TÉCNICO LISBOA

Infrastructure and Machine Learning for Credit Scoring

Francisco Falcão Amaral Caixeiro

Thesis to obtain the Master Degree in

Telecommunications and Informatics Engineering

Supervisors: Prof. Dr. Manuel Fernando Cabido Peres Lopes
Dr. Ricardo Portela

Examination Committee

Chairperson: Prof. Dr. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. Dr. Manuel Fernando Cabido Peres Lopes
Member of the Committee: Prof. Dr. Pável Pereira Calado

November 2018

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First, I would like to thank my family, parents, and girlfriend for all the unconditional support.

I would like to sincerely thank my advisors at Innovation Makers, Ricardo Portela and Francisco Chaves, for the opportunity that allowed me to accomplish this journey, and also my advisor Manuel Lopes for all support. I would also like to thank the rest of INM in this journey, especially my fellow Masters' colleague Vicente Rocha.

I also would like to thank my friends that followed this journey very closely, and helped me with advice, emotional support and motivation. I have a lot of people that helped me in one way or another. Nonetheless, I would like to highlight the following: Isabel Costa and Lucas Silva.

Abstract

Credit scoring is the analytical technique used by banks to evaluate credit risk. Measuring this risk is of utmost importance for the banks, and they are relying on statistical and machine learning models to do it. This project proposes the development of an infrastructure that supports the amounts of data generated by banks every day. It also integrates the training and testing of credit scoring models like Linear Regression, Decision Tree, and Random Forest to predict whether a customer will fail a payment or not. Apache Kafka is responsible for managing all the bank's events and processes them in real-time, allowing an easier exploration of data, in order to improve the credit scoring models, thus achieving a better performance.

Keywords: Banking, Credit Scoring, Kafka, Random Forest, Logistic Regression, Decision Tree, Data Mining

Resumo

A pontuação de crédito é a técnica analítica usada pelos bancos para avaliar o risco de crédito. É de extrema importância para o bancos conseguirem reduzir esse risco, e por isso estão, cada vez mais, a confiar em modelos estatísticos e de aprendizagem autónoma para essa tarefa. Este projeto propõe o desenvolvimento de uma infraestrutura que suporta a grande quantidade de dados gerados pelos bancos todos os dias. Este terá uma componente responsável pelo treino e teste de modelos de pontuação de crédito como Regressão Linear, Árvore de Decisão e Random Forest para prever se um cliente falhará ou não um pagamento. Apache Kafka é responsável por gerir todos os eventos do banco e processá-los quase em tempo real, permitindo uma exploração mais fácil dos dados, a fim de melhorar os modelos de credit scoring, conseguindo assim um melhor desempenho.

Palavras-Chave: Bancos, Pontuação de Crédito, Apache Kafka, Random Forest, Regressão Logística, Árvores de Decisão, Data Mining

Contents

List of Tables

List of Figures

Chapter 1

Introduction

African banks have a big challenge due to the high percentage of overdue loans. This percentage has been increasing around 21% year after year, along with credit volume, although by the end of 2015 the increase has been the lowest since 2012 at around 8% [3].

Humans are biased beings. Even with the best of intentions, it is almost impossible to make a decision without an emotional reaction or to be affected by some preconceived opinion. Machine learning (ML) can be used resolve such matters by making decisions based on data, rather than potential gender and cultural biases, even though, there are some cases where ML models have fallen into these problems [9]. Automating such processes also offsets the tendency for banks to focus on more significant customers over smaller ones, helping to democratize the process.

Granting a loan is a slow, costly, and laborious process. Most of the times it is done manually, starting by requiring some information from the customer, that passes through multiple evaluations and validations, where in the end are subjected to the decision of an employee. Finding ways to have better grounds on to whom grant loans, and lowering the risk of losses, is a priority that every bank must take into account.

Banks offer their clients a complete solution integrated across devices, channels, and products to provide a seamless experience. Therefore, banks have a full overview of customers activity at many different levels, from simple transfers to credit card movements with many details. Despite all the secrecy behind financial corporations, for some years credit scoring systems have been improved using statistical and machine learning techniques that rely on data, even if the best solution is not publicly available, lots of research has been done with some public data made available by some banks [13].

By implementing a system that gathers all the data, and assigns a credit score to each customer, most of the hassle referred could be avoided, and at the same time providing a better experience.

In order to simplify the loan process and lower the banks' overdue loans percentage, we implemented a system, that through the use of data and machine learning, which automatically attributes credit score to each customer, and also predicts if there is a risk of default in the upcoming months.

This project was developed at the company Innovation Makers[1] that made it possible for us to explore more about credit scoring in the financial world.

[1] https://www.inm.pt

The primary motivation for this project is to automate the process of credit granting and scoring, by taking advantage of machine learning algorithms that, with studied and analyzed features, predicts if credit should be granted or not, relying on the probability of a customer not being able to repay (defaulting).

In pursuance of that objective we developed an architecture able to extract all the customer data, transform it into features, and feed it into our credit scoring engine, that decides whether a customer is granted credit or not, using machine learning algorithms like Logistic Regression (LR), Decision Trees (DT), and Random Forests (RF). We started with LR, which is one of the most used statistical technique throughout the years in credit scoring, followed by the DT which allowed us to provide a straightforward explanation of the reasons behind the decision of acceptance or refusal of a credit, and in the end we chose to use RF since, recently, it has been considered to be the comparative point for new developments/implementations of credit scoring classifiers.

1.1 Contributions

This work provides a credit scoring engine capable of classifying a credit as risky or not, supported by an infrastructure that improves the development and maintenance process of exploring the data available.

1.2 Structure of the document

The rest of the document is organized as follows: Chapter 2 Fundamental Concepts presents basic concepts needed to understand the project starting with Section 2.1 Credit Scoring, followed by section 2.2 Machine Learning, and ending with section 2.3 Infrastructure. Chapter 3 presents an overview of some works related to the solution developed. Chapter 4 presents the solution to achieve the aims of our project, from the architecture to the credit scoring engine. Chapter 5 presents the evaluation of the architecture used, and also the methods used to evaluate the machine learning models. Chapter 6 presents the conclusions of this research and some future work.

Chapter 2

Fundamental Concepts

This chapter starts with an overview of Credit Scoring, allowing the reader to understand the concepts underlying our problem, followed by concepts regarding specific machine learning algorithms that are part of the ML engine. Section 2.3 presents important concepts to comprehend the Infrastructure developed, in order to support the data and the ML engine. Since it was the first time at Innovation Makers that ML was used, we were be responsible for all the process from aggregating all the data, to creating a dataset, and the deployment and validation of ML models.

2.1 Credit Scoring

Credit scoring is a set of decision models that aid banks when they need to grant a client credit. It is used to decide who will get credit, how much will it be, at which price they will get it at, and to find some strategies that will make it profitable for the bank. The most common way to understand credit score is to think about the probability of a client defaulting on a loan. The financial institutions' models that generate this probability of default from a credit score have to meet requirements in the Basel Accord banking regulations [30].

Basel is an international regulatory accord that regulates, supervision and controls the risk management within the banking sector. Since the application of the Basel Accords, banks are obliged to estimate the default risk of each segment of their loans, and as a consequence the objectives of credit scoring ranking shifted from merely ranking the customers to estimating the default risk more accurately, leading to more emphasis and importance on how the probability of default is calculated.

2.2 Machine Learning

Machine learning is a sub-field of artificial intelligence with the goal of enabling computers to learn on their own. In simple terms, a machine learning algorithm enables it to identify patterns in observed data, build models that explain the world, and predict things without having explicit pre-programmed rules and models. This observed data is typically electronic data collected, digitized human-labeled or obtained via interaction with the environment. For ML algorithms it is important that the data is of high quality and of a considerable size in order

to be able to make predictions accurately. Due to this dependence, ML is inherently related to other fields like data analysis, statistics, and optimization [25]. The challenge we have at hands is well known as a supervised learning problem. With pairs of inputs and outputs of past data, where as an input, for example, the account's age, the type of credit or the balance, and as an output, whether the client defaulted or not, the input is fed to a learner in order to determine the output.

Depending on the output being continuous or discrete, the supervised learning problem is referred to as a regression or classification problem, respectively. The learner's objective is to find a function that maps inputs to outputs, while avoiding 'memorizing' all mappings, by splitting inputs/outputs into different datasets, training and testing, and finding a function that generalizes beyond a training set, being able to classify unseen inputs into unseen outputs [20].

2.2.1 Logistic Regression

Logistic Regression (LR) is a statistical technique widely used as a start and comparison point in the context of credit scoring since it is the industry standard [4, 23].
Is known as a probability estimator that in combination with a decision rule, making dichotomous the probabilities of outcome, i.e., it can only take two possible values, is turned into a classification algorithm [2].

LR is based on the linear combination of input values, which are usually one or more independent variables, using coefficient values to predict a binary output.

The logistic function 2.1 is the one used to obtain outputs between 0 and 1 for all values of X, where X is one or more independent variables. Usually, the logistic function will fit an S-shaped curve form that allows a sensible prediction.

From the function (2.1) we can get to the logit function (2.2), where p(X) is defined as the probability of a specific binary output and Xs represent the independent variables.
In order to estimate the regression coefficients, β, the more general method of maximum likelihood is used because of its statistical properties.

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+...+\beta_k X_k}}{1+e^{\beta_0+\beta_1 X_1+\beta_2 X_2+...+\beta_k X_k}}. \tag{2.1}$$

$$p(X) = \log(\frac{p(X)}{1-p(X)}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k. \tag{2.2}$$

The basic intuition behind (2.3) to fit a logistic regression model is that it tries to estimate β_k such that when applying them to (2.1) it yields a number close to one for one of the classes and close to zero to the other one [14].

In other words, the resulting estimators are calculated iteratively and chosen to maximize (2.3). Those estimators are the ones most close to the data used. Since this method is widely used, it has been implemented into common software that provides LR [17].

$$l(\beta_0, \beta_1, ..., \beta_k) = \prod p(x_i) \prod (1 - p(x_{i'})). \tag{2.3}$$

2.2.2 Decision Tree

Decision Tree (DT) is a method that can be applied both for regression and classification problems. Tree-based methods partition the feature space into a set of regions, and then fit a simple model (like a constant) in each one. It consists of multiple splitting rules, starting at the root node, i.e., top of the tree. Regions are commonly known as terminal nodes or leaves of the tree and are the points where feature space is partitioned as internal nodes [14, 11].

In order to construct a suitable tree, the Decision Tree method selects input variables and split points on them. This selection is chosen using a greedy algorithm, known as recursive binary splitting, that uses a top-down approach, which takes the best split at each step, rather than testing every further option and decide.

There are multiple DT implementations like ID3, CART, C4.5, MARS, among others. They differ in terms of the splitting criterion, whether it builds for regression or classification, the technique used to eliminate/reduce over-fitting, and also if it can handle incomplete data or not.

DT can be applied to regression and classification problems, here we explain the basic notions using the regression case.

The tree construction is done by dividing the feature space, taking into account all of the different values for each feature, into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$. For each observation falling in a region, is done a prediction, that is the mean of the values for the response values of the observations in that region. These regions are built by diving the predictor space into high-dimensional rectangles, to be more interpretable, with the objective of minimizing the Residual Sum of Squares (RSS) given by (2.4).

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2. \tag{2.4}$$

RSS is commonly used to find an optimal parameter and is a measure of the discrepancy between the observations and the mean of the values of a region.

Multiple tweaks can be done when fitting a DT model like which splitting criteria to be used, either Gini Impurity or Entropy.

For example, in case of CART implementations, Entropy is defined in (2.5), where $p(i|t)$ is the proportion of the samples that belong to class c for a particular node t. Entropy is therefore 0 if all samples at a node belong to the same class, and is maximal if we have a uniform class

distribution.

$$-\sum_{i=1}^{c} p(i|t) log_2 p(i|t). \tag{2.5}$$

The Gini Impurity can be understood as a criterion to minimize the probability of misclassification (2.6) [11].

$$\sum_{i=1}^{c} p(i|t)(1 - p(i|t)). \tag{2.6}$$

Like Entropy, the Gini Impurity is maximal if the classes are perfectly mixed. If minimal, it means that a node contains more observations of a single class.

However, practical results from either are very similar, and since Entropy requires the computation of logarithmic functions, it would be preferable to use Gini Impurity which leads to faster implementation.

Comparing this to other types of regression models, this offers an easy way we can interpret results and a graphical representation, regardless DTs usually do not have the same level of predictive accuracy as [11].

2.2.3 Random Forest

Bootstrap aggregation, or **bagging**, is a general-purpose procedure for reducing the variance of a statistical learning method. It generates different bootstrapped training datasets by taking repeated samples from two-thirds of a single dataset, then training the method in each generated bootstrapped dataset, obtaining each prediction, and in the end averaging all of them.

To discover which number of datasets that should be generated, we can rely on a very straightforward method to estimate the test error of a bagged model. Each bagged tree takes around two-thirds of the samples, and the remaining ones are entitled **out-of-bag** (OOB) samples. After generating the classifiers, we have that one-third of samples that are not present in each bootstrapped dataset, that is used to generate an OOB classifier by aggregating all of the predictions. OOB estimate for the generalization error is the error rate of the OOB classifier on the training set [11].

Random forest (RF) is an ensemble ML method that consists off a concept where weak learners are grouped into a strong learner to make a final decision [6]. In this case, the weak learners are un-pruned DTs, where each is trained on a random sample of the data and at each split node only a random subset of the features are considered, after that each tree votes for the most popular class.

When using RF, the trees are more independent thanks to a combination of bootstrap samples and random draws of features. Since RF are a form of bagging, and the averaging over trees can substantially reduce instability that might otherwise result, the gains from averaging over a large number of trees (variance reduction) is higher.

Another noticeable gain with RF is that more information may be brought to reduce bias of fitted values and estimated splits. Usually there are a few features that dominate DTs fitting process due to their above-average performance and as a consequence other features that could be useful are rarely selected as splitting variables. RF computation of a large number of DTs, allow each feature to have several opportunities to be the feature that defines a split since only a portion of them is randomly selected at each time [14].

For this method there are 2 important parameters to be tuned, in order to meet performance and computational constraint:

- **Number of trees** The number of DTs is important to be defined with a high value between 100 and 500, so RF can perform well. As this value is increased, computational requirements are higher for the model and also, at a certain point, it is no longer beneficial;
- **Number of features sampled** At each node, DTs makes a decision taking into account some features randomly sampled, leading to the feature and split threshold that grants the best division of the classes. Breiman (2001) [6] suggests beginning with it set to the square root of the number of features in the dataset.

RF algorithm takes the following steps [31], where *ntree* is the number of DTs and *mtry* the number of features sampled:

1. Generate a random bootstrapped dataset from two-thirds of the original dataset;
2. Take *mtry* random sample of the features;
3. Generate a split using the previously selected features;
4. Repeat the 2 previous steps until the tree grows as desired, without pruning. Each tree is created based on random samples, and each split node are only considered a *mtry* random samples of features;
5. Repeat all previous steps until *ntree* is reached;
6. Evaluate the model by making predictions on OOB data for each tree.

2.3 Infrastructure

This section presents the concepts needed to comprehend what technologies support our machine learning implementation.

2.3.1 Databases

A fundamental decision in business is which type of database fit its needs as priorities and problems changes. There are two groups of databases, RDBMS (Relational Database Management Systems) and NoSQL, which one is better depends on the problem that needs to be solved.

RDBMS databases are useful if the data remains unchanged and structured, and is not going to grow exponentially. In contrary, if dealing with problems that require vast amounts of data,

and especially combined from multiple sources, leading to unstructured data, there is a big chance that a NoSQL database handles the data better.

RDBMS databases have been widely used in the past 40 years [7] and have some characteristic, such as data being stored in the format of a row and column in a table, where relationships are represented between data and tables are joined through relational links. Usually, RDBMS' support strict ACID (Atomicity, Consistency, Isolation, and Durability) transactional consistency, which is very important to guarantee database reliability.

- **Atomicity** means that all commands executed in a transaction either succeed or all fail and rollback is done, causing all data to be reverted to the state before starting the transaction;
- **Consistency** says that all committed data must be consistent with all data rules including constraints, triggers, cascades, atomicity, isolation, and durability;
- **Isolation** takes care that other operations cannot access data that has been modified during a transaction that has not yet completed;
- **Durability** consists in granting that once a transaction is committed, data survives system failures, and can be reliably recovered after an unwanted deletion.

However, we have NoSQL databases that in most cases only support weaker or eventual consistency (BASE) instead of ACID, in order to achieve availability and partition tolerance [7].

BASE concepts consist in:

- **Basically Available** since, in most cases, databases are replicated across multiple storage systems leading to a high degree of availability;
- **Soft-state** meaning that the state of the system may change without being queried, due to possible updates of replication nodes;
- **Eventually Consistent** at some point the system becomes consistent.

There are 4 types of NoSQL databases:

- **Key-value** stores are the simplest. Every item in the database is stored as an attribute name (or "key") together with its value. Riak [1], Voldemort [2], and Redis [3] are the most well-known in this category;
- **Wide-column** stores data together as columns instead of rows and are optimized for queries over large datasets. The most popular are Cassandra [4] and HBase [5];
- **Document** databases pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents. MongoDB is the most popular of these databases;

[1] http://basho.com/products/riak-kv/
[2] https://www.project-voldemort.com/
[3] https://redis.io
[4] http://cassandra.apache.org/
[5] https://hbase.apache.org/

- **Graph** databases are used to store information about networks, such as social connections. Examples can be Neo4J [6] and HyperGraphDB [7].

Regarding the decision of which database fits our needs, we must take into account some important aspects like how fast we need to read and write data, whether our data is structured or not, how much will we need to store, among others.

Lately, non-relational databases have played an integral role of enabling new developments in machine learning due to their ability to collect and store large volumes of nested, semi-structured, and unstructured data [28].

One well-known NoSQL database is MongoDB [8]. It offers the data model flexibility, as many data sources nowadays would not fit well into a rigid row and column format of a typical relational database, and allows constant experimentation to uncover new insights on data. Scalability is also one of the most reliable characteristics of MongoDB since it allows data and model parallelism that can be used to reduce models training time [26].

2.3.2 Event-Driven

In this type of architecture, an event is published when something notable happens, such as when a customer realizes a transaction. Actions are triggered on services that subscribed to it upon its reception. These actions can lead to more events being published, allowing an asynchronous flow that compartmentalises different problems that each service solves [24].

This architecture is suitable, for example, if we have multiple services from different sources, where a client generates information, an event is published every time it happens, and we want to aggregate all that data. In order to do that, a service would subscribe to all of the events and take action on every received event.

Advanced Message Queuing Protocol (AMQP) was created to eliminate the gap that existed in interoperability of different asynchronous messaging middlewares. It came to improve the middleware domain, before Java Messaging Service (JMS) was the best standard in asynchronous messaging, regardless of being limited to Java. Its design was mainly focused on finance community requirements, like scalability, reliability, and strict, precise and exacting performance [8].

RabbitMQ[9] is a message system, a variant of the Advanced Message Queuing Protocol (AMQP), well-known for its performance and facilitates the creation and management of high-availability architectures. It can be used with two different approaches, one has a queue-like system, and other has a publish-subscribe message system [19].

ActiveMQ[10] is an open source message-oriented middleware (MOM) based on JMS capable of providing high availability, performance, scalability, reliability, and security.

Apache Kafka[11] is similar to a messaging system and was built for processing massive volumes of transaction logs ingestion, and other real-time data feeds. At its essence, Kafka

[6] https://neo4j.com/
[7] http://www.hypergraphdb.org/
[8] https://www.mongodb.com/
[9] https://www.rabbitmq.com/
[10] http://activemq.apache.org/
[11] https://kafka.apache.org/

provides a durable message store, similar to a log, run in a server cluster, that stores streams of records in categories called topics. Kafka topics are used for input, output, messaging between operators, the durability of local state, replicating database tables, checkpointing consumer offsets, collecting metrics, and disseminating configuration information.

In the world of message systems, Kafka is known for a set of characteristics [8] as:

1. Long-term message storage, that consists in purging messages automatically after a defined retention time and can be configured per topic;

2. Message replay, allow consumers to replay messages when needed, for example, if after sometime collecting movements from a source, and now you want to add a category to each of the movements this feature allows you to do that not only for the incoming movements but for all the previous ones, depending on the configurations applied to Kafka;

3. Frameworks, libraries and tools offered by Kafka's ecosystem adding functionalities like Kafka Streams to perform stateful and fault-tolerant stream processing.

Figure 2.1 presents an example architecture of Kafka composed by Producers, Consumers, Connectors, and Streams. Producers are responsible for publishing data into a Kafka topic, whereas Consumers are the ones that subscribe to a topic and process the stream of records produced to it. As an alternative, Kafka Connect framework exists to facilitate scalable and reliable streaming of data into and out of Kafka. There are lots of implemented and certified connectors to import/export data into/from Kafka. Kafka Streams allows processing records of published topics and re-publish them to be consumed [21].

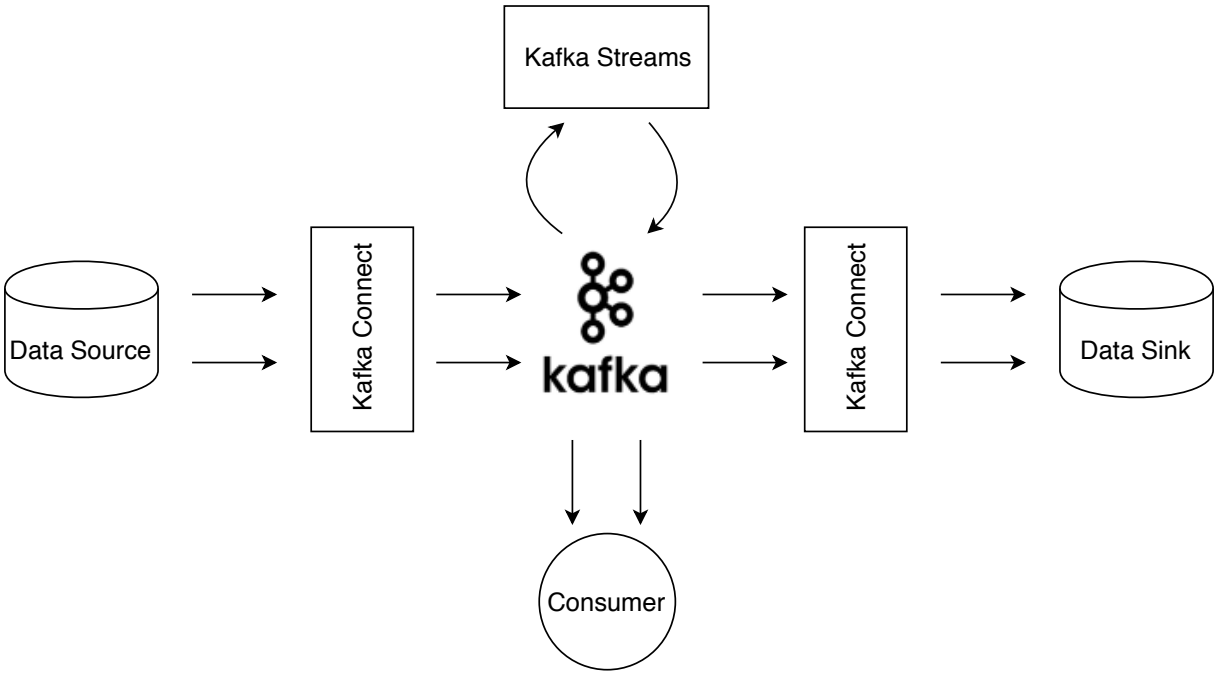

Figure 2.1: Kafka architecture and some of its components

2.4 Summary

In this chapter, we presented concepts that must be taken into account to understand the underlying problem of developing a credit scoring system based on ML. Started by introducing credit scoring as a set of decision models that aid banks' granting credit to a client. Then explained the algorithms that we took into account when developing our solution, from Logistic Regression that is the statistical technique more used in this industry, followed by Decision Tree which can be used either for regression or classification problems and is also known for its explainability, and finished with Random Forest that are based on multiple separated decision trees that are grouped into one strong learner. This chapter ends with some concepts of databases and event-driven architecture that were important to support our ML implementation.

Chapter 3

Related Work

With this chapter, we intend to address and present some related solutions to the project that has been developed. In section 3.1 we refer to solutions used to achieve some characteristics required for the company environment as well as for the bank's production system, like scalability, reliability, and near real time. Section 3.2 focus on implementations of machine learning algorithms applied to credit scoring and risk, related to the development of the machine learning component of our project.

3.1 Infrastructure

Selecting the right tools to support an ML engine is a challenging task due to the high number of options. The available tools have advantages and drawbacks, and many have overlapping uses. The importance of a well-thought infrastructure that fits all of our needs relies on how we can retrieve, process and continuously feed data to our credit scoring system. As a requirement, we had to guarantee that our system is scalable, reliable and near real-time. This section analyzes some technologies/solutions that fill our needs.

3.1.1 Event Processing

Apache Kafka implements its logs in a fault-tolerant and scalable way. It only allows access methods as appending write and sequential reads from a given offset, and as a consequence it avoids the complexity of implementing random-access indexes, being able to provide better performance than systems with richer methods as RabbitMQ and ActiveMQ [1].

Kreps et al. (2011) [21] present for the first time Kafka showing how the combination of its components, messaging, storage, and stream processing turns it into an appealing streaming platform. They conducted an experimental study against RabbitMQ and ActiveMQ, divided into two sections Producer Test and Consumer Test.

On the former, Kafka was able to publish messages at a rate of 50,000 per second for a batch size of 1, each with 200 bytes, from a total of 10 million messages. Kafka outperformed RabbitMQ by producing twice more and ActiveMQ by higher orders of magnitude. On the latter, RabbitMQ and ActiveMQ consumed four times slower, probably due to Kafka's efficient

storage format.

3.2 Credit Scoring and Default Risk

The importance of credit scoring for the banks and regulators in the financial sphere lead to increased research of this topic, from papers evaluating the impact of credit scoring concerning profitability to the development of novel machine learning techniques and the improvement of conventional statistical methods [5, 18].

Logistic regression (LR) is one of the most used statistical technique in credit scoring. It has been extensively used since the early attempts to estimate borrowers default probability by using their characteristic information.

Since a long time ago, Steenackers and Goovaerts (1989) [29] applied LR to develop a scoring system for personal loans, therefore to achieve their objective, they had to find the best variables that discriminate between a 'good' loan and a 'bad' loan. The dataset used contained good and bad loans, and also refused loans along with personal characteristics, financial and professional situation, and some more. They used a technique known as stepwise logistic regression that consists on starting from the variable with most predictive power and start adding one at a time the ones that gave the best improvement to the LR model. Among their results, they have found out that variables like the number of previous credits and possession of a house were the most important criterion, and were able to achieve 70% of correct classification for good loans and 79.1% for bad loans.

Baesens et al. (2003) [4] presented results of a comparison between state-of-the-art classification algorithms, assessing the performance of each using the percentage of correctly classified cases (PCC) and the area under the receiver operating characteristic curve (AUROC) over eight different real-life datasets. From the state-of-the-art algorithms like Neural Networks and Least-Square Support Vector Machine with Radial Basis Function kernel (RBF LS-SVM), a simple classifier like LR still yields a good performance predicting failure probability, which indicates that it LR can be a good entry point for newcomers in this industry.

Decision Tree is a classification technique commonly used in modeling credit scoring, also known as Classification and Regression Trees (CART).

Lee et al. (2006) [22] did a comparative analysis to show the efficacy of CART, and Multivariate Adaptive Regression Splines (MARS) compared to classification techniques as Logistic Regression, Neural Networks, and some others. They used a dataset provided by a local bank constituted by 8000 samples with nine different independent variables. Lee et al. (2006) state that DTs are very easy to interpret and hence marketing professionals can use the built rules in designing proper managerial decisions, which is essential due to restrictions imposed by governments nowadays. CART outperformed traditional methods like Logistic Regression having around 7% more correct classification rate.

Ensemble methods aim to increase predictive accuracy through the combination of predictions of multiple base models. Their success depends on two key factors, the strength (accuracy) of individual base models and the diversity among them. Homogeneous ensembles use as base models the same classification algorithm [15, 23].

A well known homogeneous ensemble method is Random Forest (RF). It has been first presented

by Breiman et al. (2001) [6]. Lessman et al. (2015) [23] in his study compares 16 individual classifiers, eight homogeneous ensembles, and 17 heterogeneous ensembles, tested against seven different datasets. RF is among the best classifiers after extensive testing taking into account everything that has been done in previous years regarding credit scoring. The real-world datasets that have been used had several independent variables essential to building a credit scorecard and also a binary response indicating whether or not a default event was observed. Regarding data preprocessing and partitioning, techniques like mean/mode replacement for numeric/nominal attributes imputation of missing values and a split-sample setup is well-established in the literature [4] consisting on randomly partition a dataset into a training and hold-out test set. In order to correctly evaluate all the 41 different classifiers, four accuracy indicators were used: the percentage correctly classified (PCC), the area under a receiver-operating-characteristics (ROC) curve, the H-measure, and the Brier Score (BS). Lessman et al. (2015) [23] state that between individual and homogeneous classifiers, RF was the one that gave the most accurate probability of default estimates. This benchmarking study draws some important conclusions, and it refers to RF as the comparison point for any other developments/implementation of credit scoring classifiers.

A question that is always debated in literature is the trade-off between model interpretability and complexity. Thomas et al. (2017) defend that a small improvement in predictive accuracy can lead to substantiate financial rewards. Whereas authors were rejecting this view due to the lack of compliance with regulatory frameworks such as Basel since complex models are challenging to interpret [12]. Even though, with some disagreements there are still ways like rule extraction techniques, making some complex models' mechanisms easy understandable [16].

3.3 Summary

In this chapter, we started by describing the value that message systems like Kafka, brought to our system. Followed some solutions to the credit scoring problem nowadays using the traditional logistic regression, decision tree, and random forest techniques.

Chapter 4

Solution

This chapter presents a credit scoring engine on top of an infrastructure based on Kafka. Section 4.1 starts by giving an overview of the reasons why we needed an infrastructure based on Kafka, as well as techniques applied to build our datasets and to train our models. In section 4.2, we explain how bank's database works and its organization since it is our primary source of data and the components that make part of the infrastructure, that supports our credit scoring engine by retrieving and processing data continuously. Section 4.3 focus on the Credit Scoring, where we explain the solution that led to our credit scoring engine.

4.1 Overview

Our objective is to give the bank a tool that allows them to decide to whom they should grant credit based on their clients' historical data. In order to achieve this, we started by exploring the data and where it was stored, an RDBMS known as IBM DB2, which is an old database commonly used by banks. There is a lot of business logic behind the database tables organization. Also, it is raw data, in the sense that it is unprocessed real data, requiring some pre-processing before we can feed it to a machine learning model.

While dealing with that data, we concluded that we had to use a system capable of retrieve, process and store it, so we could have a pipeline that gave us the data semi-processed. Also for the company to grow and use this system for other applications besides credit scoring and recommender systems, being able to process information in real time would be a plus. For example, movement categorization where each movement that reaches the database is classified on the fly and is shown to the client with the proper classification.

An infrastructure that gives us flexibility, scalability and efficiently processes data, is the main focus of our solution. We chose Apache Kafka to support our machine learning engine. Apache Kafka is responsible for consuming all data that the bank collects into the IBM DB2 RDBMS, from every transaction each client does to any change made to an account. Using one of its components, Apache Connect, it is continuously polling and processing new or updated data. This processing is done using a component of Kafka, Kafka Streams, that aggregates and transforms the data in a useful way.

All the processed data is stored in MongoDB, a non-relational database, due to the characteristics of some of the many projects that make use of this infrastructure and as we stored our

models' features on it.

The Credit Scoring engine is responsible for generating our datasets using the data stored in MongoDB, that was processed by Kafka, and also through ML models like Logistic Regression, Decision Tree and Random Forest, can predict if a customer will default or not and if a customer represents a good or bad investment for the bank.

4.2 Infrastructure

We were responsible for setting up all the infrastructure and multiple environments in order to be able to manipulate the data and to support new projects.

Regarding infrastructure, we have a production snapshot of bank's database IBM DB2 dated April's 2017, a Confluent Platform Open Source installation, which contains all the previously described components of Apache Kafka, a MongoDB database and the Credit Scoring engine.

Currently, we have three different environments: Development, Quality, and Production. Each one has its purpose, and we started with the Development environment where we do any experiments we need to without having to worry if anything goes wrong. The next in the line is the Quality environment in which we have all the stable features, and is where the company's client tests what has been developed in order to evaluate if every requirement is as requested. The last stage will be the Production environment where is deployed what has been approved by the bank in the previous stage and is made available to the bank's clients or workers. The bank's database IBM DB2 snapshot is the same in Development and Quality environments.

4.2.1 IBM DB2 source database

As the main component of our infrastructure, we have the bank's IBM DB2 database, which is where is all the data that we will use. This database is where information of clients like transactions, changes made to clients' account and all the credit information is stored. This data is diffused across multiple tables without any connection, and its columns' names are not explanatory enough to be understood, which turned to be a complicated task, since we had to explore this database based on its documentation, using a terminal emulator, known as Mocha TN5250[1], to access it.

We started to search for the relevant tables and tried to find others related to them through the Mocha TN5250. As we got to the relevant tables for our project and explored its columns, we found that most of them did not have nor a time stamp, nor an incrementing column, which further in our development it became troublesome due to some requirements of Kafka Connect.

The business logic that the bank has regarding a client and its accounts can be confusing in term of names. An entity is a representation of a physical client, a client can be composed of one or more entities, and each client can have one or more accounts. This relation between entities, clients, and accounts is based on four tables:

Entity table where all the information of a physical client is stored like name, employer's name, role, nationality, among others.

[1] https://www.mochasoft.dk/tn5250.htm

Client table contains data that differ for each client as if it represents a company or an individual, among other data that's not relevant for our project.

Client-Entity table is where resides all the current connections between clients and entities, and also the degree of each connection.

Account tables store information regarding each account type like different types of balances, the status of the account, currency, among others. The more relevant type of account will be the credit account.

Besides the four tables regarding clients information, there are other also relevant to the credit scoring problem:

Credit Proposal table contains data about whether the credit was accepted or not, credit amount and credit type.

Average Balance table has the accumulated balance of each account per month.

Credit Account History table has all the changes made to a credit account, from it we can get information when a credit went to the default state.

Our first approach was to gather all this information from those tables using complex SQL queries, which turned out to be infeasible due to the time it took and how unmaintainable it would be. Nevertheless, we followed the steps depicted in figure 4.1, and were able to use the data extracted to a CSV, pre-process it and then feed that dataset to our ML models.

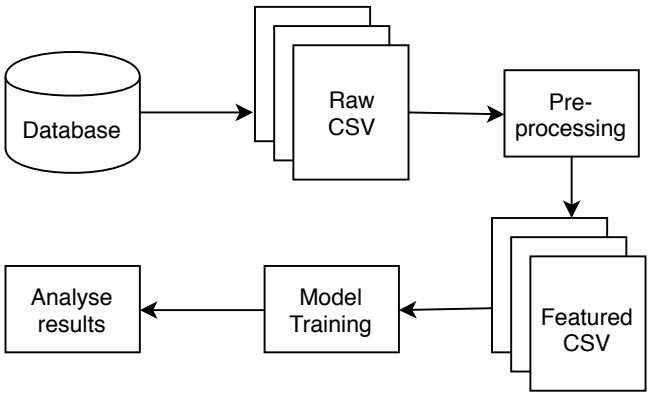

Figure 4.1: Naive approach used to investigate data and its inner relations.

Due to the nature of the problem at hands, we whether had to implement mechanics to retrieve and pre-process only the most recent data and add it to the current dataset, and also a way to detect updates on the database, or find a way to continuously retrieve the data, pre-process it to generate our dataset, and at the same time this data could be reused by other projects without having to overload the database and re-process it.

We chose to use Apache Kafka, which allowed us to continuously retrieve, pre-process and make use of the data for multiple projects within the company. To make this work, we had some setbacks due to the database organization and for being an old technology that we will further detail.

4.2.2 Confluent Platform

As previously mentioned, we chose Apache Kafka which was built to handle huge volumes of data in real-time and works on top of a publish-subscribe pattern using consumers and producers to achieve it. We decided to go with the Confluent Platform Open Source since it includes Apache Kafka and also brings in some components that will help us manage our data, like Schema Registry, pre-built connectors for Kafka Connect framework which we used to build our pipeline.

Apache Kafka

Apache Kafka is similar to a message queue system and can process events as soon as they arrive, and each of these events is published into a topic. Each topic can have zero, one or more consumers. In our solution an event will correspond to an insert or update into IBM DB2 bank's database, so every time there is new/changed data it will be published to a specific topic. For each table we want to keep track of, we will have a topic.

Regarding topic naming there is no convention, so we decided to follow a simple structure like `<message type>.<dataset name>.<data name>`, where message type can be ETL if we are using a database as source or streaming for topics created by streaming processing pipelines, dataset name in our project will be Banka, a friendly name used to refer to IBM DB2 database, and the data name will be analogous to a table name in the traditional RDBMS systems. This topic naming convention will allow us to create topics automatically and make sure that everything is correctly named when we have multiple applications publishing to Kafka.

An Apache Kafka cluster is composed by one or more Broker instances of Kafka, that are managed and coordinated through a Zookeeper instance, from which one or more Consumers and Producers will be notified regarding any changes to brokers. Kafka brokers are stateless, and that is why Zookeeper has an important role, it is responsible for the cluster coordination and to keep Producers and Consumers up to date. A Producer searches for a Broker and produces messages to it without waiting for any acknowledgment, this is one of the reasons why Kafka can handle a heavy load of traffic. Due to Kafka Brokers being stateless, a Consumer will have to keep track of how many messages have been consumed through an offset, it is the Consumer that asks the Broker to have data ready to be consumed and also is responsible for sending the offset value to the Zookeeper.

We decided to go with a multi-broker cluster so we could achieve scalability and automatic fault tolerance. We were able to set up a Kafka cluster with three brokers communicating with each other and a Zookeeper instance coordinating the cluster, unfortunately, when we added Kafka Connect also in the distributed mode we were not able to stabilize it, further, we will detail the problem.

Our source of data will be IBM DB2 bank's database, as previously mentioned, is a type of RDBMS that can be accessed using a well-known interface, Java Database Connectivity (JDBC). We will use Kafka Connect, that provides a source JDBC connector developed and tested, to be continuously polling it, retrieving any change that occurs, either inserts or updates, and publishing it into a Kafka topic.

Kafka Connect

Kafka Connect is the framework that will allow us to retrieve all the data from IBM DB2 database into Kafka. Kafka Connect is composed of 3 main components: workers, connectors, and tasks.

Workers are the processes responsible for executing connectors and tasks and can be of two types, standalone or distributed. On standalone mode, there is only a single worker responsible for all connectors and tasks, whereas the distributed mode can have multiple workers that if belonging to the same group will coordinate with each other to redistribute all the connectors and tasks. The distributed mode provides scalability and automatic fault tolerance.

A connector is a job responsible for managing data flow and instantiating one or more tasks that will be moving the data. Connectors can be of two types, a source connector is responsible for reading data from an external source and write it to a Kafka topic, and a sink connector is responsible for reading a Kafka topic and write it to an external sink.

One of our requisites was to copy all the relevant tables, on IBM DB2 database, into Kafka, and Kafka Connect facilitated all of this process by polling continuously the database. The way JDBC source connector[2] can copy data incrementally from a database depends on which mode we use. To be able to support inserts and updates, we had 2 out of 5 modes that we could use:

- **Custom Query** that allows copying tables through the use of a query, but to detect updates it must keep track of the offsets itself.

- **Timestamp and Incrementing Columns** is the most robust mode but requires that each table have an incrementing and timestamp columns to be combined.

Due to IBM DB2 database modeling choices, there were no incrementing nor timestamp columns to be used for the Timestamp and Incrementing Columns mode, which caused us to think if we could make use of Kafka Connect. An alternative to Kafka Connect would have been to implement everything that it offers and finding a way to keep track of the offsets, which would be laborious. To solve this problem we explored more IBM DB2 database and found that most of the tables had a corresponding table with the history of changes. This history tables contained columns like date, time and an operation ID, which is unique per day, that could be transformed into incrementing and timestamp columns. Then we created views that added two columns to each table, where the timestamp column was a combination of date, time and the operation ID as microseconds and the incrementing column the concatenation of the date with the operation ID.

Kafka Connect provides a REST API to manage connectors. With it we can list, create, update connectors configurations and states, and also to start, restart or stop any task in case of failure. To facilitate our work, we decided to create a bash script that allowed us to do all of those operations via more straightforward commands since it was complicated to manage all the connectors configurations one by one.

[2]https://docs.confluent.io/current/connect/kafka-connect-jdbc/source-connector/index.html

We ran into some problems while setting up Kafka Connect in distributed mode. As mentioned before, this mode would allow us to achieve scalability and automatic fault tolerance by executing multiple workers on separate Kafka machines. In this mode, when a connector is added to the cluster, it starts doing a re-balance that consists in trying to keep all the total work of the cluster balanced across the workers.

In our case, launching multiple connectors kept Kafka Connect constantly re-balancing affecting its performance, and unfortunately even after much work and testing multiple configurations we were not able to get Kafka Connect stable, so we decided to use the standalone mode.

Avro and Schema Registry

Kafka is a powerful tool that accepts bytes as input and sends bytes as an output without any protocol constraints. Avro is a data format, which has a JSON like data model but uses schemas, and that works well with streaming and it was our choice for multiple reasons:

- **Embedded Documentation** allowing each field to be documented, which for us was important since our primary source of data, IBM DB2 database, had column names that were not easy to understand and its documentation had to be read through Mocha TN5250. This way any developer who needs that data can easily understand what each field is reducing data interpretation misunderstanding;
- **Compatibility** using schemas avoids all the problems that could arise from modeling and data format changes. By keeping an evolution of each data format, for example, due to a new application that requires a specific field that others do not, it will be seamless for the application that doesn't require that field because it will be removed upon deserialization after checking that it was not being used before.

Schema Registry is responsible for storing Avro schemas sent by Producers, and keeps a versioned history in Kafka topics, that will be handled to the Consumers so they can read the messages. It is responsible for managing the schemas, by validating before receiving or sending them to producers or consumers upon request.

Using Schema Registry we can impose that from the point we parse the data and set a schema, we do not need to be worried about parsing and trying to make transformations to the data at further stages of our pipeline.

Kafka Streams

Kafka Streams is a library used to build real-time applications, working as a consumer and producer, that transform messages from a topic and publishes into a different one. It focuses on processing information continuously, concurrently, and in a message-by-message way, and at the same time allow to do it in a distributed and fault-tolerant way. It was beneficial for our project since it allowed us to process and aggregate lots of data from multiple topics, facilitating the data science work later.

Our source of data, IBM DB2 database, did not allow us to have the information of a client easily, either its personal information or its balances or all the accounts it has, so we decided

to aggregate all of this data to be able to use it later for our credit scoring engine. Moreover, all the data types on the database were mostly in ways we had to transform them: dates were decimals, most of the strings needed to be trimmed and decimals that could be integers.

To achieve our goal of facilitating data science work later, we had to create multiple streams that are divided into two groups, Client-Entity, and Credit. Each group is sub-divided into categories, domain, stream, and sink. The domain is where we will have our generated Avro classes from an Avro file. The stream is where we will be doing type transformations, and creating some fields composed by others. The sink is used to write what we have been transforming and aggregating into a sink database, in our case MongoDB.

Client-Entity group will be responsible for aggregating all the changes made to an Entity's Client, whenever an Entity is added or removed from a client. The focus will be on the Client, so we wanted to have all the changes in each relationship with an Entity, to be able to know the Clients relationships at a specific date to retrieve information like the accounts and balance history of each Entity.

We had to use four different tables that were previously retrieved into Kafka using Kafka Connect. The steps taken to achieve our objective were:

1. Extracting the Avro schema of each table from Schema Registry, where data types were not as we needed, and manually create other Avro schema with the new data formats we want to attribute and add some comments explaining each field;

2. Created a Kafka Streams instance to transform the data from the extracted Avro schema into the described formats in the new Avro schema and publishes it to another topic;

3. Created a Kafka Streams instance responsible for aggregating the different topics, by applying some business logic necessary, generating the final object containing the relationships of each Entity with Clients and publishes it back to Kafka under another topic;

4. Created a final Kafka Streams instance that will be responsible for writing our Entity object into our database of choice, MongoDB.

Credit group will be responsible for aggregating everything related to credit, information of the credit proposal and the dates when it states changed throughout the time. To accomplish this, we used two of the previously retrieved tables using Kafka Connect. The steps taken to achieve this were similar to Client-Entity group. The final object will be for each proposal, and it will contain a list of the state changes and its respective dates, that were extracted from the Credit Account History.

4.2.3 MongoDB sink database

To complete our infrastructure, we decided to store into MongoDB the information that has been transformed and piped through Kafka, and later, the pre-processing done to that information that is used by our Credit Scoring engine. We decided to use it since it offers flexibility regarding data model, which we benefited while pre-processing and generating new features and also for future projects, and due to its scaling abilities.

MongoDB is a NoSQL database based on collections of documents, where each document has key-value attributes. In our project, each collection corresponds to a database table, or in cases where we used Kafka Streams, corresponds to the final object, Credit or Entity objects referred previously.

We write to MongoDB in two different ways:

1. Using a community developed sink connector, Kafka Connect MongoDB[3], which similarly to other connectors allow through a configuration file to read a Kafka topic and write it to a MongoDB collection. This sink connector allows specifying a key composed by multiple fields, which was relevant for the cases where we just wanted a plain copy of the table from the IBM DB2 database;

2. Through Kafka Streams, we manually write using the MongoDB Java Driver and complement it with Morphia[4], which is a JVM object document mapper that we use to map our java entities into MongoDB documents. Morphia allows us to easily annotate classes and save them to MongoDB, facilitating the work when we already have done transformations and had java classes defined. It also creates collections when they do not exist and also permits to define indexes.

4.2.4 Data flow

After describing and explaining the components that our infrastructure used, we will give an example and demonstrate the role of each component, shown in figure 4.2, throughout the process of receiving data in real-time.

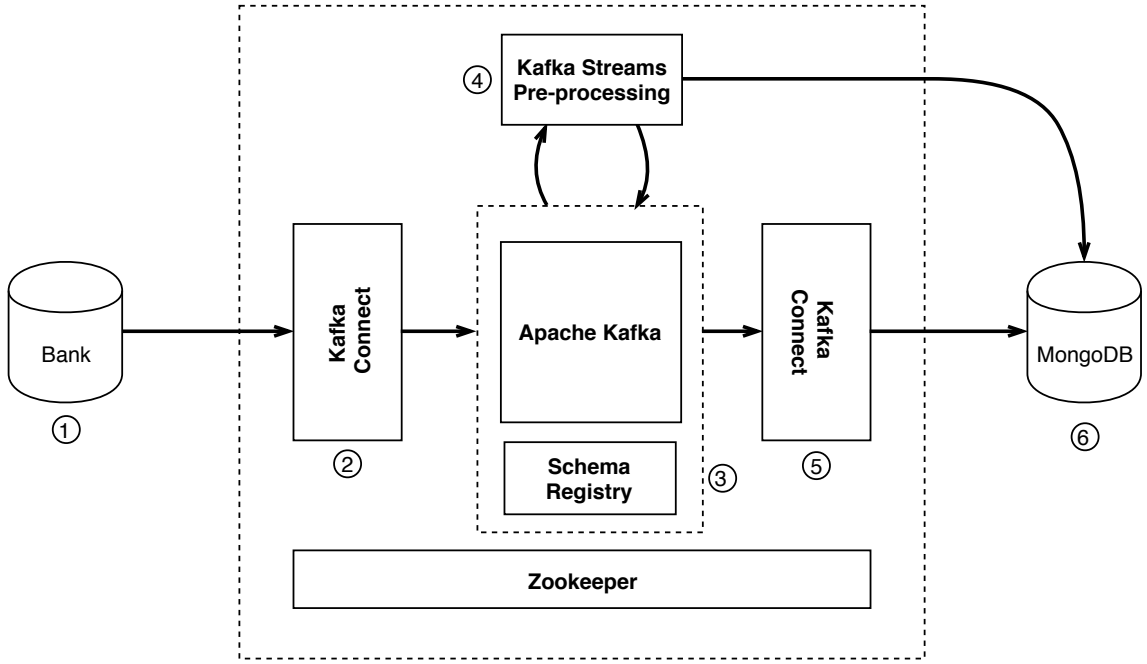

Figure 4.2: Kafka architecture

[3]https://github.com/hpgrahsl/kafka-connect-mongodb
[4]https://github.com/MorphiaOrg/morphia

As we mentioned earlier, our source of data is the IBM DB2 bank's database, and for that reason when there are inserts or updates, regarding relevant data, our infrastructure receives that data, process and save it for our credit scoring engine. This data can be generated by the Client, by the bank's employees or by database routines.

For example, when a Client goes to a bank's branch and asks for a credit, the bank's employee after discussing and presenting it to the client, submits a proposal. The action of submitting a proposal initiates the data flow process, in this case, when it reaches the bank's database, the source connector responsible for the credit proposals table, that is continuously polling it for data, will detect that there was an insert and will publish that data, in form of a message, into the respective Kafka topic.

Now that this data is on a Kafka topic, it will be processed by any Kafka Streams instance that has subscribed to that topic. This subscription means that it will be continuously asking Kafka if there is any new message on that topic. On this specific case, it will be the Kafka Stream's Credit group described earlier that will take action, after processing it will a message containing the changed data will be sent to a new Kafka topic.

The final step of our data flow consists in writing the processed messages into MongoDB. Depending on each case, there are two ways we do that, either with a Kafka Streams instance using Morphia, or with the community developed MongoDB sink connector. In this case, we use another Kafka Streams instance using Morphia to write our object to MongoDB and to create an index for the client number.

Other relevant examples could be, when a Client open or close an account, or gives access to other Client, or when a credit enters in default.

4.3 Credit Scoring Engine

Our Credit Scoring engine is responsible for generating datasets using the data that Kafka processed and stored into MongoDB, and, after training and testing ML algorithms, is also capable of either predicting if a customer represents a good or a bad credit investment, or if a customer will enter in default in a future month. We decided to create two datasets since we wanted to test two different cases.

In the first case, where we predict if a customer is a bad or a good credit investment we took into account each credit proposal accepted and gathered the information about the Client in the previous months. On the other one, we will try to predict if a customer will fail its next payment, thus entering into default, where we trained our models using information of the customer of each month since the credit was accepted.

We have tested a statistical technique and some ML algorithms. We started by using the traditional statistical technique used in credit scoring, the Logistic Regression. Followed by the Decision Tree algorithm which was essential to extract more insights of its predictions, allowing us to infer which features were essential and which ones were not relevant. Moreover, we tried to achieve the highest accuracy using a Random Forest algorithm, which also allowed us to analyze the importance of each feature.

4.3.1 Datasets creation

To the Credit Scoring engine be able to predict either of the two cases earlier presented, we had to prepare two different datasets with relevant features to feed our ML models.

As mentioned earlier, Kafka Streams allowed us to do some pre-processing to the data by cleaning and aggregating it in some cases. This data aggregations resulted in two collections, Credit and Client-Entity, each with its purpose. Although, the pre-processing done was not enough, we had to do more pre-processing to extract other relevant features from the data we retrieved.

The two datasets we created were the following:

1. Credit Dataset is the one used to predict whether a client is a good or bad fit for credit investment, where the output is 1 if it is a good investment and 0 if it is a bad investment.

2. Monthly Dataset is the one used to predict if a client will fail its credit payment in the following month, the output is 1 if it will fail a payment next month and 0 if it does not fail.

For our pre-processing development, we used Python, and two popular libraries, Pandas[5] and Numpy[6]. As our primary tool, we use Pandas DataFrames, which are two-dimensional labeled data structures that can be a dictionary of Series, one-dimensional arrays.

In order to build each of these datasets, we used both collections generated by Kafka Streams and four more tables, Entity, Client, Accounts, and Average Balance. The steps taken to build both were the following:

1. We start by retrieving the Credit Collection that contains all the credit proposals, where each entry corresponds to a credit proposal with information like client number, credit type, currency, creation date, number of deadlines, credit amount, and the number of times a credit was in each situation, from normal to closed. From the credit proposals, we can now extract the client numbers that are relevant for us from now on. By focusing only on this clients list, we avoid retrieving other clients information that would be irrelevant.

2. Using the clients' list from credit proposals as a filter, we retrieve the Client-Entity collection, where each entry represents a Client state between two dates. This state is useful since we can find which Entities were linked to a Client on a specific date, and when the end date is empty means that it is the most recent Client state.

   As previously explained, a client is composed of one or more entities, which means that a credit client can be composed of two or more entities. For this reason, when gathering information regarding a credit asked by a customer, whose client is composed by two or more entities, we take into account that each entity can have other accounts not linked to that client which are also relevant.

3. Using the credit date, we get the Entities of each Client relative to that date, by applying some filters and merges. Through these Entities, we can get a list of all the Clients of each Entity from the Clients-Entity collection, with which we can retrieve the accounts.

[5]https://pandas.pydata.org/
[6]http://www.numpy.org/

4. The Accounts collection contains information as accounts definitions, currency, and current balance, which for us is not relevant since one of our goals was to get the average balance of each Account over the six months previous to credit granting as features. These six months were decided due to the reduction in the number of samples if more were used, consequently affecting our models' accuracy.

5. To generate this financial features, we retrieved the Average Balance collection, which contains an entry for every month and every account with the cumulative balance. We calculated the monthly balance based on the cumulative balance, per year, of each month for each account.

6. At this point, we split the pre-processing in two ways to build different datasets. Both need pre-processing, as aggregating the previous months' balance, but for the monthly dataset, it is done for each month since the credit started, whereas for the proposals dataset it is done for the time when the credit was granted.

7. We merge them using the client number, account number, and dates, ending with datasets. There's also some pre-processing done to other features, either to generate new ones from the existing ones or to transform them to best fit our ML models.

8. In the end, we added a label to each of the datasets. This label is generated from the information gathered by Kafka Streams, using the Credit History table, when creating the Credit Collection.

   For the proposal dataset, depending on the number of times it entered in default and on the state to which it changed, we label each entry 1 or 0. Regarding the monthly dataset, we take into account the number of days that it was in each state in that month, and above a threshold, we label it like 1 or 0.

4.3.2 Feature Transformation

In our datasets we have two categories of features:

- **Numerical features** that are continuous real numbers, and can be features like balance or age. This type of feature usually needs some transformation depending on the algorithm used, and due to the range that each feature has, for example, age is normally between 1 and 100, whereas a balance can range between negative values and positive values. We decided to apply the mapping function to the original features and replace them with the new values, reducing the difference between features. The normalize function 4.1 is applied to each set of numerical features, where *std* is the standard deviation of the set and the *mean* is the mean value of the set.

$$Normalize(x_k) = \frac{x_k - mean}{std} \tag{4.1}$$

- **Categorical features** can either be discrete integers or strings, which can be features like the marital status or the employer. Regarding this type of features, specially the

strings, we applied some techniques like trimming and lower-casing, replacing strings that contained a common word with the same word, apply One Hot Encoding (OHE), and filtering the ones less relevant by checking the number of times it was used.

By trimming and lower-casing the strings, we are able to avoid comparisons that would fail otherwise. The replacement of strings is done to avoid the creation of similar columns, due to the technique OHE, that adds a column to the dataset for each different value that a feature has, and each it is attributed 0 or 1. For example, in the case of the marital status, we had four different value - married, single, divorced and widowed - that the OHE technique generated four different columns - MT_married, MT_single, MT_divorced and MT_widowed - and in each column attributed 1 to the entries that previously had that value, and 0 otherwise.

In the end, we removed the generated features that had less than 10 occurrences.

There were some features where we applied an integer encoding instead, for example, the qualification feature, that due to the natural order relationship between each value of the feature allows our ML models to be able to understand it without the need of applying OHE, avoiding increasing the cardinality of our features.

4.3.3 Feature Selection

After doing feature transformation, we ended up with a high number of features, increasing this way, the computational complexity needed to train our models.

So, in order to select the best set of features of our dataset to feed our models, we used the Recursive Feature Elimination and Cross Validation (RFECV) method. RFECV starts using all the features, and at each step, where a step is each time a number of features are eliminated, it evaluates the worst features, which in the case of linear models are the ones with the lowest absolute value and for the tree-based models are the ones with lower feature importance, and removes them. In each step, before selecting the number of features to eliminate it performs cross-validation using the calculated score on the validation data. It is a greedy optimization to find the best set of features for our models.

4.3.4 Sampling

We had a problem with one of our datasets was the imbalanced data, meaning that one of the classes had a higher number of samples than the other one. There are two common techniques that we used to compensate for this imbalance:

- **Over-sampling** consists in increasing the number of samples of the minority class, keeping all the samples without losing any information, but is more prone to overfitting.

  We decided to use the Synthetic Minority Over-sampling Technique (SMOTE), which first considers the K nearest neighbors of each sample of the minority class, and generates synthetic data points in between the distances to the closest neighbors.

- **Under-sampling** aims to reduce the number of samples of the majority class to balance the dataset, which has as a down-side the possibility of discarding useful information.

Despite knowing its well-known draw-back regarding the possibility of losing information, we decided to test under-sampling using Random Under-sampling (RUS) technique, which consists in randomly removing samples of the majority class to balance the class distribution [27].

4.3.5 Models Training

As previously mentioned, we trained and tested three supervised algorithms, Logistic Regression, Decision Tree, and Random Forest, over two different datasets, Credit and Monthly.

For each algorithm we applied cross-validation (CV), specifically Nested Cross Validation, which consists in fitting the data by systematically swapping out samples for testing and training, over two cross validation layers. In our case we did a 10 fold cross-validation at each layer. The inner CV layer is used to do parameter tuning of the model, whereas the outer CV layer is used for the model validation.

4.4 Summary

In this chapter we described the components that make part of our solution. We started by explaining how the bank's database works and how important it is, as it is our primary source of data. Followed by Apache Kafka, that is responsible for retrieving, process data and writing it into our chosen database, MongoDB. Then we described how our Credit Scoring engine deals with the generation of our datasets, and also how it can predict whether a customer will default or not.

Chapter 5

Evaluation

This chapter presents the evaluation of our whole system. Section 5.1 shows how our infrastructure can be useful. Followed by section 5.2 where we evaluate our Credit Scoring engine, using different algorithms and techniques to improve them.

5.1 Infrastructure

The infrastructure described previously was built to provide flexibility and efficiency for more than the development of machine learning solutions. It gives developers access to pre-processed data without having to do extra work each time they need to retrieve data.

Without having Apache Kafka continuously polling the data from the IBM DB2 database, the only way it was possible to retrieve data was through complex SQL queries and exporting it to CSV files. This method has some bottlenecks:

1. In-depth business logic knowledge needed to be able to extract relevant data from the IBM DB2 database;
2. Suffers from technology lock-in, the SQL knowledge required for a developer to be able to retrieve any data;
3. It is time-consuming since it would be running complex queries and also limited by the bandwidth and through a slow VPN connection to the bank's IBM DB2 database every time it is needed to export a CSV file with the data;
4. Lack of possibility to reuse data since queries are custom made for each problem at hand;
5. Pre-processing required each time the data is retrieved, meaning that for different projects it is done the same pre-processing.

Our Apache Kafka based infrastructure mitigates the above-stated problems, thus (highly) improving the workflow of applications that want to use the bank's database.

This infrastructure offers a continuous workflow where data is continuously available and updated on the company premises, without being limited by the bandwidth and VPN constraints. Despite Kafka still being behind a slow VPN connection and bandwidth, it is not noticeable for any developer since it retrieves the data once, making it available without constraints.

The in-depth business logic knowledge is no longer a requirement when retrieving data, considering that previously there was a steep learning curve for anyone needing to access IBM DB2 stored data due to its documentation and outdated database model. Now, this learning curve is surpassed by using Kafka Streams to pre-process and aggregate relevant data.

Kafka Streams supports any pre-process duplication across different projects since each project can access the pre-processed data on a common topic, and also add specific pre-processing on different topics for each project, without affecting the others.

The possibility of extracting the data from Kafka into any database using Kafka Connect solved the technology lock-in problem. For example, if a developer is more experienced using SQL than NoSQL, and depending on the project requirements, it is possible to create a sink connector that writes the data into an SQL database. Previously that was also possible, but the work and costs required were higher than the benefits, whereas now it can be done using Kafka Connect without much work.

For the business point of view, this infrastructure solution was adequate since it allows the creation of innovative projects with the data that is available through Kafka. Besides the current projects, a Credit Scoring engine, and a Recommender system, there are already two other projects taking advantage of Kafka, transactions categorization and customer notifications.

The transactions categorization has the objective of attributing categories like salary, groceries, restaurants, and travel, to be presented to the customer on the bank's website. The categorization is done using Kafka Streams, which creates two topics, one for the transactions of deposit accounts and one for the categories, and crosses both generating a transaction with the respective category. For the future, the aim is to apply artificial intelligence to classify each transaction using features like description, and location.

The customer notifications can be, for example, when the salary of a customer is deposited on its account, or when a transaction is above a certain threshold defined. The notifications are done by crossing the transaction topic with a notification topic, that if a consumer decides it is to notify the client, produce to another specific topic to which the consumer responsible for the notifications engine is subscribed to make it get to the customer.

These projects are only possible due to the modular infrastructure that Kafka provides through the use of consumers and producers, and also the Kafka Streams library that is powerful enough to process vast amounts of data in real-time.

5.2 Credit Scoring Engine

In this section, we present the results of our models' predictions and explain in detail each of the steps taken to improve them, by applying the different techniques described in section 4.3.

Tables 5.1, 5.2, and 5.3 have an equal format, where the left column represents the techniques applied, the middle column shows the scores for the Credit dataset for our selected algorithms, Logistic Regression, Decision Tree, and Random Forest, and the right and last column, shows the prediction scores for the Monthly dataset, for each of the mentioned algorithms.

5.2.1 Evaluation Metrics

We evaluate the prediction of our results using Area under the Receiving Operating Characteristic curve (AUROC) consists in plotting True Positive Rate (TPR), also known as recall (5.1), and False Positive Rate (FPR) (5.2) of every possible threshold. The classifier whose AUROC curve lies above the AUROC curve of a second classifier is superior. Its the combination between the FPR and the TPR into one single metric, followed by the computation of the two above metrics with many different thresholds (for example 0.0, 0.001, 0.002, ..., 1.00). After plotting TPR on y-axis and FPR on the x-axis, the resulting curve is called the ROC curve, and the metric we consider is the AUC of this curve, which we call AUROC. AUROC values lie between 0.5 and 1, where 1 is the best case, meaning that the prediction result is more precise.

For each of the datasets, TPR and FPR have different meanings. For the Monthly dataset, TPR is the percentage of defaults in the next month that should have been classified as non-defaults, and the FPR is the percentage of non-defaults in the next month that should have been classified as defaults. Whereas for the Credit dataset, TPR is the percentage of defaulted loans misclassified as non-defaulted, and the FPR is the percentage of non-defaulted loans misclassified as defaulted.

$$True\,positive\,rate = \frac{TP}{TP + FN}. \qquad (5.1) \qquad False\,positive\,rate = \frac{FP}{FP + TN}. \qquad (5.2)$$

5.2.2 Performance Comparison

In order to investigate the predictions performance, we compare our three selected models with each other: Logistic Regression, Decision Tree and Random Forest. Table 5.1 presents the comparative results of each model throughout the different phases regarding AUROC. We tested our models for two datasets, the Credit Dataset, which has 3219 samples, and the Monthly Dataset, which has 70434 samples. For each dataset were performed three different techniques, Feature Transformation, Feature Selection, and Sampling, to improve our models' predictions. We evaluate our models' predictions using AUROC, which is depicted for each technique, model and dataset in table 5.1.

Regarding the overall table, we can state that among the tested models':

1. For the Credit dataset the Logistic Regression was the one that performed better, taking advantage of feature transformation and selection;

2. For the Monthly dataset, the Random Forest performed better than the others, for a low margin.

In Sections 5.2.3, 5.2.4, and 5.2.5, we explain in detail the results and process of each technique applied.

5.2.3 Impact of Feature Transformation

We started by investigating which impact each step of our feature transformations had in our prediction performance. The three steps taken were the following:

	Credit Dataset			Monthly Dataset		
	LR	*DT*	*RF*	*LR*	*DT*	*RF*
RAW	0.6021	0.7144	0.7276	0.6917	0.7703	0.7714
FS	0.6105	0.7187	0.7288	0.6923	0.7720	0.7750
FT	0.8311	0.7309	0.7542	0.7375	0.7802	0.7821
FT + FS	**0.9002**	0.7491	0.7594	0.7392	0.7840	0.7890
FT + FS + SP	0.8250	0.7285	0.7392	0.7540	0.7862	**0.7914**

Table 5.1: Performance of AUROC on each model for every method applied. RAW represents the raw dataset, where it was not done any modification after the data aggregation. FT stands for feature transformation, which consisted of transformations of categorical and numerical features. FS represents the feature selection method applied to achieve the optimal number of features. FT + FS consists in the combination of feature transformation followed by feature selection.

	Credit Dataset			Monthly Dataset		
	LR	*DT*	*RF*	*LR*	*DT*	*RF*
RAW	0.6021	0.7144	0.7276	0.6917	0.7703	0.7714
NORM	0.6880	0.7176	0.7291	0.7092	0.7727	0.7753
OHE	0.8130	0.7254	0.7483	0.7298	0.7792	0.7789
OHE + NORM	**0.8311**	0.7309	0.7542	0.7375	0.7802	**0.7821**

Table 5.2: Performance of AUROC on each model for each feature transformation method applied. NORM stands for the normalizer applied to the numerical features. OHE stands for one-hot encoding, and represents the steps described in Section 4.3.2.

1. Applying the methods on categorical features referred in Section 4.3.2;

2. Applying the methods on numerical features referred in Section 4.3.2;

3. Combining both methods in each dataset.

The table 5.2 presents the comparison between the transformation techniques applied, either alone or combined. When we applied OHE, it originated, for the Credit dataset, around 1400 features, and for the Monthly dataset, around 1900 features, this was due to the high cardinality of the features. The high cardinality became a problem regarding computational requirements, and for that reason, we did some processing and ending up with around 1200 and 1400 features, for Credit dataset and Monthly dataset, respectively.

From the table 5.2, we can conclude that for both datasets the best result was from combining both feature transformation methods.

For the LR models, the difference between the RAW dataset and the combination of both methods is more accentuated compared to the other models. Since neither of the algorithms supports categorical features, we can see an increase after applying OHE as expected.

The normalization did not affect much the tree-based models' predictions, DT and RF, which confirmed that it was due to the models not being distance based and for being able to handle varying ranges of features.

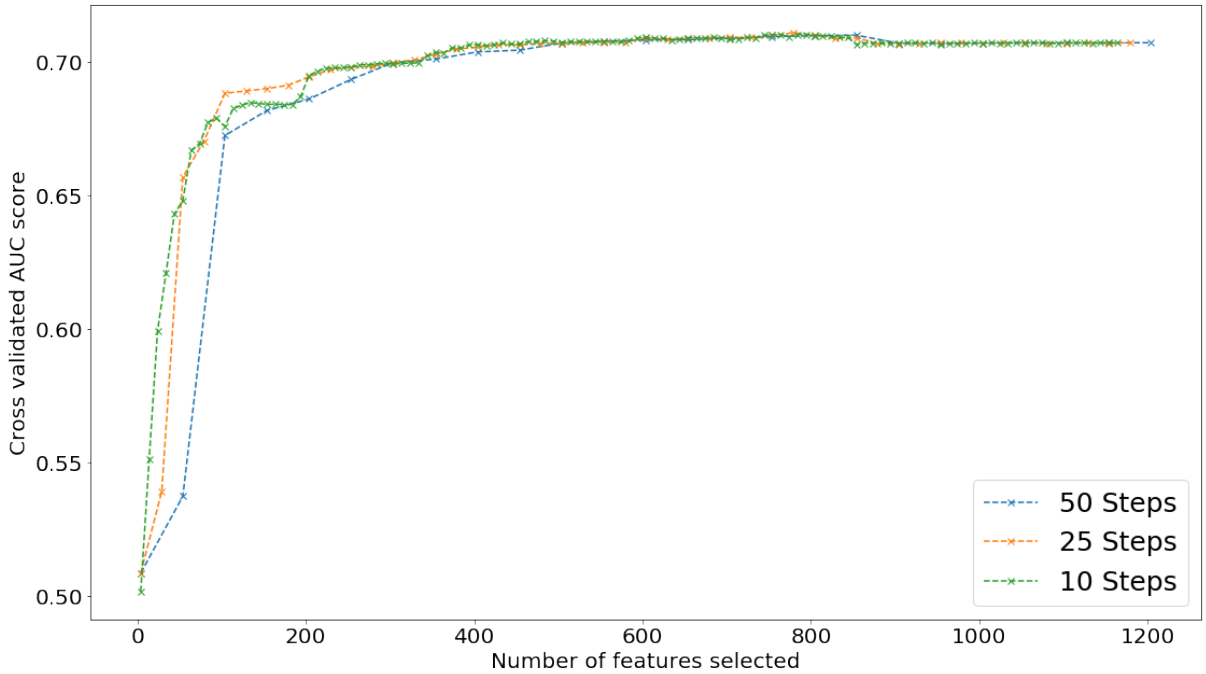

Figure 5.1: AUROC scores for different number of features using RFECV, using cross-validation at each step, for LR using Credit dataset.

5.2.4 Impact of Feature Selection

The feature selection technique used, RFECV, was tested using different steps, it means that for every model we ran RFECV, using five-fold cross-validation for three different steps, 50, 25 and 10. The step is the number of features it discards on every iteration over the total number of features.

In figure 5.1 are presented the results of applying RFECV, to find the optimal number of features for a LR, using the Credit dataset. There was a total of 1154 features that for each of the steps the LR was tested, an optimal number of features was selected. For the 50 steps, the optimal number of features was 804, and for 25 and 10 steps, was 754 and 784 features, respectively, where was obtained the highest AUROC computed from the prediction scores.

Figure 5.2 presents the results of the AUC scores for each of the optimal number of features, 804, 754, and 784, after applying a GridSearch to find the best parameters for the LR, with five-fold cross-validation. The number of features was reduced from 1154 to 804, which was the number of features that we obtained the best AUC score.

When combining feature selection with feature transformation, we can check that there is always an increase in the prediction performance of any model for both datasets. This performance increase might be due to the reduction of the number of features, that comes from applying feature transformation to features with high cardinality. This reduction in the number of features, besides improving our models' predictions, reduces the computation required to train our models.

Table 5.1 shows an increase of the model prediction in each of them after feature selection is applied to both of the raw datasets.

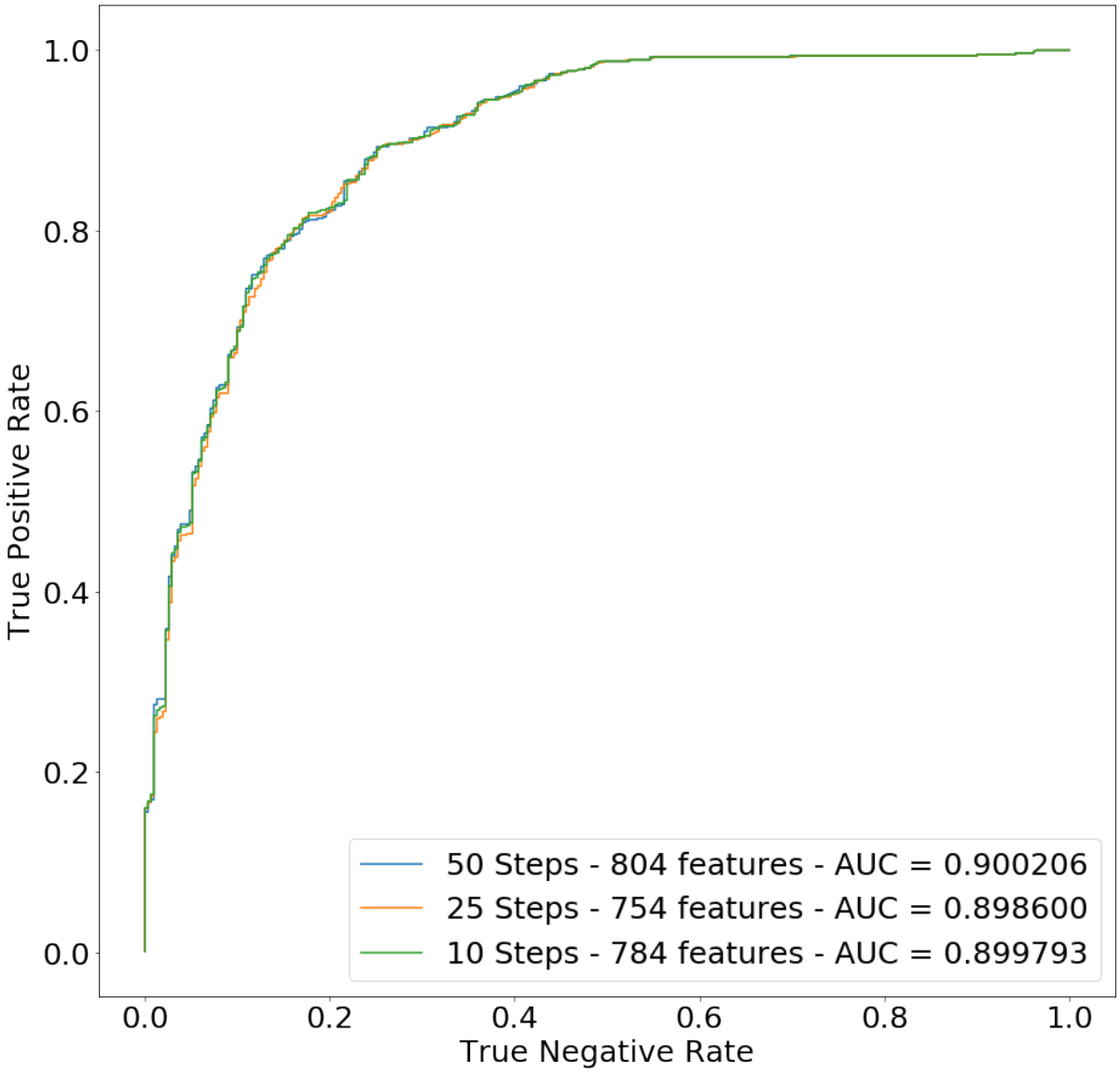

Figure 5.2: ROC curves for a distinct number of features selected using RFECV.

	Credit Dataset			Monthly Dataset		
	LR	*DT*	*RF*	*LR*	*DT*	*RF*
$FT + FS$	**0.9002**	0.7491	0.7594	0.7392	0.7840	0.7890
$FT + FS + RUS$	0.8250	0.7285	0.7392	0.7407	0.7793	0.7820
$FT + FS + SMOTE$	0.8201	0.7213	0.7359	0.7540	0.7862	**0.7914**

Table 5.3: Performance of AUROC on each model for each sampling technique applied. RUS stands for Random Over-sampling. SMOTE stands for Synthetic Minority Over-sampling Technique.

5.2.5 Impact of Sampling

The table 5.3 presents a comparison between the two sampling techniques mentioned in Section 4.3.4, Random Under-Sampling and Synthetic Minority Over-sampling Technique, after performing some feature transformations. Applying these techniques to our datasets, we were able to achieve a higher prediction performance in the Monthly dataset. The Monthly dataset has a ratio of 11.2:1, which reveals how imbalanced it is, in comparison to the Credit dataset that has a ratio of 1.9:1.

We can state from the table 5.3 that for the imbalanced dataset the over-sampling technique applied, SMOTE, performed better than the under-sampling technique, RUS. This might be due to the loss of useful information that under-sampling discards of the majority class [10].

Regarding the Credit dataset, RUS performed better than SMOTE, which can indicate that the majority class might be introducing some noise. Even though, both techniques did not perform better than the feature transformation and selection combination.

5.3 Summary

In this chapter, we discussed how important was to build our infrastructure based on Kafka, in terms of, flexibility and efficiency, allowing a continuous workflow and the ability to build more applications, even if not related to machine learning, on top of it. Now, it is not needed an in-depth business logic, due to the use of Kafka Streams and its capabilities to aggregate data continuously and in near real-time, by modeling the data in a way that the business logic is easily understood.

Our Credit Scoring engine was able to predict accurately whether a customer represents a good or bad credit for the bank, through the application of different techniques, like one-hot encoding, SMOTE, and RFECV, to improve our machine learning models, as Logistic Regression, Decision Tree and Random Forest.

Chapter 6

Conclusion

Overdue loans are a huge worldwide problem. It is challenging for a bank to decipher a client's profile and decide on whether or not to give out a loan. It is not only a matter of profit for the bank, it is also a matter of social responsibility. The decision cannot be made lightly as it can have a significant impact on a person's life. One of the most significant problems with the decision-making process is human bias. Whether we like to admit it or not, we have a substantial emotional influence when making a decision. It is almost impossible to make an unbiased decision. That is the issue we want to tackle. Create a system, let's call it artificial intelligence, capable of making faster, better and as unbiased as possible decisions.

In order for us to build a reliable and accurate profile of the client, we had to gather much information. Fortunately, the bank on which this project was developed offers a dependable multichannel service to its clients. That allows us to access clients transactions records like bill payments, credit card transactions, and salary deposits. As well as personal information like marital status, age, and certifications.

The first step was to analyze the business rules and the data we have available. Next, we gathered and processed the useful data from sources we found. After that, we trained a set of algorithms like Logistic Regression, Decision Tree and Random Forest, built our models and tested their predictability.

With this project, we created a foundation, constituted by a set of trained models, for the bank not only to further improve the decision making on loan contracts but also to provide a robust infrastructure for other company projects like products recommendation, transactions categorizations, and others to come.

6.1 Achievements

With this project, we were able to set up an infrastructure that supports multiple projects, processes information in near real-time making it available to any project, and also to our Credit Scoring engine. The Credit Scoring engine, using the data provided from our Kafka infrastructure, trains models that that can predict whether or not a customer will represent a good or bad credit for the bank, and can be used as a foundation to build a tool that the bank can use to support any decision related to credit granting.

6.2 Future Work

We propose other features and improvements for future work as:

1. Adding a new source of information like the national bank's data regarding credits across all the banks in the country, which would provide more information and the possibility of improving our models;

2. Creating client consumer profiles through clustering, using the transactions available from bank's database from Kafka Streams, applying ML to categorize transactions efficiently and correctly, to give our Credit Scoring Engine more information about the customer, thus improving our ML models' predictions;

3. Integrating our ML models with the bank, so they can easily use it to improve their process of credit granting or to evaluate better their credit candidates;

4. Making our ML models comply with regulatory specifications.

Bibliography

[1] Benchmarking Apache Kafka: 2 million writes per second (on three cheap machines), available from `https://engineering.linkedin.com/kafka/benchmarkingapache-kafka-2-million-writes-second-three-cheap-machines`. Last time accessed: October 10, 2018.

[2] H. A. Abdou, M. D. Tsafack, C. G. Ntim, and R. D. Baker. Predicting creditworthiness in retail banking with limited scoring data. *Knowledge-Based Systems*, 103:89–103, 2016.

[3] K. Angola. Resilience and Evolution - Angola Banking Survey. Technical Report 1, KPMG Angola, 2016.

[4] B. Baesens, T. Van Gestel, S. Viaene, M. Stepanova, J. Suykens, and J. Vanthienen. Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of the Operational Research Society*, 54(6):627–635, jun 2003.

[5] A. Blöchlinger and M. Leippold. Economic benefit of powerful credit scoring. *Journal of Banking and Finance*, 30(3):851–873, 2006.

[6] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[7] D. G. Chandra. BASE analysis of NoSQL database. *Future Generation Computer Systems*, 52:13–21, 2015.

[8] P. Dobbelaere and K. S. Esmaili. Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, DEBS '17, pages 227–238, New York, NY, USA, 2017. ACM.

[9] A. Fuster, P. Goldsmith-Pinkham, T. Ramadorai, and A. Walther. Predictably Unequal? The Effects of Machine Learning on Credit Markets. 2018.

[10] V. García, A. I. Marqués, and J. S. Sánchez. Improving risk predictions by preprocessing imbalanced credit data. In *International Conference on Neural Information Processing*, pages 68–75. Springer, 2012.

[11] J. Gareth, W. Daniela, H. Trevor, and T. Robert. *An Introduction to Statistical Learning with Applications in R*. Springer, 6th edition, 2003.

[12] D. J. Hand. Classifier Technology and the Illusion of Progress. *Statistical Science*, 21(1):1–14, 2006.

[13] D. J. Hand and W. E. Henley. Statistical Classification Methods in Consumer Credit Scoring: a Review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 1997.

[14] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer series in statistics New York, NY, USA, 2001.

[15] T. K. Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, Aug 1995.

[16] F. Hoffmann, B. Baesens, C. Mues, T. Van Gestel, and J. Vanthienen. Inferring descriptive and approximate fuzzy rules for credit scoring using evolutionary algorithms. *European Journal of Operational Research*, 177(1):540–555, 2007.

[17] D. W. Hosmer, S. Lemeshow, J. W. Sons, and W. I. O. Service). *Applied logistic regression*. New York : Wiley, 2nd edition, 2000.

[18] S. Hue, C. Hurlin, and S. Tokpavi. Machine Learning for Credit Scoring: Improving Logistic Regression with Non Linear Decision Tree Effects. (July):1–29, 2017.

[19] V. John and X. Liu. A survey of distributed message broker queues. *CoRR*, abs/1704.00411, 2017.

[20] A. E. Khandani, A. J. Kim, and A. W. Lo. Consumer Credit Risk Models via Machine-Learning Algorithms. *Journal of Banking & Finance*, 34(11):1–55, 2010.

[21] J. Kreps, N. Narkhede, and J. Rao. Kafka: a Distributed Messaging System for Log Processing. *ACM SIGMOD Workshop on Networking Meets Databases*, page 6, 2011.

[22] T. S. Lee, C. C. Chiu, Y. C. Chou, and C. J. Lu. Mining the customer credit using classification and regression tree and multivariate adaptive regression splines. *Computational Statistics and Data Analysis*, 50(4):1113–1130, 2006.

[23] S. Lessmann, B. Baesens, H. V. Seow, and L. C. Thomas. Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research*, 247(1):124–136, 2015.

[24] B. M. Michelson. Event-driven architecture overview. *Patricia Seybold Group*, 2(12):10–1571, 2006.

[25] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[26] MongoDB. Deep Learning and the Artificial Intelligence Revolution (white paper). 2017.

[27] A. More. Survey of resampling techniques for improving classification performance in unbalanced datasets. *arXiv preprint arXiv:1608.06048*, 2016.

[28] L. Sara, K. Taghi, R. Aaron, and H. Tawfiq. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, 2(1):1–36, 2015.

[29] A. Steenackers and M. Goovaerts. A credit scoring model for personal loans. *Insurance: Mathematics and Economics*, 8(1):31–34, 1989.

[30] L. Thomas, J. Crook, and D. Edelman. *Credit scoring and its applications*. SIAM Monographs on Mathematical Modeling and Computation, 2017.

[31] A. Vigil. *Building explainable random forest models with applications in protein functional analysis*. PhD thesis, San Francisco State University, 2016.