

Aprendizagem de Máquina I

Projeto Final

Felipe Azzolino Varella - fav3@cin.ufpe.br
Matheus Augusto Ladeira Cayres - malc@cin.ufpe.br



11 de agosto de 2024

1 Introdução

O presente documento é o relatório final de execução do projeto entregue na disciplina Aprendizado de Máquina I da terceira edição do Programa de Especialização em Software (PES 2024) da Embraer, lecionada pelo professor George Darmiton da Cunha Cavalcanti.

A base de dados escolhida para realização do projeto fornece informações sobre o conjunto de dados *"Steel Plates Faults"* do repositório de aprendizado de máquina da UCI. O conjunto de dados contém 1941 instâncias de falhas em placas de aço, classificadas em 7 tipos diferentes. Ele possui 27 características independentes que são usadas para tarefas de classificação. As variáveis incluem medidas como mínimos e máximos em X e Y, áreas de pixels, perímetros e luminosidade. O objetivo do presente trabalho é avaliar a performance de algoritmos de aprendizado de máquina de classificação.

2 Algoritmos de Aprendizagem de Máquina

Primeiramente, precisamos caracterizar a natureza do problema de aprendizagem de máquina abordado no projeto. Dado que desejamos inferir o valor de uma variável específica a partir do treinamento de uma base de dados disponível, temos que nosso problema é de natureza supervisionada. Neste caso específico, a variável estudada define, dentre 7 opções diferentes, o tipo de problema presente em placas de aço. Com isso, trata-se de um problema de classificação.

Para lidar com problemas dessa característica há uma gama de algoritmos de aprendizado de máquina que podem ser utilizados. A seguir temos uma breve definição sobre os que foram abordados na disciplina e suas respectivas vantagens, desvantagens e hiperparâmetros.

2.1 K-Nearest Neighbors (KNN)

Mitchell (1997) descreve o K-Nearest Neighbors (KNN) como um algoritmo de aprendizado baseado em instâncias que classifica um exemplo baseado na maioria dos votos dos seus vizinhos mais próximos no espaço de características. Ele destaca a simplicidade e eficácia do KNN, especialmente em problemas onde a fronteira de decisão é complexa. No entanto, Mitchell também discute as limitações do KNN, como a necessidade de um número significativo de exemplos e a sensibilidade ao ruído e à escala dos dados, além de não lidar bem com variáveis explicativas categóricas. A quantidade de vizinhos considerados na classificação (k), o método de cálculo da distância e a definição se a distância dos vizinhos será ponderada na classificação são parâmetros importantes que impactam no desempenho das métricas do modelo.

2.2 Árvore de Decisão (DT)

Com relação às Árvores de Decisão (DT - Decision Tree), Mitchell (1997) a descreve como uma técnica que utiliza uma estrutura em forma de árvore para tomar decisões baseadas em características de entrada. As Árvores de Decisão são construídas recursivamente, escolhendo o atributo que melhor divide os dados em cada nó, com o objetivo de maximizar o ganho da informação ou reduzir a impureza. Mitchell também aborda a profundidade máxima da árvore, o número mínimo de amostras por folha e o número mínimo para divisão dos nós. Esses parâmetros ajudam a controlar a complexidade e a generalização do modelo. As Árvores de Decisão são geralmente fáceis de interpretar, possuem suporte nativo para problemas multiclasse e lidam bem com variáveis explicativas categóricas. No entanto, são sensíveis a dados desbalanceados e tendem ao sobreajuste (*overfitting*) se os hiperparâmetros forem escolhidos inadequadamente (*e.g.* alta profundidade).

2.3 Random Forest (RF)

Breiman (2001) traz que o algoritmo Random Forest (RF) cria uma coleção de Árvores de Decisão (uma "floresta") para melhorar a precisão e robustez da classificação. Cada árvore é construída a partir de um subconjunto aleatório dos dados de treinamento e de um subconjunto aleatório das variáveis. Para selecionar os dados, utiliza-se uma técnica conhecida como *bootstrap*: para cada árvore da floresta, realiza-se uma amostragem aleatória no conjunto de dados de treinamento com reposição, de forma que algumas amostras possam ser selecionadas várias vezes, enquanto que outras possam não ser selecionadas, fazendo com que cada árvore da floresta seja potencialmente diferente das demais. Para cada conjunto de dados obtidos no processo de *bootstrap*, seleciona-se aleatoriamente N atributos do conjunto de dados original para o treinamento de uma árvore. Em muitos casos, N é definido como a raiz quadrada da quantidade total de atributos do problema em questão. Uma vez treinadas as árvores, as previsões finais são obtidas pela média ou pela votação majoritária das árvores individuais.

Além dos hiperparâmetros que definem o tamanho da base gerada no processo de *bootstrap* e o número de atributos selecionados para cada árvore, há outros como o número de árvores na floresta. Esses parâmetros controlam a diversidade das árvores e podem auxiliar na redução do sobreajuste. Para este projeto, optou-se por analisar os mesmos hiperparâmetros citados para a Árvore de Decisão (*i.e.*, profundidade máxima da árvore, o número mínimo de amostras por folha e o número mínimo para divisão dos nós) juntamente com a quantidade de estimadores.

Quando comparado com as Árvores de Decisão, o Random Forest mitiga problemas de

overfitting devido a maior diversidade entre as árvores, tendo, em geral, maior capacidade de generalização. Todavia, continuam sendo sensíveis a dados desbalanceados, além de exigirem um tempo de treinamento relativamente maior que uma única árvore, uma vez que esta máquina é uma composição de outras árvores.

2.4 Regressão Logística (LR)

A Regressão Logística (LR - Logistic Regression), conforme descrito por Peng e Ingersoll (2002), é um método estatístico usado para prever a probabilidade de um evento binário ocorrer com base em uma ou mais variáveis independentes. Ela utiliza uma função logit para transformar uma combinação linear dessas variáveis em uma probabilidade, facilitando a classificação em uma das duas categorias possíveis. Porém, para nossa aplicação se faz necessário um método de classificação que envolva mais que 2 classes, sendo necessário uma alternativa para sua utilização.

Para que seja possível aplicar a máquina de Regressão Logística para mais de 2 classes (Regressão Logística Multiclasse), faz-se necessário utilizar um de dois possíveis métodos: ”One-vs-Rest”(OvR) e ”One-vs-One”(OvO). O método OvR consiste em treinar vários modelos binários para cada classe do problema, tratando-a como a classe positiva e agrupando todas as demais outras classes como a classe negativa. De maneira semelhante, o método OvO também envolve a criação de múltiplos modelos binários. No entanto, ao invés de um modelo para cada classe contra todas as outras, são treinados modelos para cada combinação possível de duas classes específicas dentre todas as classes disponíveis. Ambos os métodos permitem a aplicação da Regressão Logística em cenários com mais de duas classes, sendo o OvR particularmente útil para os problemas multiclasses em que o desbalanço não é crítico, visto que ao agrupar os dados o desbalanço torna-se mais significativo, além de exigir um número menor de treinamentos (igual a quantidade de classes). Por outro lado, o método OvO é menos sensível ao desequilíbrio das classes, mas requer a criação de um grande número de modelos para cobrir todas as possíveis combinações de pares de classes, totalizando $\binom{N}{2}$ combinações (em que N é igual ao número de classes).

No processo de treinamento de uma máquina baseada em Regressão Logística, busca-se determinar quais os valores $\beta = [\beta_0 \ \beta_1 \ \dots \ \beta_N]$ utilizados na equação $f(\mathbf{x}) = \frac{e^{\beta\mathbf{x}^T}}{1+e^{\beta\mathbf{x}^T}}$ reduzem o erro associado a classificação de \mathbf{x} . Para tal, pode-se utilizar diferentes algoritmos de otimização (como o LBFGS, Newton-CG, SAG e SAGA). Além disso, pode-se variar o tipo de regularização adotada que, de forma simplificada, ajuda a prevenir o *overfitting* ao adicionar uma penalização ao tamanho dos coeficientes do modelo, controlando a complexidade da máquina. Junto a isso, pode-se definir qual a força da regularização aplicada, de

forma que valores mais baixos resultem em modelos mais simples e valores maiores resultem em modelos com menores margens, levando a maior complexidade e potencial *overfit*.

2.5 Naive Bayes (NB)

O Classificador Naive Bayes (NB) utiliza, segundo Mitchell (1997), a probabilidade condicional para prever a classe de um dado exemplo, assumindo que todas as características são independentes umas das outras, dadas a classe. Para a sua aplicação, o algoritmo exige que as variáveis sejam paramétricas ou categóricas.

Os principais parâmetros desse classificador são as probabilidades *a priori* das classes e as probabilidades condicionais dos atributos dado cada classe. Essas probabilidades são estimadas a partir dos dados de treinamento e usadas para calcular a probabilidade *posteriori* de cada classe para novas amostras, permitindo a classificação baseada na máxima probabilidade *posteriori*. O algoritmo é marcado por uma fácil interpretabilidade dos resultados enquanto que traz como barreira sua utilização em bases de dados cujas variáveis preditoras contínuas não sejam parametrizáveis.

2.6 Support Vector Machines (SVM)

Burges (1998) explica que o Support Vector Machines (SVM) é um algoritmo que funciona encontrando um hiperplano ótimo que separa os dados em diferentes classes com a maior margem possível. O conceito de margem máxima é central para o SVM, onde o objetivo é maximizar a distância entre o hiperplano de separação e os pontos de dados mais próximos de qualquer classe, conhecidos como vetores de suporte. Para resolver problemas em que as classes não são linearmente separáveis, o SVM usa o truque do kernel, que transforma os dados em um espaço de maior dimensão onde um hiperplano linear pode ser ajustado. O método lida muito bem com problemas não lineares, contudo é de difícil interpretabilidade e pode ser computacionalmente custoso para grande volumes de dados, além de ser nativamente projetado para classificação binária, exigindo a utilização de técnicas como OvR ou OvO. Alguns dos principais hiperparâmetros do SVM incluem o tipo de kernel utilizado (por exemplo, linear, polinomial, RBF), a penalização por erros de classificação, e o alcance da influência dos dados de treinamento.

2.7 Perceptron Multicamadas (MLP)

Segundo Beale e Jackson (1990), o algoritmo Perceptron Multicamadas (MLP - Multilayer Perceptron) é definido como um modelo de rede neural de múltiplas camadas de neurônios,

onde cada nó de cada camada é totalmente conectado aos nós da camada seguinte. O aprendizado consiste na determinação dos pesos e vieses da rede que são atualizados por meio de um processo de otimização realizado na etapa de retropropagação (*backpropagation*), onde os erros entre a saída prevista e a saída desejada são propagados para trás através da rede, ajustando os pesos das conexões para minimizar esses erros. Este processo é iterativo e continua até que os erros sejam reduzidos a um nível aceitável ou até um número de execuções pré-estabelecido. O MLP é capaz de aprender e representar funções complexas e não lineares, contudo requer uma grande quantidade de dados de treinamento para generalizar bem, além do treino poder ser computacionalmente caro.

Os principais hiperparâmetros descritos são a taxa de aprendizado, que controla o tamanho dos ajustes de peso durante a retropropagação, o número de camadas ocultas, a quantidade de neurônios por camada, a função de ativação de cada neurônio e a regularização. Pelo Teorema da Aproximação Universal, pode-se demonstrar que é possível gerar superfícies de separação não lineares com uma única camada oculta em uma RNA. Com isto, o foco deste projeto será na exploração e validação de redes neurais com apenas uma camada oculta. A análise será centrada em variáveis como o número de neurônios na camada oculta, funções de ativação, algoritmos de otimização, taxa de aprendizado e força da regularização. Redes com múltiplas camadas não serão abordadas.

3 Experimentos

3.1 Banco de Dados

A base de dados utilizada foi retirada da plataforma *OpenML* e trata de observações relacionadas a falhas em placas de aço. A tabela contém 27 características de entrada e 7 variáveis-alvo, totalizando 34 colunas.

3.1.1 Variáveis Independentes

Segundo Ben (2024), as variáveis independentes tem a seguinte descrição:

- ***X_Minimum, X_Maximum, Y_Minimum, Y_Maximum***: Coordenadas que representam os valores mínimos e máximos da caixa delimitadora nas direções X e Y para um defeito ou região particular na placa de aço.
- ***Pixels_Areas***: O número total de pixels na região ou defeito identificado na placa de aço.

- **X_Perimeter, Y_Perimeter:** Comprimentos do perímetro nas direções X e Y para o defeito ou região identificada.
- **Sum_of_Luminosity, Minimum_of_Luminosity, Maximum_of_Luminosity, Luminosity_Index:** Medidas de luminosidade (brilho) do defeito ou região, incluindo a soma total, mínimo, máximo e um índice que indica as características gerais de luminosidade.
- **Length_of_Conveyer:** Comprimento da esteira utilizada durante o processo de fabricação.
- **TypeOfSteel_A300, TypeOfSteel_A400:** Variáveis categóricas que indicam o tipo de aço usado (A300 ou A400).
- **Steel_Plate_Thickness:** Espessura da placa de aço.
- **Edges_Index, Empty_Index, Square_Index, Outside_X_Index, Edges_X_Index, Edges_Y_Index, Outside_Global_Index:** Vários índices relacionados à forma e posicionamento do defeito ou região dentro da placa de aço.
- **LogOfAreas, Log_X_Index, Log_Y_Index:** Transformações logarítmicas de área e índices, possivelmente usadas para normalizar ou dimensionar certas características.
- **Orientation_Index:** Um índice que indica a orientação do defeito.
- **SigmoidOfAreas:** Função sigmoidal aplicada à área do defeito.

Essas *features* fornecem coletivamente informações sobre as características espaciais, geométricas e de luminosidade dos defeitos nas placas de aço, bem como informações sobre o processo de fabricação e o tipo de aço.

Para cada uma das variáveis independentes contínuas foi aplicado um teste de hipótese para verificar a aderência da distribuição das mesmas à normalidade. A 5% de significância há evidências estatísticas suficientes para rejeitar a hipótese nula de que elas seguem uma distribuição normal. Os p-valores dos testes executados foram sumarizados e disponibilizados em uma tabela anexada ao Apêndice.

3.1.2 Variável Dependente

As 7 variáveis alvo são encontradas no conjunto de dados, sendo elas binárias e mutuamente exclusivas, ou seja, apenas uma delas pode ter valor 1 para cada uma das observações.

- **Pastry:** Refere-se a pequenas manchas ou irregularidades na superfície da placa de aço, normalmente causadas por imperfeições no processo de fabricação ou manuseio durante o transporte. Essas imperfeições podem afetar a suavidade e a aparência da superfície da placa de aço.
- **Z_Scratch:** Arranhões estreitos ou marcas na superfície da placa de aço que correm paralelamente à direção de laminação. Vários fatores, como manuseio, usinagem ou contato com materiais abrasivos durante a produção ou transporte, podem causar esses arranhões.
- **K_Scratch:** Semelhante aos **Z_Scratch**, mas correm perpendicularmente à direção de laminação. Eles também podem ser causados por manuseio, usinagem ou contato com materiais abrasivos durante os processos de fabricação ou transporte.
- **Stains:** Áreas descoloridas ou contaminadas na superfície da placa de aço. Essas manchas podem resultar de várias fontes, como ferrugem, óleo, graxa ou outras substâncias que entram em contato com a superfície do aço durante o processamento, armazenamento ou manuseio.
- **Dirtiness:** Indica a presença de sujeira ou matéria particulada na superfície da placa de aço. Isso pode incluir vários tipos de detritos ou contaminantes que se acumulam durante os processos de fabricação, manuseio ou armazenamento.
- **Bumps:** Áreas elevadas ou salientes na superfície da placa de aço. Elas podem ser causadas por irregularidades no processo de fabricação, como laminação ou resfriamento desigual, ou por danos físicos durante o manuseio ou transporte.
- **Other_Faults:** Essa categoria provavelmente abrange uma gama mais ampla de falhas ou defeitos não explicitamente categorizados nos outros tipos de falhas listados. Pode incluir vários tipos de imperfeições de superfície, irregularidades ou anomalias que afetam a qualidade ou usabilidade da placa de aço.

Objetivando-se identificar apenas os dados cuja classificação do problema é conhecida e, portanto, não genérica, optou-se por remover aqueles registros cuja falha era definida como "**Other_Faults**", alterando as proporções das classes conforme segue na Figura 1.

Ademais, buscando-se ter um conhecimento prévio sobre a similaridade das classes, utilizou-se a distância euclidiana entre os centroides de cada um dos *targets*. Como os atributos são de diferentes escalas, primeiramente padronizou-se as variáveis (deixando-as com média 0 e desvio-padrão 1), para, então, calcular os centroides e medir a distância entre eles, resultando nos valores representados na Figura 2.

Proporção das classes na base de dados utilizada

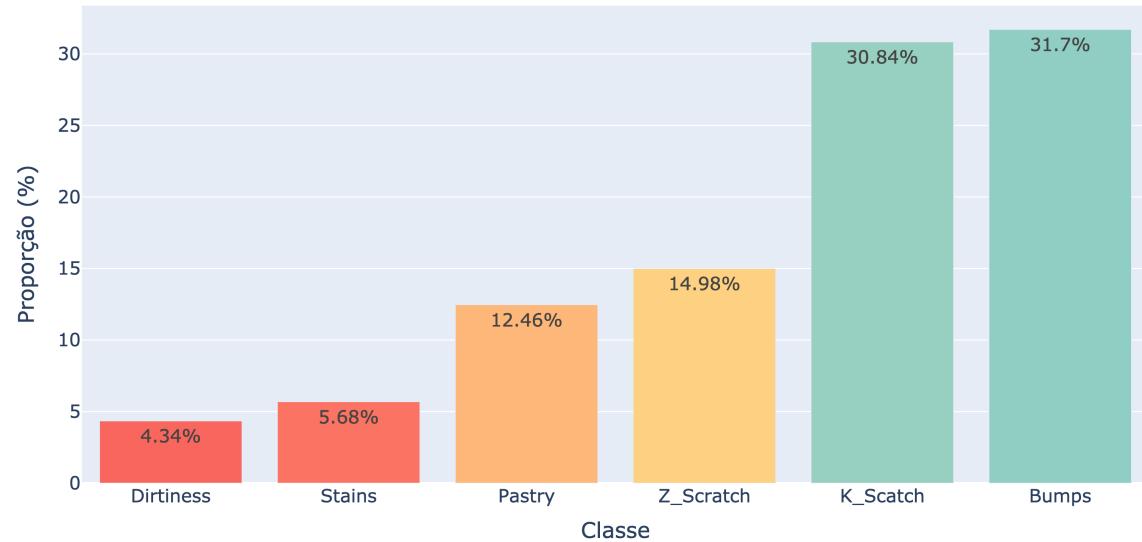


Figura 1: Proporção de classes na base de dados utilizada neste projeto.

Distâncias entre os centroides das classes

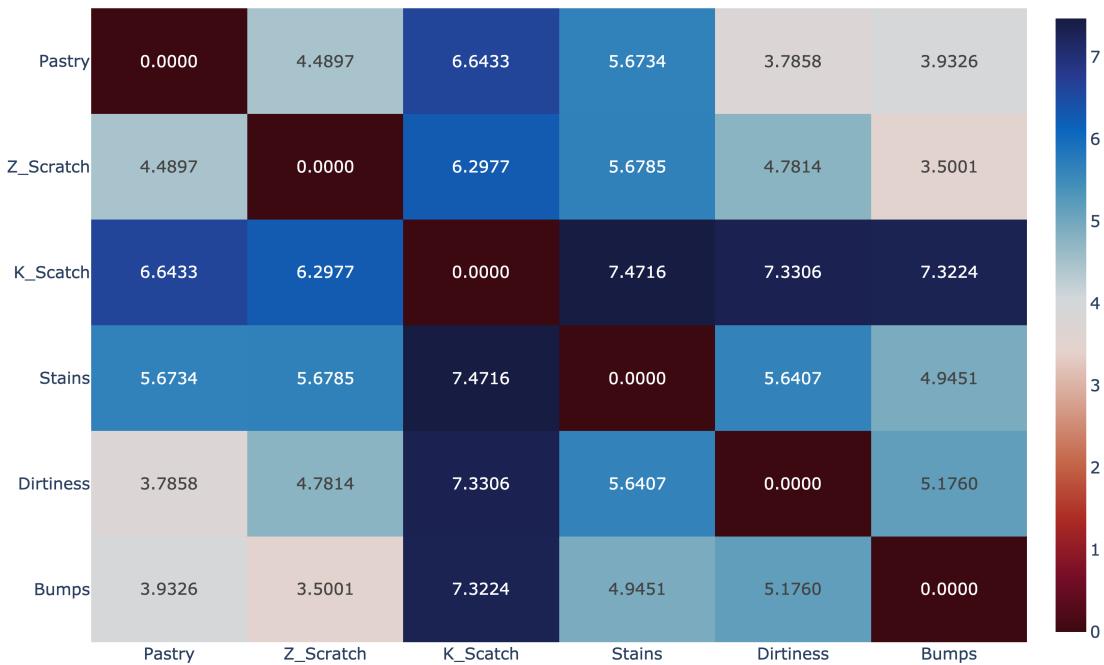


Figura 2: Distância entre os centroides dos dados normalizados.

Como a similaridade é inversa à distância, tem-se que quanto menor o valor, mais similar serão os centroides. Assim, pode-se ter verificar que as maiores similaridades dos centroides foram entre a classe ***Pastry*** e as classes ***Dirtiness*** e ***Bumps***, e entre ***Z_Scratch*** e ***Bumps***. Em contrapartida, a classe ***K_Scratch*** foi aquela que teve o centroide menos similar com as demais.

3.2 Métricas

Raschka (2020) conceitua a Matriz de Confusão como uma tabela que descreve o desempenho de um modelo de classificação, comparando as previsões feitas pelo modelo com os valores reais e oferecendo uma visão detalhada dos tipos específicos de erros que um modelo de classificação está cometendo. Isso permite uma avaliação mais precisa de onde o modelo pode ser melhorado. Ela é composta por quatro componentes principais:

- **Verdadeiros Positivos (TP)**: Casos onde o modelo previu corretamente a classe positiva.
- **Verdadeiros Negativos (TN)**: Casos onde o modelo previu corretamente a classe negativa.
- **Falsos Positivos (FP)**: Casos onde o modelo previu a classe positiva incorretamente.
- **Falsos Negativos (FN)**: Casos onde o modelo previu a classe negativa incorretamente.

Em problemas de classificação binária, a matriz de confusão é uma tabela 2×2 . No entanto, em problemas de multiclasse, essa matriz se expande para uma tabela $N \times N$, onde N é o número de classes. Cada célula (i, j) na matriz indica o número de exemplos cuja classe real é a classe i e que foram classificados como classe j pelo modelo. A diagonal principal (onde $i = j$) representa as previsões corretas, ou seja, os exemplos onde a classe prevista pelo modelo corresponde à classe real. As células fora da diagonal indicam erros de classificação, mostrando quais classes foram confundidas entre si.

A Acurácia é definida como uma medida simples e direta da performance do modelo, calculando a proporção de previsões corretas feitas pelo modelo em relação ao total de previsões, sendo mais útil em problemas de classificação balanceados, onde as classes têm aproximadamente o mesmo número de exemplos. Nesses casos, a Acurácia fornece uma boa indicação da performance do modelo.

Contudo, em conjuntos de dados desbalanceados, a Acurácia pode ser enganosa. Por exemplo, se uma classe é muito mais frequente que as outras, um modelo que sempre prevê

a classe mais frequente pode ter uma alta Acurácia, mas seu desempenho geral pode ser pobre. Outras métricas, como Precision, Recall e F1-score podem ser mais apropriadas neste contexto. Essas métricas fornecem uma visão mais detalhada do desempenho do modelo em identificar corretamente cada classe.

O Recall é particularmente útil em problemas onde a identificação de exemplos positivos é crucial, como na detecção de fraudes, diagnósticos médicos e outros casos em que não detectar um exemplo positivo pode ter consequências severas. Ele é calculado como o número de verdadeiros positivos dividido pela soma do número de verdadeiros positivos e o número de falsos negativos.

O artigo compara Recall com Precision, destacando que, enquanto Precision mede a proporção de exemplos positivos corretamente identificados em relação ao total de exemplos classificados como positivos, Recall foca na proporção de exemplos positivos corretamente identificados em relação ao total de exemplos positivos reais. O artigo também menciona que Recall é frequentemente usado em conjunto com Precision para calcular o F1-score, que é a média harmônica de Precision e Recall. O F1-score fornece um único valor que balanceia a importância de ambos, especialmente útil em contextos de classes desbalanceadas. O F1-score proporciona uma visão mais equilibrada ao considerar os *trade-offs* entre Precision e Recall. Isso é especialmente importante em aplicações onde tanto a identificação correta dos positivos quanto a minimização dos falsos negativos são críticos.

Outro recurso amplamente utilizado para avaliar a performance de máquinas de aprendizado de classificação é a curva ROC (Receiver Operating Characteristic), que é um gráfico que mostra a relação entre a taxa de falsos positivos (FP) - no eixo X - e a taxa de verdadeiros positivos (TP) - no eixo Y. A curva é traçada a partir da variação do limiar (*threshold*) de decisão do modelo (um valor a partir do qual uma variável alvo binária muda de estado). Cada ponto na curva representa um par (FP, TP) para um determinado limiar. Uma medida escalar de desempenho do classificador associada à curva é a AUC-ROC (Area Under the ROC Curve), que quantifica a área sob a curva, variando entre 0 e 1.

Tanto a curva quanto a área sob a mesma foram originalmente desenvolvidas para problemas de classificação binária. Contudo, em problemas multiclasse, não basta uma única curva ROC como no caso binário. É possível aplicar diferentes abordagens, porém à medida que o número de classes aumenta, a complexidade de cálculo e interpretação aumenta significativamente.

No projeto desenvolvido, devido ao fato de a variável alvo possuir 7 classes que não são completamente balanceadas, adotamos uma estratégia que combina as métricas de Acurácia e F1-score. Utilizamos a Acurácia como uma medida comparativa global da performance dos modelos treinados, enquanto que o F1-score foi empregado para avaliar o desempenho dos

modelos em cada uma das classes. Já em relação à curva ROC e à área sob a mesma, seu uso foi descartado, pois exigem que os modelos forneçam uma interpretação de probabilidade de classe, como é o caso, por exemplo, da Regressão Logística. No entanto, nem todos os modelos utilizados no estudo têm uma interpretação direta de probabilidade, o que inviabiliza a utilização dessa métrica, visto que é crucial que todos os modelos analisados possam ser validados com a mesma métrica para assegurar a comparabilidade entre eles.

Outras métricas avaliadas para as máquinas abordadas são o tempo médio de treinamento e o tempo médio de inferência em teste. Durante o processo de Validação Cruzada com N Folds, realiza-se uma busca pelos melhores hiperparâmetros por meio do Grid Search nos $N - 1$ Folds destinados ao treino e à validação. Com os hiperparâmetros definidos, as máquinas são treinadas com os dados de todos os $N - 1$ Folds em conjunto, e o tempo gasto nesse treinamento é registrado. Esse procedimento é repetido para cada uma das N iterações. Ao final, é possível calcular, para cada modelo, o tempo médio de treinamento correspondente. Da mesma forma, ao final de cada um dos N treinamentos, é realizada a predição do conjunto de teste, em que o tempo decorrido é registrado. Sendo assim é possível calcular, também para cada um dos modelos, o tempo médio de inferência. O tempo de treinamento ajuda a entender o custo computacional necessário para ajustar o modelo, enquanto o tempo de inferência indica a rapidez com que o modelo pode fazer previsões em novos dados, sendo crucial para aplicações em tempo real ou com grandes volumes de dados. O peso de cada métrica varia conforme o problema: o tempo médio de treinamento é crucial para modelos que precisam ser retrainados frequentemente, enquanto o tempo médio de inferência é vital em problemas onde o tempo de resposta é crítico.

3.3 Procedimento

Para avaliar a performance dos modelos perante os dados, duas estratégias foram utilizadas em conjunto. A primeira delas, a Validação Cruzada, é, segundo Berrar (2024), um método de avaliação de modelos que envolve a divisão dos dados disponíveis em múltiplos subconjuntos ou Folds. O modelo é treinado em alguns desses subconjuntos e testado nos outros, permitindo uma estimativa mais robusta da performance do modelo, comparada à simples divisão de dados em treino e teste.

Já a segunda estratégia é o Grid Search, que é definido por Liashchynskyi e Liashchynskyi (2019) como um método exaustivo para otimização de hiperparâmetros, que avalia todas as combinações possíveis destes valores. O objetivo é encontrar a combinação que oferece a melhor performance do modelo dentre aquelas testadas, medida por métricas como Acurácia ou F1-score. Embora eficaz, o Grid Search pode ser computacionalmente intensivo devido

ao número elevado de combinações testadas.

No presente projeto, adotou-se um procedimento de Validação Cruzada com N Folds. O conjunto de dados foi dividido em N partes, permitindo que os modelos fossem testados em N diferentes conjuntos de teste. Em cada iteração, $N - 1$ Folds foram utilizados como conjunto de treino e validação, enquanto o Fold restante foi usado como conjunto de teste.

Os $N - 1$ Folds foram então divididos em M outros Folds, onde se aplicou o algoritmo de Grid Search para determinar os melhores hiperparâmetros. Após a definição dos melhores hiperparâmetros, o modelo foi treinado novamente nos $N - 1$ Folds completos para, finalmente, ser testado no conjunto de teste. Este procedimento foi repetido até que todos os N Folds fossem utilizados como conjunto de teste uma vez, obtendo-se em cada iteração as métricas enunciadas anteriormente. Assim, para uma iteração $K \in [1, N]$, o procedimento descrito acima pode ser sumarizado no esquemático referenciado na Figura 3.

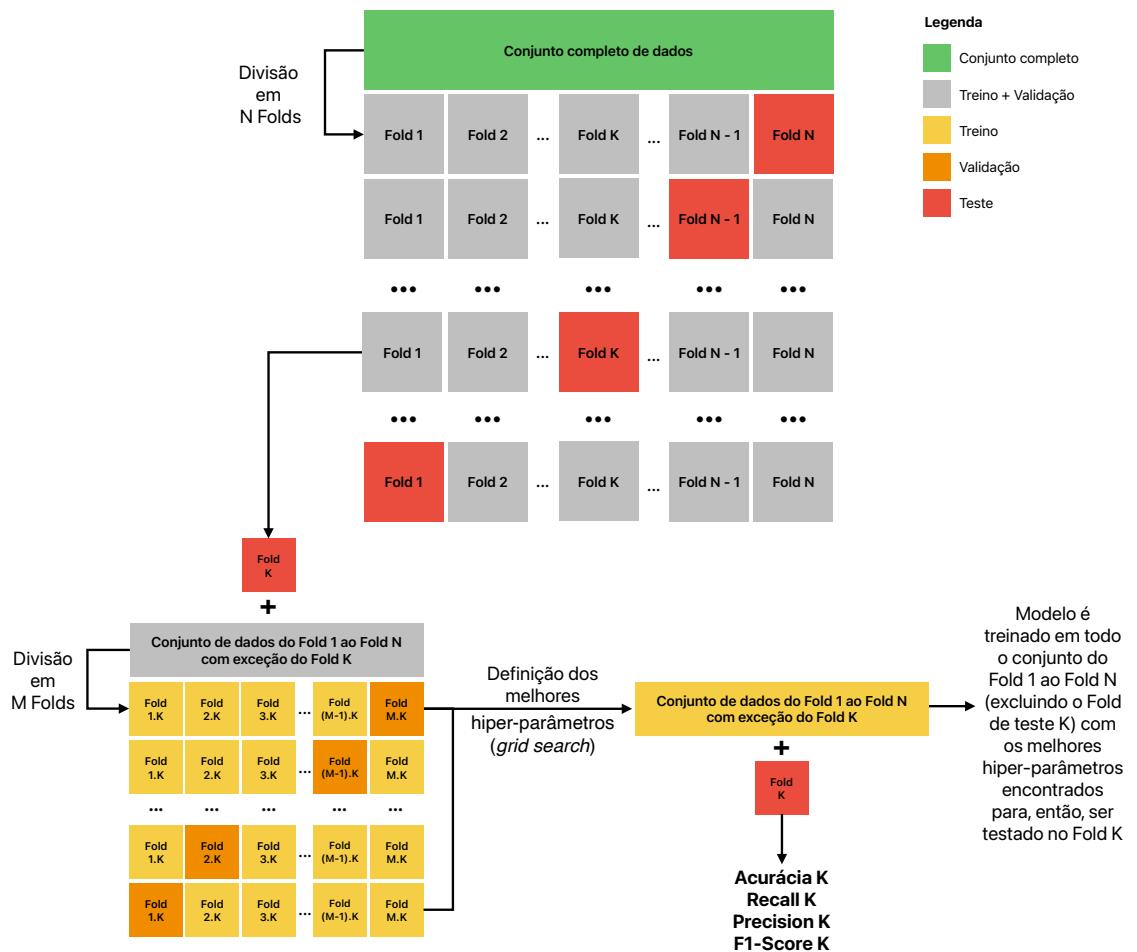


Figura 3: Esquemático do procedimento adotado neste projeto.

Para obter um bom equilíbrio entre a quantidade de execuções e o tempo computacional, foram adotados $N = 20$ e $K = 3$. Ou seja, realizou-se 20 iterações em que, em cada uma delas, 19/20 dos dados foram utilizados para treino e validação, enquanto 1/20 foi reservado para teste. No conjunto de treino e validação, os dados foram divididos em 3 Folds para a aplicação do Grid Search que, por sua vez, levou em conta os parâmetros descritos na Seção 2. Após a definição dos melhores hiperparâmetros, cada um dos modelos foi treinado nos seus respectivos 19/20 dos dados e testado no 1/20 restante. Por fim, como o problema abordado era desbalanceado, adotou-se uma estratégia de estratificação para garantir que a proporção das classes se mantivessem iguais em todos os Folds. Visando uma maior organização do relatório, os melhores conjuntos de hiperparâmetros encontrados anexados ao Apêndice.

3.4 Resultados

Para avaliar o desempenho dos algoritmos, foram gerados vários gráficos que proporcionam uma análise detalhada dos resultados encontrados após as N iterações. Entre os gráficos gerados, está a distribuição das Acurárias em teste, que permite uma visão clara da variabilidade e consistência do desempenho dos modelos ao longo das repetições. Além disso, foram gerados Box Plots dos F1-scores, que facilitam a comparação da performance dos algoritmos para as diferentes classes, destacando a mediana e possíveis *outliers* dos resultados, além de fornecer informações sobre a dispersão dessa métrica nas diferentes iterações.

Por fim, foram geradas quatro Matrizes de Confusão:

- **Matriz acumulada:** Em que somou-se as células de todas as iterações do respectivo modelo, permitindo inferir o total de acertos e erros acumulados ao longo de todas as iterações, proporcionando uma visão global da performance dos algoritmos.
- **Matriz média:** A partir da matriz acumulada, dividiu-se cada célula pelo total de iterações. Com isso, fornece uma visão do desempenho médio dos modelos em uma iteração.
- **Matriz normalizada pela linha:** Ao dividir cada célula da matriz acumulada pela soma dos elementos da sua respectiva linha, traz a visualização das taxas de acerto para cada classe, nos dando a proporção de previsões corretas e incorretas para cada classe real. Cada valor na diagonal principal representa o Recall para a classe correspondente, enquanto que cada valor fora da diagonal principal representa a taxa de falsos negativos para a classe na linha e a classe predita na coluna.
- **Matriz normalizada pela coluna:** Ao dividir cada célula da matriz acumulada pela soma dos elementos da sua respectiva coluna, ajuda a entender a precisão dos modelos

para cada classe, nos dando a proporção de previsões corretas e incorretas para cada classe predita. Cada valor na diagonal principal representa o Precision para a classe correspondente, enquanto que cada valor fora da diagonal principal representa a taxa de falsos positivos para a classe na coluna e a classe real na linha.

Essas matrizes permitiram visualizar a qualidade das classificações, evidenciando as taxas de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos para cada classe, além de auxiliar na identificação de quais classes os algoritmos estavam confundindo mais frequentemente e computar o Recall e Precision das classes para cada algoritmo. Assim, utilizando estas informações em conjunto com a informação da similaridade dos centroides (vide Figura 2), foi possível verificar se a análise feita anteriormente tinha fundamento. Devido ao grande volume de gráficos gerados e com o objetivo de manter o relatório mais enxuto e organizado, os resultados foram resumidos em tabelas e os gráficos completos foram disponibilizados no Apêndice.

3.5 Análise comparativa dos algoritmos

3.5.1 Análise de Acurárias

Uma vez rodada as N iterações, pode-se obter o valor médio da Acurácia juntamente com seu desvio-padrão para os dados de teste, conforme representado na Tabela 1.

| Modelo | Acurácia |
|--------|-----------------------|
| KNN | (88, 88 \pm 3, 92)% |
| DT | (87, 14 \pm 5, 04)% |
| RF | (91, 80 \pm 3, 96)% |
| LR | (86, 98 \pm 4, 05)% |
| SVM | (90, 54 \pm 3, 14)% |
| MLP | (89, 66 \pm 3, 32)% |

Tabela 1: Tabela comparativa das Acurárias médias.

Desta forma, pode-se verificar que o algoritmo que obteve o maior valor médio de Acurácia foi o Random Forest (RF), seguido pelo SVM. Por outro lado, a Regressão Logística (LR) e a Árvore de Decisão (DT) foram os que obtiveram os menores valores médios de Acurácia. Observando o desvio-padrão das medidas, pode-se verificar que os resultados obtidos pelos modelos de Árvore de Decisão foram aqueles que divergiram mais, o que pode ter ocorrido devido às diferentes máquinas terem sofrido *overfitting* e, portanto, terem tido performance mais destoantes em teste. Em contrapartida, os resultados dos modelos baseados em SVM

foram aqueles que tiveram menor divergência. Vale destacar que para assegurar uma comparabilidade estatística mais precisa dos resultados, uma possibilidade seria considerar a realização de testes de hipótese.

3.5.2 Análise de tempo

Buscando ter uma métrica que auxiliasse na avaliação da eficiência computacional e viabilidade prática dos algoritmos, computou-se o tempo médio de treinamento e de inferência na base de teste (conforme representado na Tabela 2).

| Modelo | Tempo médio de treinamento | Tempo médio de inferência |
|--------|----------------------------|---------------------------|
| KNN | N/A ¹ | $2,18 \times 10^{-3}$ s |
| DT | $1,08 \times 10^{-2}$ s | $1,00 \times 10^{-4}$ s |
| RF | $9,90 \times 10^{-1}$ s | $7,23 \times 10^{-3}$ s |
| LR | $2,58 \times 10^0$ s | $2,55 \times 10^{-4}$ s |
| SVM | $2,73 \times 10^{-2}$ s | $2,20 \times 10^{-3}$ s |
| MLP | $9,78 \times 10^{-1}$ s | $1,74 \times 10^{-4}$ s |

Tabela 2: Tabela comparativa dos tempos médios de treinamento e inferência para cada algoritmo.

Com isso, observou-se que os algoritmos mais rápidos para treinamento foram a Árvore de Decisão e o SVM, enquanto a Regressão Logística e o Random Forest foram os mais lentos. A Regressão Logística mostrou-se cerca de 240 vezes mais lenta que a Árvore de Decisão, e o Random Forest aproximadamente 90 vezes.

Quanto ao tempo de inferência, a Árvore de Decisão foi o modelo mais rápido, enquanto o Random Forest apresentou o maior tempo de inferência, sendo cerca de 70 vezes mais lento que o seu concorrente mais veloz.

Todavia, vale destacar que para obter conclusões mais robustas, seria útil avaliar a performance dos modelos em conjuntos de dados maiores, realizar mais iterações e calcular não apenas a média, mas também o desvio-padrão dos tempos obtidos.

3.5.3 Análise dos F1-scores

Tratando-se de um problema multiclasse e desbalanceado, a Acurácia não pode ser a única métrica analisada para validação dos algoritmos. Com isto, computou-se o F1-score

¹Como o KNN é um algoritmo baseado em instâncias, a máquina armazena os dados de treinamento e adia a generalização até a fase de previsão. Com isto, não há formalmente uma etapa de treinamento.

para cada uma das classes do problema para as N iterações conforme representado nas Tabelas 3 e 4.

| Classe | F1-score | | |
|-------------------------|--------------------|--------------------|--------------------|
| | KNN | DT | RF |
| <i>Pastry</i> | (69, 75 ± 10, 59)% | (69, 46 ± 14, 27)% | (78, 90 ± 12, 19)% |
| <i>Z_Scratch</i> | (90, 82 ± 6, 59)% | (91, 53 ± 5, 84)% | (93, 90 ± 5, 26)% |
| <i>K_Scratch</i> | (97, 71 ± 1, 83)% | (96, 77 ± 2, 89)% | (98, 08 ± 2, 34)% |
| <i>Stains</i> | (94, 79 ± 9, 25)% | (90, 88 ± 9, 99)% | (94, 10 ± 10, 19)% |
| <i>Dirtiness</i> | (89, 69 ± 13, 95)% | (84, 12 ± 16, 09)% | (89, 33 ± 16, 91)% |
| <i>Bumps</i> | (85, 09 ± 6, 30)% | (82, 21 ± 7, 47)% | (89, 55 ± 5, 31)% |
| Média | (87, 98 ± 9, 93)% | (85, 83 ± 9, 61)% | (90, 64 ± 6, 61)% |

Tabela 3: Tabela comparativa dos F1-score's médios para o KNN, DT e RF.

| Classe | F1-score | | |
|-------------------------|--------------------|--------------------|--------------------|
| | LR | SVM | MLP |
| <i>Pastry</i> | (74, 54 ± 10, 21)% | (76, 86 ± 9, 04)% | (76, 31 ± 9, 24)% |
| <i>Z_Scratch</i> | (84, 95 ± 7, 46)% | (92, 73 ± 5, 87)% | (90, 79 ± 6, 23)% |
| <i>K_Scratch</i> | (96, 29 ± 2, 40)% | (97, 58 ± 2, 25)% | (96, 82 ± 2, 31)% |
| <i>Stains</i> | (93, 78 ± 8, 32)% | (93, 92 ± 9, 49)% | (95, 21 ± 8, 02)% |
| <i>Dirtiness</i> | (69, 48 ± 24, 30)% | (84, 36 ± 16, 25)% | (80, 24 ± 18, 95)% |
| <i>Bumps</i> | (84, 54 ± 6, 21)% | (88, 01 ± 5, 57)% | (87, 17 ± 5, 63)% |
| Média | (83, 93 ± 10, 47)% | (88, 91 ± 7, 50)% | (87, 76 ± 8, 18)% |

Tabela 4: Tabela comparativa dos F1-score's médios para o LR, SVM e MLP.

Analisando o valor médio do F1-score para cada algoritmo, verificou-se que os melhores e piores resultados foram consistentes com as conclusões da análise feita para a Acurácia, sendo os melhores resultados aqueles dos modelos de Floresta Aleatória e SVM, e os piores Regressão Logística e Árvore de Decisão. Ao analisar o desvio-padrão das médias, observou-se que diferentes conjuntos de modelos apresentaram variações significativas na dispersão dos resultados. Os modelos de Regressão Logística mostraram a maior variabilidade, enquanto os modelos baseados em Random Forest exibiram a menor.

Observando-se os valores médios de F1-score para cada uma classes individualmente, pode-se verificar que ***Pastry*** foi aquela que obteve os menores valores médios de F1-score, enquanto que ***K_Scratch*** foi aquela que obteve os maiores. Ademais, apesar de ter o menor

valor médio, a classe ***Pastry*** foi a segunda que apresentou maior desvio-padrão em média, o que indica que apesar deste valor ter sido relativamente baixo em média, variou consideravelmente de teste para teste, o que é reforçado ao analisar os Box Plots desta classe. Dentre os possíveis motivos, a classe ***Pastry*** pode apresentar características mais complexas ou menos distintivas em comparação com as outras classes, o que dificulta a tarefa de classificação e a torna mais suscetível a variações. Além disso, o fato de ser uma classe com um número reduzido de registros (conforme ilustrado na Figura 1) pode ter comprometido o treinamento dos modelos, resultando em maior variabilidade nos resultados. Outra possibilidade é que a separação dos Folds de teste não tenha sido ideal, o que também pode ter influenciado negativamente o desempenho da classificação.

Por fim, vale destacar que a classe ***K_Scratch*** foi aquela que apresentou menor variabilidade dos resultados de F1-score, o que indica que os resultados foram consistentes e semelhantes nas diversas iterações. Dentre os possíveis motivos para este sucesso pode-se destacar a grande quantidade de registros desta classe na base, além de ter, possivelmente, características menos complexas ou mais distintivas com relação às demais.

3.5.4 Análise das Matrizes de Confusão

Por fim, computou-se as Matrizes de Confusão para cada um dos algoritmos citados anteriormente. Analisando os resultados, pode-se verificar que, na grande maioria dos casos, a classe ***Pastry*** foi aquela com menor Recall e Precision médios, enquanto que ***K_Scratch*** foi a com maiores, corroborando com as conclusões tiradas com base no F1-score.

Diferente de outras métricas, as Matrizes de Confusão nos permitem identificar quais classes são mais frequentemente confundidas pelos modelos. Como esperado pelos resultados de F1-score, Recall e Precision, a classe ***Pastry*** foi a que apresentou maior confusão, sendo principalmente confundida com a classe ***Bumps*** e, em menor grau, com ***Z_Scratch***. Em média, cerca de 20% de todas as instâncias de ***Pastry*** foram confundidas com ***Bumps*** e 3% com ***Z_Scratch***.

A classe ***Dirtiness*** também merece destaque, com aproximadamente 9% de suas ocorrências sendo confundidas com ***Bumps*** e 8% com ***Pastry***. Já as classes ***Stains*** e ***Z_Scratch***, apesar de terem tido altos valores de Recall, nas vezes que foram confundidas isto ocorreu, principalmente, com a classe ***Bumps*** (cerca de 6% para ambos os casos). Analisando a classe ***Bumps***, verificou-se que cerca de 8% de suas instâncias foram classificadas como ***Pastry*** e 2% como ***Z_Scratch***.

Como destaque positivo, a classe ***K_Scratch*** teve a menor taxa de confusão em relação às demais, com um Recall de aproximadamente 97%. Nas poucas ocasiões em que foi confundida, isso ocorreu principalmente com ***Bumps*** (aproximadamente 2%).

Comparando-se estes resultados com a similaridade dos centroides (vide Figura 2), pode-se verificar que, de fato, as classes em que houveram maior confusão foram aquelas cujos centroides eram mais próximos. Ademais, pode-se verificar, também, que a classe que possuía centroide mais distante (*K_Scratch*) foi aquela em que ocorreu menor confusão com as demais. Além disso, a classe *Bumps*, que apresentou a menor distância média do centroide em relação às outras classes, foi também a que teve mais confusão com as demais.

4 Conclusões

Ao longo do desenvolvimento deste projeto, pode-se verificar na prática como os diferentes algoritmos de aprendizado estudados na disciplina podem ser aplicados para resolver o mesmo problema. Ademais, também foi possível identificar a inadequação de um deles (Naive Bayes) devido às características das variáveis preditoras. A aplicação de estratégias de ajuste de hiperparâmetros, aliada à avaliação de desempenho dos modelos e à interpretação estatística, foi essencial para comparar suas performances de forma eficaz.

Para a base de dados utilizada, os modelos Random Forest e Support Vector Machine destacaram-se em termos de desempenho, avaliado a partir do prisma da acurácia, enquanto que a Regressão Logística e a Árvore de Decisão apresentaram resultados destoantes negativamente. No que se refere às classes da variável alvo, a avaliação com base no F1-score, análise das matrizes de confusão e medidas de similaridade indicaram que a classe **Pasty** foi aquela em que houve maior confusão por parte dos modelos - sendo frequentemente confundida com a classe **Bumps** -, enquanto a classe **K_Scratch** foi mais facilmente distinguível das demais. Além disso, pode-se verificar que a classe **Bumps** foi a mais presente nas confusões das demais classes, o que pode indicar que, em geral, é a que apresenta mais similaridade com as demais. Com relação aos tempos de treinamento, foi possível verificar grandes discrepâncias entre alguns algoritmos, sendo Regressão Logística o mais lento e a Árvore de Decisão o mais rápido. Já referente ao tempo de inferência, pode-se verificar que o SVM foi o mais lento e a Árvore de Decisão o mais rápido.

Todavia, é importante ressaltar que, embora os algoritmos Random Forest e SVM tenham demonstrado os melhores desempenhos médios, não é possível indicá-los como os mais adequados para este problema sem um conhecimento mais aprofundado do contexto. É crucial entender, por exemplo, qual classe é mais crítica e deve ter a menor taxa de confusão para fazer uma escolha do melhor modelo, as implicações dos erros de classificação em cada classe, além de considerações específicas do domínio, como custos associados a falsos positivos ou falsos negativos e a necessidade de interpretabilidade do modelo.

Por fim, destaca-se que para buscar melhorias nos resultados da resolução do problema, algumas estratégias adicionais poderiam ser testadas: (i) explorar um intervalo mais amplo de hiperparâmetros durante o ajuste dos mesmos; (ii) realizar o balanceamento dos dados; (iii) utilizar outros modelos que também sejam adequados ao problema; (iv) aplicar técnicas de Feature Engineering para gerar variáveis que possam capturar informações mais relevantes; e (v) implementar métodos de Ensemble Learning para combinar as previsões de vários modelos.

Referências

- BEALE, R.; JACKSON, T. *Neural Computing - An Introduction*. [S.l.]: Institute of Physics Publishing, 1990.
- BEN, J. *Steel Plate Defect Prediction Using LGBM*. 2024. Disponível em: <<https://josephben.medium.com/steel-plate-defect-prediction-using-lgbm>>.
- BERRAR, D. *Cross-validation*. 2024. Disponível em: <http://berrar.com/resources/Berrar_EBCB_2nd_edition_Cross-validation_preprint.pdf>.
- BREIMAN, L. Random forests. *Machine Learning*, 2001.
- BURGES, C. J. C. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, v. 2, p. 121–167, 1998. Disponível em: <<https://doi.org/10.1023/A:1009715923555>>.
- LIASHCHYNSKYI, P.; LIASHCHYNSKYI, P. *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS*. 2019. Disponível em: <<https://arxiv.org/abs/1912.06059>>.
- MITCHELL, T. *Machine Learning*. [S.l.]: McGraw Hill, 1997.
- PENG, K. L. L. C.-Y. J.; INGERSOLL, G. M. An introduction to logistic regression analysis and reporting. *The Journal of Educational Research*, Routledge, v. 96, n. 1, p. 3–14, 2002. Disponível em: <<https://doi.org/10.1080/00220670209598786>>.
- RASCHKA, S. *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*. 2020. Disponível em: <<https://arxiv.org/abs/1811.12808>>.

Apêndice A

Tabela com p-valores dos testes de normalidade realizados nas variáveis preditoras

Resultados dos Testes de Normalidade

| Variável ¹ | P-valor |
|---------------------------------|-----------------------|
| <i>X_Minimum</i> | 0,00 |
| <i>X_Maximum</i> | 0,00 |
| <i>Y_Minimum</i> | 0,00 |
| <i>Y_Maximum</i> | 0,00 |
| <i>Pixels_Areas</i> | 0,00 |
| <i>X_Perimeter</i> | 0,00 |
| <i>Y_Perimeter</i> | 0,00 |
| <i>Sum_of_Luminosity</i> | 0,00 |
| <i>Minimum_of_Luminosity</i> | 0,00 |
| <i>Maximum_of_Luminosity</i> | 0,00 |
| <i>Length_of_Conveyer</i> | 0,00 |
| <i>Steel_Plate_Thickness</i> | 0,00 |
| <i>Edges_Index</i> ² | 0,00 |
| <i>Empty_Index</i> | $2,49 \times 10^{-8}$ |
| <i>Square_Index</i> | 0,00 |
| <i>Edges_X_Index</i> | 0,00 |
| <i>Edges_Y_Index</i> | 0,00 |
| <i>Outside_Global_Index</i> | 0,00 |
| <i>LogOfAreas</i> | 0,00 |
| <i>Log_X_Index</i> | 0,00 |
| <i>Log_Y_Index</i> | $2,65 \times 10^{-6}$ |
| <i>Orientation_Index</i> | 0,00 |
| <i>Luminosity_Index</i> | 0,00 |
| <i>SigmoidOfAreas</i> | 0,00 |

Tabela 5: P-valores dos testes de normalidade aplicados nas variáveis preditoras.

¹As variáveis categóricas foram removidas da análise.

²Apesar do nome sugerir que é um inteiro, o índice é representado por um número decimal que relaciona-se ao posicionamento do defeito.

Apêndice B

Tabelas das ocorrências dos conjuntos de hiperparâmetros definidos nas execuções do grid search

K-Nearest Neighbors (KNN)

| Ocorrências | Hiperparâmetros | | |
|-------------|--------------------|-----------------------|-----------|
| | Número de vizinhos | Potência de Minkowski | Peso |
| 8 | 5 | 1 | Distância |
| 7 | 3 | 1 | Distância |
| 5 | 1 | 1 | Uniforme |

Tabela 6: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para o KNN.

Árvore de Decisão (DT)

| Ocorrências | Hiperparâmetros | | |
|-------------|-----------------|------------------------------|---------------------------------|
| | Profundidade | Mínimo de amostras por folha | Mínimo de amostras para divisão |
| 5 | Não limitado | 1 | 2 |
| 3 | 10 | 1 | 5 |
| 3 | Não limitado | 5 | 2 |
| 2 | 10 | 2 | 5 |
| 2 | 10 | 2 | 10 |
| 2 | 10 | 1 | 2 |
| 1 | 10 | 5 | 2 |
| 1 | Não limitado | 2 | 10 |
| 1 | 10 | 1 | 10 |

Tabela 7: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para o DT.

Floresta Aleatória (RF)

| Ocorrências | Profundidade | Hiperparâmetros | | |
|-------------|--------------|------------------------------|---------------------------------|-----------------------|
| | | Mínimo de amostras por folha | Mínimo de amostras para divisão | Número de estimadores |
| 5 | Não limitado | 1 | 2 | 500 |
| 5 | Não limitado | 1 | 2 | 200 |
| 3 | Não limitado | 1 | 5 | 500 |
| 2 | Não limitado | 1 | 2 | 100 |
| 1 | 20 | 1 | 2 | 200 |
| 1 | Não limitado | 1 | 2 | 50 |
| 1 | Não limitado | 2 | 2 | 200 |
| 1 | 20 | 1 | 5 | 500 |
| 1 | 20 | 1 | 2 | 50 |

Tabela 8: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para o RF.

Regressão Logística (LR)

| Ocorrências | Hiperparâmetros ¹ | | | |
|-------------|------------------------------|---------------|---------------------|-----------|
| | Força de penalização | Razão de L1 | Tipo de penalização | Algoritmo |
| 9 | 1 | Nao utilizado | L1 | SAGA |
| 2 | 10 | Nao utilizado | L1 | SAGA |
| 2 | 1 | Nao utilizado | L2 | Newton-CG |
| 1 | 10 | Nao utilizado | L2 | Newton-CG |
| 1 | 1 | Nao utilizado | L2 | SAGA |
| 1 | 1 | Nao utilizado | L2 | LBFGS |
| 1 | 1 | 0,25 | Elasticnet | SAGA |
| 1 | 1 | 0,5 | Elasticnet | SAGA |
| 1 | 100 | Nao utilizado | L2 | SAG |
| 1 | 10 | 0,5 | Elasticnet | SAGA |

Tabela 9: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para a LR.

Support Vector Machine (SVM)

| Ocorrências | Hiperparâmetros ² | | | |
|-------------|------------------------------|---------------------------|--------------------------------------|------------|
| | Força de penalização | Grau do kernel polinomial | Alcance da influência de treinamento | Kernel |
| 9 | 10 | Não utilizado | Escalonado ² | RBF |
| 3 | 100 | Não utilizado | 0,01 | RBF |
| 2 | 10 | 3 | Escalonado | Polinomial |
| 2 | 10 | Não utilizado | 0,01 | RBF |
| 2 | 10 | Não utilizado | 0,1 | RBF |
| 1 | 1 | 3 | 0,1 | Polinomial |
| 1 | 1 | Não utilizado | 0,1 | RBF |

Tabela 10: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para o SVM.

¹Durante a aplicação do Grid Search, filtrou-se as combinações inválidas (*e.g.*, definição da razão de L1 quando o tipo de penalização não é Elasticnet) de acordo com as restrições específicas de cada configuração, garantindo que apenas combinações viáveis fossem consideradas.

²Durante a aplicação do Grid Search, filtrou-se as combinações inválidas, garantindo que apenas combinações viáveis fossem consideradas.

Multi Layer Perceptron (MLP)

| Ocorrências | Hiperparâmetros | | | | | Algoritmo |
|-------------|--------------------|------------------------|--------------------------------------|----------------------|------|-----------|
| | Função de ativação | Força de regularização | Número de neurônios na camada oculta | Taxa de aprendizagem | | |
| 4 | tanh | 0,0001 | 50 | Constante | Adam | |
| 3 | ReLU | 0,0001 | 200 | Constante | Adam | |
| 2 | tanh | 0,01 | 100 | Constante | Adam | |
| 2 | ReLU | 0,01 | 100 | Constante | Adam | |
| 2 | tanh | 0,01 | 200 | Constante | Adam | |
| 2 | ReLU | 0,01 | 200 | Constante | Adam | |
| 1 | tanh | 0,0001 | 200 | Constante | Adam | |
| 1 | ReLU | 0,001 | 200 | Constante | Adam | |
| 1 | ReLU | 0,01 | 50 | Constante | Adam | |
| 1 | tanh | 0,01 | 50 | Constante | Adam | |
| 1 | tanh | 0,001 | 10 | Constante | Adam | |

Tabela 11: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para o MLP.

Apêndice C

Gráficos das métricas geradas para os modelos

K-Nearest Neighbors (KNN)

Matrizes de Confusão geradas após 20 iterações

| | | Matriz de Confusão agregada | | | | | | Matriz de Confusão média | | | | | | |
|------|---------|---|--------|-----------|-----------|--------|-----------|--|--------|-----------|-----------|--------|-----------|-------|
| Real | Predito | Pastry | 105 | 4 | 0 | 0 | 1 | 48 | 5.25 | 0.20 | 0.00 | 0.00 | 0.05 | 2.40 |
| | | Z_Scratch | 3 | 176 | 4 | 0 | 0 | 7 | 0.15 | 8.80 | 0.20 | 0.00 | 0.00 | 0.35 |
| | | K_Scratch | 2 | 2 | 382 | 0 | 0 | 5 | 0.10 | 0.10 | 19.10 | 0.00 | 0.00 | 0.25 |
| | | Stains | 0 | 1 | 0 | 68 | 0 | 3 | 0.00 | 0.05 | 0.00 | 3.40 | 0.00 | 0.15 |
| | | Dirtiness | 2 | 1 | 0 | 0 | 51 | 1 | 0.10 | 0.05 | 0.00 | 0.00 | 2.55 | 0.05 |
| | | Bumps | 29 | 14 | 5 | 3 | 6 | 345 | 1.45 | 0.70 | 0.25 | 0.15 | 0.30 | 17.25 |
| | | | | | | | | | | | | | | |
| | | Matriz de Confusão normalizada pelas linhas | | | | | | Matriz de Confusão normalizada pelas colunas | | | | | | |
| Real | Predito | Pastry | 0.66 | 0.03 | 0.00 | 0.00 | 0.01 | 0.30 | 0.74 | 0.02 | 0.00 | 0.00 | 0.02 | 0.12 |
| | | Z_Scratch | 0.02 | 0.93 | 0.02 | 0.00 | 0.00 | 0.04 | 0.02 | 0.89 | 0.01 | 0.00 | 0.00 | 0.02 |
| | | K_Scratch | 0.01 | 0.01 | 0.98 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.98 | 0.00 | 0.00 | 0.01 |
| | | Stains | 0.00 | 0.01 | 0.00 | 0.94 | 0.00 | 0.04 | 0.00 | 0.01 | 0.00 | 0.96 | 0.00 | 0.01 |
| | | Dirtiness | 0.04 | 0.02 | 0.00 | 0.00 | 0.93 | 0.02 | 0.01 | 0.01 | 0.00 | 0.00 | 0.88 | 0.00 |
| | | Bumps | 0.07 | 0.03 | 0.01 | 0.01 | 0.01 | 0.86 | 0.21 | 0.07 | 0.01 | 0.04 | 0.10 | 0.84 |
| | | | Pastry | Z_Scratch | K_Scratch | Stains | Dirtiness | Bumps | Pastry | Z_Scratch | K_Scratch | Stains | Dirtiness | Bumps |

Figura 4: Matrizes de confusão para o KNN.

Distribuição das Acurárias após 20 iterações

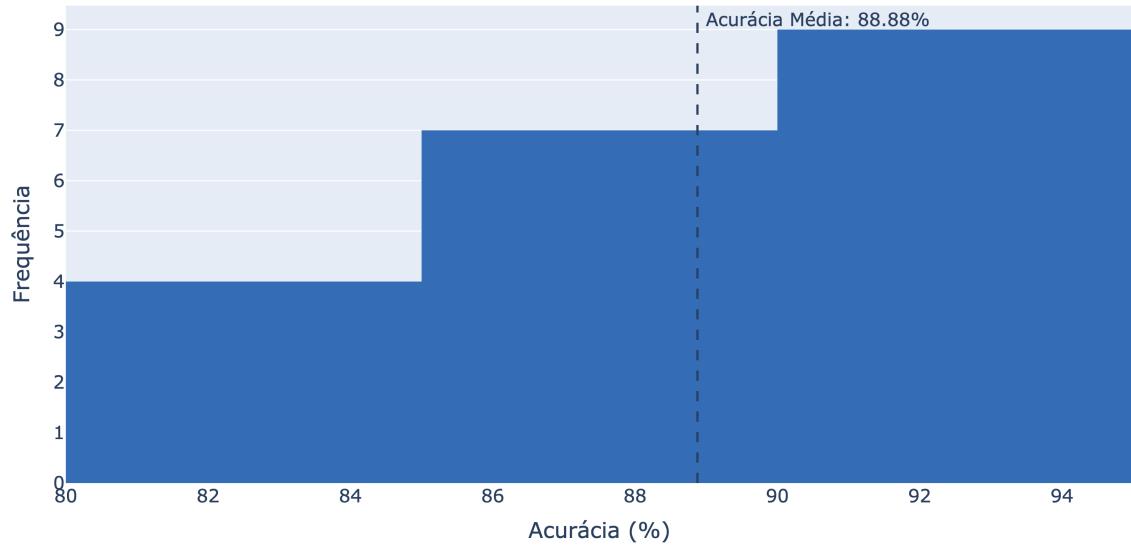


Figura 5: Distribuição das acurárias para o KNN.

Boxplots de F1-Scores por label após 20 iterações

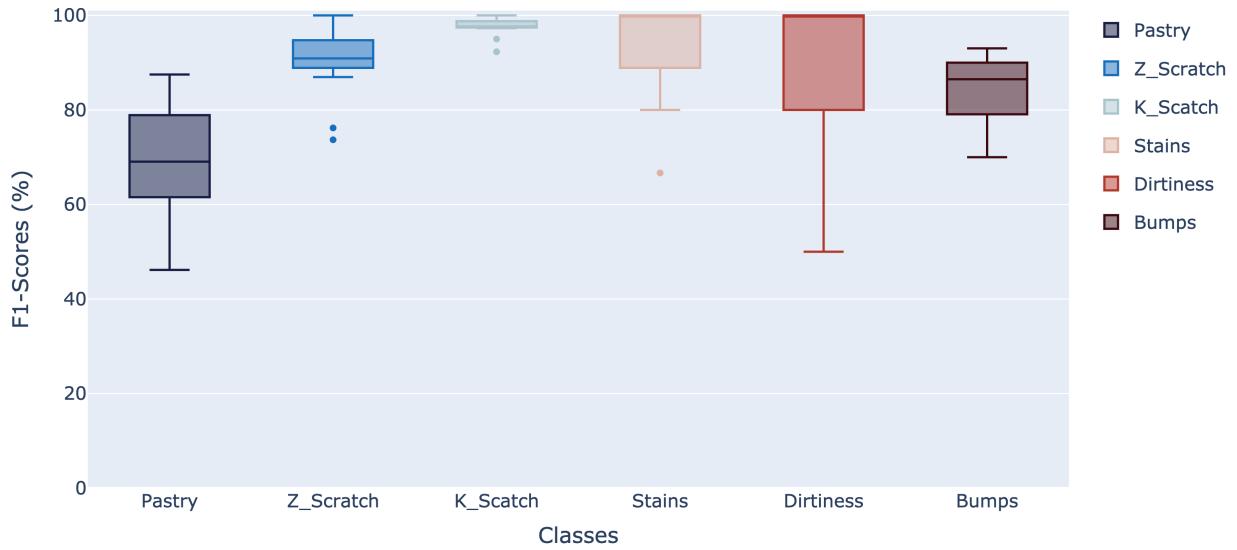


Figura 6: Box-plots dos F1-scores para o KNN.

Árvore de Decisão (DT)

Matrizes de Confusão geradas após 20 iterações

| | | Matriz de Confusão agregada | | | | | | Matriz de Confusão média | | | | | | |
|------|---------|-----------------------------|-----|-----|-----|----|----|--------------------------|------|------|-------|------|------|-------|
| Real | Predito | Pastry | 114 | 6 | 0 | 0 | 3 | 35 | 5.70 | 0.30 | 0.00 | 0.00 | 0.15 | 1.75 |
| | | Z_Scratch | 3 | 176 | 1 | 0 | 0 | 10 | 0.15 | 8.80 | 0.05 | 0.00 | 0.00 | 0.50 |
| | | K_Scratch | 1 | 4 | 376 | 1 | 0 | 9 | 0.05 | 0.20 | 18.80 | 0.05 | 0.00 | 0.45 |
| | | Stains | 0 | 0 | 0 | 66 | 0 | 6 | 0.00 | 0.00 | 0.00 | 3.30 | 0.00 | 0.30 |
| | | Dirtiness | 4 | 0 | 0 | 0 | 46 | 5 | 0.20 | 0.00 | 0.00 | 0.00 | 2.30 | 0.25 |
| | | Bumps | 48 | 8 | 9 | 6 | 4 | 327 | 2.40 | 0.40 | 0.45 | 0.30 | 0.20 | 16.35 |

| | | Matriz de Confusão normalizada pelas linhas | | | | | | Matriz de Confusão normalizada pelas colunas | | | | | | |
|------|---------|---|------|------|------|------|------|--|------|------|------|------|------|------|
| Real | Predito | Pastry | 0.72 | 0.04 | 0.00 | 0.00 | 0.02 | 0.22 | 0.67 | 0.03 | 0.00 | 0.00 | 0.06 | 0.09 |
| | | Z_Scratch | 0.02 | 0.93 | 0.01 | 0.00 | 0.00 | 0.05 | 0.02 | 0.91 | 0.00 | 0.00 | 0.00 | 0.03 |
| | | K_Scratch | 0.00 | 0.01 | 0.96 | 0.00 | 0.00 | 0.02 | 0.01 | 0.02 | 0.97 | 0.01 | 0.00 | 0.02 |
| | | Stains | 0.00 | 0.00 | 0.00 | 0.92 | 0.00 | 0.08 | 0.00 | 0.00 | 0.00 | 0.90 | 0.00 | 0.02 |
| | | Dirtiness | 0.07 | 0.00 | 0.00 | 0.00 | 0.84 | 0.09 | 0.02 | 0.00 | 0.00 | 0.00 | 0.87 | 0.01 |
| | | Bumps | 0.12 | 0.02 | 0.02 | 0.01 | 0.01 | 0.81 | 0.28 | 0.04 | 0.02 | 0.08 | 0.08 | 0.83 |

Figura 7: Matrizes de confusão para o DT.

Distribuição das Acurárias após 20 iterações

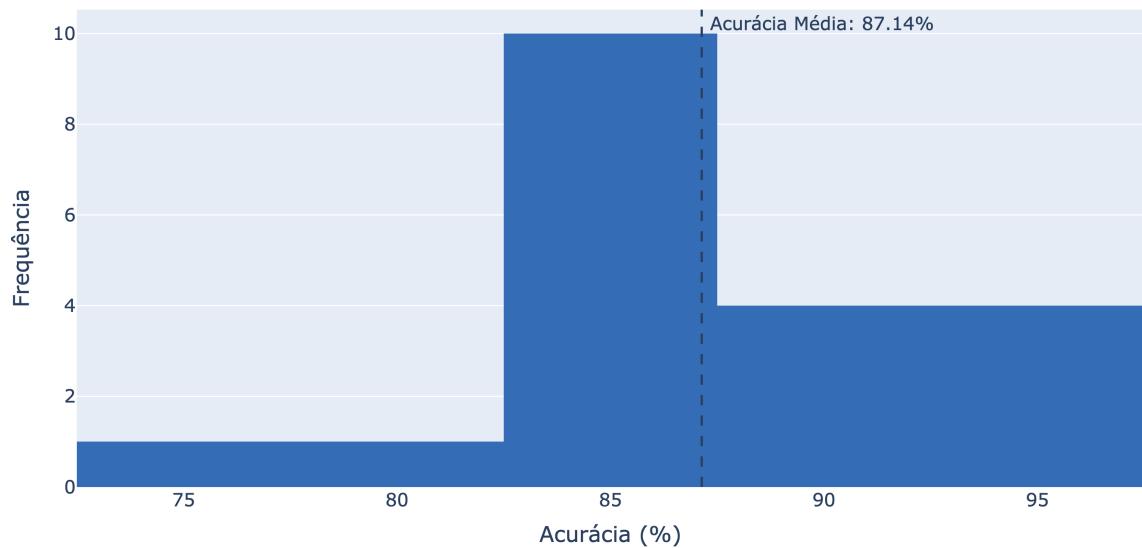


Figura 8: Distribuição das acurárias para o DT.

Boxplots de F1-Scores por label após 20 iterações

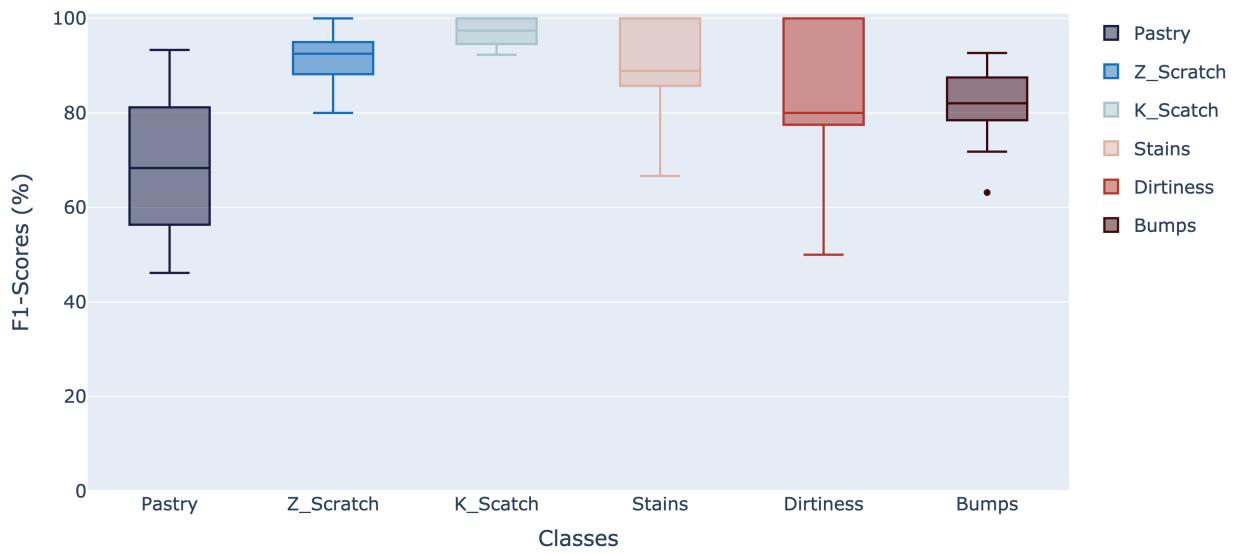


Figura 9: Box-plots dos F1-scores para o DT.

Floresta Aleatória (RF)

Matrizes de Confusão geradas após 20 iterações

| | | Matriz de Confusão agregada | | | | | | Matriz de Confusão média | | | | | | |
|------|-----------|-----------------------------|-----|-----|----|----|---|--------------------------|------|------|-------|------|------|-------|
| | | Pastry | 124 | 3 | 0 | 0 | 0 | 31 | 6.20 | 0.15 | 0.00 | 0.00 | 0.00 | 1.55 |
| Real | Pastry | 124 | 3 | 0 | 0 | 0 | 0 | 31 | 6.20 | 0.15 | 0.00 | 0.00 | 0.00 | 1.55 |
| | Z_Scratch | 3 | 173 | 3 | 0 | 0 | 0 | 11 | 0.15 | 8.65 | 0.15 | 0.00 | 0.00 | 0.55 |
| | K_Scratch | 4 | 0 | 382 | 1 | 0 | 0 | 4 | 0.20 | 0.00 | 19.10 | 0.05 | 0.00 | 0.20 |
| | Stains | 0 | 0 | 0 | 67 | 0 | 0 | 5 | 0.00 | 0.00 | 0.00 | 3.35 | 0.00 | 0.25 |
| | Dirtiness | 3 | 0 | 0 | 0 | 47 | 0 | 5 | 0.15 | 0.00 | 0.00 | 0.00 | 2.35 | 0.25 |
| | Bumps | 23 | 2 | 3 | 2 | 1 | 0 | 371 | 1.15 | 0.10 | 0.15 | 0.10 | 0.05 | 18.55 |

| | | Matriz de Confusão normalizada pelas linhas | | | | | | Matriz de Confusão normalizada pelas colunas | | | | | | | |
|------|-----------|---|------|------|------|------|------|--|--------|------|------|------|------|------|------|
| | | Pastry | 0.78 | 0.02 | 0.00 | 0.00 | 0.00 | 0.20 | Pastry | 0.79 | 0.02 | 0.00 | 0.00 | 0.00 | 0.07 |
| Real | Pastry | 0.78 | 0.02 | 0.00 | 0.00 | 0.00 | 0.20 | Pastry | 0.79 | 0.02 | 0.00 | 0.00 | 0.00 | 0.07 | |
| | Z_Scratch | 0.02 | 0.91 | 0.02 | 0.00 | 0.00 | 0.06 | Z_Scratch | 0.02 | 0.97 | 0.01 | 0.00 | 0.00 | 0.03 | |
| | K_Scratch | 0.01 | 0.00 | 0.98 | 0.00 | 0.00 | 0.01 | K_Scratch | 0.03 | 0.00 | 0.98 | 0.01 | 0.00 | 0.01 | |
| | Stains | 0.00 | 0.00 | 0.00 | 0.93 | 0.00 | 0.07 | Stains | 0.00 | 0.00 | 0.00 | 0.96 | 0.00 | 0.01 | |
| | Dirtiness | 0.05 | 0.00 | 0.00 | 0.00 | 0.85 | 0.09 | Dirtiness | 0.02 | 0.00 | 0.00 | 0.00 | 0.98 | 0.01 | |
| | Bumps | 0.06 | 0.00 | 0.01 | 0.00 | 0.00 | 0.92 | Bumps | 0.15 | 0.01 | 0.01 | 0.03 | 0.02 | 0.87 | |

| | | Predito | | | | | | Predito | | | | | |
|--|--|---------|-----------|-----------|--------|-----------|-------|---------|-----------|-----------|--------|-----------|-------|
| | | Pastry | Z_Scratch | K_Scratch | Stains | Dirtiness | Bumps | Pastry | Z_Scratch | K_Scratch | Stains | Dirtiness | Bumps |
| | | Pastry | Z_Scratch | K_Scratch | Stains | Dirtiness | Bumps | Pastry | Z_Scratch | K_Scratch | Stains | Dirtiness | Bumps |

Figura 10: Matrizes de confusão para o RF.

Distribuição das Acurárias após 20 iterações

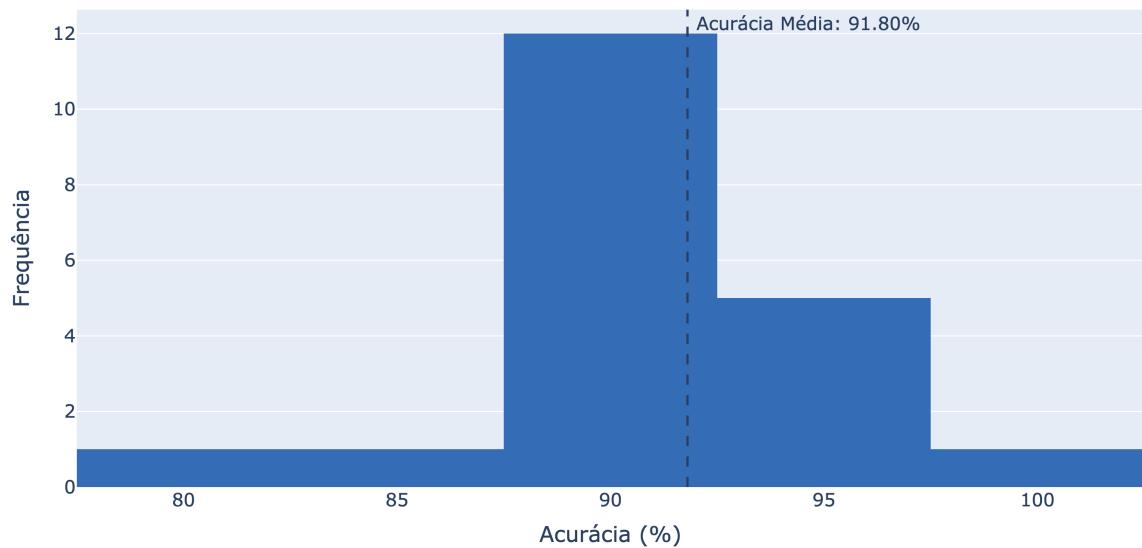


Figura 11: Distribuição das acurárias para o RF.

Boxplots de F1-Scores por label após 20 iterações

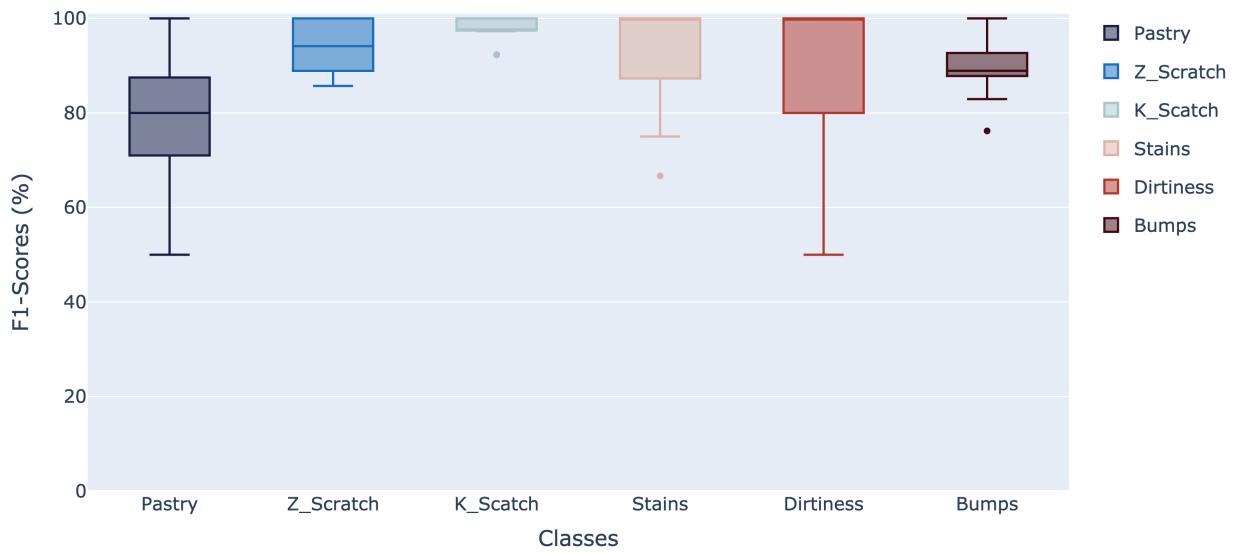


Figura 12: Box-plots dos F1-scores para o RF.

Regressão Logística (LR)

Matrizes de Confusão geradas após 20 iterações

| | | Matriz de Confusão agregada | | | | | | Matriz de Confusão média | | | | | | |
|------|---------|-----------------------------|-----|-----|-----|----|----|--------------------------|------|------|-------|------|------|-------|
| Real | Predito | Pastry | 116 | 9 | 1 | 1 | 4 | 27 | 5.80 | 0.45 | 0.05 | 0.05 | 0.20 | 1.35 |
| | | Z_Scratch | 2 | 165 | 5 | 0 | 5 | 13 | 0.10 | 8.25 | 0.25 | 0.00 | 0.25 | 0.65 |
| | | K_Scratch | 1 | 3 | 376 | 2 | 1 | 8 | 0.05 | 0.15 | 18.80 | 0.10 | 0.05 | 0.40 |
| | | Stains | 0 | 1 | 0 | 69 | 0 | 2 | 0.00 | 0.05 | 0.00 | 3.45 | 0.00 | 0.10 |
| | | Dirtiness | 6 | 1 | 0 | 0 | 40 | 8 | 0.30 | 0.05 | 0.00 | 0.00 | 2.00 | 0.40 |
| | | Bumps | 29 | 19 | 8 | 3 | 6 | 337 | 1.45 | 0.95 | 0.40 | 0.15 | 0.30 | 16.85 |

| | | Matriz de Confusão normalizada pelas linhas | | | | | | Matriz de Confusão normalizada pelas colunas | | | | | | |
|------|---------|---|------|------|------|------|------|--|------|------|------|------|------|------|
| Real | Predito | Pastry | 0.73 | 0.06 | 0.01 | 0.01 | 0.03 | 0.17 | 0.75 | 0.05 | 0.00 | 0.01 | 0.07 | 0.07 |
| | | Z_Scratch | 0.01 | 0.87 | 0.03 | 0.00 | 0.03 | 0.07 | 0.01 | 0.83 | 0.01 | 0.00 | 0.09 | 0.03 |
| | | K_Scratch | 0.00 | 0.01 | 0.96 | 0.01 | 0.00 | 0.02 | 0.01 | 0.02 | 0.96 | 0.03 | 0.02 | 0.02 |
| | | Stains | 0.00 | 0.01 | 0.00 | 0.96 | 0.00 | 0.03 | 0.00 | 0.01 | 0.00 | 0.92 | 0.00 | 0.01 |
| | | Dirtiness | 0.11 | 0.02 | 0.00 | 0.00 | 0.73 | 0.15 | 0.04 | 0.01 | 0.00 | 0.00 | 0.71 | 0.02 |
| | | Bumps | 0.07 | 0.05 | 0.02 | 0.01 | 0.01 | 0.84 | 0.19 | 0.10 | 0.02 | 0.04 | 0.11 | 0.85 |

Figura 13: Matrizes de confusão para a LR.

Distribuição das Acurárias após 20 iterações

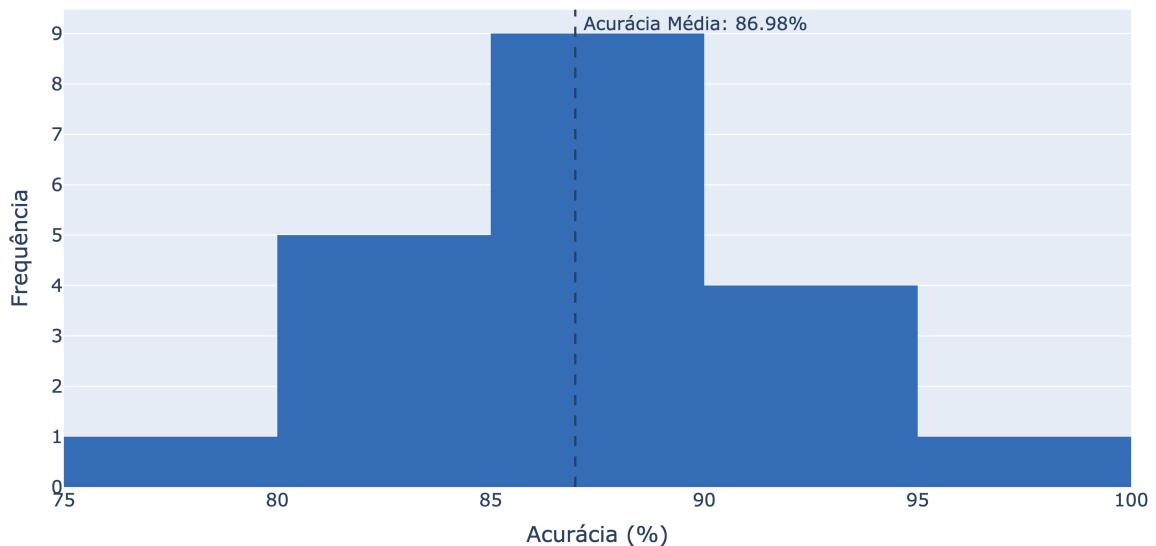


Figura 14: Distribuição das acurárias para a LR.

Boxplots de F1-Scores por label após 20 iterações

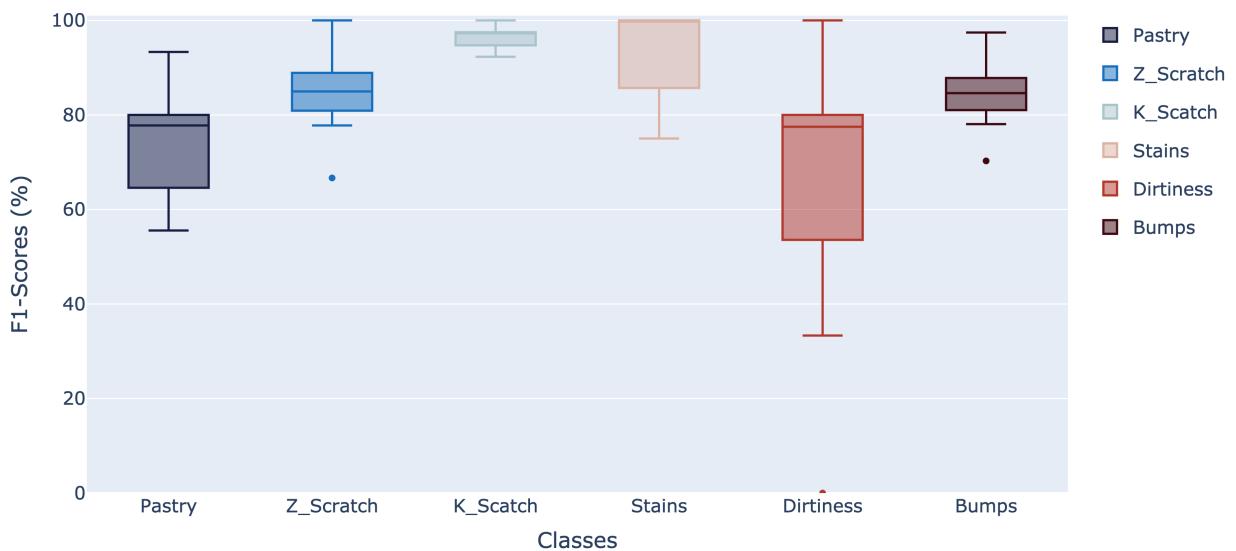


Figura 15: Box-plots dos F1-scores para a LR.

Support Vector Machine (SVM)

Matrizes de Confusão geradas após 20 iterações

| | | Matriz de Confusão agregada | | | | | | Matriz de Confusão média | | | | | | |
|------|---------|-----------------------------|-----|-----|-----|----|----|--------------------------|------|------|-------|------|------|-------|
| Real | Predito | Pastry | 120 | 2 | 1 | 0 | 3 | 32 | 6.00 | 0.10 | 0.05 | 0.00 | 0.15 | 1.60 |
| | | Z_Scratch | 2 | 177 | 5 | 0 | 0 | 6 | 0.10 | 8.85 | 0.25 | 0.00 | 0.00 | 0.30 |
| | | K_Scratch | 2 | 2 | 381 | 0 | 0 | 6 | 0.10 | 0.10 | 19.05 | 0.00 | 0.00 | 0.30 |
| | | Stains | 0 | 1 | 0 | 67 | 0 | 4 | 0.00 | 0.05 | 0.00 | 3.35 | 0.00 | 0.20 |
| | | Dirtiness | 5 | 0 | 0 | 0 | 46 | 4 | 0.25 | 0.00 | 0.00 | 0.00 | 2.30 | 0.20 |
| | | Bumps | 25 | 10 | 3 | 3 | 4 | 357 | 1.25 | 0.50 | 0.15 | 0.15 | 0.20 | 17.85 |

| | | Matriz de Confusão normalizada pelas linhas | | | | | | Matriz de Confusão normalizada pelas colunas | | | | | | |
|------|---------|---|------|------|------|------|------|--|------|------|------|------|------|------|
| Real | Predito | Pastry | 0.76 | 0.01 | 0.01 | 0.00 | 0.02 | 0.20 | 0.78 | 0.01 | 0.00 | 0.00 | 0.06 | 0.08 |
| | | Z_Scratch | 0.01 | 0.93 | 0.03 | 0.00 | 0.00 | 0.03 | 0.01 | 0.92 | 0.01 | 0.00 | 0.00 | 0.01 |
| | | K_Scratch | 0.01 | 0.01 | 0.97 | 0.00 | 0.00 | 0.02 | 0.01 | 0.01 | 0.98 | 0.00 | 0.00 | 0.01 |
| | | Stains | 0.00 | 0.01 | 0.00 | 0.93 | 0.00 | 0.06 | 0.00 | 0.01 | 0.00 | 0.96 | 0.00 | 0.01 |
| | | Dirtiness | 0.09 | 0.00 | 0.00 | 0.00 | 0.84 | 0.07 | 0.03 | 0.00 | 0.00 | 0.00 | 0.87 | 0.01 |
| | | Bumps | 0.06 | 0.02 | 0.01 | 0.01 | 0.01 | 0.89 | 0.16 | 0.05 | 0.01 | 0.04 | 0.08 | 0.87 |

Figura 16: Matrizes de confusão para o SVM.

Distribuição das Acurárias após 20 iterações

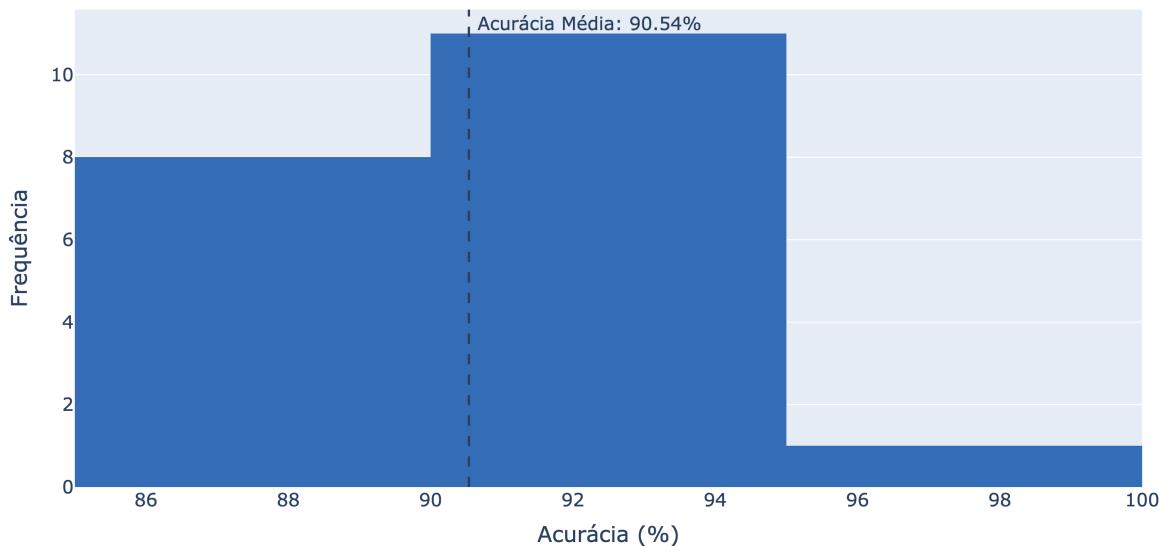


Figura 17: Distribuição das acurárias para o SVM.

Boxplots de F1-Scores por label após 20 iterações

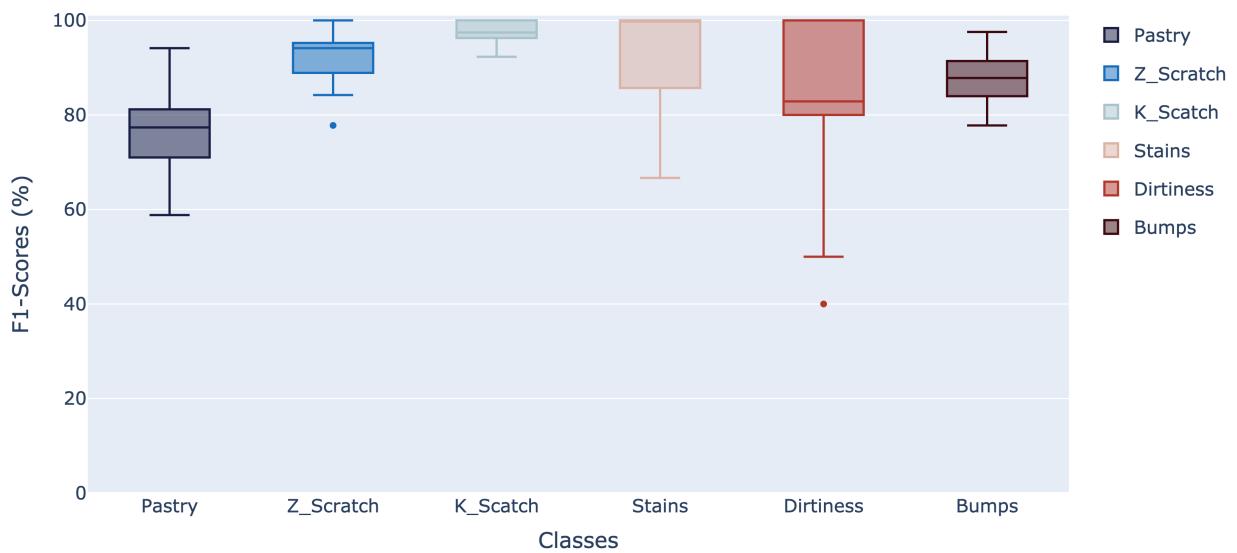


Figura 18: Box-plots dos F1-scores para o SVM.

Multi Layer Perceptron (MLP)

Matrizes de Confusão geradas após 20 iterações

| | | Matriz de Confusão agregada | | | | | | Matriz de Confusão média | | | | | | |
|------|---------|-----------------------------|-----|-----|-----|----|----|--------------------------|------|------|-------|------|------|-------|
| Real | Predito | Pastry | 121 | 4 | 2 | 0 | 3 | 28 | 6.05 | 0.20 | 0.10 | 0.00 | 0.15 | 1.40 |
| | | Z_Scratch | 1 | 172 | 6 | 0 | 0 | 11 | 0.05 | 8.60 | 0.30 | 0.00 | 0.00 | 0.55 |
| | | K_Scratch | 1 | 2 | 381 | 2 | 0 | 5 | 0.05 | 0.10 | 19.05 | 0.10 | 0.00 | 0.25 |
| | | Stains | 0 | 1 | 0 | 69 | 0 | 2 | 0.00 | 0.05 | 0.00 | 3.45 | 0.00 | 0.10 |
| | | Dirtiness | 6 | 1 | 0 | 0 | 43 | 5 | 0.30 | 0.05 | 0.00 | 0.00 | 2.15 | 0.25 |
| | | Bumps | 29 | 9 | 7 | 2 | 4 | 351 | 1.45 | 0.45 | 0.35 | 0.10 | 0.20 | 17.55 |

| | | Matriz de Confusão normalizada pelas linhas | | | | | | Matriz de Confusão normalizada pelas colunas | | | | | | |
|------|---------|---|------|------|------|------|------|--|------|------|------|------|------|------|
| Real | Predito | Pastry | 0.77 | 0.03 | 0.01 | 0.00 | 0.02 | 0.18 | 0.77 | 0.02 | 0.01 | 0.00 | 0.06 | 0.07 |
| | | Z_Scratch | 0.01 | 0.91 | 0.03 | 0.00 | 0.00 | 0.06 | 0.01 | 0.91 | 0.02 | 0.00 | 0.00 | 0.03 |
| | | K_Scratch | 0.00 | 0.01 | 0.97 | 0.01 | 0.00 | 0.01 | 0.01 | 0.01 | 0.96 | 0.03 | 0.00 | 0.01 |
| | | Stains | 0.00 | 0.01 | 0.00 | 0.96 | 0.00 | 0.03 | 0.00 | 0.01 | 0.00 | 0.95 | 0.00 | 0.00 |
| | | Dirtiness | 0.11 | 0.02 | 0.00 | 0.00 | 0.78 | 0.09 | 0.04 | 0.01 | 0.00 | 0.00 | 0.86 | 0.01 |
| | | Bumps | 0.07 | 0.02 | 0.02 | 0.00 | 0.01 | 0.87 | 0.18 | 0.05 | 0.02 | 0.03 | 0.08 | 0.87 |

Figura 19: Matrizes de confusão para o MLP.

Distribuição das Acurárias após 20 iterações

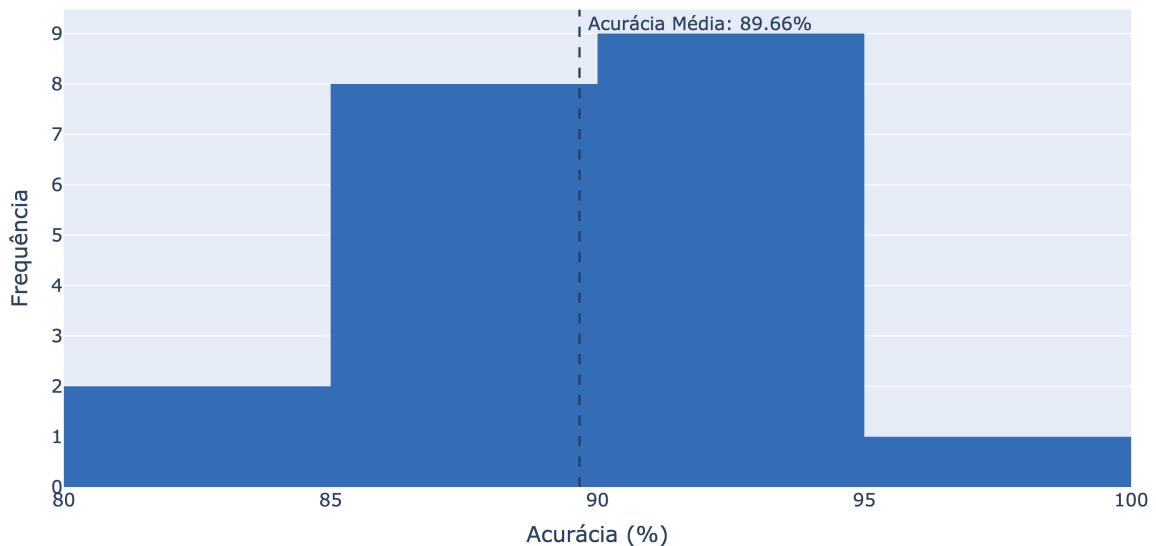


Figura 20: Distribuição das acurárias para o MLP.

Boxplots de F1-Scores por label após 20 iterações

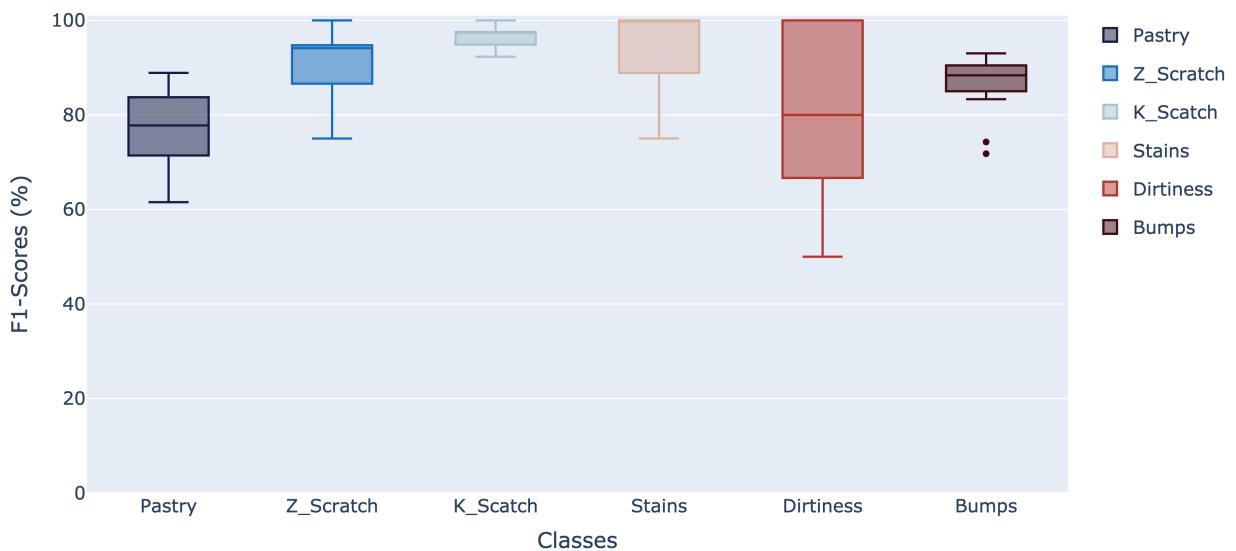


Figura 21: Box-plots dos F1-scores para o MLP.

Apêndice D

Repositório com os códigos

Link para o repositório

Os códigos desenvolvidos neste projeto estão disponíveis no GitHub e no GitLab da UFPE, oferecendo acesso completo às implementações e experimentos realizados. Para mais detalhes, consulte o repositório nos links:

- **GitHub:** Disponível em <https://github.com/azzolinovarella/steel-plates-faults>
- **GitLab UFPE** (acesso restrito a alunos e professores): Disponível em <https://gitlab.cin.ufpe.br/malc/projeto-final-aprendizado-maquina-i>