

Aprendizagem de Máquina II

## Projeto Final

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



26 de agosto de 2024

# 1 Introdução

O presente documento é o relatório final de execução do projeto entregue na disciplina Aprendizagem de Máquina II 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 deste trabalho é dar continuidade ao estudo iniciado na disciplina anterior, Aprendizagem de Máquina I, aplicando e aprofundando os conhecimentos adquiridos por meio da introdução de novas técnicas, como o balanceamento de dados e a combinação de classificadores.

## 2 Conceitos Básicos

### 2.1 Técnicas de pré-processamento

Desbalanceamento em conjuntos de dados ocorrem em problemas de classificação quando as categorias não estão representadas em volume similar. He e Garcia (2009) trataram sobre o aprendizado de máquina em problemas desta natureza. Segundo os mesmos, na prática isso significa que há a predominância de instâncias de uma (ou mais) classe majoritária, enquanto uma (ou mais) classe minoritária possui muito menos ocorrências. Este desbalanço pode levar máquinas de aprendizado a terem um bom desempenho na classe majoritária enquanto falham em identificar os padrões da classe minoritária. Para lidar com este problema, algumas técnicas de pré-processamento podem ser adotadas, como a reamostragem. Em geral, a reamostragem pode ser classificada em dois tipos diferentes: subamostragem (*undersampling*), em que a classe majoritária pode ser reduzida a partir da remoção (aleatória ou não) de algumas de suas instâncias ou por meio da geração de um número menor de instâncias; e sobreamostragem (*oversampling*), em que a classe minoritária é aumentada a partir da replicação de instâncias existentes ou da geração de novas instâncias, como no caso do método SMOTE.

He e Garcia (2009) descrevem a subamostragem aleatória (Random Undersampling) como a remoção de instâncias da classe majoritária aleatoriamente até obter uma distribuição de dados considerada balanceada.

Já Chawla et al. (2002) propuseram o SMOTE (*Synthetic Minority Over-sampling Technique*), uma técnica em que novas amostras sintéticas para as classes minoritárias são geradas a partir da interpolação entre as instâncias já existentes no espaço de características, aumentando assim o número de exemplos. Esta geração sintética de instâncias leva em conta um número  $k$  de vizinhos mais próximos para cada instância da classe minoritária. O método se demonstrou eficaz para melhorar o desempenho de classificadores desbalanceados, aumentando o número de exemplos da classe minoritária sem introduzir *overfitting* - o que geralmente ocorre com a simples replicação de instâncias existentes.

## 2.2 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.

### 2.2.1 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.

### 2.2.2 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).

### 2.2.3 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 multiclases 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*.

#### 2.2.4 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.

#### 2.2.5 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.

### 2.2.6 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.

### 2.2.7 Sistemas de Múltiplos Classificadores

Breiman (1996) desenvolveu a técnica de Bagging, cuja ideia central passa pela criação de uma coleção de classificadores para melhorar a precisão e robustez da predição. Cada classificador é construído a partir de um subconjunto aleatório dos dados de treinamento. Para selecionar os dados, utiliza-se uma técnica conhecida como *bootstrap*: para cada classificador, 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 classificador seja potencialmente diferente dos demais.

Cada um dos classificadores é treinado separadamente, e para formar a previsão final do modelo suas previsões são agregadas. Em geral, essa agregação é feita de duas maneiras principais: no caso de regressão, a média das previsões é tomada; enquanto em classificação, é realizada uma votação majoritária.

Quando comparado com o treinamento individual de máquinas de aprendizagem, o Bagging mitiga problemas de *overfitting* devido a maior diversidade entre as máquinas, tendo, em geral, maior capacidade de generalização. Todavia, por si só não trata o problema de desbalanceamento de dados, apesar de em geral performar melhor que máquinas individuais em conjuntos de dados com esta característica. Além do mais, exigem um tempo de treinamento relativamente maior que uma única máquina, uma vez que esta máquina é uma composição de outras máquinas.

## 2.3 Testes de Hipóteses

Para comparação estatisticamente significativa do desempenho das múltiplas abordagens de aprendizado de máquina aplicadas, um conjunto de testes há de ser aplicado. Fay e Proschan (2010) discutem a importância da escolha adequada de testes estatísticos e os diversos fatores que influenciam essa decisão, como o tipo de variável, a distribuição dos dados, o tamanho da amostra, o número de grupos e a independência das observações. Esses fatores são fundamentais para assegurar a validade estatística do teste e a correta interpretação dos resultados.

O Teste t de Student é recomendado para comparar as médias entre dois grupos independentes quando as variáveis são contínuas e seguem uma distribuição aproximadamente normal. Por outro lado, o Teste U de Wilcoxon é uma alternativa ao Teste t, utilizado para comparar duas amostras pareadas ou relacionadas, especialmente quando os dados não seguem uma distribuição normal.

Para a escolha adequada do teste de hipótese comparativo dentre os descritos acima, faz-se necessária a avaliação da hipótese de aderência à normalidade das amostras de resultados de desempenho das abordagens de aprendizado de máquina avaliadas. Esta hipótese pode ser verificada a partir do teste proposto por D'agostino e Pearson (1973), que se baseia em medidas de assimetria e curtose dos dados para verificar a aderência dos dados à normalidade.

# 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

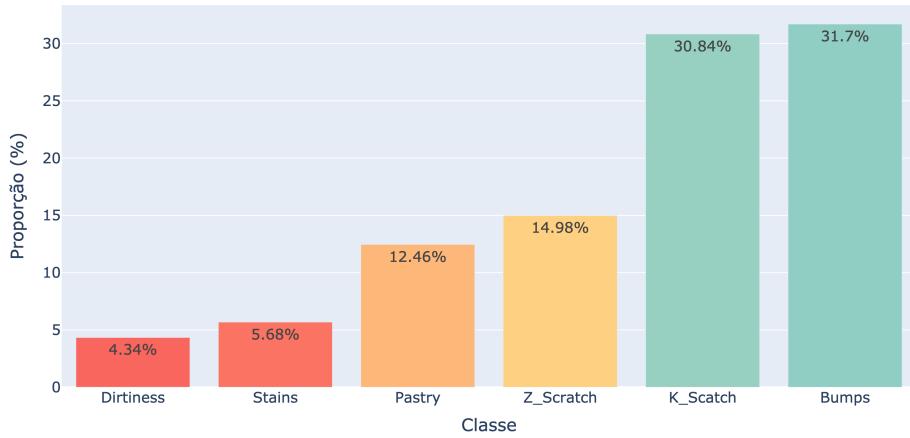


Figura 1: Proporção de classes na base de dados utilizada neste projeto.

Distâncias entre os centroides das classes

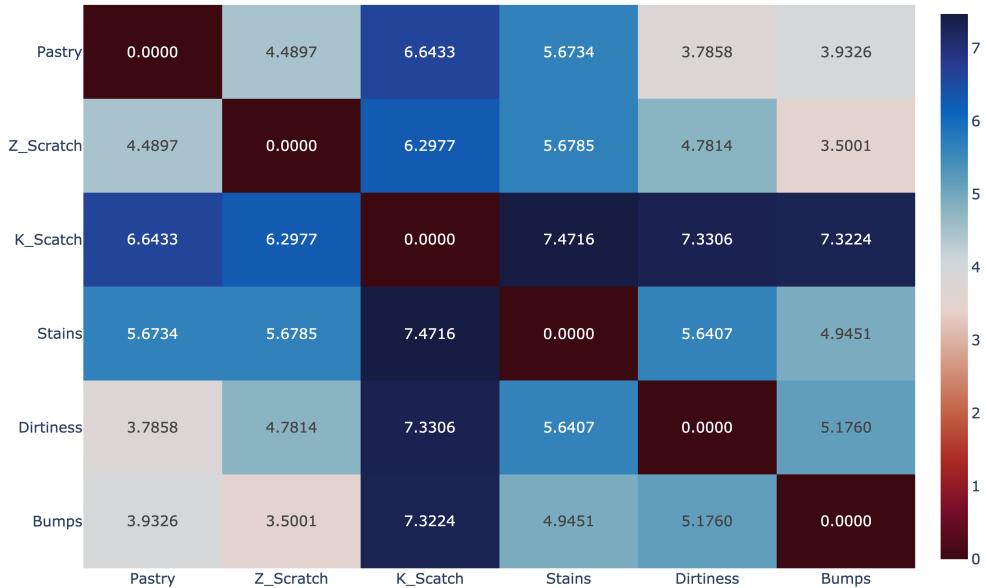


Figura 2: Distância entre os centroides dos dados normalizados.

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  $2 \times 2$ . 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

#### 3.3.1 Validação da performance das máquinas individualmente

Para avaliar a performance dos algoritmos 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.

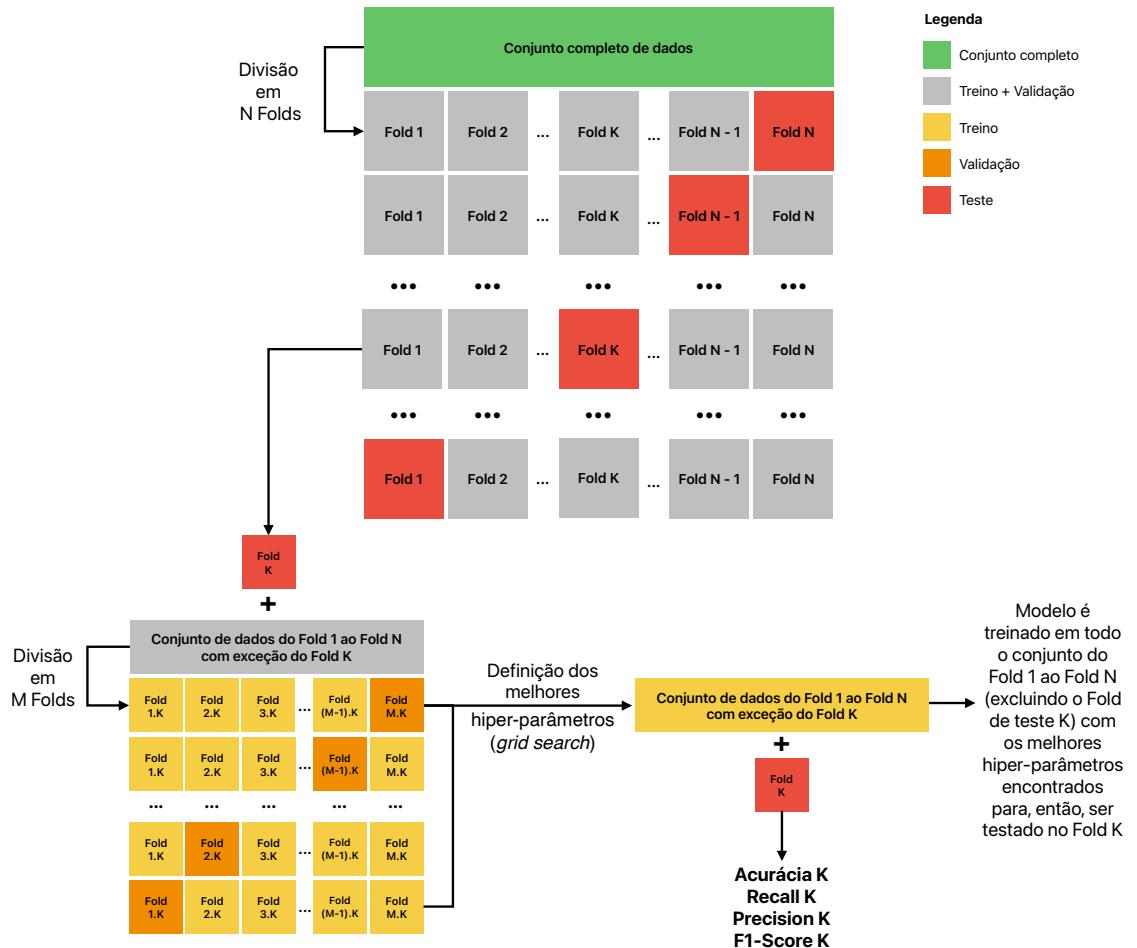


Figura 3: Esquemático do procedimento adotado neste projeto.

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 Subseção 2.2. 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.

### 3.3.2 Validação da performance das máquinas agregadas via Bagging

Uma vez realizado o procedimento anterior, aplicou-se a técnica de Bagging homogêneo para aprimorar os resultados, explorando três abordagens distintas: a primeira sem balanceamento prévio, onde o *bootstrap* foi realizado sobre toda a base de treino; a segunda realizando balanceamento via Random Undersampling; e a terceira realizando balanceamento via SMOTE.

Como o treinamento de modelos agrupados via Bagging é significativamente mais exaustivo do que o treinamento de uma única máquina, visto que há a necessidade de treinar um grande número de estimadores, realizou-se uma validação preliminar para avaliar a viabilidade do treinamento e da inferência desses modelos com base no tempo. Para isso, foi realizado um experimento no qual um modelo Bagging com 100 estimadores foi treinado três vezes usando um conjunto de treino do mesmo tamanho utilizado para treinar os modelos individuais e, em seguida, esse modelo Bagging foi empregado para prever os *targets* em um conjunto de teste, também mantendo o mesmo tamanho aplicado aos testes das máquinas individuais. Os tempos dessas operações foram registrados para determinar quais Ensemble de algoritmos eram viáveis para este estudo.

Além disso, devido ao curto prazo de desenvolvimento deste projeto, optou-se por utilizar os hiperparâmetros mais frequentes encontrados no Grid Search do estudo das máquinas individuais (disponibilizados no Apêndice) como padrão para os estimadores dos modelos de Bagging, variando apenas o número de estimadores no Ensemble. O procedimento seguiu o mesmo fluxo das máquinas individuais: o conjunto de dados foi dividido em  $N$  partes, onde, em cada iteração,  $N - 1$  partes foram utilizadas para treino e validação, e a parte restante para teste. Dentro dos  $N - 1$  Folds, foi realizado uma varredura para determinar o número ideal de estimadores, seguido por um novo treinamento completo antes de testar

no Fold de teste. Esse processo foi repetido até que cada parte servisse como conjunto de teste, permitindo a coleta das métricas em cada iteração. Os resultados foram anexados ao Apêndice, conforme feito anteriormente.

### 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 Validação da performance das máquinas individualmente

##### 3.5.1.1 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 1).

Modelo	Tempo médio de treinamento	Tempo médio de inferência
KNN	N/A <sup>1</sup>	$(2,22 \pm 5,49) \times 10^{-3}s$
DT	$(1,04 \pm 0,06) \times 10^{-2}s$	$(6,81 \pm 1,16) \times 10^{-5}s$
LR	$(1,79 \pm 1,22) \times 10^0s$	$(1,40 \pm 0,32) \times 10^{-4}s$
SVM	$(2,45 \pm 0,56) \times 10^{-2}s$	$(1,81 \pm 0,44) \times 10^{-3}s$
MLP	$(1,01 \pm 0,56) \times 10^0s$	$(1,90 \pm 0,86) \times 10^{-4}s$

Tabela 1: 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 que a Regressão Logística foi o mais demorado, sendo cerca de 172 vezes mais lento que a Árvore de Decisão.

Quanto ao tempo de inferência, a Árvore de Decisão foi o modelo mais rápido, enquanto que o KNN apresentou o maior tempo de inferência, sendo cerca de 32 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 e realizar testes de hipótese para ter evidências estatísticas de que os resultados são, de fato, diferentes. Além disto, o tempo de

---

<sup>1</sup>Como 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.

execução pode variar dependendo de muitos fatores ambientais, como carga do sistema, uso de CPU, memória disponível, e outros processos em execução. Essas variações podem tornar os resultados inconsistentes e difíceis de interpretar.

### 3.5.1.2 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 2.

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

Tabela 2: Tabela comparativa das Acurárias médias para cada algoritmo.

Desta forma, pode-se verificar que o algoritmo que obteve o maior valor médio de Acurácia foi o SVM, enquanto que 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.

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

	F1-score		
Classe	KNN	DT	LR
<i>Pastry</i>	(69, 75 ± 10, 59)%	(69, 46 ± 14, 27)%	(74, 54 ± 10, 21)%
<i>Z_Scratch</i>	(90, 82 ± 6, 59)%	(91, 53 ± 5, 84)%	(84, 95 ± 7, 46)%
<i>K_Scratch</i>	(97, 71 ± 1, 83)%	(96, 77 ± 2, 89)%	(96, 29 ± 2, 40)%
<i>Stains</i>	(94, 79 ± 9, 25)%	(90, 88 ± 9, 99)%	(93, 78 ± 8, 32)%
<i>Dirtiness</i>	(89, 69 ± 13, 95)%	(84, 12 ± 16, 09)%	(69, 48 ± 24, 30)%
<i>Bumps</i>	(85, 09 ± 6, 30)%	(82, 21 ± 7, 47)%	(84, 54 ± 6, 21)%
Média	(87, 98 ± 9, 93)%	(85, 83 ± 9, 61)%	(83, 93 ± 10, 47)%

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

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

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

Analizando 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 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 SVM 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 consideravel-

mente 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.

#### 3.5.1.4 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.

### 3.5.2 Validação da performance das máquinas agregadas via Bagging

Buscando entender o impacto que diferentes estratégias de agregação via Bagging tiveram na Acurácia, Precision e confusão das classes, realizaram-se novas análises comparando os resultados obtidos via agregação com aqueles encontrado ao analisar uma única máquina.

Conforme já enunciado na Subsubseção 3.3.2, como o treinamento de modelos Bagging é mais demorado do que o de uma única máquina, realizou-se uma validação preliminar para avaliar sua viabilidade em termos de tempo. Um modelo Bagging sem balanceamento prévio com 100 estimadores foi treinado e testado três vezes (em bases de dados com os mesmos tamanhos daquelas utilizadas para as máquinas individuais), registrando-se os tempos de operação para identificar quais algoritmos em Ensemble seriam viáveis para o estudo. Assim, obteve-se os resultados disponíveis na Tabela 5.

Modelo base	Tempo de treinamento	Tempo de inferência
KNN	N/A	$(1,02 \pm 0,26) \times 10^{-1}s$
DT	$(7,39 \pm 0,06) \times 10^{-1}s$	$(3,62 \pm 0,33) \times 10^{-3}s$
LR	$(1,93 \pm 0,01) \times 10^2s$	$(6,62 \pm 0,03) \times 10^{-3}s$
SVM	$(1,41 \pm 0,01) \times 10^0s$	$(1,14 \pm 0,01) \times 10^{-1}s$
MLP	$(7,21 \pm 0,02) \times 10^1s$	$(8,03 \pm 0,06) \times 10^{-3}s$

Tabela 5: Análise de viabilidade com base nos tempos de treinamento e inferência para cada Bagging de algoritmos.

Embora o tempo de inferência seja baixo, o treinamento dos Baggings de LR e MLP mostrou-se inviável, pois seus tempos de treinamento foram significativamente maiores em comparação com os demais. Portanto, devido ao curto prazo de desenvolvimento deste estudo, esses modelos foram excluídos das análises de Bagging, focando-se apenas nas máquinas KNN, DT e SVM.

#### 3.5.2.1 Análise de tempo

Após definir quais algoritmos seriam utilizados para o treinamento dos modelos de Bagging, foram calculados os tempos médios de treinamento e inferência, de forma similar aos modelos individuais. Os resultados foram registrados nas Tabelas 6 e 7.

Modelo base	Tempo de treinamento		
	Desbalanceado	Balanceado via <i>Undersampling</i>	Balanceado via <i>Oversampling</i>
KNN	N/A	N/A	N/A
DT	$(9,59 \pm 4,72) \times 10^{-1}$ s	$(5,84 \pm 2,07) \times 10^{-1}$	$(3,88 \pm 1,57) \times 10^0$
SVM	$(1,53 \pm 1,09) \times 10^0$ s	$(4,82 \pm 2,28) \times 10^{-1}$	$(3,04 \pm 2,36) \times 10^0$

Tabela 6: Tabela comparativa dos tempos médios de treinamento para cada máquina combinada via Bagging.

Modelo base	Tempo de inferência		
	Desbalanceado	Balanceado via <i>Undersampling</i>	Balanceado via <i>Oversampling</i>
KNN	$(6,77 \pm 5,03) \times 10^{-2}$ s	$(7,42 \pm 4,18) \times 10^{-2}$ s	$(1,61 \pm 0,77) \times 10^{-1}$ s
DT	$(4,77 \pm 2,30) \times 10^{-3}$ s	$(6,21 \pm 2,11) \times 10^{-3}$ s	$(5,67 \pm 2,23) \times 10^{-3}$ s
SVM	$(1,22 \pm 0,89) \times 10^{-1}$ s	$(7,29 \pm 3,47) \times 10^{-2}$ s	$(1,37 \pm 1,15) \times 10^{-1}$ s

Tabela 7: Tabela comparativa dos tempos médios de inferência para cada máquina combinada via Bagging.

Ao comparar os tempos de treinamento, observou-se que os Baggings de DT e SVM não apresentaram diferenças significativas dentro de uma mesma estratégia de balanceamento. No entanto, ao considerar as diferentes estratégias, percebe-se que os tempos de treinamento foram menores quando se utilizou a técnica de *undersampling*, enquanto os tempos mais longos foram observados com *oversampling*. Isso é esperado, uma vez que o *undersampling* resulta em um conjunto de treinamento reduzido, enquanto o *oversampling* aumenta o tamanho do conjunto de treinamento.

Em relação aos tempos de inferência, pode-se notar uma divergência maior entre os Baggings dentro da mesma estratégia de balanceamento. O DT mostrou-se o mais rápido, enquanto o KNN e o SVM apresentaram tempos de inferência mais longos. Além disso, os tempos de inferência para um mesmo modelo foram bastante semelhantes nas três diferentes estratégias de balanceamento, indicando que a escolha entre *undersampling*, *oversampling* ou dados desbalanceados não teve um impacto significativo no desempenho de inferência dos modelos.

Ao comparar esses resultados com os das máquinas individuais (vide Tabela 1), as diferenças tornam-se consideravelmente mais evidentes. No caso mais lento de Bagging, os tempos médios de treinamento para o DT e o SVM foram aproximadamente 373 e 124 vezes

mais longos, respectivamente. Em termos de inferência, os tempos para DT, SVM e KNN foram cerca de 83, 76 e 72 vezes mais demorados, respectivamente.

Vale destacar que os pontos levantados para a análise das máquinas individuais são os mesmos para este caso. Assim, para garantir a robustez das análises, os experimentos poderiam ser realizados com conjuntos de dados maiores e acompanhadas de testes estatísticos. Além disso, deve-se atentar que o tempo de execução, por sua vez, é influenciado por diversos fatores externos, o que pode gerar inconsistências nos resultados.

### 3.5.2.2 Análise de Acurárias

Rodadas todas as iterações, pode-se computar a Acurácia média e seu desvio-padrão nos dados de teste para cada um dos Baggings enunciados anteriormente conforme segue na Tabela 8.

Modelo base	Acurácia		
	Desbalanceado	Balanceado via <i>Undersampling</i>	Balanceado via <i>Oversampling</i>
KNN	(89, 35 ± 3, 32)%	(82, 64 ± 5, 15)%	(88, 72 ± 3, 57)%
DT	(91, 64 ± 4, 35)%	(88, 56 ± 4, 43)%	(92, 27 ± 4, 19)%
SVM	(91, 01 ± 2, 65)%	(88, 01 ± 3, 61)%	(90, 46 ± 3, 59)%

Tabela 8: Tabela comparativa das Acurárias médias para cada máquina combinada via Bagging.

Comparando-se as máquinas dentro de uma mesma estratégia, pode-se perceber que os Baggings de DT tiveram maiores valores médios de Acurácia nas diferentes abordagens. No entanto, os resultados dos Baggings de SVM foram semelhantes aos de DT, enquanto os resultados dos Baggings de KNN foram inferiores, especialmente quando se utilizou a estratégia de *undersampling*.

Comparando as diferentes estratégias, observou-se que a Árvore de Decisão (DT) obteve o maior valor médio de Acurácia com a aplicação da estratégia de *oversampling*, enquanto o KNN e o SVM apresentaram melhores resultados sem o balanceamento dos dados. Para avaliar o aprimoramento dos resultados de Acurácia ao usar a abordagem de Bagging em comparação com uma única máquina, foram realizados testes de hipótese. Como a distribuição de Acurárias foi aproximadamente normal para os Baggings de KNN e DT, utilizou-se o Teste t de Student. Para o SVM, que não apresentou normalidade, foi aplicado o Teste de Wilcoxon. Ambos os testes foram aplicados com abordagem unilateral para avaliar se a média (nos casos em que o Teste t de Student foi aplicado) ou a mediana (no caso em

que o Teste de Wilcoxon foi aplicado) tiveram um maior valor nas abordagens com Bagging quando comparadas com as máquinas individuais. Com isto, encontrou-se p-valores de 20,59%,  $4,76 \times 10^{-4}\%$  e 5,26% para o KNN, DT e SVM, respectivamente. Adotando um nível de significância de  $\alpha = 5\%$ , constatou-se que apenas o DT apresentou evidências estatísticas que indicam melhoria ao adotar a abordagem de Bagging. Visando maior organização do relatório, anexou-se os gráficos comparativos das distribuições ao Apêndice.

### 3.5.2.3 Análise dos F1-scores

Além dos valores de Acurácia, computou-se também os valores de F1-score para os diferentes Baggings de máquinas e estratégias, conforme representado nas Tabelas 9, 10 e 11.

Classe	F1-score - KNN		
	Desbalanceado	Balanceado via <i>Undersampling</i>	Balanceado via <i>Oversampling</i>
<b><i>Pastry</i></b>	(70,41 ± 10,16)%	(72,11 ± 12,02)%	(73,85 ± 9,89)%
<b><i>Z_Scratch</i></b>	(91,39 ± 4,63)%	(81,09 ± 6,87)%	(91,02 ± 6,56)%
<b><i>K_Scratch</i></b>	(97,70 ± 1,83)%	(95,71 ± 2,87)%	(97,56 ± 1,75)%
<b><i>Stains</i></b>	(94,08 ± 9,37)%	(85,10 ± 9,66)%	(92,43 ± 9,98)%
<b><i>Dirtiness</i></b>	(87,26 ± 15,58)%	(70,21 ± 17,07)%	(83,68 ± 17,99)%
<b><i>Bumps</i></b>	(86,63 ± 5,50)%	(77,16 ± 10,25)%	(85,07 ± 6,13)%
<b>Média</b>	(87,91 ± 9,53)%	(80,07 ± 9,45)%	(87,27 ± 8,30)%

Tabela 9: Tabela comparativa dos F1-score's médios para o Bagging de KNN.

Classe	F1-score - DT		
	Desbalanceado	Balanceado via <i>Undersampling</i>	Balanceado via <i>Oversampling</i>
<b><i>Pastry</i></b>	(78,67 ± 11,38)%	(77,32 ± 11,80)%	(80,82 ± 12,26)%
<b><i>Z_Scratch</i></b>	(95,00 ± 5,92)%	(91,79 ± 4,93)%	(95,15 ± 5,89)%
<b><i>K_Scratch</i></b>	(97,93 ± 2,33)%	(95,98 ± 3,02)%	(98,31 ± 2,29)%
<b><i>Stains</i></b>	(94,81 ± 10,07)%	(92,43 ± 9,98)%	(96,06 ± 8,97)%
<b><i>Dirtiness</i></b>	(88,08 ± 17,01)%	(78,35 ± 17,51)%	(85,90 ± 16,13)%
<b><i>Bumps</i></b>	(88,84 ± 6,58)%	(85,46 ± 5,89)%	(89,70 ± 5,67)%
<b>Média</b>	(90,56 ± 6,97)%	(86,89 ± 7,80)%	(90,99 ± 6,75)%

Tabela 10: Tabela comparativa dos F1-score's médios para o Bagging de DT.

	F1-score - SVM		
Classe	Desbalanceado	Balanceado via <i>Undersampling</i>	Balanceado via <i>Oversampling</i>
<b><i>Pastry</i></b>	(77, 97 ± 8, 49)%	(76, 61 ± 7, 77)%	(77, 02 ± 10, 34)%
<b><i>Z_Scratch</i></b>	(93, 56 ± 5, 80)%	(90, 95 ± 6, 26)%	(93, 59 ± 6, 46)%
<b><i>K_Scratch</i></b>	(97, 59 ± 2, 23)%	(96, 76 ± 2, 21)%	(97, 33 ± 2, 10)%
<b><i>Stains</i></b>	(95, 19 ± 9, 35)%	(89, 74 ± 10, 84)%	(94, 63 ± 9, 38)%
<b><i>Dirtiness</i></b>	(85, 07 ± 16, 62)%	(82, 63 ± 16, 76)%	(85, 45 ± 18, 20)%
<b><i>Bumps</i></b>	(88, 44 ± 4, 49)%	(82, 99 ± 6, 67)%	(87, 69 ± 5, 75)%
<b>Média</b>	(89, 64 ± 7, 32)%	(86, 61 ± 7, 22)%	(89, 12 ± 7, 81)%

Tabela 11: Tabela comparativa dos F1-score's médios para o Bagging de SVM.

Analisando os valores médios de F1-score para as diferentes estratégias, observou-se que o Bagging com dados desbalanceados e o balanceamento via *oversampling* apresentaram resultados semelhantes, enquanto o balanceamento via *undersampling* gerou médias consideravelmente mais baixas. Classe a classe, verificou-se que o *oversampling* foi a estratégia com maior consistência nos valores médios de F1-score, mostrando os maiores resultados para ***Pastry***, apesar dos menores para ***Dirtiness***.

Tomando como base que as melhores estratégias de Bagging eram aquelas com maiores valores médios de Acurácia, comparou-se também os resultados de F1-score desses Ensembles com as máquinas individuais. Para isso, analisou-se o DT com a estratégia de *oversampling* e o KNN e SVM sem balanceamento de dados. Como a distribuição de F1-scores foi aproximadamente normal para todos os Baggings, utilizou-se o Teste t de Student. Novamente, os testes foram aplicados com abordagem unilateral para avaliar se a média obteve um maior valor nas abordagens com Bagging quando comparadas às máquinas individuais. Com isto, encontrou-se p-valores de 53, 91%,  $2,49 \times 10^{-3}\%$  e 1, 16% para o KNN, DT e SVM, respectivamente. Adotando um nível de significância de  $\alpha = 5\%$ , observou-se que tanto o DT quanto o SVM apresentaram evidências estatísticas de que houve aprimoramento do F1-score ao aplicar a abordagem de Bagging. Os gráficos comparativos das distribuições encontram-se disponíveis no Apêndice.

### 3.5.2.4 Análise das Matrizes de Confusão

Inicialmente, ao analisar a estratégia de Bagging sem balanceamento, observou-se que para o KNN a classe mais frequentemente confundida, ***Pastry***, apresentou um ligeiro aumento no Recall, sendo menos confundida com a classe ***Bumps*** em comparação a máquina individual

(*i.e.*, sem ser agregada via Bagging). Contudo, a classe **Dirtness** sofreu uma redução no Recall, com um aumento na confusão com **Pastry**. As demais classes não exibiram mudanças significativas. Para o modelo DT, houve uma redução expressiva na confusão geral, o que resultou em um aumento no Recall das classes anteriormente mais confundidas (**Pastry** e **Bumps**). No entanto, estas ainda permaneceram como as classes mais frequentemente confundidas entre si. Em relação ao SVM, não foram detectadas melhorias significativas, com o maior aumento de Recall observado na classe **Pastry**, subindo aproximadamente 2%.

Ao aplicar o balanceamento por *undersampling* aleatório, verificou-se que para o KNN a classe **Pastry** experimentou um aumento considerável de Recall, enquanto as outras classes, especialmente **Bumps**, sofreram uma queda acentuada (cerca de 20%), sendo mais confundidas com **Pastry** e **Z-Scratch**. Após a redução do número de amostras, embora a confusão de **Pastry** com **Bumps** tenha diminuído, aumentou-se a confusão com **Dirtness**. Para o DT, observou-se um aumento significativo no Recall de **Pastry**, enquanto as demais classes mantiveram variações mínimas, principalmente devido à redução da confusão com **Bumps**. No caso do SVM, **Pastry** e **Dirtness** tiveram aumentos notáveis no Recall, enquanto **Bumps** apresentou uma queda significativa. Além disso, **Pastry** e **Dirtness** reduziram sua confusão com **Bumps**, ao passo que **Bumps** aumentou sua confusão com **Pastry**.

Finalmente, ao utilizar a estratégia de Bagging com balanceamento via SMOTE, observou-se que para o KNN a classe **Pastry** teve um aumento substancial de Recall, enquanto **Bumps** apresentou uma leve diminuição. No modelo DT, houve uma redução da confusão entre todas as classes, destacando-se as classes **Pastry** e **Bumps**. Para o SVM, verificou-se que todas as classes, exceto **Bumps**, apresentaram aumento no Recall, com a classe **Pastry** mostrando o maior incremento, enquanto **Bumps** sofreu uma redução considerável.

## 4 Conclusões

Ao longo do desenvolvimento deste projeto, pode-se verificar como os diferentes algoritmos de aprendizagem de máquina estudados na disciplina podem ser aplicados para resolver o mesmo problema, inclusive 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. Além disso, pode-se verificar na prática como o impacto do balanceamento de dados em conjunto com a combinação de diferentes classificadores afetam nos resultados quando comparados ao estudo de máquinas não-combinadas.

Analizando os modelos individualmente, verificou-se que o SVM destacou-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 **Pastry** 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.

Antes de iniciar o estudo de combinação de máquinas, foi realizada uma validação preliminar para avaliar a viabilidade temporal do treinamento de Bagging dos diferentes algoritmos. Após os testes, constatou-se que os Baggings de Regressão Logística e MLP apresentavam tempos de treinamento consideravelmente elevados. Com isto, decidiu-se excluí-los da análise e focar apenas nos Ensembles da máquinas que combinavam KNN, Árvore de Decisão e SVM.

Ao analisar a combinação de modelos via Bagging, observou-se que, dentro de uma mesma estratégia, os Baggings de Árvore de Decisão apresentaram as maiores Acurárias médias. Os resultados dos Baggings de SVM foram comparáveis aos de Árvore de Decisão, mas os Baggings de KNN tiveram desempenho inferior, especialmente com a estratégia de *undersampling*. Em termos de estratégias, o Ensemble de Árvore de Decisão com *oversampling* obteve o melhor desempenho geral, enquanto os de KNN e SVM apresentaram melhores resultados sem o balanceamento dos dados. Analisando pela ótica de F1-score, verificou-se

que as Árvores de Decisão com oversampling resultaram em F1-scores mais consistente entre as classes, tendo também o maior valor médio. Já os Ensembles de KNN e SVM tiveram melhor desempenho quando os dados não foram balanceados, mas ainda assim apresentaram valores médios menores que Bagging de Árvores de Decisão. Já com relação às matrizes de confusão, ao utilizar Bagging sem balanceamento, o KNN apresentou um aumento no Recall para a classe ***Pastry***, enquanto ***Dirtiness*** diminuiu. O Bagging de Árvore de Decisão reduziu a confusão geral, especialmente para ***Pastry*** e ***Bumps***, enquanto o SVM teve melhorias modestas. Com *undersampling*, o KNN teve aumento de Recall para ***Pastry***, mas queda para ***Bumps***, enquanto que a Árvore de Decisão e SVM mostraram ganhos em ***Pastry*** e ***Dirtiness***. Com *oversampling* via SMOTE, o KNN aumentou o Recall para ***Pastry***, a Árvore de Decisão reduziu a confusão entre todas as classes, e o SVM melhorou o Recall para quase todas as classes, exceto ***Bumps***.

Computadas as métricas dos Ensembles, fez-se uso de Testes de Hipótese para determinar se a utilização das estratégias de Bagging impactaram na melhoria da acurácia e F1-score em comparação com uma única máquina. Adotando-se um nível de  $\alpha = 5\%$ , observou-se que o DT apresentou evidências estatísticas de melhoria de Acurácia e F1-score, enquanto que SVM apresentou melhoria apenas nos resultados de F1-score. Para os Ensembles de KNN, não foi possível verificar aprimoramento a nível estatístico em nenhuma destas duas métricas.

É importante ressaltar que para indicar algum algoritmo (ou combinação destes) como mais adequado, é necessário ter um conhecimento mais aprofundado do contexto do problema, sendo 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) para a escolha dos melhores hiperparâmetros dos modelos de Bagging, avaliar o número de estimadores em conjunto com os demais hiperparâmetros das máquinas; (iii) utilizar outros modelos que também sejam adequados ao problema; e (iv) aplicar técnicas de Feature Engineering para gerar variáveis que possam capturar informações mais relevantes.

## 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](http://berrar.com/resources/Berrar_EBCB_2nd_edition_Cross-validation_preprint.pdf)>.
- BREIMAN, L. Bagging predictors. *Machine learning*, Springer, v. 24, p. 123–140, 1996.
- 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>>.
- CHAWLA, N. V. et al. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, AI Access Foundation, v. 16, p. 321–357, jun. 2002. ISSN 1076-9757. Disponível em: <<http://dx.doi.org/10.1613/jair.953>>.
- D'AGOSTINO, R.; PEARSON, E. S. Tests for departure from normality. empirical results for the distributions of  $b_2$  and  $b$ . *Biometrika*, Oxford University Press, v. 60, n. 3, p. 613–622, 1973.
- FAY, M. P.; PROSCHAN, M. A. Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics surveys*, NIH Public Access, v. 4, p. 1, 2010.
- HE, H.; GARCIA, E. A. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, v. 21, n. 9, p. 1263–1284, 2009.
- 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 <sup>1</sup>	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> <sup>2</sup>	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 12: P-valores dos testes de normalidade aplicados nas variáveis preditoras.

---

<sup>1</sup>As variáveis categóricas foram removidas da análise.

<sup>2</sup>Apesar 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**

# Máquinas individuais

## 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 13: 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 14: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para o DT.

## Regressão Logística (LR)

Ocorrências	Hiperparâmetros <sup>1</sup>			
	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 15: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para a LR.

## Support Vector Machine (SVM)

Ocorrências	Hiperparâmetros <sup>2</sup>			
	Força de penalização	Grau do kernel polinomial	Alcance da influência de treinamento	Kernel
9	10	Não utilizado	Escalonado <sup>2</sup>	RBF
3	100	Não utilizado	0,01	RBF
2	10	3	Escalonado	Polynomial
2	10	Não utilizado	0,01	RBF
2	10	Não utilizado	0,1	RBF
1	1	3	0,1	Polynomial
1	1	Não utilizado	0,1	RBF

Tabela 16: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para o SVM.

<sup>1</sup>Durante 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.

<sup>2</sup>Durante a aplicação do Grid Search, filtrou-se as combinações inválidas, garantindo que apenas combi-

## Multi Layer Perceptron (MLP)

Ocorrências	Hiperparâmetros				
	Função de ativação	Força de regularização	Número de neurônios na camada oculta	Taxa de aprendizagem	Algoritmo
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 17: Ocorrências dos melhores conjuntos de hiperparâmetros obtidos para o MLP.

## Máquinas agregadas via Bagging

### K-Nearest Neighbors (KNN)

Número de estimadores	Ocorrências		
	Bagging desbalanceado	Bagging balanceado via <i>undersampling</i> aleatório	Bagging balanceado via SMOTE
10	5	3	1
50	4	2	3
100	8	9	6
200	3	6	10

Tabela 18: Ocorrências do melhor número de estimadores para o Bagging de KNN.

nações viáveis fossem consideradas.

## Árvore de Decisão (DT)

Número de estimadores	Ocorrências		
	Bagging desbalanceado	Bagging balanceado via <i>undersampling</i> aleatório	Bagging balanceado via SMOTE
10	0	0	0
50	5	2	2
100	6	7	8
200	9	11	10

Tabela 19: Ocorrências do melhor número de estimadores para o Bagging de DT.

## Support Vector Machine (SVM)

Número de estimadores	Ocorrências		
	Bagging desbalanceado	Bagging balanceado via <i>undersampling</i> aleatório	Bagging balanceado via SMOTE
10	2	0	2
50	8	5	8
100	4	9	6
200	6	6	4

Tabela 20: Ocorrências do melhor número de estimadores para o Bagging de SVM.

# Apêndice C

Gráficos das métricas geradas para os modelos

# Máquinas individuais

## 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

Figura 4: Matrizes de confusão para o KNN.

Distribuição das Acurárias após 20 iterações

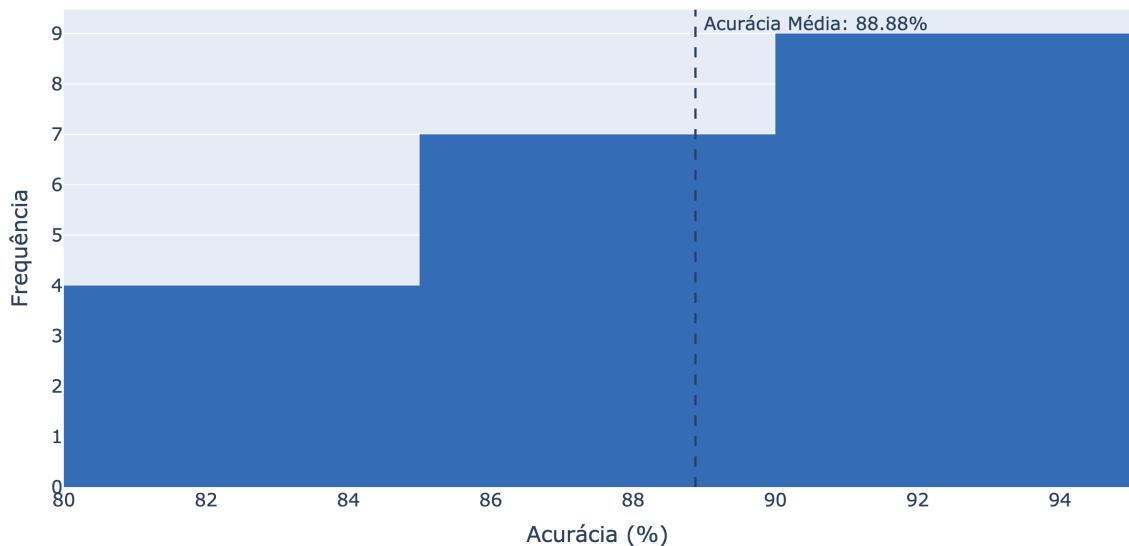


Figura 5: Distribuição das Acurárias para o KNN.

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

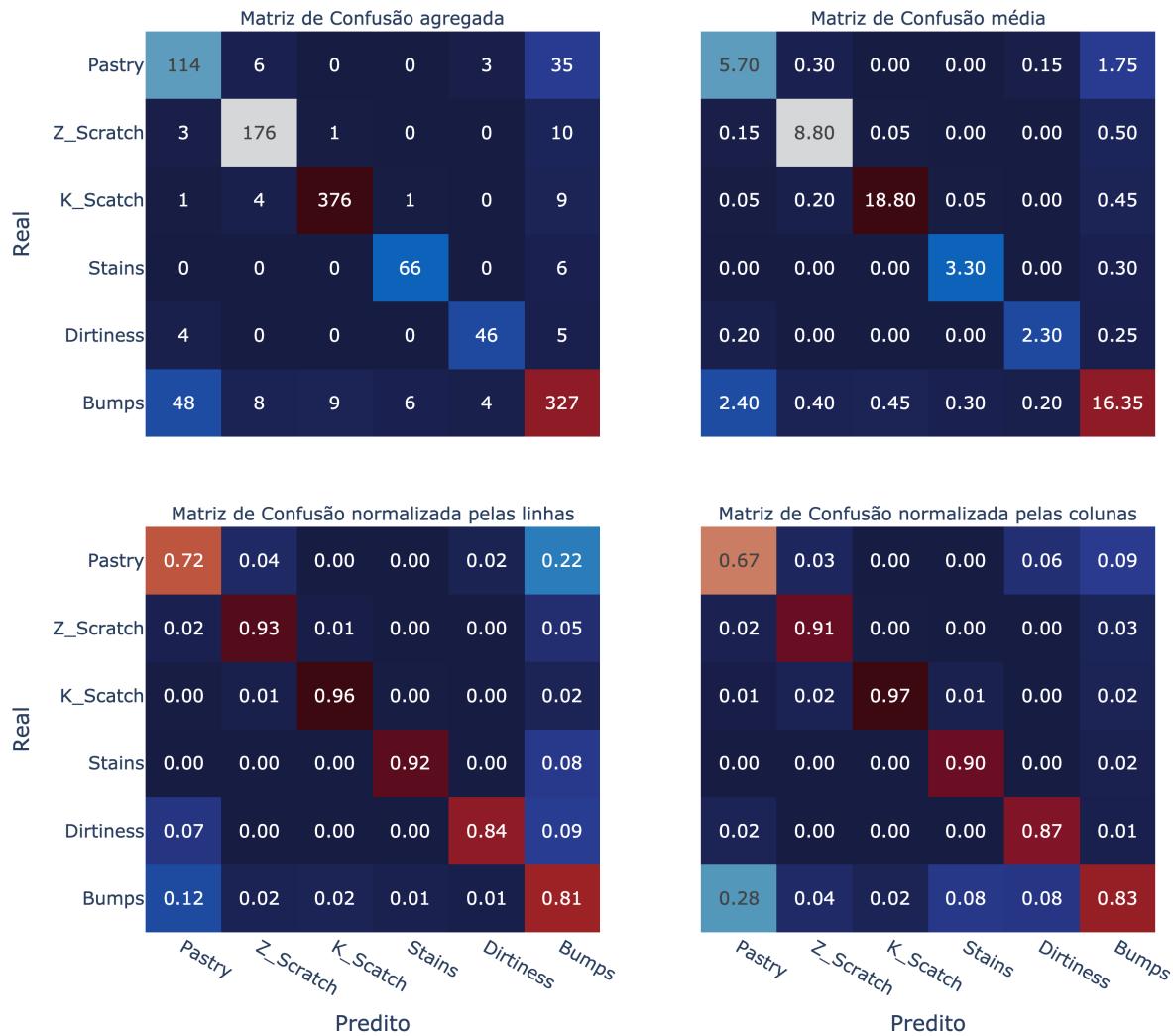


Figura 7: Matrizes de confusão para o DT.

Distribuição das Acurárias após 20 iterações

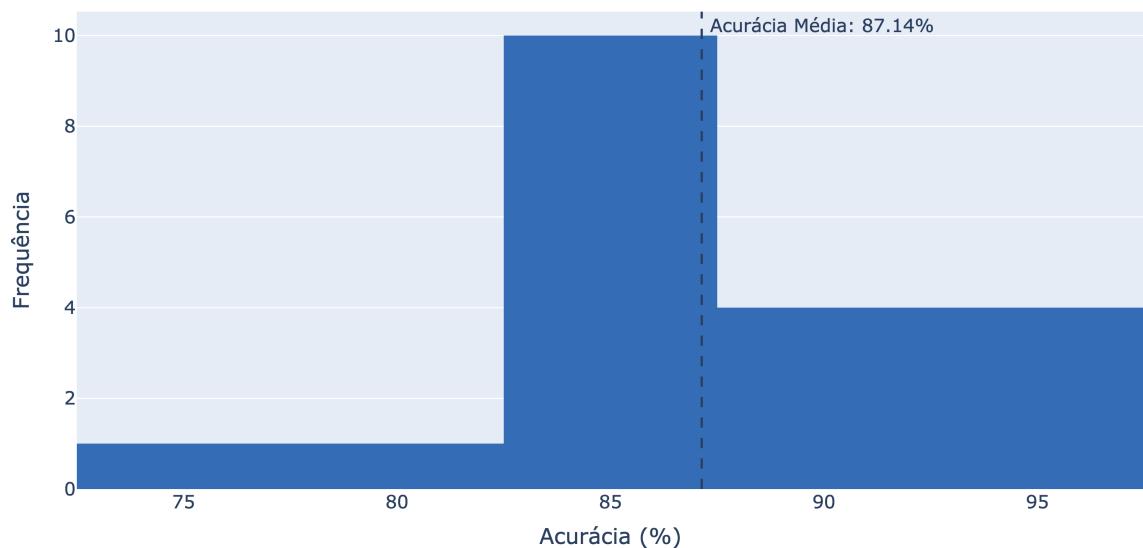


Figura 8: Distribuição das Acurárias para o DT.

Boxplots de F1-Scores por label após 20 iterações

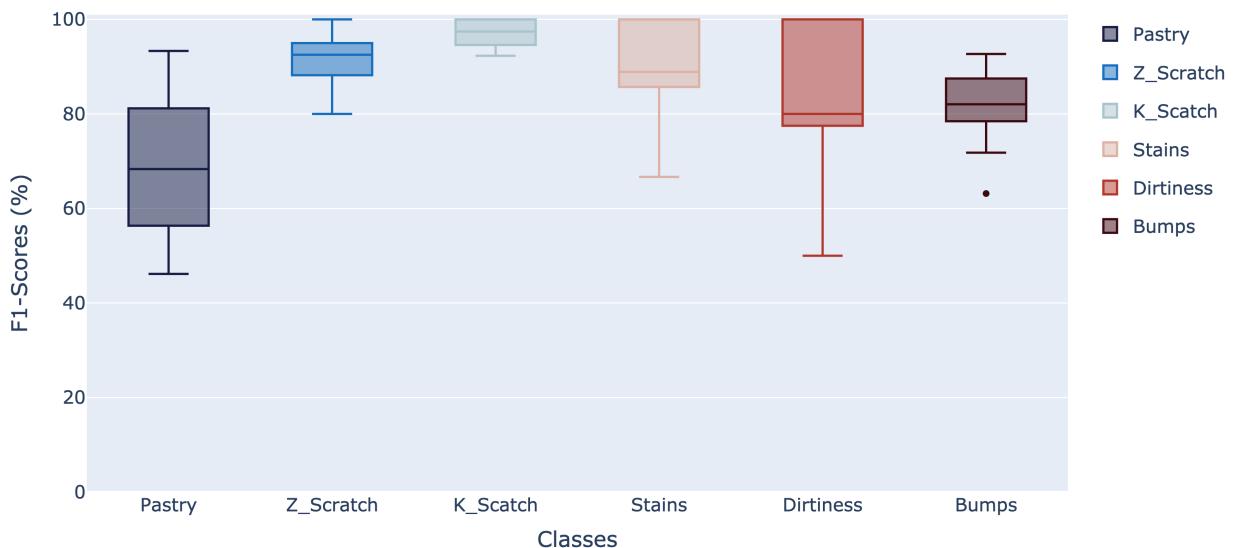


Figura 9: Box-plots dos F1-scores para o DT.

## 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
			Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps	Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps

Figura 10: Matrizes de confusão para a LR.

Distribuição das Acurárias após 20 iterações

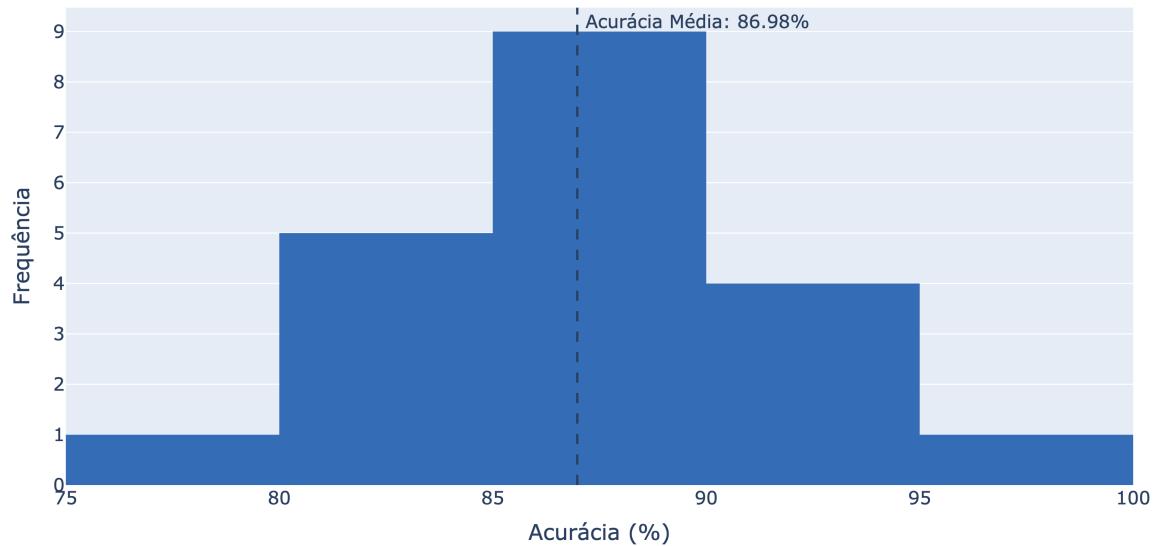


Figura 11: Distribuição das Acurárias para a LR.

Boxplots de F1-Scores por label após 20 iterações

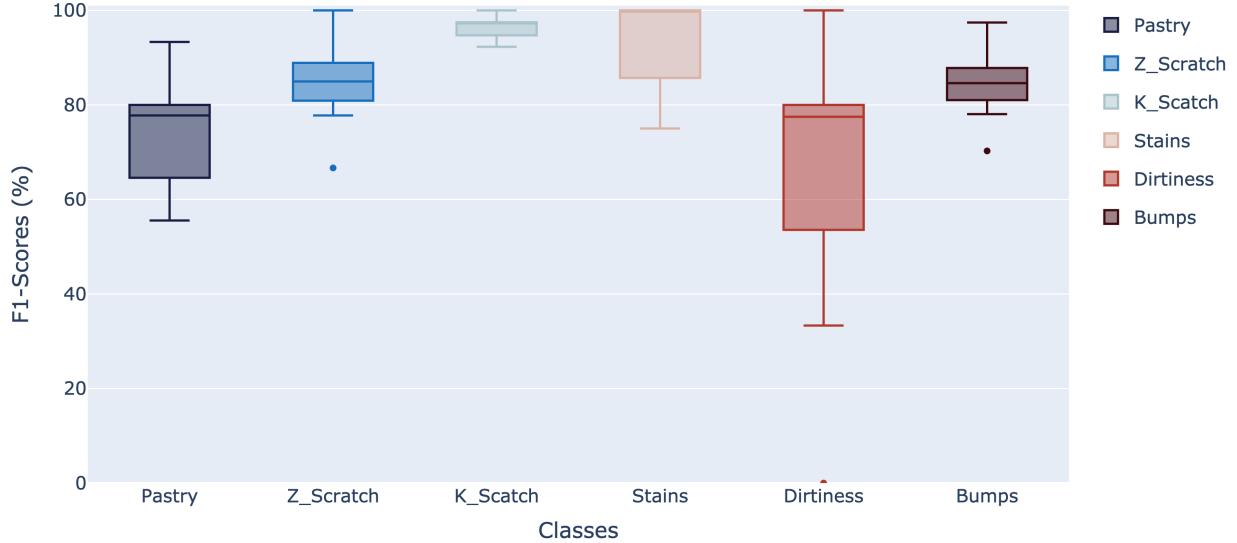


Figura 12: Box-plots dos F1-scores para a LR.

## 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 13: Matrizes de confusão para o SVM.

Distribuição das Acurárias após 20 iterações

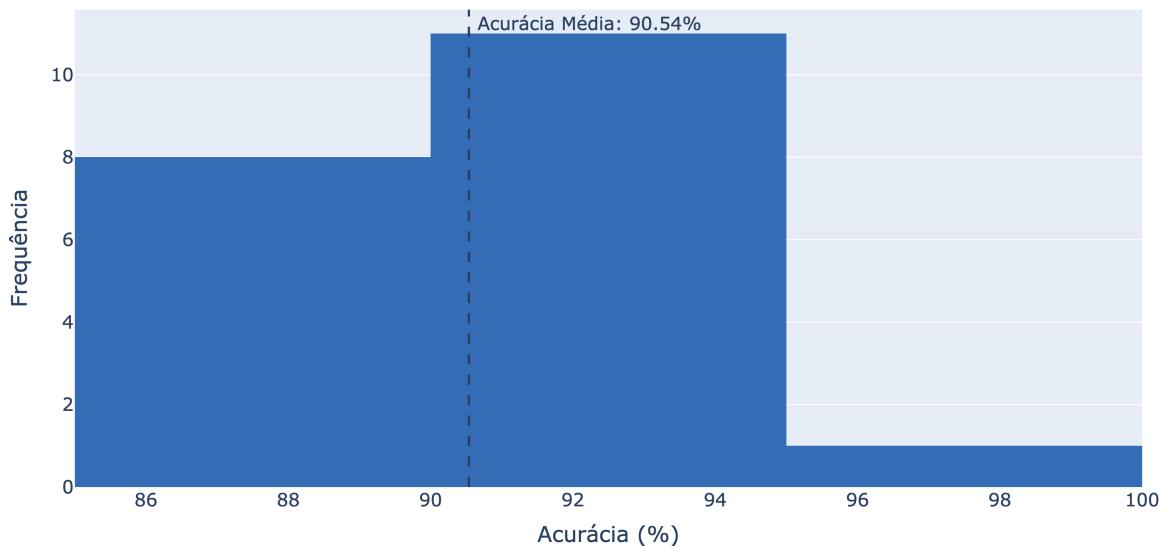


Figura 14: Distribuição das Acurárias para o SVM.

Boxplots de F1-Scores por label após 20 iterações

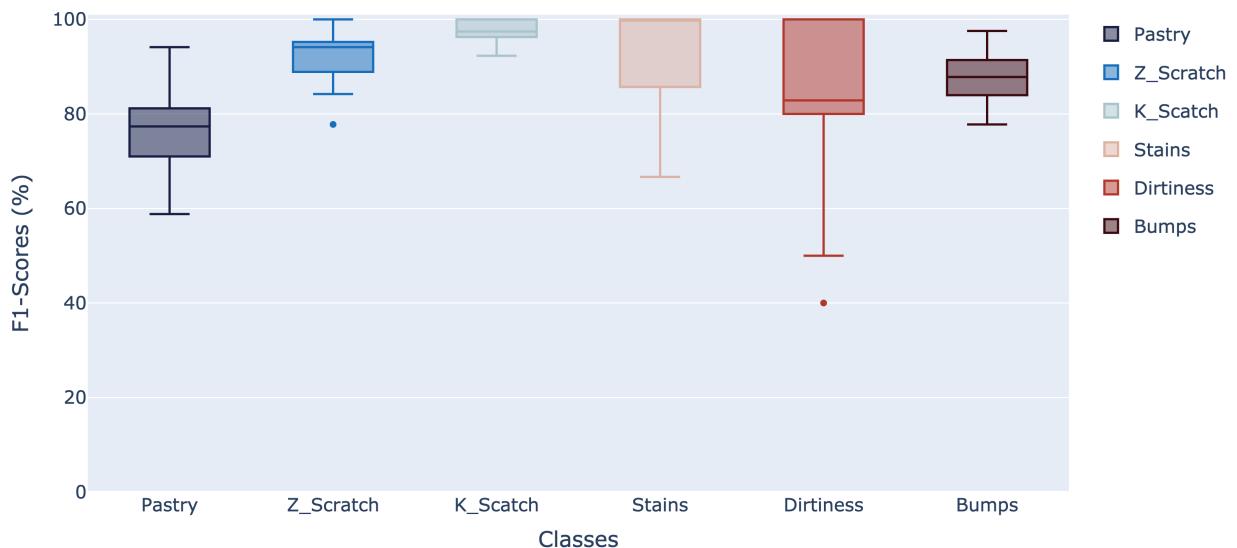


Figura 15: Box-plots dos F1-scores para o SVM.

## 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
			Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps	Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps

Figura 16: Matrizes de confusão para o MLP.

Distribuição das Acurárias após 20 iterações

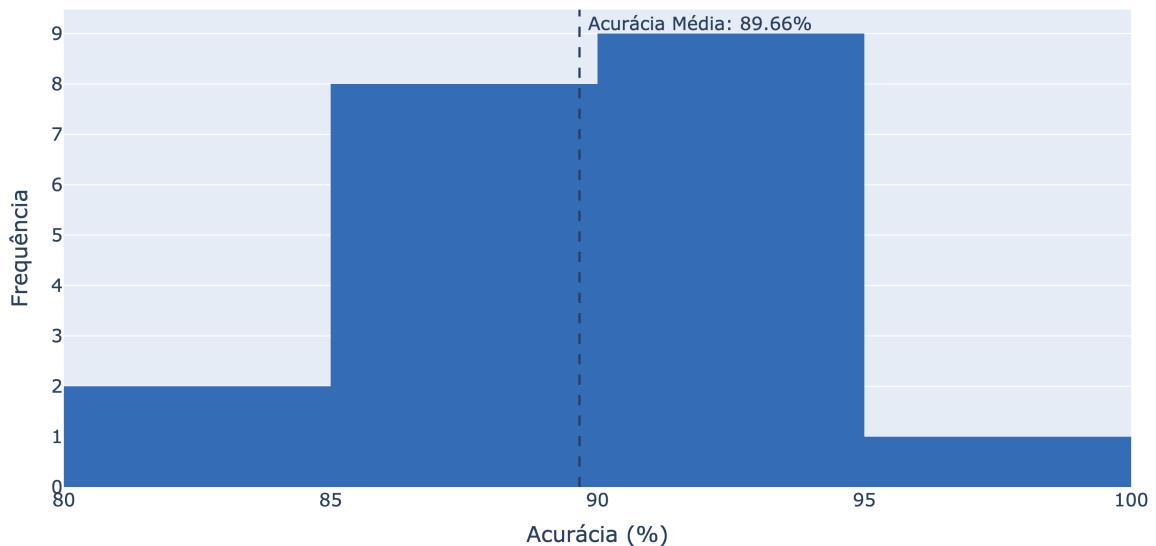


Figura 17: Distribuição das Acurárias para o MLP.

Boxplots de F1-Scores por label após 20 iterações

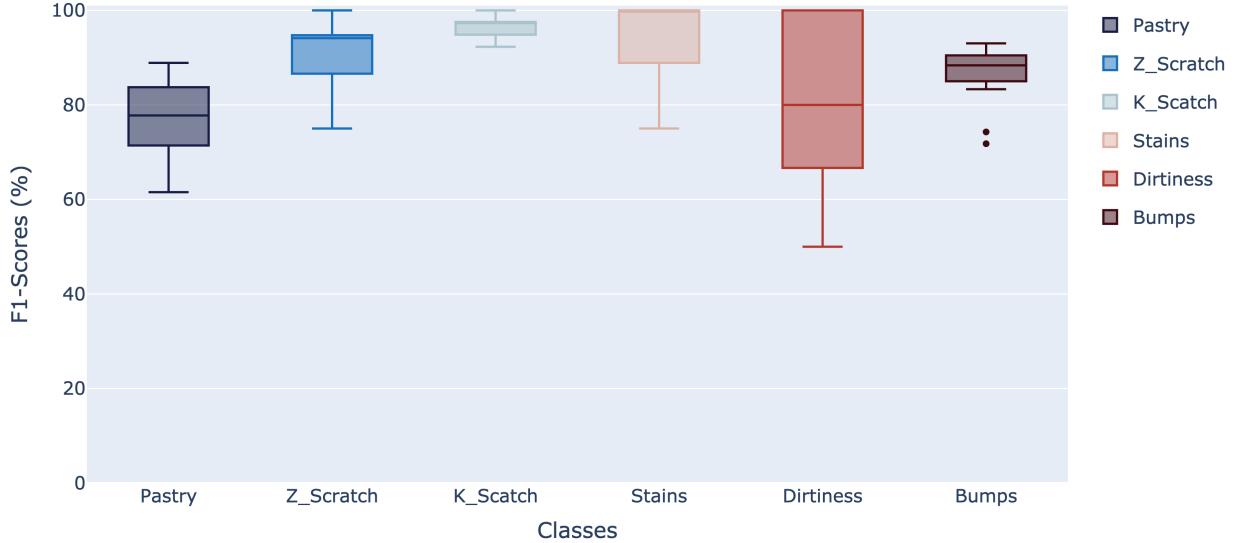


Figura 18: Box-plots dos F1-scores para o MLP.

# Máquinas agregadas via Bagging

## Bagging Desbalanceado de K-Nearest Neighbors (KNN)

Matrizes de Confusão geradas após 20 iterações

		Matriz de Confusão agregada						Matriz de Confusão média						
Real	Predito	Pastry	107	5	0	0	1	45	5.35	0.25	0.00	0.00	0.05	2.25
		Z_Scratch	3	177	4	0	0	6	0.15	8.85	0.20	0.00	0.00	0.30
		K_Scratch	2	2	381	1	2	3	0.10	0.10	19.05	0.05	0.10	0.15
		Stains	0	1	0	68	0	3	0.00	0.05	0.00	3.40	0.00	0.15
		Dirtiness	5	1	0	0	49	0	0.25	0.05	0.00	0.00	2.45	0.00
		Bumps	27	12	4	3	5	351	1.35	0.60	0.20	0.15	0.25	17.55

		Matriz de Confusão normalizada pelas linhas						Matriz de Confusão normalizada pelas colunas						
Real	Predito	Pastry	0.68	0.03	0.00	0.00	0.01	0.28	0.74	0.03	0.00	0.00	0.02	0.11
		Z_Scratch	0.02	0.93	0.02	0.00	0.00	0.03	0.02	0.89	0.01	0.00	0.00	0.01
		K_Scratch	0.01	0.01	0.97	0.00	0.01	0.01	0.01	0.01	0.98	0.01	0.04	0.01
		Stains	0.00	0.01	0.00	0.94	0.00	0.04	0.00	0.01	0.00	0.94	0.00	0.01
		Dirtiness	0.09	0.02	0.00	0.00	0.89	0.00	0.03	0.01	0.00	0.00	0.86	0.00
		Bumps	0.07	0.03	0.01	0.01	0.01	0.87	0.19	0.06	0.01	0.04	0.09	0.86

Figura 19: Matrizes de confusão para o Bagging de KNN Desbalanceado.

Distribuição das Acurárias após 20 iterações

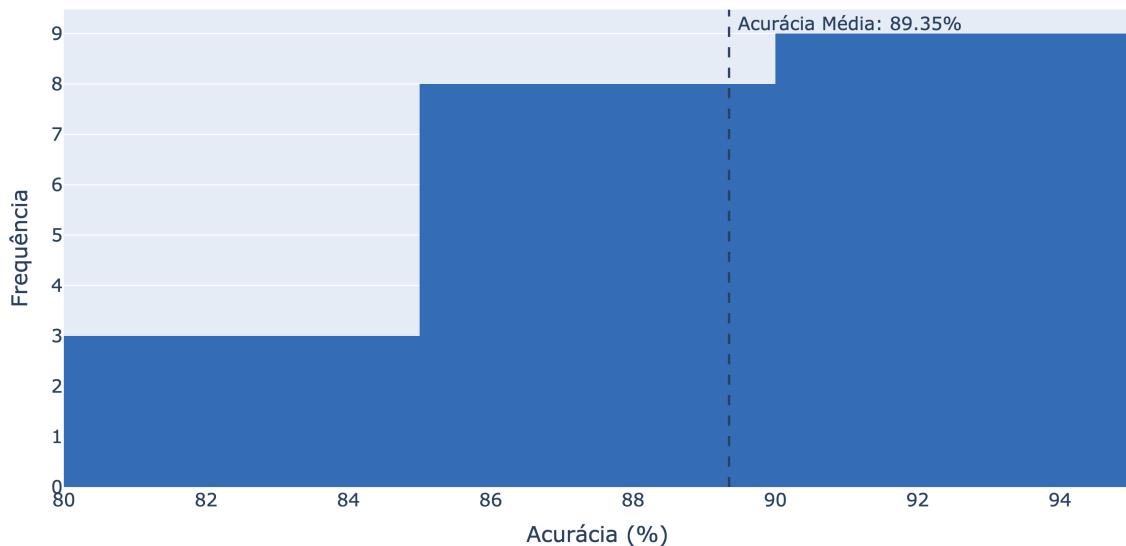


Figura 20: Distribuição das Acurárias para o Bagging de KNN Desbalanceado.

Boxplots de F1-Scores por label após 20 iterações

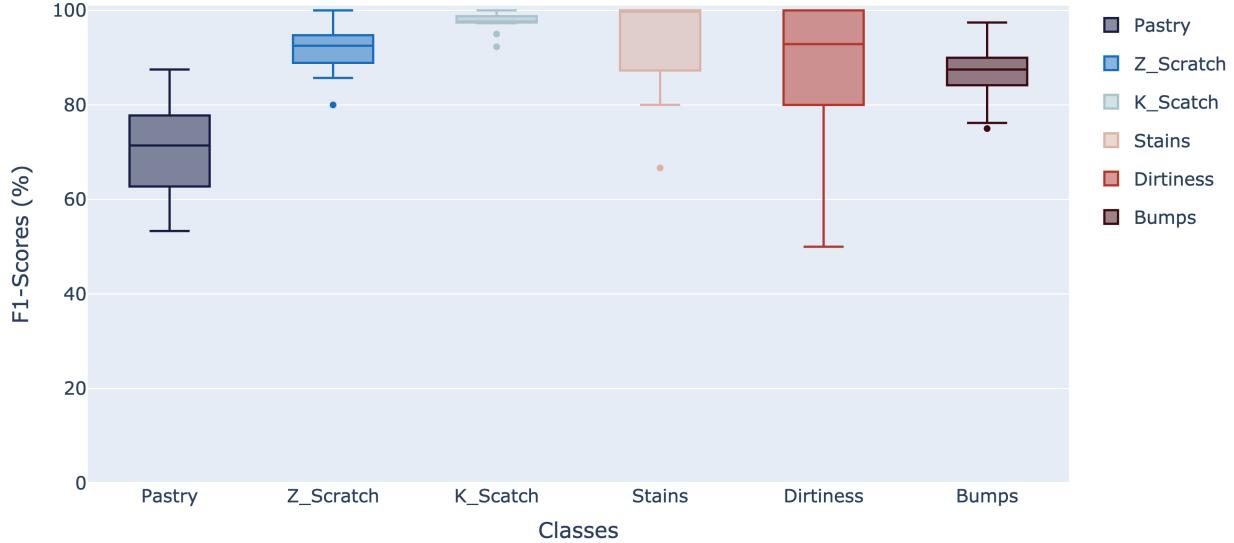


Figura 21: Box-plots dos F1-scores para o Bagging de KNN Desbalanceado.

## Bagging Desbalanceado de Árvore de Decisão (DT)

Matrizes de Confusão geradas após 20 iterações

		Matriz de Confusão agregada						Matriz de Confusão média							
Real	Predito	Pastry	126	1	0	0	1	30	6.30	0.05	0.00	0.00	0.05	1.50	
		Z_Scratch	2	179	1	0	0	0	8	0.10	8.95	0.05	0.00	0.00	0.40
		K_Scratch	4	2	381	0	0	0	4	0.20	0.10	19.05	0.00	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	4	0	0	0	47	0	4	0.20	0.00	0.00	0.00	2.35	0.20
		Bumps	27	4	5	2	2	362		1.35	0.20	0.25	0.10	0.10	18.10

		Matriz de Confusão normalizada pelas linhas						Matriz de Confusão normalizada pelas colunas							
Real	Predito	Pastry	0.80	0.01	0.00	0.00	0.01	0.19	0.77	0.01	0.00	0.00	0.02	0.07	
		Z_Scratch	0.01	0.94	0.01	0.00	0.00	0.04	0.01	0.96	0.00	0.00	0.00	0.00	0.02
		K_Scratch	0.01	0.01	0.97	0.00	0.00	0.01	0.02	0.01	0.98	0.00	0.00	0.00	0.01
		Stains	0.00	0.00	0.00	0.93	0.00	0.07	0.00	0.00	0.00	0.97	0.00	0.00	0.01
		Dirtiness	0.07	0.00	0.00	0.00	0.85	0.07	0.02	0.00	0.00	0.00	0.94	0.00	0.01
		Bumps	0.07	0.01	0.01	0.00	0.00	0.90	0.17	0.02	0.01	0.03	0.04	0.00	0.88

Figura 22: Matrizes de confusão para o Bagging de DT Desbalanceado.

Distribuição das Acurárias após 20 iterações

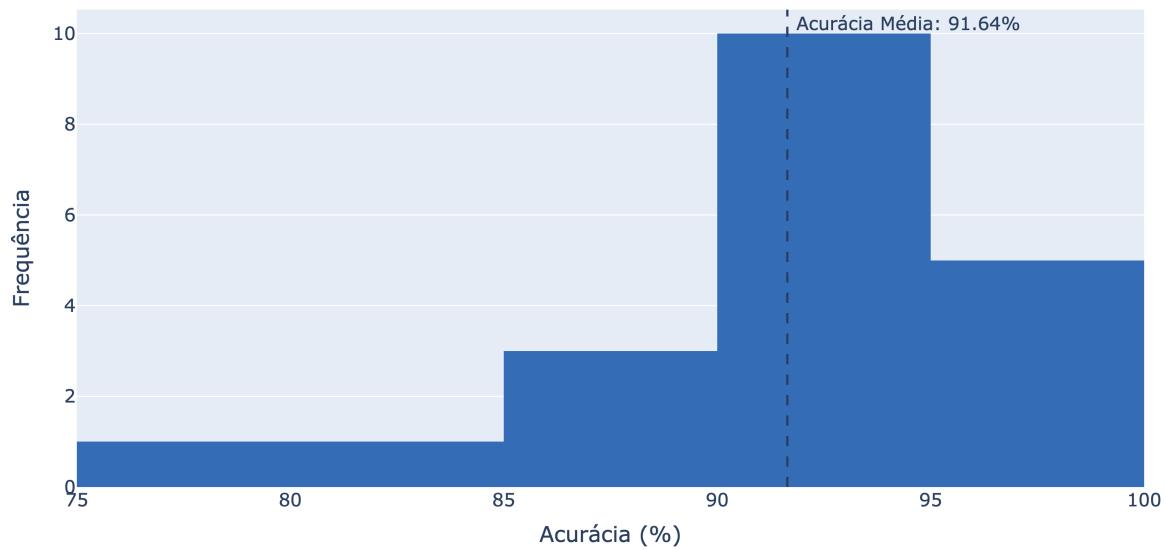


Figura 23: Distribuição das Acurárias para o Bagging de DT Desbalanceado.

Boxplots de F1-Scores por label após 20 iterações

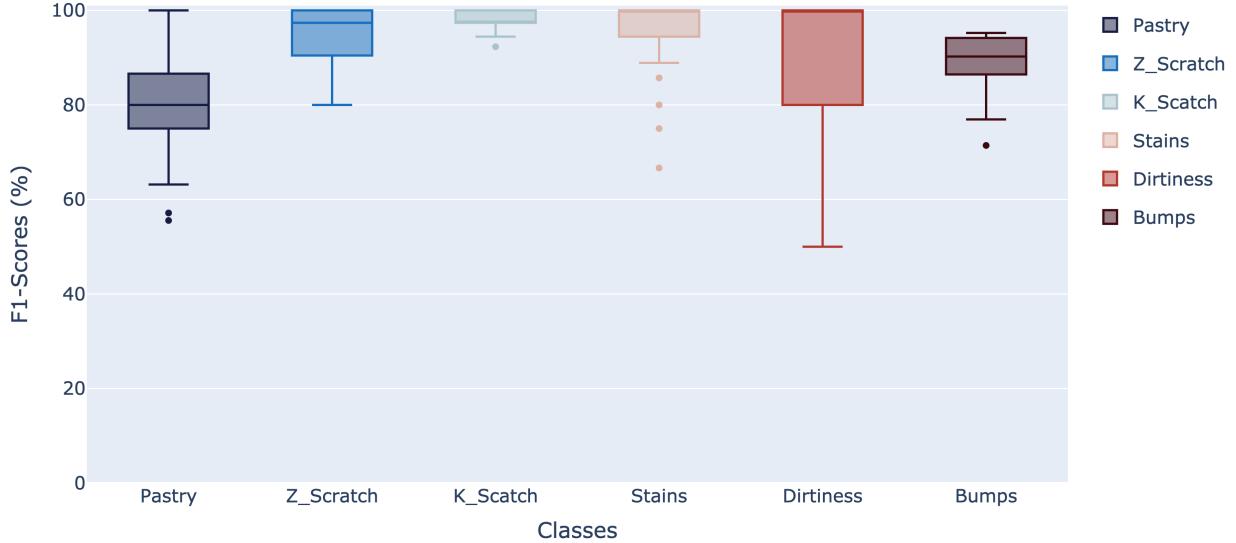


Figura 24: Box-plots dos F1-scores para o Bagging de DT Desbalanceado.

## Bagging Desbalanceado de Support Vector Machine (SVM)

Matrizes de Confusão geradas após 20 iterações

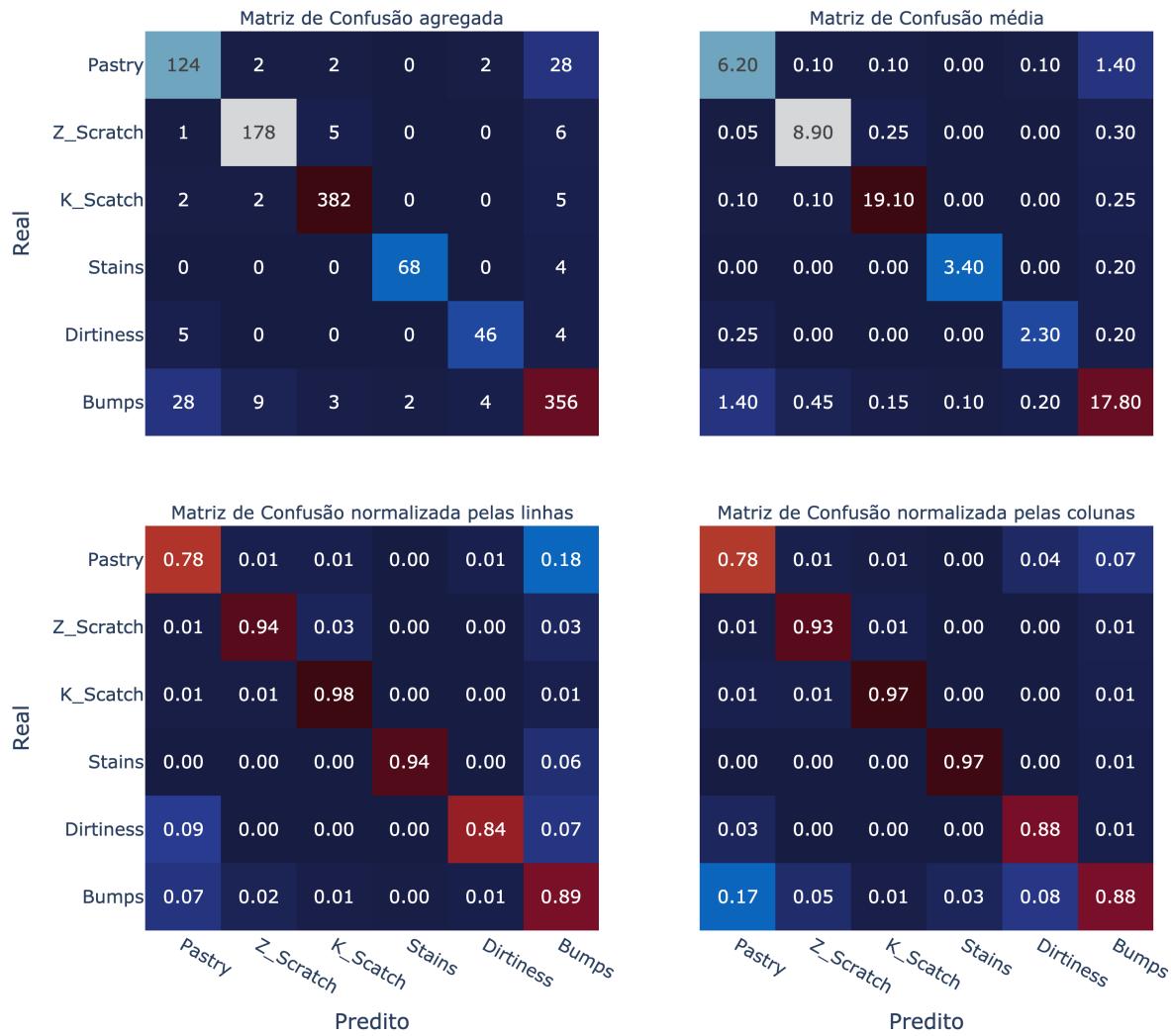


Figura 25: Matrizes de confusão para o Bagging de SVM Desbalanceado.

Distribuição das Acurárias após 20 iterações

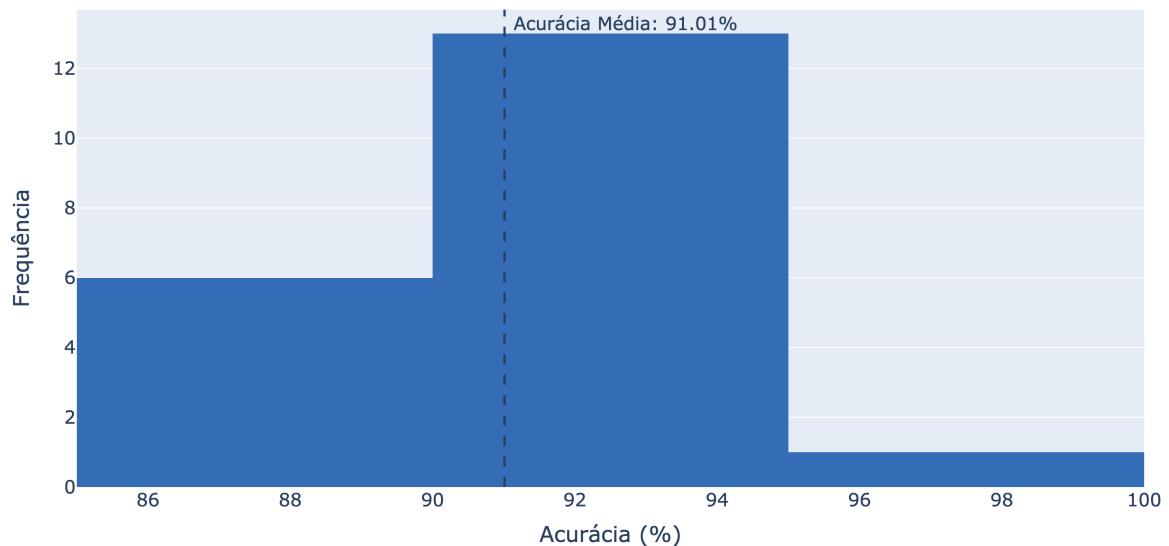


Figura 26: Distribuição das Acurárias para o Bagging de SVM Desbalanceado.

Boxplots de F1-Scores por label após 20 iterações

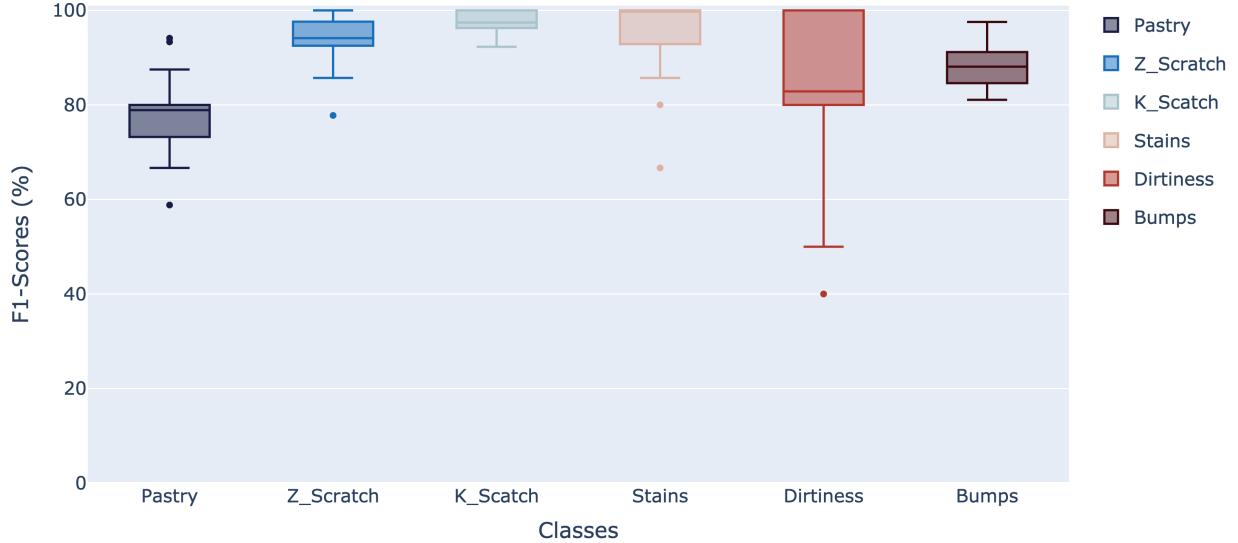


Figura 27: Box-plots dos F1-scores para o Bagging de SVM Desbalanceado.

## Bagging Balanceado via *Undersampling* de K-Nearest Neighbors (KNN)

Matrizes de Confusão geradas após 20 iterações

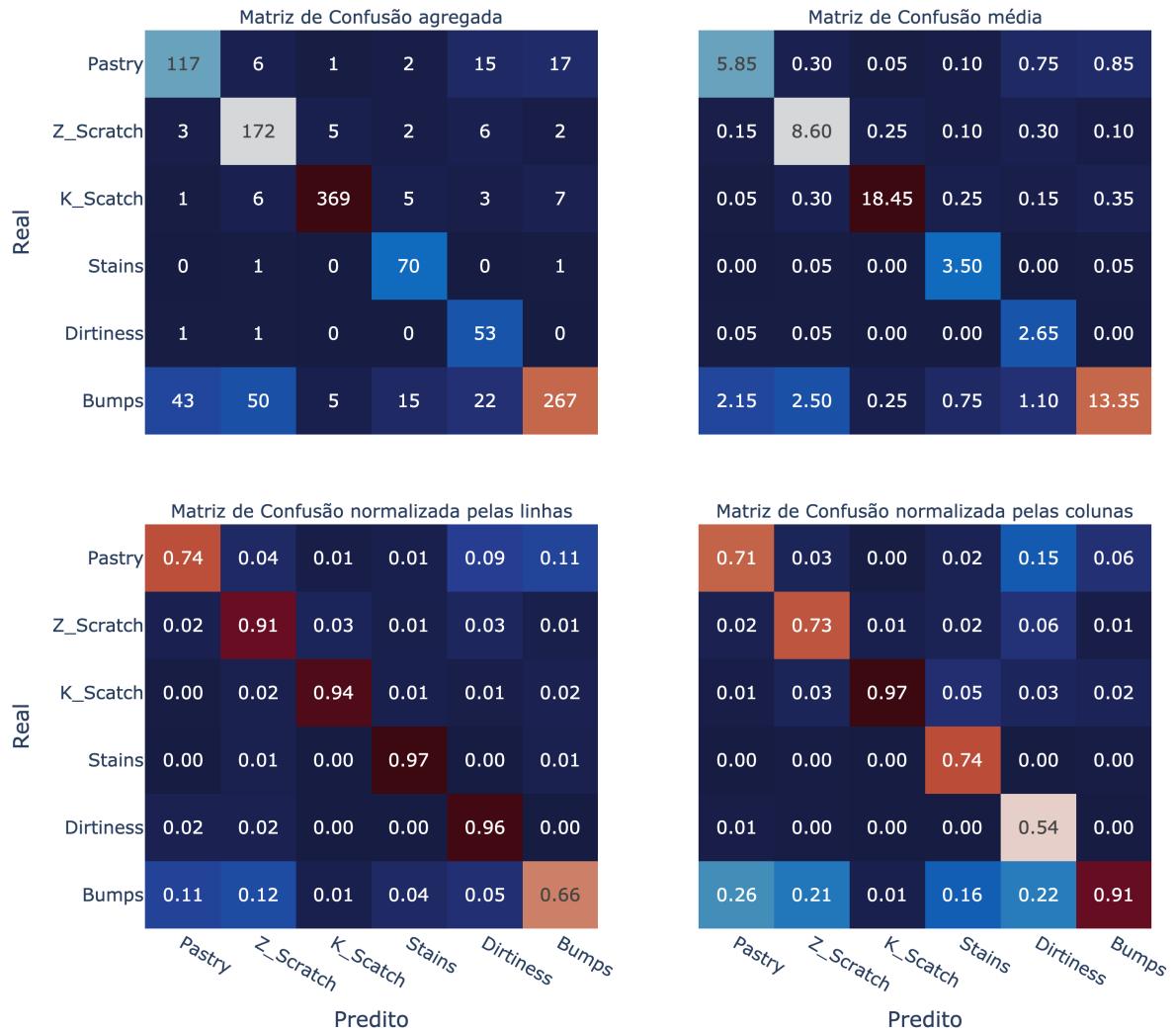


Figura 28: Matrizes de confusão para o Bagging de KNN Balanceado via *Undersampling*.

Distribuição das Acurárias após 20 iterações

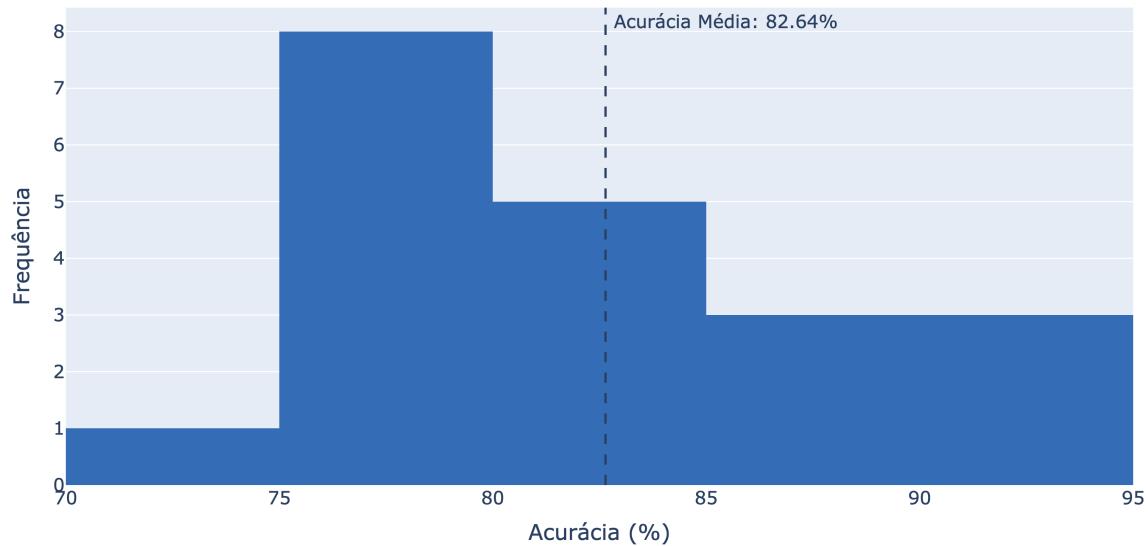


Figura 29: Distribuição das Acurárias para o Bagging de KNN Balanceado via *Undersampling*.

Boxplots de F1-Scores por label após 20 iterações

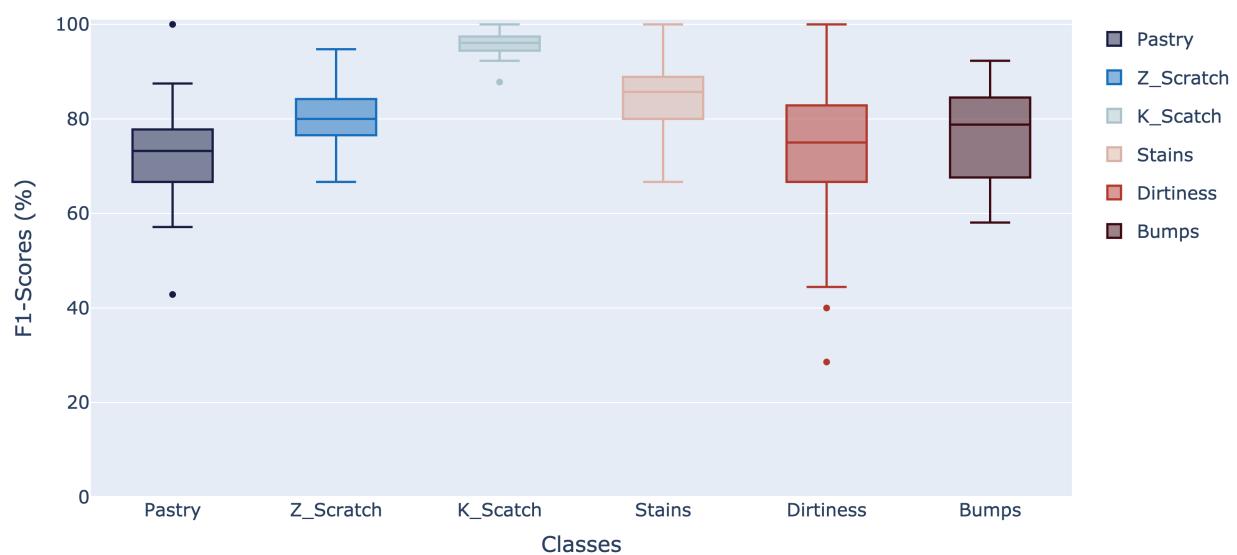


Figura 30: Box-plots dos F1-scores para o Bagging de KNN Balanceado via *Undersampling*.

## Bagging Balanceado via *Undersampling* de Árvore de Decisão (DT)

Matrizes de Confusão geradas após 20 iterações

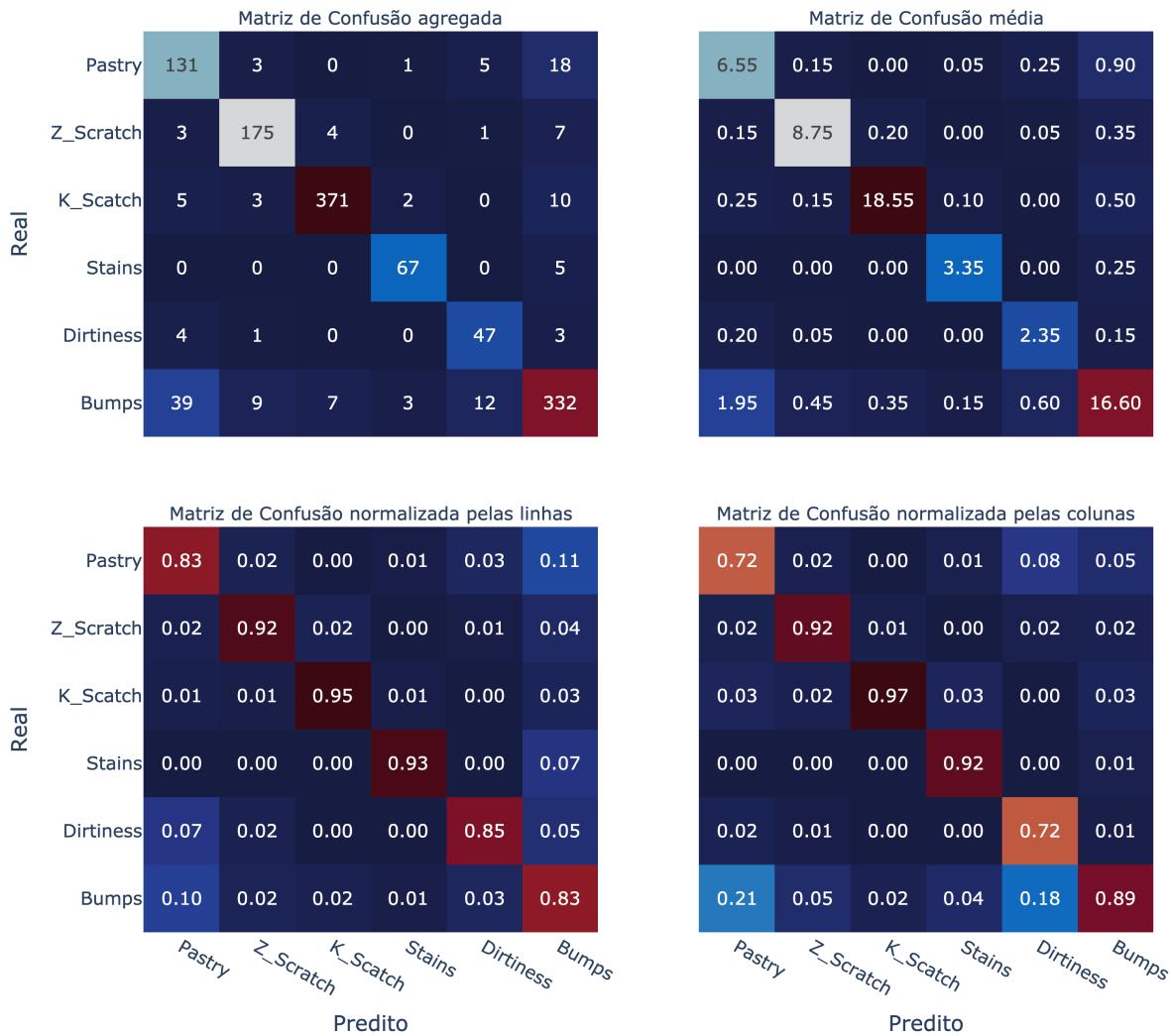


Figura 31: Matrizes de confusão para o Bagging de DT Balanceado via *Undersampling*.

Distribuição das Acurárias após 20 iterações

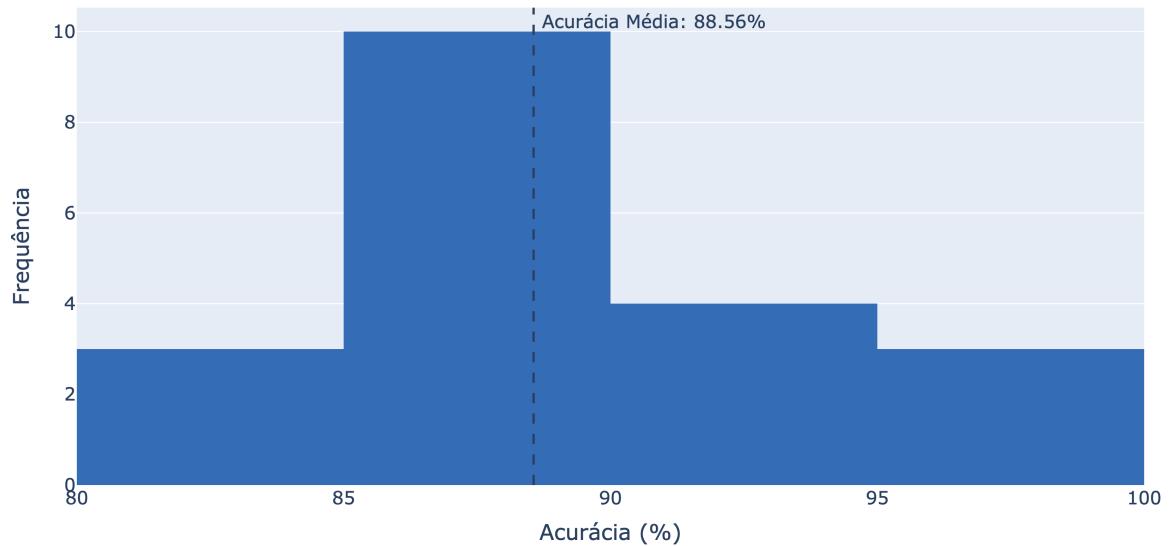


Figura 32: Distribuição das Acurárias para o Bagging de DT Balanceado via *Undersampling*.

Boxplots de F1-Scores por label após 20 iterações

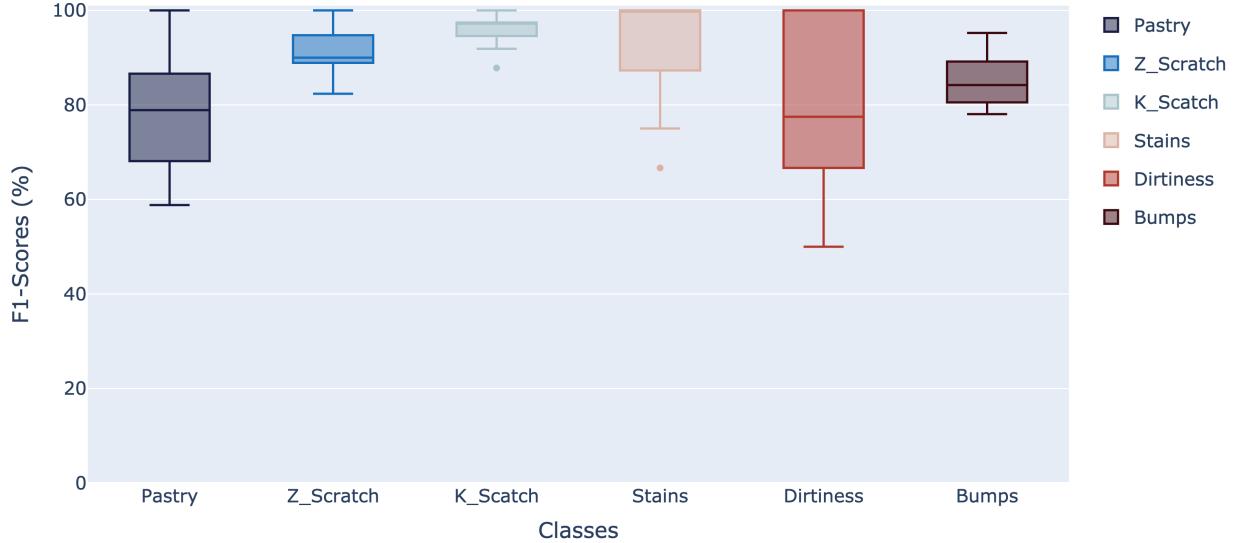


Figura 33: Box-plots dos F1-scores para o Bagging de DT Balanceado via *Undersampling*.

## Bagging Balanceado via *Undersampling* de Support Vector Machine (SVM)

Matrizes de Confusão geradas após 20 iterações

		Matriz de Confusão agregada						Matriz de Confusão média					
		Pastry	3	0	0	3	16	6.80	0.15	0.00	0.00	0.15	0.80
Real	Pastry	136	3	0	0	3	16	0.20	9.05	0.15	0.00	0.00	0.10
	Z_Scratch	4	181	3	0	0	2	0.10	0.35	18.75	0.10	0.00	0.25
	K_Scratch	2	7	375	2	0	5	0.00	0.00	0.00	3.40	0.00	0.20
	Stains	0	0	0	68	0	4	0.25	0.00	0.00	0.00	2.50	0.00
	Dirtiness	5	0	0	0	50	0	2.50	0.90	0.30	0.50	0.60	15.30
	Bumps	50	18	6	10	12	306						

		Matriz de Confusão normalizada pelas linhas						Matriz de Confusão normalizada pelas colunas						
		Pastry	0.02	0.00	0.00	0.00	0.02	0.10	0.69	0.01	0.00	0.00	0.05	0.05
Real	Pastry	0.86	0.02	0.00	0.00	0.02	0.10	0.02	0.87	0.01	0.00	0.00	0.00	0.01
	Z_Scratch	0.02	0.95	0.02	0.00	0.00	0.01	0.01	0.03	0.98	0.03	0.00	0.00	0.02
	K_Scratch	0.01	0.02	0.96	0.01	0.00	0.01	0.00	0.00	0.00	0.85	0.00	0.00	0.01
	Stains	0.00	0.00	0.00	0.94	0.00	0.06	0.03	0.00	0.00	0.00	0.77	0.00	0.00
	Dirtiness	0.09	0.00	0.00	0.00	0.91	0.00	0.25	0.09	0.02	0.13	0.18	0.92	
	Bumps	0.12	0.04	0.01	0.02	0.03	0.76							

		Predito						Predito					
		Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps	Pastry	Z_Scratch	K_Scratch	Stains	Dirtiness	Bumps

Figura 34: Matrizes de confusão para o Bagging de SVM Balanceado via *Undersampling*.

Distribuição das Acurárias após 20 iterações

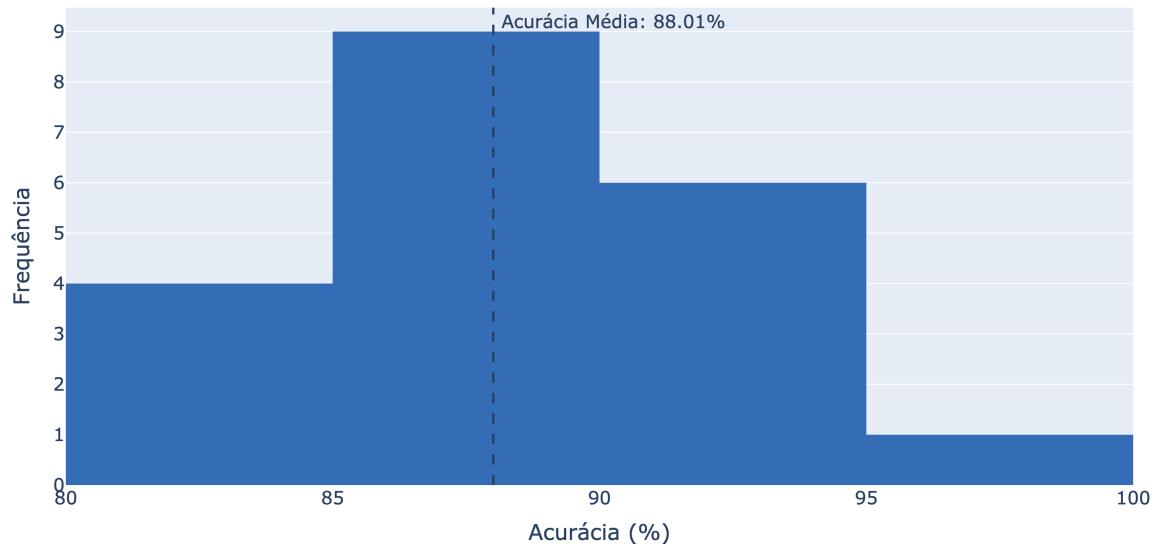


Figura 35: Distribuição das Acurárias para o Bagging de SVM Balanceado via *Undersampling*.

Boxplots de F1-Scores por label após 20 iterações

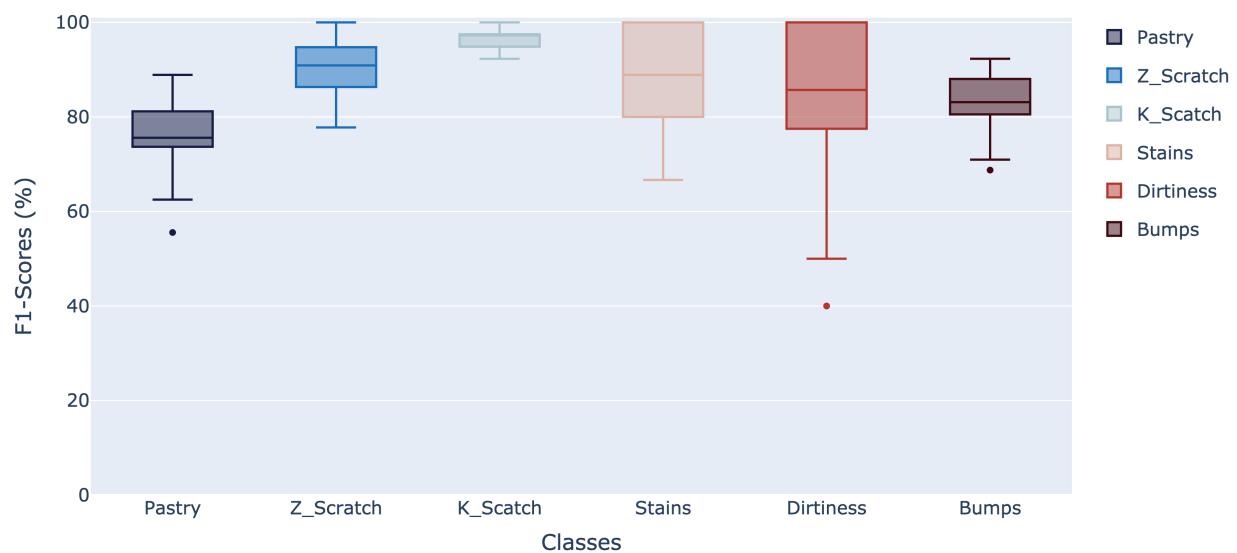


Figura 36: Box-plots dos F1-scores para o Bagging de SVM Balanceado via *Undersampling*.

## Bagging Balanceado via *Oversampling* de K-Nearest Neighbors (KNN)

Matrizes de Confusão geradas após 20 iterações

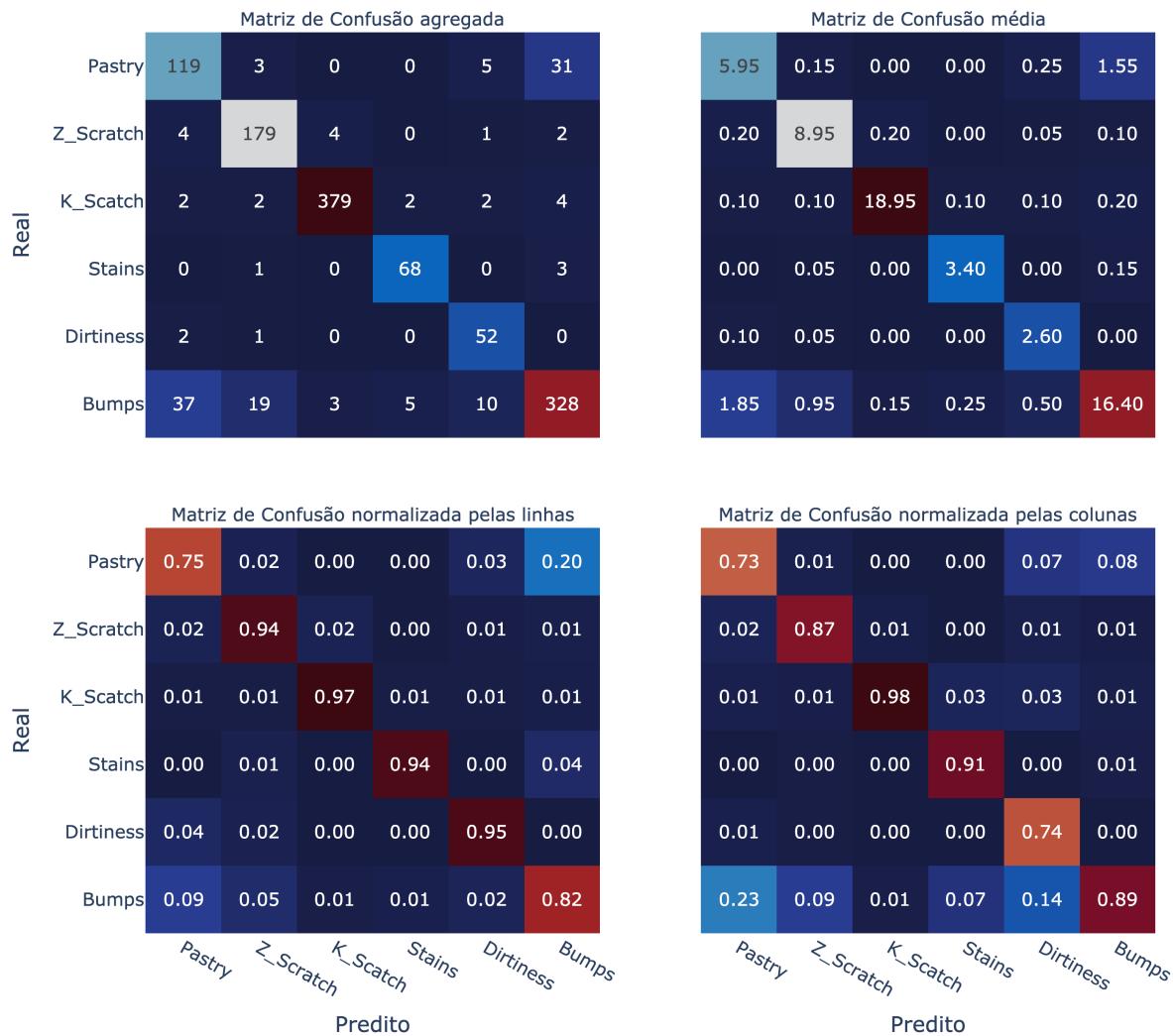


Figura 37: Matrizes de confusão para o Bagging de KNN Balanceado via *Oversampling*.

Distribuição das Acurárias após 20 iterações

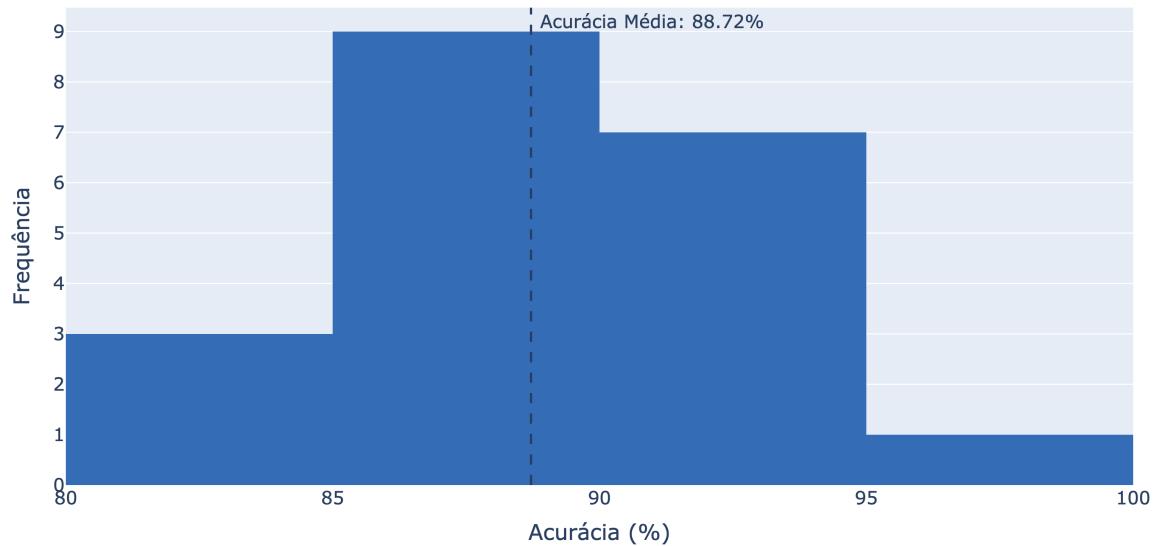


Figura 38: Distribuição das Acurárias para o Bagging de KNN Balanceado via *Oversampling*.

Boxplots de F1-Scores por label após 20 iterações

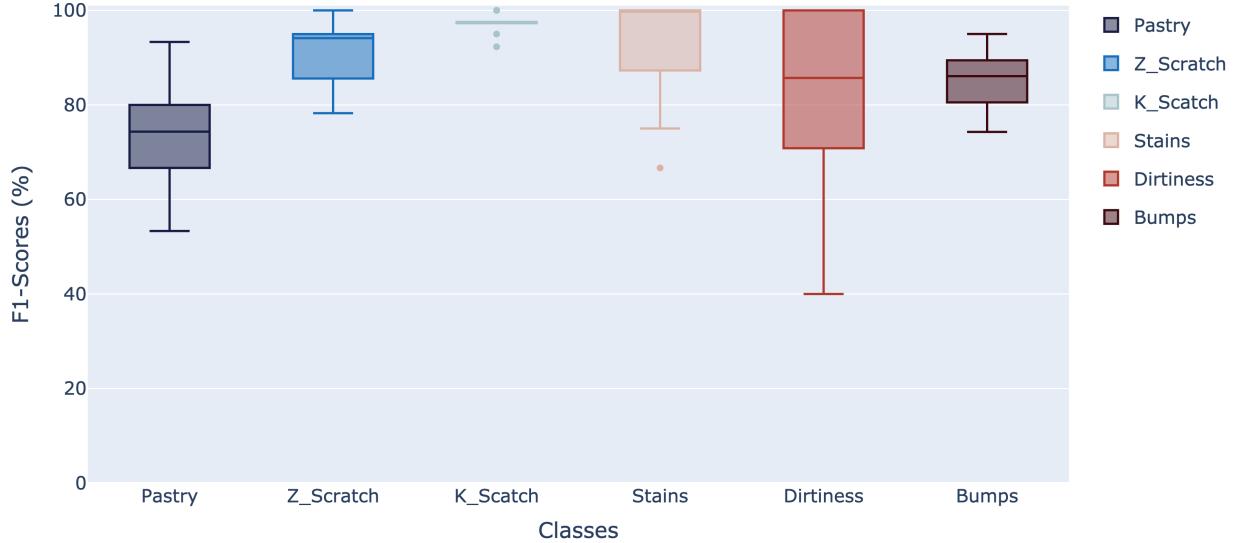


Figura 39: Box-plots dos F1-scores para o Bagging de KNN Balanceado via *Oversampling*.

## Bagging Balanceado via *Oversampling* de Árvore de Decisão (DT)

Matrizes de Confusão geradas após 20 iterações

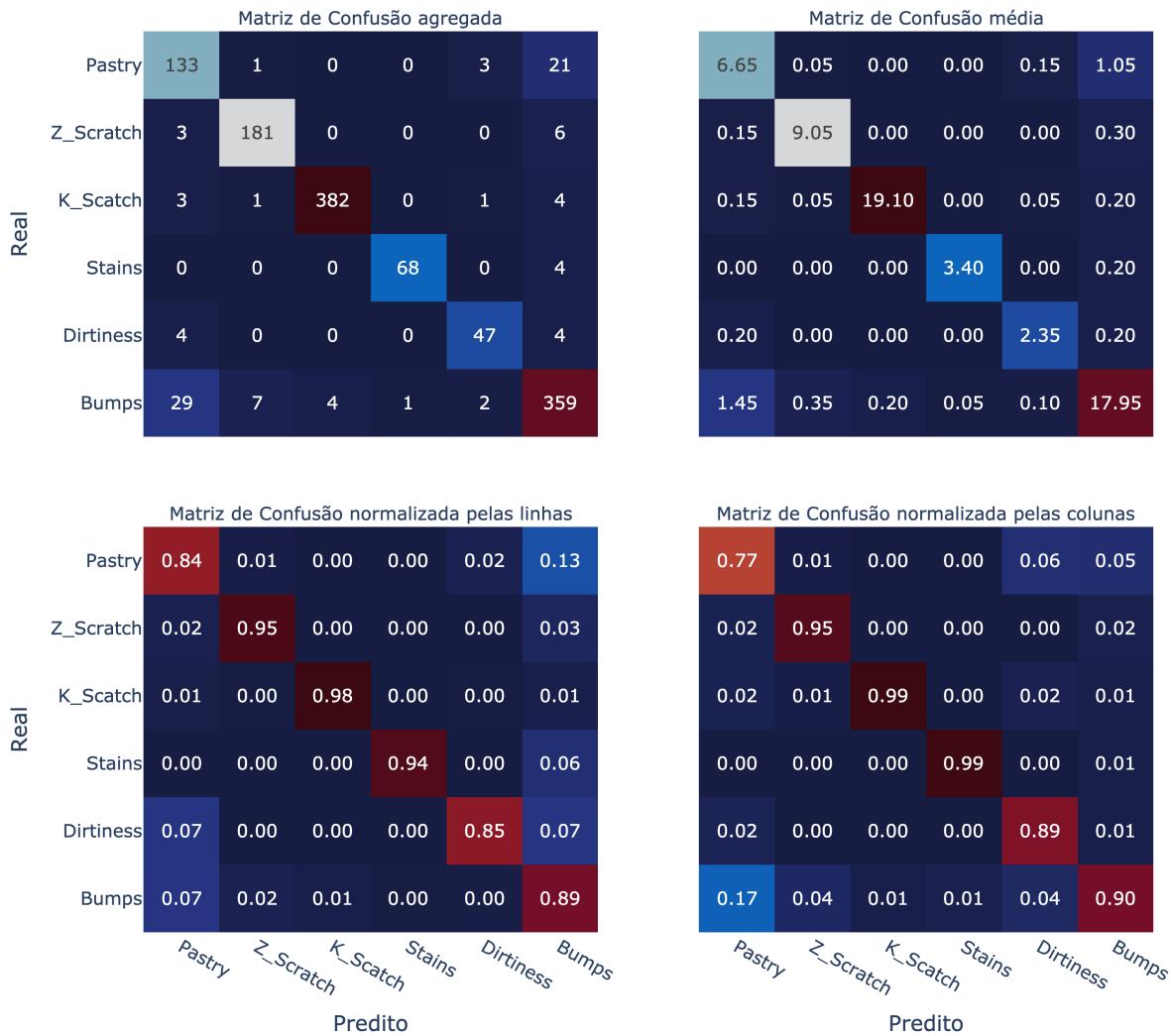


Figura 40: Matrizes de confusão para o Bagging de DT Balanceado via *Oversampling*.

Distribuição das Acurárias após 20 iterações

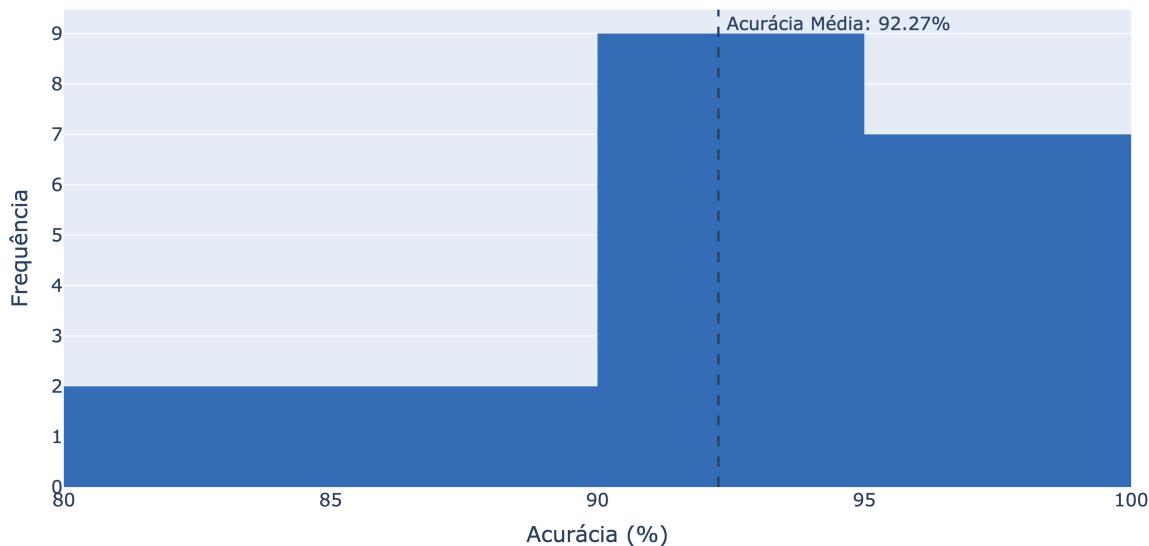


Figura 41: Distribuição das Acurárias para o Bagging de DT Balanceado via *Oversampling*.

Boxplots de F1-Scores por label após 20 iterações

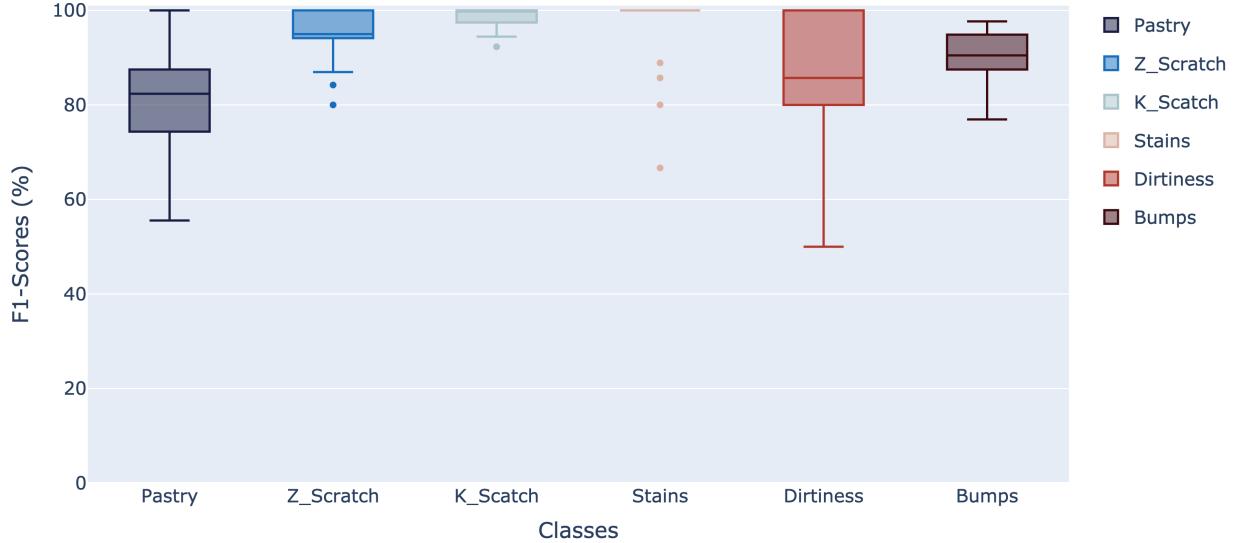


Figura 42: Box-plots dos F1-scores para o Bagging de DT Balanceado via *Oversampling*.

## Bagging Balanceado via *Oversampling* de Support Vector Machine (SVM)

Matrizes de Confusão geradas após 20 iterações

		Matriz de Confusão agregada						Matriz de Confusão média					
		Pastry	1	3	0	3	29	6.10	0.05	0.15	0.00	0.15	1.45
Real	Pastry	122	1	3	0	3	29	0.10	9.00	0.20	0.00	0.00	0.20
	Z_Scratch	2	180	4	0	0	4	0.10	0.15	19.05	0.00	0.00	0.25
	K_Scratch	2	3	381	0	0	5	0.00	0.00	0.00	3.40	0.00	0.20
	Stains	0	0	0	68	0	4	0.30	0.00	0.00	0.00	2.40	0.05
	Dirtiness	6	0	0	0	48	1	1.55	0.55	0.20	0.15	0.25	17.40
	Bumps	31	11	4	3	5	348	0.75	0.01	0.01	0.00	0.05	0.07

		Matriz de Confusão normalizada pelas linhas						Matriz de Confusão normalizada pelas colunas					
		Pastry	1	3	0	3	29	Pastry	1	3	0	3	29
Real	Pastry	0.77	0.01	0.02	0.00	0.02	0.18	0.75	0.01	0.01	0.00	0.05	0.07
	Z_Scratch	0.01	0.95	0.02	0.00	0.00	0.02	0.01	0.92	0.01	0.00	0.00	0.01
	K_Scratch	0.01	0.01	0.97	0.00	0.00	0.01	0.01	0.02	0.97	0.00	0.00	0.01
	Stains	0.00	0.00	0.00	0.94	0.00	0.06	0.00	0.00	0.00	0.96	0.00	0.01
	Dirtiness	0.11	0.00	0.00	0.00	0.87	0.02	0.04	0.00	0.00	0.00	0.86	0.00
	Bumps	0.08	0.03	0.01	0.01	0.01	0.87	0.19	0.06	0.01	0.04	0.09	0.89

Figura 43: Matrizes de confusão para o Bagging de SVM Balanceado via *Oversampling*.

Distribuição das Acurárias após 20 iterações

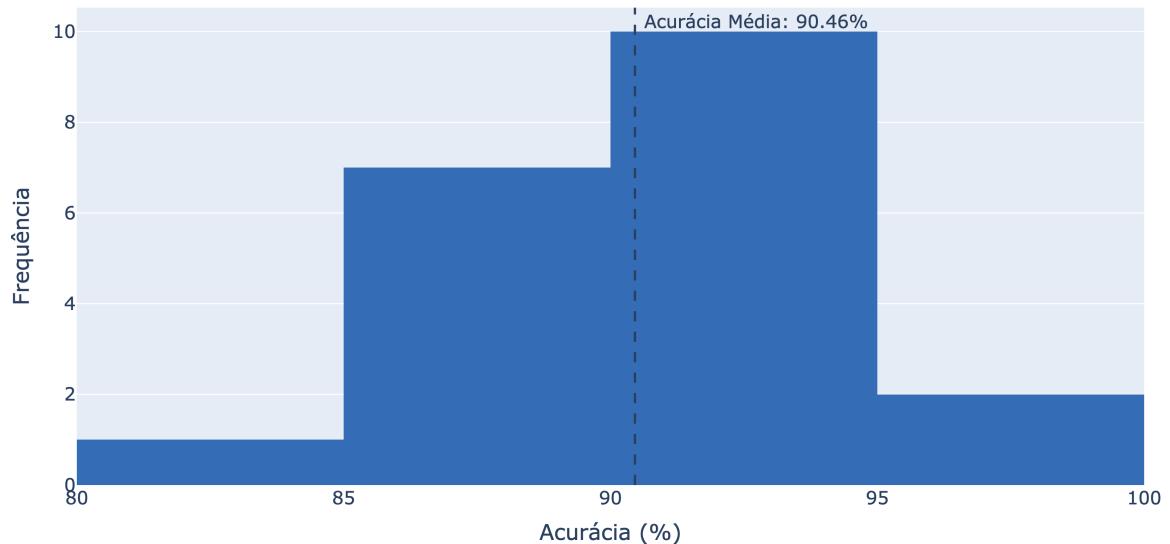


Figura 44: Distribuição das Acurárias para o Bagging de SVM Balanceado via *Oversampling*.

Boxplots de F1-Scores por label após 20 iterações

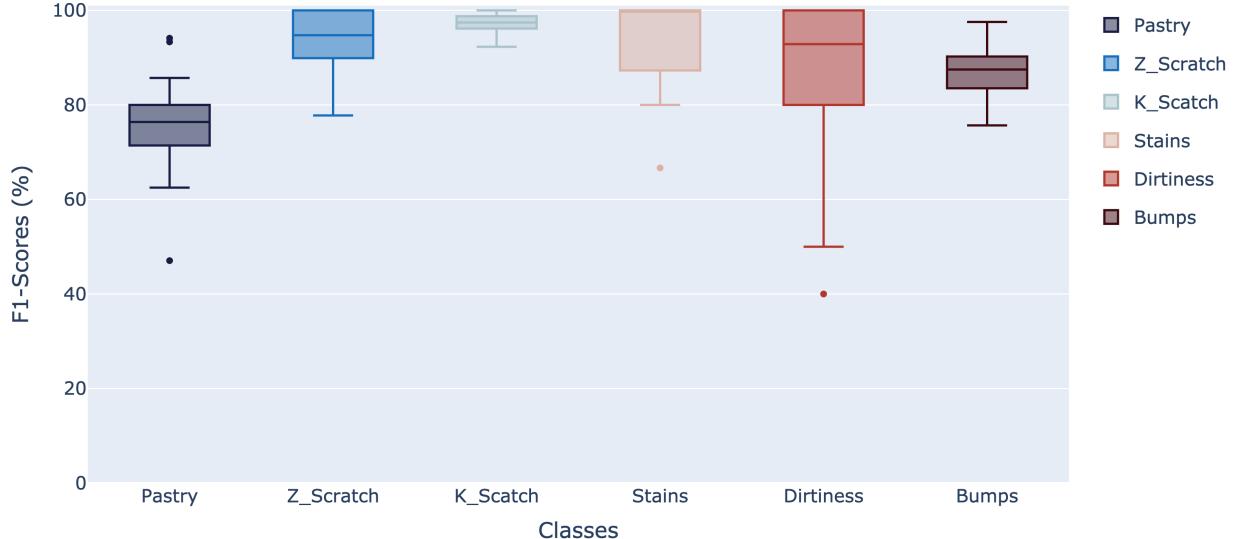
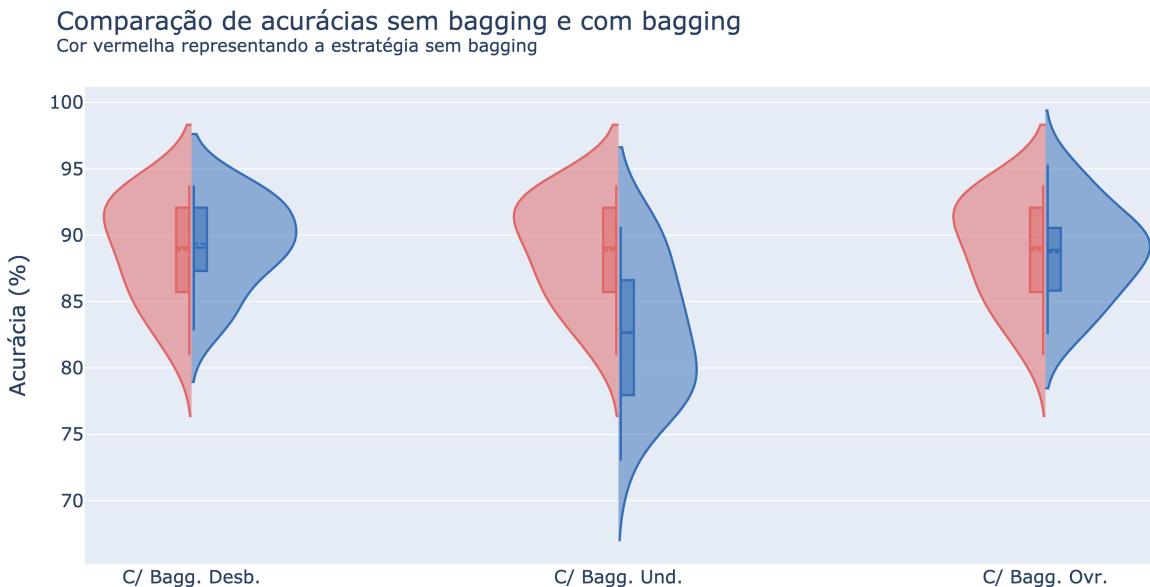


Figura 45: Box-plots dos F1-scores para o Bagging de SVM Balanceado via *Oversampling*.

## **Apêndice D**

**Resultados dos Testes de Hipótese comparativos entre Bagging e  
não Bagging**

## K-Nearest Neighbors (KNN)



(a) Comparação das Acurárias para o KNN.

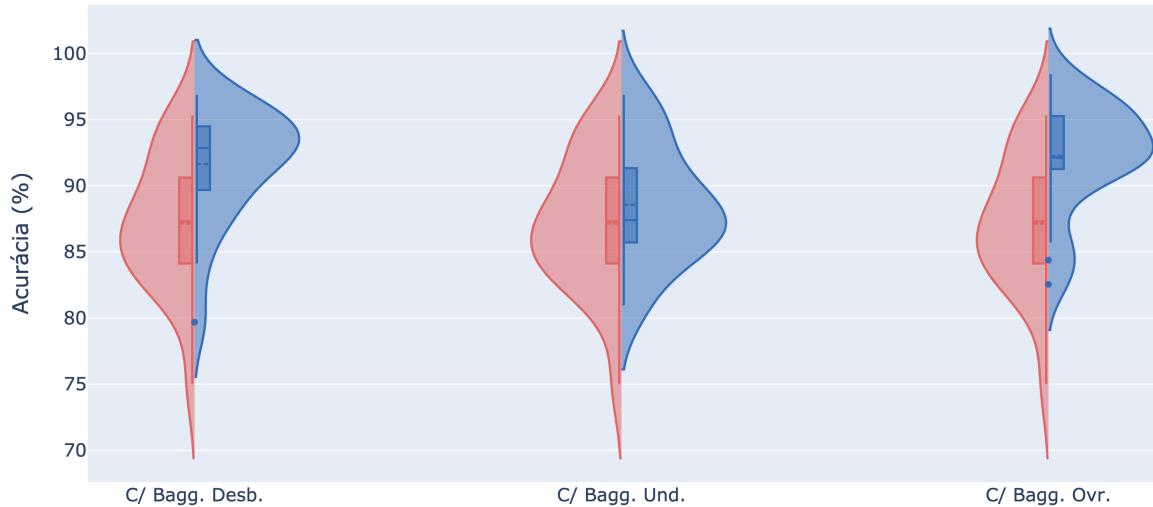


(b) Comparação dos F1-scores para o KNN.

Figura 46: Comparação da distribuição de Acurácia e F1-score entre um modelo único e com diferentes estratégias de Bagging para o KNN.

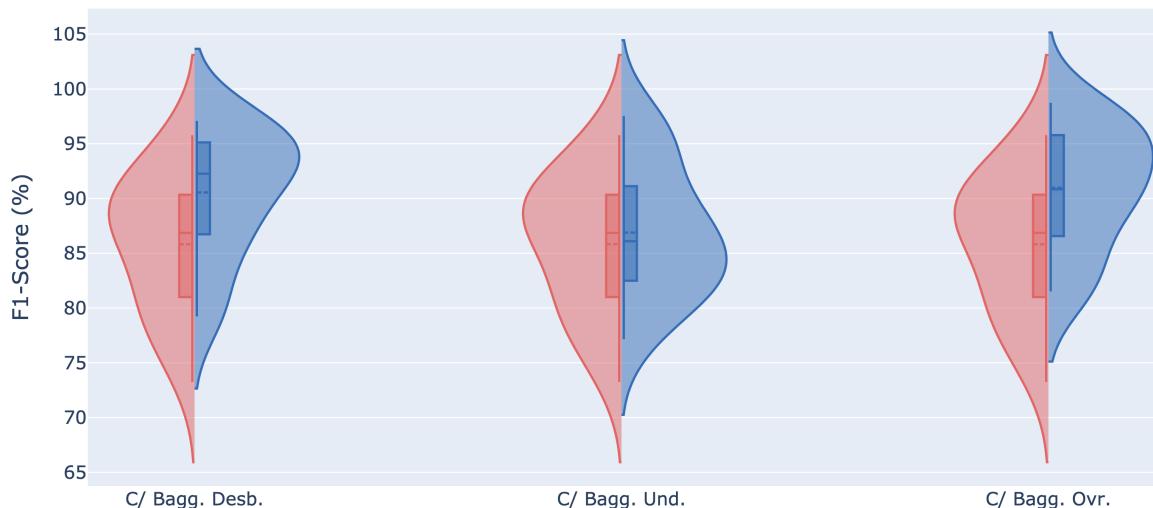
# Árvore de Decisão (DT)

Comparação de acurácia sem bagging e com bagging  
Cor vermelha representando a estratégia sem bagging



(a) Comparação das Acuráncias para o DT.

Comparação de f1-scores sem bagging e com bagging  
Cor vermelha representando a estratégia sem bagging

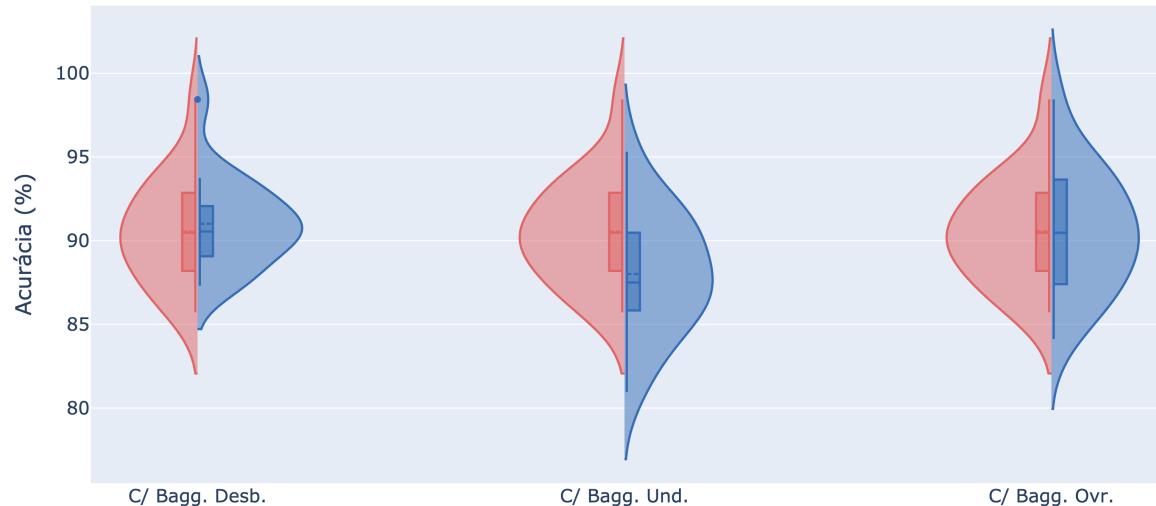


(b) Comparação dos F1-scores para o DT.

Figura 47: Comparação da distribuição de Acurácia e F1-score entre um modelo único e com diferentes estratégias de Bagging para o DT.

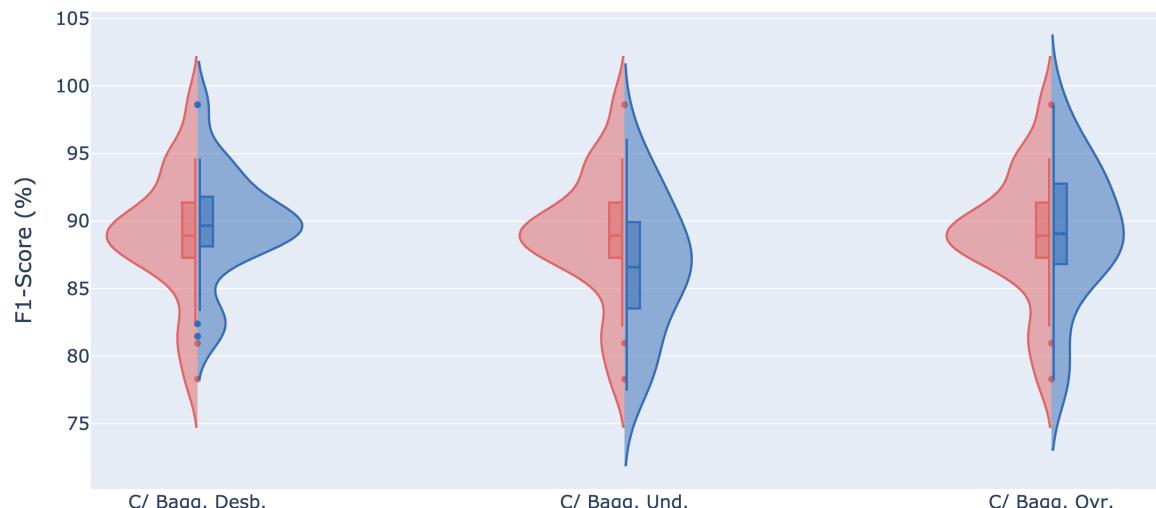
# Support Vector Machine (SVM)

Comparação de acurácia sem bagging e com bagging  
Cor vermelha representando a estratégia sem bagging



(a) Comparação das Acurárias para o SVM.

Comparação de f1-scores sem bagging e com bagging  
Cor vermelha representando a estratégia sem bagging



(b) Comparação dos F1-scores para o SVM.

Figura 48: Comparação da distribuição de Acurácia e F1-score entre um modelo único e com diferentes estratégias de Bagging para o SVM.

# **Apêndice E**

**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-aprendizagem-de-maquina>