INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistema de deteção de fraude em pagamentos eletrónicos. Estudo e Implementação usando software de distribuição livre

PAULO JORGE DOS SANTOS COSTA
Outubro de 2019

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Sistema de deteção de fraude em pagamentos eletrónicos

Estudo e Implementação usando *software* de distribuição livre

Paulo Jorge dos Santos Costa

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas de Informação e Conhecimento

Orientadora: Doutora Fátima Rodrigues

Porto, outubro de 2019

Resumo

Com a evolução dos negócios para o digital que se traduz num maior número de transações por segundo. Estes métodos de pagamento são naturalmente um alvo apetecível para os burlões. É então premente o estudo de sistemas de deteção de fraude que antecipem as tentativas destes burlões em efetuaram dano nas empresas e nos seus clientes. Este trabalho foca o seu estudo na escolha do melhor modelo de classificação de transações fraudulentas com cartões de crédito, resultante da aplicação de técnicas de pré-processamento de dados (*sampling*) e algoritmos de classificação de aprendizagem supervisionada. Foi estudada a influencia que as técnicas de undersampling e oversampling podem ter sobre os classificadores estudados. O resultado do estudo permite afirmar que não se observa uma predominância transversal para todos os classificadores de uma técnica sobre a outra. Verifica-se ainda que em alguns modelos a diferença de prestação não apresenta grande diferença entre a versão com *undersampling* e a versão com *oversampling*, sendo esta última muito mais exigente no que diz respeito à capacidade de processamento. Por fim neste trabalho são descritos os passos para a implementação da proposta para um sistema de deteção em tempo real, usando os serviços do *Google Cloud Platform*.

Palavras-chave: Undersamplig, oversampling, aprendizagem supervisionada, deteção de fraude

Abstract

With the natural digital business evolution, more online transaction are made per second on these platforms. This payment methods are more and more a target for fraudsters. The study of this kind of attacks are a must and the speed of a fraud detections is crucial to stop any attempt to damage the trust link between clients and enterprises. This study is focused on finding the best classification model for credit card fraud transactions on an online service, using pre-processing techniques, under and oversampling, applied to supervised learning classification algorithms. The results of this study, show that is no overall dominance of a sampling technique over another. In some classification algorithms the performance score between sampling techniques are minimal, and oversampling presenting a higher computing processing load. Finally, are presented the steps of a possible implementation of a real time credit card fraud detection system as a Google Cloud Platform solution.

Keywords: Undersamplig, oversampling, supervised learning, fraud detection.

Índice

Lista de Figuras

Lista de Tabelas

1 Introdução

Neste capítulo é apresentado o contexto e problema do projeto, explicando ainda quais os seus objetivos, análise de valor, resultados esperados e abordagem a ser feita, terminando com a descrição da organização do documento.

1.1 Contexto

Com a crescente transferência para o online das relações comerciais entre instituições e clientes, novos desafios se colocam. O pagamento com cartões de crédito, que sempre foi um caso de estudo no que diz respeito à fraude, coloca novos desafios neste ecossistema, nomeadamente com a adoção de novas técnicas de fraude por parte dos burlões e busca continua de encobrir os seus atos. Dado o grande volume de transações por segundo que os sistemas atualmente possibilitam, é necessário estudar e desenvolver sistemas inteligentes de apoio à decisão, que possibilitem a deteção de atos potencialmente fraudulentos em tempo útil de serem anulados.

Detetar fraudes em tempo real exige o design e a implementação de técnicas de aprendizagem escaláveis capazes de processar e analisar grandes quantidades de dados em *streaming* [1]. Avanços recentes em análise de dados e a disponibilidade de soluções de código aberto para armazenamento e processamento de *Big Data* abrem novas perspetivas na deteção de fraudes [2]. Nesta dissertação serão integradas ferramentas *Big Data* com uma abordagem de aprendizagem baseada em modelos de deteção de fraude obtidos a partir de algoritmos de data mining.

1.2 Problema

Até 2010 só sete estudos atestavam a sua implementação prática conforme levantamento efetuado nesse ano [3]. Mais recentemente foram identificados os trabalhos de Carcillo et al, 2018 [2] e Nuno Carneiro et al 2017 [4].

Na deteção de fraude os tempos de reação (compreendido entre os sintomas e a ocorrência de fraude) e de latência (compreendido entre a ocorrência e a deteção de fraude) são cruciais. No entanto é preferível evitar que a fraude aconteça e para tal garantindo tempos de reação menores, através de uma análise fidedigna e em tempo real das transações, minimizando as fraudes não detetadas.

Para o estudo proposto há a necessidade de implementar um sistema de treino de modelos de análise e processamento recorrendo ao *software* mais atual disponível, sendo privilegiado a modalidade de licenciamento de uso livre.

Devido ao característico desbalanceamento dos registos é necessário proceder a técnicas de pré-processamento. As abordagens estudadas utilizam o tradicional *undersampling* dos registos legítimos. É pertinente, como sugerido pelos diferentes estudos, verificar se a aplicação de outras técnicas, como o *oversampling*, melhoram a performance do sistema a implementar.

1.3 Objetivos

Para o treino dos modelos de deteção de fraude é necessário a análise de dados históricos de transações efetuadas num período de tempo e identificadas como sendo ou não fraudulentas. Existem algumas comunidades on-line dedicadas à área de *Data Science* que disponibilizam conjuntos de dados reais, que contêm milhares de transações, que servem para a elaboração de estudos na deteção de fraude aplicando técnicas de *data mining*.

O objetivo principal deste projeto é o treino de modelos de deteção de fraude, tendo por base um histórico de transações, para posterior análise de transações em tempo real. Tirando partido do sistema implementado pretende-se ainda identificar qual a técnica de pré-processamento mais eficiente para o tratamento de dados desbalanceados típicos deste tipo de fraude. Pretende-se ainda aferir qual o modelo, de entre os mais utilizados, que apresenta melhores resultados no que diz respeito à identificação das transações fraudulentas, minimizando os resultados de previsões erradas.

1.4 Análise de valor

Em 2017 as perdas devido a fraude com cartões de crédito a nível mundial ascenderam a cerca de 21.5 mil milhões de euros (mais 6.4% do que 2016), sendo que as fraudes onde o cartão físico não está presente, online e por telefone, representam cerca de 15% deste valor [5]. O aumento da fraude causa não só grandes prejuízos às empresas e indivíduos, como também diminui a confiança dos consumidores nas empresas que usam este recurso para efetuarem os seus negócios [6].

1.5 Resultados esperados

Como resultado esperado pretende-se obter um sistema capaz de identificar por métricas predefinidas, qual o modelo mais eficaz para a deteção de transações fraudulentas, através da análise e avaliação de várias técnicas de pré-processamento de conjuntos de dados desbalanceados, bem como da aplicação diferentes algoritmos matemáticos e estatísticos.

1.6 Abordagem preconizada

Numa primeira fase procedeu-se ao estudo de vários trabalhos publicados sobre a temática da classificação de transações online usando cartões de crédito, identificando os vários modelos e técnicas usados e resultados obtidos. Seguindo as propostas para trabalho futuro apresentadas nos estudos analisados, foi idealizado um *pipeline* para proceder ao teste dos novos modelos propostos neste trabalho. Foi necessário selecionar o conjunto de dados que servirá como base de treino e teste dos modelos. Para tal recorreu-se ao repositório online *kaggle* [7], sendo escolhida uma base de dados tipificada e identificada em estudos idênticos. Numa segunda fase procedeu-se ao desenvolvimento do código que implementa as técnicas de análise, pré-processamento e *sampling* dos dados, bem como o treino, teste e avaliação dos modelos. São testadas duas técnicas de pré-processamento (*undersampling* e *oversampling*), em quatro abordagens (regressão logística, máquinas de suporte vetorial, florestas aleatórias e *XGBoost*), e analisadas as métricas, *matriz de confusão e Area Under the Curve - Receiver Operating Characteristic (AUC-ROC*) resultantes para determinar qual o modelo com melhor desempenho. Por fim foi estudado a forma de implementação do melhor modelo num sistema de deteção em tempo real de fraude, utilizando a plataforma *Google Cloud Platform* [8].

1.7 Organização do documento

Este documento encontra-se organizado em 5 capítulos. O primeiro descreve de uma forma geral a contextualização do trabalho, o problema e os objetivos a atingir, a abordagem preconizada. O segundo capítulo apresenta a análise de valor do projeto e caracteriza a área de negócio. No terceiro capítulo é apresentado o estado de arte não só em abordagens existentes, bem como em tecnologias relevantes, são apresentados também neste capítulo os trabalhos de terceiros considerados para este estudo. O quarto capítulo descreve em pormenor as abordagens assumidas e postas em prática para atingir os objetivos propostos. São descritas com algum nível de detalhe as duas fases, a exploratória e a de implementações. É neste capítulo que são apresentados os resultados do estudo e feita a sua análise. No último capítulo, são apresentadas as conclusões do trabalho, bem como a apresentação de propostas para trabalho futuro, que surgiram no decorrer deste estudo.

2 Análise de valor

2.1 Área de negócio

O sistema de deteção de transações fraudulentas apresentado neste trabalho caracteriza-se por um módulo de análise que pode ser integrado nos sistemas de pagamentos online já existentes.

2.2 Análise de valor

Seguindo o modelo NCD (*New Concept Development Model*) de análise de valor, que julgamos ser o mais adequado às características projeto, visto que se distingue por apresentar um modelo relacional em oposição aos modelos lineares. O modelo NCD está focado na FEI (*Front-end Inovation*) e abrange as atividades que precedem a formalização, estruturação do produto e o processo de desenvolvimento. O modelo está representado na figura 1 e é composto por três elementos: a área interior, que define os cinco elementos chave compreendidos na FEI; o motor, que representa o apoio à gestão por parte de executivos e seniores que dinamizam os cinco elementos da FEI que são alimentados pela liderança e cultura organizacional; e os fatores influenciadores que consistem em capacidades organizacionais, estratégia de negócio e mundo exterior [9].

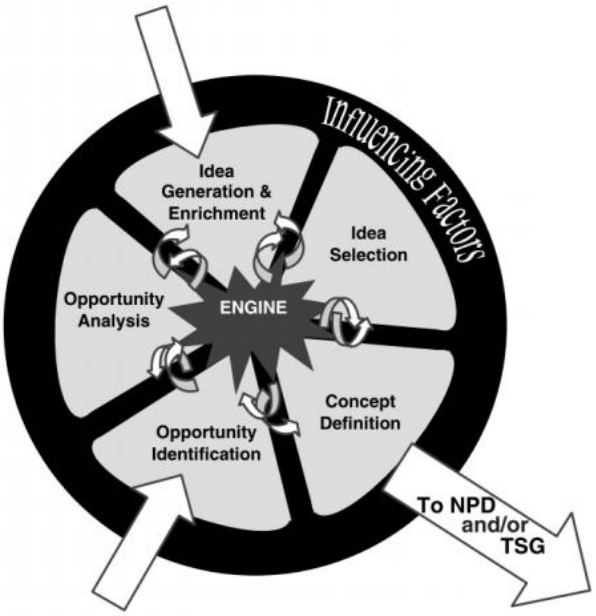


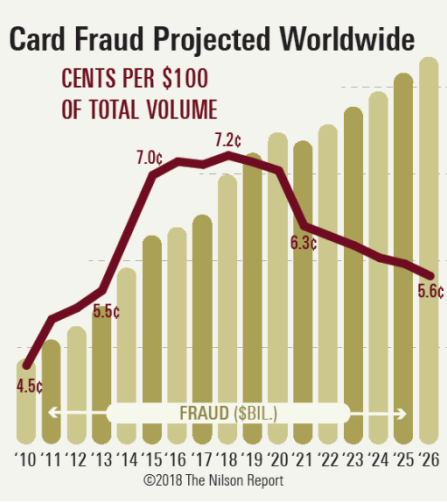
Figura 1: NCD (*New Concept Development*); fonte: Koen et al. (2002)

A forma circular do modelo NCD sugere que é esperado que as ideias e conceitos percorram de forma iterativa os cinco elementos da FEI. As setas que entram no modelo, representam pontos

de partida e indicam que os projetos começam na altura da identificação da oportunidade ou criação da ideia e enriquecimento. A seta que sai do modelo representa como os conceitos saem do modelo e entram em desenvolvimento de novo produto (NPD - *New Product Development*) [10].

- **Identificação de oportunidade**
  Em 2017 as perdas devido a fraude com cartões de crédito a nível mundial ascenderam a cerca de 21.5 mil milhões de euros (mais 6.4% do que 2016), sendo que as fraudes onde o cartão físico não está presente, online e por telefone, representam cerca de 15% deste valor [5]. Os custos associados a uma transação fraudulenta são imputados aos vendedores que no melhor cenário só lhes é cobrada por parte do banco a taxa de manutenção quando a transação é anulada. Assim o principal interessado em evitar que uma transação transite em fraudulenta é o vendedor ou prestador de serviços que para além da questão económica, vê uma crescente perda de confiança por parte clientes no seu relacionamento.

- **Análise de oportunidade**
  O esforço das principais financeiras detentoras de marcas de cartões de crédito no combate à fraude, tem resultado numa redução do valor que esta representa no valor total de lucro, no entanto com o aumento do volume de negócios que já se verifica e o que é previsto nos próximos anos, a valor atribuído à fraude aumenta cerca de 2.5 mil milhões de euros ao ano (fig. 2).



Year	Total Vol. ($tril.)	Fraud ($bil.)	¢ per $100 Vol.
2010	17.028	7.60	4.46
2011	19.394	9.84	5.07
2012	21.608	11.27	5.22
2013	25.155	13.70	5.45
2014	28.844	18.11	6.21
2015	31.310	21.84	6.97
2016	31.878	22.80	7.15
2017	34.119	24.26	7.11
2018	39.869	28.82	7.23
2019	43.901	31.34	7.14
2020	47.904	33.69	7.03
2021	52.014	32.75	6.30
2022	56.172	34.66	6.17
2023	60.611	36.64	6.04
2024	65.259	38.37	5.88
2025	69.977	40.56	5.80
2026	75.052	42.32	5.64

Figura 2: Projeção mundial da fraude com c.c.: fonte *The Nilson Report* (2018)

É necessário não só identificar quais os modelos mais eficazes na deteção de perfis de transações potencialmente fraudulentas, como também os mais económicos e rápidos, sendo a componente da análise de transações em tempo real crucial na implementação de um sistema de deteção de fraude.

- **Geração de ideias**

Foram analisados trabalhos publicados que abordam a aplicação de tecnologias de data mining, testando a eficiência de vários modelos matemáticos e estatísticos e diferentes técnicas de pré-processamento dos dados. Foram também analisados trabalhos que descrevem as arquiteturas dos sistemas a desenvolver. Foi estudado a oferta de *software* de distribuição livre, visto ser um requisito autoproposto, visando a redução de custo do sistema para uma maior dispersão da solução aumentando o leque de potenciais clientes. Foram analisadas diferentes linguagens de programação suportadas pelos *softwares* estudados.

- **Seleção de ideias**

  Partindo dos trabalhos publicados que apresentam o sistema em toda a sua plenitude Carcillo et al, 2018 [2] e Nuno Carneiro et al 2017 [4] foi definida a arquitetura a implementar. Foi identificado o *Spark* da *Apache* como o *software* indicado que será o núcleo do sistema, não só pela capacidade de processamento distribuído, bibliotecas de *machine learning*, bem como por conter todas as componentes necessárias ao sistema e ser de distribuição livre. Por indicação do trabalho publicado por Nuno Carneiro et al 2017 [4], foram selecionadas duas formas de pré-processamento de dados desbalanceados a serem testados. A escolha da linguagem de programação foi aferida pelo método AHP.

- **Definição de conceito**

  O sistema a desenvolver deve ser um sistema capaz de auxiliar os investigadores na parametrização das diferentes variáveis a serem estudadas, apresentando as métricas resultantes para a devida avaliação dos modelos estudados. É efetuada a descrição pormenorizada do desenvolvimento do sistema para permitir a implementação e integração do sistema em ecossistemas de pagamentos online.

  Analythic Hierarchy Process (AHP)

  O AHP é um método de ajuda à tomada de decisão fundamentada recorrendo a cálculos matemáticos e pertencendo ao ambiente das decisões multicritério discretas, foi desenvolvido na década de 1980 por Thomas L. Saaty [11]. Este método permite o uso de critérios qualitativos e quantitativos no processo de avaliação. A ideia principal é dividir o problema de decisão em níveis hierárquicos, facilitando a sua compreensão.

  Nos variados estudos publicados nesta área, são referenciadas duas linguagens de programação mais utilizadas, *R* e *Python*. Foram definidos o objetivo e critérios:

  Objetivo: Escolha da linguagem de programação.
  Critérios: Curva de aprendizagem, engenharia de *software*, bibliotecas de *machine learning*, visualização de dados.
  Alternativas: R e *Python*

Foi deduzida a árvore hierárquica de decisão, enunciando o objetivo geral de decisão, os critérios associados ao problema de decisão e as alternativas disponíveis e mais adequadas.

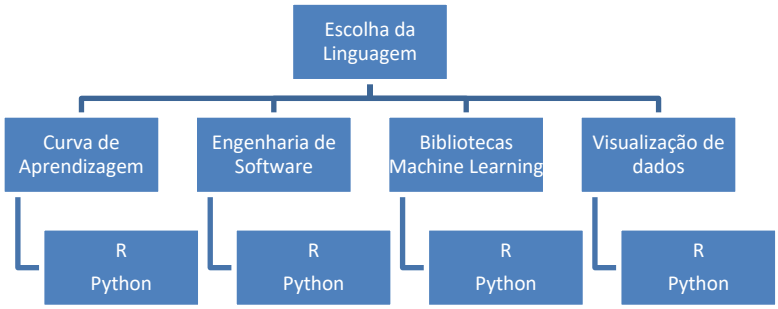


Figura 3: Árvore de decisão linguagem de programação

Considerando os 4 critérios da estrutura hierárquica foi desenvolvida a seguinte matriz de comparação para a par quadrada:

Critérios	Curva Aprendizagem	Engenharia de Software	Bibliotecas Machine Learming	Visualização dos dados
Curva Aprendizagem	1,00	0,13	0,25	0,33
Engenharia de Software	8,00	1,00	5,00	3,00
Bibliotecas Machine Learming	4,00	0,20	1,00	2,00
Visualização dos dados	3,00	0,33	0,50	1,00
soma	16,00	1,66	6,75	6,33

Tabela 1: Matriz de comparação para a par quadrada

Escrevendo os valores da tabela em forma de matriz obtemos:

$$A = \begin{bmatrix} 1 & 0.13 & 0.25 & 0.33 \\ 8 & 1 & 5 & 3 \\ 4 & 0.2 & 1 & 2 \\ 3 & 0.33 & 0.5 & 1 \end{bmatrix}$$

Para aferir a **prioridade relativa** de cada critério é necessário proceder à normalização dos valores da matriz de comparações (matriz A), para tal cada valor é dividido pelo total da sua respetiva coluna. A obtenção do vetor de prioridades é feita calculando a média aritmética dos valores de cada linha, tendo por objetivo identificar a ordem de importância de cada critério:

Critérios	Curva Aprendizagem	Engenharia de Software	Bibliotecas Machine Learming	Visualização dos dados	Prioridade Relativa
Curva Aprendizagem	0,06	0,08	0,04	0,05	0,06
Engenharia de Software	0,50	0,60	0,74	0,47	0,58
Bibliotecas Machine Learming	0,25	0,12	0,15	0,32	0,21
Visualização dos dados	0,19	0,20	0,07	0,16	0,16

Tabela 2: Matriz normalizada e prioridade relativa

A partir dos resultados obtidos, o critério engenharia de *software* aparece em primeiro lugar, seguido de bibliotecas ML, visualização dos dados e por fim curva de aprendizagem. A Matriz de comparação dos critérios normalizada é apresentada a seguir bem como o vetor de prioridades ou vetor próprio X:

$$\begin{bmatrix} 0.06 & 0.08 & 0.04 & 0.05 \\ 0.5 & 0.6 & 0.74 & 0.47 \\ 0.25 & 0.12 & 0.15 & 0.32 \\ 0.19 & 0.20 & 0.07 & 0.16 \end{bmatrix} \rightarrow X = \begin{bmatrix} 0.06 \\ 0.58 \\ 0.21 \\ 0.16 \end{bmatrix}$$

A próxima etapa procede-se ao cálculo da Razão de Consistência (RC), que mede o quanto os julgamentos foram consistentes em relação a grandes amostras de juízos completamente aleatórios.

$$A * X = \begin{bmatrix} 0.23 \\ 2.54 \\ 0.86 \\ 0.62 \end{bmatrix}$$

λmax = average{0.23/0.06, 2.54/0.58, 0.86/0.21, 0.62/0.16}=4.15

O índice de consistência IC = (λmax -n)/(n-1) =(4.15-4)/(3)=0.053

De acordo com a tabela de ponderação de critérios:

1	2	3	4	5
0.00	0.00	0.58	0.90	1.12

Tabela 3: Tabela de ponderação de critérios escolha de linguagem

RC = IC/0.90 = 0.053/0.90 = 0.059 < 0.1, podendo assim concluir que os valores das prioridades relativas estão consistentes.

Por fim é necessário construir as matrizes de comparação paritária para cada critério, considerando cada uma das alternativas:

Matriz de comparação paritária				**Matriz Normalizada**			
Curva Aprendizagem	**R**	**Python**		**Curva Aprendizagem**	**R**	**Python**	**Prioridade Relativa**
R	1,00	0,17		**R**	0,14	0,14	0,14
Python	6,00	1,00		**Python**	0,86	0,86	0,86
soma	7,00	1,17					

Tabela 4: Tabela de matriz de comparação e normalizada para Curva de Aprendizagem

Matriz de comparação paritária				**Matriz Normalizada**			
Engenharia de Software	**R**	**Python**		**Engenharia de Software**	**R**	**Python**	**Prioridade Relativa**
R	1,00	0,13		**R**	0,11	0,11	0,11
Python	8,00	1,00		**Python**	0,89	0,89	0,89
soma	9,00	1,13					

Tabela 5: Tabela de matriz de comparação e normalizada para Engenharia de *Software*

Matriz de comparação paritária				**Matriz Normalizada**			
Bibliotecas Machine Learming	**R**	**Python**		**Bibliotecas Machine Learming**	**R**	**Python**	**Prioridade Relativa**
R	1,00	2,00		**R**	0,67	0,67	0,67
Python	0,50	1,00		**Python**	0,33	0,33	0,33
soma	1,50	3,00					

Tabela 6: Tabela de matriz de comparação e normalizada para Bibliotecas de ML

Matriz de comparação paritária				**Matriz Normalizada**			
Visualização dos dados	**R**	**Python**		**Visualização dos dados**	**R**	**Python**	**Prioridade Relativa**
R	1,00	3,00		**R**	0,75	0,75	0,75
Python	0,33	1,00		**Python**	0,25	0,25	0,25
soma	1,33	4,00					

Tabela 7: Tabela de matriz de comparação e normalizada para Visualização dos Dados

Podemos agora completar a árvore hierárquica com os valores de referência:

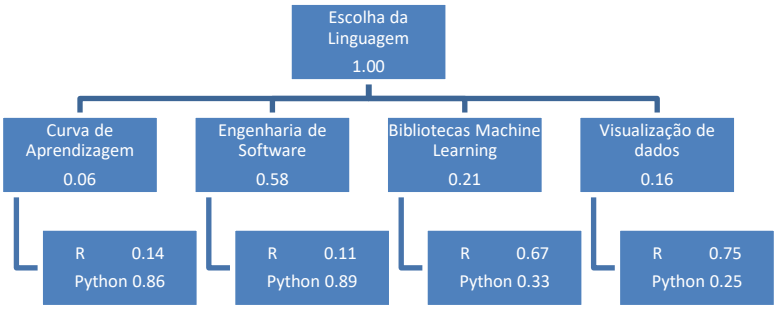


Figura 4: Árvore de decisão linguagem de programação com valores de referência

Finalmente obtemos a prioridade composta para as alternativas, procedendo à multiplicação da matriz de prioridade, pelo vetor dos pesos dos critérios:

$$\begin{bmatrix} 0.14 & 0.11 & 0.67 & 0.75 \\ 0.86 & 0.89 & 0.33 & 0.25 \end{bmatrix} * \begin{bmatrix} 0.06 \\ 0.58 \\ 0.21 \\ 0.16 \end{bmatrix} = \begin{bmatrix} 0.32 \\ 0.67 \end{bmatrix}$$

Em função dos critérios definidos e das suas respetivas importâncias, a linguagem *Python* aparece como a mais indicada no desenvolvimento dos modelos.

Criação de valor

Para definir a criação de valor neste projeto é necessário primeiro perceber o conceito de valor no contexto empresarial. Valor é a forma como reagimos às condições do ambiente onde estamos inserimos [12]. O conceito de valor é definido em diferentes contextos teóricos como necessidade, desejo, interesse, critérios, crenças, atitudes e preferências. A criação de valor é uma peça chave em qualquer negócio e qualquer atividade empresarial baseia-se na troca de bens ou serviços tangíveis e/ou não tangíveis, sendo o seu valor aceite e premiado por parceiros ou clientes, no interior ou exterior da empresa ou rede colaborativa [13]. A criação de valor é um conceito difícil de atingir, perceber, modelar ou conceptualizar. Alguns autores consideram ser uma constante negociação entre benefícios e sacrifícios percecionados pelos clientes no momento da análise da oferta. O valor pode ser comunicado em duas vertentes distintas, valores pessoais: facilidade de utilização; saúde; estética; novidade; status; inclusão social; atualização automática; ética, e valores de negócio: baixo risco; poupança de tempo; poupar ou ganhar dividendos; ativar funcionalidades; qualidade; ambiente; customização; utilidade.

Procedendo à identificação de criação de valor do projeto apresentado, são analisados os seguintes critérios:

- **Valor para o cliente (*value for the costumer*)**
  Da definição de valor para o cliente, caracterizado como sendo qualquer perceção pessoal de vantagem por parte do cliente face ao oferecido por parte da organização, que pode acontecer como redução do sacrifício; presença de benefício; a resultante de qualquer combinação ponderada entre sacrifício e benefício; ou agregação, ao longo do tempo, de todas as anteriores [14].
- **Valor percecionado (*perceived value*)**
  Em marketing o termo valor para o cliente é usado para definir a imagem que o cliente tem do fornecedor e também o inverso. Diferentes clientes percecionam diferente valor para os mesmos produtos ou serviços e ao longo da cadeia de produção as organizações envolvidas podem ter diferentes perceções do valor entregue ao cliente [15], os produtores são menos sensíveis ao preço, por exemplo, enquanto o consumidor é mais sensível à qualidade do produto [16].

Neste projeto considera-se como valor para o cliente, no âmbito do produto os seguintes benefícios: a disponibilização da informação relativa à implementação de todo o sistema de deteção de fraude; a solução permitir avaliar diferentes modelos matemáticos e estatísticos; o estudo e identificação dos modelos com melhor performance; a indicação dos processos de seleção e decisão do *software* utilizado; o uso de *software* confiável com suporte de empresas reconhecidas e com completas publicações disponíveis online de acesso publico. Por outro lado, ao incluir no projeto *software* de distribuição livre, tentamos minimizar o sacrifício dos custos que sistemas deste tipo comportam, no entanto carece de investimento de tempo e formação especializada do lado do cliente para a implementação da solução. Para o cliente o valor percebido irá depender, numa fase de uso e experiência, da eficiência do sistema detetar com a devida antecedência as transações fraudulentas no sistema de pagamentos do cliente.

A proposta de valor do negócio pode ser enunciada da seguinte forma:

Estudo e descrição da implementação de um sistema de deteção de fraude em transações online com cartões de crédito, usando *software* de distribuição livre. A Solução permite a alteração de vários parâmetros, apresentando métricas de avaliação dos modelos em teste, para aferição dos modelos mais eficientes na deteção de fraude em tempo real. São ainda divulgados os estudos efetuados com a solução apresentada, para o cenário típico de deteção de fraude sobre um conjunto de dados predefinido e tipificado, sendo apresentados os resultados e conclusões obtidas com o usos de várias técnicas de pré-processamento e para os vários modelos inferidos dos dados.

Foi utilizado o modelo de negócios *Canvas* [17], para representar esquematicamente a perspetiva de negócio deste projeto (fig. 5).

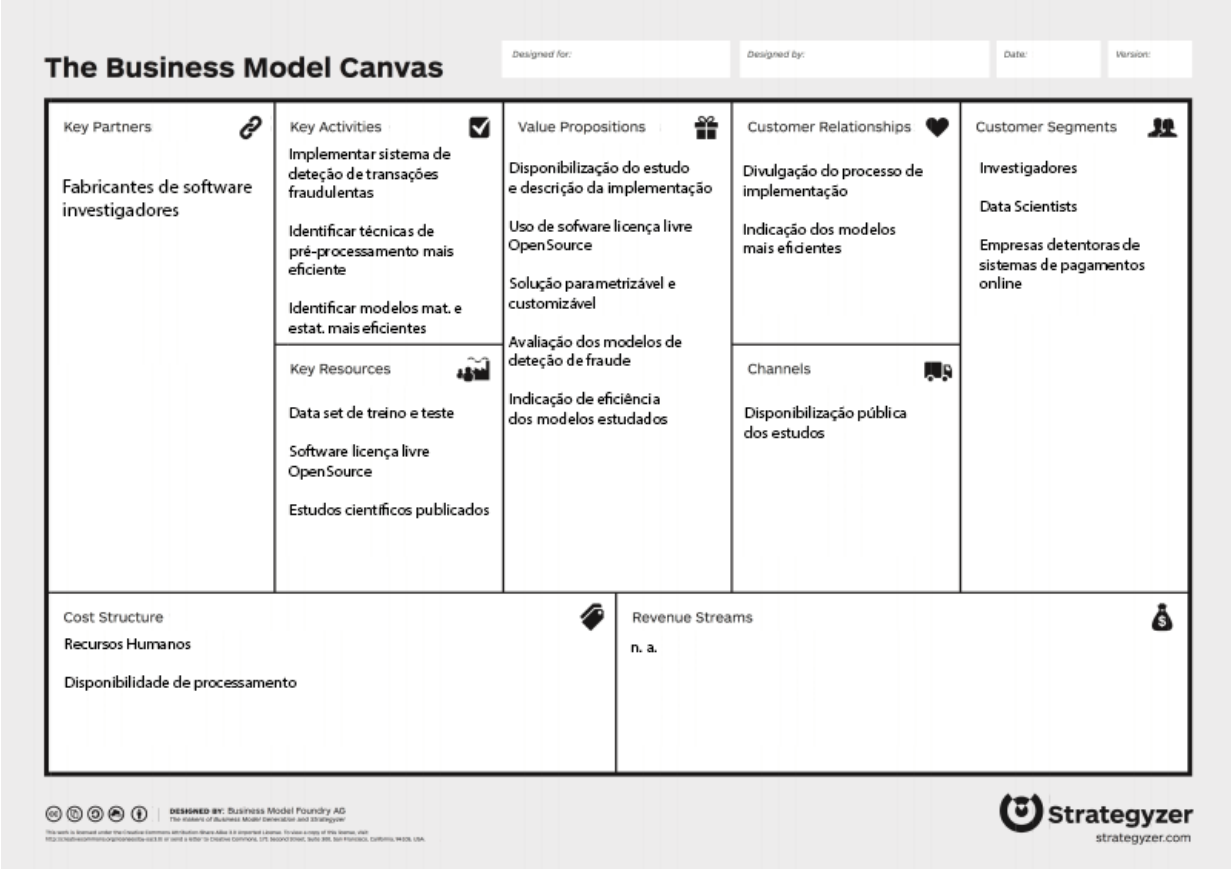


Figura 5: *Business Model Canvas* do Projeto

3 Estado de arte

3.1 Estado de arte em abordagens existentes

Muitos investigadores têm vindo a usar algoritmos de data mining para a deteção de transações fraudulentas. As abordagens que têm como base o historial de transações, podem ser divididas em duas categorias: manuais, como as provas de identidade, e as automáticas, maioritariamente data mining. Em cenários de grande volume de dados as técnicas manuais são impraticáveis sendo imperativo o uso de técnicas mais automatizadas, com foco em métodos supervisionados como as redes neurais artificiais, máquinas de vetor de suporte e regressão logística [1]. No que diz respeito aos algoritmos supervisionados como os de classificação, necessitam de uma identificação precisa das transações fraudulentas e não fraudulentas na fase de aprendizagem, informação esta que nem sempre existe ou então é limitada, resultando em sistemas onde a deteção não é executada no momento da transação. É necessário então sistemas que respondam em tempo real. As técnicas de data mining que têm em conta as transações anteriores para estimar a probabilidade de novas transações serem ou não fraudulentas, estão entre as técnicas com mais sucesso na deteção de fraude [18].

3.1.1 Técnicas de data mining para deteção de fraude

A abordagem mais eficiente na deteção de fraude é inferir possíveis provas de fraude dos dados existentes usando algoritmos [3]. As técnicas de data mining, que usam métodos estatísticos avançados e técnicas de inteligência artificial, estão divididas em duas abordagens: supervisionadas e não supervisionadas. Ambas as abordagens são baseadas no treino de um algoritmo com um conjunto de dados obtido em observações anteriores. Os métodos supervisionados necessitam que cada observação contenha uma indicação da classe a que pertence, “fraudulenta” ou “legitima”. Nem sempre é conhecida a classe de uma observação, por exemplo uma ordem onde o pagamento foi recusado, podemos não saber a causa, se era uma ordem legitima ou corretamente rejeitada. Este tipo de ocorrências são a razão para o uso de métodos não supervisionados. Estes métodos identificam dados extremados ou *outliers*. O uso das duas técnicas é aconselhado para a obtenção de resultados mais fidedignos. Alguns estudos foram efetuados sobre as diferentes abordagens. Cortes et al. [19] explora a análise de gráficos para a deteção de fraude. Quah and Sriganesh [1] propõem uma abordagem mista recorrendo ao uso de um mapa que se auto-organiza e que alimenta uma rede neural se a transação não se inserir num comportamento normal para um dado cliente. Moreau et al. [20] efetuou a comparação entre redes neurais supervisionadas e não-supervisionadas. Como resultado destas experiências, os métodos não supervisionados obtiveram resultados bastante aquém dos métodos supervisionados. Nos anos 90 e princípios dos anos 2000, a combinação de classificadores múltiplos foram propostos como tentativa da criação de sistemas escaláveis capazes de lidar com grandes volumes de dados. Mais recentemente outros estudos foram publicados onde outras técnicas de classificação são utilizadas. Sistemas de deteção de fraude

têm sido construídos usando algoritmos genéticos [21] [22]. Srivastava et al. [23] construiu um modelo baseado no Hidden Markov Model (pode ser representado como uma rede bayesiana dinâmica) com foco na deteção de fraude para emissão de cartões de crédito por parte dos bancos. Whitrow et al. [24] também trabalharam na área da fraude com cartões de crédito, estudando o uso de agregação das transações quando se usa florestas aleatórias, máquina de vetores de suporte, regressão logística e K vizinhos mais próximos. Bhattacharyya et al. [25] compara a performance entre florestas aleatórias, regressão logística, e máquina de vetores de suporte na deteção de fraude nas transações com cartões de crédito. Provou-se que as florestas aleatórias são mais eficazes e mais versáteis para estes casos. Foram também exploradas técnicas para ultrapassar o problema de amostras de fraude desbalanceadas [26]. Para este trabalho seguimos a proposta de Nuno Carneiro et al. [4] que escolhem os métodos de aprendizagem supervisionada para o problema de classificação, devido à disponibilidade de dados etiquetados.

3.1.2 Técnicas de pré-processamento para dados desbalanceados

Classes desbalanceadas são um problema comum na classificação em *machine learning* devido à desproporção do rácio de observações em cada classe. A base de dados de transações utilizada neste estudo, apresenta um desbalanceamento elevado visto as transações classificadas como fraudulentas representarem apenas 0,17% do total de transações. A maioria dos algoritmos usados em *machine learning* apresentam melhores prestações quando o número de amostras em cada classe é quase idêntico. Isto acontece porque a maior parte dos algoritmos são desenhados para maximizar o valor de *accuracy* e reduzir o erro. Neste trabalho é estudado o impacto de duas técnicas de *resampling* na prestação dos vários modelos. *Oversampling* da classe minoritária, pode ser definida como a adição de mais copias da classe minoritária e *undersampling*, que consiste na remoção de amostras da classe maioritária.

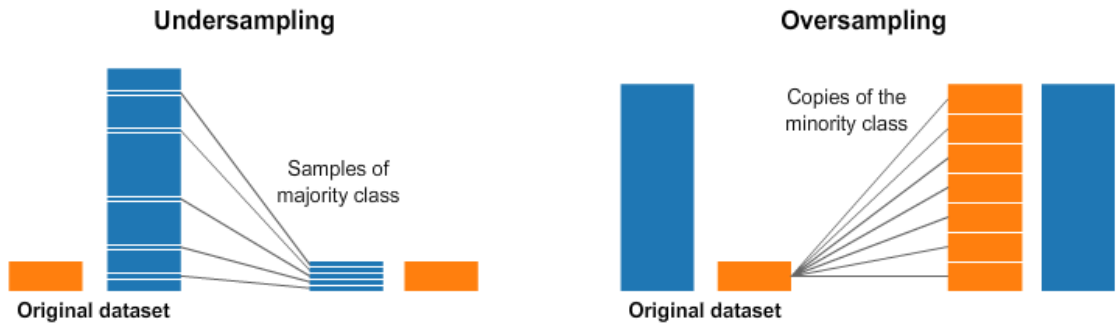


Figura 6: *Resampling* (*oversampling* e *undersampling*): fonte kaggle.com (2017)

Como técnica de *undersampling* foi usada *Cluster Centroids*, que se caracteriza pela aplicação de um algoritmo de *clustering* aos dados, assignando cada valor a um *cluster*, reduzindo as amostras em cada *cluster* em proporção ao tamanho do *cluster*. Como técnica de *oversampling* foi usada SMOTE (*Synthetic Minority Oversampling Algorithm*), esta técnica caracteriza-se pela criação de novas amostras da classe minoritária, extrapoladas de uma observação e cada um dos seus vizinhos mais próximos.

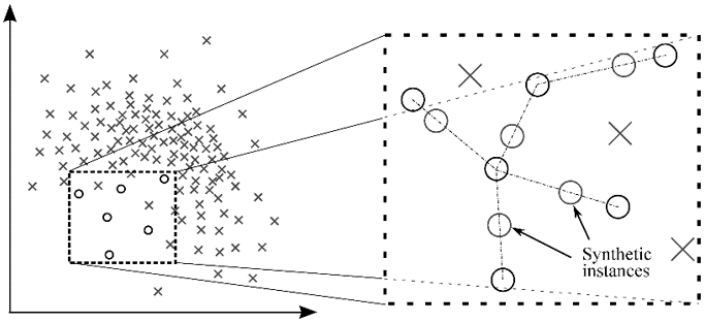


Figura 7: SMOTE *oversampling*: fonte intechopen.com (2012)

3.1.3 Algoritmos de Classificação

Neste trabalho foram estudados vários algoritmos de classificação. Para cada algoritmo existem diferentes parâmetros que podem ser afinados para melhor prestação. Foi usada para cada algoritmo a técnica de otimização de híper-parâmetros, com a abordagem *Grid Search,* sendo definido para cada parâmetro um intervalo e a cada iteração obter a melhor combinação de parâmetros para o algoritmo em teste.

Regressão Logística

O modelo de regressão logística adapta técnicas de regressão linear para a determinação de uma superfície de separação entre duas classes (numericamente 0 ou 1, classificação binária) discriminada por uma curva sigmóide.

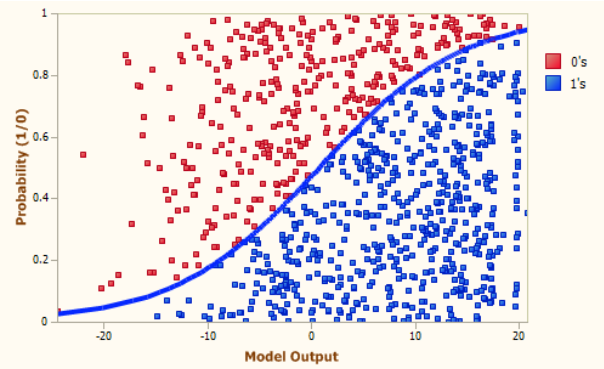


Figura 8: Regressão logística: fonte Koirala, towardsdatascience.com (2019)

A função logística apresenta uma curva em forma de "S" (Sigmóide) onde as probabilidades de ocorrer um evento são representadas no intervalo entre 0 e 1.

Máquinas de Vetores

O principal objetivo deste classificador é encontrar um hiperplano onde a margem é máxima entre os vetores de suporte dos dados. Para tal o classificador testa vários hiperplanos escolhendo o que tem menor valor de erro de classificação e maior margem entre os pontos de suporte dos vetores das diferentes classes (fig. 9).

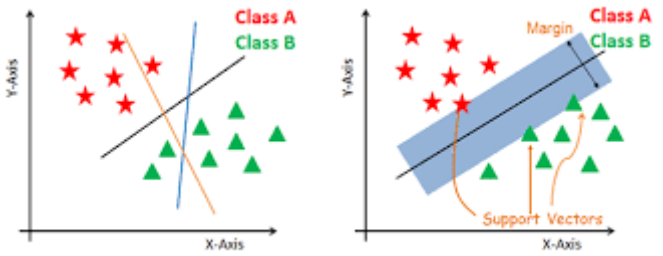


Figura 9: Máquina de vetores, plano linear, fonte: datacamp.com (2018)

No entanto para planos não lineares ou inseparáveis, o classificador faz uso de *Kernels* para elevar a dimensão do espaço inicial. Os pontos do plano são desenhados no plano x e z (z é a soma quadrática de x e y: z=x^2=y^2), sendo possível assim aplicar separação linear (fig. 10).

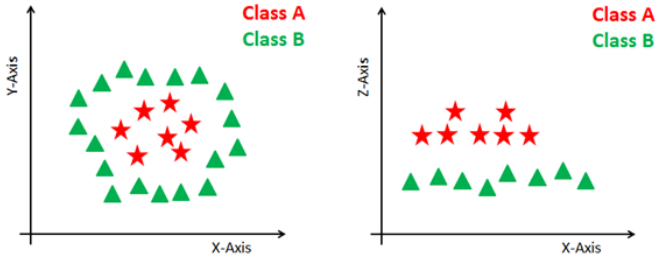


Figura 10: Máquina de vetores, plano não-linear, fonte: datacamp.com (2018)

O *kernel* transforma o espaço dos dados na forma necessária, aumentando a dimensionalidade do espaço, convertendo um conjunto não linearmente separável em vários conjuntos separáveis adicionando para tal mais dimensões. Para este trabalho foram testados os *kernels* do tipo linear, polinomial e radial.

Florestas Aleatórias

A ideia fundamental é a combinação de várias árvores de decisão num único modelo. Individualmente as previsões feitas por cada árvore pode não ser a mais correta, mas combinadas, a média das previsões das várias árvores está mais perto do objetivo. Cada árvore de decisão da floresta considera um subconjunto dos atributos e a um bloco aleatório do conjunto de treino, aumentando a diversidade de previsões e criando um modelo mais robusto. Como classificador, onde o alvo é discreto (1 ou 0) o algoritmo obtém o voto maioritário das várias árvores para classificar um novo elemento (fig. 11).

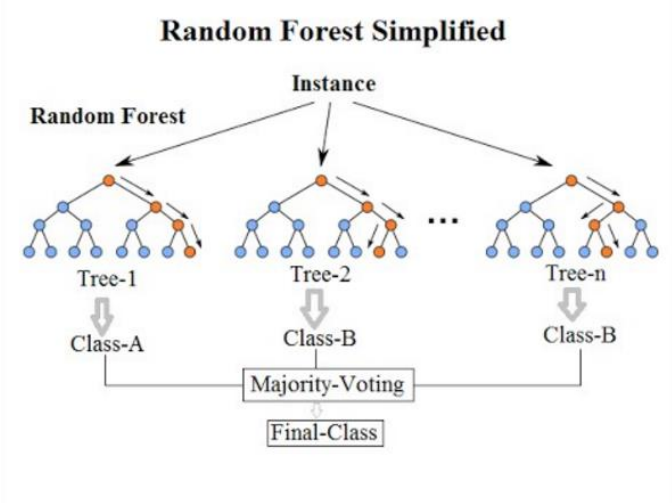


Figura 11: Florestas aleatórias, fonte: Koehrsen, médium.com (2017)

XGBoost

Este algoritmo é baseado em árvores de decisão fazendo uso de uma estrutura de *gradient boosting*. É um método que utiliza um conjunto de árvores e aplica o princípio de potenciar um *weak learner* através de uma arquitetura de gradiente descendente. O algoritmo *XGBoost* melhora as técnicas de *gradient boosting* através de otimização do sistema e melhoramentos do algoritmo [27].

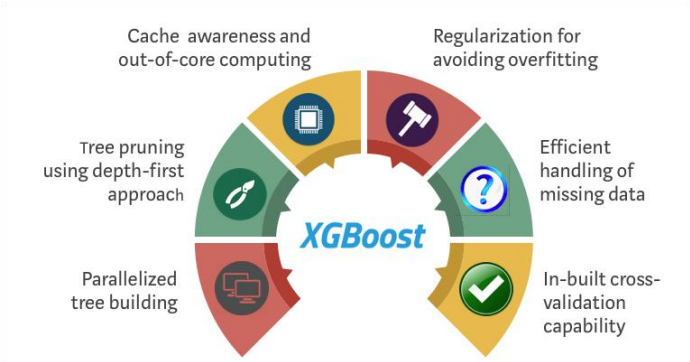


Figura 12: *XGBoost*, fonte: towardsdatascience.com (2019)

- **Otimização do algoritmo**
  Paralelização: no processo de construção sequencial das árvores é feita uma implementação paralelizada. Dada a natureza intermutável dos ciclos de ganho de conhecimento, o ciclo exterior que enumera os nós da árvore e o ciclo interior que calcula os pesos, é possível melhorar os tempos de processamento. A ordem dos ciclos é inicializada efetuando um *scan* global de todas as instâncias e ordenação usando tarefas paralelas, aumentando a performance do algoritmo diminuindo o *overhead* da paralelização durante a computação.

Pruning: uso do parâmetro "*max_depth*" para execução de *pruning* em profundidade. Otimização de *hardware*: gestão de espaço em *cache* por tarefa e gestão em disco de informação que exceda o espaço de memoria.

- **Melhoramento do algoritmo**
  Regularização: penalização de modelos mais complexos por intermédio de *LASSO* (L1) e *Ridge* (L2)[28] para evitar *overfitting*.
  Consciência de Escassez de dados: cálculo automático de dados em falta por aprendizagem do melhor valor em falta, analisando a perda durante o treino.
  Definição do quartil ponderado: aplicação do algoritmo "Weighted Quantile Sketch" [27] para encontrar os pontos ótimos de divisão dos conjuntos de dados.
  Validação cruzada: o algoritmo tem integrado o método de validação cruzada (cross-validation) automática para cada iteração.

3.2 Estado de arte em tecnologia relevante

3.2.1 Apache Spark vs Hadoop MapReduce

Com a variedade de ferramentas presentes no mercado para o tratamento de *Big Data*, por vezes é complexo saber qual a melhor opção. *MapReduce* é um modelo de programação usado para processar grandes conjuntos de dados, que podem ser automaticamente paralelizados e implementados em cluster de máquinas. Também é de fácil utilização para desenvolvedores com pouca experiência profissional tanto em sistemas paralelos como em sistemas distribuídos. Por outro lado, temos o *Apache Spark* que atua como um mecanismo base para processamento de grande volume de dados de forma distribuída e paralela. As suas funções incluem: Gestão de memória e tolerância a falhas; Interação com sistemas de armazenamento de dados; agendar, distribuir e monitorizar tarefas em cluster. Relativamente ao processamento tecnológico de *Big Data*, o *Hadoop MapReduce* está presente no mercado há mais tempo que o *Apache Spark*. Comprova-se que é uma ferramenta capaz de processar grandes volumes de dados, no entanto é menos eficiente pois recorre a algoritmos *one-pass*, enquanto que o *Spark* utiliza algoritmos *multi-pass*. Além disso o *MapReduce* força os desenvolvedores a transformar os casos de uso consoante os padrões do *MapReduce*, pois o mesmo apenas funciona em dois passos (*Map phase* e *Reduce phase*). Todos os dados processados na primeira etapa necessitam de ser armazenados num sistema de dados distribuído para que se inicie o seguinte passo, o que leva este mecanismo a ter uma pior performance.

Com o *Spark*, os desenvolvedores podem implementar pipelines de dados complexos, processados em várias etapas, utilizando o padrão *Directed Acyclic Graph* (DAG). Recorrendo a este padrão é possível que diferentes tarefas acedam aos mesmos dados, aumentando assim a velocidade de processamento. Em 2014, *Spark* venceu o prémio *Daytona GraySort Contest*, onde conseguiu ordenar 100TB de dados em apenas 23 minutos, utilizando no total de 206 máquinas, enquanto que o *MapReduce* demorou 72 minutos, e ainda teve de recorrer ao uso

de 2100 máquinas. Este resultado demonstra a elevada capacidade do *Spark* e as vantagens obtidas através do uso desta ferramenta.

3.2.2 Arquitetura adotada

Neste projeto é estudada uma implementação escalável do módulo de aprendizagem, que se baseia nas ferramentas do ecossistema Apache, a saber Kafka, Spark e Cassandra [35]. A grande vantagem destes componentes é a semelhança como executam a tolerância a falhas e o processamento distribuído de tarefas [2].

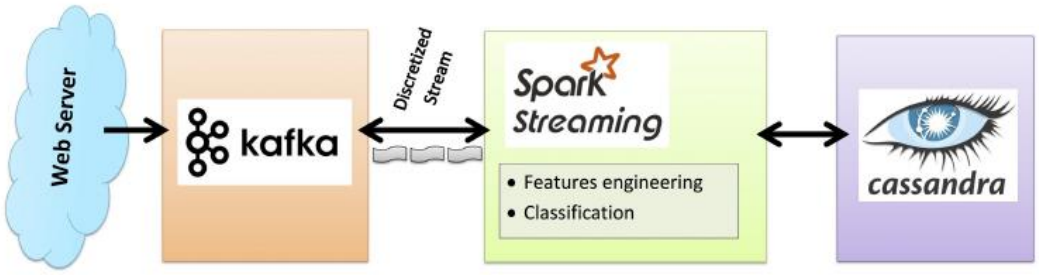


Figura 13: Ecossistema de deteção de fraude em tempo real usando o *Spark*

Os dados entram no sistema através do *Apache Kafka* que pode ser parametrizado com a quantidade de dados por segundo a processar, gerindo a partição e retenção de dados, simulando assim vários cenários de um serviço real. O *Apache Spark* tem a capacidade de gerir a computação distribuída ao nível da memória, agregando os resultados num sistema de ficheiros distribuídos. O *Spark* contém uma biblioteca própria de *machine learning* (MLib [36]) e uma biblioteca para gestão e tratamento de dados em *streaming*. Para a persistência dos dados para avaliação é usado o módulo *Apache Cassandra*, que comporta uma base de dados distribuída desenhada para suportar escalabilidade e gerir replicações em variados *datacenters*.

3.2.3 Linguagem de programação

As duas linguagens de programação mais referenciadas em *Data Mining* são a linguagem R e *Python* [29]. A linguagem R é uma linguagem *open source* para computação estatística e gráfica e apresenta um ambiente de programação próprio foi inicialmente escrita por Ross Ihaka e Robert Gentleman em 1993 [30]. O R fornece algoritmos para análise de dados, *Data Mining*, construção de sistemas de recomendação, extração de dados web, otimização matemática, entre outros. A facilidade para desenhar gráficos que ajudam a analisar os problemas é uma das suas caraterísticas principais. O R apresenta uma enorme comunidade de utilizadores e programadores que contribuem com constantes melhoramentos e atualizações resultando em novas bibliotecas de análise e um número vasto de algoritmos de *Data Mining*.

O *Python* é uma linguagem *open source* orientada a objetos foi criada em 1991 por Guido Van Rossum [31]. Tal como a linguagem R, o *Python* permite fazer análise de dados e inclui bibliotecas para usar algoritmos de *Data Mining*. Pode ser facilmente integrado com outras linguagens de programação e a sua utilização para desenvolver aplicações de *Data Mining* tem aumentado nos últimos anos. Segundo o ranking de linguagens de programação TIOBE, a linguagem *Python* estava na posição 3, no mês de fevereiro de 2019 sendo que no período homologo estava na posição 4 [32] . Após a avaliação apresentada no segundo capítulo deste documento (contexto e estado da arte), em função dos critérios definidos e das suas respetivas importâncias, a linguagem *Python* é a mais indicada no desenvolvimento dos modelos para o projeto apresentado.

3.3 Trabalhos relacionados

Dos trabalhos analisados para este projeto, dois deles foram considerados como abordagens a seguir para este projeto Fabrizio Carcillo et al., 2017 [2] e Nuno Carneiro et al., 2017 [4], pois são focados na problemática da deteção de fraude em tempo real e propõem uma arquitetura completa descrevendo os vários módulos a serem considerados. No que diz respeito aos classificadores usados no motor de *machine learning*, ambos os estudos apontam o classificador Floresta Aleatória como o mais eficiente, sendo, no entanto, testados outros como a regressão logística e máquinas de vetores de suporte. Dado o característico não balanceamento dos conjuntos de dados relativos às transações efetuadas num determinado período de tempo, que servem para treinar e testar os modelos de deteção de fraude, ambos os trabalhos efetuam um pré-processamento dos dados recorrendo à técnica de *undersampling*, propondo no entanto para trabalhos futuros que se avalie a influência na precisão do sistema do uso de outras técnicas. Como métrica de avaliação da performance dos modelos, são apurados nos estudos os valores de AUC-ROC.

Analisando os resultados obtidos em outros estudos, no cenário proposto por Nuno Carneio et al. é aplicada somente a técnica de *undersampling,* numa base de dados própria da qual só sabemos que mantem o desbalanceamento típico entre classes, onde são testados dois classificadores, Regressão Logística e Máquina de Vetores que não apresentam diferença significativa na performance, no entanto onde obtiveram melhor resultado foi na aplicação do classificador de Florestas Aleatórias sem qualquer sampling (AUC-ROC – 0.935). Já nos estudos de Martinez J. é comparada a influencia da técnica de sampling sobre o classificador de Regressão Logística, para a mesma base de dados usada nesta tese, sendo a melhor prestação com a técnica de *undersampling,* ultrapassando por pouco as prestações do modelo Máquina de vetores com undersampling. Como referência para a aplicação do classificador *XGBoost* temos o trabalho de Morgan L. que estuda a prestação de um modelo com oversampling.

Autor	Modelo	Base de Dados	AUC-ROC
Nuno Carneiro et al., 2017 [4]	Under + Regressão Log.	Própria	0.903
	Under + SVC		0.906
Martinez J. (kaggle) [40]	Under + Regressão Log	creditcard.csv	0.980
	Smote + Regressão Log		0.910
	Under + SVC		0.975
Morgan L. [41]	Smote + *XGBoost*		0.984

Tabela 8: Resultados obtidos por outros estudos

4 Deteção de fraude em pagamentos eletrónicos

4.1 Abordagem adotada

São consideradas as seguintes etapas no desenvolvimento do projeto: escolha, carregamento e tratamento do conjunto de dados para treino e teste dos modelos; testes dos diferentes classificadores cruzando com as técnicas de pré-processamento; análise das métricas de eficiência dos modelos; implementação do sistema de deteção de fraude.

De acordo com uma perspetiva baseada em descoberta de conhecimento em bases de dados e a metodologia de trabalho CRISP-DM [33] este projeto enquadra-se da seguinte forma:

1. **Compreensão do domínio de aplicação** – A análise do estado da arte permitiu concluir esta tarefa.
2. **Seleção e criação do conjunto de dados** – Para treino e teste dos modelos foi selecionado um conjunto de dados com transações reais relativas a um determinado período de tempo e tipificadas como sendo ou não fraudulentas. O conjunto de dados utilizado é disponibilizado para fins de estudo de forma gratuita na comunidade *kaggle* [7].
3. **Análise dos dados** – É efetuada a análise da qualidade e dispersão dos dados, verificando as grandezas e relações entre atributos.
4. **Normalização e limpeza dos dados** – Os dados constantes do conjunto de dados utilizado já se encontram reduzidos pela técnica PCA (Principal Component Analysis for Dimensionality Reduction) [34], pelo que foram previamente normalizados. Não constam da base de dados valores nulos ou em falta. Procedeu-se à normalização de duas colunas relativas ao valor e tempo entre transferências.
5. **Pré-processamento dos dados** – Dada a natureza dos dados extremamente desbalanceados (a quantidade de transações fraudulentas à volta de 0.17% do total dos registos), como tal é necessário aplicar técnicas de pré-processamento como *undersampling* e *oversampling*.
6. **Escolha do Classificador** – São testados os algoritmos mais indicados pela literatura para a deteção de fraude, baseados em florestas aleatórias, máquina de vetores, regressão logística e *XGBoost*.
7. **Testes e Avaliação de Modelos** – Os algoritmos serão avaliados usando as métricas de taxa de acerto e pela análise da área da curva *ROC*.
8. **Implementação de sistema de deteção em tempo real** – Estudo do sistema de deteção de fraudes em tempo real usando serviços e produtos de acesso gratuito.

Podemos separar os vários pontos da metodologia CRISP_DM em duas fases distintas neste projeto, uma fase exploratória, onde se pretende aferir qual o modelo com melhores

prestações na deteção de transações fraudulentas e uma fase de implementação de um sistema de deteção em tempo real com base no modelo eleito.

A fase exploratória comporta as tarefas relacionadas com a análise, o treino e teste dos modelos em estudo, culminando com a eleição do modelo com melhor prestação na tarefa de classificação das transações. O fluxo de tarefas a executar nesta fase está representado na figura seguinte.

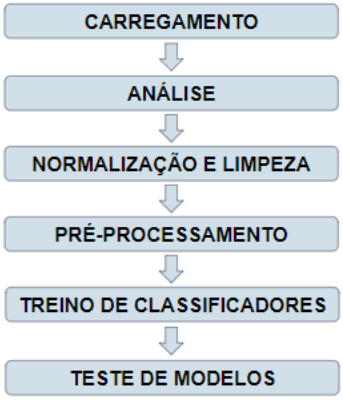


Figura 14: Design fase exploratória

Em primeiro lugar procede-se ao carregamento do conjunto de treino e teste para o ambiente de programação onde se efetua a análise de conformidade dos dados, nomeadamente o tipo de dados de cada atributo e verifica-se a necessidade de normalização e limpeza. No pré-processamento aplicam-se as técnicas de *undersampling* e *oversampling* sendo criados os conjuntos de dados que servirão para alimentar os classificadores em estudo. Após o treino e teste dos vários classificadores obtêm-se as métricas de avaliação que permitem aferir qual o melhor modelo para a deteção de fraude para o tipo de dados em estudo.

A fase de implementação é onde o sistema de deteção em tempo real é concretizado, colocando o modelo selecionado em produção.

4.2 Fase Exploratória

4.2.1 Ecossistema

A fase exploratória do projeto engloba o desenvolvimento e teste dos modelos foi usada localmente a aplicação *Jupyter Notebook* [42], que permite a criação e partilha de documentos com código, equações, visualizações e texto descritivo, em linguagem *Python*. Para o teste dos vários classificadores foi usada a plataforma gratuita *Colaboratory* da *Google* [43], que permite o uso partilhado de servidores disponibilizados em *cloud* com disponibilidade de processadores *GPU* (*Graphics Processing Unit*, ou Unidade de Processamento Gráfico) e TSU (*Tensor Processing Unit*) para implementação de tecnologias *Machine Learning (ML)*.

4.2.2 Carregamento e análise

Após o carregamento do conjunto de dados original (creditcard.csv) procedeu-se à respetiva análise dos dados. Este conjunto de dados contém transações efetuadas em setembro de 2013 por detentores de cartões de origem europeia. As operações ocorreram no período de dois dias, num universo de 284807 transações. Por razões de privacidade não é divulgada informação sobre os dados e atributos. Excetuando "*Time*" e "A*mount*", todos os restantes atributos numéricos (V1 a V28) foram obtidos utilizando a técnica de transformação PCA [34]. "*Class*" é a classe objetivo e toma o valor "1" no caso de fraude e "0" se a transação não for fraudulenta.

```
df = pd.read_csv('creditcard.csv')
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0 0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0 0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0 0.219422	0.215153	69.99	0

5 rows × 31 columns

Figura 15: Amostra do conjunto de dados original

Não há dados com entrada nula, logo não houve a necessidade de aplicação de técnicas de tratamento e substituição de registos nulos.

Como esperado a maioria das transações são não fraudulentas (99.83%), representando as transações fraudulentas uma percentagem de 0.17% do total.

Foi necessário proceder à normalização dos valores dos atributos "*Time*" e "*Amount*", usando a classe *RobustScaler()* da biblioteca *sklearn.preprocessing* [44] sendo esta menos afetada pela presença de *outliers* ao contrário da classe *StandardScaler* [45].

4.2.3 Pré-processamento

O desbalanceamento de dados já identificado no conjunto de dados original deste estudo, usado nos modelos preditivos propostos, é propenso a erros de *overfitting* [46], induzindo os modelos erradamente a assumir na maioria dos casos resultados não fraudulentos, prejudicando também a análise de correlações entre os atributos e a classe alvo. Como o pretendido é um modelo que detete padrões de fraude, os dados de treino necessitam de ser alvo de técnicas de pré-processamento de balanceamento de dados.

Antes de implementar qualquer tipo de técnica é necessário criar os conjuntos de treino e teste a partir do conjunto de dados original, dado que os modelos vão ser treinados em modelos que usam os conjuntos de dados com *under* e *oversampling,* mas testados no conjunto de teste original.

Para a criação dos conjuntos de treino e teste é utilizada a classe *train_test_split()* [47] da biblioteca *sklearn.model_selection* sobre um conjunto de dados previamente estratificado com o uso da classe *StratifiedKFold()* [48] com o valor "10" para o número de divisões (n_*splits*) e posteriormente concatenado, para garantir uma dispersão equilibrada dos dados. São criados assim os conjuntos de dados*:* "*original_Xtrain*", "*original_Xtest*", "*original_ytrain*", "*original_ytest*".

```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
Train: [ 11841  11880  12070 ... 284804 284805 284806] Test: [    0     1     2 ... 28522 28523 28524]
Train: [     0      1      2 ... 284804 284805 284806] Test: [11841 11880 12070 ... 57018 57019 57020]
Train: [     0      1      2 ... 284804 284805 284806] Test: [30496 31002 33276 ... 85497 85498 85499]
Train: [     0      1      2 ... 284804 284805 284806] Test: [ 50537  52466  52521 ... 113966 113967 113968]
Train: [     0      1      2 ... 284804 284805 284806] Test: [ 81609  82400  83053 ... 142427 142428 142429]
Train: [     0      1      2 ... 284804 284805 284806] Test: [120505 120837 122479 ... 170948 170949 170950]
Train: [     0      1      2 ... 284804 284805 284806] Test: [150654 150660 150661 ... 199403 199404 199405]
Train: [     0      1      2 ... 284804 284805 284806] Test: [154697 154718 154719 ... 227867 227868 227869]
Train: [     0      1      2 ... 284804 284805 284806] Test: [212516 212644 213092 ... 256351 256352 256353]
Train: [     0      1      2 ... 256351 256352 256353] Test: [243749 243848 244004 ... 284804 284805 284806]
--------------------------------------------------------------------------------------------------------
Label Distributions:

[0.99827174 0.00172826]
[0.99827949 0.00172051]
```

Figura 16: Distribuição dos dados após técnica de estratificação e divisão em treino e teste

Para proceder a uma análise mais completa, necessitamos de uma amostra dos dados com classes equilibradas, apresentando um número igual de transferências fraudulentas e não fraudulentas. Isto permite evitar prognósticos tendenciosos dos algoritmos: *Overfitting*, onde os classificadores assumem a maioria de casos como não fraudulentos; Correlações erradas, tendo o conjunto de dados desbalanceados não é possível aferir a verdadeira correlação dos atributos das classes e a sua influência na classificação. É usada a técnica de redução aleatória de amostras:

- Sabendo que número de registos de transações fraudulentas são 492, passamos à recolha de um igual número de registos de transações não fraudulentas, não sem antes proceder a uma reordenação aleatória dos dados.
- Procedemos à concatenação dos dois conjuntos de dados, fraudulentos e não fraudulentos, obtendo assim um novo conjunto com 984 registos com representação igual das duas classes.

Queremos saber através da matriz de correlação que atributos influenciam fortemente uma transação como fraudulenta. Para traçar a matriz de correlação foi aplicado ao conjunto de dados a função *corr()*[49]. Nas seguintes imagens pode-se ver a diferença no resultado visual da matriz de correlação nos dados antes e depois de equilibrados.

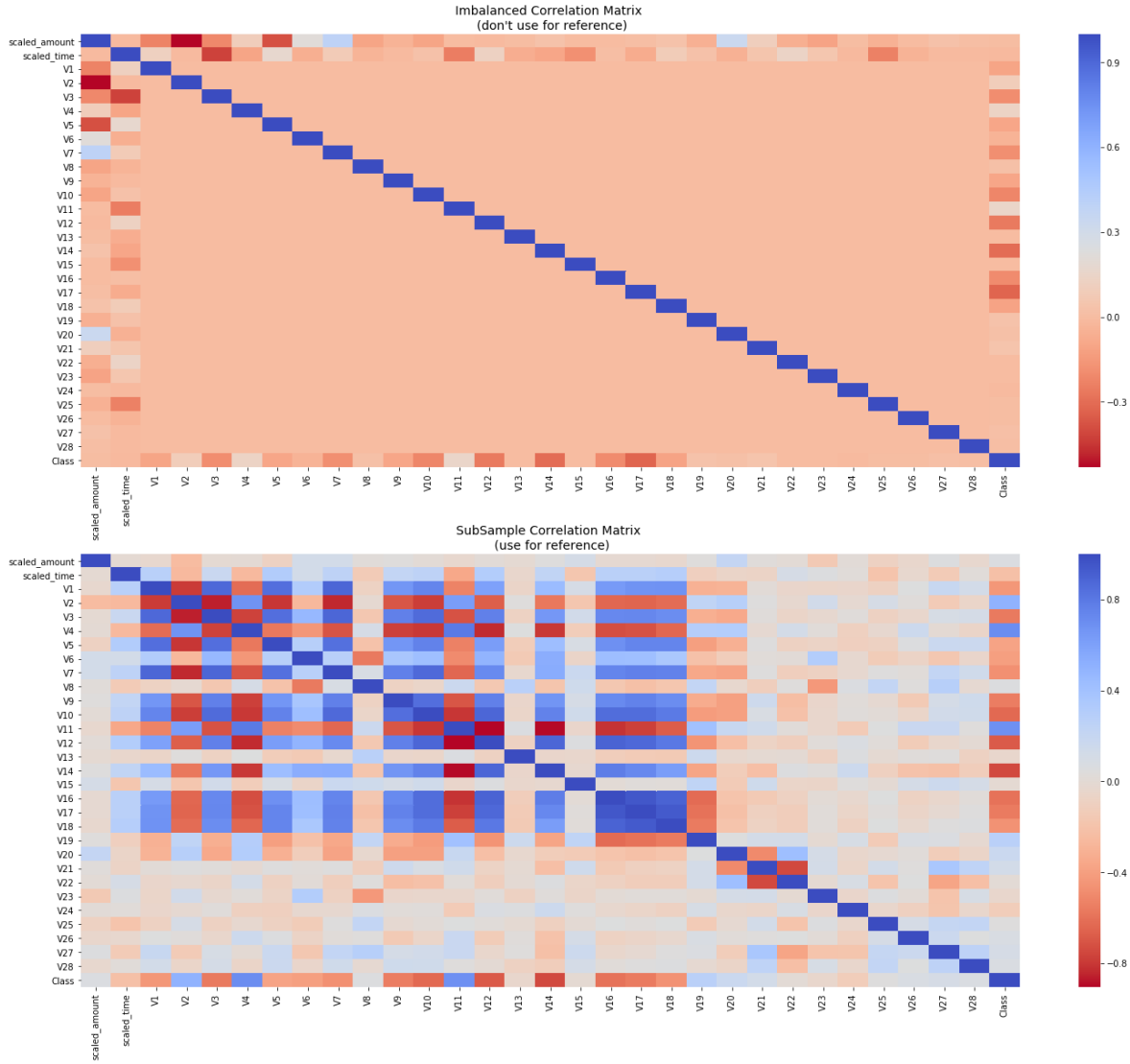


Figura 17: matrizes de correlação dados desbalanceados e equilibrados

Como se pode observar, na matriz de correlação dos dados equilibrados, V10, V12, V14 e V17 são negativamente correlacionados com a classe, o que significa que quanto menor forem estes valores, maior a probabilidade de a transação ser fraudulenta. Por outro lado, os atributos V2, V4, V11 e V19 são positivamente correlacionados com a classe, significando que quanto maiores forem estes valores, maior a probabilidade da transação ser fraudulenta.

Usando os *BoxPlots* podemos também visualizar a distribuição destes atributos pelas duas classes e grandeza de influência em cada classe (fig. 18 e fig. 19). Podemos constatar que o atributo V17 tem mais peso numa correlação negativa com a classe e o atributo V11 o atributo com mais peso numa correlação positiva com a classe.

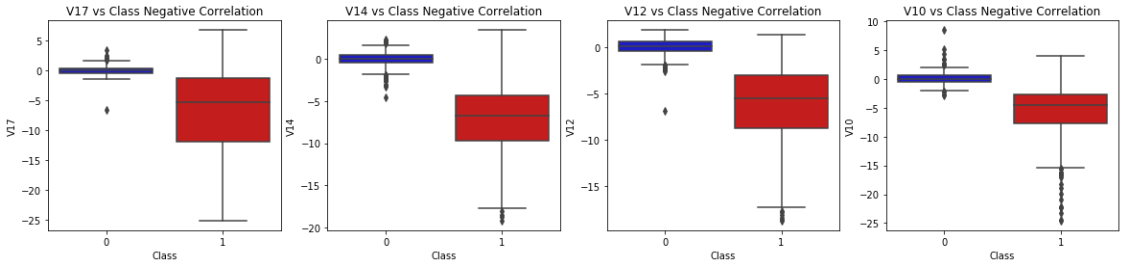


Figura 18: *BoxPlots* dos atributos com correlação negativa.

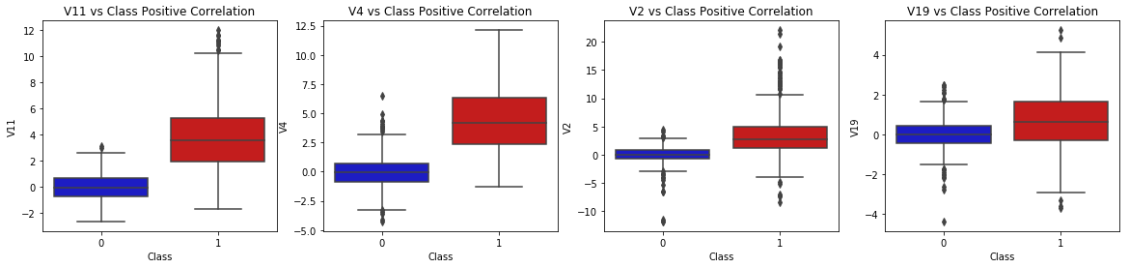


Figura 19: *BoxPlots* dos atributos com correlação positiva.

4.2.3.1 Aplicação de técnicas para dados desbalanceados

Nesta fase pretende-se criar duas amostras do conjunto de dados original contendo um rácio de 50/50 de transferências dos dois tipos (fraudulentas e não fraudulentas), aplicando para tal duas técnicas diferenciadas de *under* e *oversampling*. Sabendo que existem 492 casos de transações fraudulentas, teremos de obter novos conjuntos de dados que contenham também igual número de transa*ções* não-fraudulentas*.*

A técnica de *undersampling* é aplicada à classe maioritária, de onde se retira uma amostra dos dados em igual número à classe minoritária. Pelo contrário na técnica de *oversampling* a estratégia passa por extrapolar a classe minoritária para obter um número de registos idêntico à classe maioritária.

Como é de esperar as técnicas de *oversampling* acarretam sempre um aumento do volume de dados a processar, tornado o processo de avaliação de classificadores mais moroso e com necessidade de mais capacidade de processamento.

Neste trabalho pretende-se testar a influência destas técnicas na eficiência dos diferentes classificadores. Existem diferentes técnicas de *sampling*, para este projeto e foram selecionadas algumas com base na sua diferenciação e as mais referenciadas nos trabalhos estudados, *ClusterCentroids*() para *undersampling* e *SMOTE*() para *oversampling.*

4.2.4 Treino dos classificadores

Para a escolha do melhor modelo na classificação de transações fraudulentas ou não fraudulentas, é proposto o seguinte *pipeline*:

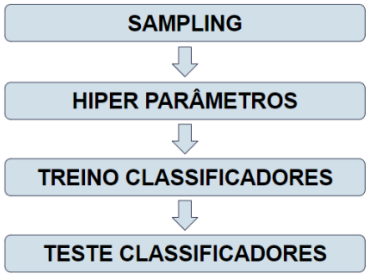


Figura 20: *Pipeline* Treino e Teste de Classificadores

4.2.4.1 Sampling

Pretende-se criar dois conjuntos de dados resultantes de cada uma das técnicas de *oversampling* e *undersampling*. Para a técnica de undersampling ao conjunto de dados inicial foi aplicada a função *ClusterCentroids*(*sampling_strategy='majority'*)[50], importada da biblioteca *imblearn.under_sampling,* com o parâmetro a forçar a aplicação à classe maioritária (transações não fraudulentas), resultando um conjunto de dados com 984 registos igualmente distribuídos pelas duas classes (0: 492, 1: 492). Para oversampling foi usada a função *SMOTE(sampling_strategy='minority')* [51], importada da biblioteca *imblearn.over_sampling*, com o parâmetro a forçar a aplicação à classe minoritária (transações fraudulentas), resultando um conjunto de dados com 568 630 registos igualmente distribuídos pelas duas classes (0: 284315, 1: 284315).

4.2.4.2 Híper-parâmetros

Para cada classificador existem vários parâmetros que devem ser afinados para um melhor resultado. Para aferir quais os melhores parâmetros para cada classificador para os diferentes cenários, é utilizada a função *GridSearchCV()*[52] da biblioteca *sklearn.model_selection*, que permite a execução de uma bateria de testes cruzando os valores dos diferentes parâmetros para intervalos pré-definidos (fig. 21).

```
# LOGISTIC REGRESSION (Undersample)
# Grid Search cross validation
import numpy as np # linear algebra
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,grid,cv=10)
logreg_cv.fit(X_undersampled,y_undersampled)

print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)

```

Figura 21: implementação de GridSearchCV(), exemplo

Como se pode ver no código apresentado, no exemplo para o classificador de regressão logística, são definidos os intervalos de valor dos parâmetros "C" e "*penalty*" como variável "*grid*", é definido também o classificador e a composição do *GridSearch* definindo o parâmetro de *cross-validation* com o valor de 10. Por fim é aplicado ao conjunto de dados e devolve o resultado. A seguinte tabela apresenta os valores e resultados dos diferentes testes.

Classificador	Intervalo de teste	Sampling	Híper-parâmetros	Accuracy	t
Regressão Logistica	C: 0.001, 0.01, 0.1, 1, 10, 100, 1000 penalty:l1,l2	Over	C: 1000.0 penalty:l2	0.945	2477.44s
		Under	C: 0.01, penalty: l2	0.932	2.19s
Máquina de Vetores	C:1 gamma: 1, 0.1, 0.001, 0.0001 kernel:rbf	Over	C: 1 gamma: 0.1 kernel: rbf	0.982	14896s
		Under	C: 1 gamma: 0.001 kernel: rbf	0.903	1.73s
Florestas Aleatórias	n_estimators: [200, 500] max_features: auto, sqrt, log2 max_depth: [4,5,6,7,8] criterion:gini, entropy	Over	criterion: gini max_depth: 4 max_features: log2, n_estimators: 200	0.867	2667.52
		Under	criterion: gini max_dept: 8 max_features: log2 n_estimators: 200	0.929	67.61s
XGBoost	colsample_bytree: uniform(0.7, 0.3), gamma: uniform(0, 0.5), learning_rate: uniform(0.03, 0.3), max_depth: randint(2, 6), n_estimators: randint(100, 150), "subsample": uniform(0.6, 0.4)	Over	-	-	-
		Under	colsample_bytree: 0.711, gamma: 0.396, learning_rate: 0.116, max_depth: 5, n_estimators: 130, subsample: 0.899	0.946	80.57s

Tabela 9: Resultados do *GridSearch,* híper-parâmetros, t - tempo de processamento.

De salientar que para o classificador XGBoost com a técnica de *oversampling,* o tempo de computação para descobrir os híper-parâmetros, excedeu a janela de 12h disponibilizada gratuitamente no serviço *google colab*, pelo que a opção foi usar os mesmos híper-parâmetros encontrados para a técnica de *undersampling* no treino do classificador.

4.2.4.3 Treino de classificadores

Cada classificador é treinado usando à vez uma técnica de sampling (under e oversampling) e com os híper-parâmetros respetivos, encontrados anteriormente para cada conjunto de dados.

```
undersample_pipeline_lr = imbalanced_make_pipeline(ClusterCentroids(sampling_strategy='majority'),
                                                   LogisticRegression(C=0.01,penalty="l2"))
undersample_model_lr = undersample_pipeline_lr.fit(X.values[train], y.values[train])
undersample_prediction_lr = undersample_model_lr.predict(X.values[test])
undersample_prob_auc_lr = undersample_model_lr.predict_proba(X.values[test])
```

Figura 22: Treino de classificador, *undersampling*, exemplo regressão logística.

No exemplo da figura 22, são definidas no mesmo processo a técnica de sampling e o classificador, sendo assim executadas durante a validação cruzada, evitando assim o potencial *overfitting* nos dados. No decurso do processo é feito o treino do modelo com o conjunto de treino e finalmente são retiradas as previsões.

4.2.4.4 Teste de classificadores

Para obter a métrica *AUC-ROC* de avaliação do modelo, é usada a função *roc_curve()*[53] da biblioteca *sklearn.metrics*. Como parâmetros desta função são definidos o conjunto de teste e a previsão resultante da fase de treino.

4.2.5 Avaliação dos modelos

4.2.5.1 Descrição do problema

Pretende-se otimizar um sistema de deteção de transações fraudulentas pelo estudo da eficiência de vários modelos compostos por técnicas de pré-processamento e algoritmos de classificação.

4.2.5.2 Definição dos objetivos

Avaliar o impacto que a escolha da técnica de pré-processamento tem sobre a eficiência dos classificadores usados nos diferentes modelos de deteção de fraude. Estudo de sistema de deteção de transações fraudulentas em tempo real usando software de distribuição livre.

4.2.5.3 Especificação da hipótese

A escolha da técnica de pré-processamento de dados desbalanceados influência a eficiência do modelo de deteção de fraude.

4.2.5.4 Identificação dos indicadores e fontes de informação

No conjunto de dados utilizado para treino dos modelos as transações estão classificadas com o atributo binário (1 – fraudulentas, 0 – não fraudulentas). Na fase de teste do modelo as

transações que são classificadas corretamente pertencentes à classe 1 são apelidadas de Verdadeiras Positivos, enquanto as classificadas corretamente como pertencentes à classe 0 são apelidadas de Verdadeiras Negativas. Temos ainda informação relativa a erros de classificação dos modelos, as transações consideradas Falso Positivas, quando a transações é prevista como positiva, mas está classificada originalmente como pertencente à classe 0 e as transações Falsas Negativas que ocorre quando na previsão são classificadas como pertencentes à classe 0, sendo na verdade da classe 1. Para melhor visualização desta informação, os valores são colocados numa matriz de confusão idêntica ao esquema da figura 23. Estes dois tipos de erro podem ter várias implicações, no que diz respeito à deteção de fraude, um erro de Falsa Positiva pode corresponder a uma transação legitima ser considerada fraudulenta. Um erro de Falsa Negativa significa que uma transação fraudulenta seja considerada como legitima.

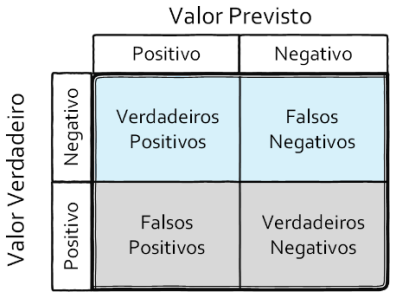


Figura 23: Modelo de matriz de confusão.

4.2.5.5 Descrição da metodologia de avaliação

Do cruzamento dos classificadores escolhidos (regressão logística, máquinas de vetores, florestas aleatórias e *XGBoost*) em conjunto com as técnicas de pré-processamento (*undersampling*, *oversampling*) resultam os modelos a avaliar para identificar qual o mais eficiente a fazer previsões, apresentados na seguinte tabela:

Modelo	Técnica P.P.	Classificador
1	Undersampling	Florestas Aleatórias
2	Oversampling	
3	Undersampling	Regressão Logística
4	Oversampling	
5	Undersampling	Máquinas de Vetores
6	Oversampling	
7	Undersampling	*XGBoost*
8	Oversampling	

Tabela 10: Modelos a analisar

4.2.5.6 Medida de avaliação

AUC - ROC (*Area Under the Curve - Receiver Operating Characteristics*), é uma medida de performance para modelos de classificação. ROC é uma curva de probabilidade e AUC representa o grau de separação de classes, indicando assim a capacidade de o modelo distinguir as várias classes. Quanto maior for a AUC, melhor é o modelo a prever os 0s como 0s e os 1s como 1s. No caso em estudo, quanto maior o AUC, melhor é o modelo a distinguir entre transações fraudulentas das não fraudulentas [37].

A curva ROC é traçada com a relação entre TPR (*True Positive Rate*), o rácio de verdadeiros positivos no eixo dos ys e FPR (*False Positive Rate*), o rácio de falsos positivos no eixo dos xs (fig. 24).

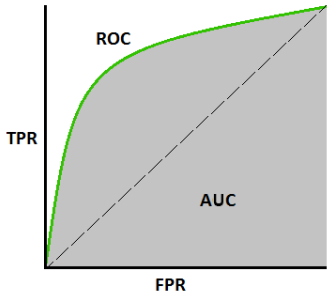


Figura 24: Curva ROC, fonte: Narkhede, towardsdatascience.com (2018)

Um modelo excelente apresenta um valor de AUC perto de 1, traduzindo a sua capacidade de separação de classes. Um modelo pobre apresenta valores de AUC perto de 0, significa que o modelo prevê os 0s como 1s e vice-versa. Um modelo incapaz de identificar a separação de classes apresenta valor de AUC de 0.5.

4.2.5.7 Método de avaliação

A partir do conjunto de dados inicial, foram criados dois conjuntos resultantes das técnicas de *sampling,* que serviram para treino e teste dos modelos de classificação, para comparar a prestação de cada modelo com cada uma das técnicas.

Os algoritmos de classificação usados em cada modelo apresentam parâmetros que podem ser customizados para melhorar a precisão dos resultados. Foi utilizada a técnica de *Grid Search* para encontrar os híper-parâmetros que resultam na melhor prestação para cada classificador.

Para validação dos modelos é aplicado o método *cross-validation* [38]. Este método consiste em dividir o conjunto de dados em vários conjuntos de treino e um conjunto de teste, usando o conjunto de treino para treinar o modelo e o conjunto de teste para avaliar quão bem o modelo generaliza para dados que ainda não viu. Nesta abordagem, chamada *k-fold cross-validation,* o conjunto de dados é dividido em k partes. O modelo é treinado com k-1 partes e validado com a parte que fica de fora. A técnica de *k-fold croos-validation* evita o efeito de *overfitting* no conjunto de dados. A medida de performance é a média dos valores aferidos em cada iteração (fig. 25).

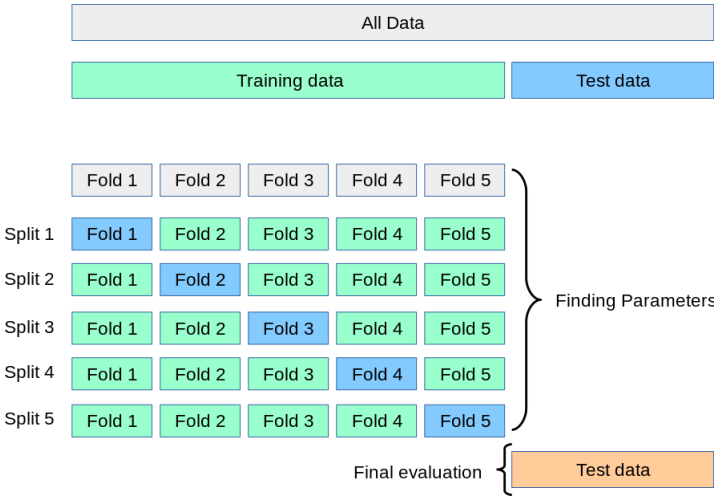


Figura 25: *k-fold cross-validation* [38]

Um dos erros comuns relatados nas publicações [39], prende-se com o facto de quando se aplicam técnicas de *sampling* estas não devem ser executadas antes de efetuar *cross-validation*. Esta prática errada influência diretamente a qualidade do *set* de validação (fig .26).

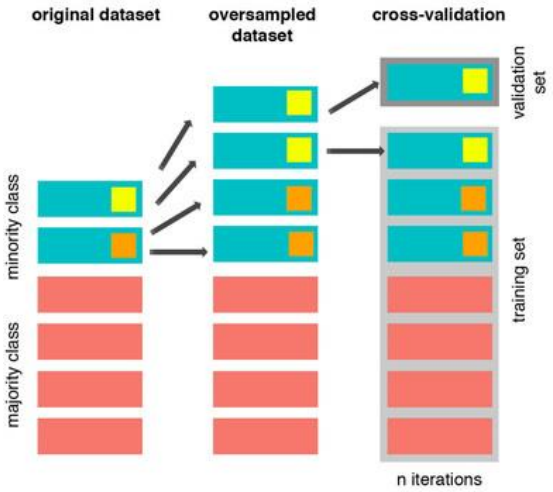


Figura 26: *Sampling* antes de *cross-validation*

A forma correta de execução deverá comportar a técnica de *sampling* (*over* e *undersampling*) durante o processo de *cross-validation*. Assim os dados criados pelos processos de *sampling* não afetam o set de validação e são criados só para o *set* de treino. (fig. 27)

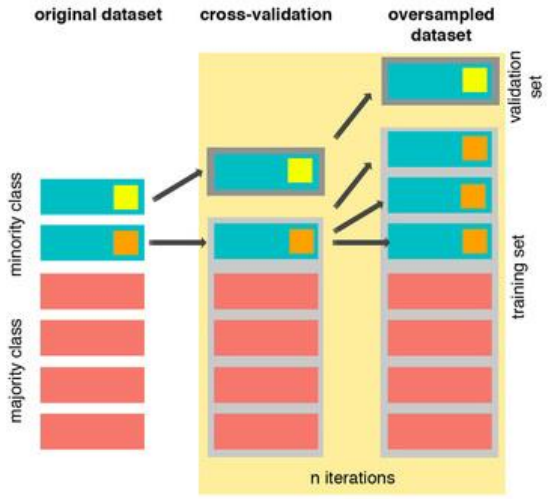


Figura 27: *Sampling* durante *cross-validation*

4.2.5.8 Resultados

Após o treino dos vários modelos de classificação procedeu-se à computação das métricas de avaliação sobre o conjunto de teste, resultando os seguintes valores para a curva ROC. Para análise mais aprofundada da eficiência dos modelos, são também analisadas as matrizes de confusão.

$$\begin{bmatrix} VP & FN \\ FP & VN \end{bmatrix}$$

VP = verdadeiros positivos, FP = falsos positivos,
FN = falsos negativos, VN = verdadeiros positivos

Modelo	Sampling	Classificador	AUC-ROC	Matriz confusão
1	Over	Regressão Logística	0.927	$\begin{bmatrix} 146 & 18 \\ 2713 & 92059 \end{bmatrix}$
2	Under		0.887	$\begin{bmatrix} 149 & 15 \\ 15509 & 79262 \end{bmatrix}$
3	Over	Máquinas de Vetores	0.509	$\begin{bmatrix} 16 & 148 \\ 369 & 94403 \end{bmatrix}$
4	Under		0.902	$\begin{bmatrix} 144 & 20 \\ 876 & 93896 \end{bmatrix}$
5	Over	Florestas Aleatórias	0.899	$\begin{bmatrix} 151 & 13 \\ 1188 & 93584 \end{bmatrix}$
6	Under		0.876	$\begin{bmatrix} 142 & 22 \\ 7166 & 87605 \end{bmatrix}$
7	Over	*XGBoost*	0.652	$\begin{bmatrix} 99 & 63 \\ 28225 & 66546 \end{bmatrix}$
8	Under		0.705	$\begin{bmatrix} 147 & 17 \\ 30588 & 64184 \end{bmatrix}$

Tabela 11: Tabela de resultados avaliação dos modelos

Dos modelos testados, o modelo 4 (Máquina de vetores com *undersampling)* e o modelo 5 (Florestas aleatórias com *oversampling*), são os que apresentam melhores prestações quando

analisados os resultados nas respetivas matrizes de confusão. Ambos os modelos apresentam valores muito reduzidos para FN (falsos negativos) não ultrapassando as duas dezenas. Estes modelos também são os que apresentam um baixo valor para os FP (falsos positivos). No problema em estudo, deteção de transações fraudulentas, é importante obter uma taxa de FN baixa, pois este indicador representa as transações realmente fraudulentas que foram erradamente classificadas como legais. O outro indicador importante é o FP, neste caso transações legais que são erradamente classificadas como fraudulentas, o que implica a recusa errada de transações aos clientes, com um elevado custo não só na reputação da empresa como de eventual gasto de tempo em verificações.

Apesar destes dois modelos (4 e 5) apresentarem prestações muito idênticas, devemos ter em atenção que usam técnicas de sampling diferentes, o que implica tempos de treino e teste também diferentes. No caso do modelo 4, máquina de vetores com *undersampling,* o processo de treino e teste do modelo é executado com um conjunto de dados menor o que se traduz num processo mais rápido em relação a outros modelos que usem técnicas de *oversampling*, como é o caso do modelo 5, sem ganho significativo de prestação.

Pelos estudos efetuados não se pode afirmar, de forma geral, qual é a melhor técnica de *sampling*. Se para os classificadores Regressão Logística e Florestas Aleatórias verificam-se melhorias com *oversampling*, o contrário acontece para Máquinas de Vetores e *XGBoost*.

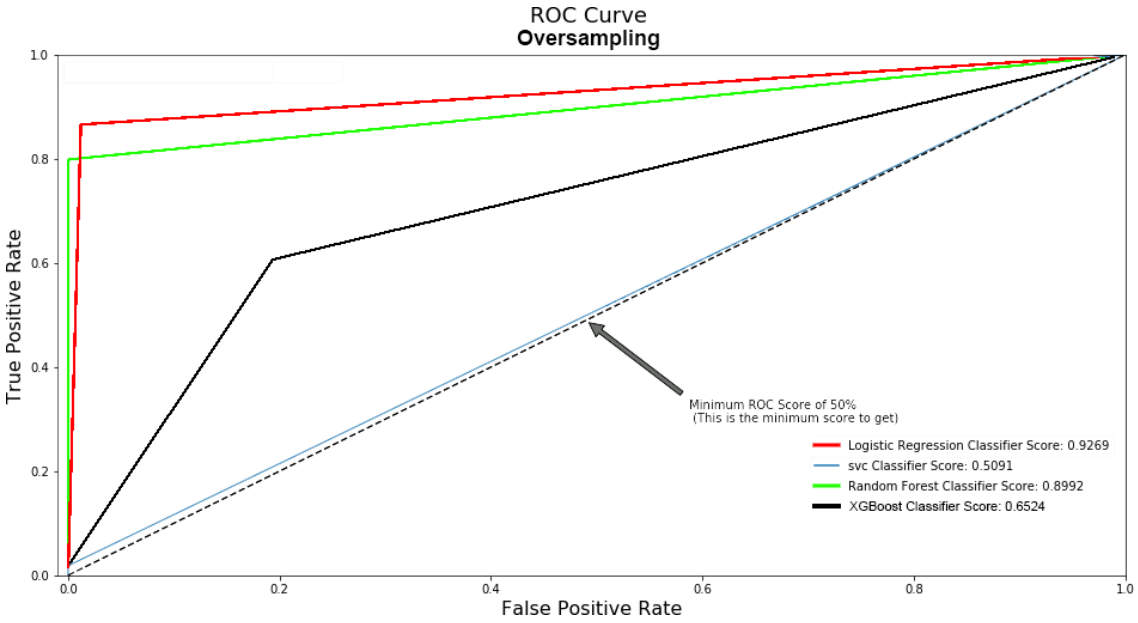


Figura 28: Curvas ROC *oversampling* resultados

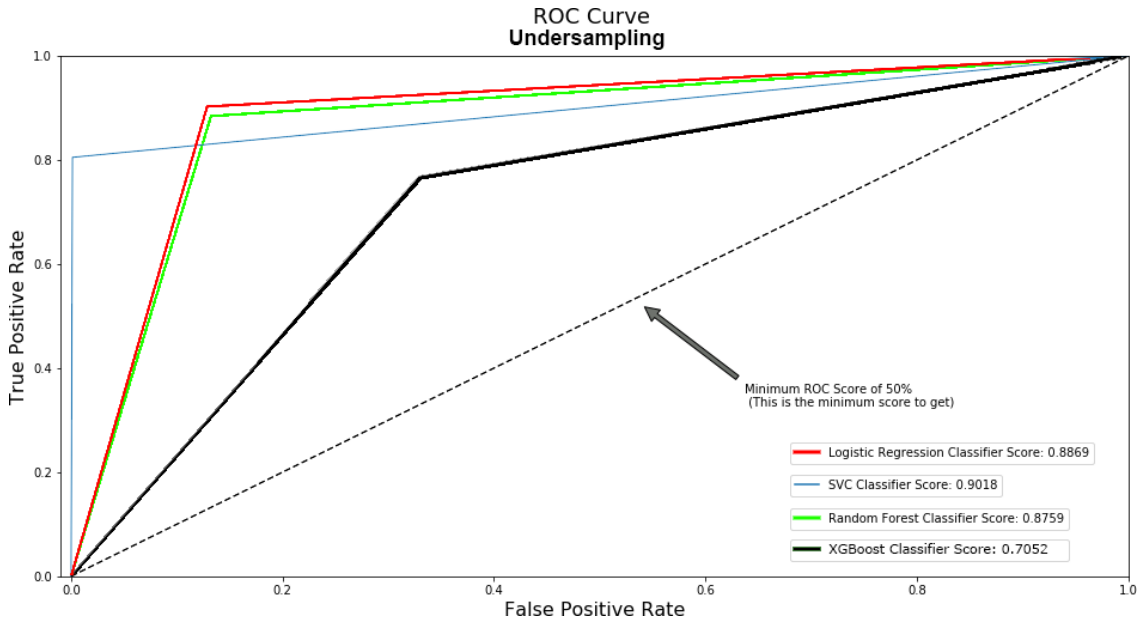


Figura 29: Curvas ROC *undersampling* resultados

4.3 Fase de implementação

Na fase de implementação foi é proposto um sistema de deteção em tempo real com o modelo eleito composto pelo classificador máquina de vetores com técnica de *undersampling*. Desenvolvido na plataforma *Google Cloud Platform (GCP)* [8], que disponibiliza módulos de *ML* que permitem a implementação dos modelos e módulos de tratamento de dados que servem para alimentar os modelos em *streaming* e persistir os resultados para consulta posterior.

Após criado um projeto na consola do *GCP* foi criado um "*bucket"* na área "Storage" que irá conter o conjunto de dados inicial e o conjunto de dados de saída com os resultados das previsões. Este "*bucket*" é onde são colocados também os ficheiros de configuração dos vários módulos e do algoritmo de classificação.

É ativado e configurado o módulo "*DataFlow*" que vai efetuar a gestão dos dados em *stream,* alimentando o sistema em tempo real. São configurados no módulo a diretoria de "*output*", o ficheiro de configuração (preprocess.py), o *project_id* e o *bucket_id.*

Como o estudo do melhor modelo já foi efetuado anteriormente através do *google colab*, não é necessário implementar todo o sistema de testes de classificadores, híper-parâmetros, sampling, treino e teste de modelos. Só há a necessidade de carregar o modelo para o sistema para ser usado. Assim após ativada a *API* de *Cloud Machine Learnig (ML-engine)* é importado o modelo (fig. 30).

```
MODEL_NAME=fraud_detection
MODEL_VERSION=v_$(date +"%Y%m%d_%H%M%S")
TRIAL_NUMBER=1
MODEL_SAVED_NAME=$(gsutil ls ${TRAINING_OUTPUT_DIR}/trials/${TRIAL_NUMBER}/export/exporter/ | tail -1)
gcloud ml-engine models create $MODEL_NAME \
--regions us-central1
gcloud ml-engine versions create $MODEL_VERSION \
--model $MODEL_NAME \
--origin $MODEL_SAVED_NAME \
--runtime-version 1.5
```

Figura 30: importação do modelo para o *Cloud* ML

Podem ser guardados diferentes versões do mesmo modelo, o "*ML-engine"* recebe um nome para o modelo e um nome único para a versão corrente criada.

As previsões são efetuadas pelo *ML-engine* alimentado pelos dados fornecidos pelo módulo "DataFlow" anteriormente configurado, com recurso ao modelo definido e carregado. Os resultados são persistidos na diretoria definida. (fig. 31).

```
JOB_NAME=${MODEL_NAME}_$(date +"%Y%m%d_%H%M%S")
FEATURES_INPUT_PATH=gs://${BUCKET_ID}/${DATAFLOW_OUTPUT_DIR}split_data/split_data_TEST_features.txt*
PREDICTIONS_OUTPUT_PATH=gs://${BUCKET_ID}/predictions/$JOB_NAME
gcloud ml-engine jobs submit prediction $JOB_NAME \
--model $MODEL_NAME \
--input-paths $FEATURES_INPUT_PATH \
--output-path $PREDICTIONS_OUTPUT_PATH \
--region us-central1 \
--data-format TEXT \
--version $MODEL_VERSION
```

Figura 31: Configuração da execução do modelo no ML-engine

5 Conclusão

Em *Machine Learning* é necessário testar o máximo de algoritmos possíveis para um determinado conjunto de dados para obter o que apresenta melhores resultados. No entanto a escolha do algoritmo não é suficiente, há que escolher também as melhores configurações dos seus parâmetros (híper-parâmetros). Outros indicadores devem também ser ponderados, como a complexidade dos algoritmos e respetivo tempo computacional. Devemos ter presentes os fatores que podem influenciar a precisão do algoritmo, como o tamanho do conjunto de dados a analisar, este não deverá ser muito grande para permitir obter resultados em tempo útil, no entanto deve conter informação suficiente para garantir a qualidade dos resultados. A distribuição dos dados pelas classes também influencia a prestação de um algoritmo, como é comprovado neste estudo a técnica de balanceamento (*sampling*) tem influência direta no algoritmo, não prevalecendo, de um modo transversal, uma técnica sobre a outra, mas sim caso a caso.

Mesmo após a identificação do melhor algoritmo para o cenário em estudo, o processo não está acabado. Os ecossistemas onde os modelos de *Machine Learning* são utilizados, estão em constante atualização e mudança, seja por coleção de novos dados, seja por alterações de comportamentos, um algoritmo pode deixar de ser o mais indicado passado algum tempo. Urge definir uma estratégia de atualização e novo estudo de cenário, que pode passar por revisitar os algoritmos e estratégias utilizadas ou incorporar novos algoritmos.

Neste trabalho apenas foram estudados algoritmos de aprendizagem supervisionada que utilizam regressão e classificação nas suas previsões e um histórico de dados já classificados. Como trabalho futuro é proposto o estudo de algoritmos que usem técnicas aplicadas a aprendizagem não supervisionada como *clustering*, *anomaly detection e* redes neuronais. Estas técnicas são úteis na análise de novos dados, pois podem identificam novos grupos e comportamentos desviantes.

Referências

[1] J. T. Quah e M. Sriganesh, «Real-time credit card fraud detection using computational intelligence, Expert Syst. Appl 35 (4) 1721-1732 http://linkinghub.elsevier.com/retrieve/pii/S0957417407003995. http://dx.doi.org/10.1016/j.eswa.2007.08.093». 2008.

[2] F. Carcillo, A. Dal Pozzolo, Y. A. Caelen, Y. Mazzer, e G. Bontempi, «SCARFF: A scalable framework for streaming credit card fraud detection with spark. Information fusion, 41», em *Decision support systems*, 2018, pp. 182–194.

[3] C. Phua, V. Lee, K. Smith, e R. Gayler, «A comprehensive survey of data miningbased accounting-fraud detection research, Int. Conf. Intell. Comput. Technol. Autom. 1», pp. 50–53, 2010.

[4] N. Carneiro, G. Figueira, e M. Costa, «A data mining based system for credit-card fraud detection in e-tail», *Decis. Support Syst.*, vol. 95, pp. 91–101, Mar. 2017.

[5] «Card Fraud Worldwide, annual increases/decreases in fraud vs. total volume, November», *The Nilson Report*, pp. 8–11, 2018.

[6] Albrecht, W. Steve, C. Albrecht, e C. C. Albrecht, «Current Trends in Fraud and its Detection. Information Systems Security 17.1», pp. 2–12, 2008.

[7] «Credit Card Fraud Detection | Kaggle». [Em linha]. Disponível em: https://www.kaggle.com/mlg-ulb/creditcardfraud. [Acedido: 19-Fev-2019].

[8] «Google Cloud incluindo GCP e G Suite: avaliação gratuita», *Google Cloud*. [Em linha]. Disponível em: https://cloud.google.com/?hl=pt-br. [Acedido: 03-Jul-2019].

[9] P. Koen *et al.*, «Providing Clarity and a Common Language to the "Fuzzy Front End".Research Technology Management; 44, 2», pp. 46–55, 2001.

[10] P. Koen *et al.*, «Fuzzy-Front End: Effective Methods, Tools, and Techniques. In PDMA toolbook 1 for new product development, Belliveau P, Griffin A and SomermeyerS (Eds.), John Wiley & Sons», pp. 2–35, 202AD.

[11] T. L. Saaty, «The analytic hierarchy process: planning, priorisetting, resource allocation.» 1980.

[12] L. von Mises, *The Ultimate Foundation of Economic Science.* Indianapolis: Liberty Fund, 2012.

[13] S. Nicola, P. F. Eduarda, e P. F. J.J., «A novel framework for modeling value for the customer, an essay on negotiation.», *Int. J. Inf. Technol. Decis. Mak.*, vol. 11, n. 03, pp. 661–703, 2012.

[14] T. Wooddall, «Conceptualising "Value for the Customer": An Attributional, Structural and Dispositional Analysis.», *Acad. Mark. Sci. Rev. January*, p. 2, 2003.

[15] W. Ulaga e A. Eggert, «Value-Based Differentiation in Business Relationships: Gaining and Sustaining Key Supplier Status.», *J. Mark.*, vol. 70, n. 1, pp. 119–136, 2006.

[16] A. Lindgreen e F. Wynstra, «Value in business markets: What do we know? Where are we going?», *Ind. Mark. Manag.*, n. 34, pp. 732–748, 2005.

[17] A. Osterwalder e Y. Pigneur, *Business Model Generation*. Hoboken NJ: John Wiley and Sons, 2010.

[18] G. F. M. C. N. Carneiro, «Decision Support Systems», *elsevier*, n. 17 fev, p. 11, 2017.

[19] C. Cortes, D. Pregibon, e C. Volinsky, *Communities of Interest.* Springer, 2001.

[20] Y. Moreau, E. Lerouge, H. Verrelst, P. Stormann, P. Burge, e K. U. Leuven, «A hybrid system for fraud detection in mobile communications.», *Neural Netw*, vol. April, pp. 447–454, 1999.

[21] E. Duman e M. H. Ozcelik, «Detecting credit card fraud by genetic algorithm and scatter search.», *Expert Syst*, vol. 38, n. 10, pp. 13057–13063, 2011.
[22] K. Ramakalyani e D. Umadevi, «Fraud Detection of Credit Card Payment System by Genetic Algorithm», *Int J Sci Eng Res*, vol. 3, n. 7, pp. 1–6, 2012.
[23] K. Srivastava, A. Kundu, S. Sural, e S. Member, «Credit card fraud detection using hidden Markov mode», *IEEE Trans Dependable Secure Comput*, vol. 5, n. 1, pp. 37–48, 2008.
[24] C. Whitrow, D. J. Hand, P. Juszczak, D. Weston, e N. M. Adams, «Transaction aggregation as a strategy for credit card fraud detection.», *Data Min Knowl Disc*, vol. 18, n. 1, pp. 30–55, 2009.
[25] S. Bhattacharyya, S. Jha, K. Tharakunnel, e J. C. Westland, «Data mining for credit card fraud: a comparative study.», *Decis Support Syst*, vol. 50, n. 3, pp. 602–613, 2011.
[26] A. Dal Pozzolo, O. Caelen, Y. A. Le Borgne, S. Waterschoot, e G. Bontempi, «Learned lessons in credit card fraud detection from a practitioner perspective.», *Expert Syst Appl*, vol. 41, n. 10, pp. 4915–4928, 2014.
[27] C. G. Tianqi Chen, «XGBoost: A Scalable Tree Boosting System». University of Washington, Jun-2016.
[28] A. Nagpal, «L1 and L2 Regularization Methods», *Medium*, 14-Out-2017. [Em linha]. Disponível em: https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c. [Acedido: 08-Out-2019].
[29] K. Gupta, «The Best Programming Languages for Data Mining», *Web, Design, Programming*, 07-Ago-2017. [Em linha]. Disponível em: https://www.freelancinggig.com/blog/2017/08/07/best-programming-languages-data-mining/. [Acedido: 19-Fev-2019].
[30] «R: What is R?» [Em linha]. Disponível em: https://www.r-project.org/about.html. [Acedido: 19-Fev-2019].
[31] «Welcome to Python.org», *Python.org*. [Em linha]. Disponível em: https://www.python.org/about/. [Acedido: 19-Fev-2019].
[32] «TIOBE Index | TIOBE - The Software Quality Company». [Em linha]. Disponível em: https://www.tiobe.com/tiobe-index/. [Acedido: 19-Fev-2019].
[33] R. Wirth, «CRISP-DM: Towards a standard process model for data mining.» 2000.
[34] L. Li, «Principal Component Analysis for Dimensionality Reduction», *Towards Data Science*, 24-Mai-2019. [Em linha]. Disponível em: https://towardsdatascience.com/principal-component-analysis-for-dimensionality-reduction-115a3d157bad. [Acedido: 24-Jun-2019].
[35] «Welcome to The Apache Software Foundation!» [Em linha]. Disponível em: https://www.apache.org/. [Acedido: 19-Fev-2019].
[36] «MLlib | Apache Spark». [Em linha]. Disponível em: https://spark.apache.org/mllib/. [Acedido: 19-Fev-2019].
[37] T. Fawcett, «An introduction to ROC analysis», *Pattern Recogn Lett*, vol. 27, n. 8, pp. 861–874, 2006.
[38] «3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.21.2 documentation». [Em linha]. Disponível em: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation. [Acedido: 03-Jul-2019].
[39] «credit-fraud-dealing-with-imbalanced-da-3f7521». [Em linha]. Disponível em: http://craftsfab.com/. [Acedido: 03-Jul-2019].
[40] «Credit Fraud || Dealing with Imbalanced Datasets | Kaggle». [Em linha]. Disponível em: https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets. [Acedido: 07-Out-2019].

[41] «Fraud Detection - SMOTE, XGBoost & Business Impact». [Em linha]. Disponível em: https://kaggle.com/lmorgan95/fraud-detection-smote-xgboost-business-impact. [Acedido: 08-Out-2019].
[42] «Project Jupyter». [Em linha]. Disponível em: https://www.jupyter.org. [Acedido: 24-Jun-2019].
[43] «Google Colaboratory». [Em linha]. Disponível em: https://colab.research.google.com/notebooks/welcome.ipynb. [Acedido: 03-Jul-2019].
[44] «sklearn.preprocessing.RobustScaler — scikit-learn 0.21.2 documentation». [Em linha]. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html. [Acedido: 24-Jun-2019].
[45] «Compare the effect of different scalers on data with outliers — scikit-learn 0.21.2 documentation». [Em linha]. Disponível em: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html. [Acedido: 24-Jun-2019].
[46] W. Koehrsen, «Overfitting vs. Underfitting: A Complete Example», *Towards Data Science*, 28-Jan-2018. [Em linha]. Disponível em: https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765. [Acedido: 24-Jun-2019].
[47] «sklearn.model_selection.train_test_split — scikit-learn 0.21.2 documentation». [Em linha]. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [Acedido: 24-Jun-2019].
[48] «sklearn.model_selection.StratifiedKFold — scikit-learn 0.21.2 documentation». [Em linha]. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html. [Acedido: 24-Jun-2019].
[49] «pandas.DataFrame.corr — pandas 0.25.1 documentation». [Em linha]. Disponível em: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html. [Acedido: 28-Set-2019].
[50] «imblearn.under_sampling.ClusterCentroids — imbalanced-learn 0.5.0 documentation». [Em linha]. Disponível em: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.ClusterCentroids.html. [Acedido: 03-Jul-2019].
[51] «imblearn.over_sampling.SMOTE — imbalanced-learn 0.5.0 documentation». [Em linha]. Disponível em: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html. [Acedido: 04-Jul-2019].
[52] «sklearn.model_selection.GridSearchCV — scikit-learn 0.21.3 documentation». [Em linha]. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Acedido: 29-Set-2019].
[53] «sklearn.metrics.roc_curve — scikit-learn 0.21.2 documentation». [Em linha]. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html. [Acedido: 26-Jun-2019].