



UNIVERSIDADE  
LUSÓFONA

# Machine Learning para deteção de fraude online de cartões bancários

Catarina Silva - 21705400

Diego Barros - 21701443

Professor Orientador: Tiago dos Santos

Trabalho Final de Curso | LEI | 04/09/2020

[www.ulusofona.pt](http://www.ulusofona.pt)

## Índice

Resumo .....	5
Abstract.....	6
1. Identificação do Problema .....	7
1.1 Introdução e Contexto.....	7
1.2 Fraude de transações de cartões bancários.....	8
2. Levantamento e Análise dos Requisitos.....	10
3. Viabilidade e Pertinência .....	12
3.1 Machine Learning na deteção da fraude .....	12
4. Solução Desenvolvida .....	14
4.1 Métodos de <i>Machine Learning</i> a explorar .....	14
4.2 O dataset: IEEE-CIS Fraud Detection .....	14
4.3 Análise e Processamento dos Dados.....	15
4.4 Escolha dos Modelos a Utilizar .....	19
4.4.1 Regressão Linear .....	20
4.4.2 Regressão Logística.....	22
4.4.3 Árvores de Decisão .....	23
4.4.4 Floresta Aleatória .....	24
4.4.5 Outros modelos não utilizados .....	25
4.5 Aplicação dos Modelos e Importância das Variáveis .....	25
4.6 Simulação das transações em tempo-real .....	28
4.6.1 Kafka .....	28
4.6.2 Zookeeper.....	29
4.6.3 Docker .....	29
4.6.4 PostgreSQL .....	30
5. Benchmarking.....	33
5.1 Ravelin .....	33
5.2 Feedzai .....	33
5.3 Stripe.....	33
6. Método e Planeamento .....	35
7. Resultados.....	36
8. Conclusão e Trabalhos Futuros .....	39
Bibliografia.....	40

Anexos.....	42
Glossário.....	43

## Índice de Figuras

Figura 1 - Diversidade dos dados .....	16
Figura 2 - Redução de Memória Utilizada .....	17
Figura 3 - Novo Ficheiro .....	17
Figura 4 - Variáveis com Valores Não Numéricos.....	17
Figura 5 - Variáveis com Valores em Falta .....	18
Figura 6 - Preenchimento de Valores em Falta .....	18
Figura 7 - Preparação dos Dados para Modelo .....	19
Figura 8 - Exemplo de Regressão Linear .....	20
Figura 9 - Função Logística.....	22
Figura 10 - Exemplo de Árvore de Decisão .....	23
Figura 11 - Exemplo de Floresta Aleatória .....	24
Figura 12 - Exemplo de Clustering .....	25
Figura 13 - Aplicação de Modelos .....	26
Figura 14 - Métricas de Performance .....	27
Figura 15 - Estrutura de um Tópico.....	28
Figura 16 - Interação entre Produtores e Consumidores .....	29
Figura 17 - Docker vs Máquina Virtual .....	30
Figura 18 - Tabela no PostgreSQL.....	31
Figura 19 - Conexão entre PostgreSQL e Python.....	31
Figura 20 - Passagem de Resultados para Base de Dados.....	32
Figura 21 - Dados de Fraude e ID da Transação .....	32
Figura 22 - Machine Learning vs Modelo de Regras .....	34

## Resumo

Este trabalho tem como objetivo auxiliar a detecção de fraudes bancárias, de forma a reduzir o prejuízo de empresas que possam ser afetadas. Para isso, explora-se e aplica-se o conceito de *Machine Learning*, para se conseguir fazer uma detecção em tempo real de uma possível fraude, durante uma transação online.

Essa detecção pode ser difícil sem algum método que ajude a prever uma possível fraude, principalmente porque podem estar a ocorrer várias transações ao mesmo tempo ou em grande número, dificultando a análise das mesmas.

Um processo mais automatizado e preciso desta tarefa de detecção de fraude contribui, não só para a triagem destas transações para que os analistas possam investigar melhor cada uma das possíveis fraudes, como, também, para a segurança de compras online, que é algo cada vez mais utilizado, dando mais conforto aos utilizadores desses serviços de compras.

## Abstract

Shopping online is becoming increasingly popular nowadays. Its popularity and the fact that it can be a person's preferred means for shopping might also instigate criminal minds, for example, by finding a way to use someone else's credit card information. For this reason, it's necessary to ensure the security of online transactions, and part of that includes fraud detection methods. With many transactions occurring, possibly at the same time, it can be difficult to analyze and understand whether a transaction is fraudulent or not.

To help with this, Machine Learning can be used to detect frauds in real time, during the transaction, which is the main goal of this project, since it automatizes this process by using available data to predict the probability of fraud. This benefits not only a customer, but also a company, as it can prevent money loss and improve financial performance.

# 1. Identificação do Problema

O objetivo deste projeto é o desenvolvimento de uma solução de *Machine Learning* que consiga detectar fraude num *stream* de dados, com a correspondente monitorização de dados que passam pelo *stream* em comparação com os dados que foram analisados aquando da criação do modelo de *Machine Learning*.

## 1.1 Introdução e Contexto

A fraude de cartões bancários é um problema que afeta a indústria em grande escala, apesar de ser apenas 0.01% do volume de transações globais, onde se estima que por cada 1\$ associado a transações de fraude exista um custo direto à indústria de 3.13\$, e em 2015 o volume calculado de fraude foi cerca de 1 800 000 000 €.

Existem várias empresas como a **Ravelin**, **Feedzai**, **Fico** e **Stripe** que comercializam produtos para ajudar bancos e vendedores a prevenirem fraude. Devido ao cariz multivariado das características da fraude, a aplicação de técnicas de *Machine Learning* tem-se mostrado bem-sucedidas na deteção em tempo real de fraudes em transações, e é uma das muitas aplicações onde se verifica que as técnicas de *Machine Learning* ajudaram a dinamizar e a melhorar a performance financeira destas empresas. Consequentemente, ajuda-se a que os cartões de débito e/ou crédito dos clientes estejam mais seguros.

Uma das disciplinas da Inteligência Artificial que mais evoluiu nestes últimos anos no contexto de adoção empresarial foi o *Machine Learning*. Esta evolução deve-se ao aumento do poder computacional assim como à existência de mais dados – fatores que todos os anos aumentam.

Estes dois fatores permitem a generalização de padrões e a compreensão destes, juntando dados e algoritmos que fazem com que a execução de um conjunto de tarefas possa ser automatizada, normalmente com uma margem de erro associada.

Sendo esta uma disciplina que requer compreensão computacional e matemática como estatística, algébrica e analítica, trabalhar neste contexto implica fazê-lo numa equipa cujo catálogo de conhecimentos sejam heterogéneos, visto que será de elevada importância perceber estes domínios.

Neste contexto, este projeto irá apresentar três tipos de conhecimento, sendo que os dois principais são:

- Um perfil de modelação matemática de dados e de aplicação de métodos estatísticos e/ou de *machine learning* – com o objetivo de fazer a modelação de deteção de fraude;
- Um perfil de modelação computacional de dados e de engenharia e arquitetura de software aplicada a dados – com o objetivo de implementar **MVP** (*Minimum Viable Product*) que dê a conhecer o contexto em que estas empresas trabalham;

Finalmente, existirá um terceiro perfil de conhecimento:

- Orientado ao negócio de como é que a fraude de cartões bancários acontece – com o objetivo de perceber como modelar, interpretar e o que extrapolar dos dados.

## 1.2 Fraude de transações de cartões bancários

Explicando com mais detalhe, a fraude de transações de cartões bancários pode acontecer em duas grandes tipologias: (a) quando o cartão está fisicamente presente na transação, e (b) quando o cartão não está fisicamente na transação, mas a sua informação é utilizada - como por exemplo uma transação online. O problema a resolver neste trabalho é sobre fraude cujo cartão não está fisicamente presente, mais especificamente em fraude no contexto de transações online - *Online Fraud* (**OF**).

A **OF**, numa visão simplista, envolve alguém que conseguiu obter o número do cartão de crédito de forma indevida (designado por *fraudster*) que utiliza essa informação para efetuar transações não autorizadas pelo detentor do cartão. Por exemplo, um *fraudster* pode comprar um item de alto valor - por exemplo um computador - a uma empresa online por 1000€ e depois vender no *Ebay* ou *OLX* por 600€. O verdadeiro dono do cartão irá, eventualmente, dar por falta do valor e assim descobrir que foi feita uma transação não autorizada e poderá depois fazer queixa no seu banco, iniciando um processo de disputa - para se apurar o que aconteceu e em real caso de fraude, perceber que parte deve ser culpabilizada: se o banco, se a empresa que vendeu o computador, ou outra parte envolvente no processo de pagamento. Se o banco do detentor do cartão decidir que a transação foi realmente fraudulenta, então o detentor do cartão é ressarcido da quantidade envolvida na fraude - sendo o custo desta fraude o próprio valor dela mais

os valores envolvidos no processo de disputa. No contexto das compras online, as empresas que lidam com pagamentos têm motivos para estarem preparadas para este tipo de ataques - e quanto maior o volume das empresas, mais peso ganha esse motivo. Neste contexto, as empresas que vendem os produtos online têm duas opções - (a) estas empresas têm a responsabilidade de decisão sobre se devem aceitar aquela transação ou não; (b) estas empresas pagam a outras empresas que trabalhem com pagamentos para estas se responsabilizarem sobre todo o processo de pagamentos - inclusive em caso de fraude.

## 2. Levantamento e Análise dos Requisitos

No âmbito de *Machine Learning*, não é muito possível seguir a convenção daquilo que é um requisito. Isto porque normalmente, no contexto de um requisito, se discute tomadas de decisão com *stakeholders* e o funcionamento do sistema é lhes apresentado de forma a que seja perceptível para todos. Com *Machine Learning*, o foco é não tanto alterar código e/ou algoritmos, mas sim processar os dados para que se consigam enquadrar nos modelos. Para isso, e para avaliar a *performance* desses modelos, há ferramentas e métricas utilizadas que podem ser difíceis para os *stakeholders* entenderem. Além disso, a tomada de decisão é algo que será feito de forma automatizada, que é o que se pretende, e não é possível estimar previamente ou assegurar qualquer tipo de resultados. [6]

Os *stakeholders* envolvidos serão sempre quem faz as transações, a empresa associada às transações e também os bancos.

No entanto, não seguindo totalmente uma convenção, levantámos os seguintes requisitos:

1. Modelo tem de ser *Machine Learning*:
  - a. O modelo deve passar por uma fase de treino, para posteriormente ser testado e fazer previsões de forma independente.
2. Quantidade, qualidade e diversidade dos dados:
  - a. Os dados devem ter quantidade suficientemente grande, tal como consistência, relevância e credibilidade, para o melhor treino do modelo. A diversidade é também importante, ou seja, tem de haver um equilíbrio nestes dados, por exemplo, se houvesse um caso de fraude nos dados usados para treino e os restantes fossem casos de não fraude, ou vice-versa, não seria possível obter bons resultados usando *Machine Learning*.
3. Análise de várias transações ao mesmo tempo:
  - a. No contexto real, irá haver ocorrências simultâneas de transações, e por isso será necessário que todas sejam validadas.
4. Redução de previsões incorretas:
  - a. O modelo deverá minimizar o número de *False Negatives* (não detetar uma fraude) e de *False Positives* (assumir como fraude uma transação)

legítima). Será escolhido o modelo que apresente as métricas mais satisfatórias.

5. Redução de *Alert Rate*:

- a. Relacionado com o requisito 4, um *Alert Rate* reduzido significa que, idealmente, só será dado um alerta quando se tratar realmente de um caso de fraude.

6. Detecção de fraude em tempo real:

- a. O modelo deve ser suficientemente rápido para que consiga detetar uma fraude assim que esta ocorra.

7. Previsão correta quando inseridos dados novos;

- a. Evitar que o algoritmo fique demasiado modelado aos dados utilizados para o seu treino (*overfitting*), o que iria causar que fosse previsto o resultado mais provável dos casos de treino, afetando as previsões de novos dados.

8. Proteção de dados de transação e identidade:

- a. Estando a ser analisados dados pessoais, de contas bancárias e das transações, deve se assegurar que esses dados não são usados para outros fins, nem são vistos por quem não tenha permissão.

## 3. Viabilidade e Pertinência

### 3.1 Machine Learning na detecção da fraude

Quando uma empresa está a gerir a responsabilidade de aceitar pagamentos - seja esta uma empresa de *retail* ou uma empresa dedicada apenas à gestão de pagamento de vários *retailers* - o processo passa sempre pela mesma tarefa - para cada transação, tem que existir uma decisão se esta é uma transação fraudulenta ou não. Sendo que as transações fraudulentas aparecem, em média, aproximadamente 1 vez em cada 10000 transações, recorrer a métodos que consigam prever a probabilidade de uma dada transação ser fraude consegue ajudar bastante nesta tarefa.

Neste tipo de empresas, são os Analistas de Fraude que têm a responsabilidade de decidir se uma dada transação que está a ser efetuada é realmente fraude ou não - mas devido ao grande volume de transações a acontecer e sendo este um requisito de tempo real, estes Analistas de Fraude precisam de utilizar ferramentas e processos que os ajudem a desempenhar este trabalho.

Tipicamente, o que acontece neste tipo de empresas hoje em dia quando é feita uma transação é o seguinte:

1. Um sistema automático (modelo estatístico, modelo de regras, ou modelo de *machine learning*) decide se a transação deve ser aceite, ou se deve ser revista pelos analistas. Um sistema destes deverá alocar um *score* à transação (por exemplo, um número real entre 0 e 1) que representa a probabilidade de aquela transação ser fraude - onde 1 representa que o sistema tem toda a certeza de que aquela transação é fraude;
2. As transações que foram enviadas para os analistas ainda não aconteceram - estas encontram-se bloqueadas até que os analistas decidam se aquela transação é ou não fraude. As transações que no ponto anterior não são enviadas para os analistas são processadas;
3. Cabe agora ao analista decidir se uma dada transação é fraude ou não. Quantas mais transações houverem para o analista decidir, maior a quantidade de trabalho que este analista tem - portanto estes sistemas devem ser o mais precisos possível. Mas estes sistemas também devem apanhar o máximo de fraude possível, para evitar prejuízo à empresa.

Quando estes sistemas são adotados pelas empresas, estes são tipicamente submetidos a testes para que as seguintes restrições possam ser garantidas:

- Cada analista só consegue ver um determinado número de transações por dia (por exemplo, 40);
- O sistema tem que apanhar o máximo de fraude possível;
- Colocar analistas a verificar transações que nunca acabam por ser fraude não é algo que faça sentido neste contexto (mas num dado dia, se não houver fraude, é natural que o sistema não apresente nenhuma transação).

Assim, quando estes sistemas enviam uma transação para os analistas, isto é considerado um alerta (no sentido em que aquela transação sofreu um alerta). Quando um alerta está realmente associado a uma transação de fraude, este diz-se que é um *True Positive (TP)*, mas quando um alerta está associado a uma transação que não é fraudulenta, então este diz-se que é um *False Positive (FP)*. Dado que o número de transações por dia é variável, costuma-se restringir não o número de alertas por dia ou por semana do modelo, mas sim o *Alert Rate (AR)* - que é o número de alertas a dividir pelo total de transações.

No contexto deste projeto, queremos ter o máximo de TP a um dado AR baixo.

## 4. Solução Desenvolvida

### 4.1 Métodos de *Machine Learning* a explorar

Neste trabalho, tem-se como objetivo ganhar conhecimento em modelos de *machine learning*.

Como tal, durante a experimentação teremos vários métodos candidatos e iremos continuar com os que apresentarem melhores resultados.

O conjunto inicial de métodos candidatos são:

- Regressão Logística (*logistic regression*);
- Árvore da Decisão (*decision tree*);
- Floresta Aleatória de Decisão (*random forest*).

A fase de experimentação será iterativa, e irá ser composta por:

1. Compreensão de dados;
2. Limpeza de dados;
3. Divisão de dados para Treino/Teste;
4. Treino de modelos;
5. Avaliação dos modelos.

As ferramentas a utilizar neste contexto serão:

- Python;
- Jupyter Lab / Notebook;
- Pandas, Matplotlib, Seaborn, Sci-kit Learn.

### 4.2 O dataset: IEEE-CIS Fraud Detection

Será usado também um *dataset*, **IEEE-CIS Fraud Detection** [1], onde o objetivo será prever a probabilidade de uma dada transação online ser fraude ou não - sendo que esta informação está binarizada na coluna **isFraud**.

O *dataset* está dividido em dois ficheiros – identidade e transações -, que irão ser juntos através da coluna **TransactionID** existente em ambos os ficheiros. De notar que nem todas as transações têm informação sobre identidade.

A informação categórica no ficheiro de transações é:

- ProductCD;

- card1 – card6;
- addr1, addr2;
- P\_emaildomain;
- R\_emaildomain;
- M1 – M9.

A informação categórica no ficheiro de identidade é:

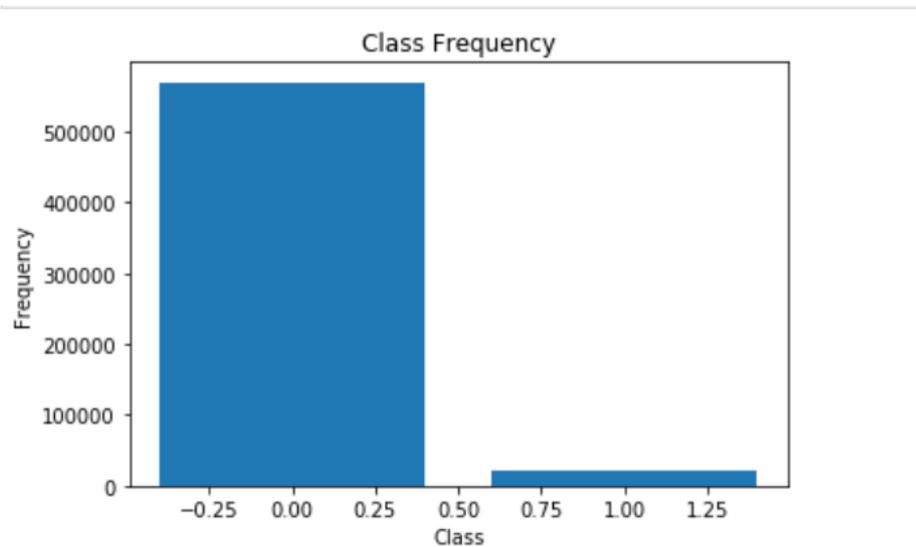
- DeviceType;
- DeviceInfo;
- id\_12 – id\_38.

Os ficheiros serão:

- train\_{transaction, identity}.csv - o conjunto de treino;
- test\_{transaction, identity}.csv - o conjunto de teste (portanto não temos a confirmação de quais são fraude ou não neste conjunto).

### 4.3 Análise e Processamento dos Dados

Numa primeira análise dos dados de treino, verifica-se que há 20663 casos fraudulentos e 569877 não fraudulentos, que leva a um rácio de cerca de 3.6%.



```
[29]: pd.crosstab(index=transaction_concat["isFraud"], columns="count")
```

```
[29]:  col_0  count
      isFraud
      ----  ----
      0.0  569877
      1.0  20663
```

Figura 1 - Diversidade dos dados

Há algum desequilíbrio em termos de diversidade, soluções poderão ser acrescentar mais dados para o caso em minoria ou remover dados do caso em maioria. No entanto, pode-se acabar por obter resultados menos corretos devido à perda ou, de certa forma, à falsificação da informação. Apesar de ser um rácio não muito elevado, continua-se a ter bastantes dados para cada caso e, por isso, não se irá fazer qualquer tipo de modificação a este nível.

O passo seguinte é processar os dados de forma a que se consiga aplicar um modelo de *machine learning*. Para isso é preciso tratar de dados em falta, filtrar classes não-numéricas e atribuir-lhes um código numérico e escolher algumas variáveis mais importantes e que mais contribuem para determinar se é mais provável um caso ser fraudulento ou não. Este passo também faz com que o algoritmo de *machine learning* seja mais rápido.

Em primeiro lugar, o elevado número de dados leva à necessidade de uma redução do uso de memória para que o processo seja computacionalmente mais viável. De uma forma sucinta, esta redução é feita mudando o tipo das variáveis, por exemplo, de *float64* para *float32*, conforme verificado na figura 2.

```
Column: id_32
dtype before: float64
min for this col: 0.0
max for this col: 32.0
dtype after: uint8
*****
__MEMORY USAGE AFTER COMPLETION:__
Memory usage is: 548.5401992797852 MB
This is 27.988510232340772 % of the initial size
```

Figura 2 - Redução de Memória Utilizada

Concluído este passo, este será o *dataset* a ser utilizado de agora em diante, e por isso é preciso guardá-lo como sendo um ficheiro que possa ser lido.

```
train.to_csv('train.csv', index=False)

train_df = pd.read_csv("train.csv")
```

Figura 3 - Novo Ficheiro

Resta agora tratar das variáveis com valores não numéricos ou em falta. Na figura abaixo exploramos a quantidade de valores não numéricos para cada coluna e na figura seguinte, os valores em falta.

```
[60]: # Explore Categorical features
print('Training set:')
for col_name in train_df.columns:
    if train_df[col_name].dtypes == 'object':
        unique_cat = len(train_df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))

print("*****")

print('Test set:')
for col_name in test_df.columns:
    if test_df[col_name].dtypes == 'object':
        unique_cat = len(test_df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))

Training set:
Feature 'ProductCD' has 5 categories
Feature 'card4' has 5 categories
Feature 'card6' has 5 categories
Feature 'P_emaildomain' has 60 categories
Feature 'R_emaildomain' has 61 categories
Feature 'M1' has 3 categories
Feature 'M2' has 3 categories
Feature 'M3' has 3 categories
Feature 'M4' has 4 categories
Feature 'M5' has 3 categories
Feature 'M6' has 3 categories
```

Figura 4 - Variáveis com Valores Não Numéricos

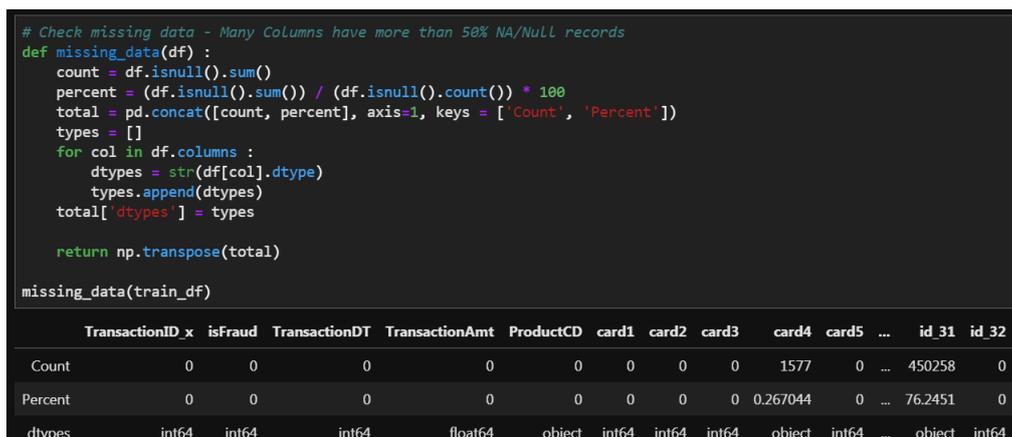


Figura 5 - Variáveis com Valores em Falta

Para preencher os dados em falta, pode ser usado mais do que um método, por exemplo, preencher com um valor arbitrário ou um valor baseado numa média de valores de outras linhas. Optou-se por preencher com um valor arbitrário e, como se pode verificar na figura, não foram encontrados valores em falta, como anteriormente.

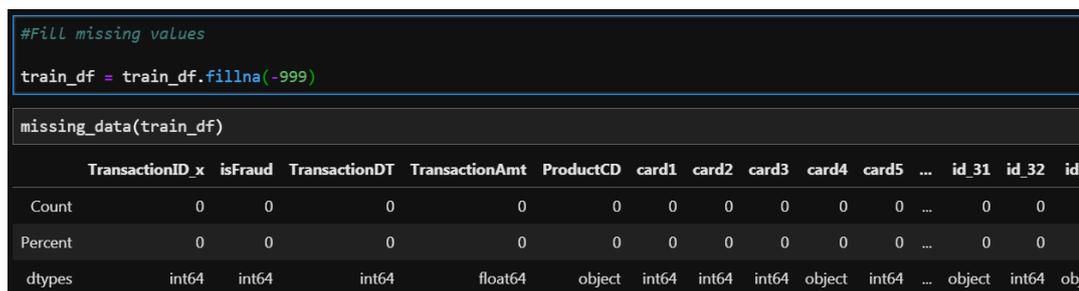


Figura 6 - Preenchimento de Valores em Falta

De seguida, separa-se a variável alvo das restantes, codificamos os valores não numéricos para que consigamos aplicar os modelos, e dividimos o *dataframe* em treino e teste. Inicialmente, possuíamos um ficheiro especificamente para o teste, isto é, que fornece dados ao modelo, depois de treinado, para prever se se trata de fraude. No entanto, esse ficheiro não nos proporciona a resposta correta, ao contrário do ficheiro de treino, pelo que não poderíamos verificar se o modelo fez uma previsão correta. Caso tivéssemos optado por criar uma nova coluna “isFraud” com valores 0 e 1 arbitrários ou aleatórios, corríamos o risco de falsificar os resultados. Sendo assim, apenas trabalharemos com o ficheiro de treino, que tem também a vantagem de conter um grande número de dados.

A divisão de treino e teste é feita de forma a que uma maior percentagem de dados seja a parte de treino, isto porque quantos mais dados forem fornecidos numa fase de treino, mais abrangente será a aprendizagem. Normalmente há uma divisão de 70-80/30-20%, no nosso caso, foi feita uma divisão de 80/20%. Todos os passos acima descritos encontram-se na figura.

De modo a obter a melhor performance, foi usado *Label Encoder* no lugar de *One-Hot Encoding*, visto que este primeiro método converte texto em valores numéricos, enquanto que o método *One-Hot Encoding* converte cada valor não numérico para uma nova coluna, assumindo assim o valor de 0 ou 1 (*true* ou *false*), consoante as linhas. Devido ao método *One-Hot Encoding* acabar por criar um elevado número de novas colunas, logo aumentando consideravelmente o tamanho do *dataframe*, foi optado ser usado *Label Encoder*.

```
train_y = train_df['isFraud']
train_X = train_df.drop('isFraud', axis=1)

for f in train_X.columns:
    #if train_X[f].dtype=='object' or test_df[f].dtype=='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(train_X[f].values))
        train_X[f] = lbl.transform(list(train_X[f].values))

#Split train into 80/20 for test
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.2)
print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)

(472432, 434) (472432,)
(118108, 434) (118108,)
```

Figura 7 - Preparação dos Dados para Modelo

Desta forma, os dados estão prontos para aplicação dos modelos de *machine learning*.

#### 4.4 Escolha dos Modelos a Utilizar

Havendo inúmeros modelos de *Machine Learning* para diversos tipos de problema, e sendo este trabalho mais focado em investigação, é importante fazer uma escolha correta de qual método usar para um melhor resultado. Assim sendo, descrever-se-á os modelos que foram escolhidos para aplicar aos dados, apresentando também uma justificação para o qual não utilizámos outro tipo de modelos.

#### 4.4.1 Regressão Linear

Modelo linear que assume uma relação linear entre as variáveis de input e a variável de output. Mais concretamente, esta variável de output pode ser calculada pela combinação linear das variáveis de input.

Existem dois tipos principais de regressão linear, regressão linear simples e regressão linear múltipla. Na regressão linear simples é usada uma única variável independente para prever o valor de uma variável numérica dependente e, na regressão linear múltipla são usadas múltiplas variáveis independentes para prever o valor de uma variável dependente.

A regressão linear tem como objetivo principal encontrar a linha de correlação que tenha menos erro entre valores previstos e valores atuais.

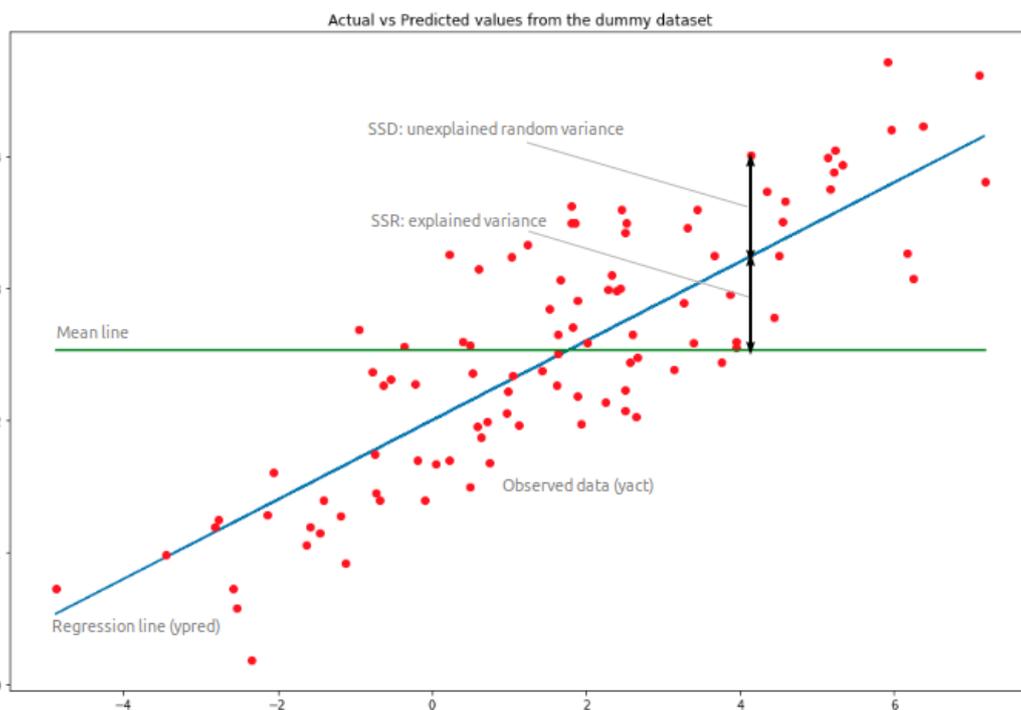


Figura 8 - Exemplo de Regressão Linear

Dependendo do ângulo da linha de regressão é possível classificar a relação linear entre as variáveis dependentes e independentes. No caso da variável dependente aumentar no eixo dos Y e a independente aumentar no eixo dos X, a relação é vista como uma relação linear positiva. No caso oposto, ou seja se a variável dependente decrescer no eixo dos Y e a independente aumentar no eixo dos X, a relação é uma relação linear negativa.

A equação linear atribui um coeficiente a cada valor de input, assim como um adicional para que a reta da equação possa ser controlada (por exemplo, mover a reta para cima ou para baixo). A otimização do coeficiente, assim como a preparação do modelo em geral, é importante para minimizar erros no mesmo, havendo várias formas de o fazer, em destaque 4, a título de exemplo. As duas formas mais utilizadas são o Gradiente Descendente e o Método dos Mínimos Quadrados.

- Gradiente Descendente - Minimiza iterativamente o erro do modelo, começando com valores de coeficiente aleatórios e, usando um parâmetro que determina o tamanho do passo de melhoria a cada iteração, os coeficientes vão sendo lentamente atualizados na direção que minimiza o erro. Este processo repete-se até que se atinja o valor mínimo de erro ou não seja possível alcançar mais melhorias. Este método é utilizado especialmente com grandes datasets.
- Método dos Mínimos Quadrados - Dada uma reta de regressão pelos dados, pretende-se obter a reta que minimiza a soma quadrática da distância entre os valores dos dados e os valores da reta. Esta abordagem trata os dados como uma matriz e usa álgebra linear para estimar os melhores valores para os coeficientes, necessitando que todos os dados estejam disponíveis e que haja memória suficiente para os mesmos, assim como para operações matriciais. Para que este método seja mais eficaz, é necessária uma boa limpeza dos dados, como por exemplo, remover variáveis de input altamente correlacionadas para evitar overfitting.
- Regressão Linear Simples - Quando há apenas uma variável de input pode se usar cálculos estatísticos, como a média, desvio padrão, correlação e covariância. Todos os dados devem estar disponíveis. Na prática, não é um método muito utilizado.

- Regularização - Extensões do treino do modelo que têm como objetivo tanto minimizar o erro, como reduzir a complexidade do modelo (como por exemplo, o número de coeficientes). Estas extensões são eficientes quando há colinearidade nos valores de input e o método dos mínimos quadrados resulta num *overfitting* dos dados de treino.

#### 4.4.2 Regressão Logística

Modelo estatístico que tem como base a função logística e como objetivo modelar uma variável dependente (alvo) binária. No nosso caso, a nossa variável alvo apenas poderá ser fraude ou não fraude, sendo por isso binária.

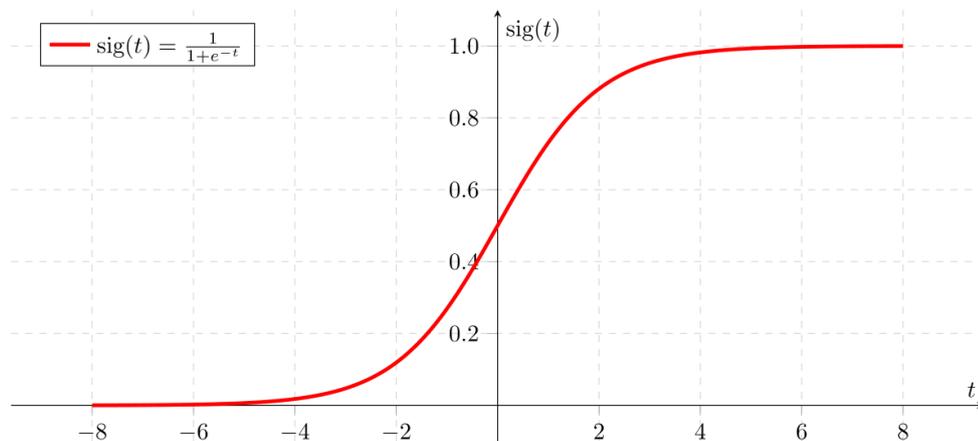


Figura 9 - Função Logística

Este modelo combina linearmente os valores de input, tendo cada um dos valores um peso/coeficiente, de forma a prever um valor de output. Esses coeficientes são estimados a partir de dados usados para treino do modelo, usando estimativa por máxima verossimilhança (*maximum-likelihood estimation*), que procura otimizar os valores dos coeficientes e, assim, minimizar o erro das probabilidades resultantes das previsões. Os melhores coeficientes serão aqueles que darão origem a um modelo que prevê um valor muito próximo de 1 para casos de fraude e um valor muito próximo de 0 para casos de não fraude.

A função logística permite que números reais sejam mapeados para um valor entre 0 e 1, e dessa forma é possível calcular a probabilidade de fraude, tendo em conta os dados

fornecidos. É, então, essa a principal diferença entre a regressão logística e a regressão linear, já que a regressão linear tem como *output* um qualquer valor numérico.

#### 4.4.3 Árvores de Decisão

Árvores de Decisão funcionam como fluxogramas em termos de estrutura, em que cada nó interno/raíz representa um teste à variável, neste caso, se é fraude ou não, podendo ter nós filhos. Os nós "folha" (nós que não possuem filhos) são a decisão tomada depois de calcular todas as variáveis, e os ramos são o conjunto de variáveis que levam a uma dada decisão.

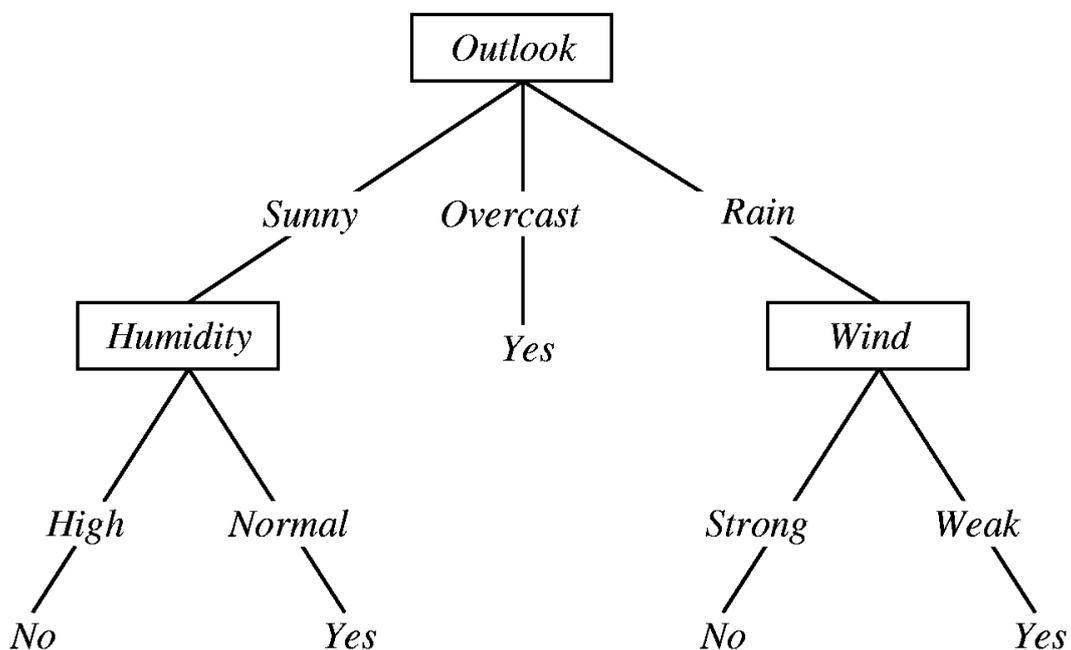


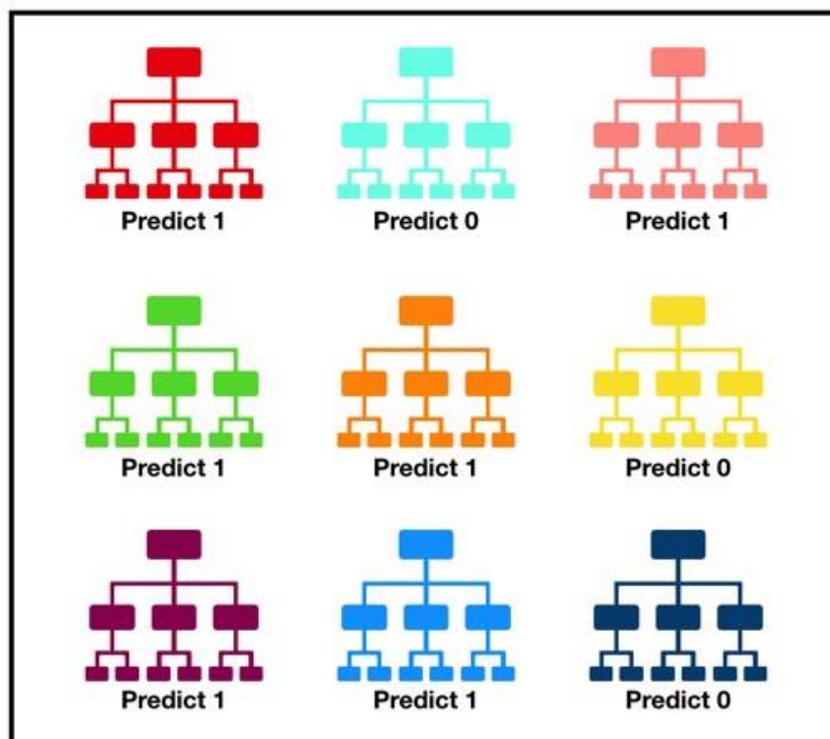
Figura 10 - Exemplo de Árvore de Decisão

Estas árvores são construídas particionando o espaço recursivamente em subconjuntos, nos quais um modelo simples é aprendido. Para uma maior eficácia do modelo, é recomendada a escolha das variáveis com maior importância, ou seja, que mais contribuem para uma determinada decisão devido à quantidade de informação que conseguem fornecer acerca das classes (fraude ou não fraude). A primeira partição será a que tiver maior ganho de informação e o processo é feito até que todos os nós filhos sejam considerados puros, isto é, pertencentes apenas a uma classe, ou até que o ganho de informação seja 0.

Os modelos de árvore de decisão são principalmente usados em análise de decisão, para ajudar a identificar a melhor estratégia de modo a atingir um certo objetivo.

#### 4.4.4 Floresta Aleatória

Este modelo é feito a partir de várias árvores de decisão. Cada árvore individual faz uma previsão de classe, e a classe com um maior número de previsões é vista como a previsão global do modelo.



Tally: Six 1s and Three 0s  
**Prediction: 1**

Figura 11 - Exemplo de Floresta Aleatória

Para um melhor funcionamento deste modelo, é necessário que as previsões feitas pelas árvores individuais tenham baixa correlação entre elas, para que não tenham todas a tendência de prever o mesmo resultado. Tal como o nome indica, o modelo adiciona aleatoriedade quando cria as árvores. Ao contrário do caso das Árvores de Decisão, que procuram as variáveis mais importantes ao fazer a partição dos nós, a Floresta Aleatória faz essa procura em subconjuntos aleatórios, o que cria uma maior diversidade e leva à geração de modelos melhores e com menor risco de *overfitting*.

Uma desvantagem deste modelo é que, se for utilizado um número elevado de árvores, que aumenta a exatidão das previsões, o processo é computacionalmente lento. Por outro lado, é um modelo bastante acessível, uma vez que origina bons resultados,

mesmo usando os parâmetros por defeito. O uso da Floresta Aleatória é bastante comum, não só em casos semelhantes ao caso em estudo, mas também, por exemplo na área da saúde, para analisar o histórico médico de um paciente, com o objetivo de identificar doenças, ou na área do comércio eletrônico para determinar se um cliente irá gostar de um produto ou não. Além disso, há uma variante deste modelo, *Extra Trees Classifier*, que funciona da mesma forma, mas adiciona mais aleatoriedade do que a Floresta Aleatória, ao utilizar variáveis aleatórias, logo não havendo nenhum tipo de procura pelas melhores variáveis, contribuindo ainda mais para a diversidade e um menor número de subconjuntos.

A Floresta Aleatória é usada para resolver problemas tanto de regressão como de classificação, tendo como objetivo a criação de várias árvores de decisão.

#### 4.4.5 Outros modelos não utilizados

Para o nosso problema, os modelos utilizados enquadram-se nos grupos de regressão (prever um atributo com valor contínuo associado a um objeto) ou classificadores (identificar a categoria a que um objeto pertence), havendo no entanto um outro tipo de modelos, de *clustering* (Figura 12), que agrupam (formam *clusters*) automaticamente objetos semelhantes, como por exemplo, segmentação de clientes, por outras palavras, agrupar clientes com características semelhantes, como localização geográfica, faixa etária, classe social, estado civil, interesses, etc. Por essa razão, não foi utilizado este tipo de modelos, visto que não foi possível adequá-lo ao caso em estudo.

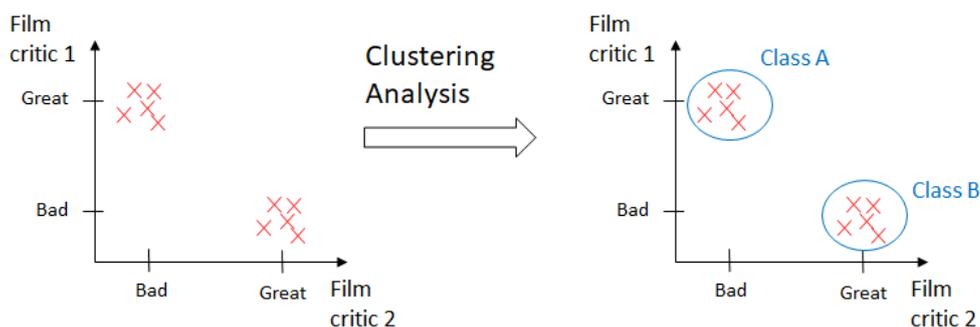


Figura 12 - Exemplo de Clustering

#### 4.5 Aplicação dos Modelos e Importância das Variáveis

A aplicação dos modelos é feita de forma semelhante e, por essa razão, apenas apresentaremos um exemplo. Os passos a concretizar são treinar o modelo com os dados de treino, prever os resultados com os dados de teste, comparar as previsões com as

respostas contidas, mas separadas, desses últimos e obter resultados de performance de cada modelo.

```
[17]: #Logistic Regression Model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

[17]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
multi_class='warn', n_jobs=None, penalty='l2',
random_state=None, solver='warn', tol=0.0001, verbose=0,
warm_start=False)

[19]: logreg.predict_proba(X_test)

[19]: array([[0.99256002, 0.00743998],
[0.99482108, 0.00517892],
[0.98702866, 0.01297134],
...,
[0.99406373, 0.00593627],
[0.99615111, 0.00384889],
[0.99189613, 0.00810387]])

[20]: logreg.score(X_test, y_test)

[20]: 0.9677498560639415
```

Figura 13 - Aplicação de Modelos

Como é observado na figura 13, o *output* obtido a partir do método “predict\_proba” é um *array* cujos valores para cada linha correspondem à probabilidade do nosso alvo ter o valor 0, ou seja, não fraude, ou valor 1, fraude. A análise destes resultados será abordada mais tarde, no capítulo Resultados.

Foram usados diversos tipos de métricas para avaliar as performance dos modelos, especificamente *Precision*, *Recall*, *MCC* e *F1*, conforme a figura 14.

```

from sklearn.metrics import f1_score, matthews_corrcoef, precision_score, recall_score

#Matthew's Correlation Coefficient
print("MCC Score for Decision Tree = ", matthews_corrcoef(y_test, dec_tree.predict(X_test)))
print("MCC Score for Random Forest = ", matthews_corrcoef(y_test, forest.predict(X_test)))
print("MCC Score for Extra Tree Classifier = ", matthews_corrcoef(y_test, extra_trees.predict(X_test)))
print("-----")

#F1 Score
print("F1 Score for Decision Tree = ", f1_score(y_test, dec_tree.predict(X_test)))
print("F1 Score for Random Forest = ", f1_score(y_test, forest.predict(X_test)))
print("F1 Score for Extra Tree Classifier = ", f1_score(y_test, extra_trees.predict(X_test)))
print("-----")

#Precision
print("Precision Score for Decision Tree = ", precision_score(y_test, dec_tree.predict(X_test)))
print("Precision Score for Random Forest = ", precision_score(y_test, forest.predict(X_test)))
print("Precision Score for Extra Tree Classifier = ", precision_score(y_test, extra_trees.predict(X_test)))
print("-----")

#Recall
print("Recall Score for Decision Tree = ", recall_score(y_test, dec_tree.predict(X_test)))
print("Recall Score for Random Forest = ", recall_score(y_test, forest.predict(X_test)))
print("Recall Score for Extra Tree Classifier = ", recall_score(y_test, extra_trees.predict(X_test)))
print("-----")

```

Figura 14 - Métricas de Performance

A *precision* é calculada através da expressão  $TP/(TP+FP)$ , e o *recall* através da expressão  $TP/(TP+FN)$ . A *precision* é a capacidade do classificador de não considerar como positivo um resultado que é, na verdade negativo, enquanto que o *recall* é a capacidade do classificador de detetar todas os resultados positivos. Os valores possíveis na *precision* e no *recall* são de 0 a 1, sendo 1 um resultado positivo e 0 um resultado negativo.

O *F1-Score* combina *precision* e *recall* relativo a uma classe positiva. O *F1-Score* pode ser interpretado como uma média da *precision* e *recall*. O *F1-Score* atinge o seu melhor valor a 1 e o pior a 0. O *MCC* (*Matthews Correlation Coefficient*) é usado para verificar a qualidade de classificadores binários. Tem em conta *true* e *false positives* e *negatives* e é considerado um método para balançar que pode ser usado mesmo que as classes tenham diferentes tamanhos. O *MCC* é interpretado com valores entre o -1 e +1, sendo que o 0 é médio, o -1 é uma previsão inversa e o +1 uma previsão perfeita.

O *F1-Score* é calculado através da fórmula:  $2 * (precision * recall) / (precision + recall)$ .

Estas métricas são calculadas apenas em modelos não-regressivos, pelo que não podemos utilizar para medir a performance dos modelos de Regressão Linear e Regressão Logística, como irá estar apresentado no capítulo Resultados.

## 4.6 Simulação das transações em tempo-real

Esta fase do trabalho consiste em produzir dados que possam ser consumidos pelos vários modelos, de forma a fazer previsões constantes, como seria expectável num contexto real. Serão descritas abaixo as ferramentas necessárias para concretizar esse objetivo.

### 4.6.1 Kafka

O Apache Kafka é um sistema distribuído de publicação e subscrição de mensagens. Este sistema guarda *streams* de mensagens em tópicos, sendo que cada mensagem contém uma chave, um valor, e um marcador temporal. Há também o conceito de produtores, que escrevem os dados nos tópicos, e de consumidores, que lêem esses mesmos dados.

Mais concretamente, um tópico é uma categoria no qual mensagens são publicadas, que pode ter um qualquer número de consumidores que subscrevem aos dados nele escritos. Para cada tópico, são mantidos *logs* divididos em partições. Sendo o Kafka um sistema distribuído, os tópicos são particionados e replicados ao longo de vários nós (figura 15).

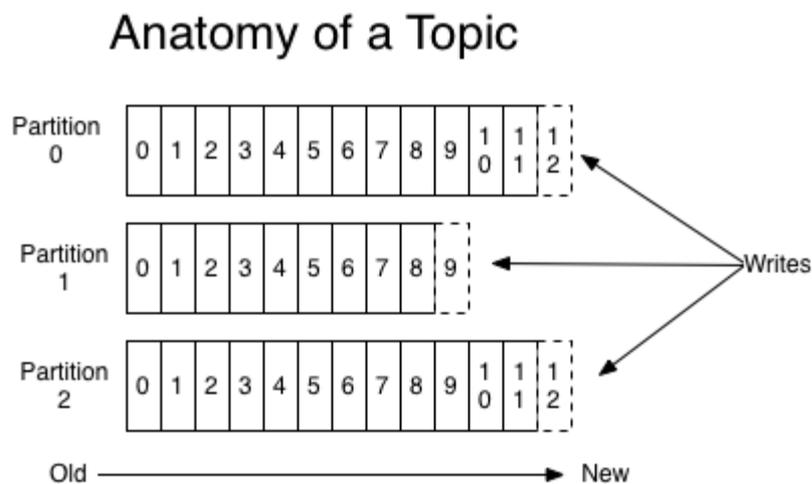


Figura 15 - Estrutura de um Tópico

Desta forma, os produtores publicam os dados para determinados tópicos, escolhendo-se a qual partição corresponderá, dentro do tópico, o que pode ser feito utilizando o método chave-valor e, assim, o produtor garante que todas as mensagens com a mesma chave pertencerão à mesma partição. As mensagens com a mesma chave são, então, entregues aos mesmos consumidores.

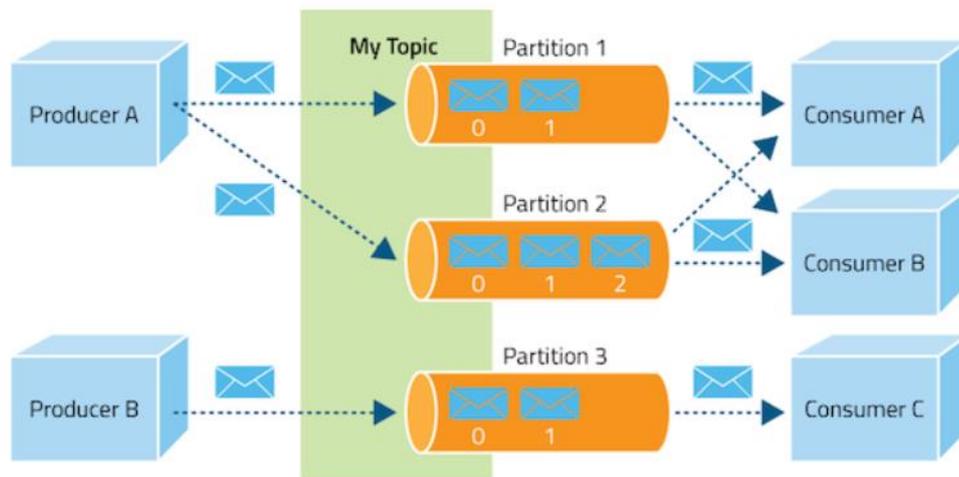


Figura 16 - Interação entre Produtores e Consumidores

Ainda no contexto do Kafka, existe um componente, Kafka Connect, que é uma *framework* com a finalidade de conectar o Kafka com sistemas externos como bases de dados, sistemas de ficheiros, etc, usando conectores, que são componentes que ajudam a importar dados desses sistemas externos para tópicos e exportar dados dos tópicos para os sistemas externos. Estes conectores são geridos usando REST API.

#### 4.6.2 Zookeeper

O Apache Zookeeper é utilizado em conjunto com o Kafka, e funciona como um serviço centralizado para manutenção de nomes e dados de configuração para uma melhor e mais flexível sincronização dentro dos sistemas distribuídos, uma vez que o Zookeeper tem em conta o estado dos nós dos *clusters* do Kafka, assim como os tópicos e partições, permitindo também simultâneas leituras e escritas por parte de múltiplos clientes. Os dados persistentes são distribuídos por vários nós, o que contribui para a consistência, já que se houver uma falha num dos nós a que um cliente se conecta, é feita uma migração para outro nó.

#### 4.6.3 Docker

Embora não seja obrigatório o uso do Docker para o funcionamento desta fase do trabalho, é uma ferramenta conveniente para criar e isolar ambientes (*containers*) independentes para lançar aplicações, com a vantagem de ser possível correr um ambiente a partir de qualquer máquina. O Docker permite que não haja problemas de dependência ou compilação, é rápido, multi-plataforma e mantém as configurações para cada ambiente.

Esta ferramenta difere de uma máquina virtual, no sentido em que, ao contrário do Docker, uma máquina virtual contém um total sistema operativo e funciona independentemente, tal como um computador. O Docker apenas partilha os recursos com a máquina anfitriã para correr os seus ambientes, tornando-o mais rápido do que uma máquina virtual, pois pode começar ou parar uma aplicação em poucos segundos.

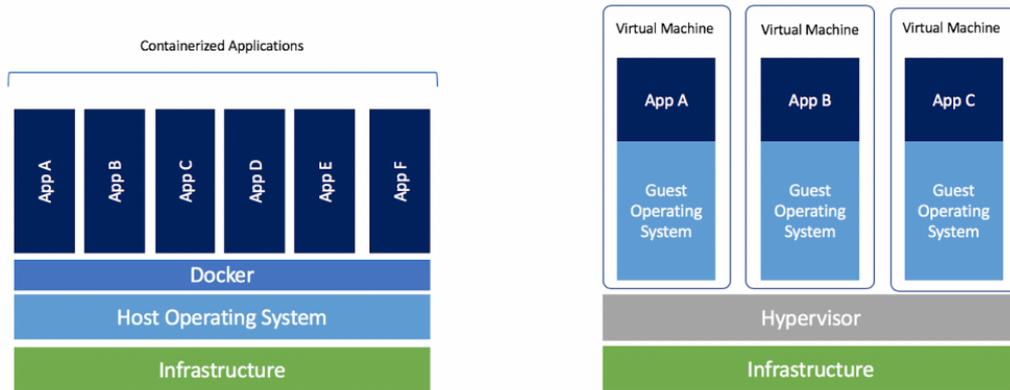


Figura 17 - Docker vs Máquina Virtual

#### 4.6.4 PostgreSQL

PostgreSQL é um sistema de base de dados relacional (*open-source*), que suporta tanto SQL (relacional) como JSON (não-relacional) *querying*. É a partir da base de dados que o Kafka, através dos conectores, terá acesso aos dados que serão produzidos para a *stream*.

Preenchemos uma base de dados a partir de um ficheiro .csv, com a ajuda de um GUI auxiliar ao PostgreSQL, “pgAdmin”. Isto permitir-nos-á posteriormente conectar essa base de dados ao Python.

	transactionid_x text	transactionid_t text	transactionamt text	productcd text	card1 text	card2 text	card3 text	card4 text	card5 text	card6 text	addr1 text	addr2 text
1	3663549	18403224	31.95	W	10409	111	150	visa	226	debit	170	87
2	3663550	18403263	49.0	W	4272	111	150	visa	226	debit	299	87
3	3663551	18403310	171.0	W	4476	574	150	visa	226	debit	472	87
4	3663552	18403310	284.95	W	10989	360	150	visa	166	debit	205	87
5	3663553	18403317	67.95	W	18018	452	150	mastercard	117	debit	264	87
6	3663554	18403323	57.95	W	12839	321	150	visa	226	debit	512	87
7	3663555	18403350	87.0	W	16560	476	150	visa	126	debit	110	87

Figura 18 - Tabela no PostgreSQL

Segue-se então o passo da conexão ao Python, para transformarmos esses dados num *dataframe* e aplicarmos o modelo (Floresta Aleatória). Para nos ajudar nesta tarefa, utilizámos uma biblioteca complementar para Python, “psycopg2”, que permite usar *queries* SQL, conforme a figura 19.

```
#From the database into dataframe
# Read data from PostgreSQL database table and load into a DataFrame instance
try:
    connection = psycopg2.connect("dbname=transactions user=postgres password=postgres")
    postgres_df = pd.read_sql("select * from transactions", connection)
    pd.set_option('display.expand_frame_repr', False)
    # Print the DataFrame
    print(postgres_df.head)
except (Exception, psycopg2.Error) as error:
    print ("Error while connecting to PostgreSQL", error)
finally:
    #closing database connection.
    if(connection):
        cursor.close()
        connection.close()
        print("PostgreSQL connection is closed")
```

Figura 19 - Conexão entre PostgreSQL e Python

Sucedendo-se este passo, é feito o mesmo tipo de tratamento de dados do que o *dataframe* de treino do modelo, isto é, tratar das variáveis não-numéricas e em falta, após o qual é-lhe aplicado o modelo já treinado para que possa fazer a previsão destes novos

dados, vindos da base de dados. O modelo que utilizámos para esta fase previu cerca de 12228 casos de fraude e 239609 de não-fraude nestes novos dados, porém não é possível determinar o quão correto o modelo está, visto que não temos acesso à informação real que nos indicaria se realmente se tratava de fraude ou não.

De modo a associarmos o resultado com as informações contidas na base de dados, estas previsões serão enviadas novamente para a mesma.

```
from sqlalchemy import create_engine
try:
    engine = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/transactions')
    conn = engine.connect()
    results_df = pd.DataFrame(postgres_isFraud, columns=['is_fraud'])
    results_df["transactionid_x"] = postgres_df["transactionid_x"].values

    transaction_ids = postgres_df.transactionid_x

    frame = results_df.to_sql("results", conn, if_exists='replace', index=False)

except (Exception, psycopg2.Error) as error:
    print("Error while connecting to PostgreSQL", error)

except ValueError as vx:
    print(vx)

except Exception as ex:
    print(ex)

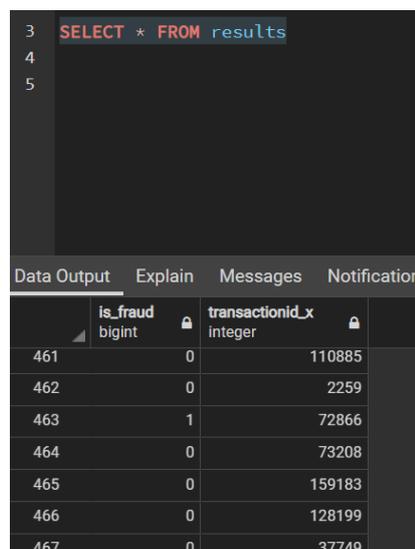
else:
    print("Table has been updated successfully.")

finally:
    #closing database connection.
    if(conn):
        conn.close()
        print("PostgreSQL connection is closed")

Table has been updated successfully.
PostgreSQL connection is closed
```

Figura 20 - Passagem de Resultados para Base de Dados

Como se pode verificar na figura 21, os dados foram passados para o PostgreSQL.



The screenshot shows a PostgreSQL query window with the following SQL statement: `SELECT * FROM results`. Below the query, a table view displays the results of the query. The table has two columns: `is_fraud` (bigint) and `transactionid_x` (integer). The results are as follows:

	is_fraud bigint	transactionid_x integer
461	0	110885
462	0	2259
463	1	72866
464	0	73208
465	0	159183
466	0	128199
467	0	37749

Figura 21 - Dados de Fraude e ID da Transação

## 5. Benchmarking

Uma vez que as soluções neste contexto do mercado não são tipicamente open source, uma das maneiras possíveis de fazer *benchmarking* da relevância deste projeto é de percebermos a saúde financeira das empresas que oferecem soluções à indústria neste contexto.

### 5.1 Ravelin

A Ravelin é uma empresa criada em 2015 cujo objetivo é providenciar aos seus clientes a solução de detecção de fraude mais precisa do mercado para fazer com que comércio online seja seguro. Neste contexto, a utilização de técnicas em *stream* para análise e decisões de modelos em tempo real é necessária e bastante valiosa. A empresa está actualmente avaliada em cerca de 123.000.000\$ (123 milhões de dólares). [2]

### 5.2 Feedzai

A Feedzai é uma empresa líder de mercado, criada em 2012, que utiliza AI para combater a fraude. A Feedzai vende soluções de gestão de risco utilizando *machine learning* e *big data*. Parte deste risco é gerido em real time, e como tal é da maior importância garantir que os modelos consigam responder ao volume de dados o mais rápido possível. A empresa está actualmente avaliada em 575.000.000\$ (575 milhões de dólares), e é uma das 50 FinTech de 2018 consideradas pela Forbes. [3]

### 5.3 Stripe

A Stripe é uma empresa de gestão de pagamentos, criada em 2009, e que também oferece aos seus clientes soluções de detecção de fraude - que também tem que ser gerido em tempo real. A Stripe está actualmente avaliada em 35.000.000.000\$ (35 mil milhões de dólares). [4]

Pode se também fazer uma comparação de modelos usados, por exemplo, modelo de regras, com o modelo de *machine learning*, conforme a figura abaixo.

Machine Learning	Rules
Proactive - tells you what's happening	Reactive - tells you what already happened
Learn across many data elements	Limited data elements
Scalable	Higher maintenance
Needs statistical significance	Can operate on a small data set
More accuracy at scale	Less accuracy at scale

Figura 22 - Machine Learning vs Modelo de Regras [5]

## 6. Método e Planeamento

De acordo com o calendário anteriormente apresentado, a parte de estudo, investigação e aplicação programática de modelos foi realizada. No entanto, devido a algumas limitações de tempo e disponibilidade por parte do orientador, houve dificuldade na tentativa de aprender e realizar a parte referente à geração de dados em tempo real, que descreveremos com mais detalhe no capítulo Conclusão e Trabalhos Futuros, visto que é uma ideia interessante para explorar.

Mais concretamente, começámos por fazer um estudo de classificadores binários, dos modelos de *machine learning* de um modo teórico, exploração de dados com Análise *Zoom In – Zoom Out* e Exploração Univariada e Multivariada, e posteriormente aplicámos programaticamente estes modelos, com a sua validação, para que pudéssemos obter uma conclusão acerca de qual modelo melhor se adequou ao problema, sendo esse um dos nossos objetivos principais.

Para complementar os estudos iniciais, também foi feita a leitura de alguns artigos acerca de transações fraudulentas *online*, para melhor compreender a evolução e crescimento deste risco. Este passo foi importante para nos contextualizarmos melhor ao problema em estudo.

## 7. Resultados

Tal como referido no capítulo Solução Desenvolvida, obtemos informação do modelo acerca da probabilidade de cada transação registada ser fraude ou não. Para todos os modelos utilizados, o valor conseguido através do método *score* esteve sempre entre os 96 e 97%, exceto no modelo de Regressão Linear, onde apenas se obteve um valor baixo. Consideramos estes resultados bastante positivos, apesar de não serem completamente perfeitos, principalmente porque se trata de um número significativo de dados, mesmo que apenas 20% dos dados de transações tenham sido utilizados para a previsão, trata-se de cerca de 118.000 casos de teste.

Individualmente, o valor de *score* para cada modelo perante os dados de teste:

- Regressão Logística – Aproximadamente 96.77%
- Floresta Aleatória – Aproximadamente 98.07%
- *Extra Trees Classifier* (Floresta “Mais” Aleatória) – 98.07%
- Árvores de Decisão – Aproximadamente 96.57%
- Regressão Linear – Aproximadamente 21.68%

De salientar que estes resultados não são necessariamente fixos, podendo flutuar ligeiramente com cada iteração. Por exemplo, o modelo *Extra Trees Classifier* oscilou entre os 97 e os 98%.

Além disso, observámos também o *score* dos dados de treino, para que os pudéssemos comparar com os de teste, assim como para identificar se estava a ocorrer *overfitting* por parte dos modelos. Isto seria evidente se os valores de *score* dos dados de treino fossem muito superiores aos de teste, ou seja, se o modelo conseguisse prever os dados de treino numa proporção muito maior, mas não apresentasse o mesmo comportamento perante dados novos.

Valor de *score* para cada modelo perante os dados de treino:

- Regressão Logística – Aproximadamente 97.34%
- Floresta Aleatória – Aproximadamente 99.93%
- *Extra Trees Classifier* – 100%
- Árvores de Decisão – 100%
- Regressão Linear – Aproximadamente 22.14%

Verifica-se que não há uma grande discrepância, ao comparar os valores dos dois tipos de dados, nem mesmo no caso da Regressão Linear, que apresentou, em ambos os casos, valores baixos e próximos.

Para além do *score* é também importante usar mais métricas de performance do modelo treinado.

Valor de *precision* e *recall* de cada modelo perante os dados de treino:

- Floresta Aleatória – Aproximadamente 93.04% (*Precision*) e 46.28% (*Recall*)
- *Extra Trees Classifier* – Aproximadamente 93.27% (*Precision*) e 46.18% (*Recall*)
- Árvores de Decisão – Aproximadamente 51.74% (*Precision*) e 59.61% (*Recall*)

Verifica-se que existe uma discrepância entre os resultados de *precision* e *recall* tanto no modelo de Floresta Aleatória, como no modelo *Extra Trees*. O modelo de Árvores de Decisão mostra uma pequena diferença em relação aos modelos anteriormente referidos, mas uma menor discrepância entre os resultados de *precision* e *recall*. Através destes resultados obtidos, é possível observar que os modelos de Floresta Aleatória e *Extra Trees* são extremamente precisos, com uma percentagem de casos positivos verdadeiros perto dos 100%, no entanto em relação ao *recall*, apenas perto de metade dos casos positivos são corretos.

Valor de *F1* e *MCC* de cada modelo perante os dados de treino:

- Floresta Aleatória – Aproximadamente 61.81% (*F1*) e 64.88% (*MCC*)
- *Extra Trees Classifier* – Aproximadamente 61.77% (*F1*) e 64.89% (*MCC*)
- Árvores de Decisão – Aproximadamente 55.40% (*F1*) e 53.87% (*MCC*)

Verifica-se que existe pouca discrepância entre diferentes modelos, mostrando, através dos resultados obtidos, que maioritariamente são positivos, tanto classificando com o *FI* ou com o *MCC*, onde se consegue verificar que são classificadores binários acima da média.

Posto isto, estes valores indicam que os modelos que mais se aproximam do resultado favorável são a Floresta Aleatória e a sua variante, sendo então estes os modelos que escolheríamos para explorar melhor e afinar para que se consiga alcançar uma maior percentagem de casos corretamente previstos. Este ponto será um pouco mais aprofundado no capítulo Conclusão e Trabalhos Futuros.

No que diz respeito aos requisitos inicialmente elaborados, cumprimos o requisito de os modelos serem de Machine Learning, assegurámos uma grande quantidade e diversidade de dados, as previsões são, na sua maioria corretas (exceto o modelo de Regressão Linear), não tendo sido possível cumprir a concretização das transações simultâneas em tempo real e a garantia de proteção de dados, já que não foi um aspeto explorado durante o trabalho.

## 8. Conclusão e Trabalhos Futuros

Com este trabalho, explorámos um pouco melhor o mundo de *Data Science* e *Machine Learning*, e conseguimos observar em primeira-mão os passos necessários para todo o processo. As maiores dificuldades sentidas durante a realização do trabalho relacionam-se com a exploração e processamento/limpeza dos dados iniciais, que para ser feito da melhor maneira, requer um conhecimento mais aprofundado de vários aspetos da área da Estatística.

Poderá ser feito como trabalho futuro um melhor tratamento dos dados iniciais, como por exemplo, perceber quais variáveis eram mais relevantes para a obtenção dos resultados e eliminar por completo as menos relevantes, para reduzir o “ruído”, que certamente afetará os resultados finais e mesmo alguns modelos que assumem que esse tratamento foi previamente feito. Em relação aos modelos, um importante aspeto a melhorar seria analisar os parâmetros intrínsecos aos mesmos, que têm como função afiná-los e adaptá-los mais concretamente ao problema para uma melhor performance, não só em termos de resultados, mas também computacionalmente. Quanto ao objetivo que não foi realizado, continuamos a considerá-lo um objetivo importante como trabalho futuro, visto que nos permitiria simular transações em tempo-real (com ajuda de uma base de dados e geração automática de dados que seriam fornecidos aos modelos) e, portanto, mais próximas de casos reais, em que há várias transações a serem feitas simultaneamente. Isto não foi possível de concretizar, devido ao facto de não estarmos suficientemente familiarizados com as ferramentas envolvidas e com a própria complexidade do processo. Também, sem a limitação de tempo, essa simulação teria benefícios ao ser feita periodicamente, com intervalos de alguns meses, para testar a contínua validade do modelo treinado e para perceber se é necessário treiná-lo de novo.

## Bibliografia

- [1] "IEEE-CIS Fraud Detection", *Kaggle*. [Online]. Available: <https://www.kaggle.com/c/ieee-fraud-detection/data>. [Accessed: 24- Nov- 2019].
- [2] "Ravelin", *Crunchbase*. [Online]. Available: <https://www.crunchbase.com/organization/ravelin> [Accessed 24 Nov. 2019].
- [3] "Feedzai on the Forbes Fintech 50 List", *Forbes*. [Online]. Available: <https://www.forbes.com/companies/feedzai/#3fdd93b180dc>. [Accessed: 24 Nov. 2019].
- [4] "Stripe Now Has A Valuation Of \$35 Billion", *Forbes*. [Online]. Available: <https://www.forbes.com/sites/donnafuscaldo/2019/09/19/stripe-now-has-a-pre-money-valuation-of-35-billion/#6fe3e1862e68>. [Accessed: 24 Nov. 2019].
- [5] "Machine Learning for Fraud Prevention: What's Next", *Sift Science*. [Online]. Available: <https://pages.siftscience.com/rs/526-PCC-974/images/ebook-machine-learning-for-fraud-prevention-whats-next.pdf>. [Accessed: 24- Nov- 2019].
- [6] Vogelsang, A. and Borg, M. (2019). "Requirements Engineering for Machine Learning: Perspectives from Data Scientists." *IEEE*. [Online] Available: <https://arxiv.org/pdf/1908.04674.pdf> [Accessed 31 Jan. 2020].
- [7] Nguyen, T., 2020. Kafka, For Your Data Pipeline? Why Not? [Online] Medium. Available at: <https://towardsdatascience.com/kafka-for-your-data-pipeline-why-not-5a14b50efe7f> [Accessed 11 September 2020].
- [8] Gulden, M., 2020. [online] Baeldung.com. Available at: <https://www.baeldung.com/kafka-connectors-guide> [Accessed 11 September 2020].
- [9] F. Technology, "What is PostgreSQL? | In One Minute", *Postgresql.fastware.com*, 2020. [Online]. Available: <https://www.postgresql.fastware.com/what-is-postgresql>. [Accessed: 11- Sep- 2020].
- [10] D. Vairub, "Hello Kafka World! The complete guide to Kafka with Docker and Python", *Medium*, 2020. [Online]. Available: <https://medium.com/big-data-engineering/hello-kafka-world-the-complete-guide-to-kafka-with-docker-and-python-f788e2588cfc>. [Accessed: 11- Sep- 2020].

- [11] Pynative.com, 2020. [Online]. Available: <https://pynative.com/python-postgresql-tutorial/>. [Accessed: 11- Sep- 2020].
- [12] "Machine Learning with Python and Postgres | ObjectRocket", Kb.objectrocket.com, 2020. [Online]. Available: <https://kb.objectrocket.com/postgresql/machine-learning-with-python-and-postgres-1114>. [Accessed: 11- Sep- 2020].
- [13] "Pandas DataFrames - Reading from and writing to PostgreSQL table | Pythontic.com", Pythontic.com, 2020. [Online]. Available: <https://pythontic.com/pandas/serialization/postgresql>. [Accessed: 11- Sep- 2020].
- [14] "Machine Learning models in Python - Scikit-learn", Scikit-learn.org, 2020. [Online]. Available: <https://scikit-learn.org/stable/>. [Accessed: 11- Sep- 2020].
- [15] "Metrics to Evaluate your Machine Learning Algorithm", Medium, 2020. [Online]. Available: <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>. [Accessed: 11- Sep- 2020].

## Anexos

<https://stripe.com/en-pt/radar/guide>

<https://www.knime.com/blog/fraud-detection-using-random-forest>

<https://towardsdatascience.com/credit-card-fraud-detection-a1c7e1b75f59>

<https://towardsdatascience.com/detecting-credit-card-fraud-using-machine-learning-a3d83423d3b8>

[Link do projeto no github](#)

## Glossário

**Baseline** - Método que usa heurística, estatística, aleatoriedade ou machine learning para criar previsões para um dataset.

**Dataset** – Uma coleção de conjuntos de informação que é composta por elementos separados, mas pode ser manipulado como uma unidade por um computador.

**False Negative** – Resultado previsto como sendo negativo, mas na realidade é positivo.

**False Positive** - Resultado previsto como sendo positivo, mas na realidade é negativo.

**Fraudster** – Pessoa que comete fraude.

**Inteligência Artificial** – A teoria e o desenvolvimento de sistemas computacionais capazes de realizar tarefas que normalmente necessitavam de inteligência humana, como percepção visual, reconhecimento de voz, tomada de decisão e tradução entre línguas.

**Machine Learning** - Uma aplicação da Inteligência Artificial que oferece a sistemas a habilidade de automaticamente aprender e melhorar de experiência sem ser explicitamente programado. O Machine Learning concentra-se na parte do desenvolvimento de programas que conseguem aceder a dados e usar para aprenderem por si próprios.

**Message Broker** – Um módulo intermédio que traduz uma mensagem de um protocolo formal de mensagens do remetente para o protocolo formal de mensagens do destinatário.

**Minimum Viable Product (MVP)** - Um produto com características suficientes para satisfazer clientes de uma fase inicial para estes darem feedback para futuro desenvolvimento de produtos.

**Overfitting** – Modelação bastante baseada nos dados de treino, formando um modelo demasiado complexo.

**Stakeholder** – Pessoa ou entidade envolvida num projeto ou negócio.

**Stream** - Sequência de elementos de dados a que se pode ter acesso segundo uma ordem.

**True Negative** – Resultado previsto como sendo negativo e que é, realmente, negativo.

**True Positive** - Resultado previsto como sendo positivo e que é, realmente, positivo.