Universidade do Minho
Escola de Ciências

Luísa Lopes Caldas

Deteção de fraude em Telecomunicações através de Machine Learning

outubro de 2019

Deteção de fraude em telecomunicações através de Machine Learning

Luísa Lopes Caldas

UMinho | 2019

Universidade do Minho
Escola de Ciências

Luísa Lopes Caldas

Deteção de fraude em telecomunicações através de Machine Learning

Dissertação de Mestrado
em Matemática e Computação

Trabalho efetuado sob a orientação do
Professor Doutor Luís Filipe Ribeiro Pinto
e do
Professor Doutor Stéphane Louis Clain

outubro de 2019

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Agradecimentos

Ao longo desta dissertação foram muitas as pessoas que me apoiaram. Um especial agradecimento:

- aos Professores Luís Pinto e Stéphane Clain por toda a disponibilidade, apoio e paciência durante a realização desta dissertação;
- à WeDo technologies pela disponibilidade e apoio na realização desta dissertação em ambiente empresarial;
- aos meus pais e irmã por todo o apoio, paciência e disponibilidade que sempre demonstraram;
- ao meu namorado por toda a ajuda no processo de concretização desta dissertação;
- e a todos os meus amigos que sempre se mostraram disponíveis.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

A fraude nas telecomunicações é um problema que tem originado elevadas perdas financeiras por todo o mundo. As operadoras procuram combater a fraude de forma a não perderem tanto dinheiro. No entanto, sempre que é descoberta uma maneira de combater a fraude, os *fraudsters* descobrem novas formas de a conseguir realizar. Nos últimos tempos, com o fim do *roaming* europeu, houve grandes mudanças nas telecomunicações europeias, o que levou a um crescimento da fraude na Europa. O objetivo desta dissertação foi a procura de novas formas de combater a fraude em telecomunicações, com recurso a técnicas de *machine learning*.

Em concreto, esta dissertação debruçou-se sobre uma base de dados de uma operadora, com informação sobre chamadas recebidas numa *gateway*, tendo por objetivo a identificação de fraudes do tipo *bypass* e *wangiri*. Em primeiro lugar, foi desenvolvida uma análise exploratória com base em análises estatísticas, para melhor conhecimento dos dados, tendo sido criados novos atributos para ajudarem os modelos. Um atributo que teve um papel fundamental nesta dissertação foi a *Range*, que se baseia no agrupamento de números telefónicos, tendo em conta a variação dos últimos dígitos dos números. Posteriormente, foram desenvolvidos modelos de *machine learning* sem supervisão: PCA, *autoencoder* e LSTM *autoencoder*.

Uma das conclusões deste trabalho é a de que os bons resultados produzidos pelo modelo PCA, sugerem que a não fraude possa ser um problema linear, apesar de produzir uma percentagem elevada de *outliers*. Os modelos de *autoencoder* por si só não produziram tão bons resultados, mas após aplicação de filtros baseados em *scores* (de forma a tentar quantificar a não linearidade dos dados), observou-se uma acentuada melhoria nos resultados. Os resultados preliminares obtidos com os modelos LSTM *autoencoders* sugerem que a sua capacidade de guardar dados em memória pode vir a produzir muito bons resultados.

Palvras-chave: Fraude em Telecomunicações, Deteção de *Outliers*, *Machine Learning*, PCA, *Autoencoder*, LSTM *Autoencoder*

Abstract

Telecommunication fraud is a problem that led to high financial losses around the world. Operators seek to combat fraud so they don't lose so much money. However, whenever a way to combat fraud is discovered, *fraudsters* discover new ways to do it. In recent times, with the end of European *roaming*, there were big changes in European telecommunications, which has led to a rise in fraud in Europe. The aim of this dissertation was to search for new ways to combat telecommunication fraud, using *machine learning* techniques.

Specifically, this dissertation focused on an operator's database, with information about calls received in a *gateway*, aiming to identify frauds such as *bypass* and *wangiri*. Firstly, an exploratory analysis based on statistical analysis was developed to better understand the data and new attributes were created to help the models. One attribute that played a key role in this dissertation was the *Range*, which is based on the grouping of telephone numbers, taking into account the variation of the last digits of numbers. Subsequently, unsupervised *machine learning* models were developed: PCA, *autoencoder*, and LSTM *autoencoder*.

One of the conclusions of this paper is that the good results produced by the PCA model suggest that non-fraud may be a linear problem, despite producing a high percentage of *outliers*. *Autoencoder* models by themselves did not produce such good results, but after applying filters based on *scores* (to try to quantify the nonlinearity of the data), there was a marked improvement in the results. Preliminary results from the LSTM *autoencoders* models suggest that their ability to store data in memory may produce very good results.

Keywords: Telecommunications fraud,*Outlier* Detection, *Machine Learning*, PCA, *Autoencoder*, LSTM *Autoencoder*

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Fraude nas telecomunicações

A fraude nas telecomunicações é um problema grave que leva à perda de cerca de 17 mil milhões de dólares por ano [9]. Apesar destas perdas estarem a ser reduzidas, [9], este é ainda um problema grave pois os *fraudsters* (pessoas que cometem a fraude) estão sempre a inovar e por isso as companhias de telecomunicações também precisam de o fazer. Uma mudança que se viu recentemente foi o facto de o *roaming* europeu acabar. Com isto a terminação de chamadas internacionais na Europa ficou bastante compensada com o uso de *gateways* (fraudulentas) dentro da zona europeia. Isto levou a um crescimento do uso de chamadas que são desviadas via *Voice over Internet Protocol* (VoIP) para *gateways* na Europa. Estas *gateways* não só prejudicam as operadoras como os consumidores que, apesar de pagarem o mesmo valor pela chamada, são servidos com uma chamada mais pobre em qualidade de som e estabilidade.

Sabe-se que os praticantes destas fraudes (*fraudsters*) aprovisionam vários cartões SIM (*subscriber identity module*) para as suas *SIMbox* e que estes cartões estão tipicamente dentro das mesmas gamas de números. Desta forma, podemos ver que um número isoladamente pode ter um comportamento praticamente descrito como normal, mas que, vendo a *range* ou as várias *ranges* como um todo,

estas correspondem a comportamentos anómalos.

1.2 *Machine learning*

Machine learning é uma área da inteligência artificial que se foca no estudo de modelos e algoritmos que permitam a um computador aprender por ele mesmo [25].

Existem tipicamente três diferentes categorizações de algoritmos:

- **Algoritmos supervisionados:** aqui entram os modelos que são treinados com dados classificados, isto é, dados que possuem uma etiqueta a indicar a classe a que pertencem (por exemplo, fraude ou não fraude podem ser as classes de um modelo);
- **Algoritmos semi-supervisionados:** estes modelos possuem apenas uma etiqueta, ou seja, geralmente apenas são classificados os dados normais;
- **Algoritmos sem supervisão:** um modelo é treinado sem qualquer classificação prévia dos seus dados, desta forma, não existe a etiqueta (por exemplo, fraude ou não fraude) que ajude o modelo a classificar de maneira mais assertiva os dados.

Para cada uma das categorias mencionadas anteriormente existem diversos métodos. No entanto nesta dissertação apenas vão ser usados métodos da última categoria, isto é, métodos sem supervisão.

Um dos métodos de *machine learning* usados nesta dissertação é o *Principal Component Analysis* (PCA), que corresponde a um modelo linear capaz de tratar problemas de redução de dimensionalidade. O outro método usado nesta dissertação é o das redes neuronais. Numa primeira fase, será usado o *autoencoder* que é um modelo mais simples dentro das redes neuronais e, depois, o *Long short-term memory* (LSTM) *autoencoder* que corresponde a uma mistura de diferentes redes neuronais.

1.3 Problema em estudo

A WeDo *technologies* é um empresa no ramo de *fraud management* e *revenue assurance*. No ramo de *fraud management*, a WeDo tem como objetivo combater a fraude nas telecomunicações e, constantemente, procura descobrir novas formas de a combater. Por esta razão foi criado este projeto. O problema abordado nesta dissertação foi lançado pela WeDo e prende-se com uma alteração recente nas telecomunicações na europa que estabeleceu o fim do *Roaming* europeu e originou um crescimento quanto aos números fraudulentos.

Uma vez que o volume de chamadas que chegam a uma *gateway* é muito elevado, a análise de cada uma das chamadas por um analista é impraticável e surge necessidade de avaliar os dados por outros meios. Desta forma, surgiu a questão da criação de um modelo de *machine learning* que facilite o trabalho dos analistas, de maneira a que estes não tenham de analisar número a número para classificá-los.

Mais especificamente, o título abordado nesta dissertação prende-se com o estudo de uma base de dados disponibilizada pela WeDo, com o objetivo de se identificar gamas de números que se associem a padrões de fraude *bypass* (ver secção 2.1) ou *wangiri* (ver secção 2.1.2).

A base de dados disponibilizada pela WeDo contém informação relativa a milhões de chamadas e corresponde a dados reais de uma operadora. Por esta razão, os dados fornecidos foram previamente anonimizados pela WeDo.

1.4 Contribuições do trabalho

Esta dissertação apresenta um estudo acerca de diferentes modelos de *machine learning* para a deteção de fraude nas telecomunicações, tomando por base os dados que chegam a uma *gateway*. Estes dados, uma vez que são enviados quase imediatamente, não possuem grandes registos de informação por cada chamada, tal como sucede com os dados fornecidos para este trabalho. Assim, numa primeira fase, foi desenvolvido um estudo com o objetivo de retirar informação destes dados

relevantes para a deteção de fraude. Este estudo passou pela criação de diversos atributos derivados dos iniciais e por algumas análises estatísticas. Na sequência deste estudo foi decidido um conjunto final de atributos que foram usados para treinar três modelos diferentes de *machine learning* que foram desenvolvidos neste trabalho, baseados nos métodos PCA, *autoencoder* e LSTM *autoencoder*.

Este trabalho, para além de ter contribuído para abrir uma porta em relação a novos modelos de *machine learning* que podem ser usados na deteção de fraude nas telecomunicações, serviu também para criar um paralelismo entre o modelo linear (PCA) e o modelo afim (*autoencoder*) e, ainda, para se poder ver uma nova utilização de dois tipos de redes neuronais combinadas.

1.5 Ferramentas

No desenvolvimento do trabalho apresentado nesta dissertação foram usadas diversas ferramentas:

- **Python**: linguagem de programação. O *Python* foi usado ao longo de todo o trabalho.
- **Jupyter Notebook**: interface do *python*. Tudo o que envolveu escrever funções/ comandos em *python* foi realizado no *jupyter notebook*.
- **Pandas**: biblioteca do *python* que lida com grandes volumes de dados. Foi usada ao longo do trabalho para tudo o que envolveu manipulação do *dataset* (conjunto de dados).
- **Scikit learn**: biblioteca do *python* para algoritmos de *machine learning*. Esta biblioteca foi usada para implementação do modelo PCA.
- **Keras**: biblioteca do *python* para algoritmos de *deep learning*. Esta biblioteca foi usada com o **TensorFlow** com *interface* na parte das redes neuronais (*autoencoder* e LSTM *autoencoder*).

1.6 Estrutura da dissertação

Nesta secção fazemos uma breve descrição dos restantes capítulos desta dissertação.

O segundo capítulo é sobre a fraude em telecomunicações, com uma explicação sobre o que é a fraude *bypass* e a fraude *wangiri* e também com uma introdução teórica ao problema da deteção de anomalias. Este capítulo conta ainda com uma secção sobre o trabalho já desenvolvido no âmbito da fraude em telecomunicações através da deteção de anomalias.

No terceiro capítulo apresentamos uma análise ao nível de atributos, quais os atributos originais, quais os atributos criados para estudo do *dataset* e quais os atributos finais. É também no terceiro capítulo onde se encontra a análise estatística da base de dados fornecida para esta dissertação.

Os três capítulos que se seguem explicam cada um dos métodos usados, PCA, *autoencoder* e LSTM *autoencoder*, a forma como foram aplicados e os resultados obtidos.

Para acabar, no último capítulo apresentamos uma conclusão com uma reflexão sobre o que foi feito e um pequeno resumo das dificuldades encontradas e dos resultados obtidos.

Capítulo 2

Deteção de fraude em telecomunicações

Nas telecomunicações, considera-se fraude o ”roubo”de serviços ou o uso de serviços para cometer alguma ilegalidade. As vítimas deste tipo de fraude podem ser os consumidores ou até mesmo as operadoras que fornecem os serviços de comunicação.

Existem diferentes tipos de fraude. Alguns dos mais conhecidos são:

- *International Revenue Share Fraud*;
- *Subscription Fraud*;
- *Roaming Fraud*;
- *Bypass Fraud*;
- *Wangiri Fraud*.

Como referido no capítulo 1, os objetivos deste trabalho centram-se na identificação de fraudes de dois tipos: *Bypass Fraud* e *Wangiri Fraud*.

2.1 Fraude *bypass* e *wangiri*

2.1.1 Fraude *bypass*

Um dos tipos de fraude nas telecomunicações que tem causado mais estragos é a denominada *bypass fraud*. Este tipo de fraude acontece quando, por exemplo, uma pessoa recebe uma chamada local mas a chamada é de alguém que está no estrangeiro. Ora, isto é possível porque a chamada internacional é passada via *internet* e transferida para um telemóvel local, que pertence a uma *SIMbox*. Assim, a pessoa que faz a chamada paga o valor de uma chamada internacional, no entanto, quem recebe a chamada recebe-a como uma chamada local, com custos muito mais pequenos. Desta forma, com a diferença entre o preço de uma chamada internacional para uma chamada local, é que quem faz este reencaminhamento ilegal da chamada (*fraudster*) consegue ganhar dinheiro [12].

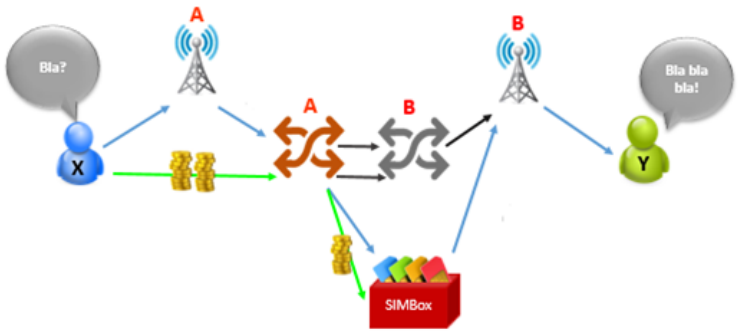


Figura 2.1: *Bypass fraud*.

A figura 2.1 esquematiza a fraude *bypass*. Nela podemos ver que existe um caminho a verde, que indica o caminho por onde passa o dinheiro, e um caminho a azul que indica o caminho por onde passa a chamada. Ora, esta figura representa uma chamada entre um cliente X de uma operadora A que liga para o cliente Y de uma operadora B. Inicialmente, o cliente X faz a chamada que é transmitida para a antena mais próxima da operadora A e o cliente X é cobrado pela chamada. Em seguida, a antena passa a chamada para o *switch* de A, que é responsável pela ligação da chamada a uma rede pública de forma a que esta possa prosseguir para

o seu destino. Numa terceira fase, a chamada é passada pelo *switch* da operadora B. Neste passo, é que o *fraudster* implementa uma SIMbox, que se faz passar por um *switch* de B, para fazer passar a chamada da *switch* de A para a SIMbox. Assim, a operadora A paga ao *fraudster* por este fazer passar a chamada. Depois, a SIMbox encaminha a chamada para a antena de B mais próxima do cliente Y. Por último, a antena de B envia a chamada para o telemóvel do cliente Y.

Este tipo de fraude acontece em maior quantidade nas chamadas internacionais. Com este tipo de fraude, não só é lesado o cliente, porque paga por um serviço mas acaba por receber um de menor qualidade (pois as chamadas via VoIP perdem qualidade), como também é lesada a operadora A, que, para além de passar a ter clientes insatisfeitos, também paga por um serviço que não é o que recebe e, a operadora B porque apenas prestou o reencaminhamento local invés do internacional.

2.1.2 Fraude *wangiri*

Um tipo de fraude que tem aparecido muito nos últimos anos é a *Wangiri fraud*. A tradução de *wangiri*, do japonês, significa "um corte". Ora, tal como o nome indica, esta fraude acontece quando é recebida uma chamada onde quem ligou apenas deixou tocar uma vez e desligou.

Existem muitos utilizadores que, quando vêem uma chamada não atendida, o que fazem é ligar para esse número, sem saber que esse número é um número *premium*, isto é, um número com chamada de valor acrescentado.

De forma a ganharem muito dinheiro, o que os *fraudsters* fazem é colocar um computador a ligar para diversos números de forma aleatória, só deixando dar um toque e desligando. Isto deixa um grande número de chamadas não atendidas em muitos utilizadores, elevando a probabilidade de algum ligar de volta.

Este tipo de fraude não faz a empresa de telecomunicações perder dinheiro diretamente. No entanto, os clientes começam a ficar descontentes com o serviço, e podem acabar por sair. Assim, a empresa pode perder dinheiro. Por esta razão, este tipo de fraude também já está a ser procurado, por mais operadoras, que

bloqueiam os números que a praticam.

2.2 O problema de deteção de anomalias

A deteção de anomalias consiste em encontrar padrões diferentes nos dados, isto é, encontrar padrões que não correspondam a um comportamento normal dos dados. Estes padrões podem ser chamados anomalias ou *outliers* [5].

Assim, podemos "definir"como *outlier* um objeto que se encontra muito longe da generalidade dos dados, ou seja, um objeto que não parece ser gerado da mesma forma que a grande maioria dos dados (ver figura 2.2) [23].

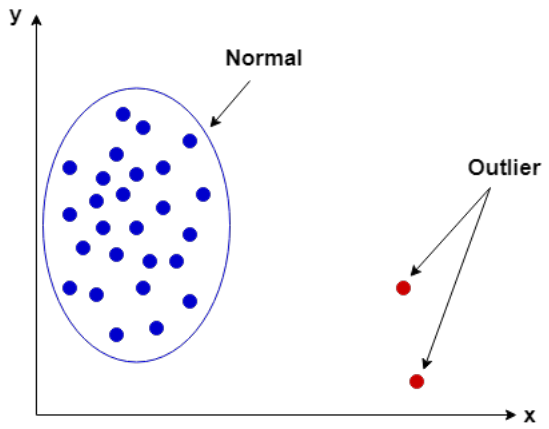


Figura 2.2: *Outliers* num gráfico de duas dimensões.

2.2.1 Tipos de anomalias

Uma parte importante na deteção de anomalias é o contexto onde estão inseridos os dados. Isto porque, dependendo do contexto, podemos querer encontrar poucos *outliers* ou conjuntos de *outliers*. Desta forma, tipicamente são considerados três tipos diferentes de anomalias.

2.2.1.1 *Point Anomalies*

O primeiro tipo de anomalia é o *point anomaly* que se baseia na descoberta de *outliers* que são pontos isolados. Este tipo de anomalia é a que se encontra na figura 2.2, onde estão dois pontos separados e isolados de todos os outros. Este é considerado o tipo mais simples de *outliers* porque estamos à procura de uma parte residual dos dados.

A maior parte das técnicas de deteção de anomalias baseia-se em *point anomalies*. Por exemplo, se considerarmos a deteção de fraude nas telecomunicações e quisermos detetar algum *outlier*, este vai ser um utilizador que de entre todos os outros ou faz mais chamadas ou faz todas as chamadas numa só hora, etc.

2.2.1.2 *Contextual Anomalies*

Contextual anomaly é um tipo de anomalia que é considerada anomalia apenas perante um contexto específico. Este tipo deriva do primeiro, no entanto, ao invés de ser anomalia em qualquer situação, está condicionado a um dado contexto. Por exemplo, no caso da fraude nas telecomunicações, um número que faça inúmeras chamadas por dia pode ser considerado um *outlier* se pertencer a uma pessoa (particular) e não ser considerado *outlier* se pertencer a uma empresa.

Este tipo de anomalias é ainda definido usando dois tipos de atributos:

- ***Contextual attributes*:** atributos que identificam o contexto em que um ponto é considerado *outlier*. Por exemplo, no caso dado anteriormente o atributo seria o número ser ou não de uma empresa.
- ***Behavioral attributes*:** atributos que definem a parte não contextual em que um ponto é um *outlier*. Por exemplo, quantas chamadas fez um número num dia.

Assim, os *behavioral attributes* são usados para definir um comportamento fraudulento dentro de um contexto. Com isto, conseguimos definir *outliers* que possam mudar consoante o contexto onde estão introduzidos.

Este tipo de anomalias é mais recorrente em pesquisas por tempo, isto é, se o que estamos a tentar detetar é uma ação que acontece ao longo do tempo.

2.2.1.3 *Collective Anomalies*

O último tipo de anomalia é a *collective anomaly* que encontra conjuntos de dados que são *outliers*. Isto é, ao invés de os *outliers* serem pontos isolados passam a ser um grupo em que todos são diferentes daquilo que é considerado "normal". Por exemplo, no caso das telecomunicações podemos pensar que um conjunto de *outliers* pode ser um conjunto de números sequenciais que estão juntos num grupo, longe dos que são considerados normais.

2.2.2 Tipos de sistemas para deteção de anomalias

Cada elemento de um *dataset* pode ser classificado como normal ou *outlier*. Podemos ter *datasets* que nos dão esta classificação de antemão, como podemos ter *datasets* que não nos dão nenhuma classificação. Na realidade, existem mais *datasets* do segundo tipo do que do primeiro, porque fazer classificação de dados pode ser muito dispendioso, tendo em conta que é um processo feito, tipicamente, por humanos.

Tendo em conta a classificação, habitualmente são considerados três tipos diferentes de sistemas para deteção de anomalias.

2.2.2.1 *Supervised anomaly detection*

O processo mais fácil para a deteção de anomalias é quando já temos um *dataset* totalmente classificado, porque, desta forma, conseguimos saber com exatidão se o nosso modelo "está ou não a ser preciso". Assim, podemos treinar um modelo com os dados classificados e quando tivermos de ver se um novo dado é ou não uma anomalia basta ver de que grupo é que este está mais próximo.

No entanto, este tipo de *datasets* também traz alguns problemas. Entre eles está

o facto de existirem muitos mais dados classificados como normais do que dados classificados como anómalos, o que provoca um desequilíbrio nos dados. Outro problema é obter exemplos representativos de dados classificados (principalmente dados anómalos), de forma a obter um modelo mais preciso. Por esta razão, ao longo do tempo, foram aparecendo algumas técnicas que colocam anomalias artificiais nos dados.

Para este tipo de problemas existem muitas técnicas de *machine learning*, entre elas redes neuronais supervisionadas, *support vector machine* (SVM), entre outras [17].

2.2.2.2 *Semi-supervised anomaly detection*

Um *dataset* semi-supervisionado é um *dataset* em que apenas os dados normais estão classificados. Desta forma, uma vez que não existe classificação de dados anómalos, este tipo de modelos acaba por ser mais fácil de aplicar que os modelos supervisionados. Isto porque não precisam de criar artificialmente cenários de anomalias, o que por vezes é muito difícil.

A maneira mais simples de construir um modelo com este tipo de dados é criar um modelo que treine apenas com dados normais (já classificados). Tal acontece porque, quando se usar o modelo criado para fazer um *predict* nos dados não classificados, aqueles que serão anómalos serão diferentes dos usados para treinar o modelo e então surgirão erros muito grandes.

2.2.2.3 *Unsupervised anomaly detection*

Por fim, o último caso que pode aparecer é um *dataset* que não tem nenhum dado classificado. No entanto, supõe-se nestes casos que o número de casos normais é muito maior do que o número de casos anómalos. Ora, se esta suposição não se verificar, vai haver um número de falsos positivos muito grande.

Para *datasets* com estas características os modelos a ser usados podem ser redes neuronais sem supervisão, técnicas de *clustering*, entre outros.

2.2.3 Deteção de anomalias: *output*

Depois de treinado e testado, um modelo de deteção de anomalias tem de retornar um *output*. O *output* pode assumir as seguintes formas:

- ***Score*:** a cada *outlier* é atribuído um grau de anomalia, quanto maior for o grau, maior a certeza de que esse ponto é um *outlier*.
- ***Label*:** a cada ponto do *dataset* é atribuída a classificação de normal ou anómalo.

Se analisarmos as duas possibilidades, podemos ver que a mais precisa será a "técnica do *score*", uma vez que nos deixa decidir a partir de que valor podemos considerar que um ponto é um *outlier*.

2.2.4 Áreas de aplicação da deteção de anomalias

Além das telecomunicações existem muitas outras áreas onde podemos aplicar a deteção de anomalias. Entre elas encontram-se:

- Deteção de Intrusos: por exemplo, detetar atividade maliciosa num computador;
- Deteção de anomalias na saúde: por exemplo, detetar através de radiografias se existe ou não tumor;
- Deteção de estragos industriais: por exemplo, detetar se uma peça está defeituosa;
- Processamento de imagens: por exemplo, detetar mudanças numa imagem ao longo do tempo;
- Deteção de anomalias em texto: por exemplo, detetar erros ortográficos.

2.2.5 Classificação baseada nas técnicas de deteção de anomalias

A classificação baseada nas técnicas de deteção de anomalias é realizada em duas fases. Na primeira fase, fase de treino, o classificador aprende o que já está classificado. Na segunda fase, fase de teste e classificação, o classificador classifica os dados como normais ou anómalos.

Tendo por base os dados que estão classificados, podemos considerar que a fase de teste e classificação pode ter duas categorias:

- ***Multi-class classification*:** na fase de treino, os dados classificados têm mais do que duas possibilidades, isto é, os dados que estão classificados distinguem entre vários tipos de dados normais. Assim, um ponto só é um *outlier* se não pertencer a nenhum dos dados normais. (Ver figura 2.3.)
- ***One-class classification*:** a fase de treinos é feita apenas com dados de uma classe, ou seja, considera que todos os dados que está a treinar são normais. Assim, todos os pontos que se encontrem longe destes dados são *outliers*. (Ver figura 2.4.)

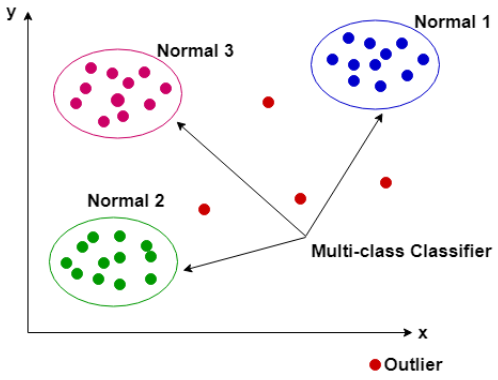


Figura 2.3: *Multi-class Classification.*

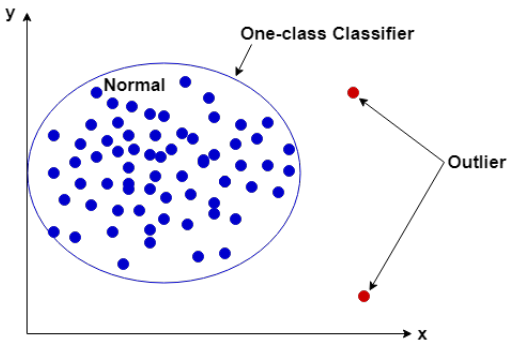


Figura 2.4: *One-class Classification.*

2.3 Deteção de anomalias aplicada à fraude em telecomunicações

Nesta secção vai ser descrito algum do trabalho que já foi feito na área desta dissertação.

No artigo "Detecting SIM Box Fraud using Neural Networks"[8], foi usada uma rede neuronal supervisionada, *Multi layer perceptron* (MLP) como classificador, para detetar *bypass fraud*. Desta experiência resultou que as redes neuronais foram muito eficientes com uma *accuracy* de 98%.

Os dados usados neste trabalho eram dados reais que pertenciam a uma companhia de telecomunicações e já estavam classificados como sendo fraude ou não.

No artigo "Classification of SIM Box Fraud Detection Using Support Vector Machine and Artificial Neural Network"[20], é feita uma comparação entre a melhor rede neuronal definida no artigo referido anteriormente [8] e o uso do método *support vector machine* (SVM). A conclusão deste trabalho [20] é que o método SVM revela uma *accuracy* de $99,06\%$ melhor que as redes neuronais $98,71\%$.

Capítulo 3

Análise preliminar e pré-processamento dos dados

Neste capítulo vai ser explicada a primeira abordagem seguida nesta dissertação que passou por uma análise preliminar dos dados disponibilizados pela WeDo e pela criação de novos atributos (atributos secundários ou derivados). Primeiro será apresentada a organização original dos dados disponibilizados pela WeDo e as operações de reparação e filtragem realizadas sobre os dados brutos. Depois será explicada a análise estatística realizada, com base em alguns métodos estatísticos que acompanharam este trabalho, desde o início ao fim. Por último serão explicados os atributos definitivos, que foram escolhidos em conjunto com a WeDo.

3.1 Organização inicial dos dados

O *dataset* fornecido pela WeDo está organizado por chamadas (eventos). O *dataset* vem em ficheiros **.csv**, um por cada dia, relativos a 3 meses. Ao todo, conta com cerca de 90 milhões de eventos. Cada evento é constituído por 5 atributos:

- **A_number**: número de telemóvel da pessoa que faz a chamada;
- **B_number**: número de telemóvel da pessoa que recebe a chamada;

- **Time**: *string* que contém a data (ano, mês, dia) e hora (horas, minutos, segundos) a que a chamada chegou à *gateway*;
- **Action**: indica se a chamada pode prosseguir (002) ou se a chamada é rejeitada (001) por o número já estar barrado ou por decisão programada da *gateway* (através de um conjunto de regras);
- **Result**: indica a decisão aplicada após a *action* (000 se passa ou 001 se não passa).

Verificam-se que deste *dataset*, a coluna dos atributos *Action* e *Result* é sempre 002 e 000, respetivamente. Desta forma, estes dois atributos não serão importantes pois são sempre iguais em todos os eventos. Assim, uma primeira decisão tomada foi a de excluir estes dois atributos: no resto da dissertação apenas são considerados os primeiros três atributos.

3.2 Reparação e filtragem dos dados

3.2.1 Organização do *dataset*

Ao analisar um dos ficheiros, reparámos de imediato que em cada ficheiro existiam alguns eventos do dia seguinte. Desta forma, o primeiro processamento a ser feito foi para "limpar os dados", ou seja, criar novos ficheiros em que, em cada um deles, apenas tivesse efetivamente eventos de um dia.

3.2.2 Comprimento do atributo *A_number*

Uma decisão tomada com base no facto de os números que são procurados como fraude serem necessariamente constituídos pelo código do país e o número local, foi a de ficar apenas os eventos com *A_number* de comprimento maior ou igual a 10. Isto, porque todos os números com um comprimento menor são assumidos ou como *ranges* de números ou como números que não são de telemóvel, por exemplo

números para ver saldo ou de apoio ao cliente. Este filtro foi aplicado a cada um dos 90 ficheiros do *dataset*.

3.3 Criação de atributos derivados

Uma vez que o *dataset* fornecido apresentava atributos que diretamente não destacavam informação útil para o modelo detetar o que podia ser fraude, houve a necessidade de conjugar a informação dos três atributos fornecidos para se poder retirar informação do *dataset*.

Desta forma, foram criados novos atributos que vão ser apresentados nesta secção. Estes atributos serviram para perceber de uma forma geral alguns dos padrões que são procurados como fraude, bem como para fazer uma análise estatística, sem usar nenhum modelo de *machine learning*.

3.3.1 Atributos *Range* e *Range_level*

Com a criação do atributo *Range* pretende-se criar grupos onde se possa englobar diferentes números. Para isso, são criados três *ranges* que dependem do atributo *A_number*:

- Uma primeira *range* que vai a cada *A_number* e lhe tira a última letra;
- Uma segunda *range* que vai a cada *A_number* e lhe tira as duas últimas letras;
- E, por fim, uma terceira *range* que vai a cada *A_number* e lhe tira as três últimas letras.

De forma a sabermos a qual destes grupos o *A_number* pertence, criámos um novo atributo que guarda esta informação.

Podemos ver na figura 3.1 um exemplo da criação destes dois atributos que dependem do atributo *A_number*.

A_number	Range	Range_level
ABCDEFGHIJ	ABCDEFGHI	1
ABCDEFGHIJ	ABCDEFGH	2
ABCDEFGHIJ	ABCDEFG	3

Figura 3.1: Exemplo da criação dos atributos *Range* e *Range_level*.

3.3.2 Atributos *ANum_Distinct_Range*, *BNum_Distinct_Range* e *Range_Calls_Day*

Uma ideia que surgiu e deu origem a vários atributos foi a de contar quantas chamadas diferentes pertencem a cada grupo (*range*). Assim, foram criados os atributos *ANum_Distinct_Range* (ver figura 3.2) e *BNum_Distinct_Range* (ver figura 3.3). O primeiro diz-nos quantos *A_number* diferentes fazem parte de cada *range* e o segundo diz-nos o mesmo mas em relação ao atributo *B_number*.

Outro atributo criado foi *Range_Calls_Day* (ver figura 3.4) que nos diz quantas chamadas foram realizadas em cada *range* por dia.

Uma vez que o objetivo deste trabalho é encontrar *ranges* suspeitas (não é identificar números em separado), foi decidido aplicar um filtro que remove todos as *ranges* que apenas têm um *A_number*.

Figura 3.2: Exemplo da criação do atributo *ANum_Distinct_Range*.

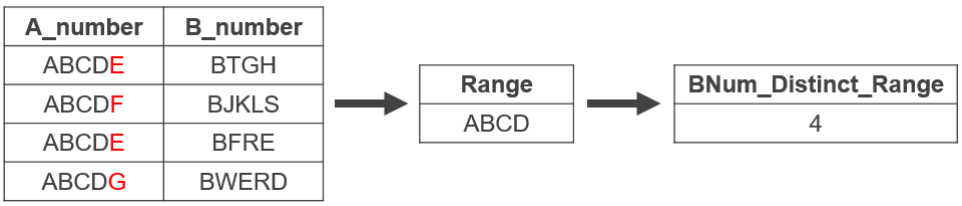


Figura 3.3: Exemplo da criação do atributo *BNum_Distinct_Range*.

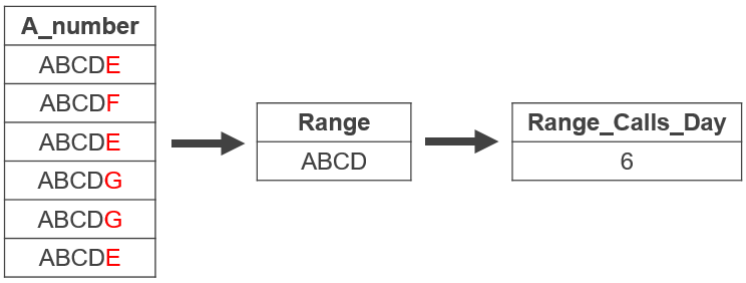


Figura 3.4: Exemplo da criação do atributo *Range_Calls_Day*.

3.3.3 Atributo *ANum_Max_Calls*

A ideia da criação do atributo *range* é, no fim, juntarmos os três *ranges* de cada dia num só ficheiro e fazer uma análise que engloba todas as *ranges* de um dia. Ora, para isto é preciso excluir os atributos *A_number*, *B_number* e *Time*, porque estes serão agrupados no atributo *range*. Assim, uma forma de retirar informação que depois será útil é guardar, para cada *range* de cada dia, o *A_number*/ *A_numbers* que fazem mais chamadas. Para este propósito foi criado este atributo *ANum_Max_Calls* (ver figura 3.5) que é composto por uma lista que contém os números que fazem mais chamadas em cada *range* por dia.

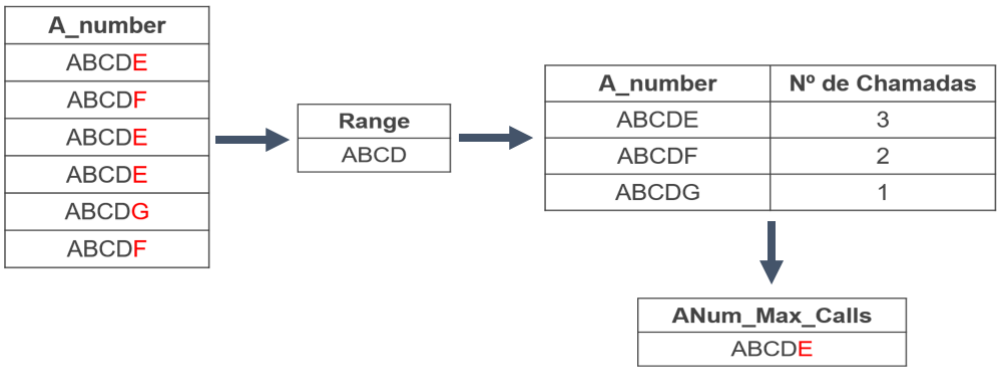


Figura 3.5: Exemplo da criação do atributo *ANum_Max_Calls*.

3.3.4 Atributo *Range_Dup_Count*

Uma situação que pode ser considerada humanamente impossível é o mesmo número de telefone fazer duas chamadas no mesmo segundo. Se pensarmos no

contexto das *ranges*, ter na mesma *range* dois números a fazer chamadas no mesmo segundo, pode ser visto como uma rotação das chamadas numa *SIMbox*.

Desta forma, a ideia foi criar um atributo, *Range_Dup_Count* (ver figura 3.6), que para além de nos dar a informação se uma *range* tinha números que fazem chamadas no mesmo segundo, ainda nos diz quantas vezes isso acontece.

Se não houver mais do que uma chamada no mesmo segundo, este atributo coloca 0, se houver coloca o número de chamadas que existem nesse segundo.

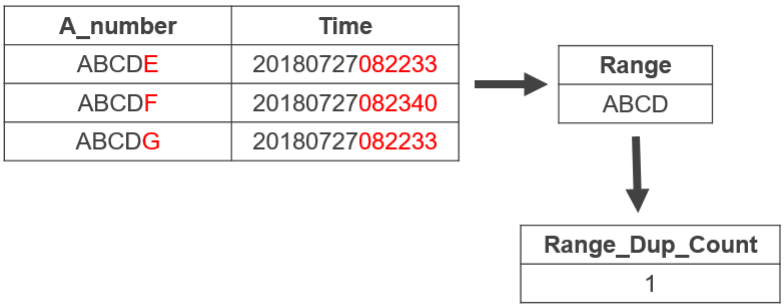


Figura 3.6: Exemplo da criação do atributo *Range_Dup_Count*.

3.3.5 Atributo *Range_Int5_Count*

Uma situação que também pode indicar fraude é se a mesma *range* só faz chamadas em certas alturas do dia, por exemplo, apenas numa determinada hora ou em alguns minutos. Para detetar esta situação dividimos um dia em intervalos de cinco minutos e criamos o atributo *Range_Int5_Count* que nos conta quantos intervalos de cinco minutos tem cada *range*.

Para criar este atributo fizemos uma função que nos coloca cada chamada no respetivo intervalo, por exemplo se uma chamada ocorrer ás 08:02 então o que a função faz é colocar este evento no intervalo 08:00. Após isto a função conta quantos intervalos diferentes existem. Podemos ver na figura 3.7 como é criado o atributo *Range_Int5_Count*.

A_number	Time	Intervals_5M
ABCDE	20180727082233	0820
ABCDF	20180727093640	0935
ABCDG	20180727082233	0820

Range	Range_Int5_Count
ABCD	2

Figura 3.7: Exemplo da criação do atributo *Range_Int5_Count*.

3.3.6 Atributos *Month*, *Day* e *Day_of_Week*

Quando no fim juntarmos todos os nossos ficheiros, vamos perder a informação sobre em que dia uma chamada foi realizada (porque o atributo *Range* agrupa os diferentes *A_numbers*). Assim, foram criados os atributos *Month* (ver figura 3.8) que guarda o mês, *Day* (ver figura 3.9) que guarda o dia e *Day_of_Week* (ver figura 3.10) que guarda o dia da semana que um evento num range aconteceu. Ora, estes atributos podem ser importantes para encontrar padrões nos dados. Por exemplo, ver se um número só faz chamadas num certo dia ou se um número só faz chamadas nos dias úteis da semana, podem ser indicadores úteis para distinguir fraude e não fraude.

Time	Month
20180727082254	07

Figura 3.8: Exemplo da criação do atributo *Month*.

Time	Day
20180727082254	27

Figura 3.9: Exemplo da criação do atributo *Day*.

Figura 3.10: Exemplo da criação do atributo *Day_of_Week*.

3.4 Análise estatística

Com a criação dos atributos e a aplicação dos filtros descritos na secção anterior, foi possível retirar do *dataset* em estudo alguns dos eventos que são considerados normais. No entanto, observou-se que alguma análise estatística permitiria reduzir ainda mais o nosso *dataset*, de forma a nos aproximarmos cada vez mais de um *dataset* só com *ranges* fraudulentos.

Para isto foram estudados dois métodos baseados em:

- Divisão dos dados em quartis;
- Desvio padrão.

Em seguida será apresentado o que é cada um destes dois métodos e como foram usados para remover eventos.

3.4.1 Divisão dos dados em quartis

Uma caixa de bigodes é um método estatístico que representa graficamente grupos de dados através da divisão em quartis. Este método começa por ordenar os dados de forma crescente para poder realizar a mediana definindo assim o segundo quartil (Q2) que representa 50% dos dados. Depois, a cada parte dividida por Q2 volta a ser calculada a mediana, obtendo-se assim o primeiro quartil (Q1) que corresponde a 25% dos dados e o terceiro quartil (Q3) que representa 75% dos dados. O menor valor é chamado de mínimo e o maior valor é chamado de máximo. Tal como podemos ver na figura 3.11 este é o aspeto de uma caixa de bigodes.

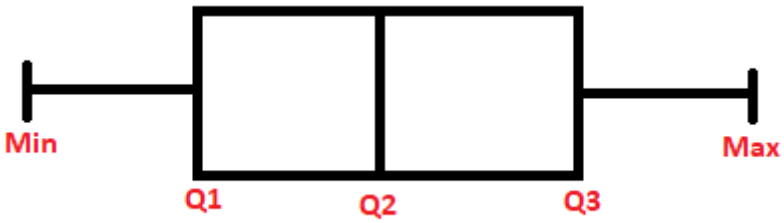


Figura 3.11: Exemplo de uma caixa de bigodes.

O espaço entre cada quartil tem como objetivo informar sobre a dispersão que existe entre os dados desses dois quartis.

3.4.1.1 Deteção de *outliers*

Um *outlier* é um valor que é muito maior ou mais pequeno que o resto dos dados [4]. Desta forma, as caixas de bigodes são úteis para a deteção de *outliers*, pois podemos definir que as extremidades ao invés de serem o mínimo e o máximo serão os valores dados por umas regras.

Uma destas regras é *Interquartile Range* (IQR) que é uma forma de cálculo da dispersão. Esta regra baseia-se na diferença entre o primeiro e o terceiro quartil. Podemos usar esta regra para calcular os pontos dos bigodes, ou seja, o bigode pequeno é dado por $Q_1 - 1.5 \times IQR = Q_1 - 1.5 \times (Q_3 - Q_1)$ e o maior por $Q_3 + 1.5 \times IQR = Q_3 + 1.5 \times (Q_3 - Q_1)$. Neste caso, consideramos como um *outlier* todos os valores abaixo do bigode mais pequeno e todos os valores acima do bigode maior [13].

No entanto, se ao invés de querermos só considerar *outliers*, quisermos também considerar valores extremos, então consideraremos que a fórmula $1.5 \times IQR$ nos dará o ponto interior do nosso intervalo e que a fórmula $3 \times IQR$ nos dará o ponto exterior [13].

De uma maneira mais geral na deteção de *outliers* este conceito de "cerca"é chamado *Tukey's fences* e é dado por $[Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)]$ onde k é uma constante não negativa. John Tukey sugere ainda que se k=1.5 deteta um

outlier e k=3 indica um dado muito longe [13].

3.4.1.2 Aplicação do método *Tukey's fences*

Um dos atributos criados foi *Range_Calls_Day* que contém o número de chamadas feitas por cada *range* em cada dia.

Existem duas alternativas para usar a caixa de bigodes:

1. Estabelecemos um valor q, com $0 \leq q \leq 1$, onde q nos dará o valor até onde o número de chamadas é normal. Por exemplo, se quisermos saber o valor de $q = 0.9$ no atributo *Range_Calls_Day* o resultado é 5 (no dia 24/07/2018). Isto significa que uma *range* que faz até 5 chamadas por dia é considerada "normal"e, por isso, estes eventos são retirados do *dataset*.

2. A outra opção é aplicar a regra IQR, isto é, se aplicarmos $Q_3 + 1.5(Q_3 - Q_1)$ ao atributo *Range_Calls_Day* o resultado é 6 (no dia 24/07/2018). Ora pelo que foi explicado na secção anterior, isto é uma forma de detetar *outliers* e aqui são retiradas todas as *ranges* que fazem até 6 chamadas por dia. Neste caso, só usámos a regra que dá o valor superior porque não interessam os *outliers* que fazem poucas chamadas.

Uma vez que a segunda opção é mais dinâmica e assim não temos de nos restringir a um valor fixo q, decidimos segui-la, removendo então todas as *ranges* abaixo do valor retornado por esta opção.

Outro atributo onde também usámos esta regra foi *BNum_Distinct_Range*, isto porque, se uma *range* não tiver uma certa variedade de números que são chamados, então também não é considerada fraudulenta.

3.4.2 Desvio padrão

O desvio padrão é um método que permite medir a dispersão dos dados. Assim, se o valor for baixo é porque não há muita dispersão nos dados, caso contrário é

porque os dados se encontram muito espalhados.

A fórmula usada para calcular o desvio padrão σ foi:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N-1}} \tag{3.1}$$

onde μ representa a média dos dados e N o número de eventos. Nesta fórmula estamos a dividir por $N-1$ porque estamos a tratar de uma amostra [22].

3.4.2.1 Coeficiente de variação

Um problema que temos no uso do desvio padrão é saber se este está ou não alto. Isto porque, precisamos do contexto para o definir como alto ou baixo. Desta forma, foi usado o método estatístico e probabilístico do coeficiente de variação (CV). Este método é definido como a divisão entre o desvio padrão e a média, e permite comparar diferentes grupos de dados.

Daqui podemos tirar que quanto maior o CV, maior o nível de dispersão e que quanto menor, mais precisos são os dados em relação à média [6].

3.4.2.2 Aplicação do desvio padrão e coeficiente de variação

Para usar o desvio padrão como um atributo útil, de forma a sabermos a dispersão dos nossos dados, a opção foi calcular o desvio padrão para cada *range* usando os *A_number*.

Numa primeira fase, foi calculada a média como o número de chamadas feitas pela *range* a dividir pelo número de *A_numbers* distintos em cada *range*. Em seguida, foi usada a fórmula 3.1 onde cada x_i representa o número de chamadas feito por cada *A_number*.

Como forma de podermos filtrar alguns eventos, foi definido o coeficiente de variação para cada uma das diferentes *ranges*. Foi definido como uma grande dispersão dos dados um coeficiente de variação acima do valor 1. No entanto, só foram retirados do *dataset* as *ranges* em que nos 90 dias de análise o coeficiente

de variação é maior que 1 e o número que está no atributo *ANum_Max_Calls* é sempre o mesmo.

Com isto, eliminamos *ranges* que são do género da figura 3.12 onde um dos *A_number* faz mais chamadas do que os outros e, passamos a ter só *ranges* como os da figura 3.13, onde, para cada *range*, os *A_number* fazem, mais ou menos, o mesmo número de chamadas. Note-se que podemos ter *ranges* que num dia têm o aspeto da figura 3.12, no entanto, se analisarmos os 90 dias, têm o comportamento da figura 3.13.

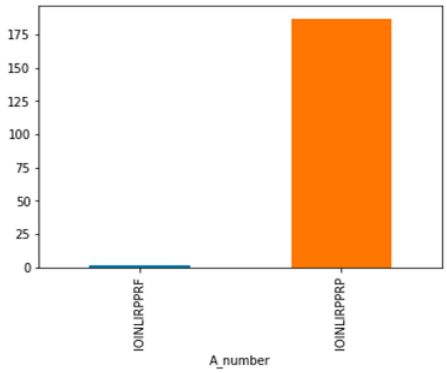


Figura 3.12: Exemplo de uma *range* retirada.

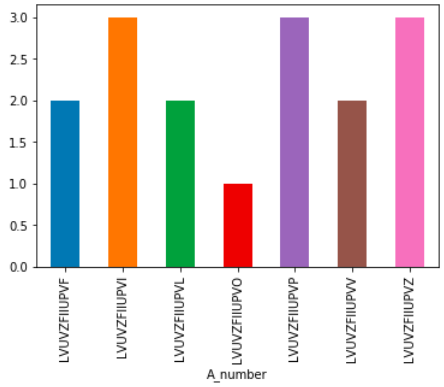


Figura 3.13: Exemplo de uma *range* aceitável.

Foi também calculado o desvio padrão em relação aos *B_numbers* que tem cada *range*. Assim, foi calculada a média como a divisão entre o número de chamadas de cada *range* e o número de *B_number* distintos em cada *range*.

3.5 Organização final dos dados

Tal como foi dito na secção 3.2.1 e na secção 3.2.2, estes foram os primeiros passos no sentido da criação dos ficheiros finais. No entanto, os atributos criados são um pouco diferentes.

Os atributos finais do *dataset* são:

1. *Range*;
2. *Range_level*;
3. *Calls_Day_CV_Ratio_mean*;
4. *AB_Ratio_mean*;
5. *A_Rotation_Ratio_mean*;
6. *B_dispersion_Ratio_mean*;
7. *Interval_5M_Consistency_Ratio_mean*;
8. *Interval_5M_Intensity_Ratio_mean*;
9. *Interval_1H_Consistency_Ratio_mean*;
10. *Interval_1H_Intensity_Ratio_mean*;
11. *Total_Calls_Day_Ratio_mean*;
12. *Days_Consistency_Ratio*.

Os dois primeiros atributos, *Range* e *Range_level*, foram obtidos da mesma forma que na secção 3.3.1. Os restantes explicam-se em seguida. A criação dos atributos foi feita em dois momentos. Primeiro são criados os atributos para cada ficheiro em separado, depois são criados os atributos *mean*, que correspondem à média dos atributos só que numa agregação por teste e numa agregação por treino.

3.5.1 Atributo *Calls_Day_CV_Ratio_mean*

Este atributo foi criado em quatro etapas.

1. Foi feita uma agregação por *A_number*, em que se contou o número de chamadas realizadas em cada dia (*Calls_Day*);

2. Depois foi feita uma agregação por *range*, onde foi calculada uma média (*Calls_Day_mean*) e um desvio padrão (*Calls_Day_std*) referente ao atributo *Calls_Day*;

3. Em seguida, foi feito o cálculo do coeficiente de variação deste atributos, ou seja,
$$Calls\_Day\_CV = \frac{Calls\_Day\_std}{Calls\_Day\_mean};$$

4. Por último, foi realizado o rácio do atributo *Calls_Day_CV* da seguinte forma:
$$Calls\_Day\_CV\_Ratio = 1 - \left|\frac{Calls\_Day\_CV}{\max(Calls\_Day\_CV)}\right|,$$
onde $\max(Calls\_Day\_CV)$ corresponde ao valor máximo do atributo *Calls_Day_CV* num dia.

Depois de cada ficheiro ter o atributo *Calls_Day_CV_Ratio* criado, é preciso ler todos os ficheiros, aplicar uma agregação por *range*, e fazer a média deste atributo (*Calls_Day_CV_Ratio*) que dá origem ao atributo *Calls_Day_CV_Ratio_mean*.

De forma resumida, podemos dizer que este atributo nos guarda o coeficiente de variação das chamadas feitas por uma *range* num conjunto de dias.

3.5.2 Atributo *AB_Ratio_mean*

Este atributo foi criado em três fases:

1. Foi realizada uma agregação por *A_number*, em que se contaram quantos *B_number* únicos existem (*BNum_Distinct*);

2. Depois, foi feita uma agregação por *range*, onde foram criados dois atributos: *ANum_Distinct_nunique*, que conta quantos *A_number* diferentes tem cada *range*, e *BNum_Distinct_sum*, que soma o número de *B_number* distintos de cada *A_number* da *range*;

3. Finalmente temos que

$$AB\_Ratio = 1 - \left|\frac{ANum\_Distinct\_nunique}{BNum\_Distinct\_sum}\right|.$$

Depois de cada ficheiro ter o atributo *AB_Ratio* criado, é preciso ler todos os ficheiros, aplicar uma agregação por *range* e fazer a média do atributo *AB_Ratio* originando o atributo *AB_Ratio_mean*.

Resumindo, podemos dizer que este atributo nos dá a média do rácio entre o número de *A_number* numa *range* sobre o número de *B_number* que foram chamados por números dessa *range* num conjunto de dias.

3.5.3 Atributo *A_Rotation_Ratio_mean*

Este atributo foi criado em duas fases:

1. Agregaram-se por *range* os elementos e criou-se o atributo *ANum_Distinct_nunique* que contém o valor de *A_number* diferentes em cada *range*;

2. Depois fez-se o seguinte cálculo:

$$A\_Rotation\_Ratio = \left|\frac{\log(ANum\_Distinct\_nunique)}{\log(10^i)}\right|,$$

onde i indica o nível da *range* (1, 2 ou 3).

Depois de cada ficheiro ter o atributo *A_Rotation_Ratio* criado, é preciso ler todos os ficheiros, aplicar uma agregação por *range* e fazer a média do atributo *A_Rotation_Ratio*, criando então o atributo *A_Rotation_Ratio_mean* para cada *range*.

Podemos dizer que este atributo nos dá a média do rácio entre o número de *A_number* numa *range* sobre o número máximo de *A_number* possíveis numa

range de acordo com o nível. Por exemplo, se estivermos no nível 1, só estamos a trocar uma letra e por isso no máximo temos 10 números diferentes.

3.5.4 Atributo *B_dispersion_Ratio_mean*

Este atributo foi criado em três etapas.

1. Foi feito uma agregação por *A_number* onde se criou os atributos *BNum_Distinct*, que indica o número de *B_number* distintos para os quais ligou cada *A_number*, e o atributo *Calls_Day* que indica quantas chamadas foram feitas num dia por cada *A_number*;

2. Em segundo lugar foi feita uma agregação por *range* onde se substituiu os atributos anteriores para *BNum_Distinct_sum*, onde temos a soma do *BNum_Distinct* de cada *A_number* daquela *range*, e *Calls_Day_sum*, onde se somou os valores de *Calls_Day* de cada *A_number* de cada *range*;

3. Por fim, calculou-se:

$$B\_dispersion\_Ratio = \frac{BNum\_Distinct\_sum}{\max(Calls\_Day\_sum)},$$

onde $\max(Calls\_Day\_sum)$ corresponde ao valor máximo do atributo *Calls_Day_sum* num dia.

Depois de cada ficheiro ter o atributo *B_dispersion_Ratio* criado, é preciso ler todos os ficheiros, aplicar uma agregação por *range* e fazer a média do atributo *B_dispersion_Ratio*, criando o atributo *B_dispersion_Ratio_mean*.

Podemos dizer que este atributo nos dá a média do rácio entre o número de *B_number* distintos chamados por uma *range* sobre o número máximo de chamadas que essa *range* fez num dia, isto é, conseguimos obter a dispersão dos *B_number*.

3.5.5 Atributo *Interval_5M_Consistency_Ratio_mean*

Este atributo foi criado em quatro fases:

1. Foi criado o atributo *Interval_5M*, onde se dividiu um dia em intervalos de 5 minutos, por exemplo, se uma chamada foi feita às 14:43, então a chamada pertence ao intervalo 14:40;

2. Em seguida, fez-se uma agregação por *A_number*, onde se contou o número de intervalos de 5 minutos distintos em que cada *A_number* realizou chamadas (*Interval_5M_Distinct*);

3. Em terceiro lugar, fez-se nova agregação, mas desta vez por *range*, onde se somou para cada *A_number* da *range* o valor do atributo *Interval_5M_Distinct*, criando-se o atributo *Interval_5M_Distinct_sum*;

4. Por último, efetuou-se o cálculo:

$$Interval\_5M\_Consistency\_Ratio = \frac{\log(Interval\_5M\_Distinct\_sum)}{\log(\max(Interval\_5M\_Distinct\_sum))}.$$

Depois de cada ficheiro ter o atributo *Interval_5M_Consistency_Ratio* criado, é preciso ler todos os ficheiros, aplicar uma agregação por *range* e fazer a média do atributo *Interval_5M_Consistency_Ratio*, criando o atribtuto *Interval_5M_Consistency_Ratio_mean*.

Este atributo ao dividir um dia em intervalos de 5 minutos, permite ver se uma *range* faz chamadas em vários momentos do dia ou se, por outro lado, concentra as chamadas apenas em alguns momentos.

3.5.6 Atributo *Interval_5M_Intensity_Ratio_mean*

Este atributo foi obtido em quatro etapas:

1. Primeiro, como em 3.5.5, foi criado o atributo *Interval_5M*, onde se dividiu um dia em intervalos de 5 minutos;

2. Depois fez-se uma agregação por *A_number* e criou-se os atributos *Interval_5M_Distinct* e *Calls_Day* (o primeiro indica o número de intervalos de 5 minutos distintos para cada *A_number* e o segundo o número de chamadas que cada *A_number* fez num dia);

3. Para cada *range*, fez-se nova agregação. Criou-se os atributos *Interval_5M_Distinct_sum* (que corresponde à soma dos atributos *Interval_5M_Distinct* de cada *A_number* de cada *range*) e o atributo *Calls_Day_sum* (que corresponde à soma dos atributos *Calls_Day* de cada *A_number* de cada *range*);

4. Calculou-se:

$$Interval\_5M\_Intensity\_Ratio = 1 - \frac{\log(Interval\_5M\_Distinct\_sum)}{\log(Calls\_Day\_sum)}.$$

Depois de cada ficheiro ter o atributo *Interval_5M_Intensity_Ratio* criado, é preciso ler todos os ficheiros, aplicar uma agregação por *range* e fazer a média dos atributos *Interval_5M_Intensity_Ratio*, dando origem ao atributo *Interval_5M_Intensity_Ratio_mean*.

Podemos dizer que este atributo, ao dividir um dia em intervalos de 5 minutos, permite avaliar a intensidade das chamadas duma *range*.

3.5.7 Atributo *Interval_1H_Consistency_Ratio_mean*

Este atributo foi criado da mesma forma que o atributo *Interval_5M_Consistency_Ratio_mean* (ver secção 3.5.5). A única diferença é que aqui, ao invés de estarmos a dividir o dia em intervalos de 5 minutos, estamos a fazê-lo em intervalos de 1 hora. Com este atributo pretendemos ver de uma maneira mais geral se uma *range* faz chamadas em vários momentos das 24 horas de um dia ou se as faz todas na mesma hora.

3.5.8 Atributo *Interval_1H_Intensity_Ratio_mean*

Este atributo foi criado da mesma forma que o atributo *Interval_5M_Intensity_Ratio_mean* (ver secção 3.5.6). A única diferença é que aqui o dia é dividido em intervalos de 1 hora. Com este atributo pretendemos ver, de uma maneira mais geral, qual a distribuição da intensidade de chamadas em cada intervalo de 1 hora.

3.5.9 Atributo *Total_Calls_Day_Ratio_mean*

Este atributo tem 3 fases:

1. Agregação do atributo *A_number* para criação do atributo *Calls_Day* (que contém o número de chamadas efetuadas por cada *A_number*);
2. Agregação do atributo *Range* para criação do atributo *Calls_Day_sum* (que conta o número de chamadas de cada *range* num dia);
3. Calcula-se:

$$Total\_Calls\_Day\_Ratio = \frac{Calls\_Day\_sum}{\max(Calls\_Day\_sum)}.$$

Depois de cada ficheiro ter o atributo *Total_Calls_Day_Ratio* criado, é preciso ler todos os ficheiros, aplicar uma agregação por *range* e fazer a média do atributo *Total_Calls_Day_Ratio*, criando o atribtuo *Total_Calls_Day_Ratio_mean*.

Portanto este atributo dá a média do rácio entre o número de chamadas de uma *range* sobre o máximo de chamadas que uma *range* fez naquele dia.

3.5.10 Atributo *Days_Consistency_Ratio*

Este atributo indica a consistência dos dias em que cada *range* faz chamadas. Assim o cálculo que é feito é:

$$Days\_Consistency\_Ratio = \frac{Day\_unique}{n},$$

onde *Day_unique* contém o número de dias distintos em que a *range* faz chamadas e n representa o número de dias em estudo, ou seja, 69 se for o ficheiro de treino ou 21 se for o de teste.

Capítulo 4

Modelo PCA

Ao longo deste capítulo será apresentado o que é o método PCA (*Principal component analysis*), e como foi efetuada a sua aplicação ao problema da deteção de fraude em telecomunicações. Serão ainda apresentados os resultados obtidos com este método que conduziram a duas soluções diferentes: uma primeira onde é feito um estudo com as três *ranges* juntas e uma segunda em que se aplica o PCA a cada *range* em separado.

4.1 Conceitos sobre o PCA

Principal component analysis (PCA) é um método que tem como objetivo diminuir a complexidade de um problema, mantendo ao máximo os padrões que se encontram nos seus dados originais. Este método transforma linearmente a informação original noutra, com menos componentes, mas mantendo a maior parte da informação.

O PCA é um método de *unsupervised learning* com o objetivo de encontrar padrões sem referência anterior à existência destes, ou sobre se as amostras vêm de diferentes grupos [19]. Quando encontra os padrões, este método projeta os dados num subespaço com dimensões iguais ou inferiores ao original. Assim, os componentes principais são os eixos ortogonais do novo subespaço (ver figura 4.1)

[18].

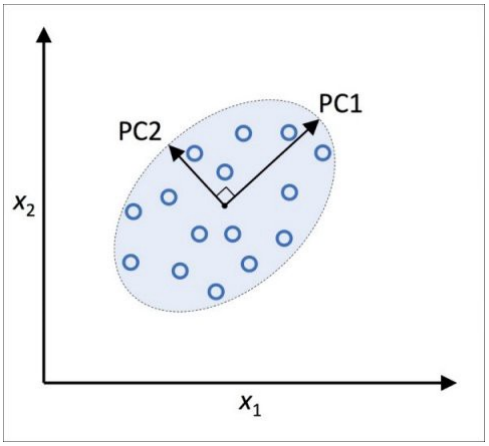


Figura 4.1: Graficamente, x_1 e x_2 são os eixos originais e $PC1$ e $PC2$ são os componentes principais [18].

O método PCA segue os seguintes passos [18]:

1. Uniformização dos dados;
2. Construção da matriz de covariância;
3. Decomposição da matriz de covariância em vetores e valores próprios e, ordenação destes;
4. Seleção dos k maiores valores próprios e correspondentes vetores próprios (com norma 1) para construção da matriz de projeção;
5. Obtenção do novo subespaço com o *dataset* e a matriz de projeção.

A primeira fase consiste em uniformizar os dados, ou seja, colocar os dados numa mesma escala com a mesma importância para todos os atributos. Em seguida, construímos a matriz de covariância. A variância mede a variação de uma variável, no entanto, a covariância indica quanto duas variáveis variam juntas. Desta forma, temos que a covariância amostral entre duas variáveis X e Y é dada por:

$$cov(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_X)(y_i - \mu_Y),$$

quando existem n observações $(x_1, y_1), ..., (x_n, y_n)$ de valores por este par de variáveis e μ_X e μ_Y são as médias dos valores observados para cada uma das variáveis. A matriz de covariância das d variáveis $X_1, .., X_d$ é a matriz C de dimensão $d \times d$ com entradas $C_{ij} = cov(X_i, X_j)$.

O terceiro passo é obter os vetores próprios da matriz de covariância, que correspondem às componentes principais, e os respetivos valores próprios. Os valores próprios contêm a magnitude do vetor próprio, assim ordenamos os valores próprios e, de acordo com a percentagem de informação útil que queremos guardar, escolhemos os k vetores próprios correspondentes aos k maiores valores próprios.

Para construir a matriz de projeção temos de pôr os vetores próprios em cada coluna, ficámos assim com a matriz $W \in R^{d \times k}$. Por último, o que falta para criar o novo subespaço é, para cada elemento X do *dataset* (vetor de elementos relativos aos valores das variáveis $X_1, .., X_d$), multiplicar pela matriz de projeção, obtendo-se $X' = XW$, onde X' é um vetor de d elementos, e corresponde à representação de X no novo subespaço.

4.2 Aplicação do PCA

Primeiramente, foram divididos os dados em 69 dias para treino e 21 dias para teste. Recorde-se que os respetivos ficheiros estão organizados por *ranges* e para cada *range* existem 13 atributos, tal como demonstrado na secção 3.5.

Depois, foram centrados os dados, ou seja, foi calculada a média de cada atributo no *dataset* e a cada valor foi retirada a média respetiva.

Em seguida, foram analisados os valores próprios e correspondentes vetores próprios (vetor de 13 atributos). Na figura 4.2 podemos ver o valor de cada valor próprio.

Tendo em conta o valor de cada valor próprio, foi analisado o gráfico que dá o número de componentes (vetores próprios) sobre a percentagem de informação mantida (ver figura 4.3). Do gráfico podemos tirar o seguinte:

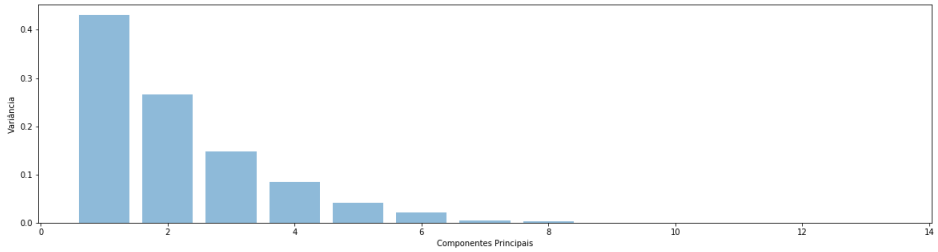


Figura 4.2: Gráfico de barras dos valores próprios.

- 1 componente = 43% da informação original;
- 2 componentes = 70% da informação original;
- 3 componentes = 84% da informação original;
- 4 componentes = 93% da informação original;
- etc.

Desta forma, foi decidido usar 4 componentes principais, de forma a preservar 93% da informação original.

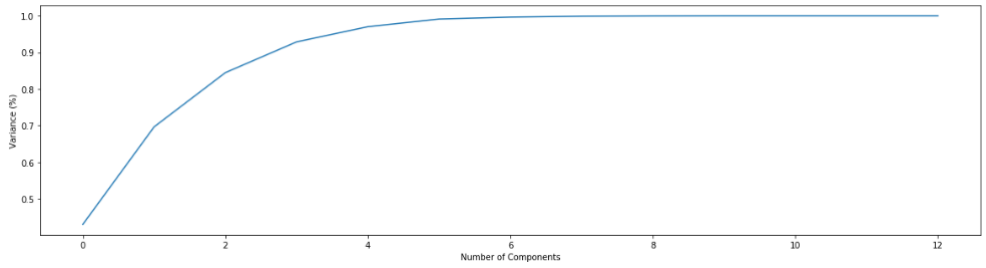


Figura 4.3: Gráfico número de componentes por percentagem de informação útil.

No PCA foi feito o treino (*fit*) com 69 dias e o teste (*transform*) com 21 dias de dados.

Para calcular depois o erro, isto é, se um range é ou não fraude, foi usada a seguinte fórmula:

$$\vec{v}_{erro} = \vec{v}_{original} - \sum_{i=1}^{4} valor_i * vetor_i,$$

onde valor corresponde ao valor próprio e vetor corresponde ao vetor próprio de norma 1.

A fórmula anterior corresponde a tirar no vetor original de dimensão 13×1, cada um dos quatro vetores próprios escolhidos, multiplicado pelo respetivo valor próprio. No fim, ficámos com o vetor do erro com dimensão 13×1. Para cada *range* do *dataset* é criado um vetor do erro. No entanto, como o objetivo é usar as *Tukey's fence* como *threshold* dinâmico, é preciso transformar um vetor de dimensão 13×1 num só valor. Assim, aplicou-se a média a cada um dos diferentes vetores de erros.

4.3 Resultados

Nesta secção vão ser apresentados os resultados da aplicação do PCA de duas formas distintas. Numa primeira abordagem, usaram-se os dados todos, isto é, juntaram-se as *ranges* dos três níveis para vermos os resultados gerais. Na segunda abordagem, aplicou-se o PCA a cada um dos três níveis de *ranges* em separado. Foram seguidos estes dois caminhos, com o objetivo de se fazer uma comparação sobre como é que o PCA funciona melhor no nosso problema.

4.3.1 PCA *ranges* dos três níveis

Os resultados que se apresentam na tabela 4.1, foram obtidos usando os seguintes parâmetros:

- **Treino:** 69 dias de dados "normais"(2 348 722 eventos),
- **Teste:** 21 dias de dados (6 890 296 eventos),
- 4 componentes principais (corresponde a 93% da informação, ver figura 4.3).

Para o treino foram usados apenas dados "normais", isto é, foi estabelecido um *threshold* que "elimina"as *ranges* que podem ser fraude, tendo em conta o número de chamadas que efetuaram. Foi seguida esta opção, porque se verificou que,

se o modelo treinar apenas com dados potencialmente normais, os erros, quando aparece algo diferente, são maiores, ou seja, quando aparece uma *range* que pode ser fraude esta tem um erro elevado.

Na tabela 4.1 apresentamos os resultados para três diferentes *thresholds*, dois foram falados anteriormente, que são os *thresholds* das *Tukey's fences* (ver secção 3.4.1), e o outro corresponde a 1% dos dados. Em relação à classificação, os dados são classificados como fraude *bypass* (ver secção 2.1), fraude *wangiri* (ver secção 2.1.2) e como não fraude, ou seja, aqueles números normais que o modelo classifica como se fossem fraude (recorde-se que há um pequeno conjunto de dados que foi classificado por analistas). Os restantes valores, por exemplo, *bypass* em *range* corresponde a *ranges* que não estão classificadas como fraude *bypass* mas que a sua *range* pertence, como sub-range, a uma já classificada.

A tabela 4.1 indica, por exemplo, que no caso da fraude *bypass* no *threshold* $Q_3 + 1.5(Q_3 - Q_1)$ encontramos 32 casos classificados como fraude *bypass* em 48 possíveis. A última linha, total de *outliers*, indica o número total de *ranges* de teste que foram classificadas como fraude em cada um dos diferentes *thresholds*.

Threshold / Classificação	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	32 em 48	14 em 48	23 em 48
Bypass em *range*	81 em 236	50 em 236	58 em 236
Wangiri	25 em 25	5 em 25	15 em 25
Wangiri em *range*	18 em 70	2 em 70	2 em 70
Não fraude	3 em 55	0 em 55	0 em 55
Não fraude em *range*	4 em 310	0 em 310	0 em 310
Total de *outliers*	456 924 (6,6%)	2 562 (0,04%)	62 713 (0,9%)

Tabela 4.1: Resultados do PCA com as *ranges* dos três níveis.

Os resultados do PCA para os diferentes *thresholds* indicam que tendo em conta os dados que temos classificados, que são uma muito pequena amostra, está a classificar bem o que nos leva a tirar a conclusão que a não fraude é um problema linear e que é por isso que este não deteta muitos falsos positivos classificados. No entanto, se virmos o exemplo do *threshold* $Q_3 + 1.5(Q_3 - Q_1)$ vemos que 6,6% de outliers é muito, isto indica que este modelo está, possivelmente, a detetar muitas *ranges* que podem não ser fraude.

Outra observação que podemos ver pelos resultados é que, quando aumentamos o *threshold*, isto é, quando restringimos mais o que é *outlier*, detetámos 0,04% de fraude. Ora, este valor poderá indicar que todas estas *ranges* são fraude, no entanto pode não estar a detetar tudo aquilo que é fraude. Desta forma, temos de definir o que é melhor, se detetar *ranges* que não são fraude ou se o melhor é aumentar o *threshold* e detetar poucos (ou nenhuns) falsos positivos. Tendo em conta o problema em estudo, acredita-se que um valor de 6,6% de fraude é muito menos provável que um valor de 0,04%, pelo que seria preferível escolher o

threshold associado a este último valor.

4.3.2 Análise das *ranges* de cada nível em separado

Em seguida, vão ser mostrados os resultados de utilizar o PCA para as *ranges* de cada nível em separado. Este estudo visa perceber o funcionamento das *ranges* neste modelo, para mais tarde se poder fazer uma comparação com outros modelos.

Pela mesma razão do caso anterior, para o treino só serão usados os dados "normais".

4.3.2.1 PCA com *ranges* de nível 1

Os resultados que se apresentam na tabela 4.2, foram obtidos usando os seguintes parâmetros:

- **Treino:** 69 dias de dados "normais"(980 450 eventos)
- **Teste:** 21 dia de dados (2 888 792 eventos)
- 3 componentes principais (corresponde a 94% da informação útil)

Threshold / Classificação	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	16 em 26	11 em 26	11 em 26
Bypass em *range*	83 em 171	54 em 171	51 em 171
Wangiri	14 em 14	10 em 14	10 em 14
Wangiri em *range*	15 em 48	5 em 48	1 em 48
Não fraude	1 em 3	0 em 3	0 em 3
Não fraude em *range*	22 em 257	0 em 257	0 em 257
Total de *outliers*	301 721 (10,4%)	98 354 (3,4%)	27 622 (1,0%)

Tabela 4.2: Resultados do PCA com as *ranges* de nível 1.

Na tabela 4.2 podemos ver os resultados do PCA aplicado às *ranges* de nível 1. Podemos ver que o número de fraudes total indicado por este modelo é de 10,4%. Este valor é muito elevado e por isso, é provável que o modelo esteja a considerar fraudulentas muitas *ranges* que não são fraude. Desta forma, apesar de 6,6% (ver tabela 4.1) também ser um valor bastante elevado é melhor que 10,4%.

Mais à frente nesta dissertação vamos comparar estes valores com os mesmos dados de treino e teste, mas com um modelo diferente.

4.3.2.2 PCA com *ranges* de nível 2

Os resultados que se apresentam na tabela 4.3, foram obtidos usando os seguintes parâmetros:

- **Treino:** 69 dias de dados ”normais”(911 080 eventos)
- **Teste:** 21 dia de dados (2 505 540 eventos)

- 3 componentes principais (corresponde a 91% da informação útil)

Threshold / Classificação	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	13 em 17	6 em 17	8 em 17
Bypass em *range*	9 em 51	5 em 51	7 em 51
Wangiri	8 em 8	2 em 8	3 em 8
Wangiri em *range*	6 em 19	1 em 19	1 em 19
Não fraude	2 em 50	0 em 50	0 em 50
Não fraude em *range*	0 em 51	0 em 51	0 em 51
Total de *outliers*	168 314 (6,7%)	3 813 (0,2%)	22 599 (0,9%)

Tabela 4.3: Resultados do PCA com as *ranges* de nível 2.

Na tabela 4.3 temos os resultados obtidos quando o treino do modelo é feito apenas com os dados de *range* de nível 2. O valor de *outliers* aqui é de 6,7%. Ora apesar deste valor ser menor que o obtido com o treino com *ranges* de nível 1, é ainda assim um valor bastante elevado.

4.3.2.3 PCA *ranges* de nível 3

Os resultados que se apresentam na tabela 4.3, foram obtidos usando os seguintes parâmetros:

- **Treino:** 69 dias de dados "normais"(457 192 eventos)
- **Teste:** 21 dia de dados (1 495 964 eventos)
- 3 componentes principais (corresponde a 92% da informação útil)

Threshold / Classificação	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	5 em 5	3 em 5	4 em 5
Bypass em *range*	0 em 14	0 em 14	0 em 14
Wangiri	3 em 3	2 em 3	2 em 3
Wangiri em *range*	0 em 3	0 em 3	0 em 3
Não fraude	0 em 2	0 em 2	0 em 2
Não fraude em *range*	0 em 2	0 em 2	0 em 2
Total de *outliers*	77 188 (5,2%)	134 (0,009%)	12 492 (0,8%)

Tabela 4.4: Resultados do PCA com as *ranges* de nível 3.

Na tabela 4.4 podemos ver que a percentagem considerada fraude com o *threshold* de $Q_3 + 1.5(Q_3 - Q_1)$ é, tal como para o treino com *ranges* de nível 1 ou 2, elevado. Assim, o objetivo desta dissertação será ver se com outro tipo de modelo este valor baixa drasticamente. Ora, queremos um valor pequeno, porque para uma operadora é melhor apanhar menos fraudes do que apanhar muitos falsos positivos. Isto porque se uma operadora bloquear números correspondentes a não fraude, classificados fraudulentos, os clientes começariam a ficar descontentes e poderiam mesmo acabar por sair.

4.4 Conclusões do uso do PCA

Na secção anterior apresentámos de uma forma mais detalhada os resultados obtidos usando o método do PCA. Nas tabelas 4.5, 4.6 e 4.7, podemos ver uma comparação entre usar as *ranges* dos três níveis juntas e em separado, para os três *thresholds* definidos anteriormente ($Q_3 + 1.5(Q_3 - Q_1)$, $Q_3 + 3(Q_3 - Q_1)$ e $Q(0.99)$.

Ranges / Classificação	*Ranges* juntas	*Ranges* separadas
Bypass	32 em 48	$16 + 13 + 5 = 34$ em 48
Bypass em *range*	81 em 236	$83 + 9 + 0 = 92$ em 236
Wangiri	25 em 25	$14 + 8 + 3 = 25$ em 25
Wangiri em *range*	18 em 70	$15 + 6 + 0 = 21$ em 70
Não fraude	3 em 55	$1 + 2 + 0 = 3$ em 55
Não fraude em *range*	4 em 310	$22 + 0 + 0 = 22$ em 310
Total de *outliers*	456 924 (6,6%)	547 223 (7,9%)

Tabela 4.5: Comparação resultados PCA: *threshold*=$Q_3 + 1.5(Q_3 - Q_1)$.

Ranges / Classificação	*Ranges* juntas	*Ranges* separadas
Bypass	14 em 48	$11 + 6 + 3 = 20$ em 48
Bypass em *range*	50 em 236	$54 + 5 + 0 = 59$ em 236
Wangiri	5 em 25	$10 + 2 + 2 = 14$ em 25
Wangiri em *range*	2 em 70	$5 + 1 + 0 = 6$ em 70
Não fraude	0 em 55	$0 + 0 + 0 = 0$ em 55
Não fraude em *range*	0 em 310	$0 + 0 + 0 = 0$ em 310
Total de *outliers*	2 562 (0,04%)	102 301 (1,5%)

Tabela 4.6: Comparação resultados PCA: *threshold*=$Q_3 + 3(Q_3 - Q_1)$.

Ranges / Classificação	*Ranges* juntas	*Ranges* separadas
Bypass	23 em 48	$11 + 8 + 4 = 23$ em 48
Bypass em *range*	58 em 236	$51 + 7 + 0 = 58$ em 236
Wangiri	15 em 25	$10 + 3 + 2 = 15$ em 25
Wangiri em *range*	2 em 70	$1 + 1 + 0 = 2$ em 70
Não fraude	0 em 55	$0 + 0 + 0 = 0$ em 55
Não fraude em *range*	0 em 310	$0 + 0 + 0 = 0$ em 310
Total de *outliers*	62 713 (0,9%)	62 713 (0,9%)

Tabela 4.7: Comparação resultados PCA: *threshold*=$Q(0.99)$.

Como se pode ver na tabela 4.5, os resultados de usar três modelos em separado (um para cada nível de *range*) pode não ser tão benéfico com o *threshold* de $Q_3 + 1.5(Q_3 - Q_1)$, uma vez que este *threshold* considera ainda uma grande percentagem de *ranges* como *outliers*. No entanto, se analisarmos a tabela 4.6, vemos que as *ranges* em separado são mais benéficas porque uma percentagem de 1,5% para fraude é aceitável e já são encontrados mais casos classificados corretamente como fraude, sem registar falsos positivos. Por último, se analisarmos a tabela 4.7, que corresponde ao *threshold* $Q(0.99)$, uma vez que este já não é um *threshold* dinâmico, obtemos os mesmos resultados usando as duas opções, *ranges* juntas e *ranges* em separado.

Capítulo 5

Modelo *Autoencoder*

5.1 Introdução aos *autoencoders*

Um *autoencoder* é uma rede neuronal que é treinada para que o seu *input* seja o mais parecido possível com o seu *output*, apenas com uma margem pequena de erro [3].

A margem de erro vem do facto de que numa primeira fase é diminuída a dimensionalidade do *input* e, só depois, é que se tenta reconstruir o *input*.

Assim, o objetivo do *autoencoder* é arranjar uma função f que seja o mais parecida possível com a função identidade.

Podemos considerar que um *autoencoder* é constituído por camadas e organizado em três partes: *encoder*, *decoder* e *code* (ver figura 5.1). Cada camada é constítuida por um conjunto de neurónios (representados como círculos na figura 5.1), ligados a neurónios da camada seguinte.

A primeira camada do *autoencoder* corresponde ao *encoder*, onde temos no início o *input* (camada x), que termina na camada *code* (camada central z). Em seguida, encontra-se o *decoder* que acaba com a camada *output* (camada $\hat{x}$). As camadas internas do *autoencoder* (que no caso da figura 5.1 são as três camadas além da camada de *input* e de *output*) são habitualmente designadas *hidden layers*.

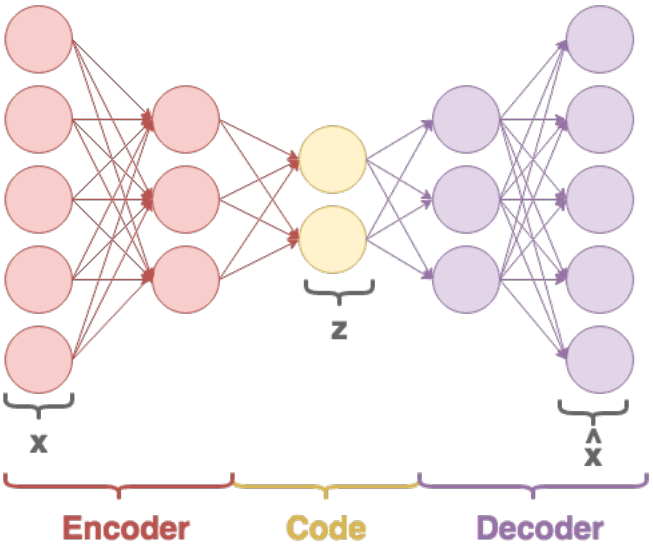


Figura 5.1: Representação de um *autoencoder* com três *hidden layers*.

Se virmos um *autoencoder* de uma abordagem matemática, podemos pensar que o *input* é um vetor x e que o *encoder* é um função f. O objetivo é aplicar a função f ao vetor x de forma a obter um resultado z. O resultado é dado pela fórmula:

$$z = f(x) = s_f(Wx + b_z)$$

onde s_f corresponde à **função de ativação do *encoder***, W é a **matriz dos pesos** e b_z é o *bias vetor*.

Por outro lado, o *decoder* é uma função g que tenta fazer o inverso de f,

$$\hat{x} = g(z) = g(f(x)) = s_g(W'z + b_x)$$

onde s_g é a **função de ativação do *decoder***, W' a **matriz dos pesos** e b_x o *bias vector*.

Numa primeira fase, o que se faz é treinar o *autoencoder*. Esta fase consiste em encontrar os melhores parâmetros W, b_z, b_x, de forma a minimizar a função *loss*, $L(x, g(f(x)))$. Esta função tem como objetivo medir a qualidade do erro de reconstrução e, por isso, quanto menor for, menor o erro da reconstrução.

5.1.1 Parâmetros do *autoencoder*

Para treinar um modelo de *autoencoder* temos de definir alguns parâmetros, tal como dito na secção anterior. Em seguida são apresentados alguns destes parâmetros.

5.1.1.1 Camadas

Um primeiro parâmetro a decidir é o número de camadas (internas) que o modelo vai ter. Para além disso é preciso decidir o número de neurónios de cada camada. Ora, não existe nenhuma "receita"que permita escolher o melhor número de camadas e de neurónios. Por isso, o que se faz na prática é aumentar ou diminuir a rede neuronal e analisar a opção que produz melhores resultados.

5.1.1.2 Função de ativação

Tal como referido anteriormente, a função de ativação é outro parâmetro a definir. Esta poderá ser uma função linear ou uma função não linear. Se for usada uma função linear, o *autoencoder* faz o mesmo que o PCA. Desta forma, a função de ativação costuma ser uma função não linear, que tem que se definir em cada camada do *autoencoder*, isto é, para cada camada podemos ter funções de ativação diferentes. As funções de ativação não lineares mais conhecidas são:

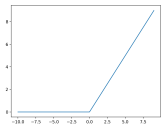
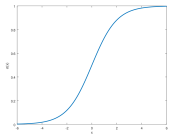
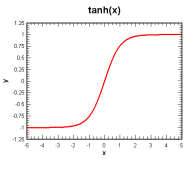
Funções de Ativação	Gráfico
ReLU	
Sigmoid	
tanh	

Tabela 5.1: Algumas funções de ativação.

5.1.1.3 Função *loss*

A função *loss* representa quanto do *input* é que foi reconstruido tendo em conta o *output*. Desta forma, o objetivo de usar esta função é minimizá-la, pois quanto mais pequeno o seu valor, melhor é a reconstrução. Assim, esta função é usada como guia quando o modelo está a ser treinado.

Na ferramenta Keras (a ferramenta que foi usada na criação de modelos de *autoencoder* para o nosso problema), existem já algumas funções *loss* definidas. Na tabela 5.2, podemos ver algumas das mais populares, bem como a sua norma associada. No entanto, a ferramenta Keras também permite criar uma função *loss* personalizada.

Função Loss	Fórmula	Norma
Mean Squared Error	$\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{n}$	L_1
Mean Absolut Error	$\frac{\sum_{i=1}^{n}\lvert x_i - \hat{x}_i\rvert}{n}$	L_{infty}
Root Mean Squared Error	$\sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{n}}$	L_2

Tabela 5.2: Algumas funções *loss*. [7]

5.1.1.4 Número de épocas

Uma época corresponde ao processamento (uma vez) pelo *autoencoder* de processar todos os dados definidos para treino. Posto isto, um outro parâmetro que podemos passar ao *autoencoder* é o número de épocas que queremos que o modelo esteja a treinar. Para este não se tornar um modelo estático, com um valor fixo de épocas, uma vez que usámos a ferramenta *Keras*, podemos utilizar uma função chamada "ReduceLROnPlateau"que permite que o modelo apenas pare de treinar quando for atingida estabilidade em relação à função *loss*, ou seja, quando esta atingir um *plateau*.

5.2 Aplicação do *autoencoder*

Tal como explicado na secção anterior, para treinar um modelo de *autoencoder* podemos definir muitos parâmetros diferentes. Os resultados que vão ser apresentados na próxima secção usaram os seguintes parâmetros:

- **Camadas** : 13-8-2-8-13 (corresponde a 13 neurónios na primeira camada, 8 neurónios na segunda, 2 neurónios na terceira e as últimas duas são simétricas às duas primeiras)

- **Função *Loss***: *Mean Squared Error*
- **Função de ativação**: ReLU (em todas as camadas)

A primeira camada tem 13 neurónios que corresponde a um neurónio por cada atributo passado como *input*. Para o *output* a estratégia seguida foi a mesma. Em relação às outras camadas foram realizadas diversas tentativas até definir que seria 8-2-8.

Para o treino dos modelos foram usados 69 dias de dados, mas foi feita uma filtragem: usando os quartis foram eliminadas as *ranges* potencialmente "anómalas"relativamente ao número de chamadas por dia, ou seja, foram eliminadas as *ranges* que fazem muitas chamadas por dia. Esta ideia deveu-se ao facto de que, se o *autoencoder* aprender apenas aquilo que é "normal", então se lhe aparecer uma *range* anómala, este consegue dar um erro muito maior.

Para o teste dos modelos foram usados os restantes 21 dias. Neste ponto, houve uma filtragem para eliminar *ranges* que não interessam, designadamente *ranges* que em 21 dias só fizeram uma chamada. Isto porque, essas *ranges* não trazem prejuízo à operadora.

Os resultados dos modelos vêm na forma do *input*, ou seja, no *input* passámos uma matriz que corresponde ao número de *ranges* por número de atributos. Desta forma, no fim, temos uma matriz com as mesmas dimensões, mas com o valor do erro da reconstrução. Por isso, foi preciso definir uma maneira de que a cada *range* só se associasse um valor. Assim, foi feita uma média dos erros dos atributos de cada *range*.

5.3 Resultados preliminares

De seguida, vão ser apresentados os resultados dos *autoencoders* para cada um dos *thresholds*:

- $Q_3 + 1.5(Q_3 - Q_1)$;

- $Q_3 + 3(Q_3 - Q_1)$;
- $Q(0.99)$.

5.3.1 *Autoencoder* com *ranges* dos três níveis

Nesta secção vão ser apresentados os resultados obtidos para o treino do modelo com as *ranges* dos três níveis juntas. Tal como sucedeu na aplicação do método do PCA (ver secção 4.3.1), os dados de treino foram apenas dados "normais".

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	32 em 48	5 em 48	5 em 48
Bypass em *range*	98 em 236	41 em 236	18 em 236
Wangiri	25 em 25	0 em 25	0 em 25
Wangiri em *range*	26 em 70	0 em 70	0 em 70
Não fraude	10 em 55	0 em 55	0 em 55
Não fraude em *range*	31 em 310	1 em 310	0 em 310
Total de *outliers*	646 250 (9,379%)	228 441 (3,315%)	65 685 (0,953%)

Tabela 5.3: Resultados do *autoencoder* com as *ranges* dos três níveis.

Se analisarmos os resultados da tabela 5.3 vemos que a percentagem de *outliers* é muito alta no caso do *threshold* $Q_3 + 1.5(Q_3 - Q_1)$, mas que no caso do *threshold* $Q_3 + 3(Q_3 - Q_1)$ pode vir a ser uma percentagem de erro mais aceitável, no entanto não apanha muitos casos classificados como fraude.

5.3.2 Análise separada das *ranges* de cada nível

Da mesma maneira que foi feito para o PCA o estudo das *ranges* de cada um dos três níveis em separado (ver secção 4.3.2), vai ser feito esse estudo para o *autoencoder*. Assim, nesta secção vamos ter três subsecções em que, em cada uma delas, vamos ter os resultados relativos ao estudo de cada um dos três níveis de *range* separadamente.

5.3.2.1 *Autoencoder* com as *ranges* de nível 1

Nesta primeira subsecção vão ser mostrados os resultados de treinar o *autoencoder* apenas com as *ranges* que correspondem ao nível 1, ou seja, *ranges* onde lhes falta apenas a primeira letra.

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	15 em 26	14 em 26	13 em 26
Bypass em *range*	97 em 171	77 em 171	75 em 171
Wangiri	14 em 14	13 em 14	13 em 14
Wangiri em *range*	21 em 48	16 em 48	13 em 48
Não fraude	1 em 3	0 em 3	0 em 3
Não fraude em *range*	49 em 257	20 em 257	20 em 257
Total de *outliers*	373 389 (12,925%)	118 216 (4,092%)	28 887 (1,000%)

Tabela 5.4: Resultados do *autoencoder* com as *ranges* de nível 1.

Tal como podemos ver na tabela 5.4, a percentagem de *outliers* mais aceitável seria a do *threshold* $Q_3 + 3(Q_3 - Q_1)$ e, que ao contrário dos resultados do *autoencoder* com as *ranges* de três níveis aqui são encontrados mais casos classificados

como fraude. No entanto, se virmos no contexto real, esta não é, ainda, uma percentagem de *outliers* satisfatória.

5.3.2.2 *Autoencoder* com as *ranges* de nível 2

Esta subsecção apresenta os resultados do modelo *autoencoder* treinado com as *ranges* de nível 2.

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	13 em 17	13 em 17	0 em 17
Bypass em *range*	15 em 51	11 em 51	0 em 51
Wangiri	8 em 8	8 em 8	0 em 8
Wangiri em *range*	6 em 19	3 em 19	0 em 19
Não fraude	29 em 50	13 em 50	0 em 50
Não fraude em *range*	0 em 51	0 em 51	0 em 51
Total de *outliers*	332 071 (13,253%)	227 296 (9,072%)	22 618 (0,903%)

Tabela 5.5: Resultados do *autoencoder* com as *ranges* de nível 2.

Na tabela 5.5 temos os resultados do treino do *autoencoder* com as *ranges* de nível 2. Aqui, as percentagens de *outliers* são todas muito altas, não mostrando assim grande utilidade.

5.3.2.3 *Autoencoder* com as *ranges* de nível 3

Por último, esta subsecção falará dos resultados do *autoencoder* com as *ranges* de nível 3.

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	5 em 5	5 em 5	5 em 5
Bypass em *range*	2 em 14	1 em 14	1 em 14
Wangiri	2 em 3	2 em 3	1 em 3
Wangiri em *range*	0 em 3	0 em 3	0 em 3
Não fraude	1 em 2	1 em 2	1 em 2
Não fraude em *range*	0 em 2	0 em 2	0 em 2
Total de *outliers*	202 793 (13,556%)	53 280 (3,562%)	14 960 (1,000%)

Tabela 5.6: Resultados do *autoencoder* com as *ranges* de nível 3.

Podemos observar que na tabela 5.6 temos uma percentagem de erro muito elevada para o *threshold* $Q_3 + 1.5(Q_3 - Q_1)$, mas uma menor percentagem para o *threshold* $Q_3+3(Q_3-Q_1)$. Apesar de esta não ser ainda a ideal, é uma percentagem que consegue apanhar quase todos os casos de fraude classificados em *ranges* de nível 3.

Se compararmos os resultados do *autoencoder* com os do PCA vemos que os do PCA são ligeiramente melhores. No entanto, com as redes neuronais surgiram novas ideias para tentar reduzir o grande problema dos dois modelos: a percentagem de *outliers*.

5.4 Filtros

Tendo em conta os resultados apresentados anteriormente com os modelos PCA (ver secção 4.3) e com os modelos *autoencoder* (ver secção 5.3), podemos ver que, de uma forma geral, o que não é fraude se comporta de uma maneira linear. Posto

isto, foram surgindo algumas ideias de forma a "combater"as elevadas percentagens de fraude que os *thresholds* $Q_3 - 1.5(Q_3 - Q_1)$ e $Q_3 - 3(Q_3 - Q_1)$ indicam.

Para a comparação dos modelos foi criado um *score* tendo em conta os valores de fraude, não fraude e a percentagem de *outliers*. Assim, uma vez que o mais importante é apanhar muitas fraudes e poucas não fraudes, foi dado um peso maior a estes e um menor peso à percentagem de *outliers*. Foi usada a seguinte expressão:

$$0.4 \times F + 0.4 \times NF + 0.2 \times PO,$$

com $F = fraude$, $NF = nofraude$ e $PO = percentagem\ de\ outliers$.

5.4.1 *Score* dos números negativos

Uma das primeiras ideias que surgiu foi usar as matrizes de pesos de um *autoencoder* para tirar algumas conclusões. Por exemplo multiplicarmos o vetor de *input* (vetor inicial dos dados) pela matriz de pesos da camada a seguir e se no vetor obtido houver resultados negativos, então isso implica que foi usada a função de ativação. Ora, se a função de ativação foi usada é porque houve um caso não linear. Assim, decidimos construir um *score* baseado na soma de todos os números negativos obtidos ao longo das várias multiplicações entre o vetor obtido e a matriz de pesos da camada a seguir.

Para utilizar esta técnica de *score*, foram usados os valores das *ranges* que foram classificadas como fraude pelo *threshold* $Q_3 + 1.5(Q_3 - Q_1)$ com o *autoencoder* descritos na secção 5.3.1. Os resultados obtidos por este *score* apresentam-se na tabela 5.7.

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	30 em 48	30 em 48	8 em 48
Bypass em *range*	86 em 236	86 em 236	11 em 236
Wangiri	24 em 25	23 em 25	10 em 25
Wangiri em *range*	18 em 70	18 em 70	0 em 70
Não fraude	9 em 55	9 em 55	5 em 55
Não fraude em *range*	29 em 310	29 em 310	1 em 310
Total de *outliers*	49 205 (0,714%)	48 234 (0,700%)	6 162 (0,089%)

Tabela 5.7: Resultados da aplicação do *score* ao *autoencoder* com as *ranges* dos três níveis.

Importa comparar estes resultado com os resultados obtidos na secção 5.3.

Por exemplo, se analisarmos os resultados das tabelas 5.3 e 5.7 vemos que houve uma melhoria grande em relação à percentagem de *outliers* detetados. Para ajudar nesta análise, podemos ver na tabela 5.8 esta comparação tendo em conta o *score* para comparação de modelos definido no início da secção 5.4. Tal como vemos, o melhor modelo é então com o *score* dos números negativos o *threshold* $Q_3 + 1.5(Q_3 - Q_1)$.

Score / *Threshold*	*Autoencoder*	*Score* dos números negativos
$Q_3 + 1.5(Q_3 - Q_1)$	84,185%	85,012%
$Q_3 + 3(Q_3 - Q_1)$	61,420%	84,215%
$Q(0.99)$	61,893%	67,679%

Tabela 5.8: Comparação entre o *score* do modelo do *autoencoder* e do modelo baseado no *score* dos números negativos: tabela 5.3 e tabela 5.7

.

5.4.2 PCA: Matriz do *autoencoder*

Uma segunda ideia foi usar o método PCA, mas, ao invés de lhe passar os dados originais para treino, passar então a matriz do *autoencoder*, isto é, multiplicar todas as matrizes de pesos do *autoencoder* até obter uma matriz final de tamanho $d \times d$, em que d é o número de atributos que neste caso é 13.

Para aplicação do PCA com a matriz final da multiplicação das matrizes de pesos do *autoencoder*, foi usada apenas uma componente principal, uma vez que apenas uma componente preserva 99% da informação útil. Assim, para calcular o erro deste *score* foi feito:

$$\vec{v}_{erro} = \vec{v}_{original} - valor_1 \times vetor_1,$$

que tal como explicado na secção 4.2, é retirado a cada vetor original (cada *range* tem o seu vetor com 13 atributos), o vertor próprio de norma 1 $\times$ valor próprio, obtendo-se o vetor erro, um por cada *range*.

Tal como na subsecção anterior, para utilizar esta técnica de *score*, foram usados os valores do *autoencoder* com as *ranges* de três níveis correspondentes ao *threshold* $Q_3 + 1.5(Q_3 - Q_1)$. Na tabela 5.9 podemos analisar os resultados obtidos.

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	7 em 48	4 em 48	17 em 48
Bypass em *range*	20 em 236	0 em 236	53 em 236
Wangiri	0 em 25	0 em 25	10 em 25
Wangiri em *range*	0 em 70	0 em 70	2 em 70
Não fraude	0 em 55	0 em 55	0 em 55
Não fraude em *range*	0 em 310	0 em 310	0 em 310
Total de *outliers*	100 (0,001%)	12 (0,00017%)	5 065 (0,1%)

Tabela 5.9: Resultados do filtro PCA: matriz do *autoencoder* baseado no *autoencoder* com as *ranges* dos três níveis.

Ao vermos a tabela 5.9 conseguimos ver que os resultados são muito ”pobres”, isto é, com este filtro quase nenhum caso classificado como fraude é encontrado, uma vez que o PCA está a diminuir em muito a percentagem de *outliers*. Se analisarmos os *scores* dá:

- $Q_3 + 1.5(Q_3 - Q_1) = 61,916\%$;
- $Q_3 + 3(Q_3 - Q_1) = 61,667\%$;
- $Q(0.99) = 75,063\%$.

Podemos, com isto, decidir que este não é um bom filtro para restringir a percentagem de *outliers*.

5.4.3 Filtros Combinados

A última ideia foi combinar as técnicas usadas anteriormente com os diferentes *scores* mencionados nesta secção.

Foram feitas três combinações:

- *Score* de números negativos/ PCA: Matriz do *autoencoder*;
- PCA: Matriz do *autoencoder*/ *Score* de números negativos;
- *Autoencoder*/ PCA.

5.4.3.1 *Score* de números negativos/ PCA: Matriz do *autoencoder*

A primeira comparação consiste em usar os dados do *autoencoder* com o *threshold* $Q_3 + 1.5(Q_3 - Q_1)$ e depois passá-los ao *score* de números negativos (ver resultados tabela 5.7). Depois, usando o *threshold* $Q_3 + 1.5(Q_3 - Q_1)$, classificar estes modelos pelo PCA treinado com a multiplicação das matrizes de pesos do *autoencoder*. Obtivemos os resultados que estão na tabela 5.10.

Classified \ Threshold	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	12 em 48	7 em 48	10 em 48
Bypass em *range*	48 em 236	20 em 236	42 em 236
Wangiri	2 em 25	0 em 25	2 em 25
Wangiri em *range*	0 em 70	0 em 70	0 em 70
Não fraude	0 em 55	0 em 55	0 em 55
Não fraude em *range*	0 em 310	0 em 310	0 em 310
Total de *outliers*	747 (0,01%)	73 (0,001%)	493 (0,007%)

Tabela 5.10: Resultados do primeiro filtro combinado baseado no *autoencoder* com as *ranges* dos três níveis.

Tal como aconteceu anteriormente, ao usar o filtro PCA: Matriz do *autoencoder*, neste caso também as percentagens são muito reduzidas, acabando por não serem úteis. Os resultados do *score* são:

- $Q_3 + 1.5(Q_3 - Q_1) = 66,598\%$;
- $Q_3 + 3(Q_3 - Q_1) = 62,916\%$;
- $Q(0.99) = 65,765\%$.

5.4.3.2 PCA: Matriz do *autoencoder*/ *Score* de números negativos

Uma segunda combinação consisitiu em usar os mesmos filtros, mas por ordem inversa. Assim, primeiro usaram-se os dados do *autoencoder* com *threshold* de $Q_3 + 1.5(Q_3 - Q_1)$, depois passaram-se estes dados para serem classificados pelo PCA previamente treinado com a multiplicação das matrizes de pesos do *autoencoder*

e, por fim, usaram-se os dados que passavam no *threshold* de $Q_3 + 1.5(Q_3 - Q_1)$ para serem usados no *score* de números negativos.

Na tabela 5.11 que se apresenta a seguir, são apresentados os resultados desta combinação.

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	0 em 48	0 em 48	7 em 48
Bypass em *range*	0 em 236	0 em 236	20 em 236
Wangiri	0 em 25	0 em 25	0 em 25
Wangiri em *range*	0 em 70	0 em 70	0 em 70
Não fraude	0 em 55	0 em 55	0 em 55
Não fraude em *range*	0 em 310	0 em 310	0 em 310
Total de *outliers*	0 (0%)	0 (0%)	99 (0,001%)

Tabela 5.11: Resultados do segundo filtro combinado baseado no *autoencoder* com as *ranges* dos três níveis.

Ao observarmos a tabela 5.11 vemos que os *thresholds* $Q_3 + 1.5(Q_3 - Q_1)$ e $Q_3 + 3(Q_3 - Q_1)$ não podem ser comparados pois não existem resultados. Assim, apenas temos o *score* do *threshold* $Q(0.99)$ que dá $62,916\%$. Com isto, podemos concluir que esta combinação também não é uma boa combinação.

5.4.3.3 *Autoencoder*/ PCA

Por último, a ideia foi passar os dados do *autoencoder* com o *threshold* de $Q_3 + 1.5(Q_3 - Q_1)$ ao PCA, previamente treinado com os dados todos (tal como na secção 4.2).

Os resultados obtidos apresentam-se na tabela 5.12.

Classified \ Threshold	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	7 em 48	4 em 48	17 em 48
Bypass em *range*	20 em 236	0 em 236	53 em 236
Wangiri	0 em 25	0 em 25	10 em 25
Wangiri em *range*	0 em 70	0 em 70	2 em 70
Não fraude	0 em 55	0 em 55	0 em 55
Não fraude em *range*	0 em 310	0 em 310	0 em 310
Total de *outliers*	100 (0,001%)	12 (0,00017%)	5 065 (0,07%)

Tabela 5.12: Resultados do terceiro filtro combinado baseado no *autoencoder* com as *ranges* dos três níveis.

Se analisarmos os valores do *score* temos:

- $Q_3 + 1.5(Q_3 - Q_1) = 62,916\%$;
- $Q_3 + 3(Q_3 - Q_1) = 61,667\%$;
- $Q(0.99) = 75,069\%$.

Tal como podemos ver o melhor *score* é com o *threshold* de $Q(0.99)$.

Se analisarmos todos os resultados que foram apresentados ao longo desta secção, podemos concluir que o melhor filtro é o *score* de números negativos. Anteriormente, também foi feita uma comparação dos resultados apresentados por este filtro com os resultados apresentados pelo *autoencoder* e vimos que, mesmo assim, este filtro apresenta melhores resultados. Em comparação com o PCA, este

modelo também aparenta ser melhor pois a percentagem de *outliers* é bastante inferior. Contudo, este filtro apresenta o problema de apanhar ainda alguns casos de não fraude como *outliers*, no entanto estes casos podem ser de números, como por exemplo, da UNICEF, que deviam estar numa *white list* (lista que contém números que fazem e recebem muitas chamadas mas que não são fraudulentos), mas, como durante este trabalho não houve acesso a uma tal lista, tivemos de usar este tipo de números também.

5.5 *Feature importance*

Uma parte importante do estudo de *machine learning* é saber como é que os diferentes atributos interagem. Especialmente no caso das redes neuronais, este estudo é muito importante porque, em muitos casos, as redes são complexas e pouco se sabe relativamente ao seu funcionamento [16].

Um método para calcular a *feature importance* em modelos supervisionados é chamado *connection weight method*. No entanto, uma vez que neste trabalho são usados modelos não supervisionados, iremos recorrer ao método *deep connection weight* [16], que pressupõe um modelo quase linear.

Este método tem duas fases:

1. **"Feature score matrix"**: baseia-se na multiplicação de todas as matrizes de pesos, ou seja,
$$\phi = \prod_{i=0}^{|L|} W(i),$$
onde L é o número de camadas e **W** representa as matrizes de pesos;

2. **"Contribution Score"**: indica o contributo de cada atributo somando os valores de cada coluna, isto é,
$$\phi_{f_i} = \sum_{j=1}^{H} \phi_{ij},$$
onde H é o número de neurónios da última camada [16].

Tendo em conta os modelos escolhidos anteriormente, em seguida, vão ser apresentados os resultados para cada atributo.

5.5.1 *Autoencoder* com *ranges* dos três níveis

Seguindo as fases referidas em cima, obtemos a seguinte importância de cada atributo:

A	**I**
Calls_Day_CV_Ratio_mean	1396.908
A_Rotation_Ratio_mean	739.563
Range_level_1	119.279
Range_level_2	-42.774
Interval_5M_Intensity_Ratio_mean	-14.152
Range_level_3	12.309
AB_Ratio_mean	8.370
Total_Calls_Day_Ratio_mean	-8.008
Interval_5M_Consistency_Ratio_mean	-6.857
Interval_1h_Intensity_Ratio_mean	-6.633
Interval_1H_Consistency_Ratio_mean	-1.326
Days_Consistency_Ratio	-1.320
B_dispersion_Ratio_mean	0.1926

Tabela 5.13: *Feature Importance*: modelo com as *ranges* dos três níveis.

C \ T	T_1	T_2	T_3
B	30/48	30/48	30/48
BR	98/236	95/236	91/236
W	23/25	23/25	23/25
WR	20/70	17/70	17/70
NF	33/55	29/55	25/55
NFR	60/310	55/310	53/310
TO	180359 (2,6%)	91374 (1,3%)	68903 (1,0%)

Tabela 5.14: Resultados do *autoencoder* com as *ranges* dos três níveis.

As abreviaturas da primeira coluna significam: $A = atributos$, $I = importncia$, $C = classificao$, $T = threshold$, $B = bypass$, $BeR = bypass\ em\ range$, $W = wangiri$, $WeR = wangiri\ em\ range$, $NF = no\ fraude$, $NFeR = no\ fraude\ em\ range$, $T_1 = Q_3 + 1.5(Q_3 - Q_1)$, $T_2 = Q_3 + 3(Q_3 - Q_1)$ e $T_3 = Q(0.99)$.

A importância que a *feature importance* dá a cada atributo é tratado como um "peso", isto é, invés de ser calculado o erro como acima, apenas com a média não ponderada, aqui é calculado com a média mas tendo em conta o valor da importância de cada atributo, ou seja, é feita uma média ponderada. Assim, a importância desta análise é que cada valor de erro por atributo tenha a sua devida importância e, desta forma, conseguimos saber quais os atributos mais importantes.

Ao compararmos os resultados obtidos com a *feature importance* e os resultados obtidos na tabela 5.3, vemos que a maior diferença está no número de *outliers*. Desta forma, podemos considerar que fazer uma média não ponderada é melhor para classificar *outliers*. Mais uma vez, aqui existe um grande número de não fraudes classificadas como fraude que pode vir do facto de haverem números de grandes organizações mundiais.

5.5.2 *Autoencoder* com *ranges* de cada nível em separado

Tal como foi feito anteriormente, aqui também vai ser realizada uma análise por cada *range* de cada nível em separado.

5.5.2.1 *Autoencoder* com *ranges* de nível 1

Nesta secção foram seguidos os passos descritos acima para calcular a *feature importance* de cada atributo e usados para classificar *ranges* de nível 1.

A	**I**
Calls_Day_CV_Ratio_mean	165.859
A_Rotation_Ratio_mean	142.753
AB_Ratio_mean	-21.888
B_dispersion_Ratio_mean	-10.365
Interval_5M_Intensity_Ration_mean	-6.235
Total_Calls_Day_Ratio_mean	5.835
Days_Consistency_Ratio	4.932
Interval_1h_Intensity_Ration_mean	4.230
Interval_5M_Consistency_Ration_mean	-3.542
Interval_1H_Consistency_Ration_mean	1.215

Tabela 5.15: *Feature Importance*: modelo com as *ranges* de nível 1.

C \ T	T_1	T_2	T_3
B	13/26	13/26	13/26
BR	85/171	85/171	87/171
W	13/14	13/14	13/14
WR	18/48	18/48	18/48
NF	0/3	0/3	0/3
NFR	60/257	60/257	60/257
TO	22586 (0,8%)	22198 (0,8%)	28834 (1,0%)

Tabela 5.16: Resultados do *autoencoder* com as *ranges* de nível 1.

Igualmente como aconteceu acima, com *ranges* de nível 1 os resultados são muito mais favoráveis do que os apresentados na tabela 5.4, porque apesar de apanhar praticamente os mesmos casos classificados, a percentagem de *outliers* é muito inferior.

5.5.2.2 *Autoencoder* com *ranges* de nível 2

Em seguida, serão apresentados os resultados da *feature importance* com *ranges* de nível 2.

A	**I**
B_dispersion_Ratio_mean	219.509
Calls_Day_CV_Ratio_mean	165.798
Total_Calls_Day_Ratio_mean	-131.331
Interval_5M_Intensity_Ration_mean	-121.015
Interval_1h_Intensity_Ration_mean	68.059
Interval_5M_Consistency_Ration_mean	42.663
Interval_1H_Consistency_Ration_mean	36.817
AB_Ratio_mean	-6.074
Days_Consistency_Ratio	-4.316
A_Rotation_Ratio_mean	1.517

Tabela 5.17: *Feature Importance*: modelo com as *ranges* de nível 2.

C \ T	T_1	T_2	T_3
B	13/17	13/17	13/17
BR	16/51	16/51	12/51
W	8/8	8/8	8/8
WR	6/19	4/19	1/19
NF	39/50	39/50	29/50
NFR	1/51	1/51	0/51
TO	269706 (10,8%)	217389 (8,7%)	25056 (1,0%)

Tabela 5.18: Resultados do *autoencoder* com as *ranges* de nível 2.

Se analisarmos a tabela 5.5 e a compararmos com a tabela acima, vemos que ao contrário das anteriores, esta não mostrou uma grande melhoria uma vez que não baixou substancialmente a percentagem de *outliers* e o número de não fraudes subiu.

5.5.2.3 *Autoencoder* com *ranges* de nível 3

Nesta subsecção são apresentados os resultados de usar a *feature importance* com *ranges* de nível 3.

A	**I**
Calls_Day_CV_Ratio_mean	110.803
A_Rotation_Ratio_mean	86.399
Interval_1h_Intensity_Ratio_mean	-3.989
Total_Calls_Day_Ratio_mean	1.028
Interval_5M_Intensity_Ratio_mean	0.971
Interval_1H_Consistency_Ratio_mean	-0.687
AB_Ratio_mean	-0.450
Interval_5M_Consistency_Ratio_mean	0.420
B_dispersion_Ratio_mean	0.347
Days_Consistency_Ratio	0.147

Tabela 5.19: *Feature Importance*: modelo com as *ranges* de nível 3.

T / C	T_1	T_2	T_3
B	5/5	5/5	5/5
BR	2/14	2/14	1/14
W	3/3	3/3	1/3
WR	0/3	0/3	0/3
NF	2/2	2/2	1/2
NFR	0/2	0/2	0/2
TO	317969 (21,3%)	315192 (21,1%)	14960 (1,0%)

Tabela 5.20: Resultados do *autoencoder* com as *ranges* de nível 3.

Tal como no caso das *ranges* de nível 2, se compararmos estes resultados com os da tabela 5.6 vemos que não apresentam melhorias uma vez que as classificações são praticamente as mesmas mas que a percentagem de *outliers* aumentou.

5.6 Perfis

Tal como foi dito anteriormente, o PCA parece tratar a não fraude como um problema linear e a fraude como um problema não linear. Perante esta hipótese, surgiu a ideia de procurar identificar quais são os neurónios ativados quando uma *range* é fraudulenta e quais os neurónios ativados quando uma *range* não é fraudulenta.

Um neurónio é ativado quando no vetor resultante da multiplicação de um vetor por uma matriz de pesos aparece um número negativo. Anteriormente, somámos os números negativos. No entanto, na presente análise, interessa conhecer as posições desses números negativos, porque nos indicarão quais são os neurónios que são ativados.

Com os dados classificados disponíveis, conseguimos criar 13 perfis diferentes, isto é, 13 maneiras diferentes de ativação dos neurónios. A ideia é usar estes perfis para categorizar as *ranges*, uma vez que se consegue colocar 128 *ranges* classificadas em apenas 13 perfis diferentes.

Por exemplo, na figura 5.2 podemos ver um dos perfis identificados, onde cada neurónio pintado corresponde a uma ativação do mesmo.

Na figura 5.3 podemos ver o gráfico de barras que corresponde aos 13 perfis identificados nos dados classificados.

Apesar de não se observar total separação daquilo que é fraude e do que não é, podemos verificar que existem perfis com mais tendência para fraude (por exemplo perfil 3 e 4) e existem perfis com mais tendência para não fraude (por exemplo perfil 13 e 12). No entanto, todo este estudo de perfis é relativamente preliminar: o seu desenvolvimento mais aprofundado requer mais dados classificados.

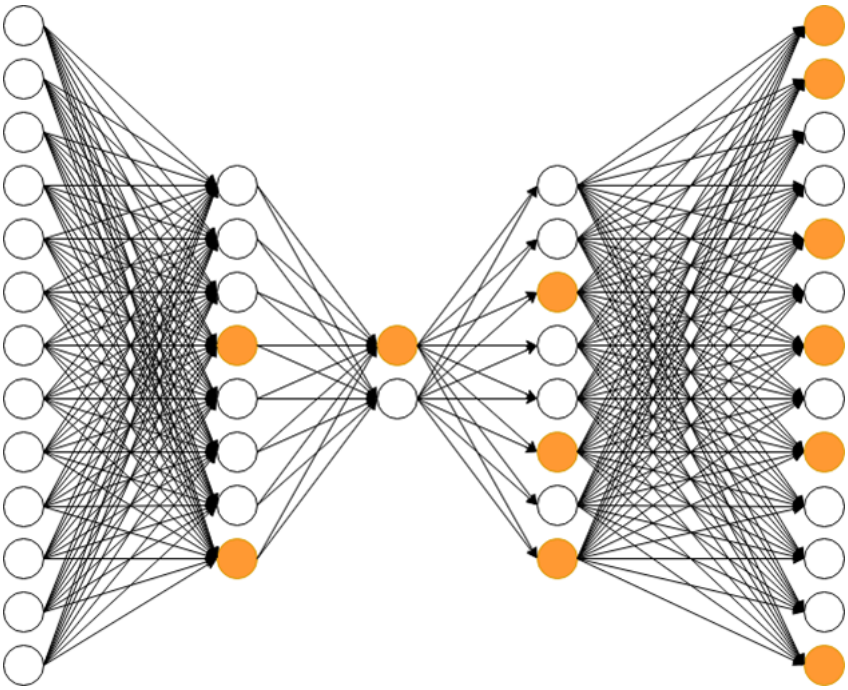


Figura 5.2: Exemplo de um perfil ([3,7];[0];[2,5,7];[0,1,4,6,8,12]), onde a laranja se encontram os nós ativados.

Podemos ver nas figuras 5.4 e 5.5 exemplos do perfil 3 e 6 definidos na figura 5.3, que são perfis com mais tendência para a fraude.

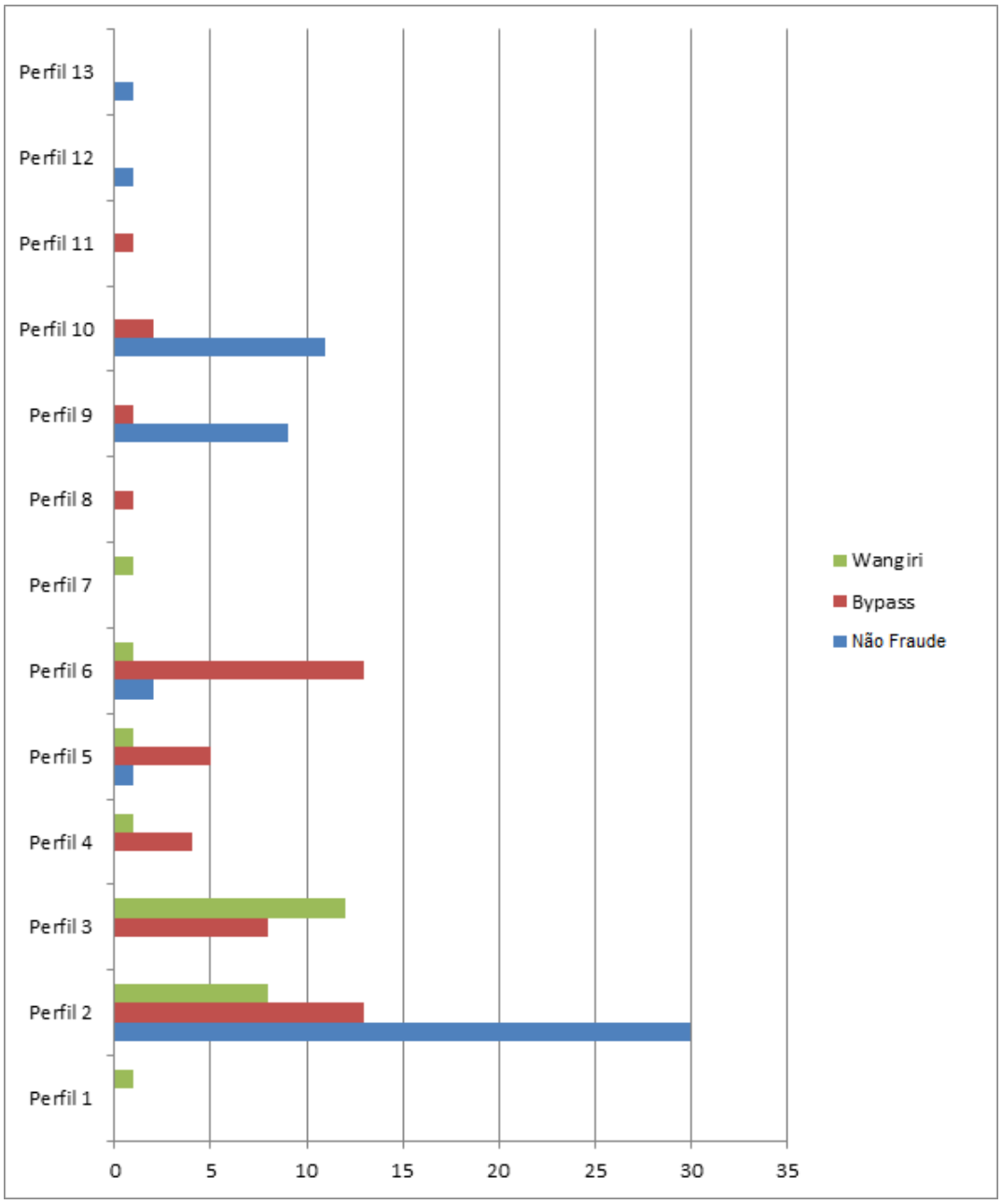


Figura 5.3: Gráfico dos perfis.

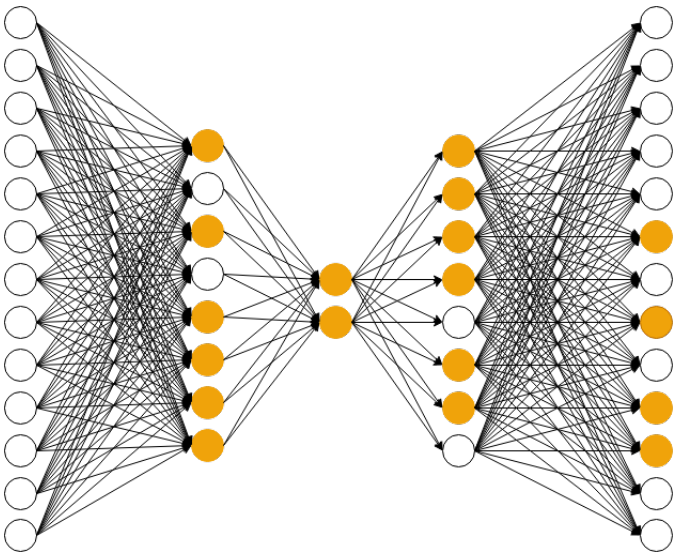


Figura 5.4: Perfil 3: ([0,2,4,5,6,7];[0,1];[0,1,2,3,5,6];[5,7,9,10])

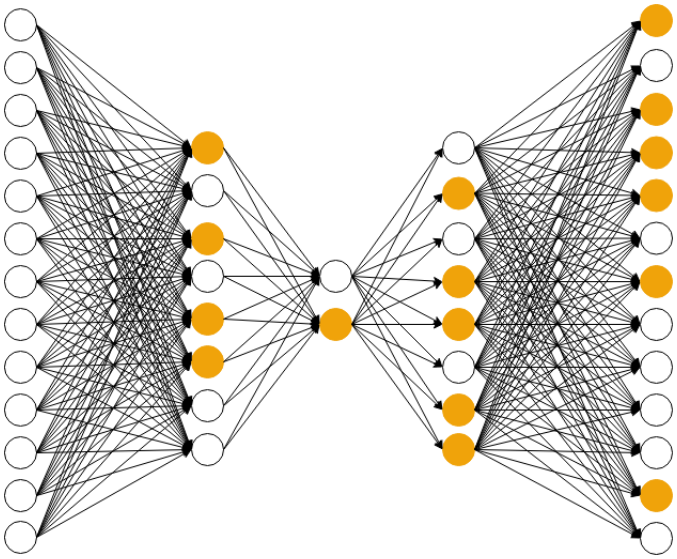


Figura 5.5: Perfil 6: ([0,2,4,5];[1];[1,3,4,6,7];[0,2,3,4,6,11])

Capítulo 6

Modelo LSTM *Autoencoder*

6.1 Introdução

No capítulo anterior foi descrito o que são *autoencoders* e como podem ser aplicados no problema da deteção de fraude em telecomunicações. No entanto, os *autoencoders* tem dois problemas quando lidam com dados sequenciais. Um é pelo facto de que, numa ordem sequencial, podemos nem sempre ter o mesmo tamanho de *inputs* e o outro é pelo facto de os *autoencoders* não terem em conta a ordem temporal.

As LSTM (*long short-term memory*) são redes neuronais que foram desenhadas para receberem dados sequenciais como *input*. Estas redes conseguem aprender sobre uma linha temporal e guardar na memória as informações importantes. Assim, uma LSTM *autoencoder* é uma rede neuronal para *autoencoders* com dados sequenciais, isto é, consiste na implementação de um *autoencoder* para dados sequenciais, com a junção de uma arquitetura de LSTM na parte de *encoding* e *decoding* do *autoencoder* [1].

As primeiras camadas deste tipo de arquitetura, *encoding*, fazem uma compressão dos dados de *input*. Em seguida, é usada uma camada de *repeat vector* de forma a distribuir os dados comprimidos pelos vários passos temporais do *decoder*. Por último, vem a camada de *decoder* que tenta reconstruir o *input* [14].

Uma das primeiras vezes que este tipo de redes foi usado foi em 2015, num modelo sem supervisão mas para processamento de imagens de vídeo [21]. Aqui os autores usaram a parte de *decoding* de duas formas, uma para prever o futuro e outra para reconstruir o erro. No caso desta dissertação foi apenas usada a parte relativa à reconstrução do erro.

A aplicação deste tipo de redes neuronais nesta dissertação deveu-se ao facto de poder haver informações escondidas na linha temporal que o *autoencoder*, por si só, poderá não conseguir extrair.

6.2 Aplicação da LSTM *Autoencoder*

Os dados que entram como *input* em redes neuronais LSTM *autoencoder* não são como os anteriores, isto é, como os dados usados no *autoencoder*, uma vez que este tipo de rede neuronal requer dados em três dimensões. Para lidar com este facto, uma primeira estratégia neste trabalho foi ter, para cada hora, uma matriz que tivesse em cada linha uma *range* e em cada coluna um atributo. No entanto, esta estratégia trazia-nos o problema do tamanho. Desta forma, houve uma reestruturação dos dados para contornar este problema, ou seja, passou-se a ter para cada *range* uma matriz em que cada linha é uma hora e cada coluna um atributo (ver figura 6.1).

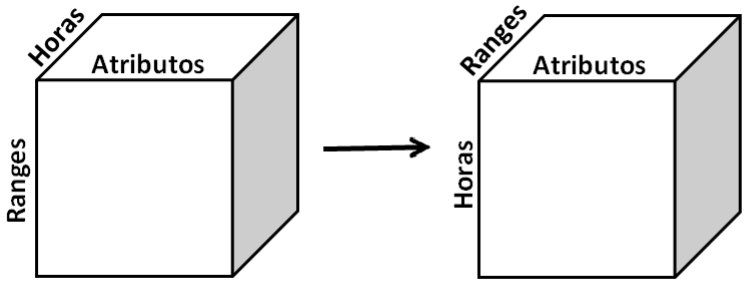


Figura 6.1: Imagem 3D do *input* das redes LSTM *autoencoder*.

Os dados que foram usados para treinar este modelo foram dados ”normais”. Quando testarmos o modelo com todos os dados (fraude e não fraude), sempre

que o modelo encontrar uma anomalia, então a reconstrução do erro terá um valor maior [14].

Por último, para calcular o erro, isto é, se um *range* é ou não fraude, foi preciso usar o *output* que continha o erro de cada *range* por cada hora em cada atributo e calcular:

1. Média dos erros por hora;
2. Média dos erros por atributo.

6.3 Resultados

De seguida, vão ser apresentados os resultados dos LSTM *Autoencoder* para cada um dos *thresholds*:

- $Q_3 + 1.5(Q_3 - Q_1)$;
- $Q_3 + 3(Q_3 - Q_1)$;
- $Q(0.99)$.

Serão apresentados os resultados para cada um dos três níveis de *ranges* em separado. Assim, vamos ter três subsecções, em que, em cada uma delas, apresentamos os resultados relativos ao estudo de cada um dos três níveis de *ranges* em separado, bem como uma comparação com os resultados obtidos no *autoencoder*.

As redes neuronais LSTM *autoencoder* foram treinadas com os primeiros 20 dias do mês de agosto e testadas com os últimos 11 dias. Deste modo, procedeu-se ao uso de *autoencoders* treinados e testados nas mesmas condições. Tal como anteriormente tinha sido feito nos *autoencoders*, também se procedeu a uma limpeza nos dados de treino relativamente a possíveis fraudes.

6.3.1 LSTM *Autoencoder* com *ranges* de nível 1

Nas tabelas seguintes são apresentados os resultados do uso de *ranges* de nível 1, primeiro com as LSTM *autoencoders* e depois com os *autoencoders.*

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	20 em 25	20 em 25	4 em 25
Bypass em *range*	80 em 139	80 em 139	23 em 139
Wangiri	14 em 14	14 em 14	8 em 14
Wangiri em *range*	21 em 52	21 em 52	4 em 52
Não fraude	3 em 5	3 em 5	0 em 5
Não fraude em *range*	5 em 93	5 em 93	0 em 93
Total de *outliers*	419 380 (11,223%)	407 373 (10,902%)	36 233 (0,970%)

Tabela 6.1: Resultados do LSTM *autoencoder* com as *ranges* de nível 1.

Classified \ Threshold	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	5 em 25	5 em 25	5 em 25
Bypass em *range*	36 em 139	3 em 139	4 em 139
Wangiri	1 em 14	1 em 14	1 em 14
Wangiri em *range*	6 em 52	2 em 52	2 em 52
Não fraude	2 em 5	0 em 5	0 em 5
Não fraude em *range*	2 em 93	0 em 93	0 em 93
Total de *outliers*	241 018 (10,896%)	14 706 (0,665%)	21 926 (0,991%)

Tabela 6.2: Resultados do *autoencoder* com as *ranges* de nível 1.

Se observarmos as tabelas 6.1 e 6.2, vemos que os resultados obtidos pelo LSTM *autoencoder* são bastante melhores. Ora, apesar de as percentagens de *outliers* serem elevadas, este modelo parece ter muita mais potencialidade do que os *autoencoders* na separação dos dados.

6.3.2 LSTM *Autoencoder* com *ranges* de nível 2

Em seguida, apresentam-se os resultados de usar as *ranges* de nível 2 para treinar uma rede neuronal LSTM *autoencoder* e outra só com um *autoencoder*.

Classified \ Threshold	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	12 em 18	12 em 18	6 em 18
Bypass em *range*	10 em 34	10 em 34	2 em 34
Wangiri	7 em 8	7 em 8	3 em 8
Wangiri em *range*	5 em 19	5 em 19	1 em 19
Não fraude	7 em 26	7 em 26	1 em 26
Não fraude em *range*	2 em 30	0 em 30	0 em 30
Total de *outliers*	269 668 (14,290%)	262 031 (13,885%)	32 599 (1,727%)

Tabela 6.3: Resultados do LSTM *autoencoder* com as *ranges* de nível 2.

Classified \ Threshold	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	13 em 18	13 em 18	13 em 18
Bypass em *range*	10 em 34	8 em 34	7 em 34
Wangiri	8 em 8	8 em 8	8 em 8
Wangiri em *range*	7 em 19	4 em 19	2 em 19
Não fraude	5 em 26	1 em 26	1 em 26
Não fraude em *range*	3 em 30	0 em 30	0 em 30
Total de *outliers*	240 364 (12,737%)	104 833 (5,555%)	18 872 (1,000%)

Tabela 6.4: Resultados do *autoencoder* com as *ranges* de nível 2.

Se compararmos as tabelas 6.3 e 6.4, vemos que a maior diferença nos resultados é o número de *outliers* identificados. Ou seja, tal como tinha acontecido na secção 5.3, treinar modelos só com *ranges* de nível 2 não é favorável.

6.3.3 LSTM *Autoencoder* com *ranges* de nível 3

Nas tabelas que se seguem estão os resultados de se treinar com as *ranges* de nível 3.

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	5 em 6	5 em 6	4 em 6
Bypass em *range*	0 em 7	0 em 7	0 em 7
Wangiri	3 em 3	3 em 3	3 em 3
Wangiri em *range*	0 em 3	0 em 3	0 em 3
Não fraude	1 em 3	1 em 3	0 em 3
Não fraude em *range*	0 em 3	0 em 3	0 em 3
Total de *outliers*	314 464 (28,819%)	254 283 (23,304%)	11 558 (1,059%)

Tabela 6.5: Resultados do LSTM *autoencoder* com as *ranges* de nível 3.

Threshold / Classified	$Q_3 + 1.5(Q_3 - Q_1)$	$Q_3 + 3(Q_3 - Q_1)$	$Q(0.99)$
Bypass	5 em 6	0 em 6	0 em 6
Bypass em *range*	0 em 7	0 em 7	0 em 7
Wangiri	3 em 3	0 em 3	0 em 3
Wangiri em *range*	0 em 3	0 em 3	0 em 3
Não fraude	2 em 3	0 em 3	0 em 3
Não fraude em *range*	0 em 3	0 em 3	0 em 3
Total de *outliers*	205 950 (18,874%)	0 (0%)	10 897 (0,999%)

Tabela 6.6: Resultados do *autoencoder* com as *ranges* de nível 3.

Tal como aconteceu com as *ranges* de nível 2, com as *ranges* de nível 3 a diferença está na percentagem de *outliers*. De uma forma global, podemos dizer que, tal como aconteceu com os *autoencoders* no capítulo anterior, as *ranges* de nível 2 e 3 em separado não apresentam muito bons resultados, no entanto, o treino com as *ranges* de nível 1 apresentam bons resultados que, como referido atrás, parecem ser melhores que os resultados obtidos pelo *autoencoder*. Uma vez que os *autoencoders* com as *ranges* dos três níveis juntas também produziam bons resultados, presumimos que se treinássemos uma rede neuronal LSTM *autoencoder* com as três *ranges* juntas poderíamos encontrar melhores que os obtidos com o *autoencoder* treinado com as três *ranges* juntas.

Capítulo 7

Conclusão

Esta dissertação teve como objetivo detetar fraude em telecomunicações através de *machine learning*. Para a realização da mesma foram necessários estudos prévios sobre o que é a fraude nas telecomunicações e como pode ser detetada, sobre *outliers* e como podem ser detetados e, por fim, sobre várias técnicas de *machine learning*.

A primeira fase prática desta dissertação teve a ver com o estudo do *dataset* fornecido pela WeDo, onde houve uma fase de exploração dos dados e uma de criação de atributos a serem usados posteriormente em diferentes modelos de *machine learning*. Em particular, foram criados atributos *range* e nível de *range* que permitiram agrupar números telefónicos e desenpenharam um papel essencial nos vários modelos desenvolvidos ao longo do trabalho.

As técnicas de *machine learning* usadas ao longo de todo este trabalho foram técnicas sem supervisão: PCA, *autoencoders* e LSTM *autoencoders*.

O modelo PCA foi aplicado de duas formas diferentes: com as *ranges* dos três níveis em conjunto e com cada uma dos três níveis de *ranges* em separado. Os resultados obtidos com o modelo PCA leva-nos a ponderar que a não fraude possa ser estudada como um problema linear, uma vez que o PCA consegue distinguir de forma razoável o que é ou não fraude.

O segundo modelo aplicado foram as redes neuronais *autoencoders*. Este modelo

foi também aplicado de duas formas distintas: com as *ranges* de três níveis juntas e com cada nível de *ranges* em separado. Este tipo de modelo produziu bons resultados nos casos em que foram consideradas os três níveis de *ranges* juntas e em que se consideraram apenas as *ranges* de nível 1. Em comparação com o PCA, pode dizer-se que o *autoencoder* apresenta uma percentagem de *outliers* maior, o que no caso de alguns *thresholds* é melhorado. Uma vez que os modelos PCA e *autoencoder* usados apresentaram percentagens de *outliers* elevadas, por alguns dos *thresholds* considerados, foram desenvolvidos alguns filtros para reduzir estas percentagens. O filtro que apresentou melhores resultados baseou-se num *score* que reflete a necessidade de usar funções de ativação no *autoencoder*, cuja utilização se pode associar a não linearidades e a *outliers*. Sobre o modelo de *autoencoder* que desenvolvemos fizemos um pequena análise de *feature importance*. Desta análise pode dizer-se que apesar da percentagem de *outliers* diminuir usando este método, o número de falsos positivos aumenta. Sobre o modelo do *autoencoder* desenvolvemos ainda a ideia de perfil. Um perfil corresponde a uma certa forma de ativação de neurónios da rede. Esta parece ser uma ideia interessante uma vez que certos perfis parecem estar associados a comportamentos fraudulentos.

O último modelo estudado foi o LSTM *autoencoder* que é um modelo com uma utilização mais recente e foi então usado para se poder comparar a sua efetividade com o *autoencoder*, ou seja, se a sua capacidade de guardar os dados em memória torna este modelo mais favorável. Os resultados obtidos parecem sugerir que há ganhos efetivos com o modelo LSTM *autoencoder*, em especial com as *ranges* de nível 1, embora não tivéssemos tido oportunidade de correr o nosso modelo LSTM *autoencoder* com as *ranges* dos três níveis juntas. Podemos dizer que o modelo LSTM *autoencoder* tem uma grande potencialidade. No entanto, seria preciso mais poder computacional, bem como mais dados classificados, para se poder avaliar melhor o potencial deste modelo.

Terminamos, relatando algumas das dificuldades com que nos deparámos neste trabalho. Um dos problemas com que nos deparámos foi o de saber se os dados que nos foram disponibilizados teriam informação suficiente para detetar o que é

fraude. Uma vez que o *dataset* fornecido provinha de dados que chegam a uma *gateway*, estes continham poucos atributos. Por esta razão, foi necessário criar novos atributos que poderiam trazer mais informação. Outro problema com este *dataset* era não existir uma lista que nos indicasse se os números eram de apoio ao cliente ou de grandes organizações, que são números que podem trazer algum engano no treino do modelo. Assim, alguns dos números não fraude que são sinalizados nos modelos podem cair numa destas duas classes. Uma outra dificuldade que retiramos ao longo de todo o trabalho foi a existência de um número reduzido de dados classificados. Esta dificuldade não nos permitiu retirar conclusões mais fundamentadas acerca dos vários modelos que desenvolvemos para a deteção de fraude nas telecomunicações.

Bibliografia

[1] *A Gentle Introduction to LSTM Autoencoder*. https://machinelearningmastery.com/lstm-autoencoders/. Accessed: 2019-08-29.

[2] *Artificial Intelligence (AI) vs. Machine Learning vs. Deep Learning*. https://skymind.ai/wiki/ai-vs-machine-learning-vs-deep-learning. Accessed: 2019-03-26.

[3] *Autoencoders in Machine Learning*. https://prateekvjoshi.com/2014/10/18/autoencoders-in-machine-learning/. Accessed: 2019-03-27.

[4] *Box Whisker Plots*. https://www.shmoop.com/basic-statistics-probability/box-whisker-plots.html. Accessed: 2019-07-01.

[5] Varun Chandola, Arindam Banerjee e Vipin Kumar. "Anomaly detection: A survey". Em: *ACM computing surveys (CSUR)* 41.3 (2009), p. 15.

[6] *Coefficient os variation CV*. https://www.insee.fr/en/metadonnees/definition/c1366. Accessed: 2019-07-01.

[7] *Common Loss functions in machine learning*. https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23. Accessed: 2019-03-27.

[8] Abdikarim Hussein Elmi, Subariah Ibrahim e Roselina Sallehuddin. "Detecting sim box fraud using neural network". Em: *IT Convergence and Security 2012*. Springer, 2013, pp. 575–582.

[9] *Fraud costs telecoms industry $17bn a year*. http://ft.com/content/3b967efa-d6cb-11e8-ab8e-6be0dcf18713. Accessed: 2019-09-04.

[10] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* "O'Reilly Media, Inc.", 2017.

[11] *How to Select Activation Function for Deep Neural Network.* `https://engmrk.com/activation-function-for-dnn/`. Accessed: 2019-03-26.

[12] Ibrahim Ighneiwa e Hussamedin Mohamed. "Bypass Fraud Detection: Artificial Intelligence Approach". Em: (nov. de 2017).

[13] *Interquartile Ranges Outliers.* `https://www.purplemath.com/modules/boxwhisk3.htm`. Accessed: 2019-07-01.

[14] *LSTM Autoencoder for Anomaly Detection.* `https://towardsdatascience.com/lstm-autoencoder-for-anomaly-detection-e1f4f2ee7ccf`. Accessed: 2019-10-03.

[15] *Machine Learning.* `https://www.sas.com/pt_br/insights/analytics/machine-learning.html`. Accessed: 2019-03-26.

[16] Jim O'Donoghue, Mark Roantree e Andrew McCarren. "Detecting feature interactions in agricultural trade data using a deep neural network". Em: *International Conference on Big Data Analytics and Knowledge Discovery.* Springer. 2017, pp. 449–458.

[17] Salima Omar, Asri Ngadi e Hamid H Jebur. "Machine learning techniques for anomaly detection: an overview". Em: *International Journal of Computer Applications* 79.2 (2013).

[18] *Principal Component Analysis for Dimensionality Reduction.* `https://towardsdatascience.com/principal-component-analysis-for-dimensionality-reduction-115a3d157bad`. Accessed: 2019-09-23.

[19] *Principal omponent nalysis.* `https://www.nature.com/articles/nmeth.4346.pdf`. Accessed: 2019-07-03.

[20] Roselina Sallehuddin, Subariah Ibrahim, Abdikarim Hussein Elmi et al. "Classification of sim box fraud detection using support vector machine and artificial neural network". Em: *International Journal of Innovative Computing* 4.2 (2014).

[21] Nitish Srivastava, Elman Mansimov e Ruslan Salakhudinov. "Unsupervised learning of video representations using lstms". Em: *International conference on machine learning*. 2015, pp. 843–852.

[22] *Standard Deviation and Variation*. `https://www.mathsisfun.com/data/standard-deviation.html`. Accessed: 2019-07-01.

[23] Pooja Thakkar, Jay Vala e Vishal Prajapati. "Survey on outlier detection in data stream". Em: *Int. J. Comput. Appl* 136 (2016), pp. 13–16.

[24] *Wangiri Fraud on the rise in UAE*. `https://blog.wedotechnologies.com/wangiri-fraud-on-the-rise-in-uae`. Accessed: 2019-03-25.

[25] *What is Machine Learning? A definition*. `https://www.expertsystem.com/machine-learning-definition/`. Accessed: 2019-09-04.