UNIVERSIDADE FEDERAL DE PERNAMBUCO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA
APRENDIZAGEM DE MÁQUINA

COMPARAÇÃO ENTRE CLASSIFICADORES

João Victor Moreira Cardoso
Marcos Heitor Carvalho de Oliveira
Matheus Viana Coelho Albuquerque
Rodrigo Beltrão Valença
Olinda/Recife/Paulista/Teresina, Maio 2024

Contents

1 Objetivo

Neste estudo, comparamos os classificadores K-Nearest Neighbors (K-NN), Learning Vector Quantization (LVQ), Support Vector Machines (SVM), Árvore de Decisão (AD), Multilayer Perceptron (MLP), Stacking Ensemble e Random Forest usando a base de dados completa do Spotify. O foco foi aplicar os protocolos de avaliação/comparação de classificadores de acordo com J. Demšar [1], visando desenvolver habilidades críticas na análise de métricas de desempenho diferentes.

Neste estudo, discutimos o uso de testes paramétricos e não paramétricos para analisar cada modelo, além de fornecer uma breve introdução sobre o funcionamento de cada classificador com suas vantagens e desvantagens. Também comparamos as métricas de acurácia, F1-score, precisão e recall para cada conjunto de parâmetros de entrada. Apresentamos um fluxograma do processo de avaliação e as conclusões derivadas dos resultados obtidos.

2 Revisão Teórica

Neste capitulo apresentaremos o funcionamento de cada um dos modelos utilizados na comparação entre classficadores, além disso destacaremos suas principais vantagens e desvantagens. Ao final do capitulo é apresentado os passos realizados na preparação dos dados do dataset do Spotify.

2.1 Funcionamento dos Modelos

2.1.1 K-NN

1. Algoritmo: O princípio do K-Nearest Neighbours (K-NN) é fazer o modelo analisar e entender quais instâncias estão mais próximas entre si e prever qual a classe de um dado vetor $\{\vec{r}_i\}$ qualquer entre seus K vizinhos. Para classificar uma nova instância x_i, o algoritmo calcula a distância entre essa instância e todas as instâncias de treinamento. Com base na distância calculada, o algoritmo seleciona os K (o valor de K é um parâmetro definido pelo usuário) vizinhos mais próximos da nova instância x_i. A classe mais comum entre os K vizinhos mais próximos é atribuída à nova instância.

2. Vantagens:

   - Simplicidade e fácil implementação.
   - Não faz suposições sobre a distribuição dos dados.
   - Apropriado para conjuntos de dados pequenos.

3. Desvantagens:

   - Sensível à escala dos dados.
   - Alto custo computacional durante a fase de teste.
   - Impactado por atributos irrelevantes ou ruído nos dados.
   - Necessidade de escolha do parâmetro K pode afetar o desempenho.

2.1.2 LVQ

1. Algoritmo: O Learning Vector Quantization (LVQ) é um algoritmo de classificação que usa protótipos de classe para representar os dados. Durante o treinamento, os protótipos são ajustados para se aproximarem das instâncias de treinamento. Para classificar novas instâncias, o LVQ calcula a distância entre elas e os protótipos, atribuindo a classe do protótipo mais próximo.

   LVQ usa a distância euclidiana ou outras medidas de distância para calcular a proximidade entre instâncias e protótipos, e a taxa de aprendizado α controla a magnitude das atualizações dos protótipos durante o treinamento.

   Durante o treinamento, os protótipos são ajustados com base na distância entre a instância de treinamento atual x e cada protótipo w_i usando a seguinte regra de atualização da equação 1

$$w_i^{n+1} = w_i^n + \alpha * d(x, w_i) * (x - w_i) \tag{1}$$

   onde n é a iteração atual, $d(x, w_i)$ é a distância entre x e w_i e α é a taxa de aprendizado.

2. Vantagens:

   - Interpretabilidade dos protótipos.
   - Eficiente para classes bem definidas.
   - Baixa complexidade computacional.

3. Desvantagens:

   - Sensível à inicialização dos protótipos.
   - Dependente da taxa de aprendizado α.
   - Dificuldade em lidar com classes sobrepostas.
   - Requer conhecimento prévio do número de classes.

2.1.3 AD

1. Algoritmo: O algoritmos da Àrvore de Decisão (AD) é um algoritmo de aprendizado de máquina que funciona como uma árvore de decisão, onde cada nó representa uma característica (ou atributo), cada ramo representa uma decisão com base nessa característica, e cada folha representa um resultado (ou classe). Cada atributo fornece informações distintas sobre as instâncias de dados e é usado para descrever e distinguir essas instâncias durante a análise ou modelagem.

   Durante o treinamento, o algoritmo divide os dados de entrada em subconjuntos cada vez mais puros, buscando separar as diferentes classes da melhor maneira possível. Isso permite que o classificador tome decisões com base nas características dos dados e preveja a qual classe uma nova instância pertence.

   O algoritmo de árvore de decisão usa uma abordagem matemática para dividir os dados e construir a árvore de decisão. Essa abordagem matemática envolve cálculos de entropia, ganho de informação ou índice Gini para determinar a melhor forma de dividir os dados e criar a estrutura da árvore de decisão.

   O cálculo do ganho de informação é feito em três passos. Inicialmente é calculado a entropia antes da divisão como na equação 2

$$H(D) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{2}$$

onde p_i é a proporção de exemplos na classe i antes da divisão. Após isso, é calculada a entropia após a divisão e o peso dessa divisão para cada atributo como mostrado na equação 3

$$H(D|A) = \sum_{i=1}^{m} \frac{N_j}{N} \times H(D_j) \tag{3}$$

onde N_j é o número de exemplos no subconjunto após j divisões, N é o número total de exemplos, e $H(D_j)$ é a entropia do subconjunto j

Calcule o ganho de informação como a diferença entre a entropia inicial e a entropia após a divisão

$$Gain(D, A) = H(D) - H(D|A) \tag{4}$$

Um ganho de informação mais alto indica que o atributo é mais importante para a divisão dos dados, pois resulta em uma redução maior na incerteza após a divisão. Este processo é repetido para todos os atributos, até que o atributo com o maior ganho de informação seja escolhido como o melhor atributo para dividir os dados em cada nó da árvore de decisão.

2. Vantagens:
   - As árvores de decisão são fáceis de interpretar.
   - Lida bem com diferentes tipos de dados.
3. Desvantagens:
   - Podem sofrer de overfitting.
   - Tem limitações na captura de correlações complexas.

2.1.4 MLP

1. Algoritmo: o Multilayer Perceptron (MLP) é um modelo de Redes Neurais Artificiais em que usamos, simultaneamente, vários neurônios em formas de perceptrons, i.e., vários núcleos de decisão que retornam determinados valores baseados em funções de ativação. Essas multicamadas existem para determinarmos a classificação dos objetos em casos que uma única reta não consegue separar os elementos. Ainda, o MLP consiste em 3 ou mais camadas (uma camada de entrada e uma camada de saída com um ou mais camadas ocultas). Uma vez que temos uma rede totalmente conectada, cada nó em uma camada se conecta, com um certo peso, em um nó da camada seguinte. Isso significa, na prática, que as sinapses são influenciadas pelos pesos que damos em um determinado intervalo de iterações de classificação, usando funções de custo fornecidas (no âmbito desse relatório, as configurações da função de custo foram hiperparâmetros pesquisados).
2. Vantagens:
   - Pode ser aplicado a problemas complexos não-lineares [2].
   - Funciona bem com datasets com muitas instâncias.

- Fornece resultados parecidos, ainda que se diminua o tamanho do dataset escolhido.

3. Desvantagens:

   - Sensível aos hiperparâmetros.
   - Treinamento computacionalmente custoso.
   - Interpretação complexa dos resultados.
   - Requer dados rotulados para o treinamento.

2.1.5 SVM

1. Algoritmo: Dado um conjunto de objetos com $\vec{x}_1, \vec{x}_2, ..., \vec{x}_n$ características de classes $y_1, y_2, ..., y_n$, respectivamente. Para um dado Kernel - a termos do classificador em questão, poderíamos escolher entre diversas hiperformas, embora tenhamos optado pela Radio Basis Function (RBF) - o SVM procura classificar bem as classes, antes de tentar maximizar a distância entre a hiperforma e as instâncias. O Kernel, aqui, é importante, pois influencia na separabilidade do problema em datasets como o escolhido do Spotify, em que há pontos, aparentemente, inseparáveis para um hiperplano (SVM linear), por exemplo.

2. Vantagens:

   - Eficiente em espaços de alta dimensão.
   - Modela relações não lineares usando funções de kernel.
   - Possui regularização embutida para evitar overfitting.
   - Eficaz mesmo em conjuntos de dados pequenos.

3. Desvantagens:

   - Sensível aos parâmetros e kernel escolhidos.
   - Treinamento computacionalmente custoso em grandes conjuntos de dados.
   - Interpretação complexa devido à classificação baseada em margens.
   - Sensível a ruídos nos dados.

2.1.6 Stacking

1. Algoritmo: O classificador Stacking é um algoritmo de aprendizado de máquina que combina as previsões de vários modelos de base para obter uma previsão final mais precisa. Cada modelo de base é treinado com os dados de treinamento. Os modelos de base fazem previsões para um conjunto de dados de validação ou teste. As previsões dos modelos de base são usadas como entradas para um meta-modelo, que é treinado para combinar essas previsões e gerar uma previsão final. meta-modelo é usado para fazer a previsão final para novos dados, combinando as previsões dos modelos de base de acordo com a estratégia de combinação definida. O Stacking é eficaz para melhorar a performance ao combinar diferentes abordagens de modelagem.

2. Vantagens:

   - Melhora da performance combinando previsões de vários modelos.

- Redução do overfitting ao combinar diferentes modelos.
- Flexibilidade para usar diversos algoritmos de base.
- Capacidade de capturar relações complexas nos dados.

3. Desvantagens:

   - Maior complexidade de implementação e ajuste.
   - Interpretação mais complexa devido à combinação de modelos.
   - Requer um conjunto de dados suficientemente grande para treinar os modelos de forma eficaz.

2.1.7 Random Forest

1. Algoritmo: O classificador Random Forest é um algoritmo que cria múltiplas árvores de decisão usando diferentes amostras de dados e atributos aleatórios. Ele combina as previsões dessas árvores para obter uma previsão final mais precisa e robusta.

   Cada árvore no Random Forest é treinada com uma amostra bootstrap dos dados de treinamento, com reposição. Isso aumenta a diversidade das árvores. Os nós de cada árvore são divididos com base em um subconjunto aleatório de atributos, reduzindo a correlação entre as árvores. A previsão para uma nova instância é feita por votação entre as árvores, escolhendo a classe mais votada como a previsão final do Random Forest.

2. Vantagens:

   - Reduz o overfitting.
   - Boa generalização.
   - Lida com dados não lineares.
   - Robusto contra ruído.
   - Pode lidar com grandes conjuntos de dados.

3. Desvantagens:

   - Complexidade computacional.
   - Menos interpretabilidade.
   - Pode ocorrer sobreajuste.
   - Requer ajuste de hiperparâmetros.
   - Pode ter viés em conjuntos de dados desbalanceados.

2.2 Preparação da Base de Dados Spotify

A base de datos usada foi a base de dados do Spotify[1], composta por 21 colunas e 114000 linhas para cada coluna. Um exemplo do dataset reduzido pode ser observado na tabela 1 abaixo:

[1] https://huggingface.co/datasets/maharshipandya/spotify-tracks-dataset

artists	album_name	...	time_signature	track_genre
Gen Hoshino	Comedy	...	4	acoustic
Ben Woodward	Ghost (Acoustic)	...	4	acoustic
Ingrid Michaelson	ZAYN	...	4	acoustic
Kina Grannis	Crazy Rich Asians (Original Motion Picture Soundtrack)	...	3	acoustic
...	...	...	...	...

Table 1: Exemplo do dataset

Após uma breve analise, é notorio que o dataset apresenta catacteristicas categoricas como numericas, além disso é possivel observar que nas colunas de artistas, albuns e disco apresentam uma entrada sem valores correspondentes para essas colunas. Também foi feita uma análise sobre a distribuição das colunas númericas assim como mostrado na figura 1 e figura 2:

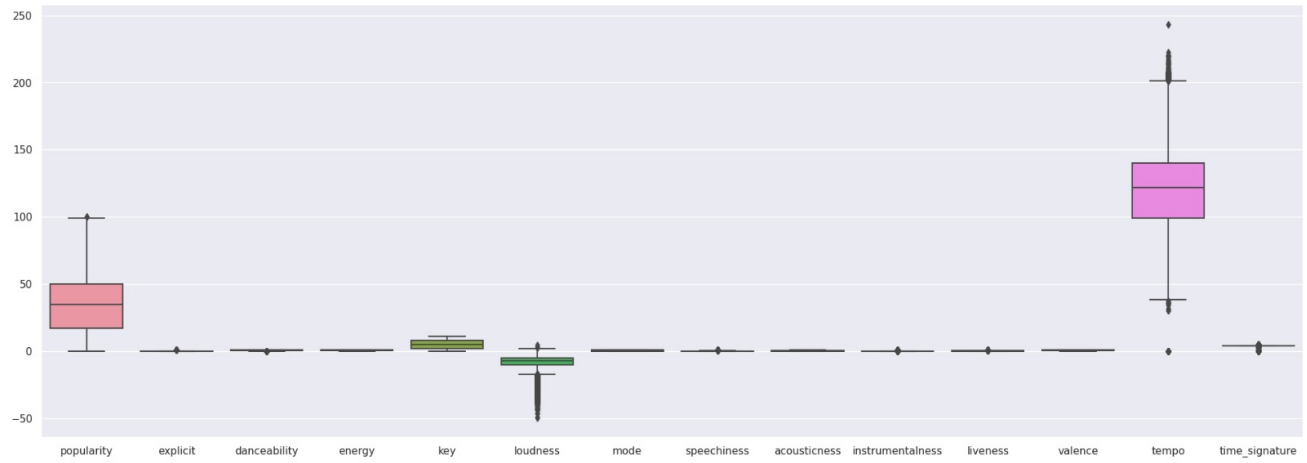

Figure 1: Distribuição das colunas númericas sem a coluna duration_ms

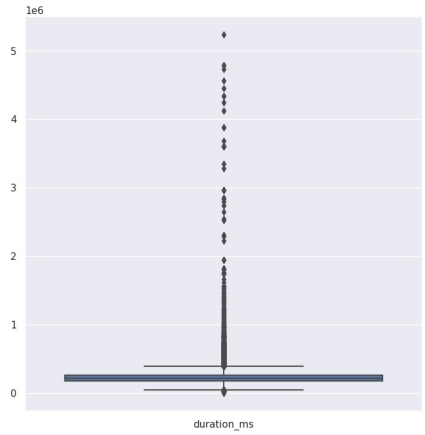

Figure 2: Distribuição da coluna duration_ms

É fácil notar pelas imagens que a ordem de grandeza dos parâmetros difere bastante, além disso temos multiplos valores de outliers para cara distribuição númerica, o que pode afetar cada um dos classificadores. Na seção abaixo será discorrido como foi feita a limpeza desses dados.

3 Protocolo de Comparação

Nesta seção, explicaremos o pré-processamento e a divisão do dataset, além dos testes estatísticos usados nos modelos e o motivo de cada decisão.

3.1 Preparação e Divisão da Base de Dados

3.1.1 Preparação dos Dados

A etapa de preparação dos dados é essencial para preparar os dados para o treinamento do modelo, nessa etapa são feitas decisões de quais e como os dados vão ser processados para serem passados para o modelo, tanto no treinamento, como em inferências após o treinamento. Essa etapa engloba também a limpeza dos dados, permitindo a remoção de dados ruidosos que possam influenciar negativamente o modelo ao longo do processo de treinamento.

Nessa etapa foram realizadas os seguintes pré-processamentos dos dados:

1. **Seleção das Variáveis**: Para esse projeto, foram selecionadas as variáveis numéricas para o desenvolvimento dos modelos. As variáveis categóricas presentes no conjunto de dados apresentam alta cardinalidade o que podem adicionar uma complexidade adicional ao depender do tipo de *encoder* utilizado para transformá-los em variáveis numéricas a serem alimentadas pelo modelo.
2. **Limpeza dos Dados**: Os registros que continham variáveis nulas foram removidas do conjunto de dados. Como eles representavam uma porcentagem muito pequena da base foi decidido removê-los ao invés de realização um pré-processamento de imputação dos dados. Essa etapa permitiu ter uma qualidade melhor dos dados, garantindo que apenas informações completas e utilizáveis fossem alimentadas ao modelo.

3. **Padronização dos Dados**: A padronização é um passo crucial no pré-processamento dos dados, pois permite que o modelo trate todas as características de maneira uniforme. Isso é especialmente importante para modelos que utilizam cálculos de distância entre registros, como o *K-Nearest Neighbors* (*KNN*) e o *Learning Vector Quantization* (*LVQ*). Por conta disso, usamos um normalizador do tipo MinMaxScaler. Esse escalador normaliza todos os pontos entre zero e um considerando o valor minimo e máximo das suas entradas.

3.1.2 Divisão dos Dados

A divisão dos dados é uma etapa necessária no desenvolvimento de modelos de aprendizagem de máquina, pois permite construir e avaliar de forma apropriada um modelo. No projeto, o conjunto de dados foi dividido em três novos conjuntos de maneira estratificada para manter a mesma proporção de cada categoria alvo em cada novo conjunto:

- **Conjunto de Treino**: O conjunto de treino é utilizado para ajustar os parâmetros do modelo ao longo do treinamento, constituindo-se pela maior porção do conjunto de dados. No projeto, a divisão dos dados resultou em 60% dos registros sendo alocados ao conjunto de treinamento, resultando na quantidade de 68,400 registros.
- **Conjunto de Validação**: O conjunto de validação é utilizado para ajustar os hiper-parâmetros e avaliar a performance do modelo durante o processo de treinamento. Esse conjunto permite avaliar a performance do modelo, evitando problemas como *overfitting*. No projeto, a divisão dos dados resultou em 20% dos registros sendo alocados ao conjunto de validação, resultando na quantidade de 22,800 registros.
- **Conjunto de Teste**: O conjunto de teste é utilizado para avaliar a capacidade de generalização do modelo em dados não vistos ao longo do processo de treinamento. É a porção final dos dados reservada para verificar a performance do modelo após ajuste do modelo com o conjunto de validação. No projeto, a divisão dos dados resultou em 20% dos registros sendo alocados ao conjunto de teste, resultando na quantidade de 22,800 registros.

3.1.3 Validação Cruzada

A validação cruzada é uma técnica que permite uma avaliação robusta do modelo, proporcionando uma estimativa mais confiável do seu desempenho. Ela permite dividir os dados de treinamento em múltiplos subconjuntos, chamados de *folds*. O processo funciona da seguinte maneira:

1. **Divisão em Folds**: Primeiramente, os dados são divididos em k *folds*. No projeto, foi utilizado o *stratified K-fold*, que divide os dados de forma que cada fold tenha aproximadamente a mesma proporção de classes, garantindo que cada subconjunto seja representativo do conjunto total. Nesse projeto, foi escolhido 5 *folds*.
2. **Treinamento e Avaliação**: Em seguida, o modelo é treinado *folds*. Em cada iteração, um fold é reservado para ser o conjunto de teste, enquanto os k - 1 *folds* são usados para o treinamento. Dessa forma, o modelo é treinado em diferentes partes dos dados e testado em diferentes subconjuntos.
3. **Resultados**: Dessa forma, a validação cruzada permite avaliar a performance do modelo utilizando diferentes divisões dos dados, proporcionando uma estimativa mais robusta do seu desempenho. Essa

abordagem gera uma distribuição de resultados que pode ser utilizada para comparar o desempenho do modelo com outros classificadores por meio de testes estatísticos.

3.2 Métricas de Avaliação

Para avaliar a performance dos classificadores nos conjuntos de treino, validação e teste foi feita a seleção das seguinte métricas:

- **Acurácia**: Essa métrica mede a proporção de previsões corretas em relação ao total de previsões feitas pelo classificador, oferecendo uma visão geral do desempenho do classificador.
- **Recall**: Essa métrica indica a proporção de verdadeiros positivos corretamente identificados, oferecendo uma perspectiva sobre a capacidade do classificador em identificar corretamente as instâncias positivas, a boa performance dela é importante em situações onde a penalidade por falsos negativos é alta. No contexto multi-classes, o recall pode ser calculado para cada classe individualmente e depois ser combinado, tipicamente usando médias (macro, micro ou ponderada).
- **Precisão**: Essa métrica mede a proporção de verdadeiros positivos em relação ao total de previsões positivas para cada classe, a boa performance dela é crucial em situações onde o custo de falsos positivos é alto. Similar ao recall, a precisão em problemas multiclasses pode ser calculada individualmente para cada classe e depois combinada.
- **F1-Score**: Essa métrica é a média harmônica entre precisão e recall, fornecendo um único valor que equilibra os dois aspectos. Similar ao recall, o F1-Score em problemas multiclasses pode ser calculada individualmente para cada classe e depois combinado usando médias. É útil em casos onde há um desequilíbrio de proporção entre as classes a serem preditas.

3.3 Procedimentos Estatísticos para Comparação

Para avaliar os classificadores e realizar os testes estatísticos de comparação, foi realizada uma validação cruzada com 5 *folds* para cada estimador, utilizando a melhor configuração de hiperparâmetros obtida através da busca. Dessa forma, foi obtida uma distribuição de 5 valores de acurácia em cada *fold*, que contém 13,800 registros, para cada modelo construído.

A partir das distribuições obtidas, foi realizado o teste estatístico *Shapiro-Wilk* para verificar a normalidade de cada distribuição. O teste *Shapiro-Wilk* foi escolhido por ser um método mais apropriado para amostras pequenas (< 50 amostras), em comparação com o teste de Kolmogorov–Smirnov (KS) [3]. Para esse e todos os testes, será utilizado o nível de significância de 5% que representa a probabilidade máxima de se rejeitar a hipótese nula, indicando que, se o valor-p for menor que 5%, rejeitaremos a hipótese nula, sugerindo que há evidências estatisticamente significativas para apoiar a hipótese alternativa.

Se o teste *Shapiro-Wilk* indicar que os dados seguem uma distribuição normal e, como as distribuições são emparelhadas, será aplicado o teste *repeated-measures ANOVA* para verificar se há uma diferença estatística entre os dados. Caso contrário, será utilizado o teste de *Friedman*. Se não houver diferença estatística entre os dados, podemos concluir que a performance dos classificadores é a mesma. Caso haja, os classificadores serão tomados dois a dois e será aplicado o *t-test* pareado, se as distribuições forem normais, ou o teste de *Wilcoxon*, se não forem, para verificar se uma diferença estatística entre os dois classificadores assim como sugerido por J. Demšar [1] .

4 Execução da Comparação

4.1 Implementação e Teste

A construção dos classificadores foi feita utilizando os algoritmos providos e detalhados na seção 5.2. Cada um desses classificadores apresenta hiperparâmetros como os comentados na seção 2.1, o que foi feito foi variar o espaço de busca para cada um dos póssiveis hiperparâmetros e depois foram utilizado dois otimizadores de hiperparâmetros. Os dois otimizadores utilizados foram o de busca de grade e de busca randomica.

A busca de grade, também conhecida como *GridSearch* testa várias combinações de hiperparâmetros para encontrar a melhor configuração de um modelo de aprendizado de máquina, usando validação cruzada estratificada para avaliar o desempenho de cada combinação. Essa busca foi usada para todos os classificadores com excessão da Árvore de decisão e do Random Forest. Para os dois modelos de árvore a busca dos hiper-parâmetros foi feita através da busca randomica, o otimizador *RandomSearch*. O *RandomSearch* é um otimizador semelhante ao *GridSearch*, no entanto, ele seleciona aleatoriamente conjuntos de valores para os hiperparâmetros e os avalia também por meio de validação cruzada estratificada. O motivo dessa decisão se deve ao fato do espaço de busca dos modelos de árvore ser maior e de que esse otimizador se prova mais útil para otimizar modelos de machine learning quando não se sabe quais valores de hiperparâmetros são os melhores.

Para o KNN os parâmetros mais relevantes do modelo são a distância entre os vizinhos e o número de vizinhos. As distâncias testadas foram a euclidiana e a de manhattan e o valor dos vizinhos foi variado entre 1 e 80.

Para o LVQ, o processamento foi muito custoso, por conta disso, os parâmetros de teste envolveram apenas 2 testes nos prototipos por classe, testando com 1 ou 3 prototipos. Além disso, foi testado também as mesmas distâncias usadas pelo KNN variando apenas a taxa de aprendizado α entre os valores 0.1 e 0.5 com 5 epocas de aprendizado.

Na Árvore de Decisão testamos todos os parâmetros disponíveis para construir a árvore, variamos a altura máxima da árvore indo de 5 até 10000, mesmo intervalo utiliado para o número minimo de amostras por folha e número máximo de nós folha. Além disso, variamos os críterios de ganho de informação comentados, como entropia, índice Gini e, adicionalmente *log loss*.

Para o MLP variamos o tamanho das camadas de aprendizagem de 50 em 50, assim como as funções de ativação: relu, tangente e logistica, o uso da taxa de aprendizagem constante e adaptativa, o fator α de penalidade de $1 * 10^{-5}$ e $1 * 10^{-4}$, o número de iterações do algoritmo, entre 300, 500 e 700 iterações e o otimizador dos pesos sendo o adam ou o sgd.

No classificador do SVM testamos os kerneis linear,

Para a implementação do algoritmo de Stacking no projeto, decidimos usar os modelos KNN, MLP e LVQ que foram previamente treinados. Desse modo conseguimos reaproveitar as melhores configurações que conseguimos de cada um dos classificadores.

Por fim, para o Random Forrest variamos os parâmetros da Árvore de Decisão, mas num intervalo menor de uma altura máxima de 50 para cada árvore, usando de 5 a 1000 estimadores, e variando apenas até 100 amostras por folha e por número mínimo de amostras para uma divisão.

O espaço de busca descrito acima e os hiperparâmetros ótimos obtidos para de cada um dos classificadores

podem ser vistos na Tabela 2 da seção 4.2. Depois de obtidos esses resultados, também calculamos as métricas descritas na subseção 3.2 para cada classificador treinado com os melhores hiperparâmetros encontrados, essas resultado pode ser observado na Tabela 3 também da seção 4.2.

4.2 Análise dos Resultados

De acordo com a metodologia proposta, obtivemos os melhores hiperparâmetros comentados como descrito na tabela 2 abaixo:

MODELOS	PARÂMETROS TESTADOS	MELHORES PARÂMETROS
KNN	"k neighbors": np.arange(1,81,2), "metric": ["euclidean", "manhattan"]	"k neighbors": [29], "metric": ["manhattan"]
LVQ	"prototype n per class": [1,3], "distance type": ["euclidean", "manhattan"], "solver params": {"max runs": 5, "step size": step} for step in [0.1, 0.3,0.5]	"prototype n per class": [3], "distance type": ["euclidean"], "solver params": {"max runs": 5, "step size": 0.1}
AD	"criterion": ["gini", "entropy", "log loss"], "splitter": ["best", "random"], "max depth": np.arange(5,10000,50), "min samples leaf": np.arange(1,10000,50), "max features": ["sqrt", "log2"], "max leaf nodes": np.arange(2,10000,50)	"criterion": ["gini"], "splitter": ["best"], "max depth": [305], "min samples leaf": [1], "max features": ["log2"], "max leaf nodes": [4952]
MLP	"hidden layer sizes": [(50,), (100,), (50, 50)], "activation": ["relu", "tanh", "logistic"], "solver": ["adam", "sgd"], "alpha": [0.0001, 0.001], "learning rate": ["constant", "adaptive"], "max iter": [300, 500, 700]	"hidden layer sizes": [(100,)], "activation": ["relu"], "solver": ["adam"], "alpha": [0.0001], "learning rate": ["adaptive"], "max iter": [700]
SVM	"kernel": ["rbf", "poly", "linear"], "C": [1,2,3,4,5,7,10,100], "gamma":[1,2,3,4,5,7,10,100]	"kernel": ["rbf"] "C": [100] "gamma": [2]
Stacking	"estimators": [["mlp","knn","lvq"], ["mlp","lvq"], ["mlp","knn"], ["knn","lvq"]], "final estimator:" [LogisticRegression, DecisionTreeClassifier]	"estimators": ["mlp","knn","lvq"], "final estimator:" [LogisticRegression]
Random Forest	"n estimators": np.arange(5, 1000, 10), "criterion": ["gini", "entropy", "log loss"], "max depth": np.arange(1,50), "min samples split": np.arange(2,100), "min samples leaf": np.arange(1,100)	"n estimators": [515], "criterion": ["entropy"], "max depth": [43], "min samples split": [19], "min samples leaf": [8]

Table 2: Informações sobre o espaço de busca e os melhores hiper-parâmetros dos classificadores obtidos.

Com as melhores configurações de hiperparâmetros, obtivemos os melhores valores de acuracia, precisão, *recall* e *f1-score* como mostrado na tabela 3:

MODELOS	Métricas no treino	Métricas na validação	Métricas no teste
KNN	"accuracy": 78.320%, "precision": 76.452%, "recall": 78.391%, "f1": 77.409%	"accuracy": 22.486%, "precision": 22.355%, "recall": 22.869%, "f1": 22.609%	"accuracy": 22.418%, "precision": 21.631%, "recall": 21.618%, "f1": 21.624%
LVQ	"accuracy": 18.710%, "precision": 15.428%, "recall": 16.756%, "f1": 16.911%	"accuracy": 17.423%, "precision": 14.327%, "recall": 15.326%, "f1": 14.810%	"accuracy": 17.103%, "recall": 14.895%, "precision": 19.501%, "f1": 16.890%
AD	"accuracy": 39.532%, "precision": 40.891%, "recall": 39.532%, "f1": 39.382%	"accuracy": 21.908%, "precision": 22.506%, "recall": 21.908%, "f1": 21.807%	"accuracy": 21.776%, "precision": 22.245%, "recall": 21.776%, "f1": 21.638%
MLP	"accuracy": 36.341%, "precision": 35.980%, "recall": 36.341%, "f1": 31.103%	"accuracy": 29.635%, "precision": 30.224%, "recall": 29.636%, "f1": 27.913%	"accuracy": 30.732%, "precision": 29.638%, "recall": 30.732%, "f1": 29.092%
SVM	"accuracy": 30.903%, "precision": 27.260%, "recall": 26.648%, "f1": 27.467%	"accuracy": 28.184%, "precision": 26.821%, "recall": 27.184% "f1": 27.001%	"accuracy": 26.570%, "precision": 25.962%, "recall": 26.570%, "f1": 25.781%
Stacking	"accuracy": 34.909%, "precision": 35.105%, "recall": 34.909%, "f1": 34.309%	"accuracy": 33.333%, "precision": 33.351%, "recall": 33.333%, "f1": 32.617%	"accuracy": 33.973%, "precision": 34.139%, "recall": 33.973%, "f1": 33.368%
Random Forest	"accuracy": 68.020%, "precision": 69.669%, "recall": 68.016%, "f1": 67.619%	"accuracy": 31.38%, "precision": 29.85%, "recall": 31.377%, "f1": 29.411%	"accuracy": 31.763%, "precision": 30.224%, "recall": 31.763%, "f1": 29.786%

Table 3: Performance de cada classificador com os melhores hiper-parâmetros obtidos no conjunto de treino, validação e teste.

Da tabela 3 podemos notar um comportamento esperado: as metricas de avaliação são melhores nos conjuntos de treinamento e validação, mas são inferiores quando os modelos são expostos ao conjunto de testes. Pela tabela, também podemos observar que o modelo que teve o melhor desempenho para o conjunto de dados foi o Stacking e o modelo de pior desempenho foi o do LVQ.

Dado o conjunto de cross-validação nós obtemos um conjunto valores para as metricas abordadas, com esses valores é póssivel calcular qual o valor de significância para cada modelo e verificar se ele passa no teste de normalidade. Para esse teste foi utilizado o nível de significância de 5%, ou seja, se o valor-p calculado com o teste for menor que do que 5%, então rejeitamos a hipótese nula de que a distribuição provêm de uma distribuição normal, caso contrário, não podemos rejeitar a hipótese nula e consideramos que os dados provêm de uma distribuição normal. O resultado da distribuição de acurácia para cada modelo e o valor-p correspondente obtido por ser visto na Tabela 4 abaixo:

Modelo	**Valor-p Shapiro-Wilk**
KNN	0.109
LVQ	0.729
AD	0.598
MLP	0.839
SVM	0.657
STACKING	PEGAR RESULTADO
Random Forest	0.713

Table 4: Valor-p resultante do Shapiro-wilk

A partir da tabela 4 podemos constatar que todas as distribuições seguem a distribuição normal. Dessa constatação da normalidade conduziu-se um teste *repeated-measures ANOVA* para verificar se existe uma diferença estatística entre as médias de cada distribuição, sendo a hipótese nula do teste que a média das distribuições é a mesma. Essa métrica foi escolhida por ser apropriada para distribuições normais e por estarmos comparando mais de duas distribuições emparelhadas. O valor de significância utilizado também foi de 5% e o valor-p resultante desse teste foi $1,235 \times 10^{-19}$, indicando que existe uma diferença significativa entre as médias das distribuições.

A partir do resultado de significancia estatistica, foi realizado um teste dois a dois entre os classificadores, de forma a obter qual classificador tem a melhor performance ao ser comparado. Para tanto, foi escolhido o *t-test* pareado, pois as distribuições manipuladas são normais, são pareadas e são calculadas na mesma população. O resultado desse teste para cada par de classificadores pode ser observado na tabela 5 abaixo:

Modelo 1	Modelo 2	Estatística t	Melhor modelo
MLP	Random Forest	-10.78	Random Forest
MLP	KNN	137.18	MLP
MLP	AD	23.14	MLP
MLP	LVQ	83.69	MLP
MLP	SVM	17.41	MLP
Random Forest	KNN	49.41	Random Forest
Random Forest	AD	45.53	Random Forest
Random Forest	LVQ	96.21	Random Forest
Random Forest	SVM	31.98	Random Forest
KNN	AD	2.99	KNN
KNN	LVQ	31.30	KNN
KNN	SVM	-44.40	SVM
AD	LVQ	10.25	AD
AD	SVM	-23.95	SVM
LVQ	SVM	-59.56	SVM

Table 5: Resultado da comparação do *t_test* pareado

5 Apresentação dos Resultados

5.1 Fluxograma do Processo de Avaliação

Preparamos também um fluxograma que indica cada etapa do projeto, esse fluxograma pode ser observado na Figura 3 (também acessável no link https://github.com/Zuluke/Projetos-AM/blob/main/spotify_activity/Fluxograma/Fluxograma.png).

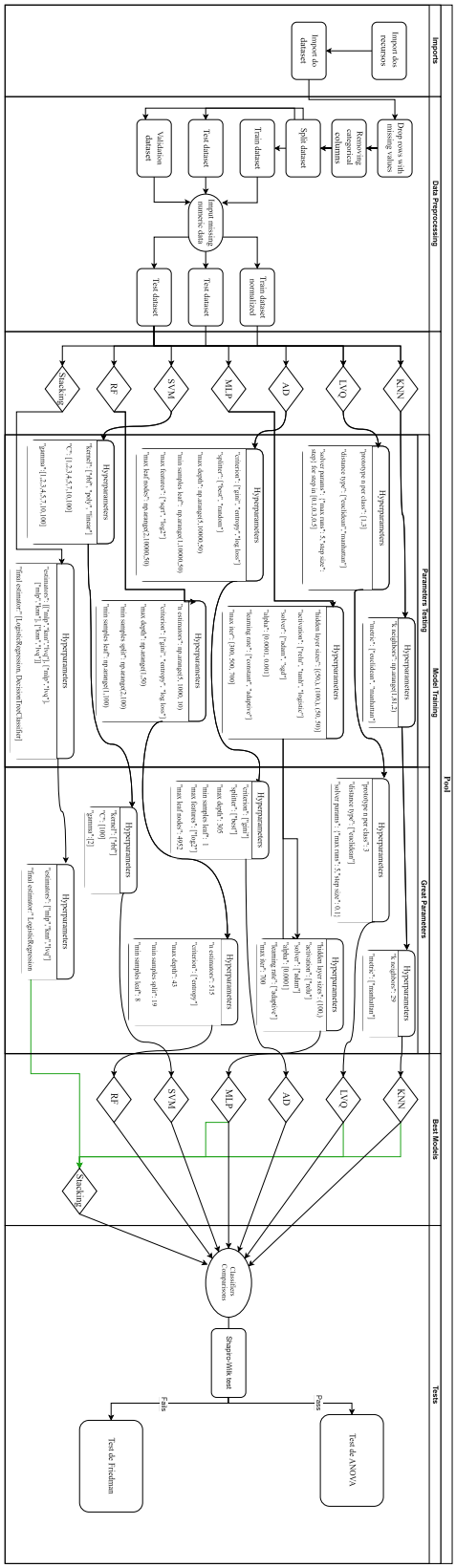

Figure 3: Fluxograma do processo de avaliação

Este fluxograma engloba todas as etapas de processamento desde a aquisição dos dados até as etapas de testes estatisticos.

5.2 Recursos Utilizados

O projeto foi todo realizado na linguagem Python, mais especificamente o Python 3.11. Para o uso dos classificadores, utilizamos as implementações do Random Forest, Stacking, SVM, MLP, AD e KNN providas pela biblioteca do scikit-learn, a implementação do LVQ foi provida pela biblioteca sklvq. Para fazer os processamentos dos dados utilizamos a biblioteca do Pandas e do Numpy. Os plots e figuras foram todos providos pela biblioteca da MatPlotLib. Os testes estátisticos de ANOVA e Shapiro-Wilk também foram os providos pela biblioteca do SciPy e, por fim, a exportação e serialização dos classificadores foi utilizada com ajuda da biblioteca do Pickle do Python.

5.3 Discussão dos Resultados

Após a busca pelos hiperparâmetros definidos na subseção 4.1, foi realizado o teste estatístico com as melhores configurações dos modelos, conforme apresentado na subseção 4.2. Analisando os resultados da Tabela 5, nota-se que o classificador INSERIR-CLASSIFICADOR obteve a melhor performance em comparação com os demais. Essa conclusão é confirmada pelo *t-test* pareado, que indica uma diferença estatisticamente significativa entre a média de acurácia de cada um dos classificadores, tomados dois a dois. Com isso, pode-se afirmar que o classificador com a maior média possui o melhor desempenho. Ao revisitar a Tabela 3, percebemos que os modelos KNN e o Random Forest sofreram sobreajuste no conjunto de treino, apresentando uma diferença de mais de duas vezes da performance do conjunto de treino com o de teste. As diferentes métricas permitem visualizar como os modelos se saem ao considerar diferentes perspectivas de acertos e erros.

6 Discussão Crítica e Conclusões

Ao avaliar os resultados, percebe-se que os classificadores Random Forest, Stacking e MLP apresentaram as maiores performances, considerando a acurácia no conjunto de teste. Esses algoritmos são complexos e robustos, o que lhes permite se adaptar melhor ao conjunto de dados. O algoritmo Árvore de Decisão (AD) não obteve uma performance tão boa no conjunto de teste, mas é um modelo fácil de interpretar, pois é possível visualizar quais decisões são feitas com os dados para realizar cada predição. O Random Forest, por sua vez, que é um método de bagging de árvores de decisão, obteve uma performance melhor no conjunto de teste, mas tornou-se um algoritmo mais complexo de visualizar e interpretar a conclusão de cada predição, embora ainda seja possível.

O K-Nearest Neighbors (KNN) foi o algoritmo que obteve a maior performance no conjunto de treino, porém essa performance não foi replicada no conjunto de validação e teste, indicando um overfitting por parte do classificador. Esse algoritmo é bastante simples e se baseia na distância entre os dados, não conseguindo generalizar a performance para os dados não vistos. O Learning Vector Quantization (LVQ), embora seja uma técnica simples e intuitiva, teve um desempenho mais modesto, pois é sensível à escolha de protótipos iniciais e pode não capturar bem a complexidade dos dados, obtendo a pior performance nos conjuntos avaliados.

O MLP, que é uma rede neural, apresentou uma boa performance, mas esse algoritmo é mais complexo

de ajustar e interpretar, sendo sensível a hiperparâmetros como o número de camadas e neurônios. O Support Vector Machine (SVM) teve uma performance razoável, ele é um classificador eficiente para problemas de classificação binária, mas pode se tornar ineficiente quando lidando com grandes conjuntos de dados ou problemas multiclasses. O algoritmo de Stacking, que combina vários classificadores, teve a melhor performance no conjunto de teste pois se beneficia dos pontos fortes de múltiplos modelos, mas é mais complexo em termos de implementação e tempo de execução.

References

[1] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. 1, pp. 1–30, 2006.

[2] B. Akkaya e N. Çolakoğlu, "Comparison of multi-class classification algorithms on early diagnosis of heart diseases," em *y-BIS 2019 Conference: ISBIS Young Business and Industrial Statisticians Workshop on Recent Advances in Data Science and Business Analytics at: Istanbul, Turkey*, 2019.

[3] S. U. G. A. S. C. K. Mishra P, Pandey CM, "Descriptive statistics and normality tests for statistical data," *Ann Card Anaesth*, 2019.