INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Pré-processamento de Dados e Comparação entre Algoritmos de Machine Learning para a Análise Preditiva de Falhas em Linhas de Produção para o Controlo

DANIEL FILIPE BAPTISTA FERREIRA DA SILVA
Junho de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Pré-processamento de Dados e Comparação entre Algoritmos de Machine Learning para a Análise Preditiva de Falhas em Linhas de Produção para o Controlo de Qualidade

Daniel Filipe Baptista Ferreira da Silva

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

Orientador: Goreti Marreiros

Júri:

Presidente:

[Nome do Presidente, Categoria, Escola]

Vogais:

[Nome do Vogal1, Categoria, Escola]

Porto, Junho 2021

Resumo

Nos dias de hoje, devido ao constante desenvolvimento tecnológico e à constante necessidade de acelerar o processo de produção, os grandes fabricantes vêem-se obrigados a implementar processos de controlo de qualidade, por forma a diminuir o número de peças defeituosas ao longo das linhas de produção e, dessa forma, aumentar a produtividade das mesmas, o que irá beneficiar, não só as organizações responsáveis por esse processo, como também os clientes, pois, ao reduzir o tempo e custo de produção de um produto, as organizações têm a possibilidade de reduzir o seu preço.

Apesar de se tratar de um problema global, este documento apenas se foca nas linhas de produção da Bosch, ao analisar uma grande quantidade de dados que foram disponibilizados para a criação de modelos de *machine learning,* com o objetivo de prever que componentes irão falhar ao longo do processo, por forma a melhorar o controlo de qualidade nas suas fábricas. Contudo, devido ao elevado número de entradas, estes dados têm que passar por várias etapas, para garantir que se encontram no melhor estado possível para serem testados e treinados pelos algoritmos. Estas etapas englobam o pré-processamento, ou seja, a correção de dados em falta, normalização, redução do tamanho do *dataset*, entre outras, e a Seleção de Características dos dados, ao selecionar apenas os atributos mais relevantes para a construção do modelo.

Para o problema abordado neste documento, optou-se pela utilização da técnica de aprendizagem supervisionada, pelo facto do *dataset* utilizado ser rotulado, pois cada coluna contém um descritivo da peça, estação e linha de produção. Após terem sido analisados vários artigos semelhantes na mesma área de estudo, e terem sido comparados os seus resultados, os algoritmos XGBoost, Random Forest e Support Vector Machine foram escolhidos como os algoritmos a utilizar no desenvolvimento do modelo. Várias métricas de avaliação foram referidas na literatura como aptas para avaliar o modelo, como é o caso da exatidão, da precisão, da métrica F1, entre outras, que foram utilizadas para esse fim.

Por fim, concluiu-se que o XGBoost foi o algoritmo que apresentou os melhores resultados no contexto deste estudo. O algoritmo SVM foi o mais rápido a efetuar as previsões e o algoritmo Random Forest é um meio termo entre velocidade e qualidade dos resultados. O melhor a aplicar nas linhas de produção da Bosch irá depender de que métrica é vista como tendo um maior peso para a organização.

Palavras-chave: Machine Learning, Análise Preditiva, Feature Engineering, Linha de Produção, Classificação, XGBoost, Random Forest, Support Vector Machine.

Abstract

Nowadays, due to the constant technological development and the constant need to accelerate the production process, the big manufacturers are obliged to implement quality control processes, in order to reduce the number of defective parts along the production lines and, thus, increasing their productivity. This will benefit not only the responsible for this process, but also the customers, since, by reducing the time and cost of manufacturing a product, organizations have the possibility to lower its price.

Despite being a global problem, this document only focuses on the Bosch production line data. A large amount of data was made available for the creation of machine learning models, in order to predict which components will fail throughout the production process, for the sake of improving the quality control in the company. However, due to the high number of inputs, this data had to go through several steps, to ensure that it was in the best possible state to be tested and trained by the algorithms. These steps include the correction of missing data, normalization, reduction of the size of the dataset, among others, and feature selection, by selecting only the most relevant attributes for the creation of the model.

For the problem addressed in this document, it was decided to use the supervised learning technique because the dataset used is labeled, as each column identifies the part, the station and the production line. After analyzing several articles of the same area of study, and comparing the results, the XGBoost, Random Forest and Support Vector Machine algorithms were chosen for this project.

Several evaluation metrics, able to correctly evaluate classification models were mentioned in the literature and were used, as is the case of the accuracy, precision, F1 score and the Mathews Correlation Coefficient, to compare the results and predictions success of the various used algorithms.

Finally, it was concluded that XGBoost is the algorithm with the best and most advantageous results in the context of this study. The SVM algorithm was the fastest to finish all the predictions and the Random Forest algorithm is a compromise between speed and quality of the results. The best technique to apply on the Bosch production lines will depend on which metric is seen as having the most valuable for the organization.

Keywords: Machine Learning, Predictive Analysis, Feature Engineering, Production Lines, Classification, XGBoost, Random Forest, Support Vector Machine.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

ML	*Machine Learning*
AUROC	*Area Under the ROC Curve*
RFID	Identificação por Radiofrequência
IoT	*Internet of Things*
MTBF	Tempo médio entre falhas
MCC	*Matthew's Correlation Coefficient*
MES	*Manufacturing Execution System*
PPMP	*Production Performance Management Protocol*
AHP	*Analytic Hierarchy Process*
NDC	*New Concept Development*
SVM	*Support Vector Machine*
RF	*Random Forest*
KNN	*K Nearest Neighbors*
CSV	Comma-separated Values

Lista de Símbolos

$\boldsymbol{n}$	Número total de previsões

1 Introdução

Neste capítulo é possível encontrar uma breve descrição do problema, contexto, objetivos e valor do projeto. São também referidas as abordagens tecnológicas para o desenvolvimento do mesmo.

1.1 Contexto

A gestão de qualidade é dos aspetos mais importantes no processo de produção e requer um constante foco nos produtos e nos processos de inovação, por forma a prover aos clientes uma grande variedade de produtos com uma qualidade excelente e a um custo reduzido, o que irá auxiliar a organização no seu crescimento e sucesso (Rosa et al. 2018).

Este projeto foca-se no controlo de qualidade das linhas de produção de uma fábrica da Bosch, e na identificação dos componentes que poderão afetar a produção na mesma, ao analisar uma grande quantidade de dados recolhidos nas linhas de produção, através de técnicas de pré-processamento e da utilização de algoritmos preditivos de *machine learning* para a criação de modelos que efetuem previsões de falhas em peças, ou secções das linhas de produção, que poderão comprometer o processo de fabricação. Esta análise promove assim um maior apoio no controlo de qualidade dos produtos confecionados.

A implementação de tecnologias de sensores mais sofisticados nas linhas de produção e a *Internet of Things* (IoT), torna possível a recolha de dados em qualquer fase do processo de fabricação (Zaslavsky et al. 2012). Ao mesmo tempo, o rápido crescimento do volume destes dados criou um novo desafio para o controle de qualidade dos produtos: como tratar o grande volume e variedade dos dados com rapidez, por forma a obter a informação útil e efetuar a previsão de falhas nas linhas de produção. A criação de modelos de previsão, através da utilização de algoritmos de *Machine Learning,* permite solucionar este problema.

1.2 Problema

Falhas nos componentes de numa linha de produção, com o seu impacto direto no tempo de inatividade da mesma e nos custos de produção, está diretamente relacionado com a capacidade de uma empresa ser competitiva em termos de custo, qualidade e desempenho.

A *Bosch*, sendo uma das principais empresas de produção do mundo, tem como objetivo garantir que as produções dos seus componentes sigam os mais altos padrões de qualidade e segurança. Parte desse trabalho é composto pela monitorização de perto das peças à medida que estas progridem nos processos de fabricação. Como a *Bosch* regista os dados em cada etapa de suas linhas de montagem, é possível efetuar análises avançadas para melhorar os processos de produção. No entanto, devido à complexidade dos seus dados e das suas linhas de produção, surgiram problemas na utilização dos métodos atualmente empregados pela organização, o que obrigou a mesma a procurar soluções mais eficientes para efetuar o tratamento dos mesmos (Kaggle 2016).

A análise e o desenvolvimento deste documento são baseados num conjunto de dados fornecido pela *Bosch* (Kaggle 2016) para prever falhas internas utilizando medições e testes realizados a vários componentes ao longo da linha de montagem.

1.3 Objetivos

O objetivo principal deste estudo é a pesquisa e a criação de modelos de *machine learning* para efetuar a previsão de falhas nas linhas de produção de uma fábrica, e comparar os resultados dos mesmos, através da utilização de várias métricas de avaliação, por forma a detetar quais os modelos e algoritmos que têm um melhor desempenho neste contexto. Os modelos criados irão permitir ao fabricante facilmente detetar no processo de fabrico, quais as variáveis que mais afetam o processo (quais os componentes mais propícios a falhas), o que irá melhorar o desempenho do mesmo e o processo de controlo de qualidade.

No final deste estudo é esperado que tenham sido obtidos os seguintes resultados:

- Revisão da literatura sobre as abordagens de manutenção preditiva e dos métodos de pré-processamento e engenharia de recursos.
- Análise de vários algoritmos preditivos de *machine learning* possíveis para solucionar o problema.
- Avaliação de vários algoritmos preditivos e seleção do melhor algoritmo para o problema em questão.
- Desenho e desenvolvimento de uma solução que incorporará o pré-processamento dos dados, a engenharia de recursos e o método preditivo mais adequado para o problema em questão.

1.4 Abordagem

Este projeto foca-se no desenvolvimento de um modelo para prever falhas nos componentes de uma linha de produção. Um *dataset* obtido no *website* da Kaggle (Kaggle 2016), disponibilizado pela Bosch, irá ser utilizado juntamente com algoritmos de aprendizagem supervisionada de forma a reduzir a dispersão dos dados.

Como dito anteriormente neste documento, o pré-processamento dos dados é das etapas mais importantes num projeto de *machine learning*, pois, a utilização de dados não tratados nem normalizados num dataset, podem afetar grandemente os resultados dos algoritmos. O conjunto de dados usado contém dados ausentes e desequilibrados. Para os dados em falta, várias estratégias podem ser aplicadas, como, por exemplo, substituir os valores ausentes pelo valor mais provável. Algumas colunas de entrada no *dataset* foram selecionadas e removidas do conjunto de dados, por se tratar de recursos irrelevantes para o contexto do problema.

Vários algoritmos serão comparados por forma a se conseguir obter o que melhor se adequa ao projeto em questão, utilizando medidas de desempenho como Matthew's Correlation Coefficient (MCC) e curva ROC. Será utilizada a linguagem Python (em mais detalhe na secção 4.1 deste documento), visto ser uma linguagem muito utilizada nesta área e com um grande suporte da comunidade.

2 Estado da Arte

Este capítulo pretende abordar em maior detalhe o problema abordado neste documento, o seu contexto, o conjunto de dados utilizado é contextualizado no ambiente deste projeto e é feita uma análise de valor do problema. É realizado também uma análise ao estado da arte de *machine learning*, aos vários tipos de algoritmos, medidas de desempenho, técnicas de pré-processamento e de aprendizagem.

2.1 Problemas nas linhas de produção

Falhas nos componentes de numa linha de produção, com o seu direto impacto no tempo de inatividade da mesma e nos custos de produção, está diretamente relacionado com a capacidade de uma empresa ser competitiva em termos de custo, qualidade e desempenho.

A previsão de falhas é baseada na deteção precoce de problemas nos componentes ao percorrerem a linha de produção. Ao prevenir tais falhas, reduzindo as falhas inesperadas, é possível a uma organização entregar produtos com maior qualidade, e de forma mais rápida, o que resulta em maiores ganhos e em menores custos de produção.

Esta pode ser implementada através da monitorização dos equipamentos, combinado com métodos de decisão inteligentes. *Machine Learning* pode ser usada para extrair perceções dos dados e prever resultados com precisão para apoiar a tomada de decisões e auxiliar as organizações a melhorar as suas operações e competitividade.

Um exemplo da importância da monitorização e previsão de falhas nas linhas de produção, e do grande impacto que esta pode ter sobre uma organização, é o defeito recente encontrado nos airbags da Takata, empresa japonesa que fabrica partes para automóveis, que resultou no maior *recall* da história da indústria automóvel. A recolha inclui atualmente cerca de 63 milhões de *airbags* (NHTSA 2020). Estes continham o risco de explodir quando fossem ativados numa colisão, o que causou vários ferimentos e mortes (NHTSA 2020). Como consequência deste grave erro por parte da organização, esta viu-se obrigada a declarar falência em 2017 (CNN 2017). No final desse ano, a empresa admitiu a manipulação de informação importante sobre os airbags defeituosos por vários anos, mesmo depois destes terem começado a explodir nos carros. Devido a tal, nos Estados Unidos da América, a empresa declarou-se culpada numa acusação criminal de fraude pela qual terá que pagar mil milhões de dólares, incluindo um fundo de 125 milhões de dólares para indenizar as vítimas e as suas famílias (CNN 2017).

Assim sendo, um defeito nas linhas de produção pode levar a um grande prejuízo financeiro, à falência da empresa, ou até mesmo causar ferimentos ou mortes, o que demonstra a importância de uma monitorização constante e eficiente das linhas de produção, por forma a entregar aos consumidores finais, o melhor produto possível.

Apesar de este assunto poder afetar qualquer linha de produção, este documento irá analisar apenas as linhas de produção da Bosch, visto que esta é uma das principais empresas de manufatura do mundo, e por ter disponibilizado um grande conjunto de dados relativos a análises e testes efetuados às suas linhas de produção.

Como esta afirma no seu *website*, "A conectividade real dos processos de fabricação é o resultado de uma avaliação significativa de todos os dados. A produção torna-se transparente, versátil e flexível." (Bosch 2020).

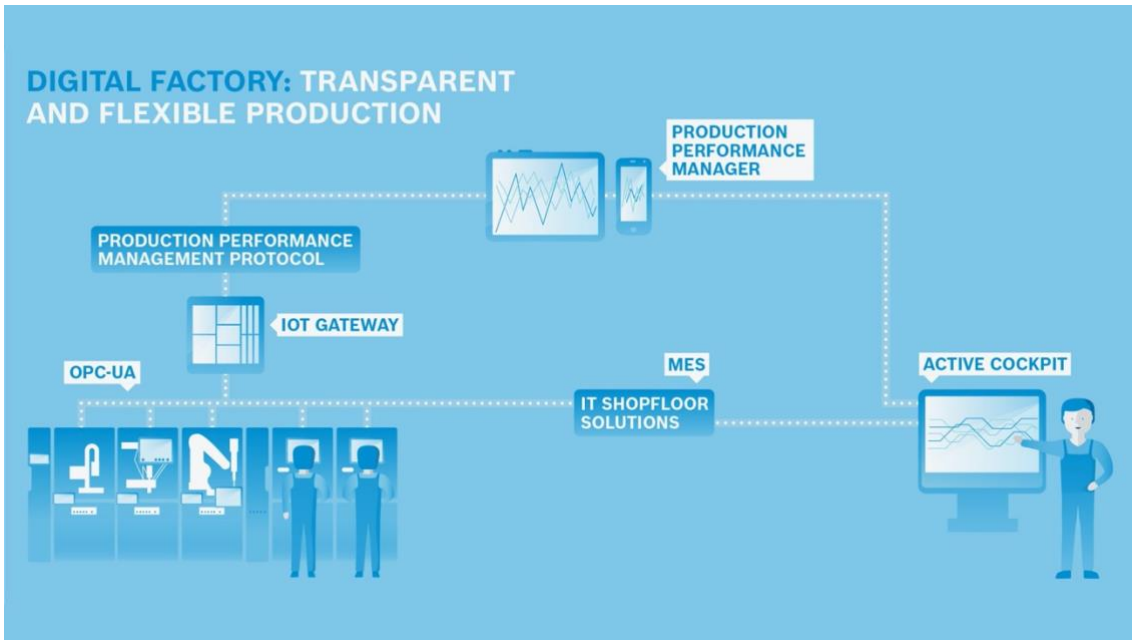


Figura 1 - Linhas de produção da Bosch. (Bosch 2020)

A figura acima representa a estrutura das linhas de produção da Bosch. Estas estão conectadas a um sistema que fornece informações sobre as peças e sobre as etapas da produção por forma a acompanhar e analisar as montagens em tempo real, chamado MES (*Manufacturing Execution System*) (Bosch 2020).

O IoT *Gateway* converte os dados em linguagem de máquina e transfere esses dados para o PPMP (*Production Performance Management Protocol*). Este sistema capta os dados das máquinas e dos sensores e apresenta-os quase em tempo real. Além disso, o sistema identifica os incidentes (falhas, defeitos, etc.) e informa o responsável pelos mesmos, por forma a melhorar a qualidade e reduzir o número de produtos abaixo dos padrões de qualidade estabelecidos e dos tempos de inatividade dos componentes ao longo das linhas de produção (Bosch 2020).

Estes dados, que foram recolhidos nas linhas de produção, serão analisados e testados no desenvolvimento deste projeto, aos quais serão aplicados algoritmos de *machine learning*, por forma a identificar e prever tais incidentes nas linhas de produção, de forma a garantir os padrões de qualidade da organização.

2.2 Machine learning

Nesta secção será realizada uma contextualização ao estado da arte de *machine learning (ML),* começando por abordar as três principais técnicas de aprendizagem, supervisionada, não supervisionada e de reforço. De seguida, abordar-se-á algumas opções para efetuar o pré-processamento de dados, etapa que é frequentemente realizada antes da aplicação dos algoritmos de *machine learning* e, por fim, serão apresentadas algumas métricas que poderão ser utilizadas para avaliar o resultado dos algoritmos, para detetar qual obteve a melhor performance num certo contexto.

Como o nome sugere, a designação pode ser vagamente interpretada como algo que permite aos computadores, "aprender". De uma forma mais concreta, a intenção da utilização de ML é permitir que as máquinas aprendam de forma autónoma, recorrendo a dados fornecidos, e que extraiam conclusões precisas, convertendo dados *raw* em informação útil.

Os dois subcampos mais utilizados em problemas de *machine learning* são a aprendizagem supervisionada, abordada em mais detalhe na secção 2.3.1 e a aprendizagem não supervisionada, mencionada na secção 2.3.2 deste documento.

2.2.3 Aprendizagem supervisionada

Na literatura, encontramos vários problemas a serem resolvidos recorrendo a técnicas de deteção supervisionada e não supervisionada.

As técnicas de deteções supervisionadas assumem que as classificações das transações passadas estão disponíveis, são fiáveis e os dados estão rotulados. Os modelos que utilizam esta técnica são iterativamente treinados para encontrar uma função que seja capaz de predizer atributos desconhecidos e prever se o resultado pertence a uma das categorias predefinidas, fraudulenta ou legitima.

De seguida, serão expostos alguns dos algoritmos mais relevantes neste tipo de aprendizagem, assim como uma breve descrição relativa a cada um.

2.2.3.1 Support Vector Machine (SVM)

SVM (Corinna et al. 1995) é um algoritmo de aprendizagem supervisionada utilizado para efetuar a classificação e a análise de regressão. O SVM recebe um conjunto de dados como entrada, e prediz, para cada entrada, qual de duas possíveis classes a entrada faz parte, o que faz do SVM um classificador linear binário não probabilístico.

Disponibilizando um conjunto de dados de treino, por norma 70% da totalidade dos dados, o algoritmo constrói um modelo, que atribui os novos exemplos a uma das duas categorias. Um modelo SVM é uma representação de exemplos como pontos no espaço, mapeados de maneira que os exemplos de cada categoria sejam divididos por esse espaço. Os novos exemplos são

então mapeados no mesmo espaço e preditos como pertencentes a uma categoria baseados em qual o lado do espaço eles são colocados.

A linha que separa os espaços é chamada de *hyperlane.* Os pontos que se encontrarem mais perto da linha recebem um valor de coeficiente maior que zero, enquanto aos restantes é-lhes atribuído o valor zero.

2.2.3.2 Random Forest (RF)

Siddhart Misra e Hao Li apresentam uma boa descrição sobre este classificador no seu livro sobre *machine learning* (Misra e Li, 2020). Os autores afirmam que o classificador de RF é um método que treina várias árvores de decisão em paralelo com *bootstrapping,* seguido de agregação. O bootstrapping indica que várias árvores de decisão individuais são treinadas em paralelo, em vários subconjuntos do conjunto de dados de treino. O *bootstrapping* garante que cada árvore de decisão individual na floresta aleatória seja única, o que reduz a variância geral do classificador de RF.

A entrada de cada árvore é uma amostra de dados do conjunto de dados original. Além disso, um subconjunto é selecionado aleatoriamente entre os recursos opcionais, de forma a aumentar a árvore em cada nó. Essencialmente, a floresta aleatória permite que um grande número de classificadores fracos ou fracamente correlacionados, forme um classificador forte (Mao e Wang, 2012).

Trata-se então de um classificador muito utilizado, visto que este tende a superar a maioria dos outros métodos de classificação em termos de precisão (Misra e Li, 2020).

2.2.3.3 K Nearest Neighbours (KNN)

O algoritmo KNN (Harrison 2018) assume que dados semelhantes estão próximos uns dos outros. Na imagem abaixo, é possível verificar esse facto. O algoritmo KNN depende que essa suposição seja verdadeira o suficiente, para que o algoritmo seja útil. KNN utiliza a ideia de proximidade através do cálculo da distância entre pontos num gráfico.

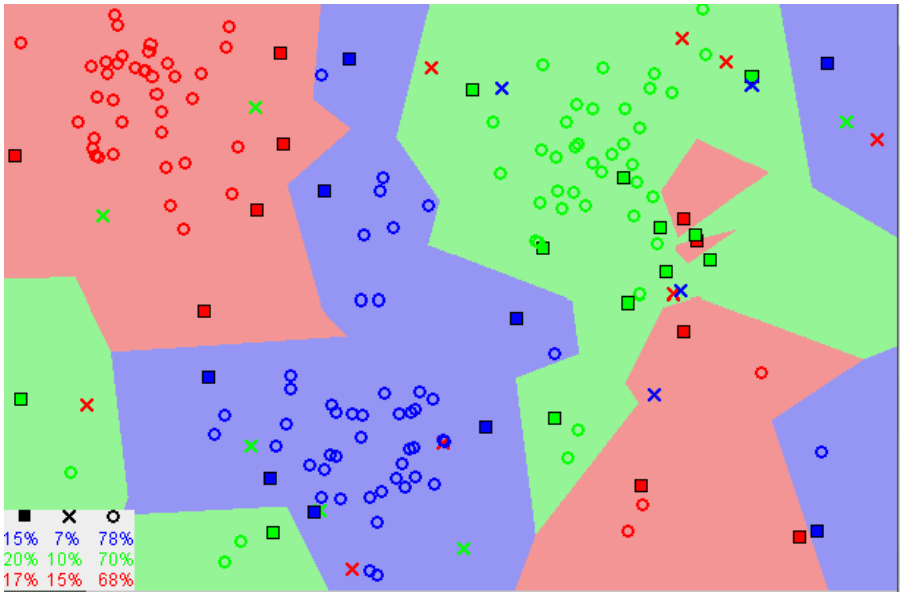


Figura 2 - Algoritmo KNN (Harrison 2018)

Para selecionar o K certo para os dados introduzidos no algoritmo, executa-se o algoritmo KNN várias vezes, com diferentes valores de K, e escolhe-se o K que reduz o número de erros que, enquanto se mantém a capacidade de o algoritmo efetuar previsões com precisão (Harrison 2018).

2.2.3.4 Naive Bayes

O classificador Naive Bayes (Gandhi 2018) é um modelo probabilístico, baseado no teorema de Bayes, que é utilizado para efetuar classificações, em que o seu output é baseado em algo conhecido anteriormente, ou seja, este permite obter a probabilidade de A acontecer, sabendo que B já aconteceu. A equação deste teorema é a seguinte:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (1)$$

Este classificador é conhecido por ter bom desempenho em contextos como a filtração de *spam* e em sistemas de recomendação.

2.2.3.5 XGBoost

XGBoost é uma biblioteca otimizada de "gradient boosting" projetada para ser altamente eficiente, flexível e portátil. Esta implementa algoritmos de *machine learning* sob a estrutura Gradient Boosting. O XGBoost fornece "*tree boosting*" paralelo (também conhecido como GBDT, GBM) que resolve muitos problemas de maneira rápida e precisa, mesmo quando estes contêm um número de entradas para além dos mil milhões (XGBoost, 2020).

Boosting é uma técnica em que novos modelos são adicionados para corrigir os erros cometidos por modelos existentes. Os modelos são adicionados sequencialmente até que nenhuma outra melhoria possa ser feita. A abordagem *Gradiente Boosting* é uma técnica em que novos

modelos são criados para prever os resíduos ou erros de modelos anteriores e, em seguida, somados para fazer a previsão final. É chamado de *Gradiente Boosting* porque utiliza um algoritmo de *Gradient Descent* para minimizar a perda ao adicionar novos modelos. Esta abordagem suporta ambos os modelos de regressão e de classificação para problemas preditivos (Jason Brownlee 2016).

O algoritmo *Optimized Gradient Boosting* (XGBoost), que se trata de uma evolução do *Gradient Boosting*, tal como foi dito no início desta secção, aplica paralelismo ao processamento, *tree-pruning* e trata os valores em falta, por forma a evitar o *overfitting.*

2.2.3.6 Decision Trees

Decision Trees (Scikit-learn 2020) é um método supervisionado que é utilizado para efetuar a classificação e regressão. O objetivo é a criação de um modelo que prevê o valor de uma variável ao aprender através de várias regras de decisão simples. Na imagem abaixo, é apresentado um exemplo em que o algoritmo aprende com os dados, por forma a se aproximar da curva com um conjunto de decisões se-então-senão. Quão mais profunda for a árvore, mais complexas serão as regras de decisão e melhor será o modelo final.

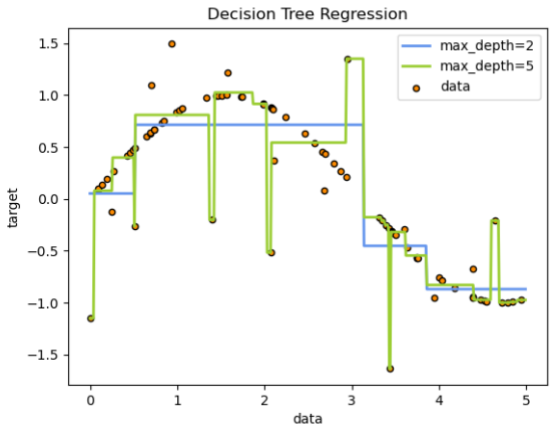


Figura 3 - Decision Trees (Scikit-learn 2020)

2.2.4 Aprendizagem não supervisionada

Ao contrário da deteção supervisionada, as técnicas de deteção não supervisionadas não contêm a informação da classificação de uma transação, ou seja, o output desejado para um certo input não é conhecido antecipadamente. Os modelos destas técnicas aprendem com os dados de uma forma iterativa e produzem conclusões através das relações da informação, sendo o seu principal objetivo, a adaptação do seu comportamento ao analisar informação desconhecida. De seguida, será apresentado um estudo, que optou pela utilização desta abordagem, desenvolvida por Quah e Sriganesh (Quah et al. 2008).

No artigo, a proposta dos autores procura minimizar o risco de fraude através de técnicas de redes neuronais, mais especificamente utilizando Self-Organizing Maps (SOM), apresentado uma abordagem multicamada consistindo em:

- As camadas iniciais de autenticação e triagem.
- A camada principal de pontuação de risco e análise de comportamento
- Uma camada de revisão adicional e tomada de decisão

As camadas iniciais de autenticação e triagem consistem em mecanismos de verificação de PIN, emails e data de validade, triagem com base em listas positivas e negativas. A camada de revisão final pode consistir numa revisão manual ou revisão através de outros mecanismos.

O objetivo da camada central é para obter a pontuação de risco e as análises comportamentais do cliente. Consiste em duas subcamadas: uma camada de SOM seguido por um sistema de pontuação de risco baseado em regras.

A camada de SOM tem várias finalidades:

- Para classificar e agrupar dados de entrada.
- Para detetar padrões ocultos nos dados de entrada.
- Atuar como um mecanismo de filtragem para outras camadas

A arquitetura desta camada SOM foi definida em sete pontos:

- Vetores de input: Representados por dados de clientes, contas e transações, tipicamente incluem número de cliente, tipo de cliente, número de conta, saldo da conta, número de transação, tipo de transação
- Pré-processamento dos vetores de input: Converter valores numéricos e normalizar para que se situem num conjunto específico.
- Matriz de neurónios: Um array bidimensional que tem como objetivo aprender a partir dos dados do vetor de input.
- Vetores de output: Defnição dos clusters obtidos
- Interpretação do output: Filtrar o número de transações que precisam ser enviadas para revisão, reduzindo assim o tempo geral de processamento, custo e complexidade
- Pontuação de risco: Atribuição de uma pontuação consoante a atividade detetada na conta, como por exemplo: logins inválidos antes de uma transação e contas que se tornaram ativas de uma forma repentina.

Esta implementação foi capaz de obter a definição de vários clusters como é possível verificar na seguinte figura:

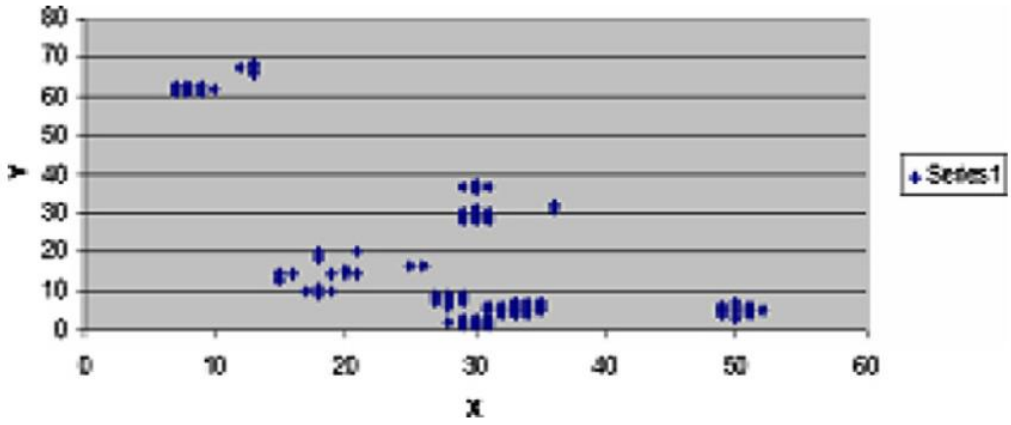


Figura 4 - Clusters obtidos no estudo (Quah et al. 2008)

2.2.4.1 Clustering

Clustering é dos algoritmos mais utilizados em aprendizagem não supervisionada e pode ser considerado como o mais importante (Mishra 2017). Tal como todos os algoritmos utilizados na deteção não supervisionada, este procura encontrar uma estrutura numa coleção de dados desconhecidos, sem rótulo.

Um *cluster* pode ser definido como sendo um conjunto de dados que contêm características similares entre eles, e que não são similares entre objetos de outros *clusters.* O objetivo principal desta técnica é determinar os grupos com características semelhantes, num conjunto de dados sem rótulo. Porém, não há um critério que defina qual será a melhor solução, tudo depende do resultado pretendido pelo utilizador, após este definir os critérios de semelhança pretendidos entre os *clusters*.

Existem três formas possíveis de efetuar o cálculo da distância entre os pontos, a distância Cosine, Jaccard e Euclidiana. O tipo de equação terá de ser escolhido pelo utilizador, dependendo do caso de uso, pois nem todas se aplicam, nem garantem o melhor resultado, para o mesmo caso.

Os algoritmos que utilizam o método de *clustering* podem ainda ser divididos em 4 grupos (Mishra 2017):

- *Exclusive Clustering;*
- *Overlapping Clustering;*
- *Hierarchical Clustering;*
- *Probabilistic Clustering*;

No primeiro grupo, os dados são agrupados numa forma exclusiva, ou seja, se um ponto pertence a um certo *cluster*, não pode pertencer a mais nenhum outro. Um algoritmo que segue esta abordagem, é o algoritmo K-Means. Pelo contrário, no segundo grupo, um ponto pode pertencer a dois ou mais *clusters,* como se pode verificar ao utilizar o algoritmo Fuzzy K-Means.

No agrupamento por hierarquia, um *cluster* é definido como a junção dos dois *clusters* mais próximos, sendo que os primeiros *clusters* são representados por cada um dos pontos individualmente. Por último, num *cluster* probabilístico, como o nome indica, é usada uma abordagem probabilística, como é o caso no algoritmo "*Mixture of Gaussians*" (Mishra 2017).

2.2.4.2 Autoencoder

Autoencoder (Keras 2016) é uma rede neuronal usada para ler dados de forma eficiente de uma forma não supervisionada. *Autoencoding* é um algoritmo de compressão em que as suas funções são "data-specific", "lossy" e aprendem automaticamente de exemplos anteriores. "Data-specific" significa que estes apenas conseguem comprimir dados parecidos aos dados com que estes foram treinados. O termo "lossy" significa que os outputs gerados pelo algoritmo se encontrarão degradados quando comparados com os inputs.

Este possui uma camada interna escondida que contém um código, que é utilizado para representar o input e é constituído por duas partes principais: o *encoder,* que mapeia o input para dentro do código, e um *decoder,* que mapeia o código e reconstrói o input.

No entanto, esta cópia iria duplicar o sinal, e é por isso que os *autoencoders* estão restritos numa forma que forçam a reconstrução aproximada do input, o que preserva assim as partes mais relevantes dos dados copiados. Na figura abaixo é visível um diagrama da arquitetura do algoritmo *autoencoder*.

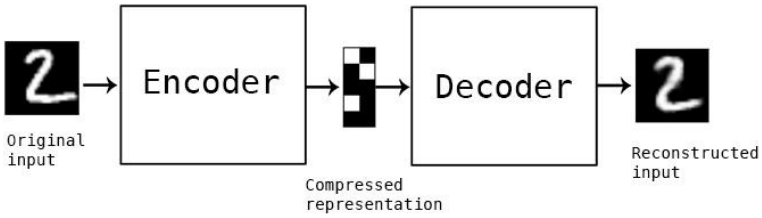


Figura 5 - Autoencoders (Keras 2016)

2.2.5 Aprendizagem reforçada

A aprendizagem reforçada, é um tipo de aprendizagem que se baseia na interação com o ambiente. Esta está em constante crescimento e utilização, juntamente com a produção de uma grande variedade de algoritmos de aprendizagem que podem ser usados para várias aplicações.

Nesta técnica, há sempre um início e um estado final para um agente, no entanto, pode haver caminhos diferentes para chegar ao estado final. O agente aprende para chegar a um objetivo, num ambiente incerto e potencialmente complexo. Este é o cenário em que a aprendizagem reforçada é capaz de encontrar uma solução para um problema. Exemplos de aprendizagem por reforço incluem aspiradores inteligentes, veículos autónomos, entre outros (Deepsense.ai 2018). Por estas razões, esta técnica não foi tida em conta para o problema em questão.

2.2.6 Pré-processamento dos dados

A etapa de pré-processamento dos dados é uma etapa crucial na utilização de algoritmos de *machine learning,* por forma a garantir que os dados disponibilizados aos algoritmos são completos e consistentes, de forma a obter o melhor desempenho e os melhores resultados.

Por norma, a etapa de pré-processamento incorpora as seguintes fases (Sivakumar e Gunasundari 2017):

- Limpeza dos dados;
- Integração dos dados;
- Transformação dos dados.
- Redução dos dados;

2.2.6.1 Limpeza dos dados

É comum que o *dataset* contenha dados em falta, que podem surgir durante a criação do *dataset*, ou através de regras de validação. De qualquer forma, é necessário efetuar o tratamento destes valores em falta. Existem várias alternativas para corrigir este problema, tais como (Sivakumar e Gunasundari 2017):

- Eliminar as linhas do *dataset* com valores em falta;
- Estimar o valor em falta com uma média ou mediana;

2.2.6.2 Integração dos dados

Esta etapa é utilizada aquando da combinação de dados de múltiplas fontes numa *store* de informação concisa. A *metadata* das várias bases de dados é utilizada para impedir a ocorrência de erros nas validações dos *schemas,* inconsistências e redundâncias nos atributos do *dataset* final (Sivakumar e Gunasundari 2017).

2.2.6.3 Transformação dos dados

Nesta etapa, os dados são transformados em formatos próprios para serem utilizados pelos modelos. Esta transformação engloba as seguintes etapas (Sivakumar e Gunasundari 2017):

- Normalização dos dados;
- Técnicas de *Smoothing* como *clustering* e regressão;
- Agregação dos dados;
- Substituição do valor de atributos para o formato categórico.

2.2.6.4 Redução dos dados

Os *datasets* podem conter um grande número de entradas, podendo tornar o processo demasiadamente lento e pouco eficiente quando diretamente utilizados nos algoritmos. Desta forma, frequentemente são realizadas técnicas de redução do tamanho do *dataset*, para a obter um *dataset* com dimensão inferior, mas, ao mesmo tempo, mantendo a sua qualidade e

integridade. Nesta fase poder-se-ão remover atributos irrelevantes ou reduzir o número de exemplos disponibilizados ao algoritmo.

Algumas estratégias para efetuar a redução de um conjunto de dados, são as seguintes (Sivakumar e Gunasundari 2017):

- Operações de agregação dos dados;
- Redução da dimensão do *dataset* pela eliminação de valores irrelevantes, redundantes, fracos, etc.;
- Compressão do dataset através de mecanismos de *encoding;*
- Redução da numerosidade dos dados: os dados são substituídos conjuntos mais pequenos como modelos paramétricos (regressão ou modelos *Log-Linear*) ou não paramétricos (*clustering*, *sampling* ou o uso de histogramas);
- *Concept Hierarchy Generation:* dados em formato *raw* são substituídos por valores conceptuais de mais alto nível, o que possibilita a mineração dos dados a múltiplos níveis de abstração.

2.2.6.5 Matriz de Correlação

Num *dataset* de grande volume, a criação de uma matriz de correlação é importante para a fácil visualização da correlação entre os pares de variáveis do conjunto de dados. Esta matriz representa a correlação os pares, calculada através do coeficiente de correlação, que quantifica a associação entre as várias *features* do *dataset.* Estas *features* devem ser removidas do *dataset* por transmitirem informação similar, o que não melhora em nada o modelo. Abaixo é apresentado um exemplo de uma matriz de correlação. Na matriz, as diagonais têm sempre o valor 1 na interseção de x e y. Nas restantes interseções, um valor perto de 1 significa que existe uma forte correlação positiva entre essas 2 variáveis, enquanto um valor perto de -1 representa uma forte correlação negativa, e o valor 0 significa que não existe nenhuma correlação entre as *features* (Mokhtar Ebrahim 2020).

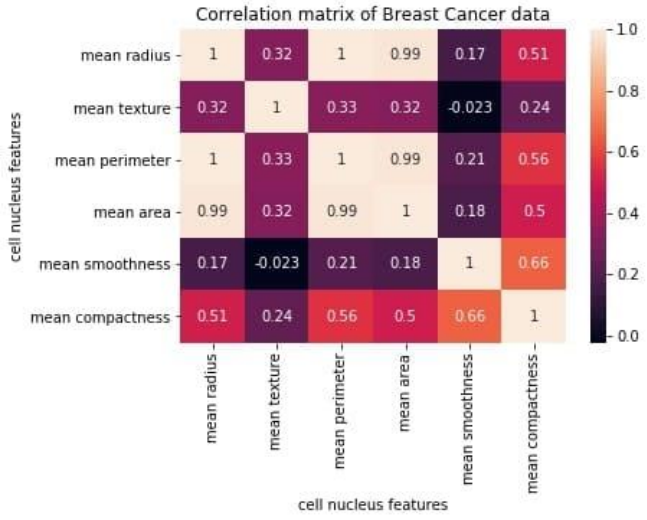


Figura 6 - Exemplo de matriz de correlação (Mokhtar Ebrahim 2020)

2.2.6.6 Skewness

Skewness é uma medida de distorção na distribuição simétrica dos dados, utilizada para medir o desvio na distribuição de uma variável. Uma distribuição normal não contém nenhuma distorção, pelo que a curva gerada pelo gráfico de *skewness* é simétrica dos 2 lados, como é visível no exemplo apresentado abaixo. Se os valores de distorção forem positivos, significa que os dados são positivamente distorcidos e a curva no gráfico gerado encontra-se à direita do centro. Caso contrário, para dados negativamente "skewed", a curva encontra-se à esquerda do centro. Se os dados se encontrarem perfeitamente simétricos, o valor de skewness é zero e a curva é simétrica. Esta métrica é utilizada em projetos de machine learning porque se os valores de uma variável forem distorcidos, esta pode violar as previsões do modelo ou impedir a interpretação da importância das *features* (Nathaniel Jermain 2019).

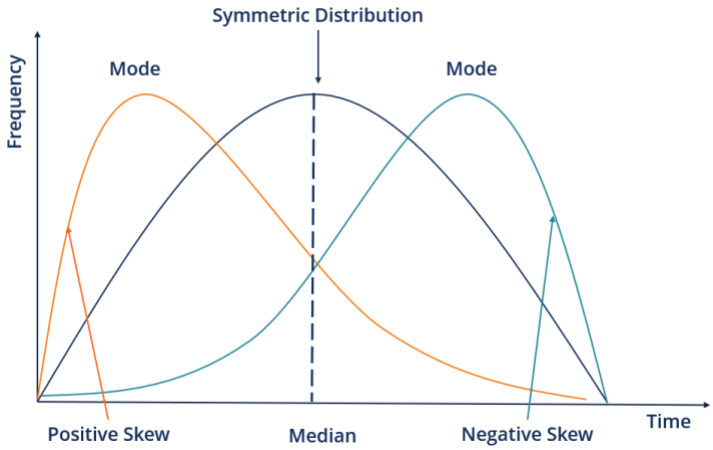


Figura 7 - Exemplo de um gráfico de skewness (CFI 2021)

2.2.6.7 Remoção de variáveis constantes

Variáveis constantes são o tipo de recurso que contém apenas um valor para todos os *outputs* no conjunto de dados. As features com um valor constante não fornecem informações importantes para ajudarem na classificação dos dados em questão e, por isso, são removidas.

2.2.7 Métricas de avalização

Nesta secção serão abordadas várias métricas como a precisão, exatidão, *recall*, especificidade, AUC e F1, que são utilizadas para avaliar os resultados de um algoritmo, para saber se este foi preciso nas suas previsões. Muitas das métricas de avaliação utilizam os seguintes valores nos seus cálculos:

- *True Positive* (TP) – número de exemplos da classe positiva corretamente previstos;
- *True Negative* (TN) – número de exemplos da classe negativa corretamente previstos;
- *False Positive* (FP) – número de exemplos da classe positiva erradamente previstos;
- *False Negative* (FN) – número de exemplos da classe negativa erradamente previstos;

2.2.7.1 Precisão

A precisão *(Agarwal 2019)* de um algoritmo representa à proporção de valores da classe positiva corretamente previstos. Pode ser calculado utilizando a função abaixo:

$$\frac{TP}{TP + FP} \tag{2}$$

2.2.7.2 Exatidão

A exatidão *(Agarwal 2019)* de um algoritmo é das métricas de avaliação mais importantes. Esta é calculada ao dividir número de sucessos pelo número total de previsões, que é representada pela letra n na seguinte equação:

$$\frac{TN + TP}{n} \tag{3}$$

2.2.7.3 Recall

Recall (Agarwal 2019) corresponde à taxa de sucesso de previsão da classe positiva e pode ser calculada utilizando a função abaixo:

$$\frac{TP}{TP + FN} \tag{4}$$

2.2.7.4 F1

A métrica F1 é um número entre 1 e 0 e representa a média harmónica entre a precisão e o *recall (Agarwal 2019)*. Pode ser calculada utilizando a função abaixo.

$$2 * \frac{precisão * recall}{precisão + recall} \tag{6}$$

2.2.7.5 Matthew's Correlation Coefficient (MCC)

Este coeficiente (Boughorbel et al. 2017), conhecido também como o coeficiente Phi, utiliza os valores de TP e FP e é considerado como uma medida equilibrada e que poder ser utilizado mesmo quando as classes são de tamanhos muito diferentes. Este pode ser calculado diretamente da matriz de confusão com a seguinte fórmula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{6}$$

2.3 Abordagens existentes

Existem vários estudos sobre a previsão de falhas e o controlo de qualidade em ambientes de produção na indústria. Na tabela abaixo é apresentado o título, a referência, os algoritmos e as métricas de avaliação utilizadas em alguns desses estudos, de forma resumida.

Tabela 1 –Abordagens existentes em sistemas de previsão de falhas e controlo de qualidade

Título	**Referência**	**Algoritmo**	**Métrica de avaliação**
"Deep Transfer Learning Based on Sparse Autoencoder for Remaining Useful Life Prediction of Tool in Manufacturing,"	(Sun et al. 2019)	*Sparse autoencoder*	Exatidão.
"Implementation of Cyber-physical Production Systems for Quality Prediction and Operation Control in Metal Casting"	(Lee et al. 2018)	*Decision Tree;* *Random Forest;* *Artificial Neural Network;* *SVM.*	Exatidão; Recall; Precisão.
"Multistage Quality Control Using Machine Learning in the Automotive Industry"	(Peres et al. 2019)	*XGBoost;* *Random Forest;* *K-nearest;* *Logistic regression;* *Naive Bayes.*	Recall; Precisão; F1 Score; Curva ROC.
"A hybrid big data analytics method for yield improvement in semiconductor manufacturing."	(Lee et al. 2015)	*SVM;* *Back Propagation Neural Network;*	Exatidão;
"Data mining solves tough semiconductor manufacturing problems".	(Gardner e Bieker 2000)	*Neural Networks;* *Clustering;* *Rule Induction.*	Exatidão; Análise dos *clusters*;
"Using Big Data to Enhance the Bosch Production Line Performance: A Kaggle Challenge"	**(Mangal 2016)**	**XGBoost;** ***Random Forest;*** ***Logistic Regression;***	***Area under ROC curve;***
"Predict Failures in Production Lines"	**(Zhang et al. 2016)**	***Clustering*;** **SVM;** **Random Forest;**	**MCC*;*** ***Area under ROC curve;***

Como é possível verificar nos vários estudos apresentados acima, os algoritmos utilizados, assim como as métricas de avaliação, são muito variadas e estão dependentes da técnica utilizada, do tipo de dados e dos resultados esperados. Este documento teve em consideração estes estudos para a escolha da técnica de aprendizagem, algoritmos e métricas de avaliação a utilizar.

Nos estudos (Sun et al. 2019) e (Gardner e Bieker 2000) foram escolhidas técnicas de aprendizagem não supervisionada, porém, como já foi abordado anteriormente neste documento, para o problema em questão decidiu-se utilizar e comparar técnicas supervisionadas, devido à sua elevada utilização e sucesso na área.

No estudo (Peres et al. 2019), foram comparados vários algoritmos de aprendizagem supervisionada como *XGBoost, Random Forest, K-nearest, Logistic regression* e *Naive Bayes,* avaliando os mesmos através de várias métricas, tais como o Recall, a Precisão, o valor de F1 e a Curva ROC, sendo assim um estudo muito completo sobre esta técnica. Como conclusão, foi possível observar que os algoritmos Random Forest e XGBoost obtiveram os melhores resultados e foram capazes de modelar com sucesso um ambiente muito complexo.

No estudo (Lee et al. 2018) foram, de igual forma, comparados vários algoritmos da técnica supervisionada e foi concluído, através da utilização de várias métricas de avaliação como Recall, exatidão e precisão, que os modelos de *Artificial Neural Network e Random Forest* obtiveram os melhores resultados. Porém, o algoritmo *Random Forest* foi consideravelmente mais rápido, demorando apenas 23 segundos, ao invés dos 87 segundos do algoritmo *Artificial Neural Network.* O algoritmo SVM, por sua vez, demorou apenas 21s e obteve uma exatidão de 0.915, ao invés de 0.926 do algoritmo *Random Forest,* sendo assim também uma boa escolha devido à sua velocidade.

No estudo (Mangal 2016), que teve por base o mesmo conjunto de dados disponibilizados pela Bosch, foram comparados os algoritmos XGBoost, Random Forest e Logistic Regression, e foi utilizada a métrica Area Under ROC Curve para avaliar os resultados de cada. O algoritmo XGBoost obteve a melhor classificação, com o valor 0.718, seguido do algoritmo Random Forest, que obteve 0.709 e, por fim, com o pior resultado, o algoritmo Logistic Regression com 0.614.

Visto que se irá utilizar a técnica supervisionada neste documento, os algoritmos Random Forest, XGBoost e Support Vector Machine mostram ser os algoritmos mais adequados para o problema em questão.

Como métricas de avaliação, a Exatidão, Precisão, Recall, F1 Score, e o MCC são adequadas e serão utilizadas no desenvolvimento deste projeto, para a avaliação dos algoritmos utilizados.

3 Análise de Valor

Nesta secção é apresentada a análise de valor do projeto, tendo em conta os modelos de negócio Peter Koen, Canvas, AHP e a cadeia de valor de Porter.

3.1 New concept development model

"O modelo NCD (*New Concept Development*) fornece uma linguagem comum e uma visão holística do *front end*." (Koen 2020). O modelo divide-se em três partes. A primeira representa o centro do modelo, responsável pela visão, estratégia e cultura. A segunda parte define os cinco elementos de atividade do *front end*. Por fim, a terceira parte consiste nos fatores ambientais externos (Koen 2001). O modelo é representado por cinco elementos-chave: identificação da oportunidade, análise da oportunidade, criação da ideia, seleção da ideia e definição do conceito. Estes elementos serão analisados nas secções seguintes deste documento.

3.1.1 Identificação da oportunidade

Como apresentado na secção 2.1 deste documento, falhas nos componentes nas linhas de produção têm um grande impacto no tempo de inatividade da mesma e nos custos de produção e estão diretamente relacionados com a capacidade de uma empresa ser competitiva em termos de custo, qualidade e desempenho.

Assim sendo, a identificação da oportunidade surgiu pelo facto de este assunto ser um fator muito importante para qualquer fabricante de produtos, como é o caso da Bosch, que disponibilizou vários dados recolhidos nas suas linhas de produção para a criação de um modelo que previsse futuras falhas nos seus componentes.

3.1.2 Análise da oportunidade

Um caso extremo da importância da monitorização dos componentes ao longo da linha de produção é o exemplo, também apresentado na secção 2.1, sobre a empresa japonesa Takata, em que o defeito dos seus componentes levou à falência da empresa e até mesmo à morte de várias pessoas que conduziam veículos com airbags defeituosos fabricados pela mesma.

Num estudo publicado pela Elsevier (Rosa et al. 2018) foi efetuada uma comparação do tempo gasto em cada tarefa em várias linhas de produção, antes e após a análise de vários dados recolhidos e da implementação de soluções por forma a acelerar o processo. O resultado encontra-se apresentado abaixo na figura 8.

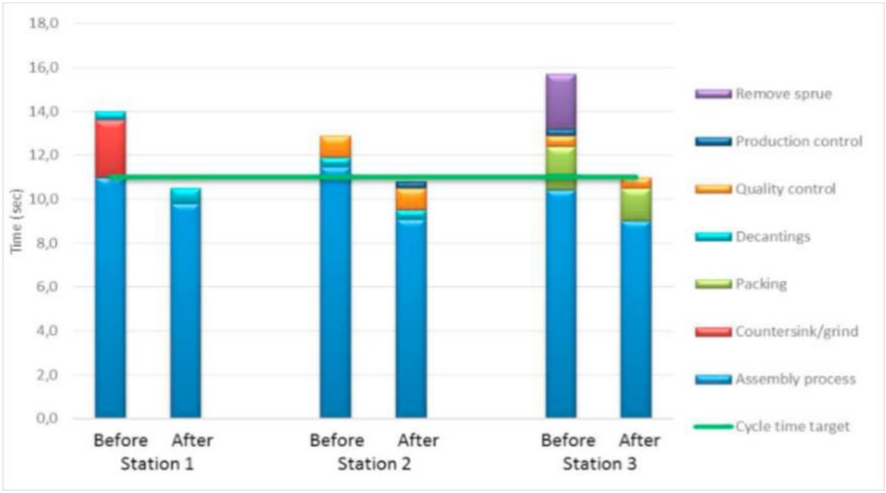


Figura 8 - Tempo de cada tarefa, antes e depois de serem aplicadas as devidas soluções (Rosa et al. 2018)

É possível verificar uma diminuição significativa do tempo gasto em cada estação para cada uma das tarefas, atingindo assim o tempo alvo pré-definido, o que mostra a importância da análise dos dados de uma linha de produção e da implementação das medidas corretas para acelerar o processo, por forma a garantir o cumprimento das metas da organização e a fabricação da maior quantidade de produtos no menor tempo possível, mantendo os padrões de qualidade, por forma a diminuir os custos de produção dos componentes e aumentar o lucro da organização.

3.1.3 Criação da ideia

A ideia já existe e já tem sido analisada ao longo dos anos, porém, as técnicas de *machine learning* têm vindo a melhorar devido aos avanços tecnológicos, e cada vez é mais fácil a aplicação destas técnicas por qualquer organização. Uma dessas organizações é a Bosch. Esta disponibilizou um *dataset* para o estudo e para a criação de modelos de *machine learning,* com vista a detetar componentes defeituosos ou prever em que etapas de produção é mais provável que tais componentes surjam com defeitos.

3.1.4 Seleção da ideia

A ideia selecionada trata-se criação de um modelo para a previsão de componentes defeituosos nas linhas de produção, através do uso de algoritmos de *machine learning,* utilizando os dados disponibilizados pela Bosch.

Outros estudos já foram realizados sobre este assunto, porém, neste projeto será realizada uma comparação entra vários algoritmos de *machine learning* por forma a descobrir qual será o melhor a aplicar nesta situação em específico. Terão ainda de ser aplicadas medidas tendo em

conta o tamanho do *dataset,* que contém mais de 1 milhão de entradas, por forma a conseguir obter bons resultados ainda que com um nível baixo de processamento disponível.

3.1.5 Definição do conceito

Desenvolver um modelo de *machine learning* capaz de prever quais os componentes mais suscetíveis a falhas nas linhas de produção, o que reduzirá o número de componentes com defeito ao longo das mesmas, permitindo aos fabricantes a produção de peças com melhor qualidade, no menor tempo possível, aumentando assim, consequentemente, o lucro da organização.

3.2 Valor

"O objetivo principal da análise de valor é avaliar como aumentar o valor de um item ou serviço com o menor custo, sem sacrificar a qualidade." (Susana Nicola 2020).

No contexto deste estudo, o valor é visto como a prevenção de falhas nas linhas de produção de um fabricante, com o objetivo da redução de componentes defeituosos, por forma a conseguir aumentar o número de componentes produzidos e, ao mesmo tempo, aumentar os lucros da organização, a lealdade dos clientes, a reputação da organização e a segurança no trabalho.

3.2.1 Valor para o cliente

Através da análise e do tratamento de dados, provenientes das linhas de produção de uma organização, é possível obter informações importantes, tais como: que secção da linha de produção é mais demorada, que componente surge com mais defeitos, previsão de falhas nas linhas de produção, entre outros, o que irá proporcionar um aumento de produtividade, da qualidade dos produtos, da capacidade da fábrica, uma redução no custo de produção e, por sua vez, uma possível redução no custo dos produtos para o cliente final.

De seguida, é possível, a partir dos resultados e das previsões obtidas, efetuar alterações às linhas de produção, por forma a melhorar a eficiências e eficácia das mesmas. No estudo publicado pela publicado pela Elsevier (Conceição Rosa, F. J. G. Silva, Luís Pinto Ferreira, Teresa Pereira, Ronny Gouveia 2018) sobre a aplicação de medidas para aumentar a produção as linhas de montagem, após a análise de vários dados sobre as mesmas, como o tempo de montagem, foram identificadas estações com *bottleneck,* balanços inadequados nas linhas de produção, entre outros problemas. A sua resolução levou a uma diminuição considerável nos tempos de produção nas três estações, como foi apresentado acima na secção 2.2.1.2 desde documento. Foi ainda possível verificar um aumento no número de componentes produzidos, como é possível verificar abaixo na tabela 2, mantendo o número de operários.

Tabela 2 - Resultados alcançados através da implementação de soluções. (Rosa et al. 2018)

	Antes da implementação das soluções	**Após a implementação das soluções**
Operadores de linha	3	3
Peças/hora	229	327
Tempo do ciclo (seg)	15.7	11.0

3.2.2 Valor percetível

Nesta secção serão abordados os benefícios e os sacrifícios que este estudo e a sua implementação poderão trazer para o cliente final, utilizando uma perspetiva longitudinal de valor.

A recolha dos dados necessários para a utilização dos mesmos em algoritmos de *machine learning* para a previsão de falhas nas linhas de produção, traz o custo do recrutamento de profissionais capazes de realizar essa recolha e da análise posterior dos resultados, para além da estrutura e dos sistemas informáticos necessários para lidar com grandes quantidades de dados.

Na tabela abaixo é possível encontrar uma perspetiva longitudinal de valor para o estudo em questão.

Tabela 3 - Perspetiva longitudinal de valor

	Benefícios	**Sacrifícios**
Antes da compra	-	Custo da aquisição da infraestrutura necessária;
Na altura da transação	-	Custos de implementação;
Depois da compra	-	Custos de manutenção da infraestrutura; Custos relacionados com os profissionais de tratamento dos dados;
Depois da implementação	Sistema de previsão de falhas; Produtividade aumentada; Custos de produção reduzidos; Custo de produto reduzido; Melhor reputação da organização;	Custos de manutenção da infraestrutura; Custos relacionados com os profissionais de tratamento dos dados;

3.2.3 Proposta de valor

A proposta de valor deste estudo é a criação de um modelo de controlo de qualidade de uma linha de montagem, utilizando algoritmos de *machine learning* que, quando fornecido com dados das linhas de produção, consiga prever, com grande precisão, falhas no processo de fabricação.

Uma fábrica poderá utilizar um modelo deste tipo para efetuar um melhor, e mais eficiente, controlo das linhas de produção, devido ao grande volume e variedade de dados que são gerados, que serão analisados e tratados pelo modelo, de forma a gerar informação chave, que sirva de apoio ao fabricante, para que este consiga aumentar a sua produtividade e eventualmente trazer produtos de melhor qualidade para os clientes.

3.3 Modelo Canvas

Nesta secção será apresentado, abaixo na figura 9, o modelo Canvas para o estudo em questão.

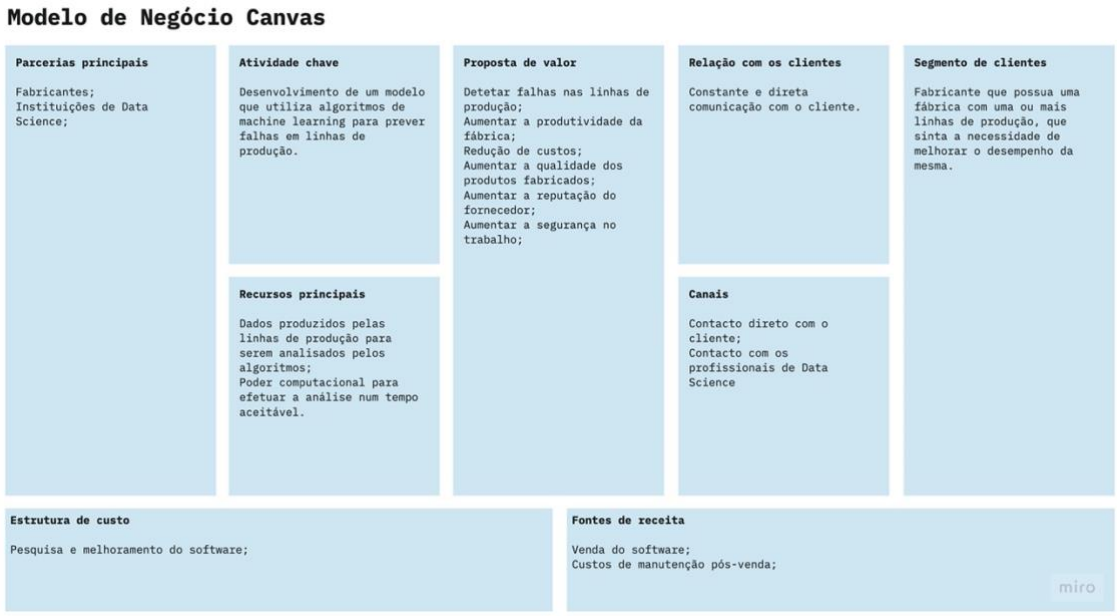


Figura 9 - Modelo de Negócio Canvas

Como é possível verificar no modelo Canvas acima, é espectado conseguir utilizar o modelo de previsão de falhas nas linhas de produção, com vista a aumentar a produtividade da fábrica, a qualidade dos produtos fabricados, a reputação do fabricante, a segurança no trabalho e reduzir os custos de produção.

As parcerias e clientes principais serão qualquer fabricante que sinta a necessidade de melhorar o desempenho das suas linhas de produção e efetuar a previsão de falhas nas mesmas. A fonte de receita será a venda do software e o futuro suporte efetuado ao mesmo, enquanto os custos subjacentes são relativos aos custos de pesquisa e de manutenção do *software*. Para a implementação do software serão necessários dois recursos principais, os dados relativos às linhas de produção e poder computacional suficiente disponível, de forma a conseguir obter os resultados num tempo aceitável, dependendo da quantidade de dados disponibilizados.

3.3.1 Cadeia de Valor de Porter

O modelo de Cadeia de Valor de Porter representa um conjunto de atividades que uma empresa realiza, a fim de entregar um produto com valor para o mercado. O conceito foi criado por Michael Porter em 1985, no seu livro intitulado “Competitive Advantage: Creating and Sustaining Superior Performance” (Porter 1985). Na figura abaixo é explicada a forma de como se poderia utilizar o modelo de Cadeia de Valor para a analisar o valor do negócio do estudo em questão.

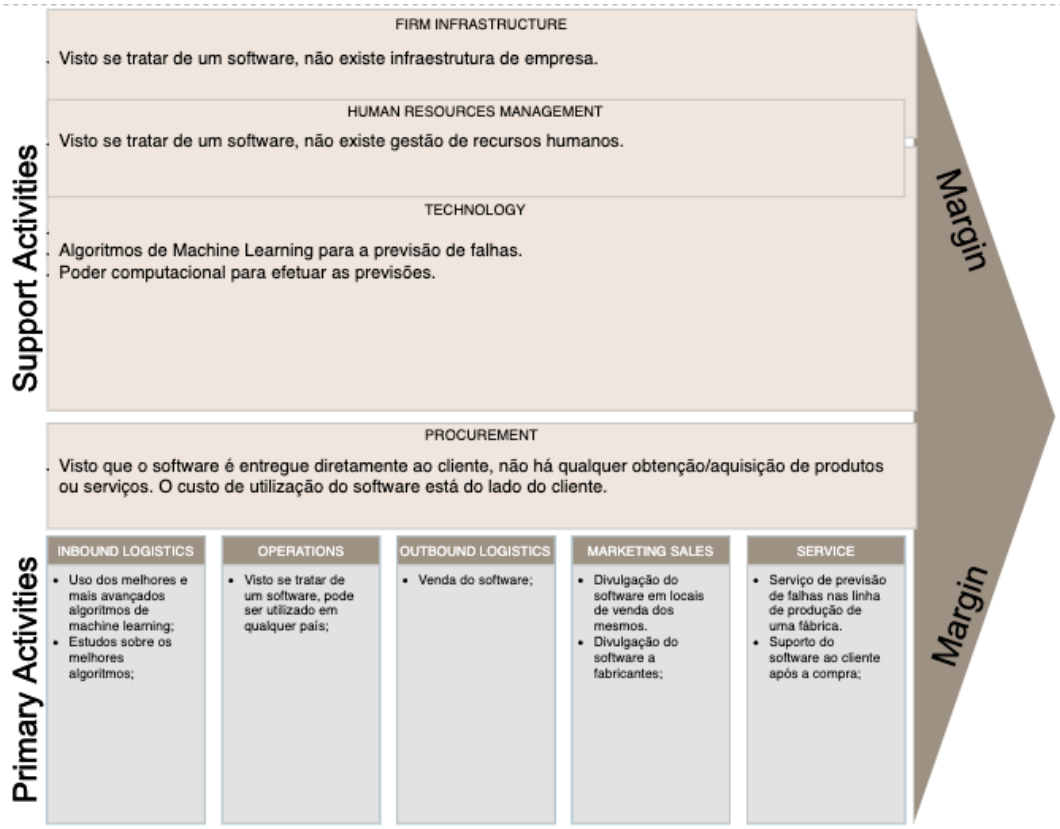


Figura 10 - Cadeia de Valor de Porter

Visto se tratar de um software, não existe infraestrutura nem gestão de recursos humanos, como numa organização. Tecnologia é o ponto principal, visto se tratar de uma análise a um software. Este utiliza algoritmos de *machine learning* e necessita de ser corrido em computadores para efetuar as previsões.

Nas atividades principais, não existe *procurement*, ou seja, não existe qualquer aquisição de produtos. Como logística de entrada, efetuam-se estudos sobre os melhores algoritmos de *machine learning* a utilizam-se os mesmos para aprimorar o modelo. As operações podem ser efetuadas em qualquer local, visto de tratar de um *software.* Como logística de saída, efetua-se a venda do *software*. Em termos de *marketing*, o *software* pode ser divulgado em vários locais online, próprios para tal, e dados a conhecer diretamente aos fabricantes. É disponibilizado um modelo de previsão de falhas em linhas de produção, assim como o suporte e manutenção do mesmo após a compra.

3.3.2 Método AHP

"AHP (*Analytic Hierarchy Process*) é um método de tomada de decisão que foi desenvolvido pela empresa Saaty, a técnica usada para organizar relacionamentos complexos entre elementos em estrutura ou um sistema com base em julgamento subjetivo, como a experiência" (Mohammed e Rami 2015). AHP é uma técnica de medição realizada através de

comparações de pares. Esta depende do julgamento de especialistas para derivar escalas de prioridade" (Saaty 2008).

O primeiro passo no desenvolvimento do método AHP é a criação de uma estrutura hierárquica, em que o primeiro patamar é o objetivo principal, seguido dos atributos/critérios e, por fim, as alternativas, como terceiro nível. Na figura abaixo é apresentada essa estrutura.

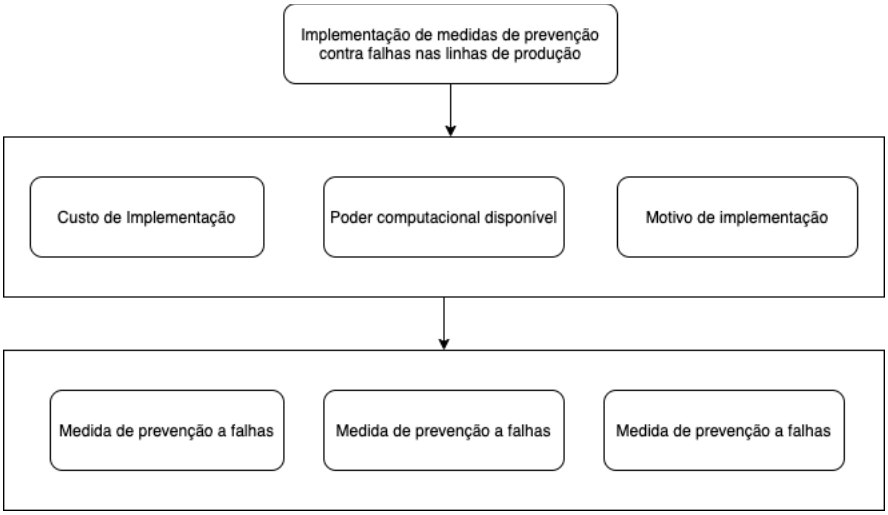


Figura 11 - Estrutura hierárquica

A implementação de medidas de prevenção contra falhas nas linhas de produção é tida como o objetivo principal. Foram também definidos três critérios: o custo de implementação, o poder computacional disponível e o motivo da implementação. Tendo em conta estes critérios, é então possível implementar medidas de prevenção a falhas. Na tabela abaixo foram definidos pesos para cada um dos critérios.

Tabela 4 - Matriz AHP de comparação de pares

	Motivo de implementação	**Poder computacional disponível**	**Custo de implementação**
Motivo de implementação	1	7	5
Poder computacional disponível	1/7	1	1/5
Custo de implementação	1/5	5	1
Soma	1.34	13	6,2

Como é possível verificar na tabela 4, o motivo de implementação é o que possui o maior impacto, visto ser o motivo que leva um fabricante a decidir implementar medidas para acelerar

o processo de produção e, assim, melhorar a produtividade das linhas de montagem. O segundo ponto com mais impacto é o custo de implementação, pois, dependendo do tamanho da fábrica e do número de linhas de produção, este custo pode escalar muito rapidamente e provocar um grande impacto financeiro na organização, apesar de no longo prazo, e com as medidas implementadas, a organização poder vir a recuperar esse investimento. Por último, o poder computacional é visto como o de menor importância.

Na tabela 5 é apresentado a matriz AHP de pares normalizada, em que se efetuou a divisão de cada célula da matriz pelo valor da soma correspondente.

Tabela 5 - Matriz AHP normalizada

	Motivo de implementação	**Poder computacional disponível**	**Custo de implementação**
Motivo de implementação	0,746	0,538	0,806
Poder computacional disponível	0,106	0,077	0,032
Custo de implementação	0,149	0,384	0,161

4 Design

Neste capítulo serão efetuadas comparações entre várias alternativas para a implementação deste projeto, tendo em consideração o que foi apresentado anteriormente neste documento, tal como, a análise dos vários estudos realizados sobre temas similares.

4.1 Linguagens e *Frameworks*

Este capítulo aborda e compara várias linguagens de programação e *frameworks* que são possíveis de serem utilizadas na realização deste projeto e o porquê de se optar por uma ao invés de outra.

Um ponto importante na escolha de uma linguagem de programação é a sua percentagem de utilização pela comunidade, pois isso irá afetar o apoio disponível por parte da mesma. Os resultados do questionário realizado pelo Stack Overflow em 2020 (Stack Overflow 2020), encontram-se na tabela abaixo.

Tabela 6 - Comparação entre linguagens de programação

Linguagem	**Popularidade**	**Preferência em continuar a utilizar**	**Preferência em deixar de utilizar**	**Gostariam de começar a utilizar**
Python	44.1%	66.7%	33.3%	30%
R	5.7%	44.5%	55.5%	5.1%
Julia	0.9%	62.2%	37.8%	2.3%

Como é possível verificar, o resultado do inquérito aponta para uma diferença elevada na popularidade e na percentagem de programadores que gostariam de começar a utilizar Python, em comparação com as outras linguagens. Na percentagem de programadores que gostariam de deixar de utilizar, Python recebe o valor mais baixo, ao contrário de R, mostrando assim que Python é, em comparação com as restantes, a linguagem preferida pela comunidade.

Python é então a quarta linguagem de programação mais utilizada no mercado e com grande popularidade, tendo assim um grande apoio online por parte da comunidade (Stack Overflow 2020). Esta é, ao mesmo tempo, reconhecida como sendo das linguagens de programação mais fáceis de aprender, tendo uma curva de aprendizagem relativamente pequena. Por estas razões, decidiu-se que esta seria a linguagem a utilizar no desenvolvimento deste projeto.

Outras ferramentas utilizadas no desenvolvimento do projeto são as seguintes:

- Pandas;
- Matplot;
- Scikit-learn;
- Seaborn;
- Numpy;
- Scipy;

4.2 Abordagem

Neste capítulo são apresentados o tipo e o estado dos dados do *dataset* que irá ser utilizado e testado no desenvolvimento do projeto, assim como os processos de pré-processamento dos mesmos. Será ainda apresentado um design da implementação dos modelos de *machine learning* e um diagrama de estados do processo final do sistema.

4.2.1 Conjunto de dados

O conjunto de dados contém um grande número de recursos anónimos. Os recursos são nomeados de acordo com uma convenção que informa a linha de produção, a estação na linha e um número da peça. A exemplo, no *dataset* numérico L2_S22_F2222 é um recurso processado na linha 2, na estação 22 e trata-se da *feature* com o número 2222, enquanto no dataset temporal, L2_S22_D2222 representa que a peça passou na linha 2, na estação 2 e que o valor da *feature* corresponde à data com ID 2222.

Devido à grande dimensão do conjunto de dados, num total de 1,183,747 entradas, 969 *features* numéricas e 1157 *features* temporais. Os mesmos foram disponibilizados em vários ficheiros CSV, de acordo com as características que contêm: numérico, categórico e temporal e divididos em conjuntos de treino e teste. Cada parte contém um identificador único, que corresponde à coluna 'Id'. O objetivo é prever que partes irão falhar na linha de produção, representadas pelo atributo '*Response'* com o valor 1. Em suma, o dataset contém:

- Número de entradas: 1 183 747;
- Número de *features* numéricas: 968;
- Número de *features* temporais: 1157;
- Número de *features* categóricas: 2140;
- Percentagem de produtos com defeito: 0.58%;
- Percentagem de valores em falta: 78.5%

Na tabela 7 é apresentado um excerto do conjunto de dados relativo às datas:

Tabela 7 - Excerto do dataset de datas

	Id	L0_S0_D1	L0_S0_D3	L0_S0_D5	...
0	4	82.24	82.24	82.24	...
1	6	NaN	NaN	NaN	...
2	7	1618.70	1618.70	1618.70	...
...	...	...	...	...	...
9999	19923	1379.75	1379.75	1379.75	...

Na tabela 8 é apresentado um excerto do conjunto de dados relativo aos valores numéricos:

Tabela 8 - Excerto do dataset numérico

	Id	L0_S0_F0	L0_S0_F2	L0_S0_F4	...
0	4	0.030	-0.034	-0.197	...
1	6	NaN	NaN	NaN	...
2	7	0.088	0.086	0.003	...
...	...	...	...	...	...
9999	19923	-0.003	-0.019	-0.015	...

4.2.2 Técnicas de pré-processamento

Os dados encontram-se muito desequilibrados (apenas 0.58% com defeito) e com muitos valores em falta (78.5%) porque muitas estações nas linhas de produção têm o mesmo propósito, o que leva a que uma peça apenas passe por uma fração das linhas de produção e, como resultado, grande porção do *dataset* contém valores em falta.

Assim sendo, é necessário efetuar técnicas de pré-processamento de dados, abordadas em maior detalhe anteriormente na secção 2.3.3 deste documento, sendo estas a eliminação das linhas do *dataset* com valores em falta, reduzindo o tamanho do *dataset*, o que pode, no entanto, causar a eliminação de dados importantes. A segunda alternativa é estimar o valor em falta com um valor provável, como uma média ou mediana, mantendo assim o tamanho do *dataset* original. Esta técnica remove o risco de serem eliminados valores importantes, mas, no entanto, esta ação pode levar à introdução de dados falsos no *dataset*. Assim sendo, a primeira abordagem foi escolhida como sendo a melhor técnica para o conjunto de dados, tendo em conta a grande dimensão do mesmo.

A remoção de *features* com menor ocorrência é uma etapa importante, pois esta irá reduzir a quantidade de dados com pouca importância para o projeto. Esta etapa tem o nome de "*Feature Selection*" ou Engenharia de Recursos e pode ser realizada ao utilizar várias funções como "*feature_selection*" e "*feature_importance*" disponibilizadas pela ferramenta SkLearn e ao analisar gráficos como o "*heatmap*", disponibilizada pela ferramenta Seaborn, sobre a comparação da importância entre as várias *features*. Este processo será apresentado em maior detalhe no capítulo 5 deste documento.

Devido ao elevado número de entradas e à limitada capacidade de processamento disponível, alguns dados terão de ser ignorados, por forma a tornar possível a obtenção de um resultado num tempo aceitável.

4.2.3 Modelo

Dos vários estudos apresentados e comparados anteriormente na secção 2.3 deste documento, os algoritmos selecionados para a implementação deste projeto e futura comparação de resultados, foram os algoritmos Random Forest, XGBoost e Support Vector Machine, por mostrarem ser dos mais indicados para este tipo de problema, por serem algoritmos de aprendizagem supervisionada, por serem dos mais utilizados nos estudos apresentados na secção 2.3 e por terem obtido bons resultados nesses mesmos estudos.

Como os conjuntos de dados foram disponibilizados em formato CSV, é necessário importar os mesmos através da implementação de funções de leitura em Python para esse efeito. Após os valores terem sido importados para o sistema, é possível aplicar as técnicas de pré-processamento dos dados mencionadas na secção anterior, normalizando os valores. Caso os dados sejam de qualidade, é possível passar à etapa de *feature selection* e, de seguida, definir os parâmetros para a utilização dos conjuntos de treino e de teste, disponibilizados para treinar e testar os algoritmos. Como estes conjuntos foram disponibilizados de antemão, não é necessário dividir os conjuntos para este efeito.

Como métricas de avaliação dos mesmos, várias técnicas de avaliação de aprendizagem supervisionada podem ser utilizadas, como é o caso da Exatidão, da Precisão, do Recall, do F1 Score, e da Curva ROC.

Caso tenham sito obtidos bons resultados, é possível obter as previsões das peças que irão falhas nas linhas de produção, caso contrário, os passos anteriores terão de ser repetidos, até ser obtido um bom resultado. Abaixo é apresentado um fluxograma que representa o *flow* deste sistema.

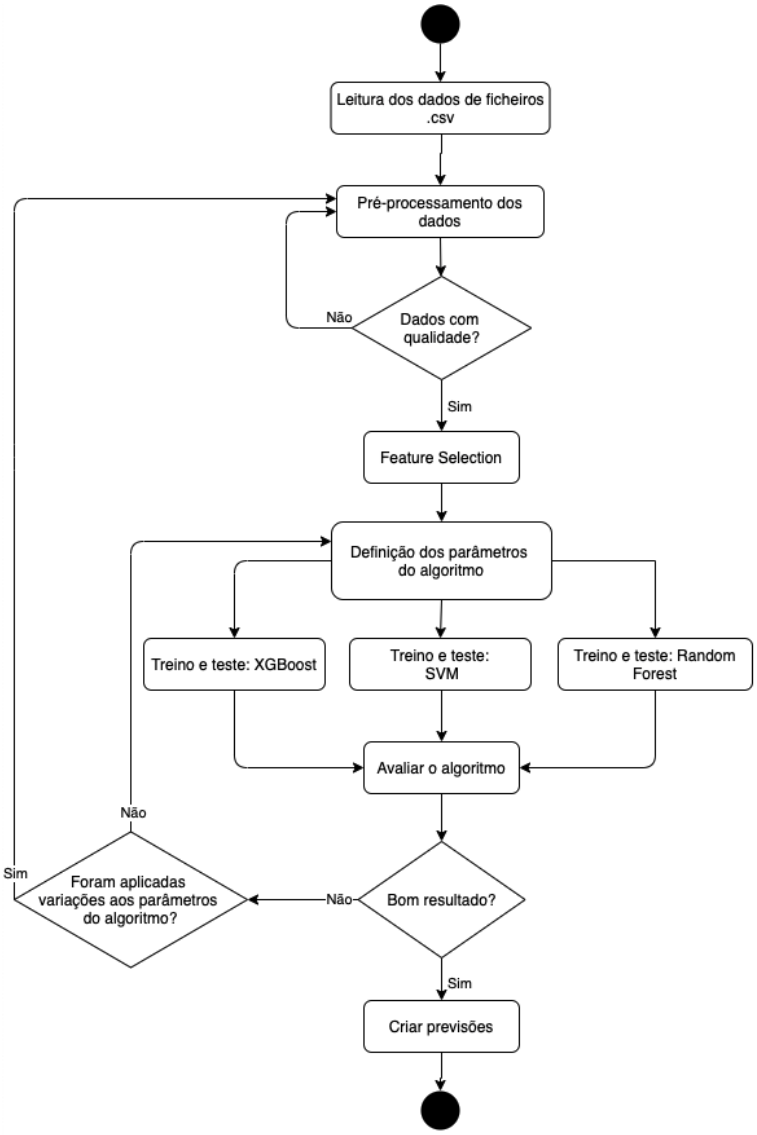


Figura 12 - Fluxograma dos modelos

5 Implementação

Este capítulo demonstra e justifica a forma como as decisões apresentadas nos capítulos anteriores deste documento foram implementadas, juntamente com várias capturas de ecrã, quer do código implementado, quer dos resultados e avaliações finais realizadas.

5.1 Ambiente de desenvolvimento

Um dos primeiros passos para o desenvolvimento de um projeto de programação é a escolha de um editor de código. Para este projeto foi escolhido o editor *open source* intitulado *Visual Studio Code,* por ser gratuito, leve, e por ter um grande suporte da comunidade, com muitas extensões uteis para o desenvolvimento com várias linguagens de programação (Visual Studio Code 2021).

Em segundo lugar, e tal como foi referido anteriormente neste documento, a linguagem de programação escolhida para a implementação do modelo foi Python pelas razões previamente presentadas. Para a instalação de Python na máquina onde o projeto foi desenvolvido, foi utilizado Homebrew, um gerenciador de pacotes para macOS, por forma a simplificar o processo, ao correr o comando “brew install python3” no terminal (Homebrew 2021).

Tendo o interpretador de Python sido instalado na máquina, foi necessário instalar uma extensão para o *Visual Studio Code* para ser possível codificar na linguagem. O editor possui várias extensões que permitem a utilização de Python, disponíveis no *Marketplace* do editor.

Para o desenvolvimento do projeto foram importadas várias bibliotecas importantes que permitem a utilização de funções matemáticas, de *machine learning*, de criação de gráficos, de avaliação, etc., algumas referidas anteriormente na secção 4.1 deste documento, como é visível na figura abaixo:

```
import pandas as pd
import numpy as np
import xgboost as xgb
from xgboost import plot_importance
from xgboost import plot_tree
from sklearn.model_selection import train_test_split
from sklearn import datasets, metrics, model_selection, svm
import scikitplot as skplt
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import confusion_matrix
from sklearn import model_selection
from sklearn import preprocessing
from sklearn.metrics import accuracy_score, f1_score, precision_score
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sn
import traceback
import re
```

Figura 13 - Bibliotecas importadas

Foi ainda criado um repositório Git privado, para efetuar o controlo de versões do código desenvolvido, por forma a garantir as boas práticas de engenharia.

5.2 Tratamento dos dados

Neste capítulo será demonstrada a importação dos dados dos vários ficheiros CSV utilizados, e o pré-processamento efetuado aos mesmos, antes de estes serem aplicados aos algoritmos de *machine learning*.

5.2.1 Leitura dos dados

Os dados utilizados neste projeto foram disponibilizados em ficheiros CSV, visíveis na figura abaixo. Para a sua importação foram testadas duas funções Python, das bibliotecas Dask e Pandas, por se tratar de uma grande quantidade de dados, como é possível verificar pelo tamanho dos ficheiros abaixo na figura 14.

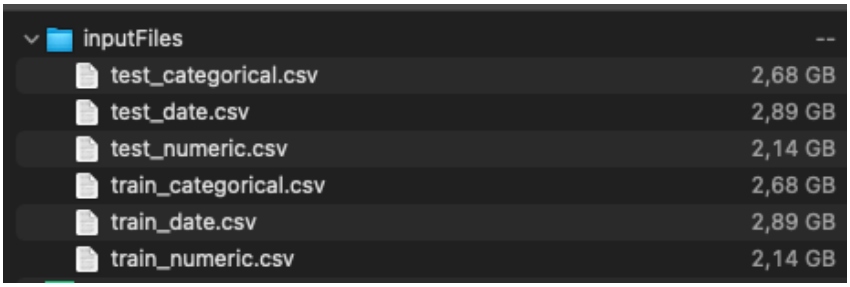


Figura 14 - Ficheiros utilizados e o seu tamanho

Em primeiro lugar foi utilizada a função *read_csv*() da biblioteca Dask e, pelo facto de que a função analisa o ficheiro por partes, utiliza paralelismo e não carrega o resultado em memória, esta é substancialmente mais rápida a terminar o processamento, que a função da biblioteca Pandas, quando ficheiros de grande dimensão são carregados. Porém, a biblioteca e os *dataframes* Pandas são dos mais utilizados em projetos de *machine learning* por serem facilmente aplicáveis nos algoritmos de *machine learning* e, durante o desenvolvimento, foi notável que Pandas é mais utilizada pela comunidade, pela diferença considerável em suporte por parte da mesma.

Dask disponibiliza ainda a função *compute*() para converter um *dataframe* Dask num *dataframe* Pandas. No entanto, devido ao tamanho dos ficheiros, esta função tornou-se extremamente lenta ao ser executada. Sendo assim, decidiu-se importar os dados diretamente para memória com a função *read_csv*() de Pandas, como é visível na imagem 15 abaixo.

```
# load data
chunk_numeric = pd.read_csv(data_dir + "train_numeric.csv", nrows=chunksize, engine="c",
                            names=header_numeric, index_col="Id")
chunk_date = pd.read_csv(data_dir + "train_date.csv", nrows=chunksize, engine="c",
                         names=header_date, index_col="Id")
chunk_categorical = pd.read_csv(data_dir + "train_categorical.csv", nrows=chunksize,
                                names=header_categorical, index_col="Id", dtype="object",
                                engine="c")
```

Figura 15 - Leitura de dados dos ficheiros CSV

No excerto de código apresentado na figura 15, é realizada a importação para memória dos três ficheiros CSV de treino, que contêm os dados numéricos, as datas e os dados categóricos, respetivamente presentes nos ficheiros "train_numeric.csv", "train_date.csv" e "train_categorical.csv".

Novamente, devido ao tamanho dos ficheiros, e à capacidade de processamento limitada, decidiu-se apenas carregar parte de cada um dos ficheiros, através dos parâmetros introduzidos nas funções de leitura, "*nrows*" e "*names*". A primeira limita o número de linhas lido de cada ficheiro e a segunda restringe as colunas importadas. As variáveis "*header_numeric*", "*header_date*" e "*header_categorical*" são *arrays* que contêm o nome das colunas a importar. Por forma a acelerar as funções de importação dos dados e o futuro processamento dos dados, decidiu-se reduzir o número de linhas a importar de cada ficheiro para 10000 (*chunksize* = 10000), ao invés de reduzir o número de *features*.

Para acelerar o processo decidiu-se também não importar os ficheiros de teste disponibilizados e, ao invés, utilizar a função *train_test_split*() da biblioteca sklearn para dividir os dados em *dataframes* de teste e treino, antes de estes serem aplicados aos algoritmos.

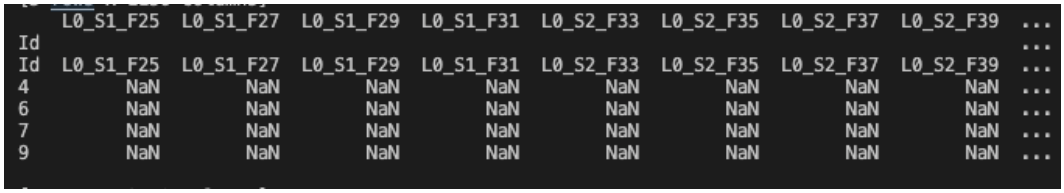
Na imagem 16, 17 e 18 abaixo são apresentadas as primeiras 4 entradas de cada um dos ficheiros importados.

	L0_S0_F0	L0_S0_F2	L0_S0_F4	L0_S0_F6	L0_S0_F8	L0_S0_F10	L0_S0_F12	L0_S0_F14	L0_S0_F16	...
Id										...
Id	L0_S0_F0	L0_S0_F2	L0_S0_F4	L0_S0_F6	L0_S0_F8	L0_S0_F10	L0_S0_F12	L0_S0_F14	L0_S0_F16	...
4	0.03	-0.034	-0.197	-0.179	0.118	0.116	-0.015	-0.032	0.02	...
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
7	0.088	0.086	0.003	-0.052	0.161	0.025	-0.015	-0.072	-0.225	...
9	-0.036	-0.064	0.294	0.33	0.074	0.161	0.022	0.128	-0.026	...

Figura 16 – Primeiras 4 entradas do ficheiro train_numeric.csv importado

Id	L0_S0_D1	L0_S0_D3	L0_S0_D5	L0_S0_D7	L0_S0_D9	L0_S0_D11	L0_S0_D13	L0_S0_D15	L0_S0_D17	...
263										
4	82.24	82.24	82.24	82.24	82.24	82.24	82.24	82.24	82.24	...
NaN										
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
NaN										
7	1618.7	1618.7	1618.7	1618.7	1618.7	1618.7	1618.7	1618.7	1618.7	...
NaN										
9	1149.2	1149.2	1149.2	1149.2	1149.2	1149.2	1149.2	1149.2	1149.2	...
NaN										

Figura 17 – Primeiras 4 entradas do ficheiro train_date.csv importado



	L0_S1_F25	L0_S1_F27	L0_S1_F29	L0_S1_F31	L0_S2_F33	L0_S2_F35	L0_S2_F37	L0_S2_F39	...
Id									...
Id	L0_S1_F25	L0_S1_F27	L0_S1_F29	L0_S1_F31	L0_S2_F33	L0_S2_F35	L0_S2_F37	L0_S2_F39	...
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

Figura 18 - Primeiras 4 entradas do ficheiro train_categorical.csv importado

Como é possível verificar, existem muitos valores NaN, principalmente no ficheiro com dados categóricos, pelo que terão de ser tratados, por forma a melhorar o resultado dos algoritmos. Os métodos aplicados para tal efeito serão apresentados na secção seguinte.

5.2.2 Pré-processamento dos dados

Nesta fase decidiu-se, em primeiro lugar, efetuar uma limpeza aos dados, ao eliminar os valores em falta, por se estar a lidar com uma grande quantidade de dados, ao invés de outras opções, como a substituição por outro valor. Para tal, foi utilizada a função *DataFrame.dropna*() da biblioteca Pandas, para cada um dos *dataframes*, como é demonstrado na figura abaixo:

```
chunk.dropna(how="all", inplace=True)
```

Figura 19 - Função dropna

O primeiro argumento da função determina se a linha só é apagada quando todos os campos são vazios, com o valor "*all*", ou se basta apenas um ser vazio para a mesma ser apagada, com o valor "*any*". O atributo *inplace* com o valor *True* apenas significa que nenhum valor é retornado, a operação é efetuada diretamente no *dataframe* "*chunk*", senão, a operação será efetuada e retornada num novo *dataframe*.

De seguida, procedeu-se ao processamento dos dados e à adição de algumas *features* que se achou importantes, apresentadas de seguida. Durante este processo notou-se que era desgastante o facto de que a cada alteração efetuada ao modelo, era necessário executar todo o código novamente. Dessa forma, decidiu-se guardar os dados carregados em memória, utilizando uma tabela HDF5 (*Hierarchical Data Format*), que é largamente utilizada em projetos de *data mining* e *machine learning* com um grande volume de dados, por ser consideravelmente mais rápida a guardar e a ler os mesmos dados em ficheiros .csv (Ilia Zaitsev 2019).

Antes de os dados serem carregados na tabela HDF, adicionou-se 3 novas *features* aos dados numéricos, o valor mínimo, máximo e médio de cada secção por linha, com os nomes "*name _min*", "*name_max*" e "*name _mean*" em que *name* é o número da secção, através das funções *mean*(), *max*() e *mean*(), apresentado abaixo na figura 19, em que o parâmetro *axis* representa o número da coluna do *dataset* a que o cálculo irá ser aplicado e, visto que o índice do *dataframe* se encontra na posição 0, o cálculo irá ser aplicado à coluna 1.

```
# new features
chunk[name + "_min"] = chunk.min(axis=1)
chunk[name + "_max"] = chunk.max(axis=1)
chunk[name + "_mean"] = chunk.mean(axis=1)
```

Figura 20 - Novas features adicionadas aos dados numéricos

Para os dados que contêm os valores temporais, que originaram do ficheiro train_date.csv, foram também criadas novas *features* que se achou que faziam sentido para o caso em questão. Tal como nos dados numéricos, foram adicionadas as *features* temporais com os valores mínimos e máximos, assim como o tempo decorrido entre a maior e a menor data. O código resultante é apresentado de seguida:

```
chunk[name + "min_date"] = chunk.min(axis=1)
chunk[name + "_max_date"] = chunk.max(axis=1)
chunk[name + "_elapsed_time"] = pd.to_numeric(chunk[name + "_max_date"], errors='coerce')
    - pd.to_numeric(chunk[name + "_min_date"], errors='coerce')
```

Figura 21 - Novas features adicionadas aos dados temporais

Na *feature* "*name_elapsed_time*" em que *name* é o valor da secção, tendo em conta que os *dataframe* são do tipo "*object*", foi necessário a conversão dos *dataframes* para um tipo de dados numérico, float64 ou int64, dependendo do valor de cada campo, para a subtração ser possível. O valor "*coerce*" no atributo *errors* indica que quando uma conversão for dada como inválida, o campo irá ser substituído pelo valor NaN. Outros valores possíveis seria "*raise*", que lançaria uma exceção, ou "ignore", que simplesmente mantinha o valor original (Pandas 2021).

Para os valores categóricos não foi adicionada nenhuma nova *feature.* Depois da conversão dos dataframes para um tipo numérico e da subtração dos valores, foi detetado que alguns campos apresentavam os valores "inf" e "-inf", o que causava erros na aplicação dos algoritmos de *machine learning*. Desta forma, foi necessário remover estes valores e substituí-los pelo valor "NaN", ao utilizar a função *replace*() de Pandas, da forma abaixo apresentada:

```
chunk.replace([np.inf, -np.inf], np.nan, inplace=True)
```

Figura 22 - Substituição dos valores inf e -inf por NaN

Por fim, carregou-se os *dataframes* para uma tabela HDF através da função *put*() (visível abaixo na figura 23), com o parâmetro *key* a representar o nome de uma linha ou secção, o parâmetro *value* com o *dataframe* relativo a essa linha ou secção, e por fim o parâmetro *format* com o valor "*table*". Este último parâmetro indica que o *dataframe* irá ser guardado como uma estrutura PyTable que, possivelmente, terá um desempenho inferior ao outro possível formato "*fixed*", porém, este é mais flexível e permite operações de pesquisa, que poderiam ser necessárias no futuro.

```
hdfTable.put(name, value=chunk, format="table")
```

Figura 23 - Função que permite guardar *dataframes* Pandas numa tabela HDF

Com os dados carregados numa tabela HDF é possível a qualquer momento e de forma mais rápida, carregar para memória os valores previamente importados e tratados dos ficheiros .csv e aplicá-los aos algoritmos de *machine learning*. Para importar dados da tabela é necessário apenas invocar a função "hdfTable.get(key)" e guardar o retorno num *dataframe*.

De seguida, efetuou-se a análise da "*skewness*", ou distorção, dos dados importados, através da função *skew*() da biblioteca *scipy*, como é mostrado na figura abaixo. Se os valores de distorção forem positivos, significa que os dados são positivamente distorcidos e a curva no gráfico gerado encontra-se à direita do centro. Caso contrário, para dados negativamente "*skewed*", a curva encontra-se à esquerda do centro. Se os dados se encontrarem perfeitamente simétricos, o valor de *skewness* é zero e a curva é simétrica.

```
# Skewness
num_feats = table.dtypes[table.dtypes != 'object'].index
skew_feats = table[num_feats].skew().sort_values(ascending=False)
skewness = pd.DataFrame({'skew': skew_feats})
sns.distplot(skew_feats)
plt.show()
```

Figura 24 - *Skewness* dos dados

Após a execução da função, gerou-se um gráfico com o resultado. Este gráfico é apresentado de seguida na figura 25. Ao analisar o gráfico gerado, é possível afirmar que os dados não se encontram distorcidos.

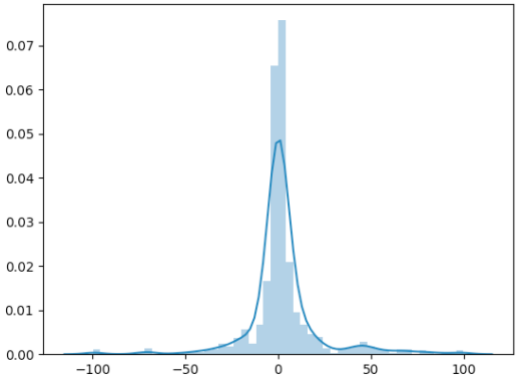


Figura 25 - Gráfico de Skewness

De seguida, removeu-se as *features* constantes. Estas são *features* que contêm apenas um valor para todos os outputs, o que as tornam pouco importantes e redundantes para o modelo. Para isso, utilizou-se a função *VarianceThreshold* da biblioteca sklearn. A função requer um parâmetro de limite, ao qual se passou o valor zero, o que significa que esta filtrará todos os recursos com variação zero. O seguinte código foi então criado para este efeito:

```
# REMOVING CONSTANT FEATURES

constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(table)

print(len(table.columns[constant_filter.get_support()]))

constant_columns = [
    column for column in table.columns if column not in table.columns[constant_filter.get_support()]]

for column in constant_columns:
    print(column)

table = constant_filter.transform(table)
```

Figura 26 - Remoção de *features* constantes

Por fim, construiu-se a matriz de correlação dos dados que irão ser aplicados aos modelos, através da função corr() da biblioteca sklearn. Esta matriz representa a correlação entre pares de variáveis nos dados, calculada através do coeficiente de correlação, que quantifica a associação entre as várias *features* do *dataset.* Estas *features* devem ser removidas do *dataset* por transmitirem informação similar, o que não melhora em nada o modelo. Devido ao tamanho do *dataset*, selecionou-se uma pequena parte do mesmo para criar um gráfico do tipo *heatmap*, apresentado abaixo na figura 27, da matriz de correlação dessa amostra.

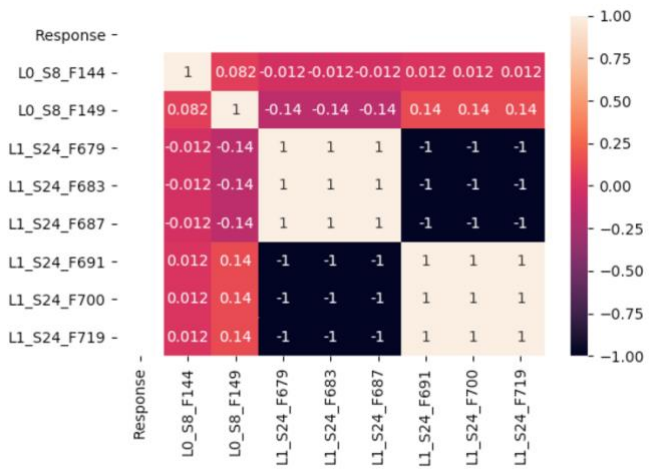


Figura 27 - Matriz de correlação de uma amostra do dataset

Como é possível verificar no gráfico acima, em que cada linha e coluna representam *features* e cada valor representa o coeficiente de correlação entre as variáveis da sua linha e coluna, existem correlações entre várias *features* no *dataset*, representadas pelo valor 1, pelo que serão removidas. O seguinte código foi então criado para esse efeito:

```
# CORRELATION MATRIX

cor = table.corr()
feature_name = set()
for i in range(len(cor.columns)):
    for j in range(i):
        if abs(cor.iloc[i, j]) > 0.9:  # cuttof de 0.9
            colname = cor.columns[i]
            feature_name.add(colname)

print('Correlated features: ')
print(feature_name)

table.drop(labels=feature_name, axis=1, inplace=True)
```

Figura 28 - Matriz de correlação e remoção das *features* resultantes

No excerto de código apresentado acima, foi calculada a matriz de correlação para o *dataset* inteiro e a mesma foi percorrida, de forma a que, quando um valor na matriz fosse maior que 0.9 (valor de cut-off), o nome dessa *feature* era guardada num novo *array*, para de seguida ser removida do *dataset* original, através da função drop(). Na captura de ecrã abaixo (figura 29), é visível o nome de parte das variáveis que foram removidas.

```
Correlated features:
set(['L1_S24_F1079', 'L1_S24_F1172', 'L1_S25_F2985', 'L1_S24_F1170'
L1_S25_F2462', 'L1_S25_F2466', 'L1_S25_F2464', 'L3_S30_F3529', 'L1_
_S29_F3348', 'L1_S25_F2458', 'L3_S30_F3689', 'L3_S30_F3809', 'L3_S2
S25_F3020', 'L1_S25_F3022', 'L1_S25_F2751', 'L1_S24_F1212', 'L1_S24
0_F3644', 'L0_S12_F336', 'L3_S29_F3479', 'L3_S30_F3559', 'L3_S29_F3
```

Figura 29 – Algumas *features* correlacionadas que foram removidas

No final de todas as etapas de pré-processamento aplicadas, o *dataset* final contém 710 *features*, das 1000 inicialmente, ou seja, em 1000 *features*, 290 foram dadas como tendo pouca importância para os modelos.

5.2.3 Divisão dos dados em amostras de teste e treino

Outra etapa fundamental para treinar e avaliar um modelo é a divisão dos dados em conjuntos de dados de treino e de teste, usados respetivamente para treinar e avaliar o modelo. Por norma, esta divisão é feita com cerca de 30% dos dados para teste e os restantes para treino. No caso dos dados disponibilizados para este projeto, optou-se por dividir os dados obtidos dos ficheiros de treino, em dados de treino e de teste, ao invés de efetuar novamente uma leitura de ficheiros .csv, visto que os ficheiros de treino possuem dados suficientes para efetuar essa divisão, e por forma a poupar tempo e processamento.

Esta divisão pode ser efetuada através da função *train_test_split*() da biblioteca sklearn, como é apresentado abaixo na figura 30. A coluna "Response" foi usada por representar o resultado esperado. O atributo *random_state* representa um valor numérico, utilizado para baralhar os dados durante a divisão.

```
X_train, X_test, y_train, y_test = train_test_split(table.drop(["Response"], axis=1), table["Response"],
                                                    test_size=0.3,  # 30% for the evaluation set
                                                    random_state=10)
```

Figura 30 - Divisão dos dados em conjuntos de teste e treino

5.3 Treino dos algoritmos

Nesta secção, serão demonstradas e explicadas todas as decisões tomadas no treino dos algoritmos de *machine learning*, *XGBoost*, *Random Forest* e *Support Vector Machine*, previamente apresentados nas secções anteriores neste documento.

5.3.1 XGBoost

XGBoost é uma biblioteca que implementa algoritmos de *machine learning* sob a estrutura *Gradient Boosting*, otimizada e projetada para ser altamente eficiente e flexível (XGBoost, 2020). Visto que o algoritmo XGBoost não pode ser aplicado a dados do tipo *object* ou *string*, foi necessário modificar as variáveis de saída, para que estas sejam do tipo numérico. Pode-se facilmente converter o tipo dos dados para valores inteiros com o *LabelEncoder* da biblioteca sklearn, como é apresentado na imagem abaixo:

```
encoder = preprocessing.LabelEncoder()
y_train = encoder.fit_transform(y_train)
y_test = encoder.fit_transform(y_test)
```

Figura 31 - Conversão dos dados para o tipo numérico

Como estamos a lidar com vários *dataframes* Pandas, foi necessário utilizar o classificador XGBoost e a função fit() para aplicar os dados ao algoritmo, ao invés da função train(), que apenas aceita dados do tipo xgb.DMatrixes. O algoritmo necessita de receber ainda um array Map com os parâmetros de configuração. Estes foram definidos e modificados várias vezes, até se chegar ao melhor conjunto, tendo em conta os resultados das métricas de avaliação aplicadas, que serão apresentadas mais à frente neste documento, e são apresentados na figura abaixo:

```
params = {"max_depth" : 9, "eta":0.2, "subsample" : 0.8, "colsample_bytree" : 0.4,
          "objective": "binary:hinge", "booster":"gbtree"}
```

Figura 32 – Exemplo de configuração dos parâmetros para o algoritmo XGBoost

O primeiro parâmetro representa a profundidade máxima de uma árvore. Por defeito, o valor é 6 e aumentando este valor irá aumentar a complexidade do modelo. "*Eta*" representa o tamanho de cada passo, utilizado para prevenir um sobre ajuste e, por defeito, o valor é 0.3. *Subsample* é um ratio das instâncias de treino, em que por defeito o valor é 1. Alterar este valor para 0.8 significa que o algoritmo irá, de forma aleatória, escolher 80 por cento dos dados de treino antes de aumentar uma árvore, o que previne os sobre ajustes. O parâmetro *colsample_bytree* define, por sua vez, um ratio de uma subamostra das colunas ao construir uma árvore. O parâmetro *objective*, com o valor "*binary:hinge*" é utilizado para definir o valor das previsões para 0 ou 1, o que condiz com os dados da coluna "*response*" dos dados (XGBoost Parameters 2021).

Após várias tentativas com estes valores, chegou-se à conclusão de que o melhor resultado era obtido utilizando os valores por defeito para todos os campos, com exceção do campo "objective". Assim sendo, o array map dos parâmetros final foi o seguinte:

```
params = {"objective": "binary:hinge"}
```

Figura 33 - Array final com os parâmetros para o algoritmo XGBoost

Os parâmetros definidos e os dados de treino foram então aplicados ao algoritmo da seguinte forma:

```
model = xgboost.XGBClassifier()
model.set_params(**params)
model.fit(X_train, y_train)
```

Figura 34 - Treino do algoritmo XGBoost

De seguida, foram efetuadas previsões com os dados de teste, através da função predict(), da seguinte forma:

```
y_pred = model.predict(X_test)
```

Figura 35 - Aplicação dos dados de teste ao modelo

Com o modelo criado e treinado, foi ainda possível obter e visualizar num gráfico de barras, as *features* mais importantes para o mesmo, através da função plot_importance(), disponibilizada pela biblioteca XGBoost. O resultado é apresentado na figura 36 abaixo.

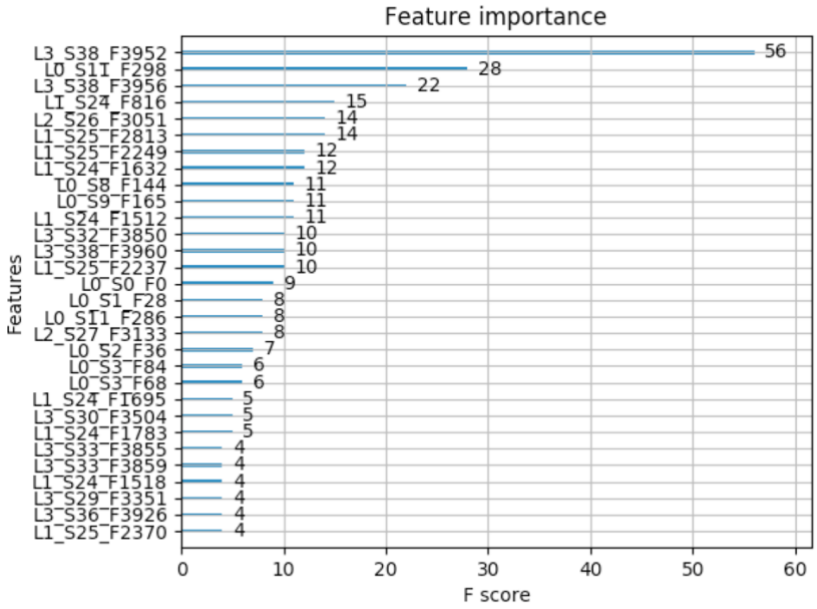


Figura 36 - Gráfico das Features mais importantes

É possível utilizar estas *features* para treinar novamente os modelos, o que irá aperfeiçoar os mesmos, ao se reduzir o ruído introduzido por *features* com pouca ou nenhuma importância para os modelos, melhorando assim as suas previsões.

Por fim, efetuaram-se as avaliações ao modelo através da utilização de várias métricas, que serão apresentadas em maior detalhe de seguida, junto com os seus resultados, na secção 6 deste documento.

5.3.2 Support Vector Machine

O algoritmo Support Vector Machine, tal como já foi previamente apresentado neste documento, é um algoritmo de aprendizagem supervisionada utilizado para efetuar a classificação e a análise de regressão. O SVM recebe um conjunto de dados como entrada, e prediz, para cada entrada, qual de duas possíveis classes a entrada faz parte, o que faz do SVM um classificador linear binário não probabilístico (Corinna et al. 1995).

Mais uma vez, a biblioteca sklearn disponibiliza várias funções que permitem a utilização deste algoritmo. A sua sintaxe é apresentada abaixo na figura 37.

```
svmModel = svm.SVC()
svmModel.fit(X_train, y_train)
y_SVM_pred = svmModel.predict(X_test)
```

Figura 37 -Aplicação do algoritmo SVM

Em primeiro lugar é necessário importar o modelo SVM e de seguida aceder ao classificador (função SVC). Neste algoritmo é possível também se definir vários parâmetros, porém, para simplificar a solução optou-se pela utilização dos valores por defeito. De seguida, treina-se o modelo SVM com os dados de treino, através da função fit() e efetuam-se as classificações através da função predict().

Este algoritmo foi também avaliado utilizando várias métricas como precisão, F1, recall, etc. que serão apresentadas no capítulo seguinte.

5.3.3 Random Forest

O algoritmo *Random Forest,* resumindo, é um algoritmo que, como o nome indica, consiste num grande número de árvores individuais de decisão, que operam em conjunto. Cada árvore da "floresta" efetua uma previsão e a classe com o maior número de votos torna-se a previsão final do modelo.

Para a aplicação deste algoritmo no contexto deste problema, utilizou-se novamente a biblioteca sklearn para aceder ao classificador *Random Forest*. Tal como nos restantes classificadores, é possível introduzir parâmetros que modificam o comportamento do algoritmo. Neste caso, modificaram-se os parâmetros "*n_estimators*" e "*random_state*", a que foram atribuídos os valores 1000 e 42, respetivamente. O primeiro parâmetro representa o número de árvores da floresta e, quando maior este valor, maior será a complexidade do modelo. Por defeito, este valor é 100. O segundo parâmetro introduzido trata-se de uma *seed* que controla do estado aleatório e do *bootstrapping* do modelo. Valores populares para este parâmetro encontram-se entre o valor 0 e 42 (Random Forest Classifier 2021). As funções encontram-se visíveis abaixo na figura 38.

```
clf = RandomForestClassifier(n_estimators=1000, random_state=42)
clf.fit(X_train, y_train)
y_pred_rf = clf.predict(X_test)
```

Figura 38 - Aplicação do algoritmo SVM

Este algoritmo foi, juntamente com os algoritmos apresentados anteriormente, avaliado segundo várias métricas como precisão, F1, recall, etc. que serão apresentadas no capítulo seguinte.

6 Avaliação

Neste capítulo serão descritas as experiências e avaliações realizadas à solução preconizada e as grandezas que se utilizaram para a avaliação dos modelos criados, assim como uma comparação entre os resultados dos vários algoritmos aplicados e, por fim, será realizada uma breve comparação com outros estudos semelhantes realizados e mencionados anteriormente na secção 2.3 deste documento.

6.1 Avaliação dos modelos

Como já foi referido em mais detalhe na secção 2.2.4 deste documento e, visto se tratar de um projeto que utiliza algoritmos de *machine learning,* com o objetivo de prever que peças irão falhar ao longo de uma linha de produção, as métricas para avaliar os resultados serão métricas de avaliação dos algoritmos utilizados.

Como forma de avaliação dos modelos, foram utilizadas várias métricas de avaliação de algoritmos de aprendizagem supervisionada de classificação, como é o caso da Exatidão, da Precisão, do Recall, do F1 Score e do MCC.

As métricas Precisão, Recall e F1 Score, foram calculadas utilizando médias dos tipos (Maria Khalusova 2021):

- Micro: Todas as amostras irão contribuir de igual forma para o resultado final
- Macro: Todas as classes irão contribuir de igual forma para o resultado final
- Com peso: Cada classe contribui para a média consoante o seu tamanho

O código criado para avaliar o modelo XGBoost é apresentado de seguida. Para os restantes modelos, o código foi semelhante, tendo sido alteradas apenas as variáveis utilizadas.

```
predictions = [round(value) for value in y_pred]

auc_score = roc_auc_score(y_test, y_pred)
print('AUC XGBoost: %.3f%%' % auc_score)

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy XGBoost: %.3f%%" % (accuracy * 100.0))

precision_macro = precision_score(y_test, y_pred, average='macro')
print("Precision macro XGBoost: %.3f%%" % (precision_macro * 100.0))

precision_micro = precision_score(y_test, y_pred, average='micro')
print("Precision micro XGBoost: %.3f%%" % (precision_micro * 100.0))

precision_weighted = precision_score(y_test, y_pred, average='weighted')
print("Precision weighted XGBoost: %.3f%%" % (precision_weighted * 100.0))

print("Matthews Correlation Coeffinciant XGBoost:",
      matthews_corrcoef(y_test, y_pred))

# F1 score
f1_score_macro = f1_score(y_test, y_pred, average='macro')
print("F1 Score macro XGBoost: %f" % f1_score_macro)
f1_score_micro = f1_score(y_test, y_pred, average='micro')
print("F1 Score micro XGBoost: %f" % f1_score_micro)
f1_score_weighted = f1_score(y_test, y_pred, average='weighted')
print("F1 Score weighted XGBoost: %f" % f1_score_weighted)

# Recall score
recall_score_macro = recall_score(y_test, y_pred, average='macro')
print("Recall Score macro XGBoost: %f" % recall_score_macro)
recall_score_micro = recall_score(y_test, y_pred, average='micro')
print("Recall Score micro XGBoost: %f" % recall_score_micro)
recall_score_weighted = recall_score(y_test, y_pred, average='weighted')
print("Recall Score weighted XGBoost: %f" % recall_score_weighted)
```

Figura 39 - Código de avaliação do modelo XGBoost

Para cada uma das métricas de avaliação, foram utilizadas as funções disponibilizadas pela biblioteca sklearn, para esse efeito, apresentadas abaixo na figura 40.

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_auc_score
```

Figura 40 - Métricas importadas da biblioteca sklearn

Para cada um dos modelos, obteve-se os seguintes resultados, tendo os algoritmos sido treinados com um *dataset* de 10000 linhas (chunksize inicial) e 710 *features* selecionadas:

Tabela 9 - Comparação de resultados

	XGBoost	SVM	Random Forest
Exatidão	0.9962	0.9927	0.9940
Precisão macro	0.6647	0.4978	0.5694
Precisão micro	0.9959	0.9957	0.9940
Precisão com peso	0.9931	0.9913	0.9923
MCC	0.1584	0.0326	0.1019
F1 Score macro	0.5613	0.4981	0.5484
F1 Score micro	0.9958	0.9956	0.9939
F1 Score com peso	0.9938	0.9935	0.9931
Recall macro	0.5381	0.5000	0.5374
Recall micro	0.9957	0.9956	0.9939
Recall com peso	0.9956	0.9955	0.9933
Tempo de processamento	20s	12s	17s

Como é possível verificar na tabela acima, os 3 algoritmos apresentaram resultados semelhantes. No entanto, o algoritmo XGBoost foi sempre ligeiramente superior aos restantes, ainda que por uma pequena margem, nas várias métricas aplicadas, tendo, por outro lado, sido o que demorou mais tempo a executar.

O algoritmo a escolher irá depender do que é pretendido por parte dos utilizadores. Caso os resultados se mantenham relativamente semelhantes ao se efetuar o processamento com a totalidade dos dados, o algoritmo SVM será o mais aconselhado caso o que se pretenda seja a melhor performance possível, visto ter sido consideravelmente mais rápido a executar que os restantes algoritmos. Caso se pertenta o melhor resultado possível, o algoritmo XGBoost será o mais aconselhado.

6.2 Comparação com estudos semelhantes

Na secção 2.3 deste documento, foram apresentados vários estudos que tinham por base a aplicação de algoritmos de *machine learning* num grande conjunto de dados. De todos esses artigos, os estudos de JuneHyuck Lee (Lee et. al 2018), Ankita Mangal (Mangal 2016) e Darui Zhang (Zhang et. Al 2016) foram os de maior importância e os mais realçados, pelo facto de utilizarem algoritmos de aprendizagem supervisionada e de classificação em linhas de produção, e de no estudo de Ankita Mangal e Darui Zhang, ter sido utilizado o mesmo conjunto de dados disponibilizado pela Bosch.

No estudo de JuneHyuck Lee (Lee et. al 2018) foram utilizados os algoritmos Decision Tree, Random Forest, Artificial Neural Network e Support Vector Machine para a previsão da qualidade na fundição de metais. Os metais encontravam-se divididos em três classes, “Good”, “Cold Shut” e “Bubble”. O dataset foi dividido em conjunto de teste e treino e cada conjunto continha um certo número de entradas de cada uma das classes, apresentados abaixo na figura 41:

	Good	Cold Shut	Bubble
Training set	1047	978	970
Test set	482	386	416
Total	1529	1364	1386

Figura 41 – Descrição dos *datasets* utilizados no estudo de JuneHyuck Lee (Lee et. al 2018)

Após os algoritmos terem sito treinados, foram efetuadas as previsões. Os resultados são apresentados e comparados abaixo na figura 42:

		Predicted		
		Good	Cold shut	Bubble
Actual	Good	286	90	106
	Cold shut	28	294	64
	Bubble	25	26	365

(a) Decision tree model

		Predicted		
		Good	Cold shut	Bubble
Actual	Good	421	38	23
	Cold shut	12	363	11
	Bubble	7	4	405

(b) Random forest model

		Predicted		
		Good	Cold shut	Bubble
Actual	Good	442	21	19
	Cold shut	10	367	9
	Bubble	6	4	406

(c) Artificial neural network model

		Predicted		
		Good	Cold shut	Bubble
Actual	Good	416	41	25
	Cold shut	18	359	9
	Bubble	7	8	401

(d) Support vector machine model

Figura 42 - Comparação das previsões do estudo de JuneHyuck Lee (Lee et. al 2018)

De seguida, foram aplicadas quatro métricas de avaliação aos resultados de cada algoritmo, a precisão, exatidão, *Recall* e tempo de processamento, e os resultados são apresentados na figura 43.

	Class	Precision	Recall	Overall Accuracy	Average Model Creating Time
Decision tree model	Good	0.8437	0.5934	0.7360	12 s
	Cold Shut	0.7171	0.7617		
	Bubble	0.6822	0.8774		
Random forest model	Good	0.9568	0.8734	0.9260	23 s
	Cold Shut	0.8963	0.9404		
	Bubble	0.9226	0.9736		
Artificial neural network model	Good	0.9643	0.8963	0.9384	1 m 27 s
	Cold Shut	0.9129	0.9508		
	Bubble	0.9355	0.9760		
Support vector machine model	Good	0.9433	0.8631	0.9159	21 s
	Cold Shut	0.8799	0.9301		
	Bubble	0.9218	0.9639		

Figura 43 - Métricas aplicadas, e os seus valores, no estudo de JuneHyuck Lee (Lee et. al 2018)

Ao analisar a figura 43, é possível retirar que, para o estudo em questão, o algoritmo Decision Tree foi consideravelmente mais rápido, porém, este não obteve os melhores resultados. De seguida, o algoritmo com melhor performance foi o Support Vector Machine, seguido do Random Forest, o que também se verificou neste documento. Em termos de precisão dos resultados, ambos são semelhantes, sendo o algoritmo Random Forest ligeiramente mais preciso a prever cada uma das classes. O mesmo se verificou nos resultados apresentados neste documento, na secção 6.1.

No estudo de Ankita Mangal (Mangal 2016) foi utilizado o mesmo *dataset* da Bosch, com a totalidade dos dados, e foram comparados os algoritmos XGBoost, Random Forest, Extra Trees Classifier e Logistic Regression. Como forma de avaliar as previsões e classificações dos vários algoritmos, apenas foi utilizada a métrica 3-fold Cross Validation Training AUC. Os resultados obtidos neste estudo são apresentados abaixo na figura 44:

Classification Model Used	3-fold Cross Validation Training AUC
Logistic Regression	0.614 ± 0.004
Extra Trees Classifier	0.685 ± 0.003
Random Forest Classifier	0.709 ± 0.003
Extreme Gradient Boosting (XGB)	0.718 ± 0.001

Figura 44 - Resultados do estudo de Ankita Mangal (Mangal 2016)

Ao analisar a figura acima, é possível verificar que o algoritmo XGBoost obteve uma melhor avaliação do que o algoritmo Random Forest, com o valor 0.718 ao invés de 0.709, seguido do algoritmo Extra Trees Classifier e, por último, o algoritmo Logistic Regression. Nos resultados apresentados neste documento, na secção 6.1, o mesmo resultado se verificou entre os algoritmos XGBoost e Random Forest, porém, como foram utilizados menos dados, os resultados das métricas foram superiores e o tempo de processamento foi consideravelmente inferior, tendo demorado cerca de 20 segundos a executar o algoritmo XGBoost, ao invés de 30 minutos no estudo de Ankita Mangal.

No estudo de Darui Zhang (Zhang et. Al 2016) também foi utilizado o mesmo *dataset*, mas este seguiu uma abordagem diferente e foram utilizados a totalidade dos dados disponíveis. Em primeiro lugar, os dados foram divididos em vários clusters e, de seguida, a cada cluster foram aplicados vários algoritmos de classificação como Logistic Regression, Naive Bayes, Decision Trees, Gradient Boosting e Random Forest e por fim, foi escolhido o melhor algoritmo e foram aplicadas métricas de avaliação a cada cluster. Os clusters foram divididos da seguinte forma:

Cluster #	Centroid	Number of samples	Number of features
0	1.37, 2.78	230831	813
1	2.14, -3.28	177562	882
2	-2.19, 2.48	88781	692
3	-1.99, -3.93	65106	874
4	-11.7, 1.81	17756	737
5	-11.49, -4.55	11837	697

Figura 45 - Divisão dos clusters no estudo de Darui Zhang (Zhang et. Al 2016)

De seguida, foram comparados os vários algoritmos de aprendizagem supervisionada, através das métricas MCC e AUROC.

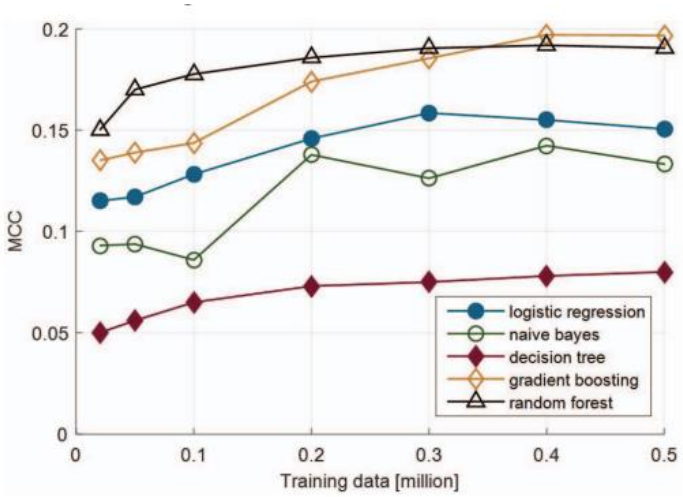


Figura 46 - Comparação dos algoritmos com a métrica MCC no estudo de Darui Zhang (Zhang et. Al 2016)

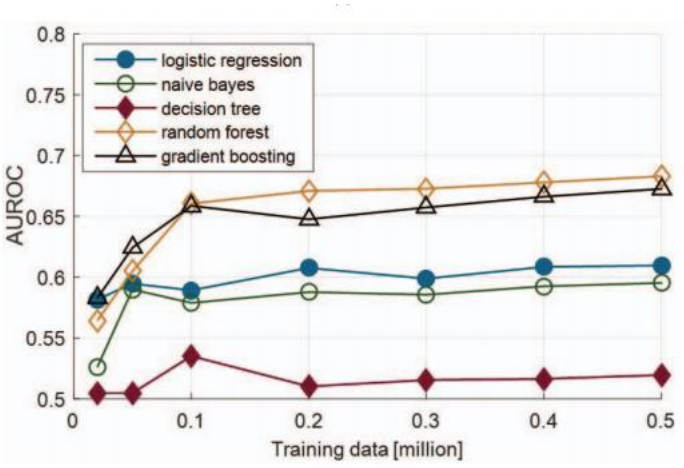


Figura 47 - Comparação dos algoritmos com a métrica AUROC no estudo de Darui Zhang (Zhang et. Al 2016)

O algoritmo Random Forest foi o escolhido neste estudo, pelo facto de que, apesar do algoritmo de Gradient Boosting por vezes apresentar melhores resultados, o tempo de treino do algoritmo Random Forest é inferior. Este algoritmo foi então utilizado nos vários clusters e foram aplicadas as mesmas métricas de avaliação a cada um dos clusters. Os resultados obtidos são apresentados abaixo na figura 48:

Cluster #	Classifier Parameters*		MCC	AUROC
	max depth	*min sample leaf*		
0	25	6	0.169	0.665
1	25	6	0.308	0.736
2	20	8	0.148	0.664
3	5	6	0.145	0.672
4	20	8	0.184	0.606
5	20	5	0.179	0.568
Total	--	--	0.211	0.692

Figura 48 - Avaliação dos vários clusters no estudo de Darui Zhang (Zhang et. Al 2016)

Ao analisar os resultados, é notável que os valores das métricas de avaliação são relativamente baixos, o que mostra a complexidade da previsão de falhas nas linhas de produção. Os valores obtidos demonstram que cluster 1 é o mais fácil de classificar e o cluster 3 o mais difícil. Foi também retirado deste estudo que a *performance* de cada cluster é muito variável.

O estudo de Darui Zhang é interessante em comparação com o trabalho apresentado neste documento, pelo facto de que a abordagem aplicada foi muito diferente. Ao comparar os resultados, a métrica MCC obteve valores mais altos na aplicação do algoritmo Random Forest em clusters, o que será interessante e poderá ser aplicado neste documento, como trabalho futuro.

7 Conclusão

Este documento investigou, comparou e avaliou vários algoritmos de *machine learning* de aprendizagem supervisionada de classificação, para a criação de modelos para a previsão de falhas nas linhas de produção. A complexidade deste projeto deve-se ao facto do tamanho dos dados disponibilizados e da performance limitada que esteve disponível, para efetuar o processamento de algoritmos e de funções complexas, o que obrigou à redução dos dados importados e das variáveis utilizadas.

Para efetuar as previsões, optou-se pelos algoritmos XGBoost, Support Vector Machine e Random Forest. Para avaliar os mesmos utilizaram-se as métricas de precisão, F1 score, recall, Mathews Correlation Coeficient, a exatidão e o tempo de processamento. Nas primeiras três foram ainda utilizadas três formas para calcular as médias, do tipo macro, micro e consoante o seu peso. Ao comparar o valor do resultado das várias métricas aplicadas às previsões dos vários algoritmos foi possível saber qual, ou quais, os melhores algoritmos para aplicar ao problema em questão, ou seja, qual o algoritmo que melhor prevê possíveis problemas numa linha de produção e que melhor promove o controlo de qualidade, dependendo do que é mais importante para o utilizador.

Ao aplicar este sistema numa linha de montagem, irá reduzir de forma significativa, o tempo e os custos de produção das peças, o que torna esta análise muito importante para um fabricante de grande dimensão, como a Bosch.

7.1 Trabalho futuro

Apesar de se ter alcançado bons resultados no estudo efetuado, há ainda vários pontos de melhoramento para trabalho futuro, tais como:

- Correr os modelos numa máquina com melhor performance (instância AWS, etc.);
- Importar a totalidade dos dados fornecidos;
- Explorar e adicionar mais *features*, por forma a melhorar os modelos;
- Utilizar outras técnicas de pré-processamento e limpeza dos dados;
- Explorar e usar outros algoritmos de *machine learning*;

Referências

(Rosa et al. 2018)	Conceição Rosa, F. J. G. Silva, Luís Pinto Ferreira, Teresa Pereira, Ronny Gouveia, School of Engineering, Polytechnic of Porto, ISEP & Research Center of Mechanical Engineering, CIDEM, 2018, Porto, Portugal, viewed December 1 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978918312137
(Zaslavsky et al. 2012)	Zaslavsky, Arkady & Perera, Charith & Georgakopoulos, Dimitrios, 2012. Sensing as a Service and Big Data. CoRR. [Online]. Available: https://www.researchgate.net/publication/234017925_Sensing_as_a_Service_and_Big_Data
(Kaggle 2016)	"Bosch Production Line Performance", Bosch, 2016, viewed December 1 2020. [Online]. Available: https://www.kaggle.com/c/bosch-production-line-performance/overview
(NHTSA 2020)	National Highway Traffic Safety Administration. "Takata Recall Spotlight" nhtsa.gov, viewed December. 8, 2020. [Online]. Available: https://www.nhtsa.gov/equipment/takata-recall-spotlight
(CNN 2017)	Mullen, "Takata, brought down by airbag crisis, files for bankruptcy", money.cnn.com, viewed December 8, 2020. [Online]. Available: https://money.cnn.com/2017/06/25/news/companies/takata-bankruptcy/index.html
(Bosch 2020)	Bosch, "Humans, machines, and processes in perfect collaboration", bosch.com, viewed December 8, 2020. [Online]. Available: https://www.bosch.com/stories/industry-4-0-production-line/
(Koen 2020)	Peter Koen, "What is the new concept development NCD model", viewed December 12, 2020. [Online]. Available: http://frontendinnovation.com/fei/what-is-the-new-concept-development-ncd-model
(Koen 2001)	Koen. "Providing clarity and a common language to the 'fuzzy front end'". Research Technology Management, 2001. p44-45, Viewed December 12, 2020
(Nicola 2020)	Susana Nicola, "Análise de valor", Viewed December 13, 2020. [Online]. Available: https://moodle.isep.ipp.pt/pluginfile.php/85346/mod_resource/content/1/Aula%201_18_Nov_2020.pdf
(Mohammed e Rami 2015)	Mohammed B, Rami A. "Application of the Analytical Hierarchy Process (AHP) to multi-criteria analysis for contractor selection." American Journal of Industrial and Business Management. 2015; 581-589.

(Saaty 2008) Saaty. “Decision making with the analytical hierarchy process”. International Journal of Services Sciences. 2008; 83-98.

(Quah et al. 2008) Quah, J. T. S., & Sriganesh, M. (2008). Real-time credit card fraud detection using computational intelligence. *Expert Systems with Applications*, *35*(4), 1721–1732, viewed December 18, 2020. [Online]. Available: https://doi.org/10.1016/j.eswa.2007.08.093

(Surthi 2019) Surthi, T. (2019). Credit Card Fraud Detection Using Supervised Learning Techniques. *Science, Technology and Development, VIII*(XI), 203-210.

(Cortes et al. 1995) Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks". Machine Learning.

(Misra e Li, 2020) Siddharth Misra, Hao Li, 2020. “Machine Learning for Subsurface characterization”. 243-287

(Mao e Wang, 2012). Wenji Mao, Fei-Yue Wang, 2012. “New Advances in Intelligence and Security Informatics” 91-102

(Harrison 2018) Onel Harrison, 2018, “Machine Learning Basics with the K-Nearest Neighbors Algorithm”, viewed December 19, 2020. [Online]. Available: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

(Gandhi 2018) Rohith Gandhi, 2018, “Naïve Bayes Classifier”, viewed December 19, 2020. [Online]. Available: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

(Pumsirrirat 2018) Apapan Pumsirrirat, Liu Yan, 2018, “Credit Card Fraud Detection using Deep Learning based on Auto-Encoder and Restricted Boltzmann Machine”, School of Software Engineering, Tongji University, viewed December 19, 2020. [Online]. Available: https://thesai.org/Publications/ViewPaper?Volume=9&Issue=1&Code=IJACSA&SerialNo=3

(Rahul Agarwal 2020) Rahul Agarwal, 2019, “The 5 classification metrics every Data Scientist must know“, viewed December 19, 2020. [Online]. Available: https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226

(Sun et al. 2019) C. Sun, M. Ma, Z. Zhao, S. Tian, R. Yan and X. Chen, "Deep Transfer Learning Based on Sparse Autoencoder for Remaining Useful Life Prediction of Tool in

Manufacturing," in IEEE Transactions on Industrial Informatics, vol. 15, no. 4, pp. 2416-2425, April 2019.

(Lee et al. 2018) JuneHyuck Lee, Sang Do Noh, Hyun-Jung Kim, Yong-Shin Kang, “Implementation of Cyber-physical Production Systems for Quality Prediction and Operation Control in Metal Casting”, Department of Systems Management Engineering, Sungkyunkwan University, Korea, May 4 2018

(Peres et al. 2019) R. S. Peres, J. Barata, P. Leitao and G. Garcia, "Multistage Quality Control Using Machine Learning in the Automotive Industry," in IEEE Access, vol. 7, pp. 79908-79916, 2019

(Lee et al. 2015) C.-H. Lee, H.-C. Yang, S.-C. Cheng, and S.-W. Tsai. “A hybrid big data analytics method for yield improvement in semiconductor manufacturing.” In Proceedings of the ASE BigData & Social Informatics 2015, pages 9:1–9:4, New York, NY, USA, 2015.

(Gardner e Bieker 2000) M. Gardner and J. Bieker. “Data mining solves tough semiconductor manufacturing problems”. In Proc. 6th ACM SIGKDD Conference, Simoff, pages 376–383, 2000.

(Stack Overflow, 2020) Stack Overflow. Accessed on: February 12 2021. [Online]. Available: https://insights.stackoverflow.com/survey/2020#technology

(XGBoost, 2020) XGBoost. Accessed on: February 13 2021. [Online]. Available: https://xgboost.ai/about

(Sivakumar e Gunasundari 2017) A. Sivakumar, R. Gunasundari, “A Survey on Data Preprocessing Techniques for Bioinformatics and Web Usage Mining”, Department of Computer Science, Karpagam University, Coimbatore [Online]. Available: https://acadpubl.eu/jsi/2017-117-20-22/articles/20/68.pdf

(Mangal e Kumar 2016) Ankita Mangal, Nishant Kumar, “Using Big Data to Enhance the Bosch Production Line Performance: A Kaggle Challenge”, 2016. [Online]. Available: https://arxiv.org/pdf/1701.00705.pdf

(Zhang et al. 2016) Darui Zhang, Bin Xu, Jasmine Wood, “Predict Failures in Production Lines”, 2016. [Online]. Available: https://www.researchgate.net/profile/Bin-Xu-14/publication/313456174_Predict_failures_in_production_lines_A_two-stage_approach_with_clustering_and_supervised_learning/links/5a731989aca2720bc0dac590/Predict-failures-in-production-lines-A-two-stage-approach-with-clustering-and-supervised-learning.pdf

(Boughorbel et al. 2017) Sabri Boughorbel, Fethi Jarray, Mohammed El-Anbari, “Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric”, 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5456046/

(Keras 2016) Keras, “Building Autoencoders in Keras” 2016, Accessed: February 2021 [Online]. Available: https://blog.keras.io/building-autoencoders-in-keras.html

(Scikit-learn 2020) Scikit-learn, “Decision Trees”, 2020, Accessed: February 2021 [Online]. Available: https://scikit-learn.org/stable/modules/tree.html

(Visual Studio Code 2021) Visual Studio Code, “Code editing. Redefined.”, 2021, Accessed: May 2021 [Online]. Available: https://code.visualstudio.com/

(Homebrew 2021) Homebrew, 2021. Accessed: May 2021 [Online]. Available: https://brew.sh/

(Ilia Zaitsev 2019) Towards Data Science, “The best format to save Pandas data”, 2019. Accessed: May, 2021 [Online]. Available: https://towardsdatascience.com/the-best-format-to-save-pandas-data-414dca023e0d

(Pandas 2021) Pandas, “pandas.to_numeric”, 2021. Accessed: May 2021 [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.to_numeric.html

(XGBoost Parameters 2021) XGBoost, “XGBoost Parameters”, 2021. Accessed: May 2021 [Online]. Available: https://xgboost.readthedocs.io/en/latest/parameter.html

(Random Forest Classifier 2021) Sklearn, “Random Forest Classifier”, 2021. Accessed: May 2021 [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

(Maria Khalusova 2021) Maria Khalusova, “Machine learning model evaluation metrics part 2: multi-class classification”, 2021. Accessed: May 2021 [Online]. Available: https://www.mariakhalusova.com/posts/2019-04-17-ml-model-evaluation-metrics-p2/

(Mokhtar Ebrahum 2020) Mokhtar Ebrahum, “Python Correlation matrix tutorial”, 2020. Accessed: May 2021 [Online]. Available: https://likegeeks.com/python-correlation-matrix/

(Nathaniel Jermain 2019) Nathaniel Jermain, “Transforming Skewed Data for Machine Learning”, 2019. Accessed: May 2021 [Online]. Available: https://opendatascience.com/transforming-skewed-data-for-machine-learning/

(CFI 2021) CFI, “Skewness”, 2021. Accessed: May 2021 [Online]. Available: https://corporatefinanceinstitute.com/resources/knowledge/other/skewness/

(Deepsense.ai 2019) Deepsense.ai, “What is reinforced learning”, 2019. Accessed: May 2021 [Online]. Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

(Jason Brownlee 2016) Jason Brownlee, “A gentle introduction to XGBoost for applied machine learning”, 2016. Accessed: May 2021 [Online]. Available: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

(Porter 1985) Porter, Michael E. “Competitive advantage, creating and sustaining superior performance”, 1985.