Infrastructure and Machine Learning for Credit Scoring

Francisco Falcão Amaral Caixeiro
Instituto Superior Técnico, Universidade de Lisboa
Advisors: Prof. Dr. Manuel Fernando Cabido Peres Lopes and
Dr. Ricardo Portela

***Abstract*—Credit scoring is the analytical technique used by banks to evaluate credit risk. Lowering 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 everyday. 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 process 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

I. 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% [1].

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 [2]. 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 [3].

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.

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.

A. 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] https://www.inm.pt

II. 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 II-C 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.

A. 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 [4].

B. 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 [5].

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.

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 [6], [7].
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 [8].

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.

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 [9], [10].

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.

3) Random Forest: **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 [11]. 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 [9].

C. Infrastructure

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

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.

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 [12].

One well-known NoSQL database is MongoDB [2], 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 [13].

2) Event-Driven: In this type of architecture, an event is published when something notable happens, such as when a new 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 [14].

Figure 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 [15].

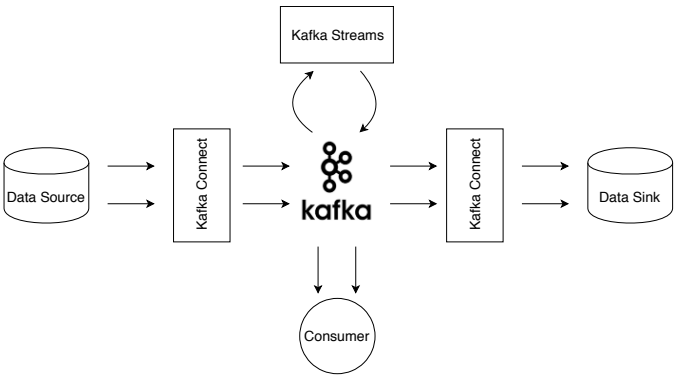

Fig. 1: Kafka architecture and some of its components

D. 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. Then explained the algorithms that we took into account when developing our solution. This chapter ends with some concepts of databases and event-driven architecture that were important to support our ML implementation.

III. Related Work

With this chapter, we intend to address and present some related solutions to the project that has been developed. In section III-A 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 III-B 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.

A. Infrastructure

Selecting the right tools to support a 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 are able to 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.

1) Event Processing: Apache Kafka implements his 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 [16].

Kreps et al. (2011) [15] 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 4 times slower, probably due to Kafka's efficient storage format.

B. 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 [17], [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) [19] 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

[2]https://www.mongodb.com/

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) [6] 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 receiver operating characteristic curve (AUROC) over 8 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) [20] 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 [7], [21].

A well known homogeneous ensemble method is Random Forest (RF). It has been first presented by Breiman et al. (2001) [11]. Lessman et al. (2015) [7] in his study compares 16 individual classifiers, 8 homogeneous ensembles, and 17 heterogeneous ensembles, tested against 7 different datasets. RF is among the best classifiers after extensive testing taking into account everything that has been done in the previous years regarding credit scoring. The real-world datasets that have been used had several independent variables essential to build 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 [6] 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) [7] 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.

C. 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.

IV. Solution

This chapter presents a credit scoring engine on top of an infrastructure based on Kafka. Section IV-A 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 IV-B, 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 IV-C focus on the Credit Scoring, where we explain the solution that led to our credit scoring engine.

A. 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.

Apache Kafka is responsible for consuming all data that the bank collects into the IBM DB2 RDBMS. It is continuously polling and processing new or updated data. This processing is done using a component of Kafka, Kafka Streams, and 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.

B. 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.

1) IBM DB2 source database: As the main component of our infrastructure, we have the bank's IBM DB2 database, which is where all the data that we will use. This database is where information of clients like transactions, changes made to a clients' account and all the credit information is stored.

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.

Client table contains data that differ for each client as if it represents a company or an individual.

Client-Entity table is where resides all the current connections between clients and entities.

Account tables store information regarding each type of account. The more relevant type of account will be the credit account.

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 will use to build our pipeline, and a REST Proxy allowing us to access Kafka and control it via a REST API.

Apache Kafka

Apache Kafka is similar to a message queue system and can process events as soon as they arrive, 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.

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.

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 continually 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 will facilitate all of this process by polling continually the database. The way JDBC source connector can copy data incrementally from a database depends on which mode we use.

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;
- **Compatibility** using schemas avoids all the problems that could arise from modeling and data format changes.

Schema Registry is responsible for storing Avro schemas, sent by producers, and keeps a versioned history in Kafka topics. 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.

3) MongoDB sink database: 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.

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 2, throughout the process of receiving data in real-time.

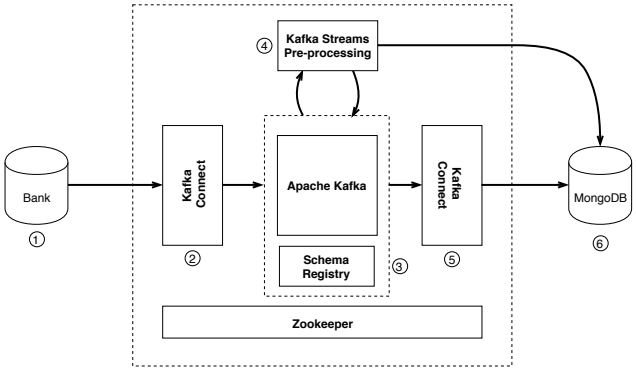

Fig. 2: Kafka architecture.

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 continually 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.

C. 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, we want to 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.

1) Datasets creation: 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 some more pre-processing to extract other relevant features from the data we retrieved.

The two datasets we created were the following:

1) Proposals 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.

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. We decided to apply the mapping function to the original features and replace them with the new values, reducing this the difference between features. A normalize function is applied to each set of numerical features.
- **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 One Hot Encoding (OHE), and filtering the ones less relevant by checking the number of times it was used.
  In the end, we removed the generated features that had less than 10 occurrences.

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) 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 a down-side the possibility of discarding useful information.

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.

V. 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.

VI. Evaluation

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

A. Infrastructure

The infrastructure described previously was built to provide flexibility and efficiency for more than the development of machine learning solutions.

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) 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;
3) 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.

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.

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.

B. 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 IV-C.

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), known also as recall, and False Positive Rate (FPR) of every possible threshold. The classifier whose AUROC curve lies above the AUROC curve of a second classifier is superior.

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 I 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.

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.

3) Impact of Feature Transformation: The table II 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. This became a problem in terms of 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.

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 support categorical features, we can see an increase after applying OHE as expected.

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 a 5 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.

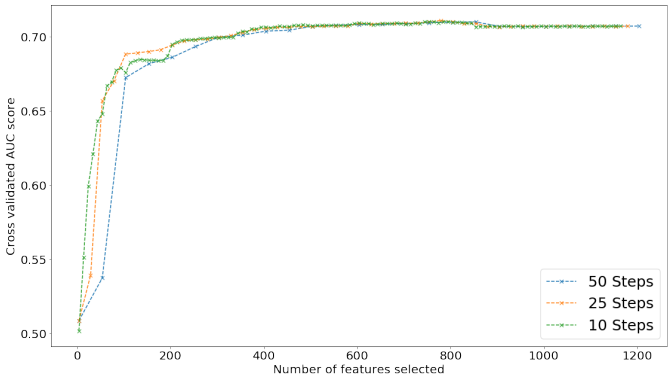

Fig. 3: AUROC scores for different number of features using RFECV, using cross-validation at each step, for LR using Credit dataset.

In figure 3 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 4 presents the results of the AUC scores for each of the optimal number of features, 804, 754, and 784. The number of features was reduced from 1154 to 804, which was the number of features that we obtained the best AUC score.

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

5) Sampling: The table III presents a comparison between the two sampling techniques mentioned in Section IV-C4, 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 III 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 [22].

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.

	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 I: Performance of AUROC on each model for every method applied. RAW represents the raw dataset. FT stands for feature transformation. FS represents the feature selection method . 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 II: 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 IV-C3.

	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 III: 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.

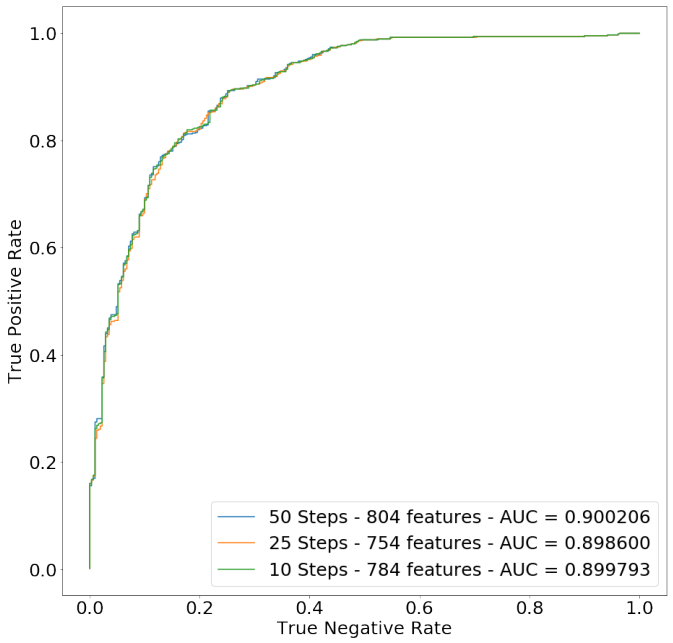

Fig. 4: ROC curves for a distinct number of features selected using RFECV.

C. 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.

Our Credit Scoring engine was able to predict accurately whether a customer represents a good or bad credit for the bank.

VII. Conclusion

Overdue loans are a huge worldwide problem. It is very difficult for a bank to decipher a client's profile and decide on whether or not to give out a loan. It's not only a matter of profit for the bank, it's also a matter of social responsibility. One of the biggest problems with the decision making process is the human bias. Whether we like to admit it or not, we have a huge emotional influence when making a decision. It's almost impossible to make an unbiased decision.

In order for us to build a solid and accurate profile on the client, we had to gather a lot of information. We had access to 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 solid infrastructure for other company projects like products recommendation, transactions categorizations and others to come.

A. Achievements

With this project, we were able to setup an infrastructure that supports multiple projects, processes information in near real-time making it available to any project. The Credit Scoring engine trains models that that are able to 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.

B. 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, 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.

REFERENCES

[1] K. Angola, "Resilience and Evolution - Angola Banking Survey," KPMG Angola, Tech. Rep. 1, 2016. [Online]. Available: https://assets.kpmg.com/content/dam/kpmg/pt/pdf/pt-2016-angola-banking-survey-en.pdf

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

[3] 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.

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

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

[6] 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*, vol. 54, no. 6, pp. 627–635, jun 2003. [Online]. Available: http://link.springer.com/10.1057/palgrave.jors.2601545

[7] 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*, vol. 247, no. 1, pp. 124–136, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.ejor.2015.05.030

[8] 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*, vol. 103, pp. 89–103, 2016.

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

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

[11] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[12] 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*, vol. 2, no. 1, pp. 1–36, 2015.

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

[14] B. M. Michelson, "Event-driven architecture overview," *Patricia Seybold Group*, vol. 2, no. 12, pp. 10–1571, 2006.

[15] J. Kreps, N. Narkhede, and J. Rao, "Kafka: a Distributed Messaging System for Log Processing," *ACM SIGMOD Workshop on Networking Meets Databases*, p. 6, 2011. [Online]. Available: http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

[16] 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.

[17] A. Blöchlinger and M. Leippold, "Economic benefit of powerful credit scoring," *Journal of Banking and Finance*, vol. 30, no. 3, pp. 851–873, 2006.

[18] S. Hue, C. Hurlin, and S. Tokpavi, "Machine Learning for Credit Scoring: Improving Logistic Regression with Non Linear Decision Tree Effects," no. July, pp. 1–29, 2017.

[19] A. Steenackers and M. Goovaerts, "A credit scoring model for personal loans," *Insurance: Mathematics and Economics*, vol. 8, no. 1, pp. 31–34, 1989.

[20] 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*, vol. 50, no. 4, pp. 1113–1130, 2006.

[21] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, Aug 1995, pp. 278–282 vol.1.

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