pool).

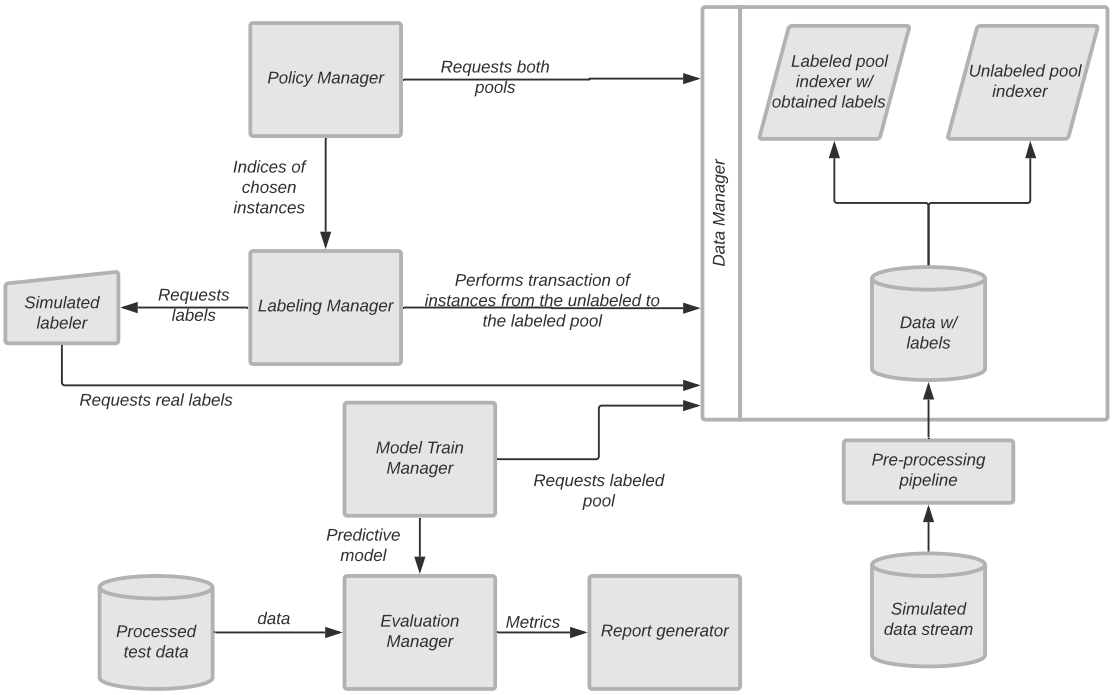


Figure 3: The components in our active learning experimental framework.

- Simulated labeler — since it is not affordable to introduce an actual analyst in these preliminary experiments, we simulate one, as explained in section 3.5.2.
- Evaluation manager — the component in charge of measuring each of the AL configurations.

A diagram with these components can be visualized in figure 3.

3.5.1 *Simulated data stream*

In a real scenario, data is incoming in real-time from a stream provided by the entity that processes the payments. Data instances are iteratively chosen by the AL policy to be sent to the analysts. The model only starts being trained with this recently labeled data after the time it took to be reviewed by the analysts. During this period, the data stream continues collecting recently arrived transactions, so the size of the unlabeled pool continues increasing.

In our experimental setting, a simulation was used instead of an actual stream. For that purpose, we use historical data from different financial data contexts (section 3.1) and implement a streaming simulation tool. In order to simulate the effect of the unlabeled pool growing while the analysts review transactions, there is a variable that identifies the current timestamp of the simulated environment. Every time a batch of data is reviewed, the time it took to review is added to the current timestamp. Then, the simulated data stream appends all transactions between the previous timestamp and the current timestamp to the unlabeled pool.

3.5.2 *Simulated labeler*

Since we simulate the flow of data, the historical data source used already contains labels. Hence, we implement a simulated labeler that, when it receives a request to review a batch of transactions, accesses the labels already stored in the historical data source. In order to also simulate somewhat accurately the time it takes for an average proficient analyst to review transactions, our simulated labeler reviews 1000 transactions per day. Hence it signals the system that 86.4 seconds have passed for each labeled transaction (i.e., 24 hours $\times$60 minutes $\times$60 seconds /1000).

3.5.3 *Evaluation Manager*

The evaluation manager, whose purpose is to measure the quality of the labeled pool at each iteration, is used every time a new batch is labeled. Once the labeled pool is incremented with this batch, a ML model is trained from scratch with the whole pool. Then, this model scores the test data and the configured performance metrics are computed (e.g., area under the ROC curve, recall at some false-positive rate, etc.) and stored.

3.5.4 *Processed test data*

The pre-processing pipeline might contain operations that must be applied to the raw data collected in the data stream (e.g., dimensionality reduction). Thus, the output of the pipeline can be highly dependent on the data on which it was fitted (even then it might be stochastic depending on the pre-processing algorithm used). For this reason, data that is used by the system must go through the exact same pre-processing pipeline. For a consistent simulation, this pipeline is trained on the first unlabeled pool before the AL process is started (e.g., data collected in an initial waiting period of one day). This pipeline, besides being used to process the rest of the streaming data, is also stored in a persistent way. Hence, whenever an evaluation is performed, the configured test set is processed using this pipeline, thereby preparing the data to be scored by the model.

3.5.5 *Report generator*

In order to be able to visualize results of a single experiment-evaluation pair, we introduce the report generator. This accesses results, such as the metrics stored in the evaluation, to automatically produce a report containing learning curves with their variance bands as well as Key Performance Indicator (KPI) metrics (to be detailed in section 4.4).

3.6 Active learning policies

In this work we implemented several policies mentioned in chapter 2 as well as new variations and new methods, which we now describe in further detail.

3.6.1 *Outlier discriminative active learning (ODAL)*

Given that AL is about creating a pool of unlabeled instances with minimal size, in this section we propose a policy that follows a principle similar to the discriminative AL method [Gissin and Shalev-Shwartz (2019)]. As explained in section 2.2.6, at each iteration that method finds which instances of the unlabeled pool are less well represented in the labeled pool. Our policy, however, achieves this goal using a different approach. On every iteration, we train an outlier detection algorithm on the labeled pool and then use it to score every instance of the unlabeled pool. Our assumption is that the greatest outliers of the unlabeled pool relatively to the labeled one are the transactions that are worse represented in the labeled pool. Hence, these transactions should be reviewed and added with their labels. As outlier detection methods, we experiment with Isolation Forests [Liu et al. (2008)] and Elliptic Envelope [Rousseeuw and Driessen (1999)][1]. ODAL has the great advantage that it does not require labels to be used. This is helpful in the fraud domain, where the classes are extremely imbalanced and sometimes finding a first positive case in AL might take a lot of time. It can also be considered a computationally light solution, given that the outlier detection algorithm is trained solely on the labeled pool, which should be relatively small (and orders of magnitude smaller than the unlabeled pool).

3.6.2 *Query by committee*

As explained in 2.2.2, query-by-committee (QBC) is a method where several models, belonging to a committee, are trained on the labeled pool. Then the level of disagreement between models, on the predictions of the unlabeled pool, is used to define which transactions are most relevant to query. The models we used in our committee were as follows (for all unspecified hyper-parameters the default values from the *scikit-learn* library were used, since they typically provide finer control that only needs to be changed for very specific problems):

- A random forest with 100 trees and max depth of 3.
- A logistic regression.
- Gaussian Naive Bayes.
- Gradient boosting ensemble, with early stopping at 20 iterations without improvement (this implementation requires at least two instances with label from each class for the model to be trained).

Since each model outputs scores with a different distribution of values (they are of a different nature), we implemented our disagreement measure using the differences between rankings in the classification instead of the score entropy proposed originally in Seung et al. (1992). Further implementation details are presented in algorithm 3.

1 All the ML models used in this work are implementations from the *scikit-learn* package [Pedregosa et al. (2011)]. For further details on the models and their defaults, see https://scikit-learn.org/0.22/user_guide.html.

Algorithm 3 Query by committee disagreement measure.

Require: $pred_m$ is the list of prediction arrays from every model in the committee. $\#models$ is the number of models in the committee and $\#pred$ is the number of predictions per model, which is the number of instances in the unlabeled pool.

$ranks_m \leftarrow []$ ▷ Initializes a list for the arrays with rankings of each prediction per model.
for $predictions \in pred_m$ **do**
    $prediction\_ranks \leftarrow \text{RANK}(predictions)$ ▷ Computes the rank of each prediction.
    $ranks_m.\text{APPEND}(prediction\_ranks)$
$differences_{combination} \leftarrow []$ ▷ Initializes a list for the differences of each combination of two models.
$i_1 \leftarrow 0$
while $i_1 < \#models$ **do**
    $i_2 \leftarrow i_1 + 1$
    while $i_2 < \#models$ **do**
        $rank\_differences \leftarrow \text{ABSOLUTE}(ranks_{i_1} - ranks_{i_2})$ ▷ Computes the absolute differences array between the ranks.
        $differences_{combination}.\text{APPEND}(rank\_differences)$
        $i_2 += 1$
    $i_1 += 1$
$difference\_means \leftarrow \text{AVG}(\text{array}=differences_{combination}, \text{axis}=0)$ ▷ Computes the average differences for each instance.
$scores \leftarrow \text{NORMALIZE}(difference\_means)$ ▷ Performs normalization so that scores are between 0 and 1.
return $scores$

3.6.3 *Expected model change*

Expected model change, as explained in section 2.2.3, is a policy that requires the usage of a gradient-based ML model. It is used to compute the expected change that each transaction of the unlabeled pool would cause if the current model was trained with it. The higher the expected model change due to an instance, the more relevant its label is. In the experiments we use the *scikit-learn* logistic regression model with default hyper-parameters.

3.6.4 *Uncertainty sampling*

As discussed in section 2.2.1, uncertainty sampling queries instances for which the model produces scores that represent a higher uncertainty level. We implemented this method using three different approaches:

- model score distance to 0.5 — the instances whose score is closer to 0.5 are the ones considered most uncertain. This method has the disadvantage of the used threshold (0.5) being independent of the model. In some scenarios, this value might not represent uncertainty (e.g., when training a random forest with a dataset that has very few positive labels, we observed, in some of our experiments, that the model only outputs scores below this threshold)
- distance to fraud percentile — the fraud rate of the labeled pool is computed and then that value is used to compute the percentile of the predictions on the unlabeled pool, which is then used as uncertainty threshold.
- epistemic uncertainty — as defined in Shaker and Hüllermeier (2020), uncertainty can be separated in two main categories: aleatoric and epistemic. Aleatoric uncertainty is intrinsic to the data generating process and can never be removed. Epistemic uncertainty can be reduced by collecting more data. We use the implementation proposed in Ascensão et al., which instead of total uncertainty only uses the epistemic uncertainty as measure of relevance for labeling.

The uncertainty sampling methods are also highly influenced by the choice of the model used. In our case we chose to use the same model and hyper-parameters that we are also using for evaluation.

3.6.5 *Density-weighted methods*

Density weighted methods attempt to create a labeled pool that covers the whole feature-space of the data by identifying clusters and querying from them all (section 2.2.5). We implement a variant of these methods. Briefly, for every batch sent for review, this method works in the following steps:

1. An unsupervised ML model identifies the clusters of the whole data (labeled and unlabeled pools). We experimented with the K-means and DBSCAN algorithms [Xu and Wunsch (2005)] (implementations from Pedregosa et al. (2011)), with several different sets of hyper-parameters.
2. The relative frequencies of instances from the labeled pool in each cluster are computed.
3. Instances in the unlabeled pool from clusters that have lower relative frequency values are the ones chosen for review.

As speculated in section 3.2, this method turns out to be unfeasible to use in our experiments. The growth of the unlabeled pool requires the unsupervised model to be fitted on every iteration, which results in a very high computational cost.

3.7 Active learning phases

The experiments we will present begin with an empty labeled pool, as explained in section 1. Therefore, at the start of our AL process, data queries must be performed without resorting to the labeled pool. Hence, we implement our framework to support multiple phases, particularly an initial phase of unsupervised querying, as discussed below.

3.7.1 *Two-phase active learning*

In the first batch of experiments to be presented, we use only two phases, each with a policy of the following types:

1. **Cold policy** — a method that does not use the labeled pool and selects instances at random or using an outlier detection algorithm trained on the unlabeled pool.
2. **Hot policy** — a method that starts being used only after the cold policy found enough positive instances (this constraint is necessary because most of the models used by these policies, except for the discriminative ones, require positive instances in the labeled pool).

3.7.2 *Three-phase active learning*

In the second round of experiments, we apply a three-phase AL strategy, introduced to solve a problem detected in the two-phase AL experiments. Specifically, given the very low prevalence of the positive class in the fraud domain, we will see that some AL training runs only switch to the hot policy very close to the end of the established query budget. This behavior is not desirable because then some runs may only use the random policy from start to end without exploiting an AL policy that uses the labeled pool.

Therefore, we introduce an AL process composed of three phases, each with a different policy, as follows:

1. **Cold policy** — One single batch of randomly queried data[2].
2. **Warm-up policy** — A method that uses both pools but not the label values (e.g., a discriminative policy).
3. **Hot policy** — A supervised AL policy triggered when at least one fraudulent instance is found, except in the case of the query-by-committee policy, for which at least two are needed (see discussion in section 3.6.2).

2 We will see in the two-phase experiments that, in some cases, the outlier detection methods are superior to random sampling. However, that behavior is not stable across different time folds. Since the cold policy is only used for a single batch, and the random policy is computationally more efficient and unbiased, we opted to run three-phase experiments only with a random cold policy for the first batch.

3.8 POLICY COMBINATION

Different AL policies follow different sampling principles, each with its advantages and disadvantages. Therefore, one can hypothesize that different policies might query data from different regions in the feature space, resulting in prediction models that, while with similar overall performance, might classify specific instances differently. This hypothesis motivates the use of policies that combine sets of different policies. A combination may yield better performance than single models since it can potentially cover more regions of the feature space.

3.8.1 *Policy divergence diagnostic*

In order to decide if we should combine AL policies and, if so, which ones should be combined, we introduce a method to analyze the divergence between two policies.

For this assessment a metric to compute the divergence/distance between two data distributions (i.e., the labeled pools) must be defined. In this section we will introduce two such metrics, the *cluster frequency distance* and the *model disagreement distance*.

Since most AL policies are stochastic and each random seed likely results in different labeled pools, when comparing two policies we run several experiments for each of them with N different seeds, to be able to evaluate the policy's variance. We can also exploit this data to estimate the distribution of divergence values between any two policies. Therefore, given a divergence metric, all combinations of pairs of labeled pools between the two policies are compared, which results in a total of $\binom{N}{2}$ divergences to be computed (e.g., for 35 seeds we obtain 595 pairs). Then, we assess the degree of divergence between the two policies by comparing the distribution of divergences between their labeled pools, with the distributions of self-divergences between each policy with itself. For example, given two policies with a distribution of divergences that appears large, if the distributions of self-divergences are equally large, then the difference cannot be considered significant.

Cluster frequency distance

As stated before, it is possible that different policies are fetching data from different regions of the feature space (see illustration in figure 4). Therefore we introduce a metric to compute a measure of similarity (or divergence) between two data pools that works in the following steps:

1. It uses an unsupervised ML algorithm to cluster the whole labeled data (this is the union of all N labeled pools in both policies).
2. For each pair of labeled pools to compare, it computes the frequencies of instances that each pool has in the clusters, i.e., a histogram of cluster label frequencies for each pool.
3. It computes the Jensen-Shannon divergence (JSD) [Lin (1991)] between the two histograms, which serves as the measure of divergence between the two data pools.

This process is summarized in the algorithm 4.

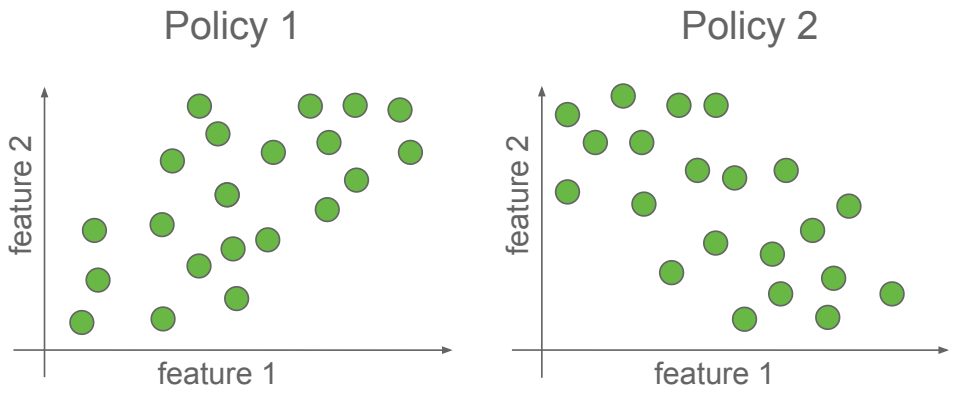


Figure 4: Illustration of the final labeled pools from two different active learning policies which cover different regions of the feature space.

Algorithm 4 Distance between two data pools.

Require: L_1 and L_2 are the final labeled pools for policy 1 and 2, respectively; CLUSTER is a clustering algorithm already trained on all of the data.

1: $labels_1 \leftarrow$ CLUSTER(L_1) ▷ Gets the cluster labels for each pool
2: $labels_2 \leftarrow$ CLUSTER(L_2)
3: $bins_1 \leftarrow$ HISTOGRAM($labels_1$)
4: $bins_2 \leftarrow$ HISTOGRAM($labels_2$)
5: $distance \leftarrow$ JSD($bins_1, bins_2$) ▷ Computes the Jensen-Shannon Divergence (JSD) between the two histograms' bin frequencies
6: **return** $pool_{distance}$

Model disagreement distance

The metric introduced in the previous section is only sensitive to the distribution of the features regardless of the label. Another different approach is to define a metric that compares model predictions instead. Two different AL policies likely generate different labeled pools. When training two equal ML models, each with one of the two pools, they might each produce better predictions on different instances of the data. In order to compute a measure of distance between two ML models we use the ratio between the number of instances on which the models disagree and the total number of scored instance, which we refer to as *ratio of prediction disagreement*. This is detailed in algorithm 5.

Algorithm 5 Distance between two ML models based on prediction disagreement.

Require: m_1 and m_2 are predictive models trained on the final labeled pools for policy 1 and 2, respectively; X_{test} and y_{test} are the test set features and labels, respectively.

$pred_1 \leftarrow m_1(X_{test})$ ▷ Obtains the models' predictions on the test set
$pred_2 \leftarrow m_2(X_{test})$
$\#disagreements \leftarrow 0$
$\#accurates \leftarrow 0$
$i \leftarrow 0$
while $i \leq \#y_{test}$ **do** ▷ Iterates through y_{test}, $pred_1$ and $pred_2$
    if $pred_1^i = y_{test}^i$ & $pred_2^i = y_{test}^i$ **then**
        $\#accurates = \#accurates + 1$
    if $pred_1^i \neq pred_2^i$ **then**
        $\#disagreements = \#disagreements + 1$
    $i = i + 1$
$distance \leftarrow \#disagreements / (\#disagreements + \#accurates)$
return *distance*

3.8.2 *Policy combination methods*

In this section, we introduce three methods for combining AL policies: one based on weighted-scoring, another one on alternating policies, and the last one on a pipeline of policies that filter the instances. The second one is a sub-type of the first one, as discussed further.

Weighted ranking combiner

An approach to policy combination based on uncertainty and information density through weighted ranking, has been proposed in Li and Guo (2013). This is applicable to any weighted combination of policies through the formula

$$f_w(x) = p_1(x) \times w + p_2(x) \times (1 - w)$$

where x is the instance being scored for ranking, p_1 and p_2 are the policies to be combined and $w \in [0, 1]$ is the weight for p_1. The problem of selecting the unknown value of w is addressed by

starting with a predefined set of representative values $W = \{w_1, w_2, ..., w_n\}$, where the values w_i are all the possible values of w to adaptively select in each query (e.g., $W = \{0, 0.25, 0.5, 0.75, 1\}$).

In Li and Guo (2013), the combined policy queries through the following procedure:

1. Finds the top ranked instance for each value of W, by building the set

$$\left\{ \arg\max_{x \in U} f_w(x) \forall w \in W \right\}$$

2. Applies the estimated error reduction method, explained in section 2.2.4, on the top ranked instance for each weight w_i to determine the most interesting one.

Even though this method was designed for a batch size of 1, it can be easily modified to find the top-N instances for each value of W, with N constrained such that

$$batch\_size = N \times n \ .$$

In this modified setup we note that the estimated error reduction step might be unnecessary and heavy operation, so we implement a simplified version without it. Then N and n are then adjusted for the equality to hold so that the batch size is met.

Alternating combiner

Two policies may be good candidates for complementing each other but score instances in a mutually exclusive way, i.e., data instances that one of the policies scores as highly relevant, the other one scores as not relevant at all. In this case, the intermediate weights of the *weighted ranking combiner* (e.g., $w = 0.5$), would not combine these policies adequately. This is because with intermediate weight values, instances that obtain a high relevance score with one policy and a low one with the other one will always obtain only an intermediate final relevance score. For that reason, we introduce a policy that combines two policies by simply alternating between them. This is easily achieved by using the *weighted combined learner* with $W = \{0, 1\}$.

Cascade combiner

The last strategy we consider for policy combination is to start by pre-selecting N instances with *policy 1* and then filtering the *batch size* best instances out of those N with *policy 2*. This procedure is defined in algorithm 6.

Algorithm 6 Cascade combiner.

Require: X_U is the unlabeled data pool; p_1 and p_2 are policies 1 and 2, respectively; N is the number of instances for policy 1 to select; *batch_size* is the number of instances to send to the analyst.

Ensure: $N > batch\_size$

1: $X_{ranked} \leftarrow p_1(X_U)$ ▷ Ranks X_U by the interest of p_1
2: $X_{best} \leftarrow X_{ranked}[: N]$ ▷ Filters the best N instances
3: $X_{ranked} \leftarrow p_2(X_{best})$ ▷ Ranks X_{best} by the interest of p_2
4: $X_{final} \leftarrow X_{ranked}[: batch\_size]$ ▷ Filters the best *batch_size* instances
5: **return** X_{final}

4

EVALUATION STRATEGY

In this Chapter, we define an evaluation strategy to test the several experiments we are performing, analyze them individually and compare them with each other. We need to define:

- The datasets we are using. It is important that, at least in the final experiments, besides covering all the fraud business cases mentioned in section 3.1, we also cover different dataset characteristics such as the fraction of fraudulent instances.
- How the datasets are split. Each of the datasets contains several months of data and we want to partition them in a way that allows us to understand the impacts of applying AL in different periods so that we can test AL under the effect of concept drift [Gama et al. (2014)].
- The ML model to be used for evaluating labeled pools and obtaining the metrics. The model, and its hyper-parameters, must be chosen such that it is effective within AL.
- The performance metrics to evaluate the experimental results, namely:
  - Visualization techniques to observe the behaviour of a single experiment.
  - Key Performance Indicators (*KPIs*) to aggregate each experiment into quantifiable metrics that allow us to perform large-scale comparisons.

In the following sections, we start by describing the datasets we are using for each fraud business case (Section 3.1). We also explain how we are exploiting one of the datasets to perform experiments that allow us to understand which types of labels we should be using to improve performance in AL. After defining the datasets, we define how we are splitting them into several time folds. Then, we define the ML models we are performing experiments with in order to understand the impact that they have on the evaluation, and to decide which one we are using for the final experiments. Finally, we explain which performance metrics we will using to analyze and measure performance of AL configurations, including our visualization techniques for single experiment and the KPIs.

4.1 DATASETS

In table 1, we provide details on each dataset to be used in the experiments, as well as on how they were partitioned in time folds. We provide three fraud business cases (see also section 3.1): financial business case (the banks), merchants and merchant acquirers. We have three datasets from banks. The first two are from the same bank, where in the second one it was applied an additional card-not-present (CNP) filter. The fourth and fifth datasets are for the other two use cases. In the table the following details are included for each dataset:

Dataset	# transactions	Sampling by card	Fraud rate	Starting dates of extracted folds	# transactions per fold
Bank 1	416M	1.0%	0.20%	2016-01-18	300K
				2016-02-15	302K
Bank 1 w/ CNP filter	163M	3.0%	0.33%	2016-01-18	360K
				2016-02-15	362K
				2016-03-14	356K
				2016-04-11	362K
				2016-05-09	373K
Bank 2	43.4M	11.0%	0.03%	2014-11-17	323K
				2014-12-15	317K
				2015-01-12	280K
				2015-02-09	289K
				2015-03-09	292K
Merchant	18.3M	100.0%	3.10%	2018-01-15	276K
				2018-02-12	281K
				2018-03-12	290K
				2018-04-09	355K
				2018-05-07	411K
Merchant acquirer	221M	2.5%	1.20%	2018-10-22	416K
				2018-11-19	448K
				2018-12-17	401K
				2019-01-14	397K
				2019-02-11	280K

Table 1: Table with details about each dataset used.

- The number of transactions in the full dataset before under sampling (in millions).
- The fraction of stratified sampling by card (explained in section 3.4.1).
- The fraction of fraudulent instances.
- The dates of the first instance of each time fold. Each time fold has four weeks of data starting on a Monday just after the midnight of the previous Sunday.
- The number of transactions for each time fold after under-sampling.

The reason why we only extracted two time folds from *Bank 1*, while using five for the others, is because this dataset was only used in a first stage of preliminary experiments.

4.1.1 *Label scenarios*

For the *Merchant* dataset, we have access to additional information on how the label was obtained. The diagram of figure 5, illustrates the possible ways in which an instance's label can be obtained. The different classes of labels are as follows:

- No feedback (NF) — when no analyst has reviewed the transaction nor there has been a chargeback. This is presumed to be a legitimate transaction.

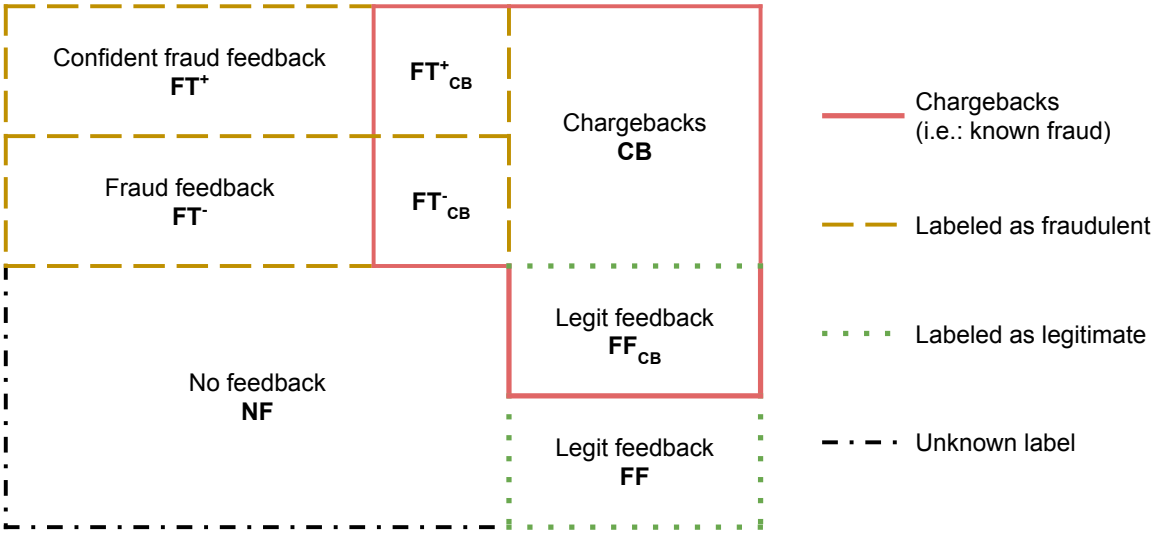


Figure 5: Label types diagram.

- Chargeback (CB) — when the client/fraud victim complains about the illegitimate transaction to get their money back. These are the labels least likely to be incorrect.
- Legit feedback (FF) — when an analyst has reviewed a transaction and considered it to be legitimate. When this is subscript as FF_{CB}, it means that there was a chargeback, hence the transaction was actually fraudulent, despite the analyst having marked it as legitimate.
- Fraud feedback (FT^-) — an analyst considered the transaction to be fraudulent but without much confidence. When (FT_{CB}^-), the analyst suspicion was confirmed by a client's chargeback
- Confident fraud feedback (FT+) — an analyst was confident the transaction is fraudulent. When FF_{CB}^+, it was confirmed by a chargeback.

Having this detailed information on the label collection procedure allows us to investigate, in part, a challenging effect that arises in this problem: label noise. In particular, we can apply AL to query only transactions that have been reviewed by analysts. In contrast, in other datasets where this information is not available, our simulations may also select queries to label (e.g., chargebacks) that might not be collected in a realistic setup, or for which analysts would have difficulty labeling.

In our experiments, we manipulate this dataset in order to simulate the three following setups:

1. Analysts' feedback — in a real production system, a team of analysts would be labeling the transactions that the AL policy queries. We simulate this by only using positive instances that have been reviewed by analysts.
2. Confident analysts' feedback — analysts are humans and therefore can make mistakes. One way to try to overcome the challenges that their mistakes cause is to only use their high confidence positive labels. This is possible in our experiments, since we have "confident fraud feedback" labels.
3. Chargebacks — when a card holder complains about a fraud event, a chargeback is issued. Due to its nature, these are the labels that are less likely to be wrong. For comparison with

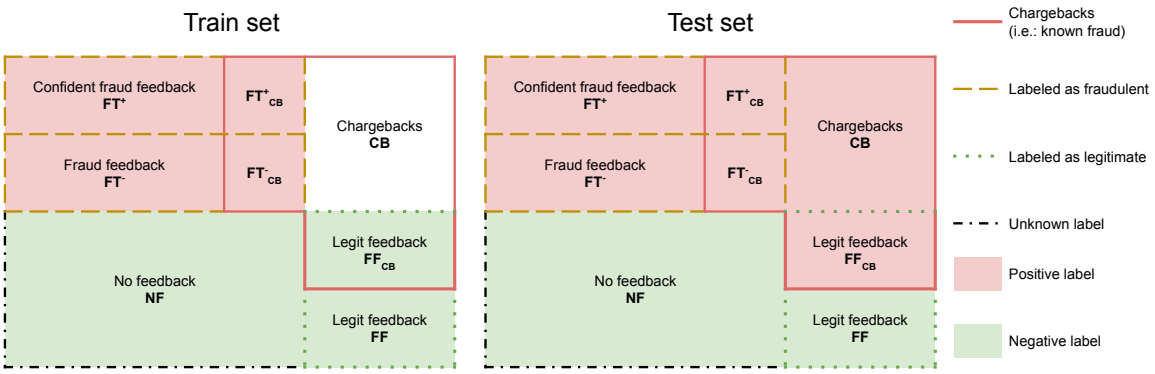


Figure 6: Label scenarios used for only analysts' feedback setup.

the scenario where only analysts' feedback is used, we will also present experiments in this idealized opposite scenario where we only have access to (positive) chargeback labels.

Figures 6, 7 and 8 illustrate how the label scenarios are used in setups 1, 2 and 3, respectively. The areas highlighted in green are the label scenarios for which we attribute a negative/legit label and in red a positive/fraud label. The white space represents transactions that are removed from the dataset.

In setup 1, where only analysts' feedback is used (figure 6), we exclude the CB labels from the train set. In addition, we assign the FF_{CB} labels as legitimate. We do this because, even though these are known to be fraudulent (since a chargeback has been issued), an analyst made a mistake that we want to simulate (i.e., an analyst's false-positive label). In the test set, assign all labels to their most likely true label (i.e., chargebacks override analyst labels). In both sets and all of the setups we assume the NF labeled transactions are legitimate, since this is already a common practice in fraud detection for data that is old enough so that all fraud complaints are likely to have occurred.

In setup 2, where only confident analysts' feedback is used (figure 7), we exclude the usage of the CB, FT^- and $FT_{CB}{}^-$ labels in the train set. Only labels that display confidence in the analyst's decision (FT^+ and $FT_{CB}{}^+$) are used for the positive labels. The remaining transactions that are either labeled negative by analysts or in NF are considered legitimate. The test set is similar to the one used in setup 1. The only difference is that it does not use positive labels that have only been reviewed by analysts with low confidence (FT^-)

Setup 3 is where only chargebacks are used. For both the train and the test set, all positive labels for which there is no chargeback (FT^- and FT^+) are excluded. In this setup, since we are not trying to simulate the analyst's mistake, we assign the FF_{CB} scenario transactions as fraudulent. Therefore, we only attribute negative labels to the NF and FF scenarios.

4.2 TIME FOLDS

In order to analyze the effectiveness and temporal stability of AL we split each of the datasets in various time folds. This allows us to observe if concept drift[1], which is common to occur in fraud

1 Concept drift is the phenomenon that occurs when the statistical properties of data change over time [Gama et al. (2014)].

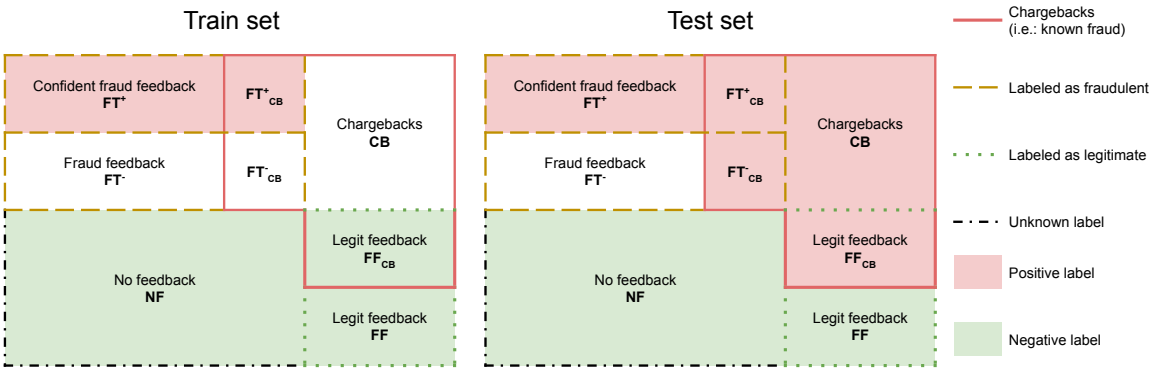


Figure 7: Label scenarios used for only confident analysts' feedback setup.

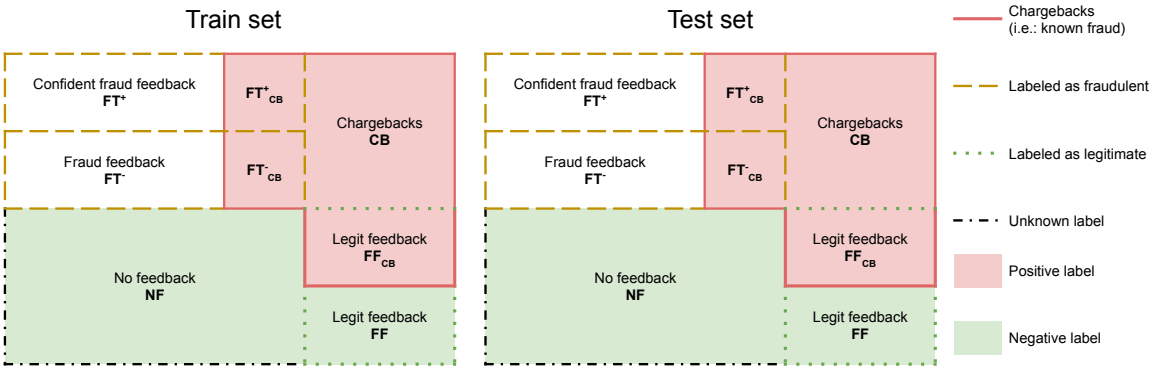


Figure 8: Label scenarios used for the chargeback only setup.

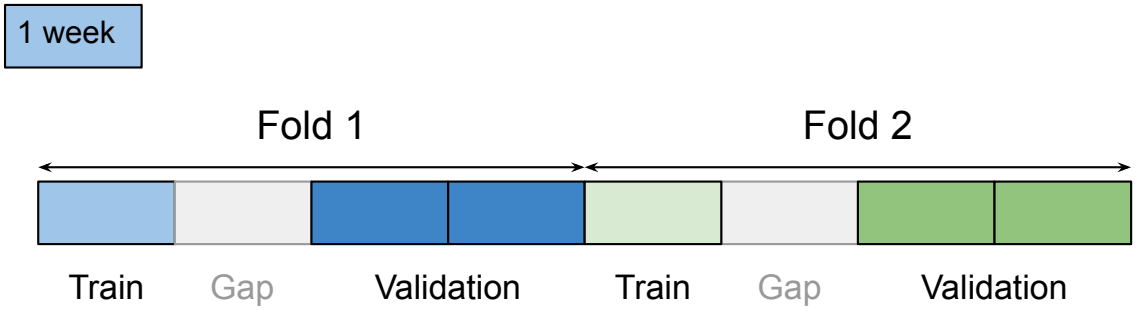


Figure 9: Representation of two simulation periods for two experiments (Fold 1 for the first experiment and Fold 2 for the second experiment).

datasets, has an effect on the results and to assess if an AL strategy under-performs in a particular period exceptionally.

We partitioned each dataset into several intervals of two weeks and used every pair of consecutive intervals as a train-test pair. The unlabeled data pool starts empty and grows over time until it reaches 7 days of data, which results in a week-long gap between the first (train) and second (test) period. The main reason for this gap was so allow for the option to run some experiments up to two weeks, which is also a typical timescale for chargeback labels to start arriving for various datasets. We will show, however, some experiments with an AL training of two weeks for the third banking dataset, which has an extremely high imbalance.

This time-folding setup is depicted in figure 9, where two train-test split folds are represented.

4.3 MODELS

In the next sections, we will see that a suitable approach to measure the performance of an AL experiment is given by aggregating a sequence of performance metric values obtained from several ML models produced as the labeled pool grows. Hence, this evaluation may be sensitive to the choice of the model.

In the fraud detection domain, given the large scale of the data it is common that ML models are trained with hundreds of thousands of transactions. Training a model in an AL context is very different from this, since the goal is to train with as little data as possible. For that reason a model with a certain choice of hyper-parameters that is trained on a large training data set and has a high performance on a test set, might not be the best choice for the same dataset when trained with AL.

In light of this, we will present a small study of ML models and their hyper-parameters in order to learn what are reasonable choices that perform well in our evaluation. Our goal is not only to find appropriate machine learning models for the evaluation of our experiments, but also to understand the impact of different levels of model regularization on performance within AL (more regularization means that a model is more generalist and therefore less prone to over-fitting).

The models and the values for the hyper-parameters we will use for this experiment are as follows (the values in brackets are the different candidate values we will be experimenting with separately):

- Random forest:

  - number of trees: $\{200, 1000\}$
  - maximum depth of the trees: $\{3, 5\}$
- Support vector machine:
  - Regularization parameter C: $\{1, 10\}$
  - Kernel: {Radial basis function}
- Multi-layer perceptron:
  - Number of hidden layers: $\{1\}$
  - Number of neurons in hidden layer: $\{50\}$
  - Regularization parameter α: $\{0.01, 0.5\}$
- Gradient boosting ensemble with decision trees
- Gaussian Naive Bayes

The proposed models and hyper-parameters cover various levels of regularization levels, which is specially relevant due to the challenge of training with such small quantities of data. For each model with varying hyper-parameters, the level of regularization increases when:

- Random forest — the number of trees increases or the maximum depth decreases.
- Support vector machine — the value of C decreases.
- Multi-layer perceptron — the value of α increases.

Even though this is not a very extensive experimentation, there is some diversity in the models used and their sets of hyper-parameters. Results are presented in section 5.3.

4.4 Performance metrics

We now define the performance metrics that will be used to observe an measure the quality of a single AL configuration, as well metrics to allow the comparison between various large-scale sets of experiments across time folds and datasets.

4.4.1 *Learning curves*

The performance of a single AL run is tracked by the performance of the model as the labeled pool grows (explained in section 3.5). In figure 10 we present a visualization of the model performance (vertical axis) as a function of the size of the labeled pool (horizontal axis). The represented performance values are the target metric[2] (e.g., recall at some value of false positive rate), obtained by training the model at each iteration and making predictions on the test set. We can observe that this run's performance keeps increasing until the labeled pool has around 4000 transactions, where it starts to plateau.

However, given the stochastic nature of most AL policies, we cannot draw conclusions from a single run. For that reason, we run all experiments in our study with 35 different random seeds, to

2 In this dissertation we do not specify the target metrics being used due to confidentiality issues.

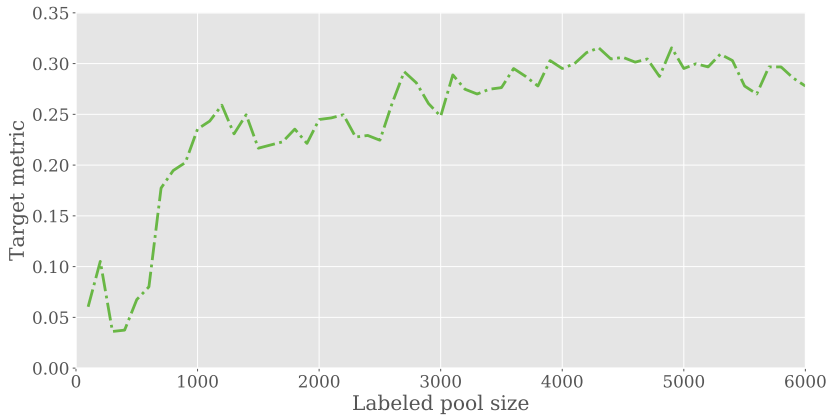


Figure 10: Visualization of the performance throughout a single AL run. This example was taken from a random policy.

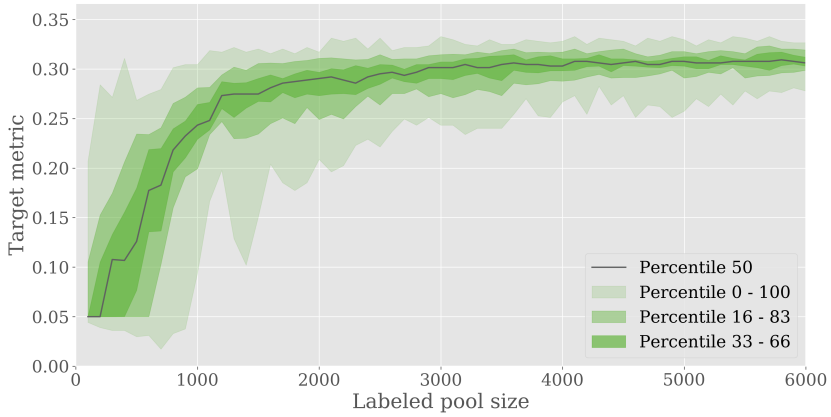


Figure 11: Visualization of the variance of the performances throughout 35 AL runs with different random seeds. This example was taken from the same experiment as in figure 10

obtain a measure of the variance of the learning curves.[3] This allows us to compare the stability of different policies. In figure 11 we present an example of a visualization of the aggregation over the 35 runs. For each number of instances in the labeled pool, several percentiles of the performance were computed (using the 35 values) and the bands between the intervals of such percentiles were plotted. This allows us to clearly observe the median (50^{th} percentile), the worst and best case scenarios (0^{th} and 100^{th} percentiles) and the positions of some intermediate percentiles.

3 The choice of 35 seeds provides a reasonable trade off between having a good chance of collecting at least one point in the extreme tails of the distribution of values (for each point of the learning curve) and keeping the number of repetitions small enough to run the experiments in a practical amount of time. In particular, for 35 runs there is an $\sim 84\%$ probability to observe a point in the 5% upper or lower tail of the distribution - see, e.g., equation 3 in Pinto et al. (2019)

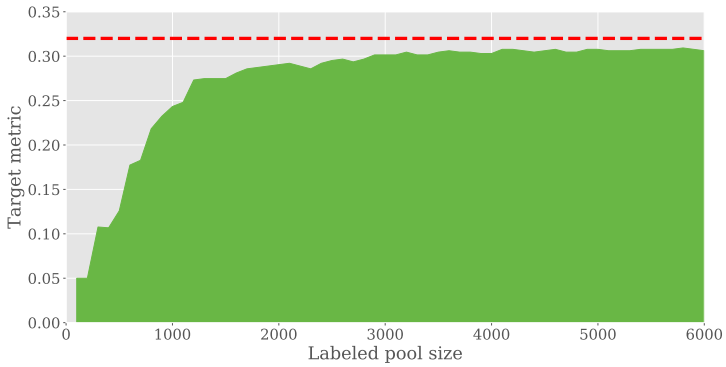


Figure 12: Visual representation of the KPI based on the area below the 50th percentile of the learning curves. The larger this KPI, the better the policy because it will have achieved the plateau earlier and its value will be higher. The horizontal dashed line represents the full train data baseline value.

4.4.2 *Key performance indicators (KPIs)*

Due to the large number of experiments in this study, it is not feasible to analyze and compare all the different results by inspection of the corresponding visualizations. To be able to summarize the performance of a single experiment and to be able to compare various experiments, we introduce two KPIs that allow us to sort experiments:

- Area under the 50th percentile — This metric is represented in figure 12. It is the area below the 50th percentile line of the plot in figure 11 (the horizontal dashed line is the full train data baseline value, which will be explained further in this section). It gives us a number that represents the median performance of the experiment. The higher this value is, the better the configuration.
- Area between the 10th and 90th percentiles — This metric is represented in figure 13. It is the area between the 10th and 90th percentile lines of the plot in figure 11. Since it represents the difference between the almost worst and best scenarios, it gives us a number to represent the variance in the performance of the runs. The lower this value is, the lower is the variance of experiment, which is good.

The direct comparison of area under the 50th percentile values between experiments provides a good indication of which performs better. However, when observing these values individually, we cannot tell how good they are, e.g., it could be the case that all experiments had bad results simply because the dataset is hard on the given fold. A more meaningful metric is obtained by normalizing these metrics by the corresponding value for a model simply trained with the full two weeks of train data instead of using AL. We call this the *full train data baseline*. This is conjectured to be the best possible performance one can obtain on the test fold. In figure 12 this is displayed as a horizontal dashed line. By dividing the area under the 50th by the area of the rectangle below the full train data baseline value, we obtain a ratio of the AL performance by its optimal performance. This normalization also makes it easier to compare KPIs across different test folds.

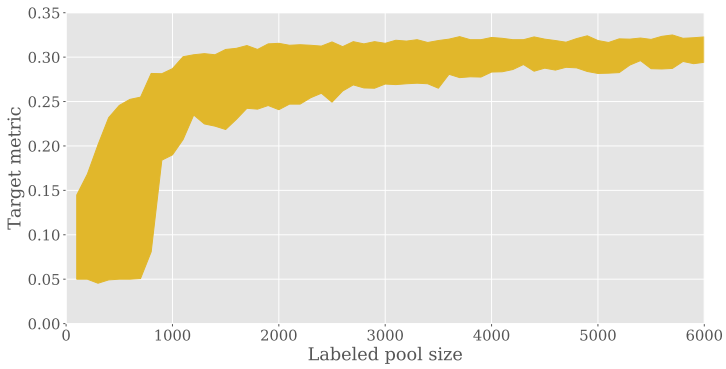


Figure 13: Visual representation of the KPI based on the area between the 10^{th} and 90^{th} percentiles of the performances. The smaller it is the most stable the policy is because its learning curve will have a lower variance.

5

EXPERIMENTAL RESULTS

In section 3, the set of approaches used to find the strongest AL configurations were described. The current Chapter describes the experiments performed towards that goal and the corresponding results. The experiments were designed to try to answer the following questions:

1. Without any of the improvements we propose in previous chapters, how do the proposed AL policies work and which perform best?
2. Which of the proposed data pre-processing methods (section 3.4.1) is more effective for AL?
3. Which ML model and hyper-parameters (section 4.3) are more suitable for training with the small amounts of data in AL?
4. Is our proposal of a three-phase framework for AL (section 3.7) beneficial?
5. Can a combination of policies, using our proposed methods (section 3.8), be better than its isolated parts?
6. Having narrowed down the best configurations by answering the previous questions, how do the AL policies behave across the several fraud business cases (section3.1)?

In all the experiments we will present, we only provide results with a fixed batch size of 100. We repeated several of the various experiments, at an early stage, with smaller batch sizes (10 and 50) and did not find differences in performance that were significant enough to justify the heavier computational cost of using smaller batch sizes (which for a batch size of 10 would increase the run time by 10 times).

In this Chapter, we start by presenting a set of preliminary experiments on a single dataset, to understand the landscape of AL policies that we have proposed to study (using a simple two-phase setup where the AL policy to study is preceded by a cold policy initialization). Then, we present two dedicated studies to investigate the effect of varying two parameters of the experiments that are not directly related to AL, namely the data pre-processing methods (Section 5.2) and the ML model (Section 5.3). This provides validation for our earlier choices. After these preliminary studies, we move on to assess the usefulness of our main proposals on the same dataset, namely: the three-phase AL framework (Section 5.4), the policy divergence diagnostics, and the policy combination methods (Section 5.5). Finally, after narrowing down the sequences of policies to a set of representative cases to investigate, we conduct a set of large-scale experiments on various datasets to verify the best performing policies in each of the fraud business cases we proposed to study (Section 5.6).

Cold policy	*Hot policy*	Fold 1 Normalized Area	Fold 1 Rank	Fold 2 Normalized Area	Fold 2 Rank	**AVG rank all**	**AVG VAR all**
Isolation forest outlier detection	Uncertainty sampling	0.81	2	0.83	1	**1.5**	0.22
Isolation forest outlier detection	Isolation forest ODAL	0.78	4	0.74	2	**3.0**	0.23
Elliptic envelope outlier detection	Uncertainty sampling	0.81	1	0.63	8	**4.5**	0.29
Isolation forest outlier detection	Expected model change	0.81	3	0.69	6	**4.5**	0.20
Isolation forest outlier detection	Elliptic envelope ODAL	0.76	6	0.74	3	**4.5**	0.22
Isolation forest outlier detection	Query by committee	0.75	8	0.73	4	**6.0**	0.27
Isolation forest outlier detection	None	0.68	11	0.70	5	**8.0**	0.27
Elliptic envelope outlier detection	Isolation forest ODAL	0.75	7	0.58	12	**9.5**	0.27
Random	Uncertainty sampling	0.64	12	0.64	7	**9.5**	0.44
Elliptic envelope outlier detection	Expected model change	0.77	5	0.56	15	**10.0**	0.21
Elliptic envelope outlier detection	Elliptic envelope ODAL	0.72	9	0.56	14	**11.5**	0.28
Random	Isolation forest ODAL	0.63	14	0.60	11	**12.5**	0.40
Random	Expected model change	0.63	15	0.61	10	**12.5**	0.45
Random	None	0.59	16	0.61	9	**12.5**	0.40
Random	Query by committee	0.63	13	0.56	13	**13.0**	0.40
Elliptic envelope outlier detection	Query by committee	0.71	10	0.51	17	**13.5**	0.29
Random	Elliptic envelope ODAL	0.58	17	0.53	16	**16.5**	0.39
Elliptic envelope outlier detection	None	0.47	18	0.49	18	**18.0**	0.39

Table 2: Results of the preliminary experiments.

5.1 PRELIMINARY EXPERIMENTS

Before investigating in detail the impact of specific components of the AL system, we present a set of preliminary experiments to understand the basic behavior of the various policies we proposed to study. In order to keep this exploration simple and to be able to explore many combinations, we only use two time folds of only one (banking) dataset. To isolate as much as possible the behavior of each policy, these experiments were only performed in two phases: a cold policy (an AL policy that does not require a labeled pool, which we can never remove in our problem setup) and a hot policy. This simplification also allows us to test further combinations, in particular more than one policy implementation of a given type.[1] The switch between policies happens when the third fraudulent instance is found in the labeled pool. This is because we want to fit the policies with several positive cases while allowing to perform the switch at an early stage. For the data preprocessing (section 3.4.1), we use the option where the autoML tool generates many features and then the number of features is reduced with PCA. We chose this option because it takes into account correlations between features in a multi-variate way (i.e., several features at a time). Regarding the ML model we use a random forest with 200 trees and a maximum depth of 3 nodes, because we expect it to provide a good level of regularization for the small amounts of data used in AL, while letting us run experiments in a manageable amount of time. Since these are preliminary experiments, we delay the validation of these choices to the next sections.

The results are presented in table 2. In this table, the rows are candidate policies (i.e., the cold/hot policy combinations) and the columns are time folds. The values in the *"Normalized Area"* columns

1 These experiments were conducted at an earlier stage of the project that was prior to the experiments leading to the 3 phases strategy explained in section 3.7 – presented later in this section.

are the area under the 50^{th} percentile *KPI* normalized by the area below the full train data baseline (introduced in section 4.4.2) . The full train data baseline is the performance of the model when trained with all of the transactions in the first two weeks of the time fold (while the AL runs are accessing only the first week). Here we introduce this normalization to facilitate interpreting the policy performance relative to the full train data baseline. This also allows clearer comparisons between different time folds (which may have very different full train data baselines due to concept drift). For each time fold an extra column is added with the ranks of the candidates, according to the normalized area for the fold. We also compute an average of ranks over the two folds that is used to sort the table, so that the candidate policies are ordered by performance. Finally, we introduce a column (*"AVG VAR ALL"*) with the average variance over the folds (i.e., area between the 10^{th} and 90^{th} percentiles *KPI*). The variances were also normalized by the same full train data baseline area.

Regarding the results in table 2, we observe the following:

- Using an outlier detection algorithm as cold policy is almost always better than using a random policy, especially the isolation forest.
- Within experiments with the same cold policy, the variances are mostly similar. This means that it is the cold policy, and not the hot policy, that is having most of the impact on the variance.
- The random cold policy introduces much more variance in the experiments than the outlier detection ones.
- For each cold policy, using uncertainty sampling is always the best hot policy and isolation forest ODAL is the second best.

5.2 DATA PRE-PROCESSING

The data pre-processing methods we choose to apply in our experiments will affect not only the instances that the AL policies select, but also the performance the ML models obtained. Therefore, it is desirable to validate if the previously used pre-processing method is good and if it should be used in the remaining experiments. In section 3.4.1 several pre-processing methods were presented. We proceed by presenting a set of methods combinations:

- Execution of an autoML pipeline containing profiles with 6 different time windows, followed by PCA dimensionality reduction to decrease to approximately 90 features.
- Execution of the same autoML pipeline as above, but then applying a pair-wise correlation filter in order to produce approximately 90 features.
- Execution of an autoML run with fewer profiles (4 time windows only and removal of some grouping entities according to domain knowledge, to remove features that are most likely redundant).

This component is expected to affect all policies in a similar way (it is not an AL component). Therefore, this simple study should suffice to select a reasonable preprocessing method. A more detailed study of the effect of the pre-processing hyper-parameters is beyond the scope of this study and will be left for future work. For simplicity, these experiments were only performed with a reduced set of policies and datasets. Only two time folds of data were used from a banking dataset.

Feature engineering time windows	Feature selection	Uncertainty sampling				Elliptic envelope ODAL				Random				Query by committee				AVG rank all	AVG VAR all
		Fold 1		Fold 2		Fold 1		Fold 2		Fold 1		Fold 2		Fold 1		Fold 2			
		Area	Rank	Area	Rank	Area	Rank	Area	Rank	Area	Rank	Area	Rank	Area	Rank	Area	Rank		
7 days, 1 day, 12 hours, 1 hour, 30 mins, 5 mins	PCA (90 features)	0.21	1	0.27	1	0.19	1	0.22	2	0.18	1	0.25	1	0.19	1	0.25	1	**1.13**	**0.15**
7 days, 1 day, 1 hour, 5 mins	Domain knowledge (201 features)	0.15	3	0.26	2	0.12	3	0.22	1	0.15	2	0.23	2	0.13	2	0.24	2	**2.13**	**0.18**
7 days, 1 day, 12 hours, 1 hour, 30 mins, 5 mins	Pair-wise correlation selection (90 features)	0.17	2	0.25	3	0.13	2	0.19	3	0.13	3	0.22	3	0.13	3	0.20	3	**2.75**	**0.18**

Table 3: Results of the experiments with different pre-processing.

Regarding the AL policies, we narrowed them down to: uncertainty sampling, elliptic envelope ODAL, a random policy and query by committee[2]. And again, for simplicity, we used a random forest with 200 trees and a maximum depth of 3 nodes.

The results are presented in the table 3. In this table, the rows correspond to the candidate methods (i.e., the data pre-processing pipelines) and the columns correspond to metrics for specific fold/AL policy combinations.

The values in the *"Area"* columns are the area under the 50^{th} percentile KPI (introduced in section 4.4.2). For each fold/AL group an extra column is added with the ranks of the candidate methods, according to the area under the 50^{th} percentile KPI. We also compute an average of ranks (used to sort the rows of the table) so that the candidates are ordered by average rank of the performance metric. We use the average ranks and not the average KPIs, for this and all further experiments, because our main goal is to understand which AL policies most commonly outrank the others. By averaging the KPIs this measure would be too susceptible to outlier KPIs and to changes in the overall scale of the performance metric due to drift across time folds. Finally, we also introduce a column (*"AVG VAR ALL"*) with the average variance (i.e., average along each row of the area between the 10^{th} and 90^{th} percentiles KPI) for each fold/AL policy combination).

Regarding table 3, we observe the following:

- All candidates exhibit similar low variance.
- The data pre-processing pipeline with 6 time windows and pair-wise correlation selection is almost always the worst performer.
- The method with PCA feature selection, which we used for the preliminary experiments, is almost always the best among the candidates.

Since explainability is not a priority at such an early stage of the project, we choose to keep using the best performing method for the remaining experiments, despite containing PCA feature selection, which produces less interpretable features.

2 At this early stage exploration we used AL with only with 2 phases (the 2 versus 3 phase setup is explained in section 3.7).

		Uncertainty sampling				Elliptic envelope ODAL				Random				Query by committee				AVG rank all	AVG VAR all
		Fold 1		Fold 2		Fold 1		Fold 2		Fold 1		Fold 2		Fold 1		Fold 2			
Model	*Parameters*	Area	Rank	Area	Rank	Area	Rank	Area	Rank	Area	Rank	Area	Rank	Area	Rank	Area	Rank		
Random Forest	#trees=1000 max depth=3	0.23	1	0.29	1	0.19	1	0.23	1	0.19	2	0.26	1	0.21	1	0.26	1	**1.13**	**0.13**
Random Forest	#trees=1000 max depth=5	0.21	2	0.28	2	0.18	2	0.22	2	0.19	1	0.25	2	0.21	2	0.24	2	**1.88**	**0.14**
Random Forest	#trees=200 max depth=3	0.20	3	0.25	3	0.18	3	0.21	3	0.18	3	0.24	4	0.19	3	0.22	3	**3.13**	**0.14**
SVM	C=1 kernel=rbf	0.05	6	0.05	8	0.14	4	0.14	4	0.11	4	0.24	3	0.14	4	0.10	5	**4.75**	**0.23**
Multilayer perceptron	hidden layer size=50 alpha=0.5 max #iterations=2000	0.07	5	0.11	4	0.09	6	0.11	5	0.07	6	0.13	5	0.08	7	0.11	4	**5.25**	**0.08**
Gradient boosting classifier	None	0.08	4	0.09	5	0.07	7	0.07	7	0.07	5	0.10	6	0.09	6	0.08	6	**5.75**	**0.13**
SVM	C=10 kernel=rbf	0.04	8	0.06	6	0.10	5	0.09	6	0.04	8	0.08	7	0.10	5	0.07	7	**6.50**	**0.23**
Multilayer perceptron	hidden layer size=50 alpha=0.01 max #iterations=2000	0.05	7	0.05	7	0.05	9	0.05	9	0.05	7	0.07	8	0.05	9	0.06	8	**8.00**	**0.05**
Gaussian Naïve Bayes	None	0.03	9	0.05	9	0.06	8	0.06	8	0.04	9	0.05	9	0.05	8	0.06	9	**8.63**	**0.05**

Table 4: Comparison of the AL performance for several models.

5.3 MODEL COMPARISONS

The ML model used for our experiments must be suited to an AL setup. Particularly, this model is to be fitted on small amounts of data, therefore the level of regularization is especially relevant. In section 4.3, we proposed a set of models to study this dependence on the choice of the evaluation model. The results of the corresponding experiments are present in table 4. In this table, the rows are evaluation models (and the respective hyper-parameters) and the columns contain groups with specific time fold/AL policy combinations.

Similarly to the results from the previous section, the values in the *"Area"* columns are areas under the 50^{th} percentile *KPI* (introduced in section 4.4.2). For each group we include a column with the ranking of the models, according to the area under the 50^{th} percentile. We also compute the average of all ranks (in the *"AVG rank all"* column) which is used to sort the rows of the table, so that the models are ordered from higher to lower performance. Finally, we also introduce a column (*"AVG VAR ALL"*) with the average variance over all groups (i.e., the average area between the 10^{th} and 90^{th} percentiles *KPI*).

Similarly to the data pre-processing pipeline experiments (section 5.2), this provides a study of the effect of changing the ML model and its hyper-parameters to determine the best choice for the remaining experiments, so the time folds and AL policies used are the same.

In the results table we can observe that:

- The more regularization the better — models of a fixed type with hyper-parameters values that provide greater regularization always perform better (the relevance of this topic was also discussed in section 4.3). This is observable in the performance gain of the following models:
  - Random forests, with either a larger number of trees (for fixed maximum tree depth) or a smaller maximum depth (with fixed, and large, number of trees).
  - *SVM*, with a smaller value of the inverse regularization parameter C (smaller values equate to larger regularization).
  - The multilayer perceptron with a larger value of α, which also results in stronger regularization.
- Some models have considerably lower performance and lower variance, such as the Gaussian Naïve Bayes and the multilayer perceptron. In those cases, the lower variance is justified by the fact that the performance never reaches larger values.
- The random forest algorithm, despite the hyper-parameters used, is generally the best performer.
- Within the random forests, the top 3 models, the differences in the hyper-parameters have a moderate impact on the absolute value of the areas.

We consider that the moderate positive impact of the random forest's hyper-parameters on AL performance, especially using a much larger number of trees, does not outweigh the additional computational complexity, which would result in much longer training time. For this reason, in the final experiments we will continue using a random forest with 200 trees and a maximum depth of 3 nodes.

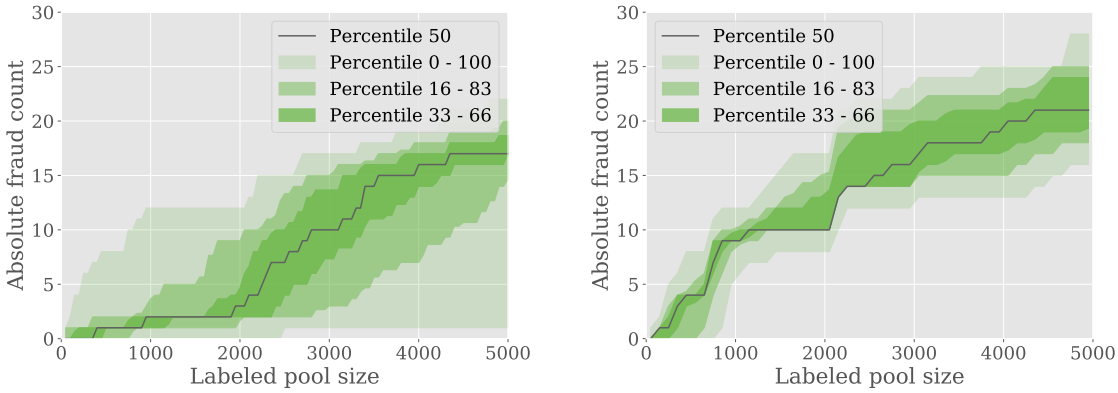
5.4 AL with 3 phases

As explained in section 3.7, most of the hot policies we are using resort to the labels in the labeled pool. This prevents them from starting sooner if there are only labels from one class in the labeled pool (i.e., there must be both negative and positive class instances). This can be a problem for datasets with a high class imbalance, like in the fraud domain, because there will be a delay in the switch from cold to hot policy. Motivated by this observation, we introduce an intermediate stage with a policy that resorts to the labeled pool but not to its labels, so that the collected labels can start being exploited earlier. We call this the *warm-up policy* stage. In order to evaluate the benefits of this approach, we conduct a simple set of experiments.

The only policies that can be used as warm-up policies are of the *ODAL* type (see section 3.6.1), since they do not use label values. We discard the usage of the *elliptic envelope ODAL*, because in section 5.1 we observed that it under-performed the *isolation forest ODAL*.

In section 5.1, we also concluded that using random as a cold policy instead of outlier detection increases variance and decreases median performance. However, in a three-stage strategy the cold policy is only used once on the first (typically small) batch when the performance is still low regardless of the cold policy method. Since the random policy is computationally much lighter and it is unbiased[3], we argue that it is more suitable for the first batch.

3 Despite the good results for the outlier detection warm-up for this banking dataset, it is not clear if the same type of bias will help in other datasets.



(a) Two phases

(b) Three phases

Figure 14: Count of positive instances found throughout an AL process with two and three phases.

		Fold 1		Fold 2		AVG rank all	AVG VAR all
Warm-up policy	*Hot policy*	Normalized Area	Rank	Normalized Area	Rank		
Isolation forest ODAL	Uncertainty sampling	0.80	1	0.94	1	**1.0**	**0.15**
Isolation forest ODAL	None	0.75	2	0.82	2	**2.0**	**0.13**
Isolation forest ODAL	Query by committee	0.72	3	0.80	3	**3.0**	**0.18**
Isolation forest ODAL	Expected model change	0.69	4	0.75	4	**4.0**	**0.14**
None	Uncertainty sampling	0.64	5	0.73	5	**5.0**	**0.43**
Random	None	0.51	6	0.70	6	**6.0**	**0.43**

Table 5: Results of the experiments with the 3 phases.

The plots in figure 14 represent the number of positive instances in the labeled pool throughout the AL process, for both two and three phases. For the experiments corresponding to this figure we use a cold policy that is random. For the two phase AL the policy switch happens sometimes when the labeled pool has 2500 instances when the first fraud label is found, which is why the median learning curve rises much slower (the switch only occurs when the first positive instance is found), i.e., halfway through the experiment. With three phase AL the *ODAL* policy (the warm-up policy) starts being used immediately after the first batch. This policy is not only much better than random at constructing an effective labeled pool, but it also makes it more likely to find positive instances and switch to the hot policy earlier.

The comparison of results of the experiments can be observed[4] in table 5. In this table, the rows are candidates (i.e., the cold/hot policy combinations) and the columns are time folds. The structure of the table and the metrics in figure 5 is the same as in table 2 (see discussion therein). The only difference is that we omit the cold policy because the first batch is always selected with the random policy.

4 Note that there are small differences in the values of the two phase uncertainty policy compared to table 2. This is because for the preliminary experiments we used the model proposed in section 5.3, but in these ones we use the random forest with 400 trees and a maximum depth of 3.

These experiments show that all experiments with three phases and the one only with ODAL (which always switches immediately after the first batch) have better performance than the experiments with two phases. Besides, the uncertainty sampling and random policies performed in a two-phase framework have much higher variance.

5.5 POLICY COMBINATION

In section 3.8 we introduced the idea of creating methods to combine AL policies. This could be useful if a set of AL policies follow sufficiently different principles in a way that they complement each other. To investigate this conjecture, in this section we start by presenting results of experiments to test policy diversity using *policy divergence diagnostic* methods. Then we proceed to the experiments on policy combination to determine whether specific combination methods provide labeled pools that result in a model with a higher performance.

5.5.1 *Policy divergence diagnostic*

In section 3.8.1, we introduced two policy divergence diagnostic methods: one based on computing the difference between the labeled pools produced by the AL policies and another one based on the differences in the predictions made by the models trained with those pools.

We present results for these diagnostic methods comparing the following AL policies:

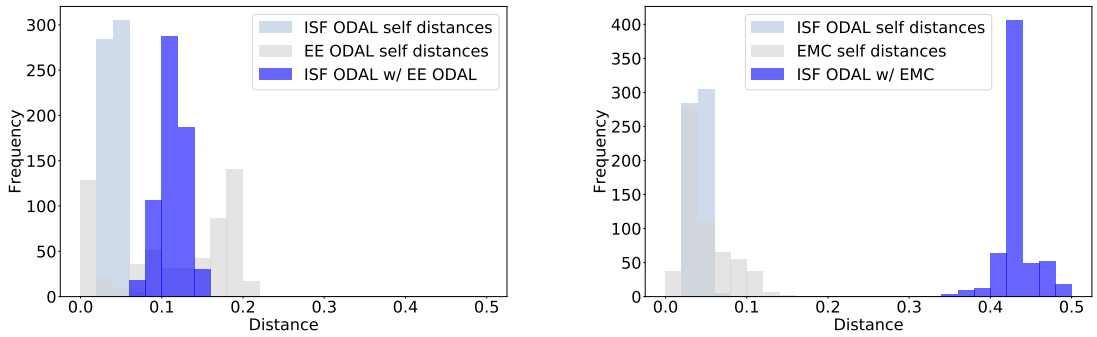
- Uncertainty sampling (*UNC*)
- Expected model change (*EMC*)
- Query by committee (*QBC*)
- Elliptic envelope ODAL (*EE ODAL*)
- Isolation forest ODAL (*ISF ODAL*)

Cluster frequency divergence

In this section we discuss results for the method that compares the features spaces of the labeled pools presented in section 3.8.1. Recall the main steps of the method:

1. train an unsupervised ML algorithm to cluster all the data to establish a binning (each cluster is a bin)
2. compare two labeled pools using the histograms of frequencies of the instances of each pool according to such binning (using the Jensen–Shannon divergence).
3. repeat for all pairs of available pools (equivalently, for each pair of seeds) from either policy and produce a distribution of divergence values.

The histograms in figure 15 provide examples of the outputs of this analysis. The x-axis represents the divergence metric (in this case, the cluster frequency divergence) between data pools, hence, the more to the right the greater the divergence. In each figure, three distributions of divergences are represented: the self-divergences distribution for each policy (i.e., the divergences between different



(a) Isolation forest ODAL (*ISF ODAL*) and elliptic envelope ODAL (*EE ODAL*).

(b) Isolation forest ODAL (*ISF ODAL*) and expected model change (*EMC*).

Figure 15: Distributions of divergences between policies (first fold) using the *cluster frequency* method.

labeled pools of the same policy with different random seeds) and the divergences distribution between the two different policies. The left figure is an example of a comparison between two policies which appear to not complement each other (isolation forest and elliptic envelope *ODAL*). We conclude this because the divergence between policies is distributed in a similar way as the self-divergence distributions of either policy. On the right, this is not the case when comparing isolation forest ODAL and expected model change, where the distribution of self-divergences has support in a domain of values that is much lower and well separated from the distribution of divergences between policies.

To quantify the differences between policies we present tables 6 and 7, with the average divergence between every pair of policies (including the self-divergences in the diagonal), for two time folds. The color scale provides a heat-map, with larger values in darker color. As expected, the self-divergences have much lower values (light green).

From these results we conclude the following:

- There are pairs of policies with higher distributions of divergences than the self-divergences (e.g., isolation forest *ODAL* with expected model change), thus indicating that they cover the feature space differently.
- The obtained values are consistent between time folds.
- Most AL policies appear to be complementary to each other, with the exception of the combination between isolation forest and elliptic envelope *ODAL*. This is expected, given that they are two policies of the same nature.
- Expected model change appears to be the method which most complements the other methods (this is easily visible since it is the most highlighted row in the heat-map).

In conclusion, this metric indicates that, from a feature space perspective, there might be benefits in combining some policies.

	Isolation forest ODAL	Elliptic envelope ODAL	Uncertainty sampling	Query by committee	Expected model change
Isolation forest ODAL	0.04	-	-	-	-
Elliptic envelope ODAL	0.11	0.12	-	-	-
Uncertainty sampling	0.22	0.21	0.07	-	-
Query by committee	0.22	0.2	0.22	0.12	-
Expected model change	0.43	0.44	0.29	0.44	0.05

Table 6: Average *cluster frequency* divergences between policies in the first time fold.

	Isolation forest ODAL	Elliptic envelope ODAL	Uncertainty sampling	Query by committee	Expected model change
Isolation forest ODAL	0.04	-	-	-	-
Elliptic envelope ODAL	0.09	0.09	-	-	-
Uncertainty sampling	0.17	0.2	0.06	-	-
Query by committee	0.15	0.15	0.2	0.15	-
Expected model change	0.38	0.39	0.24	0.36	0.06

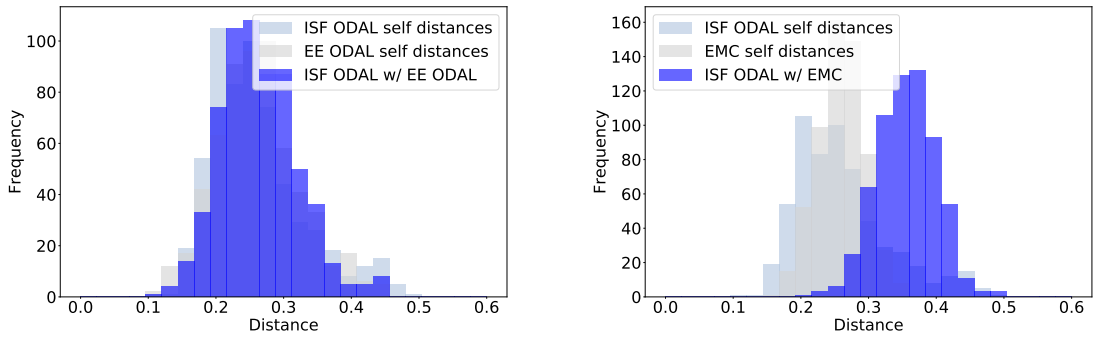
Table 7: Average *cluster frequency* divergences between policies in the second time fold.

Model disagreement divergences

In section 3.8.1, we proposed a method that computes the difference between two policies by how much their resulting models disagree on the predictions on the test set. The divergences is the ratio between the number of instances on which the models disagreed divided by the number of instance that at least one model classified correctly (see section 3.8.1 for further details).

Figure 16 shows examples of the distributions of the *model disagreement divergences*. The represented policies and fold are the same as in figure 15. These were produced, similarly to the previous section, by building the distribution of all possible pairs of comparisons to obtain the histograms. In these plots the approach to compare the distribution of divergences is similar to the one used in figure 15. The combination between the two *ODAL* methods results in very overlapping distributions, which indicates they are similar. The analysis of the combination between isolation forest *ODAL* and expected model change with the *cluster frequency divergence* resulted in a large divergence. With the *model disagreement divergence*, the difference between policies is still clear, even though it is a lot less pronounced.

In tables 8 and 9 (first and second fold, respectively), we show average values computed with the same pairs of comparisons used in the previous section for the *cluster frequency* method. We can see that there is a lot of overlap between the three distributions for most pairs of policies (i.e., the two self-divergences and the divergence between the two policies). Nevertheless, these results are still partially consistent with the ones obtained with the *cluster frequency* method: the *ODAL* policies still appear to not complement each other and the expected model change method also appears to complement most other methods. However, the first and second fold appear to be more inconsistent,



(a) Isolation forest ODAL (*ISF ODAL*) and elliptic envelope ODAL (*EE ODAL*).

(b) Isolation forest ODAL (*ISF ODAL*) and expected model change (*EMC*).

Figure 16: Distributions of divergences between policies (first fold) using the *model disagreement* method.

	Isolation forest ODAL	Elliptic envelope ODAL	Uncertainty sampling	Query by committee	Expected model change
Isolation forest ODAL	26.0%	-	-	-	-
Elliptic envelope ODAL	26.4%	26.2%	-	-	-
Uncertainty sampling	30.0%	29.0%	25.4%	-	-
Query by committee	35.3%	34.4%	29.8%	28.0%	-
Expected model change	35.6%	34.5%	33.3%	37.0%	25.9%

Table 8: Average *model disagreement* divergences between policies in the first time fold.

	Isolation forest ODAL	Elliptic envelope ODAL	Uncertainty sampling	Query by committee	Expected model change
Isolation forest ODAL	26.0%	-	-	-	-
Elliptic envelope ODAL	28.3%	27.6%	-	-	-
Uncertainty sampling	26.9%	26.7%	18.7%	-	-
Query by committee	33.1%	32.5%	31.0%	32.5%	-
Expected model change	29.6%	29.2%	28.1%	34.3%	19.8%

Table 9: Average *model disagreement* divergences between policies in the second time fold.

e.g., in the first fold, uncertainty sampling and elliptic envelope ODAL have a good divergence compared to their self-divergence, however, in the second fold it is actually smaller than *ODAL*'s self divergence.

In conclusion, this metric, which is more focused on model decisions and hence the target label, does not seem to indicate very clear benefits in combining most pairs of policies.

5.5.2 *Policy combination experimental results*

In section 3.8 we introduced three different policy combination methods. We will now present experimental results using the following choices of parameters:

- Weighted ranking combiner — with weights $\{0, 0.25, 0.5, 0.75, 1\}$;
- Alternating combiner — this is a sub-type of the *weighted ranking combiner* with weights $\{0, 1\}$;
- Cascade combiner — with the first policy filtering $N = 300$ instances, from which the second policy selects a batch of 100.

Given the results obtained in the policy divergence diagnostics (section 5.5.1), the only policy we excluded from these experiments was the elliptic envelope *ODAL*, since it appears to be redundant compared to the isolation forest *ODAL*. All the experiments were performed in a three-stage setup using isolation forest *ODAL* as the warm-up policy (see section 3.7).

The results are presented in table 10. In this table, the rows are candidates, i.e., the policy type and, if it is a combiner policy, its components, and the columns are time folds. The metric columns provided and the color formatting of the values is similar to previous tables. We also include the best performing policy without combination, uncertainty sampling, for comparison.

The results show that uncertainty sampling alone for the hot policy, i.e., without any of the combination methods to combine hot policies, obtains the best average performance across these two time folds (even though it is not the best policy on the first fold by a small amount). One interesting observation is that the first seven combiners in the ranking all use uncertainty sampling. Also, expected model change, has predicted in section 5.5.1, was the best to complement uncertainty sampling.

In conclusion policy combination does not seem to bring much value, so it is discarded from further experiments.

5.6 Large scale experiments

Having narrowed down all configurations, we now present large-scale experiments for which the following parameters were used:

- Batch size = 100 (as in all other experiments).
- Cold policies (section 3.6):
  - Random policy.
  - Isolation forest outlier detection. This policy is used only as a (one-phase) baseline, for comparison, where an outlier detection method is applied to the unlabeled pool to select the outliers for labeling.

			Fold 1		Fold 2		AVG	AVG
Hot policy	*Policy 1*	*Policy 2*	Norm. Area	Rank	Norm. Area	Rank	**Rank**	**VAR**
Uncertainty sampling	None	None	0.68	3	0.76	1	**2.0**	**0.05**
Weighted ranking combiner	Expected model change	Uncertainty sampling	0.69	1	0.73	5	**3.0**	**0.05**
Cascade combiner	Uncertainty sampling	Isolation forest ODAL	0.66	6	0.75	2	**4.0**	**0.04**
Alternating combiner	Isolation forest ODAL	Uncertainty sampling	0.65	7	0.74	4	**5.5**	**0.05**
Cascade combiner	Query by committee	Uncertainty sampling	0.66	5	0.73	6	**5.5**	**0.06**
Cascade combiner	Expected model change	Uncertainty sampling	0.64	10	0.74	3	**6.5**	**0.05**
Alternating combiner	Expected model change	Uncertainty sampling	0.69	2	0.72	12	**7.0**	**0.05**
Alternating combiner	Query by committee	Uncertainty sampling	0.64	11	0.72	8	**9.5**	**0.05**
Alternating combiner	Expected model change	Isolation forest ODAL	0.67	4	0.71	16	**10.0**	**0.06**
Weighted ranking combiner	Query by committee	Uncertainty sampling	0.65	9	0.72	11	**10.0**	**0.05**
Weighted ranking combiner	Expected model change	Isolation forest ODAL	0.64	13	0.72	10	**11.5**	**0.04**
Cascade combiner	Uncertainty sampling	Query by committee	0.62	20	0.72	7	**13.5**	**0.06**
Weighted ranking combiner	Isolation forest ODAL	Uncertainty sampling	0.64	14	0.71	14	**14.0**	**0.05**
Alternating combiner	Expected model change	Query by committee	0.65	8	0.70	20	**14.0**	**0.05**
Weighted ranking combiner	Expected model change	Query by committee	0.64	12	0.70	18	**15.0**	**0.06**
Cascade combiner	Isolation forest ODAL	Uncertainty sampling	0.63	16	0.71	15	**15.5**	**0.05**
Cascade combiner	Uncertainty sampling	Expected model change	0.61	22	0.72	9	**15.5**	**0.05**
Cascade combiner	Expected model change	Isolation forest ODAL	0.63	15	0.71	17	**16.0**	**0.05**
Cascade combiner	Isolation forest ODAL	Expected model change	0.60	24	0.71	13	**18.5**	**0.05**
Cascade combiner	Isolation forest ODAL	Query by committee	0.63	17	0.64	24	**20.5**	**0.06**
Weighted ranking combiner	Isolation forest ODAL	Query by committee	0.62	19	0.64	23	**21.0**	**0.06**
Alternating combiner	Isolation forest ODAL	Query by committee	0.62	18	0.63	25	**21.5**	**0.07**
Cascade combiner	Query by committee	Isolation forest ODAL	0.62	21	0.65	22	**21.5**	**0.07**
Cascade combiner	Expected model change	Query by committee	0.59	25	0.70	19	**22.0**	**0.06**
Cascade combiner	Query by committee	Expected model change	0.60	23	0.69	21	**22.0**	**0.06**

Table 10: Results comparison of policy combinations.

- Warm-up policies:
  - Isolation forest ODAL, is used both as the warm-up learner, but also as a single policy (after the random initialization).
  - Expected Model Change, Query By Committee and Uncertainty Sampling (w/ uncertainty defined as 0.5 threshold). To validate our three-phase framework we include two-phase experiments with these policies for comparison.
- Hot policies:
  - Uncertainty Sampling with three different uncertainty definitions proposed in 3.6.4, respectively: the models scores closest to 0.5; the models scores closest to the fraud percentile and the epistemic uncertainty.
  - Query By Committee and Expected Model Change.
- Switching criterion:
  - The switch between the cold and warm-up policy happens immediately after the first iteration (once the labeled pool is not empty).
  - Between the warm-up and the hot policy the switch happens when the first fraud is found (except for Query By Committee that requires at least two fraudulent instances).
- The initial unlabeled pool size corresponds to one day of data.
- Stopping criteria — the AL process stops running when seven days have passed (one day to train the preprocessing pipeline plus six additional days as, explained in section 3.5.1). The exception is the case of *Bank 2*, which has an extremely low number of positive instances, leading us to extend this period to 13 days.
- Labelers — we use one simulated labeler that reviews 1000 transactions per day, which makes the final labeled pools always have 6000 transactions, except for *Bank 2*, which finishes with 13000 transactions.

We can observe how the sequence of policies is specified in the results tables by inspecting one of the first ones (to be discussed in further detail). For example, in table 11 the first three columns display, respectively for every row, the cold, warm-up and hot phases/policies. When the keyword *None* is used, it means that the corresponding phase is absent, and the sequence of policies is for a one-phase or two-phase experiment.

As stated in section 4.1, we will use five time-folds for each of the six different groups of experiments (for the various business cases explained in section 3.1):

1. *Bank 1 w/ card-not-present filter* — the same use case presented in previous experiments but removing transactions that involve a payment with a physically present card. This filter is a common one in fraud detection, since fraud is often less probable when a card is present (we provide this new sub-case for comparison).
2. *Bank 2* — another bank, however with a much smaller fraud rate. This dataset does not have information on whether the transactions are with card present or not.
3. *Merchant* — with this dataset we have access to the types of labels, which allows us to also perform experiments where only real analysts' feedback labels are used, as explained in section 4.1.1.

			Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		AVG	AVG
Cold policy	*Warm-up policy*	*Hot policy*	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	rank all	VAR all
Random	Isolation forest ODAL	Uncertainty sampling (epistemic)	0.73	1	0.74	1	0.93	1	0.68	2	0.69	3	**1.6**	0.16
Random	Isolation forest ODAL	Uncertainty sampling (0.5)	0.73	2	0.74	2	0.93	2	0.69	1	0.69	4	**2.2**	0.16
Random	None	Uncertainty sampling (0.5)	0.72	3	0.73	4	0.91	5	0.67	3	0.68	5	**4.0**	0.18
Random	Isolation forest ODAL	None	0.54	5	0.67	5	0.90	7	0.56	6	0.71	2	**5.0**	0.13
Random	Isolation forest ODAL	Uncertainty sampling (fraud percentile)	0.55	4	0.74	3	0.91	3	0.55	7	0.52	10	**5.4**	0.26
Random	Isolation forest ODAL	Query by committee	0.53	7	0.59	6	0.81	9	0.57	5	0.61	6	**6.6**	0.21
Isolation forest outlier detection	None	None	0.36	11	0.53	11	0.90	6	0.52	8	0.71	1	**7.4**	0.19
Random	None	Query by committee	0.54	6	0.58	8	0.79	11	0.58	4	0.56	8	**7.4**	0.26
Random	Isolation forest ODAL	Expected model change	0.42	9	0.55	9	0.91	4	0.44	11	0.56	7	**8.0**	0.15
Random	None	None	0.48	8	0.58	7	0.81	10	0.50	9	0.52	11	**9.0**	0.26
Random	None	Expected model change	0.41	10	0.54	10	0.90	8	0.45	10	0.55	9	**9.4**	0.19

Table 11: Results' comparison of the experiments with *Bank 1 CNP*.

4. *Merchant acquirer* — this covers the last business case.

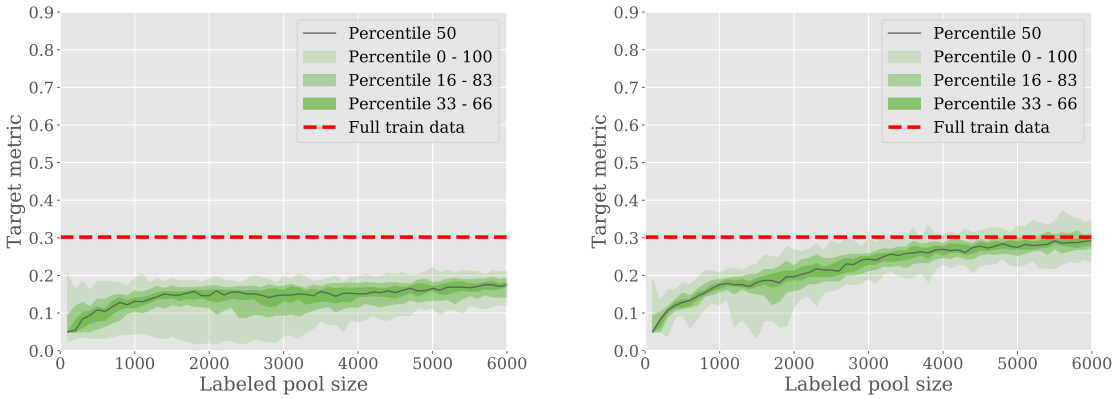
The results tables to be discussed in these experiments (tables 11, 12, 14, 13, 15 and 16) follow a structure similar to the ones presented in previous sections: each row corresponds to a different candidate policy and the groups of columns with numerical results are for different time folds.

For each dataset we present at least two example learning curve bands, (figures 17, 18, 20, 23, 25 and 26) which are the visualizations explained in section 4.4.1. The vertical axis is the target metric (e.g., recall at a target false-positive rate) and the horizontal axis is the size of the labeled pool. The plots are the intervals between percentiles of the target metric scores, so we can see the worst, median and best cases and the percentiles in-between. The horizontal red dashed line is the value of the full train data baseline, which is used to normalize the areas presented in the tables. In these analyzes, for each dataset, we present examples of the learning curves of the random policy and the best ranked policy (for that dataset) on the first time fold. In some cases, we present other learning curves to highlight other specific cases.

5.6.1 *Bank 1 with card-not-present filter*

All the experiments presented in the previous chapters were performed on a dataset from the banking use case. Frequently, in this business case, the goal is to classify transactions where the physical card was not present, such as online payments. This usually results in a dataset with a larger fraud rate (section 4.1). Therefore, to cover this sub-use case, we apply a CNP filter.

In table 11 we present the results of the experiments for this use case, from which we can conclude the following:



(a) Random policy

(b) Three-phase epistemic uncertainty sampling.

Figure 17: Performance in the first fold of *Bank 1* with the random policy and the best ranked policy.

- the various uncertainty sampling implementations and the standalone isolation forest ODAL, similar to the results presented in earlier sections for this dataset without *CNP* filter, are the best performing methods.
- every hot policy in a 3-phase setup outperforms its two-phase counterpart.
- the learning curves for this dataset have a relatively low variance, in particular the best performing policies (see also figure 17).
- epistemic uncertainty sampling outperforms all other methods in most folds, achieving relatively low values of variance, (though the differences are very small, and likely not significant, compared to other uncertainty measures).

In figure 17 we can see two example learning curve plots for these experiments: random policy (left panel) and epistemic uncertainty sampling (right panel), for the first fold. We can observe that the random policy has much more variance up to the 3500 instances. Furthermore, it plateaus at a considerably smaller central value close to 0.2. On the other hand, the best ranked policy has a significantly smaller variance and it reaches the full train data baseline performance near the end of the run (with a central value close to 0.3).

As expected the results do not differ much from the *Bank 1* dataset without CNP filter used in earlier sections. Since the results are consistent across the five time folds, the study for this dataset suggests that the usage of the three-phase epistemic uncertainty sampling policy is an appropriate choice for banking datasets with fraud rates similar to this one.

5.6.2 *Bank 2*

The dataset for *Bank 2* represents a great challenge even when training a ML model in an environment without AL, due to its extremely low fraud rate of 0.03%. Because of this very low rate, instead of running experiments only for seven days we extend it to two weeks. Hence, the maximum labeled

			Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		AVG	AVG
Cold policy	*Warm-up policy*	*Hot policy*	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	**rank all**	**VAR all**
Random	Isolation forest ODAL	Uncertainty sampling (epistemic)	0.16	5	0.54	1	0.55	3	0.33	3	0.55	2	**2.8**	0.47
Random	Isolation forest ODAL	Uncertainty sampling (0.5)	0.16	4	0.53	2	0.50	4	0.35	1	0.43	7	**3.6**	0.45
Random	Isolation forest ODAL	Uncertainty sampling (fraud percentile)	0.17	2	0.52	3	0.49	5	0.33	4	0.44	5	**3.8**	0.47
Random	None	Uncertainty sampling (0.5)	0.15	6	0.35	8	0.48	6	0.33	2	0.59	1	**4.6**	0.54
Random	Isolation forest ODAL	Expected model change	0.11	9	0.48	4	1.05	1	0.18	9	0.44	6	**5.8**	0.31
Random	None	None	0.20	1	0.34	9	0.26	9	0.27	6	0.39	8	**6.6**	0.47
Random	None	Expected model change	0.14	7	0.31	11	0.59	2	0.12	10	0.54	3	**6.6**	0.57
Random	Isolation forest ODAL	None	0.17	3	0.42	7	0.25	10	0.21	7	0.31	10	**7.4**	0.23
Random	Isolation forest ODAL	Query by committee	0.12	8	0.42	6	0.34	7	0.21	8	0.32	9	**7.6**	0.32
Random	None	Query by committee	0.11	11	0.31	10	0.33	8	0.31	5	0.52	4	**7.6**	0.46
Isolation forest outlier detection	None	None	0.11	10	0.45	5	0.22	11	0.10	11	0.15	11	**9.6**	0.16

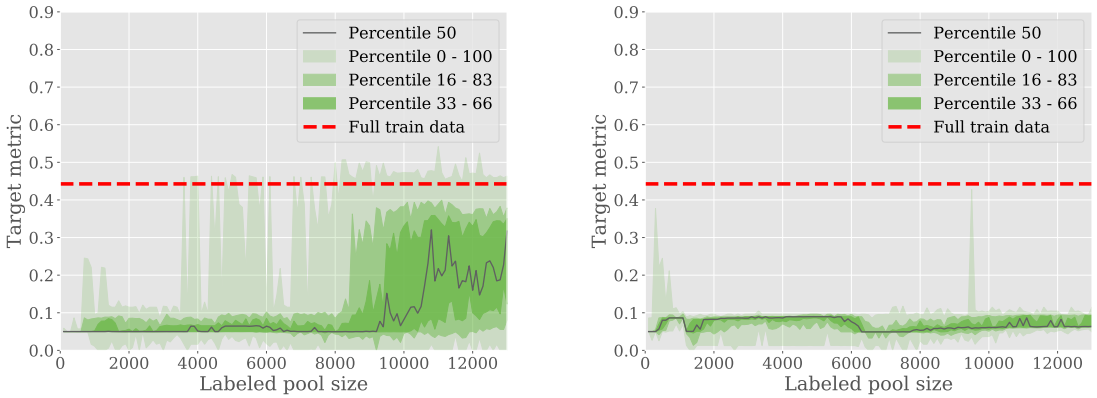
Table 12: Results of the experiments with *Bank 2*.

pool size is then 13000, corresponding to a period of 1 day to collect data for pre-processing and 13 days for labeling.

The results in table 12 lead us to the following observations:

- Most AL experiments with this dataset have very noisy learning curves when compared to *Bank 1*, since the values of the average variance are much larger.
- The normalized areas under the median learning curve, especially in folds 1 and 4, are very low, indicating that these AL experiments are far under-performing the full train data baseline.
- Exceptionally for fold 1, the random policy is the best performer because its high variance gives it a chance to reach larger values (while other policies struggle to learn any useful pattern to select queries). This behaviour is clear in figure 18. Hence, even though in this fold it often performs better, it is very unreliable. In contrast, for this fold, epistemic uncertainty is always in a plateau between 0 and 0.1.
- In fold 3, the three-phase Expected Model Change obtains a much higher normalized area when compared to the other policies. We can see the comparison of the learning curves of this policy with the overall best ranked policy (epistemic uncertainty sampling) in figure 19. We can observe that, the epistemic uncertainty sampling has a regular growth throughout the process and reaches at least the full train data baseline for a large fraction of the learning curves. On the other hand, the policy with Expected Model Change (on the right) tends to jump to a value above the full train data baseline when the labeled pool has a size around 3000 transactions.
- the best ranked policy is, once again, epistemic uncertainty sampling.

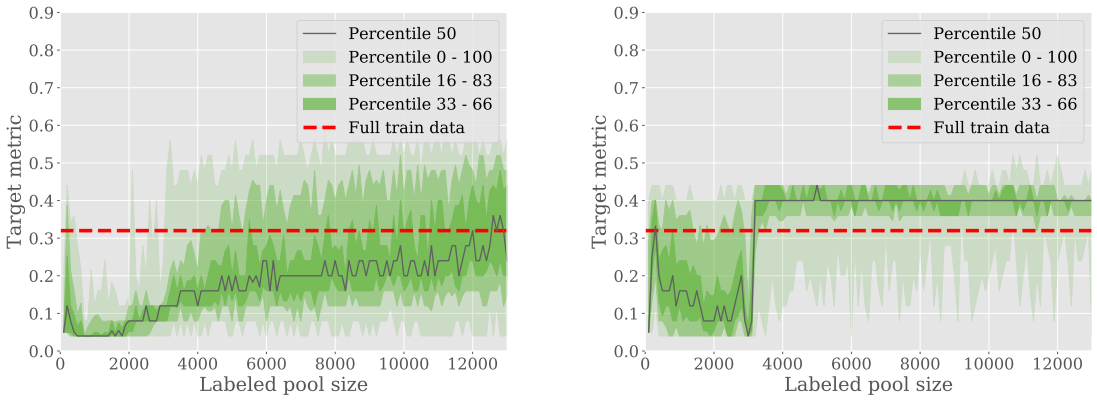
Regarding the full train data baseline, it is substantially larger than most AL results (see, e.g., figure 18 where its value is around 0.43) – with the exception of the examples presented in figure 19



(a) Random policy

(b) Three-phase epistemic uncertainty sampling.

Figure 18: Performance in the first fold of *Bank 2* with the random policy and the best average rank policy.



(a) Three-phase epistemic uncertainty sampling.

(b) Three-phase *Expected Model Change*.

Figure 19: Performance in the third fold of *Bank 2* with the best overall ranked policy (left figure) and an odd result (right figure).

and table 12, where we observe some higher performance values near the end of the run. This suggests that we may be able to obtain a higher performance, i.e., more comparable to *Bank 1*, by increasing the daily review budget to collect more labels.

Overall, this means that with such a conservative analyst budget (corresponding to a single analyst reviewing 1000 transactions per day) AL is not always able to deal with such an extreme class imbalance. By increasing the review budget we may be able to decrease the noise in the learning curves and obtain better results.

5.6.3 *Merchant*

As explained in section 4.1.1, the *Merchant* dataset has the advantage of containing details on how each label was obtained. We exploit this in order to perform different sets of experiments with datasets containing labels only obtained through:

- Analysts' feedback — transactions reviewed by analysts.
- Confident analysts' feedback — transactions that analysts have reviewed and stated they were very confident of the assigned label.
- Chargebacks — transactions that a client has complained about. Due to its nature, this is the type of label that is less likely to contain mistakes.

Analysts' feedback only

This scenario, allows us to only use transactions with labels assigned by an analyst to simulate a realistic case of an analyst who can make some mistakes. In order to produce an evaluation that is as robust as possible, we also use chargebacks in the test set, since they are the most robust labels and may have a label that is opposite to what an analyst would have decided in the AL process.

In table 13 we present the results for this scenario, where the following can be observed:

- In the first fold, the normalized areas do not differ much between different policies, even when using the random policy, which indicates that, for this exceptional case, the policy choice is not very important. This is especially clear when observing the learning curves of the random policy and the overall best ranked policy (in this dataset) in figure 20. Both are very similar and have very low variance, achieving a very stable plateau at the full train data baseline. We speculate that this could indicate that the fraud patterns in this train fold are sufficiently varied that we are collecting a representative sample by just randomly sampling, whereas for other folds there could be a dominant pattern and then AL helps finding the complementary ones faster. Another possibility is that the test set contains only transactions that are very easy to label.
- There are normalized areas much above 1 (e.g., most policies in fold 4), hence, the AL runs are outperforming the full train data baseline. In figure 21 we present two learning curves on the 4^{th} time fold, the random policy and the one with the largest normalized area (epistemic uncertainty sampling). The random policy grows with a regular slope throughout the process. On the other hand the very high performance epistemic uncertainty sampling curve (right

Cold policy	Warm-up policy	Hot policy	Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		AVG rank all	AVG VAR all
			Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank		
Random	Isolation forest ODAL	Uncertainty sampling (0.5)	0.93	2	1.02	2	0.79	2	2.24	3	1.36	3	**2.4**	0.35
Random	None	Uncertainty sampling (0.5)	0.93	3	1.02	1	0.79	3	2.29	2	1.35	4	**2.6**	0.36
Random	Isolation forest ODAL	Uncertainty sampling (epistemic)	0.90	5	0.88	4	0.79	1	2.33	1	1.11	5	**3.2**	0.29
Random	None	Expected model change	0.89	8	0.89	3	0.74	9	1.94	6	1.54	1	**5.4**	0.39
Random	Isolation forest ODAL	Expected model change	0.89	9	0.86	5	0.74	8	1.96	5	1.52	2	**5.8**	0.39
Random	Isolation forest ODAL	None	0.91	4	0.60	7	0.75	7	1.52	7	0.66	7	**6.4**	0.30
Isolation forest outlier detection	None	None	0.90	6	0.72	6	0.69	10	1.97	4	0.78	6	**6.4**	0.31
Random	None	None	0.90	7	0.45	8	0.77	6	0.87	8	0.62	8	**7.4**	0.39
Random	None	Query by committee	0.88	10	0.10	10	0.78	4	0.30	9	0.38	9	**8.4**	0.16
Random	Isolation forest ODAL	Query by committee	0.88	11	0.11	9	0.78	5	0.30	10	0.37	10	**9.0**	0.17
Random	Isolation forest ODAL	Uncertainty sampling (fraud percentile)	0.96	1	0.09	11	0.24	11	0.25	11	0.37	11	**9.0**	0.23

Table 13: Results of the experiments with the Merchant using only analysts' feedback.

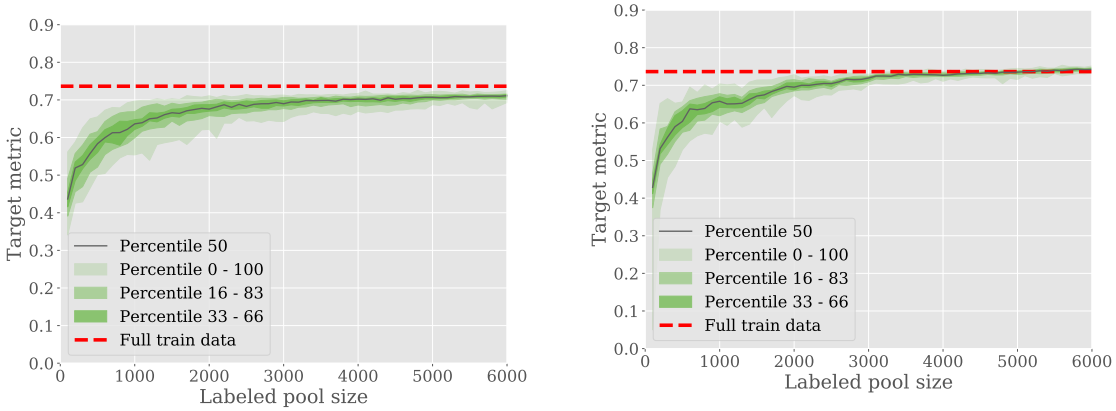
panel), plateau, in most cases, at around 2000 transactions with a target metric central value close to 3 times the full train data baseline value.

- In all folds, AL is either providing a clear advantage over random sampling (folds 2, 4 and 5), or it is at least as good.
- Uncertainty sampling with the uncertainty target as the threshold 0.5 is the overall best ranked policy.

We conjecture two possible reasons to blame for the very low performance of the full train data baseline in some folds:

- Since we are using the same model as for AL, it might not be appropriate for training with so much more data and therefore ends up under-fitting it.
- The baseline model is trained with data extending over a wider period (i.e., the full train data baseline has access to two weeks of data, whereas AL only uses a sample from the first week), so its performance could be harmed by the existence of noise, on the second week, that is not present in the AL labeled pools.

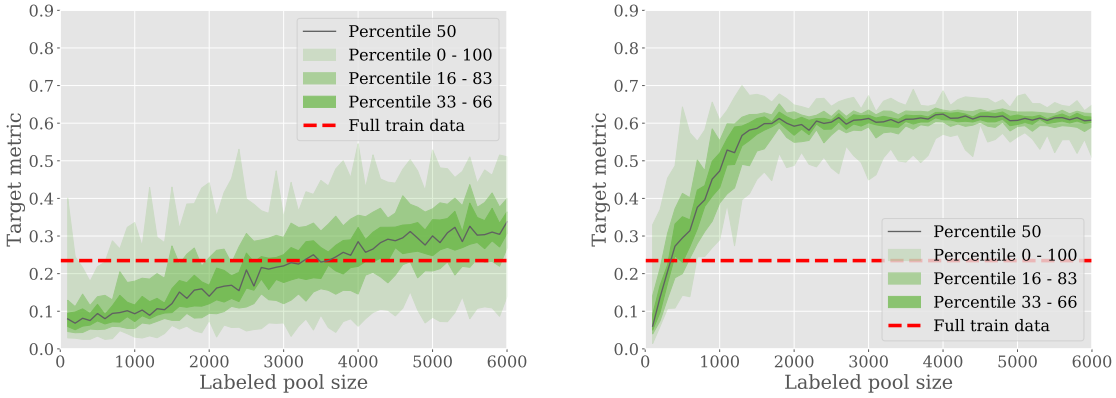
The (likely) best solution would be to perform hyper-parameter tuning and maybe even model selection to the process of computing the full train data baseline. Done properly, this would ensure the best ML model configuration for (the given data) is always being used and would result in a full train data baseline that actually represents a full train data baseline for the given train/test split.



(a) Random policy

(b) Three-phase uncertainty sampling (using 0.5 threshold).

Figure 20: Performance in the first fold of the *Merchant* with analyst feedback only using the random policy and the best ranked policy.



(a) Random policy.

(b) Three-phase epistemic uncertainty sampling.

Figure 21: Performance in the 4^{th} fold of the *merchant* dataset with analysts' feedback only, with the random policy (left figure) and an odd result (right figure), due to having a very high normalized area (2.23).

Cold policy	Warm-up policy	Hot policy	Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		AVG rank all	AVG VAR all
			Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank		
Random	Isolation forest ODAL	Uncertainty sampling (epistemic)	0.94	2	0.68	3	0.79	1	2.02	1	0.68	5	**2.4**	0.26
Random	None	Uncertainty sampling (0.5)	0.93	3	0.81	2	0.78	2	1.96	3	0.85	3	**2.6**	0.31
Random	Isolation forest ODAL	Uncertainty sampling (0.5)	0.93	4	0.82	1	0.78	3	1.96	2	0.84	4	**2.8**	0.31
Random	None	Expected model change	0.91	8	0.64	5	0.74	8	1.47	4	1.16	1	**5.2**	0.35
Random	Isolation forest ODAL	Expected model change	0.91	7	0.66	4	0.74	9	1.41	5	1.13	2	**5.4**	0.34
Random	Isolation forest ODAL	None	0.92	5	0.32	7	0.75	7	1.07	7	0.39	7	**6.6**	0.26
Random	None	None	0.89	9	0.29	8	0.78	4	0.47	8	0.42	6	**7.0**	0.33
Isolation forest outlier detection	None	None	0.92	6	0.40	6	0.71	10	1.27	6	0.38	8	**7.2**	0.25
Random	Isolation forest ODAL	Uncertainty sampling (fraud percentile)	0.96	1	0.07	9	0.44	11	0.18	11	0.27	11	**8.6**	0.21
Random	None	Query by committee	0.87	10	0.07	10	0.77	6	0.23	10	0.27	9	**9.0**	0.14
Random	Isolation forest ODAL	Query by committee	0.87	11	0.07	11	0.77	5	0.24	9	0.27	10	**9.2**	0.13

Table 14: Results of the experiments with the Merchant using only confident analysts' feedback.

Confident analysts' feedback only

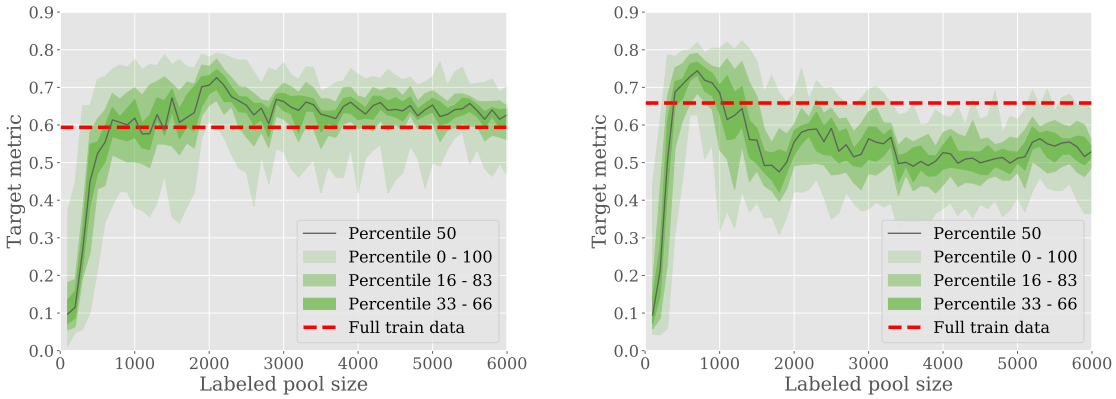
In this experiment we only provide labels that have been reviewed by analysts and were assigned with a high level of confidence. This allows us to understand if using only labels that should have lower levels of noise is beneficial in AL.

The results of this experiment are presented in table 14, from which we conclude the following:

- Just like when using only analysts' feedback, the overall results are good. Fold 4 also has extreme values. The main differences are that the normalized areas are lower in fold 2 and 5 and that, here, the best ranked policy is epistemic uncertainty sampling instead of the 0.5 threshold uncertainty method. Note, however, that if we look at the numbers, the ranking of these two policies would be swapped if we used an average of the KPI instead of an average ranking. This is because the absolute differences in the KPI values are, in many cases, small, so the differences in the ranks do not have significance.
- The best ranked policy is the epistemic uncertainty sampling, but note that overall, the differences in average ranks of the top three policies are not very significant.

In order to understand the differences in the normalized areas, in figure 22 we can see a comparison between the two datasets with the same policy in fold two. We can observe that there are two main reasons why the normalized areas are inferior when using only confident labels:

- the median target metric score at plateau, when using only confident feedback (approximately 0.55), is lower than when using all feedback (approximately 0.65)
- the full train data baseline score is larger, which makes the normalization term larger, hence lowering the KPI.



(a) Only analysts' feedback.

(b) Only confident analysts' feedback.

Figure 22: Learning curves of the same policy (0.5 threshold uncertainty sampling) on the second fold of the *Merchant* dataset with different types of labels.

One interesting detail about the learning curve of using only confident feedback is that it first achieves a peak in performance and then it decreases again. Given that the performance starts decreasing after almost one day of the AL process,[5] we conjecture that in the first days the data could be initially cleaner and after that label noise could be present. Another explanation could be that, since we have a fixed test set, maybe the latter is dominated by a particular fraud pattern that is very easily caught by the first models at early stages of AL but then the final models become less specialized.

Looking at figure 23 we see the learning curves of the random and best ranked policy of the dataset on fold 1. The conclusion is very similar to the corresponding example with all feedback (figure 20): random has slightly larger variance but both curves plateau at the full train data baseline. However, the full train data baseline performance with only confident feedback is larger than when using all feedback (a difference from 0.75 to 0.9, approximately). This means that using only confident feedback can be beneficial even outside an AL context.

Chargebacks only

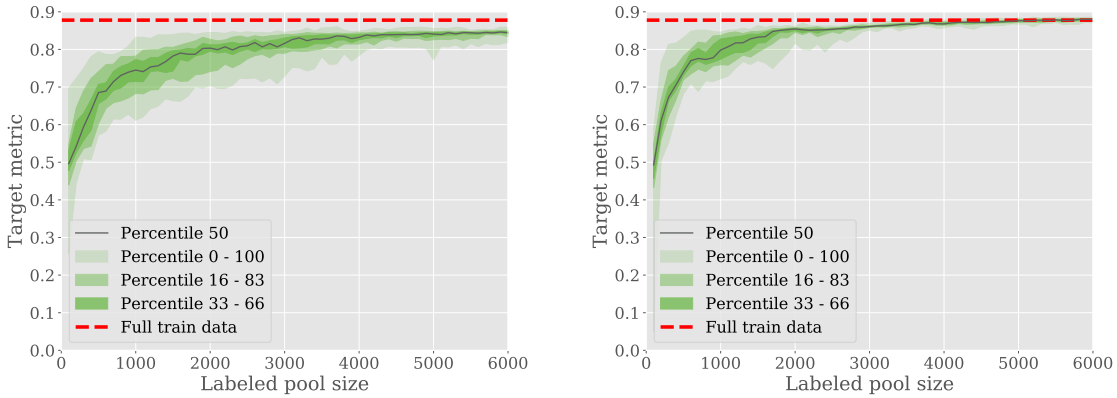
In this scenario we use only labels that originated from a client complaint, meaning that we only use positive instances that have the highest degree of confidence possible.[6] Even though this is not realistic in an AL context (because chargeback labels usually take several days or weeks to arrive), experimenting only with this less noisy type of transactions may help us understand how efficient an ideal AL model trained on this subset of high quality labels would be.

In table 15 we present the results. By observing the numbers we can conclude the following:

- the variance in the experiments is much larger than in other labeling scenarios. This is, perhaps, not surprising since we are using a much smaller subset of all fraud labels.

5 a labeled pool size of 1000 indicates one day has passed, since 1000 transactions are being added to the labeled pool per day

6 On the other hand, because a big portion of the analyst labels are also true fraud, this biases the dataset towards a particular subset of all fraud.



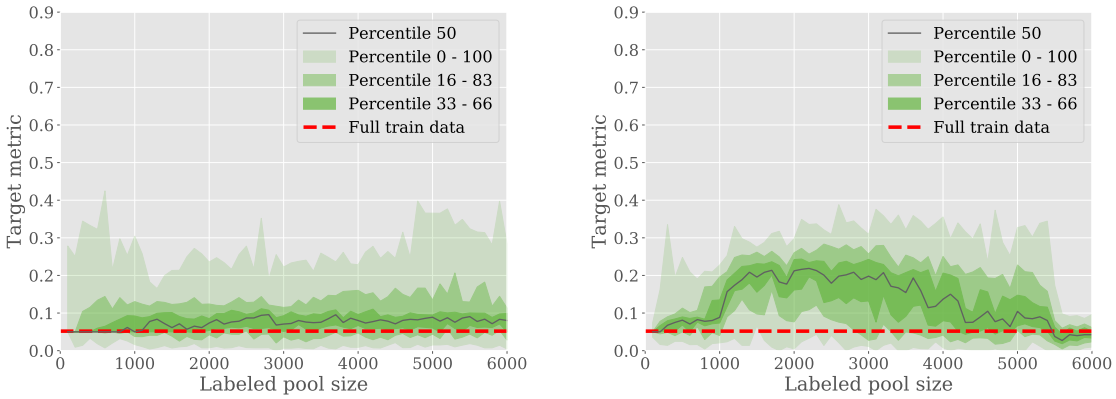
(a) Random policy

(b) Three-phase epistemic uncertainty sampling.

Figure 23: Performance in the first fold of the *Merchant* only using confident analyst feedback the random policy and the best ranked policy.

			Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		AVG	AVG
Cold policy	*Warm-up policy*	*Hot policy*	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	**rank all**	**VAR all**
Random	Isolation forest ODAL	Uncertainty sampling (epistemic)	0.76	2	0.45	1	0.85	2	0.40	3	1.15	5	**2.6**	0.60
Random	Isolation forest ODAL	Uncertainty sampling (0.5)	0.75	3	0.39	3	0.87	1	0.41	1	1.13	6	**2.8**	0.58
Random	None	Uncertainty sampling (0.5)	0.73	4	0.41	2	0.84	4	0.40	2	1.08	8	**4.0**	0.64
Random	Isolation forest ODAL	Query by committee	0.64	9	0.13	9	0.85	3	0.31	5	2.49	1	**5.4**	0.86
Random	Isolation forest ODAL	Expected model change	0.72	6	0.25	4	0.83	6	0.25	10	1.23	4	**6.0**	0.42
Random	None	Query by committee	0.58	10	0.15	7	0.79	10	0.31	4	1.89	2	**6.6**	0.99
Random	Isolation forest ODAL	Uncertainty sampling (fraud percentile)	0.77	1	0.13	11	0.84	5	0.24	11	1.03	9	**7.4**	0.44
Random	None	Expected model change	0.70	7	0.23	5	0.81	9	0.27	9	1.11	7	**7.4**	0.45
Random	Isolation forest ODAL	None	0.65	8	0.14	8	0.83	7	0.30	6	1.01	10	**7.8**	0.37
Random	None	None	0.52	11	0.19	6	0.72	11	0.29	8	1.40	3	**7.8**	0.82
Isolation forest outlier detection	None	None	0.73	5	0.13	10	0.82	8	0.30	7	0.91	11	**8.2**	0.37

Table 15: Results of the experiments with the Merchant using only chargebacks.



(a) Random policy

(b) Query By Committee.

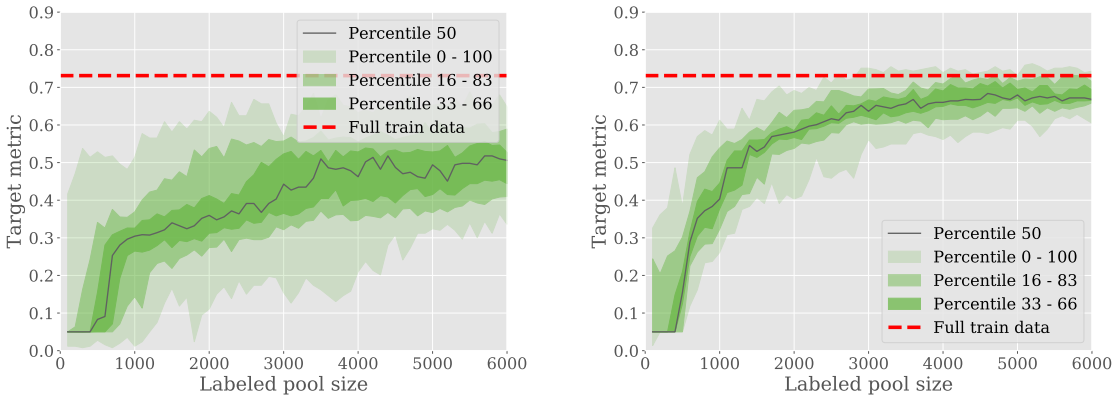
Figure 24: Performance in fold 5, with two different policies, of the *Merchant* with chargebacks only, which results in anomalous normalized areas.

- all folds appear to have smaller normalized areas relative to the previous datasets, except for fold 5 where almost all values are above 1, meaning it is much above the full train data baseline. In figure 24, where we have the learning curves of the random policy (left figure) and Query By Committee (right figure), we can observe that this is mostly happening because the full train data baseline score is very small (near 0.05, i.e., a random model), hence the normalized areas of the curves tend to be very large. This problem (with the very low scores) happens with both AL and the full train data baseline, meaning that this is an issue likely associated with lack of enough labels representing all fraud parts, since we are using only chargebacks, or noisy data.
- the best ranked policy is epistemic uncertainty sampling (but the second ranked uncertainty policy is very similar).

In figure 25, we can see the learning curves in fold 1 of the random policy and the overall best ranked policy in this dataset. If we compare to the same examples from the other merchant datasets, we note that we now have much more variance (especially for the random policy) whereas the full train data baseline is performing similarly to the ones where analysts' feedback is used.

Comparison of scenarios

From the results of experiments across different labelling scenarios, we conclude that using only chargebacks is not very suitable since it causes a lot more variance and, in some folds (e.g., fold 1, 2, 3), a smaller normalized area. Between using all analysts' feedback or only the confident analysts' feedback the difference is not so clear. Even though the normalized areas when using only confident feedback tends to be smaller, in some cases this is also because the full train baseline is also performing better (as seen in figure 22). Therefore, the only conclusion we can draw is that both labelling setups are good, even though it varies, from case to case, which one is better.



(a) Random policy

(b) Three-phase epistemic uncertainty sampling.

Figure 25: Performance in the first fold of the *Merchant* with chargebacks only using the random policy and the best ranked policy.

5.6.4 *Merchant acquirer*

A merchant acquirer is an entity that aggregates several merchants and processes transactions between them and the consumer, which is a completely different challenge from the other business cases. They are different from merchants, which only process transactions made to a single entity (themselves), and they are different from banks since it is not always possible to identify, in payments made to two different merchants, if the consumer was the same.

Table 16 presents the results for the experiments performed in this business case:

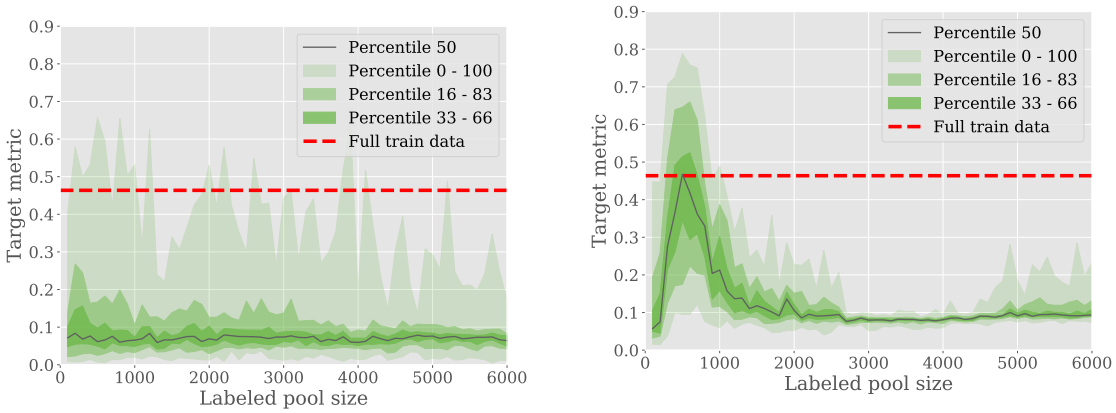
- The results are extremely inconsistent across time folds. Regarding normalized areas, fold 1 and 4 have low values, while fold 2 has extremely high values (much higher than one) and fold 3 and 4 has normal values (close to 1).
- The levels of variance in the results are much larger than in all previous experiments.
- Even though the abnormality of the results must be accounted for, the best ranked AL policy is uncertainty sampling with the 0.5 threshold.

In figure 26, where the learning curves of the random and uncertainty sampling (0.5) policies are displayed, we can see the reason why the normalized areas of fold 1 are so low. The plateau of the learning curves is at a very low target metric score, while the full train data baseline has a much higher performance. One interesting detail is the decrease in performance at the beginning of the learning curve with uncertainty sampling, where having a labeled pool with only 500 instances leads to better performance than with 1000 or more. This is similar to what we observed in fold two of the Merchant dataset using only confident feedback and may be caused by the same conjectures presented.

In figure 27 we can understand the extreme values in the second fold. While the policy with Expected Model Change achieves median values very high (close to 0.9), the full train data baseline performance is much lower (around 0.18). We can visualize that both policies are unreliable due to

			Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		AVG	AVG
Cold policy	*Warm-up policy*	*Hot policy*	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	Norm. Area	Rank	**rank all**	**VAR all**
Random	Isolation forest ODAL	Uncertainty sampling (0.5)	0.27	3	2.75	4	1.08	1	0.48	3	0.93	5	**3.2**	0.90
Random	None	Uncertainty sampling (0.5)	0.25	4	2.17	5	1.07	3	0.47	4	0.99	2	**3.6**	0.84
Random	Isolation forest ODAL	Expected model change	0.36	2	4.39	1	0.95	5	0.51	2	0.38	10	**4.0**	0.84
Random	Isolation forest ODAL	Uncertainty sampling (epistemic)	0.22	5	2.75	3	1.07	2	0.32	5	0.80	6	**4.2**	0.78
Random	None	Expected model change	0.39	1	4.01	2	0.95	6	0.51	1	0.34	11	**4.2**	0.89
Isolation forest outlier detection	None	None	0.18	7	1.86	6	0.98	4	0.11	6	1.02	1	**4.8**	0.77
Random	Isolation forest ODAL	None	0.19	6	1.23	7	0.88	7	0.07	8	0.96	3	**6.2**	0.68
Random	None	None	0.15	8	0.97	8	0.73	9	0.08	7	0.61	9	**8.2**	1.06
Random	Isolation forest ODAL	Uncertainty sampling (fraud percentile)	0.06	11	0.22	10	0.84	8	0.05	9	0.95	4	**8.4**	0.60
Random	Isolation forest ODAL	Query by committee	0.07	9	0.36	9	0.64	11	0.03	11	0.77	7	**9.4**	0.74
Random	None	Query by committee	0.07	10	0.18	11	0.66	10	0.04	10	0.75	8	**9.8**	0.53

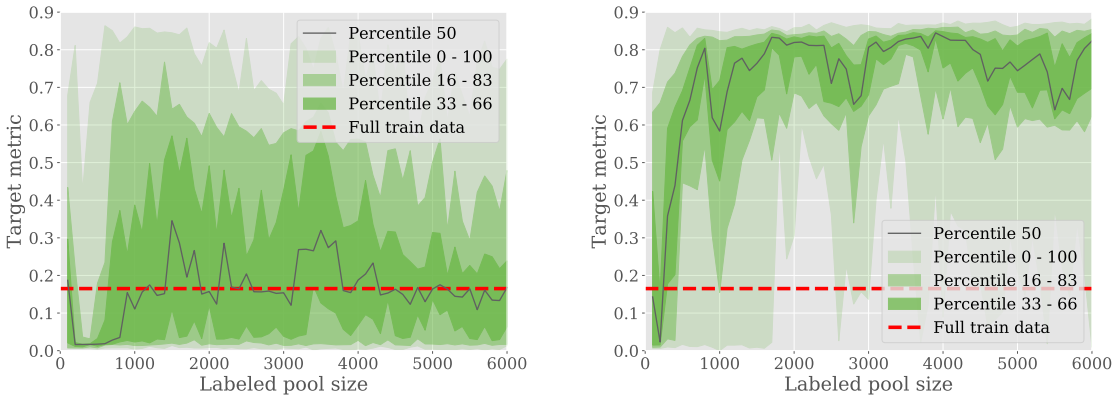
Table 16: Results of the experiments with the *merchant acquirer*.



(a) Random policy

(b) Three-phase uncertainty sampling (using 0.5 threshold).

Figure 26: Performance in the first fold of the *Merchant Acquirer* using the random policy and the best ranked policy.



(a) Random policy

(b) Three-phase Expected Model Change.

Figure 27: Learning curves of the second fold of the *Merchant Acquirer* using the random policy and the best ranked policy in that fold.

their extremely high variance. In both cases, even when the labeled pool has close to 6000 instances, the performance range goes from a target metric score of near 0 to near 0.9.

5.6.5 *Key takeaways*

Table 17 presents every policy used in these experiments and their ranks in each of the datasets. The policies are sorted by the average of those ranks (rightmost column).

From this extensive set of experiments on several fraud business cases, the overall conclusions are as follows:

- The three-phase framework brings undeniable value given that in all studies the top ranked candidate is always a set of three policies. Also, as visible in table 17, no hot policy in a two-phase setup tends to outperform the same hot policy in a three-phase setup.
- In all comparisons, either epistemic uncertainty sampling or uncertainty sampling at the 0.5 threshold are always the best ranked policies and perform very similarly. This is consistent with the fact that epistemic uncertainty should dominate the total uncertainty (so both policies should be similar). Furthermore, these policies never demonstrate significantly higher variance levels than other policies.
- Only Query By Committee and outlier detection have average ranks that are worse than the random policy's.

Cold policy	*Warm-up policy*	*Hot policy*	Bank 1	Bank 2	Merchant w/ analysts feedback	Merchant w/ confident analysts feedback	Merchant w/ chargebacks	Merchant acquirer	**AVG**
Random	Isolation forest ODAL	Uncertainty sampling (epistemic)	1	1	3	1	1	4	1.8
Random	Isolation forest ODAL	Uncertainty sampling (0.5)	2	2	1	3	2	1	1.8
Random	None	Uncertainty sampling (0.5)	3	4	2	2	3	2	2.7
Random	Isolation forest ODAL	Expected model change	9	5	5	5	5	3	5.3
Random	None	Expected model change	11	6	4	4	7	4	6.0
Random	Isolation forest ODAL	None	4	8	6	6	9	7	6.7
Random	Isolation forest ODAL	Uncertainty sampling (fraud percentile)	5	3	10	9	7	9	7.2
Random	None	None	10	6	8	7	9	8	8.0
Isolation forest outlier detection	None	None	7	11	6	8	11	6	8.2
Random	Isolation forest ODAL	Query by committee	6	9	10	11	4	10	8.3
Random	None	Query by committee	7	9	9	10	6	11	8.7

Table 17: Every policy used in the large-scale experiments and their ranks in the datasets. The policies are sorted by average rank.

6

CONCLUSIONS

In this dissertation, we studied active learning (AL) methods to build a ML model for fraud detection. We explored a realistic setup of a new system where there is no historical data and the system needs to be deployed to collect data and start building a useful ML model. We designed a versatile system architecture (Section 3.4) that allows us to deploy AL in a live production system. In designing this solution we took into account:

- how to perform pre-processing to prepare the data for the AL framework.
- how to manage the data, which is incoming from a live data stream to be stored in an unlabeled data pool and is then gradually sampled to create a labeled data pool.
- that it iteratively selects batches of data instances to be labeled when configured with a given set of AL policies and switching criteria between policies.
- it contains a component that, given a team of labelers (i.e., the analysts/oracles) and a distribution schedule, queries the analysts for the chosen transactions' labels.
- the possibility to train ML models at any AL iteration in any way, not only for deployment but also to perform evaluations of the current system performance.

Then, we extended this architecture to be able to conduct simulation experiments of a production system using historical datasets (Section 3.5). This includes:

- the implementation of a data stream to process transactions in a time ordered fashion and to insert transactions in the labeled pool according to the time that analysts take to review the batches;
- the development of a simulated labeler that uses the real labels of the transactions (since we used a historical dataset) and spends a given time labeling a transaction;
- the development of an evaluation component that can train ML models on the labeled pool and make predictions on a test set throughout the AL run to measure its performance.

When evaluating our experiments (Chapter 4), for each policy we performed experiments on several time folds. A time fold is defined as a period of 4 weeks with the first two weeks available for AL and the other two used as a test set. In practice we only used the first week for AL except for one dataset with an extremely large class imbalance (for which we used the two weeks). We introduced a visualization technique to be able to analyze the behavior of the learning curves of the AL runs, as well as Key Performance Indicators (KPIs) based on these learning curves. This allows us to quantify the performance of an AL configuration and compare several policies or time folds.

The in-depth literature review on the current state-of-the-art in the AL field (Chapter 2), led us to a set of policies to experiment with, as well as to our custom made methods. We implemented (Section 3.6):

- Expected model change.
- Uncertainty sampling, which queries instances that obtain scores closer to 0.5. We also implemented our own two variants of this method: the fraud percentile uncertainty sampling and epistemic uncertainty sampling.
- Query by committee with our own modification on how the model disagreement measure is computed.
- A density-weighted method that turned out to be too computationally heavy for fraud detection due to the large dataset sizes (therefore discarded).
- Our novel Outlier Discriminative Active Learning (ODAL) method.

We launched a set of preliminary experiments (Section 5.1), with only two time folds of a dataset, that led us to the conclusion that uncertainty sampling and isolation forest ODAL were performing better.

We also performed a set of experiments to validate the data pre-processing method used in the preliminary experiment. This assists in the decision on which method should be used in the final large scale experiments for policy comparison (Section 5.2). The main candidates were:

- Feedzai's AutoML tool to generate many features based on the user profiles and then apply either: i) feature reduction with Principal Components Analysis (PCA) or ii) pairwise correlation selection.
- Feedzai's AutoML tool to generate less features already using expert knowledge to manually narrow down the number of profiles.

From the results of the experiments we concluded that generating many features with the AutoML tool and then reducing them with PCA was the best alternative. We also performed a similar set of experiments to validate ML model options and decide which one we should be using further on (Section 5.3). From several ML models, each with varying sets of hyper-parameters, we concluded that:

- For a given fixed model type, varying the hyper-parameters in a way that increases regularization increases performance, as expected given the small sample sizes produced by AL.
- The random forest models turned out to be the most effective candidates.

In light of these preliminary experiments, we then moved on to observing that one of the main challenges of applying AL in fraud detection is the extreme class imbalance of some datasets. Given that most AL policies require the labeled pool to contain labels of all classes, this sometimes results in AL runs that only switch to this hot policy in very late stages of the process. The standard approach in the AL literature is to produce an initial random sample until the hot policy can be used, which often results in a later switching. We introduced a solution for this problem (Section 3.7): a three-phase framework where there is a cold, a warm-up and a hot policy. In all later experiments (Section 5.4

and 5.6) we concluded that, for any hot policy, its performance in the three-phase framework (with ODAL as warm-up policy) is always better than a two-phase framework.

Another hypothesis we considered was that the combination of two AL policies could perform better than each separate policy. In order to understand the potential of combining policies, we developed policy divergence diagnostics that can assess how different two AL policies are (Section 3.8.1): cluster frequency divergence and model disagreement divergence. We performed experiments using these diagnostic metrics (Section 5.5.1) and concluded that, as expected, between pair of policies that are intrinsically similar (e.g., elliptic envelope ODAL and isolation forest ODAL) they signal very low levels of divergence whereas for some of the other pairs higher values were found. This is a good indication that the diagnostics metrics work as they should. However, while the cluster frequency divergence indicated that some policies could definitely benefit from combination, the model disagreement divergence results were not so clear about such benefit. This hints that, from a performance point of view, the benefits of the combination would be smaller.

In order to combine two policies, we then introduced three methods (Section 3.8.2): the weighted ranking combiner, the alternating combiner and the cascade combiner; In the experiments performed with these combiners (Section 5.5) we concluded that none of the combined policies outperformed the best performing single policy (uncertainty sampling). Thus, we concluded that it was not worth including policy combination in the final large scale experiments.

Finally, armed with a good understanding of which AL configurations should be further investigated, we presented the final large scale experiments on various datasets. For these, we introduced four datasets (Section 4.1): Bank 1 with a card-not-present (CNP) filter, Bank 2 with a very small fraud rate, a Merchant that processes payments from its consumers and a Merchant acquirer that processes payments for several merchants from their consumers. The Merchant dataset contained further information on how each of the labels was obtained. This allowed us to generate three variations of this dataset that used only:

- all analysts' feedback (allowing us to simulate the effect of the analysts' mistakes).
- confident analysts' feedback (to allow us to understand if only using labels marked as confident could reduce the negative impact introduced by label noise)
- chargeback labels (from direct client complaints, which should have low label noise).

With everything set, we launched the final large-scale experiments and analyzed their results (Section 5.6). The conclusions were as follows:

- AL appears to be an appropriate solution for datasets similar to Bank 1. It always reaches the full train data baseline performance and the levels of variance are considerably lower than the random policy. This means that, in this domain, the empirical results indicate that a good model can be obtained with a small budget of queries.
- With datasets that have very low fraud rates, like bank 2, AL does not bring very stable results with the budget size we used.
- The Merchant, for which we generated three different types of datasets for different label scenarios, led us to understand that using only chargebacks (the scenario that should have much less noise) does not provide stable results. On the other hand using analysts' feedback

we do obtain good results, with minor differences between using only confident feedback or all feedback.

- The Merchant acquirer dataset, just like Bank 2, had several poor results that still need to be understood to find further improvements, before using AL in this business case.
- In a meta-analysis presented in table 17, to combine the policy rankings from different datasets, we concluded that the best overall ranked set of policies was our proposed three-phase setup with isolation forest ODAL as a warm-up policy and uncertainty sampling (either epistemic or with threshold at 0.5) as hot policy.

The conclusions presented in this study open up several further questions that are left for future work. Namely:

- There were a few experiments/folds for Bank 2, for the Merchant with chargebacks only and for the Merchant acquirer with anomalous learning curves or optimistic baselines with lower performance. Trying to understand the reasons behind such issues and finding ways to overcome them would be an important next step.
- In particular for Bank 2, one of the reasons for those issues could be simply related to the extremely low fraud rate (which results in a very large variance). If that is the case, experiments with a higher budget of analysts should be performed to determine if better results can be obtained that are more similar to Bank 1.
- For the Merchant dataset with chargebacks only, contrarily to the datasets using analysts' feedback, the test set only had chargebacks. This difference may have harmed the results, because there is less fraud labels in chargebacks. Redoing the evaluations with the same labelling setup on the test set would make the three evaluations more directly comparable.
- In all the experiments, we trained models using the exact same model and hyper-parameters when the labeled pool has 50 or 6000 instances and we also do not adapt them according to the datasets. Performing hyper-parameters tuning in AL and when computing the full train data baselines could have an impact on our evaluation. This could also be helpful to address the issues mentioned above.
- One increasing concern in the fraud detection field is money laundering. This is a domain for which it is harder to build a ML model since, unlike with other types of fraud, there are no client complaints. This results in a label scarcity problem. As already demonstrated in Lorenz et al. (2020), where the tools developed in our work were used, this is a field where AL seems to be the most promising strategy. Therefore, further research on the usage of AL for anti-money laundering could bring interesting results.
- As typically done in fraud detection, transactions for which there is no feedback (i.e., there were no complaints but no analyst have reviewed them either) are presumed to be legitimate. This could be a source of label noise and using AL may help find those instances and bring value by reducing label noise.
- Our experiments resorted to a simulated analyst. Performing experiments with actual human analysts is an important next step since it might either reveal limitations or suggest adjustments, which may be necessary to apply to our proposed AL methods to make them successful.

BIBLIOGRAPHY

João Tiago Ascensão, Pedro Bizarro, Ricardo Barata, Miguel Leite, Ricardo Pacheco, and Marco O.P. Sampaio. Active learning for online training in imbalanced data streams under cold start.

Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000. ISSN 0163-5808. doi: 10.1145/335191.335388. URL https://doi.org/10.1145/335191.335388.

Fabrizio Carcillo, Yann-Aël Le Borgne, Olivier Caelen, and Gianluca Bontempi. Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization. *International Journal of Data Science and Analytics*, 5(4):285–300, 2018.

Weijie Fu, Meng Wang, Shijie Hao, and Xindong Wu. Scalable active learning by approximated error reduction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1396–1405, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3219954. URL http://doi.acm.org/10.1145/3219819.3219954.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.

Daniel Gissin and Shai Shalev-Shwartz. Discriminative active learning. *CoRR*, abs/1907.06347, 2019. URL http://arxiv.org/abs/1907.06347.

S. Huang, R. Jin, and Z. Zhou. Active learning by querying informative and representative examples. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):1936–1949, Oct 2014. ISSN 1939-3539. doi: 10.1109/TPAMI.2014.2307881.

Yufeng Kou, Chang-Tien Lu, Sirirat Sirwongwattana, and Yo-Ping Huang. Survey of fraud detection techniques. In *IEEE International Conference on Networking, Sensing and Control, 2004*, volume 2, pages 749–754. IEEE, 2004.

Kenneth Lang and Eric Baum. Query learning can work poorly when a human oracle is used, 1992.

David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning Proceedings 1994*, pages 148 – 156. Morgan Kaufmann, San Francisco (CA), 1994. ISBN 978-1-55860-335-6. doi: https://doi.org/10.1016/B978-1-55860-335-6.50026-X. URL http://www.sciencedirect.com/science/article/pii/B978155860335650026X.

X. Li and Y. Guo. Adaptive active learning for image classification. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 859–866, June 2013. doi: 10.1109/CVPR.2013.116.

Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, page 413–422, USA, 2008. IEEE Computer Society. ISBN 9780769535029. doi: 10.1109/ICDM.2008.17. URL https://doi.org/10.1109/ICDM.2008.17.

Joana Lorenz, Maria Inês Silva, David Aparício, João Tiago Ascensão, and Pedro Bizarro. Machine learning methods to detect money laundering in the bitcoin blockchain in the presence of label scarcity. *arXiv preprint arXiv:2005.14635*, 2020.

Paulo César Gonçalves Marques, Miguel Ramos de Araújo, Bruno Casal Laraña, Nuno Miguel Lourenço Diegues, Pedro Cardoso Lessa e Silva, and Pedro Gustavo Santos Rodrigues Bizarro. Semantic-aware feature engineering, March 19 2020. US Patent App. 16/567,761.

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

Fábio Pinto, Marco OP Sampaio, and Pedro Bizarro. Automatic model monitoring for data streams. *arXiv preprint arXiv:1908.04240*, 2019.

Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, 175 (4):7–9, 2017.

Peter J. Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999. doi: 10.1080/00401706.1999.10485670. URL https://www.tandfonline.com/doi/abs/10.1080/00401706.1999.10485670.

Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 441–448, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.655646.

Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 582–588, Cambridge, MA, USA, 1999. MIT Press.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: 10.1145/130385.130417. URL http://doi.acm.org/10.1145/130385.130417.

Mohammad Hossein Shaker and Eyke Hüllermeier. Aleatoric and epistemic uncertainty with random forests. In *International Symposium on Intelligent Data Analysis*, pages 444–456. Springer, 2020.

Asli Uyar, Ayse Bener, H Nadir Ciray, and Mustafa Bahceci. A frequency based encoding technique for transformation of categorical variables in mixed ivf dataset. In *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 6214–6217. IEEE, 2009.

Andreas Vlachos. A stopping criterion for active learning. *Computer Speech & Language*, 22(3):295–312, 2008.

Min Wang, Fan Min, Zhi-Heng Zhang, and Yan-Xue Wu. Active learning through density clustering. *Expert Systems with Applications*, 85:305 – 317, 2017. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2017.05.046. URL http://www.sciencedirect.com/science/article/pii/S095741741730369X.

Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16 (3):645–678, 2005.

Yazhou Yang and Marco Loog. A benchmark and comparison of active learning for logistic regression, 2016.

Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863, 2003.

Yifan Zhang, Peilin Zhao, Shuaicheng Niu, Qingyao Wu, Jiezhang Cao, Junzhou Huang, and Mingkui Tan. Online adaptive asymmetric active learning with limited budgets. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

Jingbo Zhu, Huizhen Wang, Eduard Hovy, and Matthew Ma. Confidence-based stopping criteria for active learning for data annotation. *ACM Transactions on Speech and Language Processing (TSLP)*, 6 (3):3, 2010a.

Xingquan Zhu, Peng Zhang, Xiaodong Lin, and Yong Shi. Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(6):1607–1621, 2010b.

Indrė Žliobaitė, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. Active learning with evolving streaming data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 597–612. Springer, 2011.