Trabalho de Conclusão de Curso

Uso de Meta-aprendizado para avaliar a tunagem para o algoritmo Máquina de vetores de suporte

Jadson Crislan Santos Costa
jcsc@ic.ufal.br

Orientador:

Prof. Dr. Bruno Almeida Pimentel

Maceió, Julho de 2022

Jadson Crislan Santos Costa

Uso de Meta-aprendizado para avaliar a tunagem para o algoritmo Máquina de vetores de suporte

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Prof. Dr. Bruno Almeida Pimentel

Maceió, Julho de 2022

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecária Responsável: Helena Cristina Pimentel do Vale – CRB4 - 661

C837u Costa, Jadson Crislan Santos.
Uso de meta-aprendizado para avaliar a tunagem para o algoritmo maquina de vetores de suporte / Jadson Crislan Santos Costa. – Maceió, 2022.
48 f. : il. color.

Orientador: Bruno Almeida Pimentel.
Monografia (Trabalho de Conclusão de Curso em Ciência da Computação) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2022.

Bibliografias: f. 46-48.

1. Aprendizagem de maquina. 2. Hiperparametros. 3. Maquina de vetores de suporte. 4. Inteligência artificial. I. Título.

CDU: 004.8

Agradecimentos

Agradeço ao Professor Bruno Pimentel pela orientação nos anos que trabalhamos juntos, à Professora Roberta Lopes, professor Tiago Cordeiro e Professor Marcus Braga, também agradeço aos professores e técnicos do Instituto de computação que fizeram parte da minha trajetória na graduação.

Agradeço a minha Família e em especial ao meu pai Jailton Costa, minha mãe Josineide Costa, meus irmãos Guilherme e Jawemerson, meus tios Jailson Costa e Cledilma Costa pelos anos que me acolheram em Maceió para que eu pudesse realizar meus estudos. Agradeço a minha namorada Kamila por todo apoio no decorrer do curso, e também aos meus colegas de curso, em especial ao Jardel Costa, Ester de Lima, Thalyssa Monteiro e Samuel de Lima.

Agradeço a Deus pela força e amparo em todos esses anos de graduação.

"Não há lugar para a sabedoria onde não há paciência."

– Agostinus, *Aurelius*

Resumo

Aprendizagem de máquina é um dos campos de estudo da área da Inteligência Artificial, onde faz uso de algoritmos que realizam classificação e agrupamento de dados com o objetivo de extrair conhecimento para resolver um determinado problema. Os algoritmos de Aprendizagem de Máquina utilizam hiperparâmetros para melhorar seu funcionamento, sendo esses hiperparâmetros ajustáveis para se adaptar ao problema. A busca por esses hiperparâmetros pode ser um tanto complexa, principalmente quando for aplicado em bases de dados complexas. O presente trabalho compreende em utilizar a técnica de meta-aprendizado para predizer se existe uma real vantagens de melhorar os hiperparâmetros para o algoritmo de Máquina de vetores de suporte utilizando os algoritmos de Busca Aleatória, Busca em Grade, Algoritmo Genético e Otimização por enxame de partículas. Para mensurar os desempenho dos algoritmos, foi adotados métricas de desempenho de f1-score e tempo de execução, e aplicado teste de hipótese 5x2CV pareado para comparar os algoritmos e Mann-Whitney para comparar as distribuíções de tempo, e com isso, os resultados indicam que o uso do algoritmo de Busca Aleatória tem uma performance satisfatória em bases não complexas, por outro lado, em bases complexas os hiperparâmetros default tem uma melhor eficácia, e para os resultados da utilização dos meta-aprendizes, os resultados mostram que eles obtiveram melhor desempenho em relação ao algoritmo de Linha de Base e melhoram os resultados quando o meta-aprendiz foi tunado.

Palavras-chave: Aprendizagem de máquina, hiperparâmetros, Máquina de Vetores de Suporte

Abstract

Machine learning is one of the fields of study in the area of Artificial Intelligence, which uses algorithms that perform classification and clustering of data to extract knowledge to solve a given problem. Machine Learning algorithms use hyper-parameters to improve their performance, and these hyper-parameters are adjustable to adapt to the problem. The search for these hyper-parameters can be quite complex, especially when applied to complex databases. This work uses the meta-learning technique to predict if there is a real advantage of improving the hyper-parameters for the Support Vector Machine algorithm using the Random Search, Grid Search, Genetic Algorithm, and Particle Swarm Optimization. The results indicate that Random Search algorithm has satisfactory results on non-complex bases, and on complex bases, the default hyper-parameters have better efficiency., and for the results of using meta-learning, the results show that they had the best relationship with the baseline and improve the results when the meta-learning was fine-tuned.

Key-words: Machine Learning, hyperparameters, Suport Vector Machine

Lista de Figuras

Lista de Algoritmos

Conteúdo

1

Introdução

A área da Inteligência Artificial (IA) faz parte do campo de estudo da Ciência da Computação, o desenvolvimento da área ganhou fama nos anos 50 com a publicação do artigo Turing (2009) de Alan Turing, em seu artigo, ele descreve o funcionamento de uma máquina tão inteligente que poderia se passar por um humano sem ser detectado como uma máquina, conhecido como teste de Turing. Ao longo dos anos, diversos cientistas das áreas da computação, estatística, matemática, psicologia, etc, buscam técnicas para aprimoramento de algoritmos capazes de realizar tarefas de forma melhor ou mais rápida/eficiênte do que um ser humano, que para isso, a máquina precise aprender a ser inteligente por meio de algum tipo de aprendizado que, por vezes, demandam de grande quantidade de dados. Com o avanço da *internet* e de seu fácil acesso, os algoritmos de IA são utilizados em muitas aplicações do dia a dia das pessoas, como mapas, redes sociais, sistemas de banco, sistemas médicos, e em meio a múltiplos tipos de problemas, a IA se tornou um grande campo da ciência que engloba subáreas como Aprendizado de máquina, aprendizado profundo, visão computacional, entre outras.

O campo da Aprendizagem de máquina utiliza técnicas para realizar classificação e agrupamento de elementos, utilizando esses grupos para prever, obter informações Mitchell and Mitchell (1997). Os algoritmos de aprendizagem de máquina são usualmente empregados em soluções para problemas de automatização, regressão e classificação, existem diversos algoritmos de aprendizagem de máquina conhecidos, como: Árvore de decisão, Floresta Aleatória, Máquina de Vetores de Suporte, Redes neurais, *K-Means*, *Kernel K-Means*, *Gustafson-Kessel*, *Fuzzy C-Means*, *Possibilistic C-Means*. Essa variedade se origina principalmente pelo viés que cada algoritmo trás Muhammad and Yan (2015) alguns exemplos mais comuns são entropia, distância e probabilidade que tem seu desempenho severamente afetado dependendo da natureza do problema fazendo com que não exista uma solução melhor para todos os tipos de problemas de aprendizado de máquina.

Sendo assim, esses algoritmos utilizam de diferentes técnicas para melhoramento do seu desempenho, pois cada algoritmo utiliza parâmetros, que podem ser adaptados para cada si-

tuação, o algoritmos de Máquina de vetores de suporte por exemplo tem C, Gamma e Kernel. Configurar esses hiperparâmetros requer conhecimento específico, muitas vezes, tentativa e erro. Para isso, predições são realizadas com a utilização de hiperparâmetros para auxiliar nessa busca, porém, essa abordagem de aperfeiçoamento apresenta um alto custo de processamento e tempo, especialmente em base de dados muito grandes, alguns algoritmos conseguem variar entre uma performance medíocre e uma performance de estado da arte apenas com ajustes nos seus hiperparâmetros, além disso existe uma tendência dos algoritmos se tornarem cada vez mais complexos, resultando em um aumento na quantidade de hiperparâmetros Hutter et al. (2015).

1.1 Motivação

As técnicas mais tradicionais que realizam a busca pelos melhores hiperparâmetros são por métodos exaustivos, como a Busca aleatória, que define um conjunto de valores, mínimos e máximos, e o algoritmo busca de forma aleatória o melhor hiperparâmetro. Para o algoritmo Busca em grade é necessário escolher um conjunto de valores para realizar os testes de todas as combinações do conjunto escolhido. Já os algoritmos genéticos criam a população inicial de hiperparâmetros, que são os cromossomos iniciais, e passam por mutações e cruzamentos realizando a evolução dos cromossomos até uma quantidade de gerações definidas. O algoritmo Otimização por enxame de partículas cria um conjunto de partículas aleatórias, onde cada partícula é um hiperparâmetro, assim, as partículas tendem a se direcionar ao melhor ponto, ou seja, para o melhor hiperparâmetro. Existem soluções de meta-aprendizado que busca treinar um modelo de aprendizado de máquina para conseguir generalizar estratégias de escolhas de hiperparâmetros, essa abordagem tem como vantagem o baixo custo de replicação pois dado que o modelo se encontra treinado com boas bases de dados e conseguindo generalizar as regras de aprendizagem, o meta aprendiz consegue replicar a estratégia aprendida com um custo computacional menor do que as outras alternativas.

1.2 Objetivos

Este trabalho tem o objetivo criar um meta-aprendiz que busca predizer se vale a pena tunar ou não os hiperparâmetros para o algoritmo de Máquina de vetores de suporte. Para isso foi coletado uma base de dados para avaliar o algoritmo utilizando os métodos de melhoria dos hiperparâmetros, onde foram utilizados os algoritmos de Busca Aleatória, Busca em Grade, Algoritmo Genético e Otimização por exame de partículas. Para verificar se algum deles obteve um desempenho significativamente melhor do que os hiperparâmetros padrão, é aplicado um teste de hipótese com nível de significância de 0,05. Então é gerado o meta-aprendiz que tem seu treinamento realizado com a meta-data, sendo essa constituída por características das bases

de dados e os resultados dos melhores modelos obtidos por cada busca. Como resultado, será obtido a arquitetura que irá predizer se é necessário realizar a busca pelo melhor hiperparâmetro e para validar esses resultados será utilizado o teste de hipótese *Mann-Whitney*.

1.3 Organização do documento

Após esse capítulo de introdução do trabalho, o trabalho foi organizado da seguinte maneira:

Capítulo 2, onde é apresentado toda a fundamentação utilizada no trabalho, como os princípios da Aprendizagem de máquina, a partir de alguns dos algoritmos e técnicas que foram usadas, o processo de descoberta de conhecimento, os algoritmos usados para a busca dos melhores hiperparâmetros, métricas de avaliação de algoritmos e meta-aprendizado.

No Capítulo 3 é descrito a metodologia usada, relatando a escolha e coleta da base de dados, meta-features, seguido da descrição de como é feito a escolha dos melhores hiperparâmetros, treinamento dos meta-aprendizes e o teste de hipótese que foi utilizado.

O Capítulo 4 apresenta os resultados e discussões dos modelos e testes que foram realizados, e para uma melhor análise, os resultados são divididos em desempenho dos métodos de busca em bases complexas (grandes e pequenas) e bases não complexas (grandes e pequenas). Os resultados dos meta-aprendizes também foram divididos pelas meta-features usadas, que foram *Statistical*, *inf_theory* e *general_meta*.

O Capítulo 5 traz por fim a conclusão dos resultados mencionando possíveis causas para os resultados, limitações e contribuições do trabalho.

2

Fundamentação Teórica

2.1 Aprendizado de máquina

Aprendizado de máquina (AM) é a área de pesquisa que possibilita que computadores desenvolvam a habilidade de aprender sem ser expressamente programado Samuel (1959). Essa campo de pesquisa faz parte da área de Inteligência Artificial de uma forma mais afunilada, enquanto a IA pode ser definida de um modo mais amplo da Ciência da Computação, AM é uma vertente mais específica, onde busca treinar a máquina a aprender com dados. A abstração do conhecimento é dado por representações matemáticas e uma série de avaliações do modelo criado pelo algoritmo para avaliar seus resultados.

AM busca resolver uma série de problemas que podem ser categorizados em quatro tipos: Classificação, CLusterização, Regressão e Otimização. Este trabalho aborda os problemas de classificação que será tratado com mais detalhes nesta seção. Problemas de classificação podem ser resolvidos com a aplicação de algum dos diversos algoritmos de AM, para tal escolha, é preciso levar em consideração o tipo do problema e seguir as etapas necessárias para a aplicação do algoritmo. Os algoritmos de AM podem ser classificados em três partes: Aprendizado supervisionado, Aprendizado não supervisionado e Aprendizado por reforço, a seguir é apresentado detalhadamente cada um deles.

2.1.1 Aprendizado supervisionado

No Aprendizado supervisionado a ação é preditiva, onde é preciso fornecer o modelo que o algoritmo deve fazer, fornecer suas entradas e saídas desejadas, assim, busca aprender uma função onde associa um dado à uma determinada classe, como por exemplo, definir se um paciente tem ou não uma doença, nesse caso, os algoritmos buscam solucionar problemas de classificação. Então, se o problema for associado em encontrar algum número, como por exemplo, a previsão da temperatura em uma região, nesse caso o problema é chamado de regressão.

Os algoritmos mais comuns dessa técnica são: *k-Nearest Neighbors*, Máquina de vetores de suporte, Regressão Linear, regressão logística, Árvore de decisão e Floresta Aleatória.

O aprendizado supervisionado é o método de aprendizado mais popular em AM e que consegue ter um melhor resultado se tornando metodologia dominante no campo de AM, as etapas para construção de algoritmos podem ser simplificada como treinar, testar e otimizar o algoritmo para a base de dados fornecida Nasteski (2017)

Máquina de vetores de suporte (Support Vector Machines)

O algoritmo de Máquina de vetores de suporte - MVS é uma técnica em que realiza a classificação binária ou regressão em um conjunto de dados, ele busca gerar o melhor Hiperplano que divide a base de dados em dois grupos. Para problemas de classificação ou que contenha o máximo de pontos, problemas de regressão, nesse caso, para obter esse hiperplano o algoritmo inicialmente tenta classificar todos os pontos corretamente com o hiperplano e depois otimiza as margens para assim obter o melhor hiperplano dado o problema.

O Teorema de Cover descrito em Haykin (2010) afirma que *"Um problema complexo de classificação de padrões, lançado em um espaço de alta dimensão não linearmente, tem mais probabilidade de ser linearmente separável do que em um espaço de baixa dimensão, desde que o espaço não seja densamente povoado."*, este teorema permite que o MVS classifique dados que não são linearmente separáveis lançando esses dados em um espaço de dimensão maior do que o original, para fazer essa transformação não linear é preciso definir uma função de transformação não linear que será chamada de kernel, a Figura 2.1 trás uma representação gráfica do teorema.

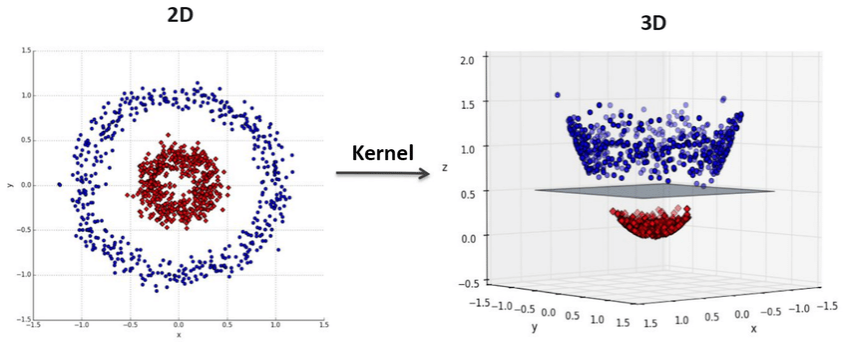


Figura 2.1: Representação do Kernel, Hachimi et al. (2020)

A Figura 2.2 trás uma representação visual do Kernel em uma base de dados. Entretanto, existe diferentes tipos de Kernels, a seguir é citados alguns deles:

- Kernel Linear (*Linear Kernel*): O Kernel linear é o mais simples dos kernels e devido a sua simplicidade consegue ter um baixo custo computacional que o faz ser bastante utilizado em problemas que possui uma alta dimensão, onde aumentar a dimensão não se

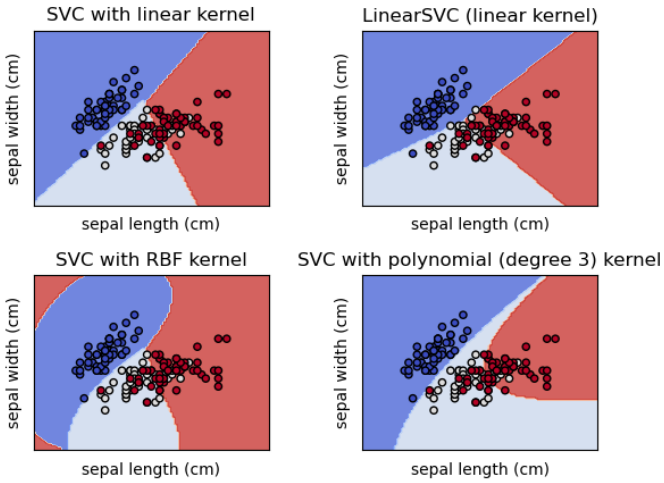


Figura 2.2: Tipos de Kernels Pedregosa et al. (2011)

faz necessário para melhorar a classificação. Essas características fazem esse kernel ser muito popular para a classificação de textos.

$$F(x,x_i) = x.x^T \tag{2.1}$$

- Kernel Polinomial (*Polynomial Kernel*): O Kernel polinomial utiliza um hiperparâmetro a mais para realizar a otimização, tornando esse tipo de Kernel mais complexo, o grau do polinômio podendo ir até o infinito Hsu et al. (2003), contudo, esse kernel pode ser utilizado para Processamento de LInguagem Natural (NLP) **?**

$$F(x,x_i) = (1 + x.x_i^T)^d \tag{2.2}$$

- (Função de base radial gaussiana (RBF)): O RBF é o kernel padrão para a maioria dos problemas devido a sua capacidade de aumentar a dimensão dos dados, tornando mais fácil de separar as classes. O Kernel Linear é um caso particular do RBF Kernel Keerthi and Lin (2003) sendo assim o desempenho do RBF é igual ou superior ao Kernel Linear.

$$F(x,x_i) = e^{-\gamma||x-x_i||^2} \tag{2.3}$$

- Sigmoid Kernel O *Sigmoid* Kernel vem da área de redes neurais e torna o MVS com um funcionamento similar a um perceptron de duas camadas que usa *Sigmoid* como função de ativação, tem um desempenho similar ao Kernel RBF para intervalos específicos Lin and Lin (2003), porém, é mais complexo e tem uma tendência maior de ter problemas de

sobreajuste do que o Kernel RBF.

$$F(x_i, x_j) = tanh(\gamma x_i^T x_j + r) \tag{2.4}$$

Como o algoritmo de MVS busca realizar a classificação binária, é necessário modelar o problema para se adaptar ao tipo de metodologia aplicada pelo MVS. a Figura 2.3 mostra a diferença entre classificações binárias e multiclasses. As duas estratégias mais utilizadas são: *one-Vs-one* e *one-Vs-res*.

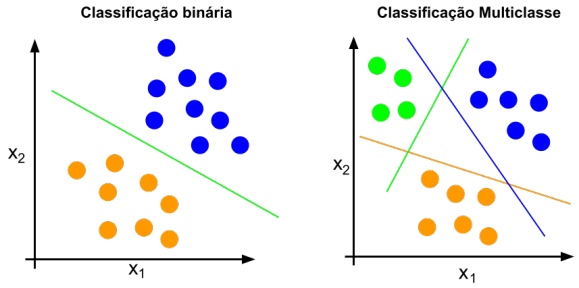


Figura 2.3: Classificação binária vs multiclasse

A técnica *one-Vs-one* consiste em gerar um modelo de classificação binária para cada par de classes, o resultado final é dado pelo voto majoritário pelos modelos, a técnica é demonstrada pela Figura 2.4 . O número de modelos é dado pela fórmula:

$$\frac{NumClasses * (NumClasses - 1)}{2} \tag{2.5}$$

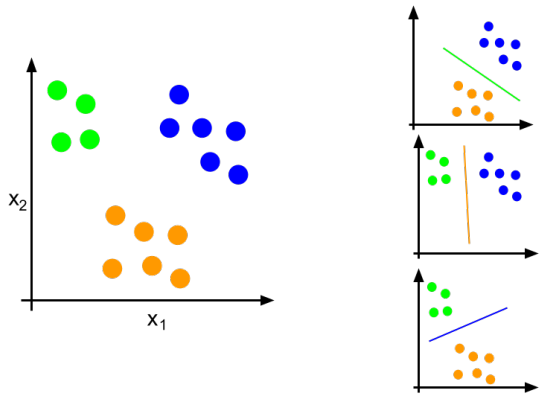


Figura 2.4: One-vs-One

A técnica *one-Vs-res* consiste em realizar a classificação binária entre uma das classes com o restante das classes, como mostrado na Figura 2.5

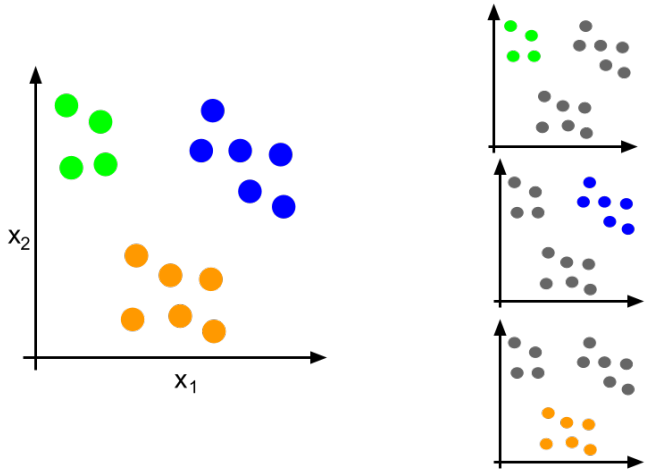


Figura 2.5: One-vs-Res

Árvore de decisão (Decision Tree)

Na computação, uma árvore é uma estrutura de dados que armazenam algum tipo de informação, chamados de nós, e toda árvore tem um nó com nível mais hierárquico chamado raiz, que é o ponto de partida da árvore e ligações com outros elementos, denominados filhos, um nó que não possui filhos é chamado de folha. Uma árvore de decisão é uma árvore que armazena regras em seus nós, e as folhas armazena uma classe Quinlan (1996), essa decisão é tomada através do percurso entre os vários nós da árvore até chegar em uma folha.

Árvores de decisão tem uma tendência a se sobreajustar a bases de dados, então em geral árvores pequenas são preferíveis pois são mais fáceis de entender, além de apresentar uma maior resistência à sobreajustes. Uma abordagem utilizada é deixar a árvore crescer o máximo possível e depois começar eliminar sub-árvores que pouco contribuem para a performance do algoritmo Quinlan (1996). Uma das vantagens da árvore de decisão é sua fácil interpretação e resistência a ruídos, principalmente de dados faltantes na base de dados.

Floresta Aleatória (Random Forest)

O algoritmo de Floresta Aleatória pode ser utilizado para classificações e regressões, seu funcionamento é de criar um conjunto de árvores de decisão com o objetivo de melhorar a acurácia e deixar a predição mais estável, para isso são criadas várias árvores de decisões usando subconjuntos aleatórios da base de dados original, depois que essas árvores são geradas cada uma dá seu voto para os dados oferecidos em uma classe e a classe mais votada entre todas as árvores será a classe escolhida Breiman (2001).

A Floresta aleatória pode ser utilizada em uma ampla variedade de problemas, quando comparado com a árvore de decisão experimentos demonstraram que a floresta aleatória obtém um melhor resultado geral do que uma árvore de decisão Ali et al. (2012), além de oferecer uma maior resistência à sobreajuste, maior resistência a pontos fora da linha e menor número de

hiperparâmetros.

Naive bayes

O algoritmo Naive bayes é baseado no Teorema de Bayes, criado por Thomas Bayes Bayes (1763), que tem como princípio a suposição de independência entre as variáveis do problema, então, realiza uma classificação probabilística de observações, caracterizando-as em classes pré-definidas.

Apesar de simplificar sua análise da base de dados, pois pressupõe a independência das características da base de dados o Naive bayes consegue na prática um resultado que em muita das vezes compete com outros métodos de classificação mais sofisticados. Alguns experimentos mostram que o Naive Bayes consegue ter um bom desempenho, onde esse desempenho está diretamente relacionado com a quantidade de informações perdidas sobre as classes por causa do pressuposto de independência de características Rish et al. (2001).

2.1.2 Aprendizado não supervisionado

No Aprendizado supervisionado a ação é descritiva onde é necessário apenas informar a base de dados, então o algoritmo irá encontrar semelhança entre os dados e assim agrupá-los. Esse tipo de aprendizado engloba os problemas de clusterização, onde o próprio modelo julga os dados de entrada e com base em semelhanças, cria grupos para eles. Algoritmos usados para problemas de Clusterização: *K-Means*, *DBSCAN*, *Hierarchical Cluster Analysis (HCA)*. Detecção de anomalia: *One-class MVS*, *Isolation Forest*, Redução de dimensionalidade: *PCA*, *t-SNE*. Em geral o objetivo do aprendizao não supervisionado é que o algoritmo consiga por se próprio sem nenhum rotulo na base de dados ou um sistema de recompensa e punição conseguir criar representações com os dados de entrada que podem ser utlizados para tomada de decisão, predizer futura entradas, melhorar a descrição da base de dados para outro algoritmo etc, o aprendizado não supervisionado é bastante utilizado para detectar ruídos nas bases de dados, redução de dimensionalidade e agrupamento de dados Ghahramani (2003).

2.1.3 Aprendizado por reforço

O aprendizado por reforço segue um treinamento do agente que toma decisões baseadas no ambiente, que para cada situação o algoritmo utiliza tentativa e erro para encontrar a solução, recebendo uma punição ou recompensa pelo resultado, dessa forma, o algoritmo busca maximizar as recompensas. A execução desses tipos de algoritmo consiste em inserir o agente em um ambiente, a cada novo passo o agente recebe uma entrada e o estado inicial do ambiente, nesse passo o agente toma uma decisão, gerando a saída, essa saída então altera o ambiente, e ao passo que o agente recebe as punições ou recompensa, ele irá aprender a tomar a melhor decisão nos próximos passos Kaelbling et al. (1996). Exemplos desse tipo de aprendizado por

reforço são carros automáticos, software do mercado financeiro, sistema de recomendações de vídeos, Bots de jogos.

2.1.4 Processo de descoberta de conhecimento

Com aprendizado de máquina é possível extrair conhecimento de base de dados grande o suficiente em que seria inviável um ser humano fazer uma análise manual, porém, antes de aplicar algum dos algoritmos de AM, é importante seguir um conceito bem consolidado na literatura, conhecido como KDD, Fayyad et al. (1996), onde essa metodologia considera 5 passos para realização da extração do conhecimento, demonstrado na Figura 2.6.

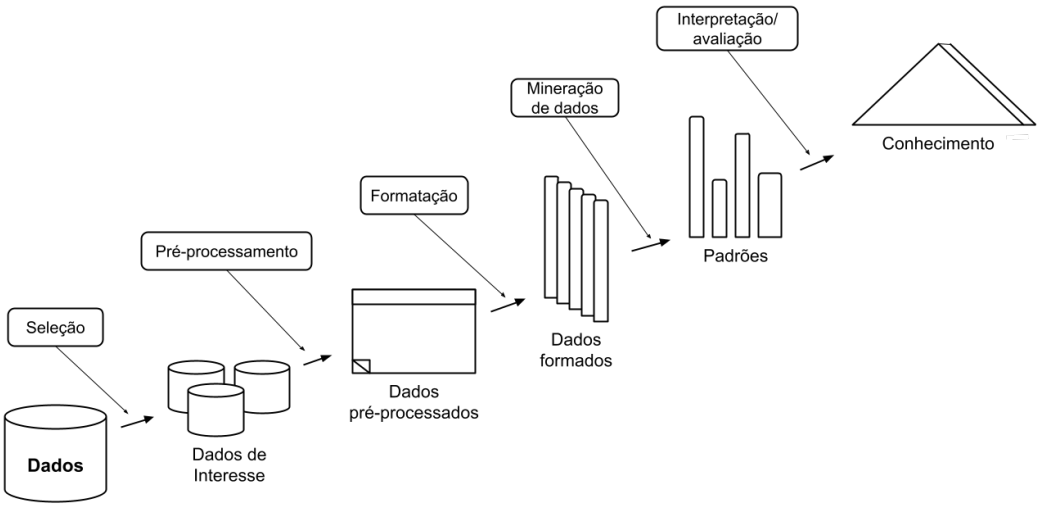


Figura 2.6: Processo do KDD

Ao obter a base de dados é necessário realizar o pré-processamento, que consiste em uma limpeza e enriquecimento da base, como a remoção de valores ausentes, valores com erros, ou valores faltantes, podendo ser uma linha da tabela por exemplo, ou uma coluna que não será útil para o aprendizado.

A transformação dos dados consiste em procurar atributos que sejam realmente úteis para o modelo levando em consideração o problema abordado. O padrão que os dados se encontram pode dificultar o modelo, pois podem conter textos, então nesta etapa os dados serão adequados de forma que a máquina possa processar e sem perda de informação. Algumas técnicas comuns para transformações de dados é são normalização, a discretização de atributos quantitativos e transformações de atributos qualitativos em quantitativos.

A fase de mineração de dados é onde se aplica algum dos algoritmos de AM, nessa fase, já com a base de dados pré-processadas, eles são divididos em partes, onde uma parte é usada para o treinamento do modelo e a outra para realizar o teste do modelo, então nessa parte é usado

apenas os dados que serão treinados. É preciso escolher algum algoritmo de classificação, agrupamento ou de regressão.

Por fim, com o modelo finalizado, o próprio é testado com a parte da base separada para isso, o método de separação da base em parte de teste e parte de treino pode ser feito de diferentes formas, como o holdout, subamostragem aleatória e a validação cruzada, neste trabalho foi realizado a validação cruzada, que consiste em dividir os dados em k partes de mesmo tamanho, onde os treinamentos são executados nas k-1 partes e utilizado 1 parte para o teste. E após os obter os resultados dos testes, com a acurácia do modelo, é realizado as interpretações do conhecimento, para isso é importante ter o conhecimento do domínio do problema.

2.2 Busca em grade

Busca em grade (*Grid Search*) é uma técnica usada para otimização de hiperparâmetros, onde o algoritmo recebe um conjunto de valores de hiperparâmetros e testa todas as combinações dentro do conjunto dos hiperparâmetros selecionados, ao final o algoritmo lista o desempenho de cada combinação e mostra a melhor escolha. Essa técnica atribui igual importância para todos os hiperparâmetros, contudo, em diversos cenários alguns hiperparâmetros podem se sobressair em relação a performance do que aos demais, além disso o algoritmo é fortemente impactado pelo conhecimento de escolher o melhor conjunto de hiperparâmetros para serem testados, visto que alguns algoritmos podem mudar drasticamente esse conjunto para cada base de dados. Busca em grade junto com a busca manual são os métodos mais populares de busca de otimização, além de conseguir ter bom desempenho em espaços de 1 ou 2 dimensões Bergstra and Bengio (2012).

2.3 Busca aleatória

A técnica de Busca aleatória (*Random Search*) consiste em selecionar valores aleatórios dentro de um intervalo determinado, testar esses valores selecionados aleatoriamente e avaliá-los Bergstra and Bengio (2012). Essa técnica tem demonstrado um ótimo desempenho com relativo baixo custo computacional e desempenho semelhante a buscas com heurísticas como o Algoritmo Genético e o PSO tendo um custo computacional menor. Uma grande vantagem dessa técnica em relação ao Busca em grade é que ela quase não é afetada pela falta de conhecimento prévio de escolha de melhor hiperparâmetros, além disso ela consegue com o mesmo número de busca da Busca em grade explorar um número maior de valores para cada hiperparâmetro criando um espaço de busca mais amplo.

2.4 Algoritmo Genético

Algoritmo Genético (*Genetic Algorithm*) é uma técnica de busca e otimização inspirada nos princípios Darwinianos da seleção natural e reprodução genética Pacheco et al. (1999), na qual essa teoria afirma que indivíduos mais qualificados tem a maior probabilidade de se reproduzirem, assim, indivíduos com mais descendentes têm mais chances de preservar seus códigos genéticos por mais gerações, onde esses códigos genéticos compõem a identidade de cada indivíduo, representados por cromossomos, a tabela 2.4 apresenta uma relação das características dos componentes da biologia evolutiva, um pseudocódigo do algoritmos genéticos é apresentado em 2.1.

Biologia	Algoritmo genético
Cromossomos	Binário, vetor, etc.
Genes	Característica do problema
Alelo	valor da característica
Loco	Posição na palavra, vetor
Genótipo	Estrutura
Fenótipo	Estrutura submetida ao problema
Indivíduo	Solução

Tabela 2.1: componentes da biologia evolutiva Vs algoritmos genéticos

Algoritmo 2.1 Pseudo Código - Algoritmo Genético

```
t <- 0
Gerar população inicial P(t)
Avaliar P(t)
While not CriterioDeParada do
for i <- 1 to N/2 do
Selecionar dois indivíduos de P(t)
Aplicar crossover aos dois indivíduos com probabilidade pc
Mutar os novos indivíduos gerados com probabilidade pm
Inserir os novos indivíduos em P(t+1)
end for
t <- t + 1
end While
```

No contexto de computação genética, um cromossomo é um conjunto de dados que representa uma das soluções que se espera encontrar, a execução do algoritmo realiza o processo evolucionário do cromossomo, esse processo infere em avaliar, selecionar, recombinar (crossover) e realizar mutações.

A avaliação é realizada através de uma função heurística que melhor representa o objetivo do problema e que possa medir a aptidão de cada indivíduo na população da geração que se encontra. Essa função não precisa ter seu comportamento conhecido, pois ela irá apenas dirigir o processo de busca do algoritmo.

A seleção se baseia nos indivíduos mais aptos (melhores avaliados pela função heurística), assim, esses indivíduos têm uma probabilidade maior de se reproduzirem. Existe diferentes formas de seleção, como: Seleção por roleta, seleção por fitness, seleção por torneio e seleção por ranking. Nesse trabalho foi usado a seleção por torneio, que consiste em selecionar aleatoriamente dois indivíduos e seleciona o mais apto dos dois.

O cruzamento (*crossover*) é o processo de reprodução dos indivíduos, onde seleciona os cromossomos pai para gerar os cromossomos filhos que serão gerados com base nos genes dos cromossomos pais. A divisão dos cromossomos pai podem ser realizadas de algumas formas, como cruzamento de um ponto, onde os pais serão divididos em uma posição aleatória e os filhos serão uma combinação da primeira parte do pai[1] com a segunda parte do pai[2], o cruzamento de dois ponto corta os pais em duas parte em uma posição aleatória e recombina as partes geradas.

As mutações tem o objetivo de diversificar a geração e evitar que o algoritmo direcione muito cedo para algum mínimo local, modificando um ou mais genes de um cromossomo. Todos esses passos descritos do algoritmo genético são executado por uma certa quantidade de vezes, chamada de gerações, e após isso, a população de cromossomos conta com indivíduos mais aptos, A Figura 2.7 apresenta uma representação de uma geração.

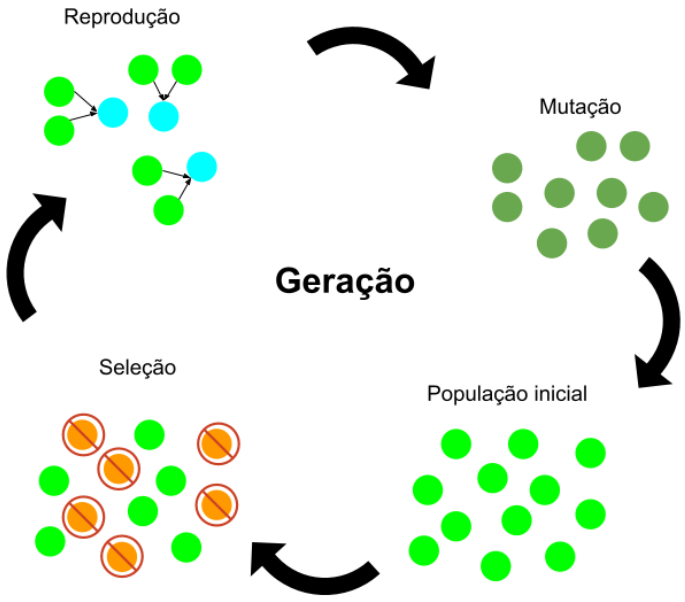


Figura 2.7: Representação de uma geração

2.5 Otimização por enxame de partículas (OEP)

A otimização por enxame de partículas (*Particle Swarm Optimization*) é um algoritmo bioinspirado que se baseia no comportamento de enxame de animais, criado Kennedy and Eberhart (1995). A figura 2.8 representa visualmente o comportamento das partículas e um pseudocódigo apresentado em 2.2.

Algoritmo 2.2 Pseudo Código - PSO

Crie e inicialize um PSO n-dimensional: S
Repita
Repita cada partícula i
Se f(x$_i$) $< f(y_i)$
Então$y_i = x_i$
Se $f(y_i) < f(y)$
Então$y = y_i$

Diferente do Algoritmo Genético, o OEP não descarta nenhuma partícula enquanto avança o número de interações, a abordagem do OEP concede que a melhor partícula faça o papel de líder, denominada como gbest, então ela exerce uma influência sobre todas partículas, guiando-as para mais próximo de si com a intenção de levar as partículas para uma região onde tende a ter melhores soluções. Além disso, cada partícula armazena a informação do seu melhor resultado individual, denominada como pbest. Para evitar que todas as partículas congruíssem juntas com a melhor partícula para um mínimo ou máximo local, os parâmetros C2 da equação controlam o peso que a melhor partícula avaliada tem de influenciar as outras partículas, e o parâmetro C1 controla a influência do melhor resultado que cada partícula exerce sobre seu próprio movimento, e com esses dois parâmetros é possível controlar qual a prioridade as partículas devem considerar para se movimentar, a Equação mostrada a seguir é utilizado para essa técnica.

$$\mathbf{V}_{t+1} = w\mathbf{V}_t + c_1 r1(\mathbf{p}best - \mathbf{p}) + c_2 r2(\mathbf{g}best - \mathbf{p})(a) \tag{2.6}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_t(b) \tag{2.7}$$

Onde as variáveis da equação são:

- v t+1: Velocidade atualizada da partícula
- x t+1: Posição atualizada da partícula
- w: Coeficiênte de inércia
- p best: Melhor posição conhecida da partícula
- g best: Melhor posição conhecida dentre todas as partículas

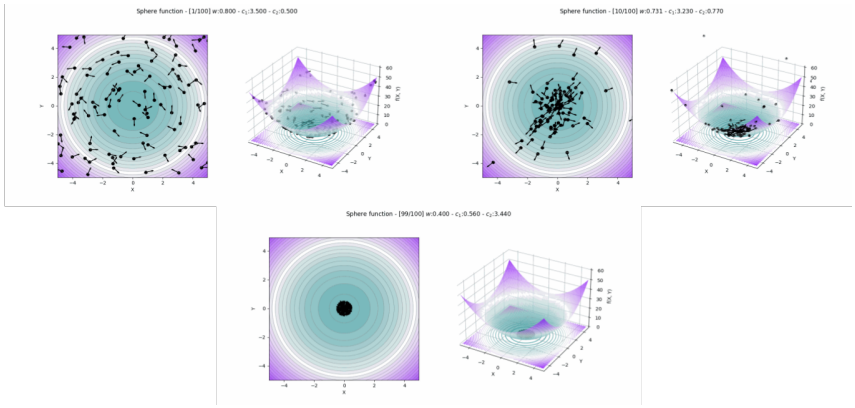


Figura 2.8: Comportamento das particulas Thevenot (2020)

- c1: Constantes de aceleração referentes ao melhor individual
- c2: Constantes de aceleração referentes ao melhor global
- r1 e r2: Números aleatórios extraídos do intervalo [0,1]

2.6 Métricas de avaliação

Para conseguir mensurar o desempenho de algoritmos de AM se faz necessário o uso de métricas de avaliação que possuem métodos e conceitos para avaliar a performance de um algoritmo em uma certa base de dados, essas métricas são separadas em dois grandes grupos, o primeiro deles são as métricas de classificação que mensura apenas o desempenho do algoritmo em identificar as classes da base de dados, o segundo grupo são as métricas de regressão que mensuram o quão perto o valor predito está próximo do real valor obtido Handelman et al. (2019).

2.6.1 Metricas de Classificação

Problemas de classificação consiste em rotular um conjunto de dados em um dos grupos pré-definidos. Com problemas de classificação binária o algoritmo classifica o dado em dois possíveis grupos, em geral, será obtido uma das duas respostas, positivo (P) ou negativo (N), portanto, a avaliação do modelo será feito por meio de comparações entre o que o modelo predisse e da classe verdadeira do dado. Para visualizar de forma mais simples a performance do modelo, é realizado a montagem de uma matriz de confusão, representado pela Figura 2.9, com dados fictícios, na qual dispõe da quantidade de exemplos de cada grupo como: false positivo (FP), falso negativo (FN), verdadeiro positivo (TP) e verdadeiro negativo (TN). Com a matriz de confusão bem definida para o problema é aplicado diferentes formas de realizar a avaliação, sendo elas:

- **Acurácia**: A Acurácia diz quantos dos exemplos foram classificados corretamente, independente da classe, essa métrica é uma boa indicação geral de como o modelo performou.

		Predito	
Real		Positivo	Negativo
	Positivo	80 %	20 %
	Negativo	9 %	91 %

Figura 2.9: Exemplo de matriz de confusão

A acurácia é definida pela razão entre o que o modelo acertou e todos os exemplos, como pode ser visualizado pela fórmula:

$$\textbf{Acurácia} = \frac{(VP)+(VP)}{(VP)+(VP)+(FP)+(FN)} \tag{2.8}$$

Porém, para algumas situações a acurácia pode não revelar uma boa visão da performance do modelo, como por exemplo, se quiser saber a acurácia de um modelo que detecta uma doença em uma base de dados com 1000 exemplos, sendo 990 pessoas sem doença e 10 com doença e o modelo classificar todos como sem doença, ele teria uma acurácia de 99%, o modelo então teria uma performance muito boa, o que não é verdade.

- **Precisão**: A precisão dá mais ênfase em problemas que falsos positivos são considerados mais prejudiciais que os falsos negativos, dado pela fórmula:

$$\textbf{Precisão} = \frac{(VP)}{(VP)+(FP)} \tag{2.9}$$

- **Revocação**: A revocação realiza de forma contrária à precisão, ela dá mais ênfase aos erros por falsos positivos, definido pela fórmula:

$$\textbf{Revocação} = \frac{(VP)}{(VP)+(FN)} \tag{2.10}$$

- **f1-Score**: A métrica *f1-Score* considera tanto a precisão quanto a revocação. É definida pela média harmônica entre as duas:

$$\textbf{F1-Score} = 2 * \frac{Precision * recall}{Precision + recall} \tag{2.11}$$

- **Curva ROC**: Essa métrica exibe quão bom o modelo pode diferenciar entre duas classes, ela é realizada medindo a taxa de falso positivo e a taxa de verdadeiro positivo, como na

fórmula:

$$\textbf{Taxa (VP)} = \frac{(VP)}{(VP)+(FN)} \textbf{Taxa (FP)} = \frac{(FP)}{(FP)+(VN)} \tag{2.12}$$

2.6.2 Metricas de Regressão

Problemas de regressão consistem em definir uma estimativa entre variáveis, o modelo busca uma equação matemática que melhor representa a relação entre as variáveis. Com o modelo pronto, é preciso realizar uma avaliação dele, que pode ser de diferentes formas:

- **R-Quadrado**: Essa métrica mostra o quão próximo as medidas reais estão do modelo criado, essa métrica é usada para expressar a quantidade de variância dos dados que é explicado pelo modelo. O valor varia entre 0 e 1 e geralmente é representado em porcentagem, sendo definida pela fórmula:

$$R^2 = 1 - \frac{varinciaResidual}{varinciaTotal} = 1 - \frac{\sum_{i=1}^{n}(y_i - y_i)^2}{\sum_{i=1}^{n}(y_i - y)^2} \tag{2.13}$$

- **Erro Quadrático Médio (MSE)**: Essa métrica consiste na média do erro das previsões ao quadrado. Com essa métrica, as predições muito longe do real valor aumentam o valor da medida muito fácil, dessa forma essa métrica de avaliação é eficaz para problemas que erros grandes não são tolerados. A métrica é definida pela fórmula:

$$(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i)^2 \tag{2.14}$$

- **Erro Percentual Absoluto Médio (MAPE)**: Essa métrica é a diferença do valor predito e do valor real, onde resulta em uma porcentagem, definido pela fórmula:

$$(MAPE) = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{(y_i - y_i)}{y_i}\right| \tag{2.15}$$

2.7 Meta-Aprendizado

Meta-Aprendizado é o campo de AM que estuda como os algoritmos podem ter seu desempenho melhorado, onde tem como objetivo criar modelos que possam criar suas estratégias de aprendizado através de outros modelos relacionando suas performances, a Figura 2.10 trás sua arquitetura baseada em Brazdil et al. (2008). Para realizar esse aprimoramento, essa técnica utiliza o conceito de meta-data, que é a coleção de meta-features e a performance de algoritmos avaliados.

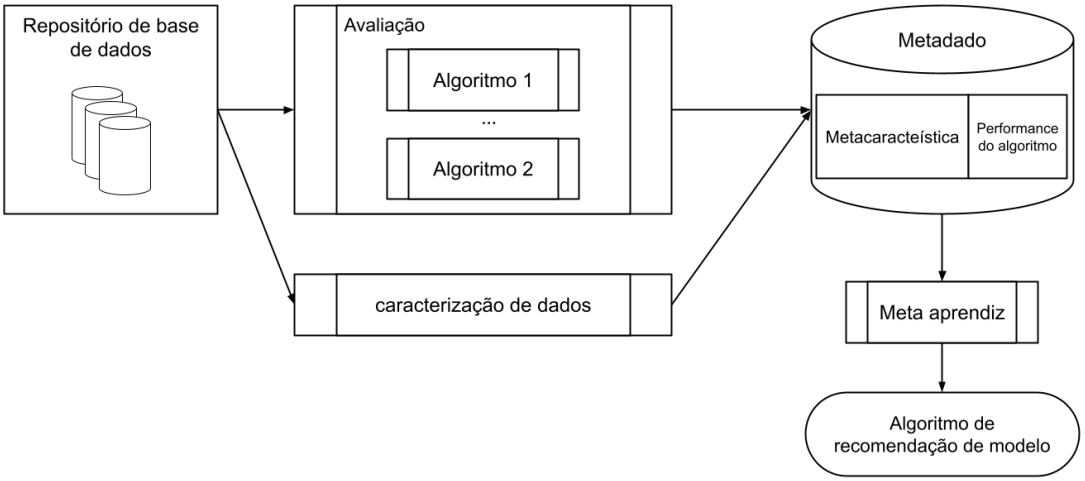


Figura 2.10: Meta-Aprendizado

2.7.1 Meta-feature

Meta-feature é definida na literatura como uma forma de descrever a base de dados Brazdil et al. (2008) Rivolli et al. (2018), existem grupos diferentes de meta-features baseado em diversos princípios de aprendizado e vieses dos algoritmos, como número de classe, distância entre as classes, número de instâncias entre outros princípios que os algoritmos usam para aprender.

Para este trabalho foi utilizado a Biblioteca *pymfe: Python Meta-Feature Extractor* Alcobaça et al. (2020) que permite a extração sistemática de meta-características para meta-aprendizado, fornece um conjunto de recursos em linguagem *Python* podendo definir os hiperparâmetros e medição de tempo decorrido.

2.7.2 Meta-data

Para conseguir treinar os meta-aprendizes, além das informações sobre as bases de dados em forma de meta-feature utilizadas é preciso também das informações do ranking de desempenho dos modelos testados e a junção dessas duas base de dados, denominada meta-data, com elas, serão utilizada para treinar os meta-aprendizes.

A qualidade da meta-data é vital para a criação de um bom meta-aprendiz, por isso a meta-data deve representar uma boa visão geral do problema a ser solucionado, caso contrário, o meta-aprendiz vai ser incapaz de fazer boas previsões. O uso de base de dados sintéticas para a criação de Meta-data é desencorajada pois essas bases tendem a não representar bem as bases de dados de problema reais tornando os meta-aprendizes treinado em uma Meta-data dessas incapaz de ser utilizado para problemas do mundo real.

2.7.3 Meta-aprendiz

O meta-aprendiz é um algoritmo de aprendizado de máquina que é treinado com uma Meta-data que contém as descrições das bases de dados em forma de meta-features e o desempenho dos algoritmos na base de dados descrita pela meta-feature. Os meta-aprendizes podem ser utilizados para recomendar tanto algoritmos quanto hiperparâmetros para bases de dados apenas extraindo suas meta-features em um processo que demora menos do que os tradicionais processos de tentativa e erro ou busca heurística, assim o propósito do meta-aprendiz é aprender como os algoritmos avaliados se comportam a partir das meta-features extraídas das bases de dados.

3

Metodologia

Toda a execução do experimento foi realizado em uma máquinas com as seguintes especificações:

- Processador: AMD *Ryzen* 5 5600G
- Memória RAM: 24 GB, 3600mhz
- Sistema Operacional: *Windows* 11
- SSD: 1 TB, Leituras: 3500MB/s, Gravações: 3000MB/s

A linguagem de programação utilizada foi *Python*, juntamente com a biblioteca *Scikit-learn* e foi habilitada o processamento paralelo utilizando n_jobs = 5 para realizar a validação cruzada.

3.1 Base de dados

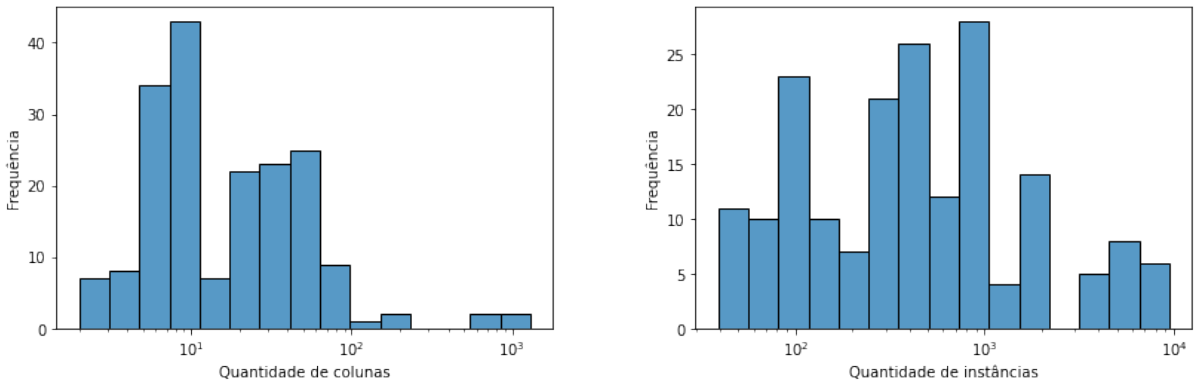
Para esse estudo foram coletadas 185 bases de dados do site *Open-ml* Feurer et al. (2019), a lista com seus nomes e Ids se encontra na Tabela 3.1 e a descrição de quantidade de colunas e instância da base da base de dados é apresentado pelo histograma da Figura 3.1. Para a escolha da base de dados foi pretendido escolher bases com maior número de instâncias e colunas, visto que na literatura base as bases de dados testadas eram relativamente pequenas Mantovani et al. (2015). Como técnica de pre-processamento, foi utilizado padronização que consiste em remover a média e dividir pelo desvio padrão das amostras, resultando em uma base com média igual a 0 e desvio padrão igual a 1.

3.2 Meta-features

As Meta-features utilizadas foram obtidas da biblioteca *Pymfe* para treinar os meta-aprendizes. Seu grupo, nome e descrição são demonstrados na Tabela 3.2.

Id	Nome	Id	Nome	Id	Nome	Id	Nome
3	kr-vs-kp	11	balance-scale	12	mfeat-factors	14	mfeat-fourier
16	mfeat-karhunen	18	mfeat-morphological	22	mfeat-zernike	23	cmc
28	optdigits	30	page-blocks	31	credit-g	37	diabetes
44	spambase	54	vehicle	59	ionosphere	60	waveform-5000
61	iris	182	satimage	187	wine	307	vowel
329	hayes-roth	337	SPECTF	377	synthetic_control	458	analcatdata_authorship
472	lupus	679	rmftsa_sleepdata	683	sleuth_ex2015	694	diggle_table_a2
715	fri_c3_1000_25	716	fri_c3_100_50	717	rmftsa_ladata	718	fri_c4_1000_100
721	pwLinear	723	fri_c4_1000_25	728	analcatdata_supreme	729	visualizing_slope
730	fri_c1_250_5	731	baskball	732	fri_c0_250_50	733	machine_cpu
736	visualizing_environmental	743	fri_c1_1000_5	746	fri_c1_250_25	749	fri_c3_500_5
750	pm10	751	fri_c4_1000_10	754	fri_c0_100_5	755	sleuth_ex1605
758	analcatdata_election2000	759	analcatdata_olympic2000	762	fri_c2_100_10	769	fri_c1_250_50
770	strikes	772	quake	775	fri_c2_100_25	776	fri_c0_250_5
777	sleuth_ex1714	779	fri_c1_500_25	780	rabe_265	782	rabe_266
783	fri_c3_100_10	787	witmer_census_1980	793	fri_c3_250_10	794	fri_c2_250_25
797	fri_c4_1000_50	800	pyrim	803	delta_ailerons	805	fri_c4_500_50
806	fri_c3_1000_50	807	kin8nm	808	fri_c0_100_10	812	fri_c1_100_25
815	chscase_vine1	816	puma8NH	820	chatfield_4	824	fri_c1_500_10
827	disclosure_x_noise	829	fri_c1_100_5	832	fri_c3_250_25	833	bank32nh
837	fri_c1_1000_50	838	fri_c4_500_25	841	stock	845	fri_c0_1000_10
847	wind	849	fri_c0_1000_25	850	fri_c0_100_50	855	fri_c4_500_10
857	bolts	859	analcatdata_gviolence	860	vinnie	866	fri_c2_1000_50
870	fri_c1_500_5	873	fri_c3_250_50	874	rabe_131	876	fri_c1_100_50
877	fri_c2_250_50	878	fri_c4_100_10	879	fri_c2_500_25	880	mu284
884	fri_c0_500_5	885	transplant	886	no2	888	fri_c0_500_50
889	fri_c0_100_25	892	sleuth_case1201	893	visualizing_hamster	894	rabe_148
896	fri_c3_500_25	900	chscase_census6	903	fri_c2_1000_25	904	fri_c0_1000_50
905	chscase_adopt	906	chscase_census5	907	chscase_census4	908	chscase_census3
910	fri_c1_1000_10	911	fri_c2_250_5	912	fri_c2_1000_5	913	fri_c2_1000_10
914	balloon	916	fri_c3_100_5	917	fri_c1_1000_25	919	rabe_166
920	fri_c2_500_50	922	fri_c2_100_50	925	visualizing_galaxy	926	fri_c0_500_25
927	hutsof99_child_witness	929	rabe_176	932	fri_c4_100_50	933	fri_c4_250_25
935	fri_c1_250_10	936	fri_c3_500_10	943	fri_c0_500_10	946	visualizing_ethanol
962	mfeat-morphological	969	iris	970	analcatdata_authorship	971	mfeat-fourier
974	hayes-roth	978	mfeat-factors	980	optdigits	995	mfeat-zernike
996	prnn_fglass	997	balance-scale	1004	synthetic_control	1005	glass
1011	ecoli	1015	confidence	1020	mfeat-karhunen	1038	gina_agnostic
1041	gina_prior2	1042	gina_prior	1048	jEdit_4.2_4.3	1049	pc4
1050	pc3	1056	mc1	1061	ar4	1063	kc2
1064	ar6	1065	kc3	1066	kc1-binary	1068	pc1
1069	pc2	1071	mw1	1073	jEdit_4.0_4.2	1075	datatrieve
1413	MyIris	1441	KungChi3	1443	PizzaCutter1	1444	PizzaCutter3
1446	CostaMadre1	1448	KnuggetChase3	1449	MeanWhile1	1451	PieChart1
1452	PieChart2	1464	blood-transfusion-service-center	1465	breast-tissue	1466	cardiotocography
1467	climate-model-simulation-crashes	1473	fertility	1475	first-order-theorem-proving	1494	qsar-biodeg
1515	micro-mass						

Tabela 3.1: Base de dados



(a) colunas (b) Instâncias

Figura 3.1: Descrição das bases de dados

3.3 Busca dos melhores hiperparâmetros

Para realizar a busca do melhor hiperparâmetros para cada base de dados, foi executado o MVS com os hiperparâmetros default, e em seguida foi executado a busca pelos melhores hiperparametros, para isso, foi utilizado os algoritmos de: Busca em Grade, Busca Aleatoria, OEP e Algoritmo Génetico. Com eles, foi executado e encontrado o melhor hiperparametro que cada busca encontrou e então comparado com o MVS com os hiperparametros default. Em seguida, foi realizado o teste 5x2cv com 95% de confiança para ver se realmente alguma busca conseguiu encontra um conjunto de hiperparametros melhor do que o default. A escolha dos algoritmos de busca utilizados se deu pelo fato de serem amplamente utilizados na literatura.

As buscas foram limitadas ao intervalo do hiperparâmetro C 2^{-4} até 2^{12} e Gamma 2^{-4} até 2^{12}, os algoritmos foram avaliados utilizando validação cruzada com a técnica k-fold com k=10, e utilizando uma media ponderada dos *F1-Score* de cada classe dando um peso relacionado com o tamanho da população de cada classe e foi registrado o tempo de execução de cada método de busca.

Para a Busca em Grade foram escolhidos 15 valores diferentes de C e 15 valores diferentes para o hiperparâmetro Gamma, totalizando 225 interações, para a Busca Aleatória foi realizada 225 testes com C e Gamma totalmente aleatórios totalizando 225 interações, para o Algoritmo Genético foram executadas 50 gerações com a populações de 30 cromossomos, totalizando 1550 interações, a técnica de mutação escolhida foi por substituição e com probabilidade de mutação definida como 0,5 foi utilizado BLX-α como técnica de cruzamento e seleção por torneio com n=3 e com probabilidade de 0.6, o tamanho da população e a quantidade de gerações foi adaptado devido a limitação de poder computacional disponível. Para o OEP foram executadas 50 gerações com 20 partículas totalizando 1000 interações, foi definida a topologia global como topologia a ser utilizada com seus parâmetros definidos como C1 = 2.05, C2 = 2.05 e W = 0.729, os valores de C1, C2 e W foram escolhidos apos uma busca aleatória e teste em um subconjunto de 30 bases escolhido aleatoriamente das 185 bases utilizadas, o intervalo de busca

Grupo	Nome	Descrição
General	attr_to_inst	Calcular a razão entre o número de atributos.
General	cat_to_num	Calcular a razão entre o número de feições categóricas e numéricas.
General	freq_class	Calcular a frequência relativa de cada classe distinta.
General	inst_to_attr	Calcular a razão entre o número de instâncias e atributos.
General	nr_attr	Calcular o número total de atributos.
General	nr_bin	Calcular o número de atributos binários.
General	nr_cat	Calcular o número de atributos categóricos.
General	nr_class	Calcular o número de classes distintas.
General	nr_inst	Calcular o número de instâncias (linhas) no conjunt de dados
General	nr_num	Calcular o número de recursos numéricos.
General	num_to_cat	Calcular o número de recursos numéricos e categóricos.
Statistical	can_cor	Calcular correlações canônicas de dados.
Statistical	cor	Calcular o valor absoluto da correlação de pares de colunas de conjuntos de dados distintos.
Statistical	cov	Calcular o valor absoluto da covariância de pares de atributos de conjuntos de dados distintos.
Statistical	eigenvalues	Calcular os autovalores da matriz de covariância do conjunto de dados.
Statistical	g_mean	Calcular a média geométrica de cada atributo.
Statistical	gravity	Calcular a distância entre o centro de massa das classes minoritária e majoritária.
Statistical	h_mean	Calcular a média harmônica de cada atributo.
Statistical	iq_range	Calcular o intervalo interquartil (IQR) de cada atributo.
Statistical	kurtosis	Calcule a curtose de cada atributo.
Statistical	lh_trace	Calcular o rastreamento de Lawley-Hotelling.
Statistical	mad	Calcular o Desvio Absoluto Mediano (MAD) ajustado por um fator.
Statistical	max	Calcular o valor máximo de cada atributo.
Statistical	mean	Calcular o valor médio de cada atributo.
Statistical	median	Calcular o valor mediano de cada atributo.
Statistical	min	Calcular o valor mediano de cada atributo.
Statistical	nr_cor_attr	Calcular o número de pares de atributos altamente correlacionados distintos.
Statistical	nr_disc	Calcular o número de correlação canônica entre cada atributo e classe.
Statistical	nr_norm	Calcular o número de atributos normalmente distribuídos com base em um determinado método.
Statistical	nr_outliers	Calcular o número de atributos com pelo menos um valor discrepante.
Statistical	p_trace	Calcular o traço de Pillai.
Statistical	range	Calcular o intervalo (max - min) de cada atributo.
Statistical	roy_root	Calcule a maior raiz de Roy.
Statistical	sd	Calcular o desvio padrão de cada atributo.
Statistical	sd_ratio	Calcular um teste estatístico para homogeneidade de covariâncias.
Statistical	skewness	Calcular a assimetria para cada atributo.
Statistical	sparsity	Calcular a métrica de esparsidade (possivelmente normalizada) para cada atributo.
Statistical	t_mean	Calcular a variação de cada atributo.
Statistical	var	Calcular a variância de cada atributo.
Statistical	w_lambda	Calcular o valor Lambda de Wilks'.
Info-theory	attr_conc	Calcular coef de concentração. de cada par de atributos distintos.
Info-theory	attr_ent	Calcular a entropia de Shannon para cada atributo preditivo.
Info-theory	class_conc	Calcular o coeficiente de concentração entre cada atributo e classe.
Info-theory	class_ent	Calcular a entropia de Shannon do atributo alvo.
Info-theory	eq_num_attr	Calcular o número de atributos equivalentes para uma tarefa preditiva.
Info-theory	joint_ent	Calcular a entropia conjunta entre cada atributo e classe.
Info-theory	mut_inf	Calcular as informações mútuas entre cada atributo e destino.
Info-theory	ns_ratio	Calcular o ruído dos atributos.

Tabela 3.2: Meta-feature utilizadas

utilizada foi o recomendada por Alam (2016), o numero de partículas e gerações foi adaptado devido a limitação de poder computacional disponível.

3.4 Comparação dos algoritmos de classificação

Para realizar a comparação dos algoritmos executados com os hiperparâmetros default e com hiperparâmetros recomendados, foi utilizado a comparação t-teste pareado 5x2CV Dietterich (1998) (*5x2cv paired t test for classifier comparisons*). Esse teste compara dois modelos, em consiste em dividir aleatoriamente a base de dados em duas metades iguais, em seguida ele

treina os modelos em uma das metades e os avalia na outra metade e em seguida os algoritmos são treinados na metade que antes era de avaliação e avaliados na metade que antes era de treino, o que resulta em 2 medidas distintas de desempenho, e então calculam-se a média e a variância da diferença dos desempenhos, esse processo é repetido cinco vezes e então a variância é usada para calcular a estatística t.

3.5 Treinamento dos Meta-aprendizes

Os algoritmos utilizados para criação dos meta-aprendizes foram MVS, Floresta Aleatória, Árvore de Decisão e Naive Bayes, esses algoritmos foram comparados com um algoritmo de Linha de Base que é um algoritmo fictício que apenas vota na classe majoritária da base de treinamento, cada modelo foi avaliado com o *f1_score* macro obtido por uma validação cruzada com k = 10, o teste 5x2cv foi utilizado como teste de hipótese para comparar o desempenho dos algoritmos, cada algoritmo foi treinado com valores default oferecido por Pedregosa et al. (2011) e tunados com Busca Aleatória 2000 interações.

3.6 Testes de hipóteses

Para realizar a discussão dos resultados e realizar as comparações, foi utilizado o teste de hipótese U de *Mann-Whitney* McKnight and Najab (2010), esse é um teste não paramétrico aplicado para duas amostras independentes. Esse teste é utilizado em situações em que um par de amostras independentes e quer testar se as populações que originam as amostras podem ser consideradas semelhantes ou não.

4

Resultados e Discussões

Nesta seção será apresentado os resultados obtidos com os métodos de busca e meta-aprendizes.

4.1 Desempenho dos Metodos de busca

A tabela 4.1 mostra os resultados do desempenho e tempo de execução dos metodos de busca utilizados.

F1-score					
Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	0.605	0.237	0.402	0.593	0.837
Algoritmo Genético	0.706	0.224	0.537	0.762	0.879
Busca aleatória	0.781	0.180	0.683	0.828	0.914
Busca em Grade	0.766	0.178	0.659	0.803	0.895
default	0.785	0.148	0.691	0.814	0.891
Tempo					
Algoritmo	Média	Desvio padrão	Q1	Q2	Q3
OEP	21.041	65.807	0.192	0.844	3.523
Algoritmo Genético	15.696	51.873	0.222	0.513	1.874
Busca aleatória	3.403	11.218	0.018	0.109	0.567
Busca em Grade	4.548	14.626	0.022	0.162	0.690

Tabela 4.1: *f1-score* e tempo das bases de dados - Geral

Em uma análise com todas as bases de dados juntas, os resultados obtidos de *f1-score* do algoritmo OEP possui um desempenho pior em relação aos outros métodos, com uma média de 0.605. Os valores *default* e Busca Aleatória obtiveram melhores resultado, com média de 0.781 e desvio padrão de 0.180, enquanto o *default* obteve uma média de 0.782 e desvio padrão de 0.148, contudo, ao realizar o teste de hipótese pode-se concluir que não obteve uma diferença

significativa entre os dois métodos, pois o p-valor foi de 0.581. Já na análise dos quartis, a Busca Aleatória alcança um melhor desempenho no terceiro quartil e perde desempenho no primeiro e segundo quartis, de acordo com o boxplot 4.1 a Busca Aleatória sofre com *outliers* negativos que afetam sua média, porém, consistentemente conseguem ter um desempenho levemente superior ao *default*, enquanto o *default* consegue mitigar o problema de *outliers* conseguindo obter um desempenho em geral mais consistente, com os resultados dos P-valores, os métodos de Busca Aleatória, Busca em Grade e o *default* tem um desempenho melhor do que os métodos OEP e Algoritmo Genético.

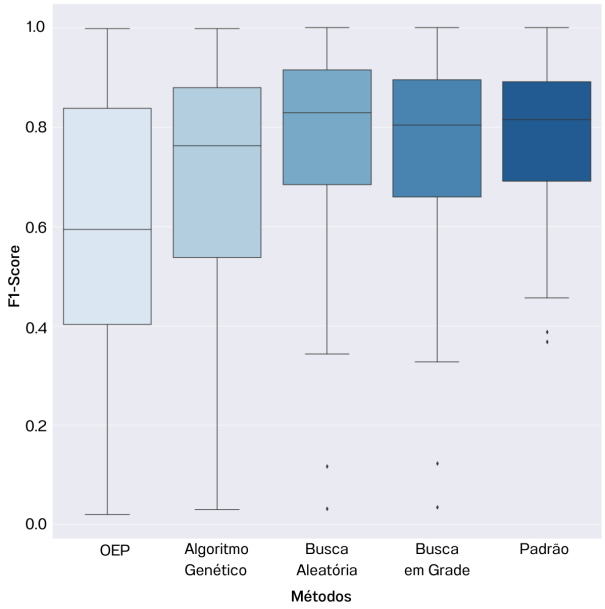


Figura 4.1: F1-Sscore - Geral

Referente ao consumo de tempo o *default* não foi considerado na análise pois ele não é um método de busca e consequentemente seria mais rápido do que qualquer outro método, uma visão geral do gasto de tempo pode ser visualizado na Figura 4.2. O OEP obteve o pior desempenho com uma média de 21.041 minutos de execução e com desvio padrão de 65.807, quando esses dados são analisados em conjunto com o terceiro quartil que fica em 3.523 minutos indica que em bases muito grandes o OEP tem o seu custo computacional drasticamente afetado e gastando um tempo maior em relação aos outros métodos. Os métodos de Busca em Grade e Busca Aleatória obtiveram um desempenho consideravelmente melhor do que os métodos de OEP e Algoritmo Genético, esse resultado podem ser confirmado pelo teste de hipótese apresentado na Tabela 4.1, mesmo com a Busca Aleatória obtendo o desempenho levemente melhor do que a Busca em Grade, ele alcançou um P-valor de 0.604, isso mostra que esse desempenho não é considerável.

Com o intuito de se aprofundar nos resultados, os eles foram divididos em bases complexas e bases não complexas. Foram consideradas bases complexas as bases com mais de 50 colunas contabilizando um total de 37 bases e bases não complexas as bases com menos de 50 colunas

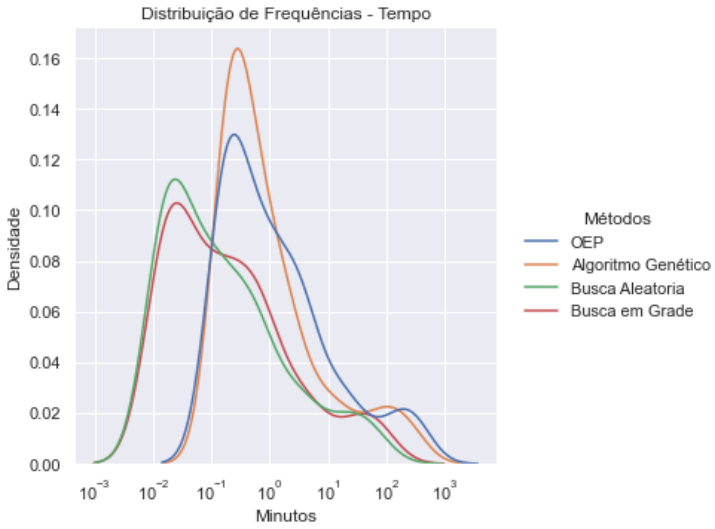


Figura 4.2: Tempo - Geral

F1-score

-	OEP	AG	BA	BG	Padrão
OEP	-	0.0	0.0	0.0	0.0
AG	0.0	-	0.001	0.02	0.004
BA	0.0	0.001	-	0.307	0.584
BG	0.0	0.02	0.307	-	0.599
Padrão	0.0	0.004	0.584	0.599	-

Tempo

-	OEP	AG	BA	BG
OEP	-	0.411	0.0	0.0
Algoritmo Genético	0.411	-	0.0	0.0
Busca aleatória	0.0	0.0	-	0.212
Busca em Grade	0.0	0.0	0.212	-

Tabela 4.2: f1-score e tempo - P valor

que totalizaram 148 bases, esses conjuntos foram subdivididos em bases grandes e pequenas, bases grandes foram consideradas bases que tinham um número igual ou superior a 1000 instâncias, enquanto as pequenas foram consideradas bases com menos de 1000 instâncias. O conjunto de bases pequenas e complexas contemplou um total de 18 bases, o conjunto de bases grandes e complexas totalizou 19 bases, o conjunto de bases não complexa e pequena totalizou 110 bases e o conjunto não complexa e grande totalizou 38 bases.

4.1.1 Bases Complexas

Ao analisar de forma geral as bases complexas, o *default* tem uma vantagem sobre os outros métodos analisados obtendo um *f1-score* médio de 0.786 e desvio padrão de 0.180, essa van-

tagem é significativa dado os valores de p-valores (Tabela 4.1.1), por outro lado, o OEP acaba sendo o pior método com um *f1-score* médio de 0.455 e desvio padrão de 0.229, esse resultado pode ser confirmado no teste de hipótese, e observados na Figura 4.3. Referente ao custo computacional, os métodos de Busca Aleatória e Busca em Grade atingiram 8.992 e 9.976 minutos de média respectivamente, e tem desempenho superior aos métodos de OEP e Algoritmo Genético que obtêm médias de 45.348 e 39.163 minutos respectivamente, o p-valor obtido é de 0.604 negando a hipótese que a diferença entre a busca em grade e a busca aleatória é significativa, mas os p-valores obtidos podem confirmar que existe uma diferença significativa da Busca Aleatória e da Busca em Grade em relação ao OEP e ao Algoritmo Genético, essa variação de tempo pode ser observado na Figura 4.4. Os dados de P-valores completos são apresentados na Tabela 4.1.1

F1-score					
Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	0.455	0.229	0.344	0.403	0.475
Algoritmo Genético	0.552	0.268	0.403	0.503	0.762
Busca aleatória	0.636	0.256	0.426	0.580	0.857
Busca em Grade	0.657	0.253	0.492	0.626	0.895
default	0.786	0.180	0.662	0.750	0.976
Tempo					
Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	45.348	92.531	0.438	3.284	21.025
Algoritmo Genético	39.163	89.250	0.325	1.820	15.288
Busca aleatória	8.992	19.623	0.052	0.583	3.668
Busca em Grade	9.976	21.118	0.059	0.701	4.418

Tabela 4.3: *f1-score* e tempo das bases de dados complexa

F1-score					
-	OEP	AG	BA	BG	Padrão
OEP	-	0.038	0.0	0.0	0.0
AG	0.038	-	0.16	0.062	0.0
BA	0.0	0.16	-	0.06	0.006
BG	0.0	0.062	0.06	-	0.011
Padrão	0.0	0.0	0.006	0.011	-
Tempo					
-	OEP	AG	BA	BG	
OEP	-	0.665	0.005	0.01	
Algoritmo Genético	0.665	-	0.012	0.029	
Busca aleatória	0.005	0.012	-	0.604	
Busca em Grade	0.01	0.029	0.604	-	

Tabela 4.4: *f1-score* e tempo - P valor das bases complexas

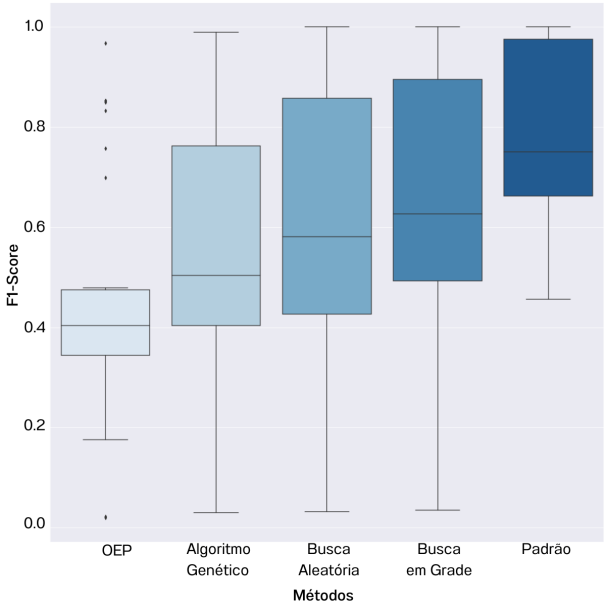


Figura 4.3: Bases complexas - Score

Bases Complexas e Pequenas

Aprofundando a análise para as bases complexas e pequenas, é observado que a média do *f1-score* de todos os métodos diminuiu referente a análise geral (Tabela 4.1.1), indicando que menor quantidade de dados é um problema devido a complexidade das bases de dados, nesse cenário o *default* continua com um desempenho melhor do que os outros métodos, obtendo uma média de 0.722, embora essa diferença não é mais tão significativa referente a Busca em Grade, que conseguiu ter a segunda melhor média de *f1-score* com uma média de 0.594, pois o p-valor que na análise geral era de 0.011, agora aumentou para 0.060, esses resultados também podem ser visualizados com a Figura 4.6(b). Já na análise das médias de tempo, o OEP obteve a maior média de 1.478 minutos em contraste com a maior média de tempo de todas as bases complexas, que também foi do OEP, mas com um valor de 45.348 minutos. Além disso, é observável que a maior complexidade das bases não aumentou muito o tempo de execução do algoritmo (Figura 4.5(b)). Os dados de P-valores completos são apresentados na Tabela 4.1.1

Bases Complexas e Grandes

Nos resultados das bases complexas e grandes, a média do *f1-score* subiu em relação à análise geral e em relação à análise das bases complexas e grandes, como pode ser visto na Tabela 4.1.1, em especial para o *default* que foi o único método que conseguiu uma média de *f1-score* maior até do que a sua média para a base como um todo, obtendo uma média de 0.847 e desvio padrão de 0.161, em contraste com 0.785 de média e 0.148 de desvio padrão obtidos para a base como um todo, esse desempenho é significativamente melhor do que os métodos OEP, Algoritmo Genético e busca aleatória como mostra os resultados do p-valor do teste de hipótese.

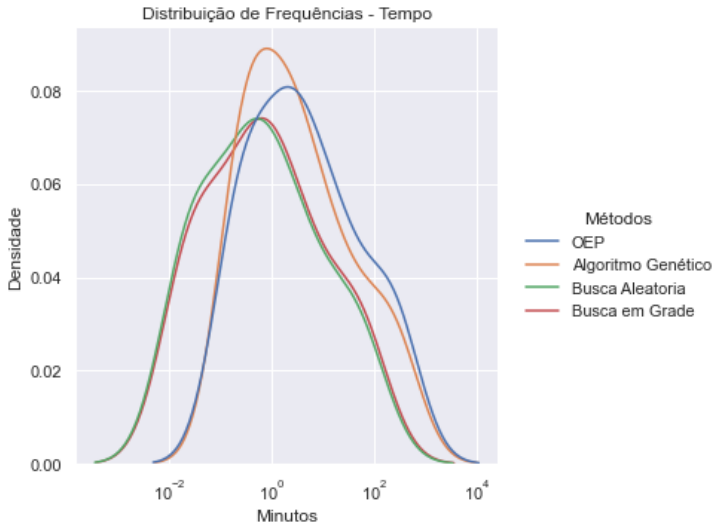


Figura 4.4: Bases complexas - Tempo

F1-Score

Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	0.426	0.194	0.374	0.407	0.436
Algoritmo Genético	0.540	0.218	0.415	0.498	0.653
Busca aleatória	0.564	0.245	0.414	0.482	0.773
Busca em Grade	0.594	0.245	0.438	0.557	0.770
default	0.722	0.181	0.567	0.711	0.817

Tempo

Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	1.478	2.681	0.193	0.407	1.447
Algoritmo Genético	1.052	1.680	0.239	0.316	0.898
Busca aleatória	0.232	0.430	0.019	0.049	0.201
Busca em Grade	0.255	0.444	0.020	0.057	0.234

Tabela 4.5: *f1-score* e tempo das bases de dados complexa e pequena

Em relação a Busca em Grade, o p-valor tem resultado de 0.054, que fica muito próximo do limite de aceitar a hipótese, indicando que se o número de bases de dados fossem maior esse resultado poderia ser significativamente maior. Todos os métodos melhoraram suas médias de *f1-score*, demonstrando que o maior número de informação ajuda o algoritmo a aprender mais sobre o problema e obter um desempenho geral melhor, uma distribuição dos scores pode ser visualizados na Figura 4.6(a). Com relação ao tempo, todos os métodos obtiveram aumento de custo computacional significativo nesse subconjunto de bases (Figura 4.5(a)), em uma análise de quartil todos os métodos tem seu primeiro quartil maior para esse subconjunto do que o seu terceiro quartil, para a base como um todo esse resultado indica que para esse subconjunto os métodos de busca dos melhores hiperparâmetros são muito custosos e o desempenho do *default* torna ele uma ótima opção para esse cenário. Os dados de P-valores completos são apresentados na Tabela 4.1.1

F1-score

-	OEP	AG	BA	BG	Padrão
OEP	-	0.023	0.018	0.005	0.0
AG	0.023	-	0.975	0.466	0.009
BA	0.018	0.975	-	0.506	0.024
BG	0.005	0.466	0.506	-	0.06
Padrão	0.0	0.009	0.024	0.06	-

Tempo

-	OEP	AG	BG	BG
OEP	-	0.812	0.0	0.001
Algoritmo Genético	0.812	-	0.0	0.001
Busca aleatória	0.0	0.0	-	0.58
Busca em Grade	0.001	0.001	0.58	-

Tabela 4.6: *f1-score* e tempo - P valor das bases complexas e pequena

F1-Score

Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	0.482	0.260	0.342	0.402	0.765
Algoritmo Genético	0.562	0.314	0.369	0.540	0.831
Busca aleatória	0.704	0.255	0.553	0.751	0.945
Busca em Grade	0.717	0.253	0.574	0.762	0.956
default	0.847	0.161	0.700	0.920	0.978

Tempo

Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	86.908	115.627	3.827	21.025	147.386
Algoritmo Genético	75.268	114.456	3.583	15.288	90.383
Busca aleatória	17.291	24.910	0.667	3.668	25.192
Busca em Grade	19.186	26.604	0.757	4.418	30.396

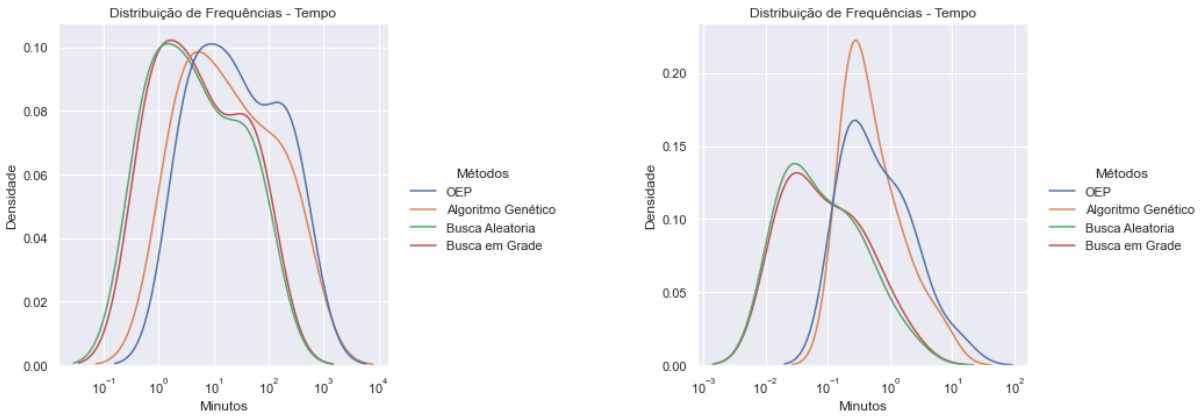
Tabela 4.7: *f1-score* e tempo das bases de dados complexa e grande

O subconjunto das bases complexas afetou negativamente o desempenho de todos os métodos de busca tanto em relação ao *f1-score*, mas principalmente no tempo de execução em contraste com o *default*, que conseguiu manter uma boa performance até mesmo nesse cenário, indicando que a alta complexidade das bases de dados torna menos atrativo a busca por melhores hiperparâmetros, pois essa busca se torna muito mais custosa e mais complexa de se otimizar. Esse resultado pode indicar que seria necessário uma busca maior para o desempenho dos métodos de busca conseguir se manter relevantes, porém, o alto custo computacional tenderia aumentar mais ainda para apenas conseguir um desempenho parecido com o *default*.

F1-score					
-	OEP	AG	BA	BG	Padrão
OEP	-	0.342	0.006	0.006	0.0
AG	0.342	-	0.194	0.129	0.003
BA	0.006	0.194	-	0.726	0.044
BG	0.006	0.129	0.726	-	0.054
Padrão	0.0	0.003	0.044	0.054	-

Tempo				
-	OEP	AG	BG	BG
OEP	-	0.502	0.011	0.021
Algoritmo Genético	0.502	-	0.047	0.08
Busca aleatória	0.011	0.047	-	0.484
Busca em Grade	0.021	0.08	0.484	-

Tabela 4.8: *f1-score* e tempo - P valor das bases complexas e grande



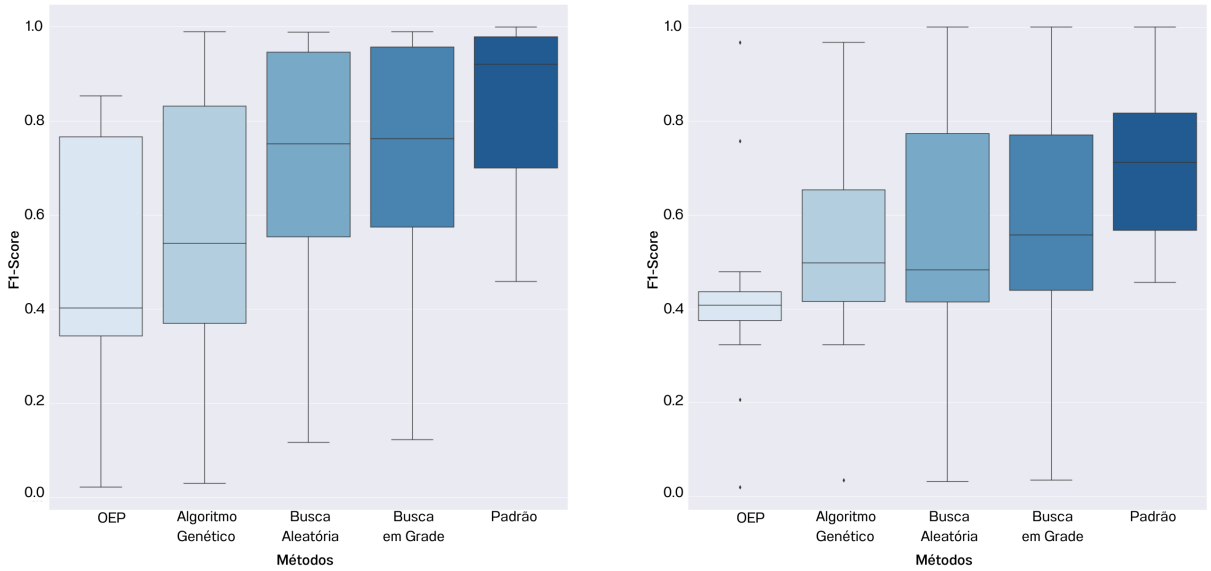
(a) Complexa e grande - Tempo

(b) Complexa e pequena - Tempo

Figura 4.5: Bases complexas (grandes e pequenas) - Tempo

4.1.2 Bases Não Complexas

Todos os métodos de busca obtiveram médias melhores do *f1-score* para bases não complexas comparado com o resultado da médio geral das bases, e o *default* conseguiu manter a mesma média. A Busca Aleatória obteve o melhor desempenho em relação ao *f1-score* conseguindo uma média de 0.817 e um desvio padrão de 0.142, seguido da Busca em Grade que ficou com um resultado da média de 0.793 e um desvio padrão de 0.142, o p-valor mostra que a Busca Aleatória conseguiu um desempenho significativamente melhor do que os outros métodos com exceção da Busca em Grade, onde o p-valor foi de 0.137. O OEP e o Algoritmo Genético mantiveram como os piores métodos, obtendo uma performance média de 0.643 e 7.44 respectivamente, e o teste de hipótese mostra que os outros métodos obtiveram desempenho significativamente melhor do que ambos, além disso, o OEP alcançou um desempenho significativamente pior do que o Algoritmo Genético, todos os dados estão dispostos na Tabela 4.1.2 e com uma visualização



(a) Complexa e grande

(b) Complexa e pequena

Figura 4.6: Bases complexas (grandes e pequenas)

da distribuição na Figura 4.8.

F1-Score					
Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	0.643	0.224	0.414	0.673	0.853
Algoritmo Genético	0.744	0.194	0.649	0.793	0.888
Busca aleatória	0.817	0.134	0.743	0.845	0.916
Busca em Grade	0.793	0.142	0.705	0.827	0.895
default	0.785	0.139	0.708	0.816	0.883
Tempo					
Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	14.965	56.016	0.173	0.600	2.924
Algoritmo Genético	9.829	35.270	0.211	0.416	1.372
Busca aleatória	2.006	7.307	0.017	0.074	0.406
Busca em Grade	3.191	12.218	0.019	0.100	0.618

Tabela 4.9: *f1-score* e Tempo das bases de dados não complexa

No que se refere ao tempo, todos os métodos obtiveram uma média menor do que o geral (Figura 4.8), com a Busca Aleatória e a Busca em Grade obtendo o melhor desempenho com 2 minutos de média e 7.307 de desvio padrão para a Busca Aleatória e média de 3.191 e desvio padrão de 12.218 para a Busca em Grade. O teste de hipótese mostra que ambos obtêm um desempenho significativamente melhor do que o OEP, que obteve uma média de 14.965 e desvio padrão de 56.016 e o Algoritmo Genético com média de 9.829 e desvio padrão de 35.270. O teste de hipótese mostra que o OEP obteve um pior desempenho comparado com todos os outros métodos, já a diferença entre a Busca Aleatória e a Busca em Grade não é significativa.

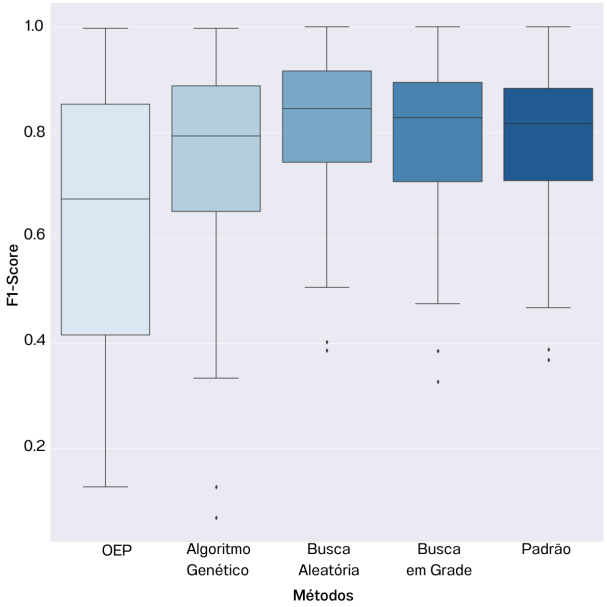


Figura 4.7: Score - Base não complexa

Os dados de P-valores completos são apresentados na Tabela 4.1.2.

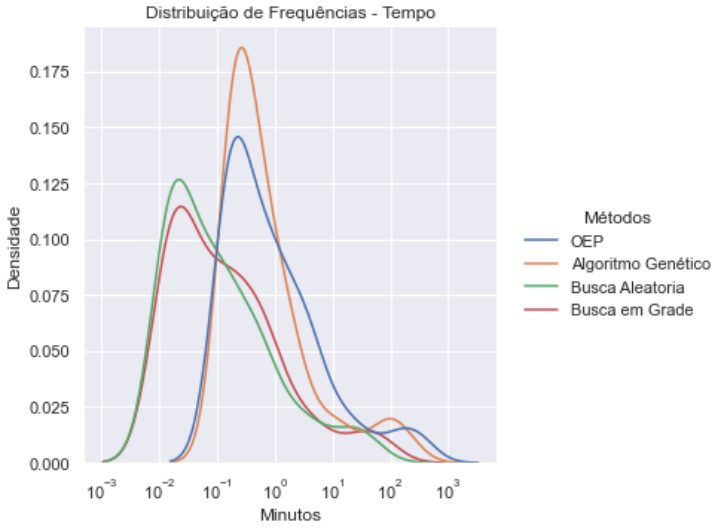


Figura 4.8: Tempo - Base não complexa

Bases não complexas e grandes

No cenário de bases não complexas e grandes os algoritmos obtiveram um salto de desempenho em relação à média geral, mostrando que esse cenário o algoritmo MVS, conseguiu ter uma ótima performance, os dados podem ser vistos na Tabela 4.1.2, apesar da melhora o ranking de desempenho, que se manteve o mesmo de todo o conjunto de bases não complexas com o OEP novamente obtendo o pior desempenho com um *f1-score* médio de 0.630 e desvio padrão de 0.26. Com a Busca Aleatória, que obteve 0.840 de média e 0.260 de desvio padrão, e a

F1-score

-	OEP	AG	BA	BG	Padrão
OEP	-	0.0	0.0	0.0	0.0
AG	0.0	-	0.003	0.096	0.279
BA	0.0	0.003	-	0.137	0.03
BG	0.0	0.096	0.137	-	0.531
Padrão	0.0	0.279	0.03	0.531	-

Tempo

-	OEP	AG	BG	BG
OEP	-	0.475	0.0	0.0
Algoritmo Genético	0.475	-	0.0	0.0
Busca aleatória	0.0	0.0	-	0.217
Busca em Grade	0.0	0.0	0.217	-

Tabela 4.10: *f1-score* e tempo - P valor das bases não complexas

Busca em Grade com média de 0.832 e 0.142 de desvio padrão (Figura 4.10(a)). Seguindo a mesma tendência, os métodos de busca com exceção da Busca Aleatória obtiveram um aumento na média do tempo de execucação mas mantendo o mesmo ranking de desempenho geral do conjunto de bases não complexas, essa variação pode ser vizualizada na Figura 4.9(a). Os dados de P-valores completos são apresentados na Tabela 4.1.2

F1-Score

Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	0.630	0.260	0.385	0.681	0.862
Algoritmo Genético	0.798	0.192	0.746	0.850	0.934
Busca aleatória	0.840	0.139	0.793	0.867	0.948
Busca em Grade	0.832	0.142	0.766	0.854	0.940
default	0.825	0.143	0.780	0.855	0.929

Tempo

Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	56.575	100.363	3.539	7.761	32.671
Algoritmo Genético	37.109	62.563	1.715	4.827	55.596
Busca aleatória	7.598	12.996	0.521	1.229	5.107
Busca em Grade	12.135	21.966	0.669	1.624	6.224

Tabela 4.11: *f1-score* e Tempo das bases de dados não complexa e grande

Bases não complexas e pequenas

Nas bases não complexas e pequenas o método OEP obteve uma leve melhora média de desempenho no *f1-score*, que passou de 0.643 de média e 0.224 de desvio padrão no conjunto total das bases não complexa para 0.647 de média e 0.211 de desvio padrão no conjunto não complexo

F1-score

-	OEP	AG	BA	BG	Padrão
OEP	-	0.006	0.0	0.001	0.002
AG	0.006	-	0.421	0.636	0.844
BA	0.0	0.421	-	0.689	0.53
BG	0.001	0.636	0.689	-	0.767
Padrão	0.002	0.844	0.53	0.767	-

Tempo

-	OEP	AG	BG	BG
OEP	-	0.071	0.0	0.0
Algoritmo Genético	0.071	-	0.0	0.002
Busca aleatória	0.0	0.0	-	0.139
Busca em Grade	0.0	0.02	0.139	-

Tabela 4.12: *f1-score* e tempo - P valor das bases não complexas e grande

e pequena, os outros métodos obtiveram uma leve queda de desempenho (Figura 4.10(b)), mas em geral manteve o ranking de desempenho do conjunto de bases não complexas 4.1.2. No que diz respeito ao tempo, houve uma diminuição no tempo em relação ao conjunto das bases não complexas, esse resultado já era esperado para as bases menores, pois o custo computacional do algoritmo esta diretamente ligado ao tamanho e complexidade da base, essa variação pode ser visualizada na Figura 4.9(b).

F1-Score

Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	0.647	0.211	0.452	0.663	0.840
Algoritmo Genético	0.726	0.192	0.608	0.764	0.878
Busca aleatória	0.809	0.132	0.714	0.828	0.908
Busca em Grade	0.780	0.140	0.692	0.796	0.890
default	0.771	0.136	0.686	0.802	0.873

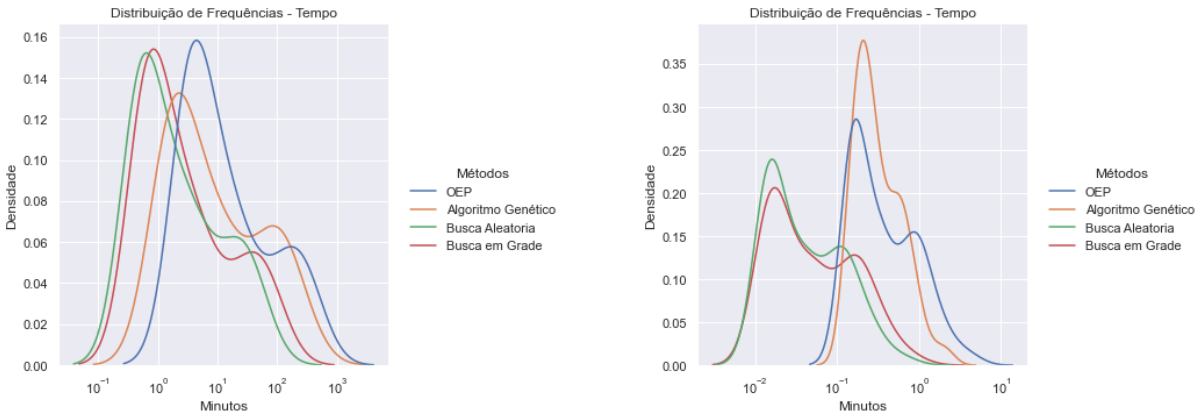
Tempo

Algoritmo	média	Desvio padrão	Q1	Q2	Q3
OEP	0.591	0.712	0.163	0.316	0.843
Algoritmo Genético	0.406	0.354	0.196	0.270	0.511
Busca aleatória	0.074	0.102	0.015	0.034	0.106
Busca em Grade	0.102	0.141	0.017	0.045	0.160

Tabela 4.13: *f1-score* e Tempo das bases de dados não complexa e pequena

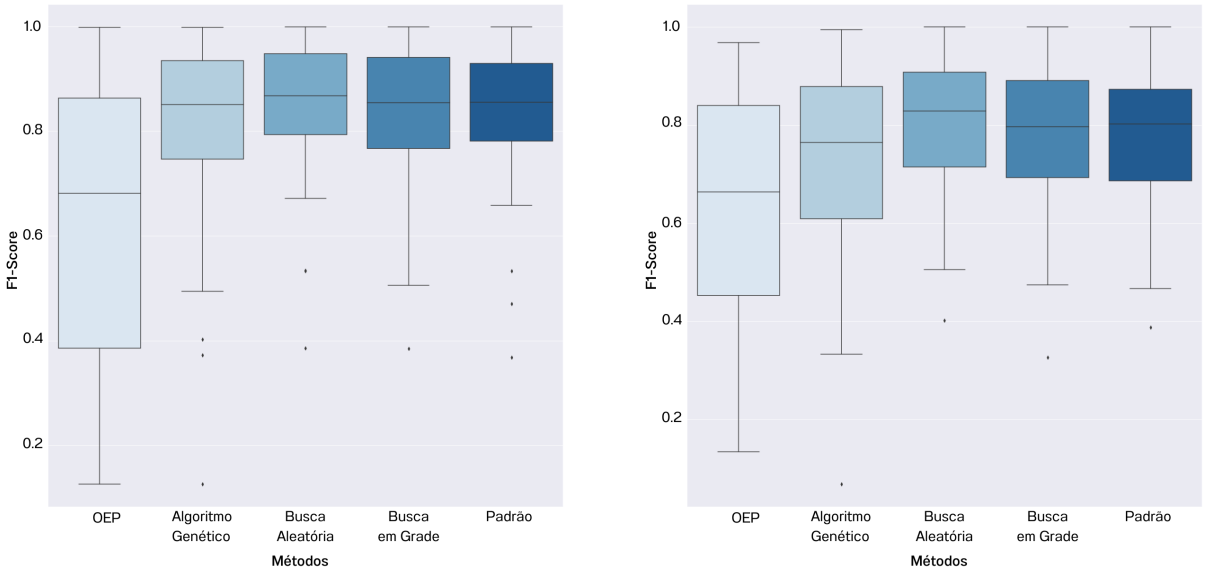
4.2 meta-aprendiz

As resultados dos meta-aprendizes foram organizados de acordo com o conjunto de meta-features utilizadas, sendo a ultima seção a combinacao das três meta-features.



(a) Não complexa e grande - Tempo (b) Não complexa e pequena - Tempo

Figura 4.9: Bases não complexas (Grandes e Pequenas) - Tempo



(a) Não complexa e grande (b) Não complexa e pequena

Figura 4.10: Bases não complexas (Grandes e Pequenas) - Score

F1-score

-	OEP	AG	BA	BG	Padrão
OEP	-	0.008	0.0	0.0	0.0
AG	0.008	-	0.002	0.085	0.24
BA	0.0	0.002	-	0.113	0.026
BG	0.0	0.085	0.113	-	0.57
Padrão	0.0	0.24	0.026	0.57	-

Tempo

-	OEP	AG	BG	BG
OEP	-	0.625	0.0	0.0
Algoritmo Genético	0.625	-	0.00	0.0
Busca aleatória	0.0	0.0	-	0.121
Busca em Grade	0.0	0.0	0.121	-

Tabela 4.14: *f1-score* e tempo - P valor das bases não complexas e pequena

4.2.1 Statistical

Entre os meta-aprendizes não tunados a Floresta Aleatória se destaca com a melhor performance obtendo um *f1-score* médio de 0.536 e desvio padrão de 0.162, o Naive Bayes se destaca negativamente obtendo um *f1-score* médio de 0.310 com desvio padrão de 0.168, para comparação o algoritmo de Linha de Base conseguiu média de 0.449 e desvio padrão de 0.006. O teste de hipótese mostra que não teve uma diferença de desempenho significativo entre nenhum dos meta-aprendizes não tunados, no cenário dos meta-aprendizes tunados todos os algoritmos aumentaram suas médias de *f1-score* tornando a Árvore de Decisão, que antes tinha 0.490 de média de *f1-score* aumentar seu desempenho e alcançando a marca de 0.604 de *f1-score* médio, porém, o teste de hipótese que pode ser visto na Tabela 4.2.1 mostra que apenas o MVS obteve uma diferença de desempenho real quando comparado o meta-aprendiz não tunado com o meta-aprendiz tunado. Por outro lado, após a tunagem dos meta-aprendizes, o Naive Bayes obteve uma melhora no seu desempenho alcançando o desempenho igual do algoritmo de Linha de Base, isso pode indicar um possível problema de sub-ajustamento. Em contrapartida, os outros algoritmos conseguiram além de aumentar seu desempenho médio, obtiveram uma diferença de desempenho significante segundo o teste de hipótese em relação ao algoritmo de Linha de Base, mostrado na Tabela 4.2.1.

4.2.2 Inf_theory

O desempenho dos meta-aprendizes não tunados treinados com a meta-feature *inf_theory* obtiveram um desempenho inferior aos meta-aprendizes treinados com as meta-features *statistical*, com exceção do Naive Bayes que melhorou seu desempenho obtendo média de 0.310 e desvio padrão de 0.168, em relação ao *f1-score* quando treinado com a meta-feature *Statistical* para

Sem tunar		
Algoritmo	Média	Desvio Padrão
MVS	0.449	0.006
Floresta Aleatória	0.536	0.162
Árvore de Descisão	0.490	0.078
Naive Bayes	0.310	0.168
Linha de Base	0.449	0.006
Tunado		
Algoritmo	Média	Desvio Padrão
MVS	0.562	0.103
Floresta Aleatória	0.580	0.162
Árvore de Descisão	0.604	0.138
Naive Bayes	0.449	0.006
Linha de Base	0.449	0.006

Tabela 4.15: Statistical - Desempenho

Sem tunar					
-	MVS	FA	AD	NB	LB
MVS	-	0.111	0.32	0.299	1
FA	0.111	-	0.708	0.113	0.111
AD	0.32	0.708	-	0.234	0.32
NB	0.299	0.113	0.234	-	0.299
LB	1	0.111	0.32	0.299	-
Tunado					
-	MVS	FA	AD	NB	LB
MVS	-	0.362	0.837	0.048	0.048
FA	0.362	-	0.722	0.049	0.049
AD	0.837	0.722	-	0.014	0.014
NB	0.048	0.049	0.014	-	1
LB	0.048	0.049	0.014	1	-

Tabela 4.16: Statistical - P valor

Algoritmo	P-valor
MVS	0.048
Floresta Aleatória	1.000
Árvore de Decisão	0.442
Naive Bayes	0.299

Tabela 4.17: P valor - Statistical tunado x não tunado

0.419 e desvio padrão de 0.107 quando treinado com a meta-feature *inf_theory*. O MVS manteve a média de 0.449 e desvio padrão de 0.006 para ambas as meta-features, quando analisado os meta-aprendizes tunados MVS, Floresta Aleatória e Árvore de Decisão conseguiram um de-

sempenho significativamente melhor do que o algoritmo de Linha de Base como pode ser visto na Tabela 4.2.2, e assim como no conjunto de meta-features *Statistical* apenas o MVS obteve uma real diferença de desempenho quando comparado com tunado e não tunado, esse resultado pode ser observado na Tabela 4.2.2.

Sem tunar		
Algoritmo	Média	Desvio Padrão
MVS	0.449	0.006
Floresta Aleatória	0.476	0.107
Árvore de descisão	0.514	0.113
Naive Bayes	0.419	0.048
Linha de Base	0.449	0.006
Tunado		
Algoritmo	Média	Desvio Padrão
MVS	0.544	0.113
Floresta Aleatória	0.490	0.114
Árvore de descisão	0.575	0.139
Naive Bayes	0.449	0.006
Linha de Base	0.449	0.006

Tabela 4.18: Inf_theory - Desempenho

Sem tunar					
-	MVS	FA	AD	NB	LB
MVS	-	0.316	0.044	0.978	1
FA	0.316	-	0.129	0.663	0.316
AD	0.044	0.129	-	0.148	0.044
NB	0.978	0.663	0.148	-	0.978
LB	1	0.316	0.044	0.978	-
Tunado					
-	MVS	FA	AD	NB	LB
MVS	-	0.383	0.120	0.011	0.011
FA	0.383	-	0.830	0.379	0.390
AD	0.120	0.830	-	0.478	0.477
NB	0.011	0.379	0.478	-	1.000
LB	0.011	0.390	0.477	1.000	-

Tabela 4.19: inf_theory - P valor

4.2.3 General_meta

Entre os meta-aprendizes não tunados a Floresta Aleatória obteve o melhor desempenho com um *f1-score* médio de 0.554 e desvio padrão de 0.178, enquanto o algoritmo de Naive Bayes

Algoritmo	P-valor
MVS	0.011
Floresta Aleatória	0.879
Árvore de Decisão	0.323
Naive Bayes	0.979

Tabela 4.20: P valor - inf_theory tunado x não tunado

foi o pior com um *f1-score* médio de 0.403 e desvio padrão 0.068, porém, os p-valores obtidos no teste de hipótese não podem afirmar que houve uma diferença real entre os meta-aprendizes não tunados. Os meta-aprendizes tunados novamente obtiveram um melhor desempenho do que os não tunados, mas após submeter as amostras ao teste de hipótese apenas o MVS tunado obteve uma real diferença de desempenho em relação ao meta-aprendiz de Linha de Base, e na comparação tunado e não tunado o MVS foi o único que teve uma diferença real de desempenho após ser tunado.

Sem tunar		
Algoritmo	Média	Desvio Padrão
MVS	0.449	0.006
Floresta Aleatória	0.554	0.178
Árvore de descisão	0.517	0.120
Naive Bayes	0.403	0.068
Linha de Base	0.449	0.006
Tunado		
Algoritmo	Média	Desvio Padrão
MVS	0.528	0.108
Floresta Aleatória	0.584	0.153
Árvore de descisão	0.599	0.141
Naive Bayes	0.549	0.134
Linha de Base	0.449	0.006

Tabela 4.21: General Meta - Desempenho

4.2.4 Statistical + Inf_theory + General_meta

Unindo todos os conjuntos de meta-features a Árvore de Decisão obteve o melhor desempenho em relacão ao *f1-score* médio de 0.482 com desvio padrão de 0.364 (Tabela 4.2.4), mas o resultado do teste de hipótese apresentado na Tabela 4.2.4 mostra que não existe uma diferença significativa entre os desempenhos dos meta-aprendizes não tunados e tunados. A Árvore de Decisão seguiu obtendo um melhor resultado de média de *f1-score* como mostrado na Tabela 4.2.4, mas ao analisar os testes de hipótese apenas o MVS conseguiu um resultado significamente melhor do que o algoritmo de Linha Base, obtendo um p-valor de 0.003, na comparação

Sem tunar					
-	MVS	FA	AD	NB	LB
MVS	-	0.132	0.853	0.14	1.0
FA	0.132	-	0.29	0.039	0.151
AD	0.853	0.29	-	0.085	0.845
NB	0.14	0.039	0.085	-	0.146
LB	1.0	0.151	0.845	0.146	-
Tunado					
-	MVS	FA	AD	NB	LB
MVS	-	0.882	0.254	0.090	0.007
FA	0.882	-	0.32	0.128	0.093
AD	0.254	0.32	-	0.223	1.000
NB	0.090	0.128	0.223	-	0.174
LB	0.007	0.093	1.000	0.174	-

Tabela 4.22: General Meta - P valor

Algoritmo	P-valor
MVS	0.010
Floresta Aleatória	0.876
Árvore de Decisão	0.953
Naive Bayes	0.305

Tabela 4.23: P valor - general_meta tunado x não tunado

meta-aprendiz tunado contra meta-aprendiz não tunado o MVS seguiu sendo o único que conseguiu uma melhora de desempenho significante obtendo um p-valor de 0.008, os resultados das meta-features juntas podem ser visto na Tabela 4.2.4.

Sem tunar		
Algoritmo	Média	Desvio Padrão
MVS	0.449	0.006
Floresta Aleatória	0.476	0.069
Árvore de descisão	0.482	0.095
Naive Bayes	0.364	0.150
Linha de Base	0.449	0.006
Tunado		
Algoritmo	Média	Desvio Padrão
MVS	0.540	0.148
Floresta Aleatória	0.544	0.144
Árvore de descisão	0.607	0.180
Naive Bayes	0.542	0.103
Linha de Base	0.449	0.006

Tabela 4.24: general_meta + inf_theory + statistical - Desempenho

Sem tunar					
-	MVS	FA	AD	NB	LB
MVS	-	0.784	0.300	0.303	1.000
FA	0.784	-	0.343	0.593	0.757
AD	0.300	0.343	-	0.095	0.302
NB	0.303	0.593	0.095	-	0.315
LB	1.0	0.757	0.302	0.315	-
Tunado					
-	MVS	FA	AD	NB	LB
MVS	-	0.711	0.362	0.223	0.003
FA	0.711	-	0.425	0.556	0.518
AD	0.362	0.425	-	0.256	0.054
NB	0.223	0.556	0.256	-	0.941
LB	0.003	0.518	0.054	0.941	-

Tabela 4.25: general_meta + inf_theory + statistical - P valor

Algoritmo	P-valor
MVS	0.008
Floresta Aleatória	0.576
Árvore de Decisão	0.209
Naive Bayes	0.535

Tabela 4.26: P valor - general_meta + inf_theory + statistical tunado x não tunado

5

Conclusão

Nessa sessão será feita a discussão dos resultados obtidos dos treinamentos dos meta-aprendizes.

Os resultados dos métodos de busca indicam que principalmente para bases menos complexas vale a pena usar algum método de otimização para conseguir obter um melhor resultado, os métodos de Busca em Grade e Busca Aleatória conseguiram obter bons resultados de otimização sem um grande aumento do custo computacional, por outro lado, o OEP e o Algoritmo Genético obtiveram o pior desempenho entre os métodos de busca, esse resultado contrasta com os resultados de Mantovani et al. (2015), onde o OEP e Algoritmo Genético obtiveram um desempenho melhor do que o default e um resultado muito próximo a Busca Aleatória, essa diferença pode ser devido a necessidade de uma quantidade maior de buscas para o OEP e o Algoritmo Genético para conseguirem explorar mais resultados, esse resultado também pode ser influenciado por ter uma maior quantidade de bases de dados, maiores e mais complexas. Um ponto em comum entre os dois estudos foi o bom desempenho da Busca Aleatória, em um cenário de grandes volumes de dados, utilizar os hiperparâmetros default é uma escolha segura, pois consegue obter um bom desempenho com um custo computacional inferior a qualquer método de busca, por outro lado, para pequenas bases de dados utilizar métodos de otimização como a Busca Aleatória conseguiu obter bons resultados nos cenários analisados, principalmente no cenários das bases de dados não complexas e sem aumentar muito o seu custo computacional. A Busca em Grade também provou resultados consistentes em todos os cenários, mas esses resultados podem ser afetados por experiência de conhecimento de domínio. Os métodos de otimização bioinspirados como o OEP e o Algoritmo Genético mostraram se tornar pouco atrativos para cenários de grandes volumes de dados, onde obtiveram um alto custo computacional e não conseguiram obter uma boa performance, mostrando que eles precisavam fazer mais buscas que elevariam ainda mais o custo computacional.

O treinamento dos meta-aprendizes demonstraram que o conjunto de meta-features statistical foi superior aos outros grupos de meta-features aumentando o resultado dos meta-aprendizes

para melhor até mesmo do quando se utilizou os três conjuntos de meta-features juntos para treinar os meta-aprendizes. Apesar disso, o desempenho dos meta-aprendizes não passou do *f1-score* médio de 0.604, indicando que a complexidade do problema não está sendo totalmente representada através dos grupos de meta-features utilizado e quando o aumento da quantidade de meta-features utilizadas é analisada, alguns meta-aprendizes tiveram seu desempenho afetado negativamente indica que existe informações que podem até mesmo prejudicar o desempenho da aprendizagem dos meta-aprendizes, uma possível solução para isso seria criar um novo conjunto de meta-features que pudessem representar melhor a complexidade do problema para esses meta-aprendizes, outra solução seria a utilização de uma nova base de dados onde as classes estivessem mais balanceadas. O algoritmo Naive Bayes obteve os piores resultados como um meta-aprendiz, sendo pior do que o algoritmo de linha de base e não conseguindo obter um desempenho significativamente melhor do que a linha de base em nenhum experimento, esse resultado possivelmente é fruto da natureza do aprendizado do algoritmo que não consegue encontrar dependência entre variáveis atenuando ainda mais a hipótese que os conjuntos de meta-features utilizados não conseguem expressar o quão tunavel uma base de dados é, e para conseguir um resultado melhor os algoritmos cruzam as informações, a árvore de decisão conseguiu obter os melhores resultados de média de *f1-score* entre os meta-aprendizes, porém, muitas vezes esse desempenho melhor não era refletido no teste de hipótese, possivelmente isso acontece pois a árvore de decisão tem uma maior facilidade para se sobreajustar a casos pontuais e fora da curva, levando a crer que a base de dados realmente não conseguiu representar a complexidade de separação das classes do problema fazendo com que pontuais sobre ajustes a pontos fora da curva se tornem o diferencial entre um meta-aprendiz e outro. Em uma análise do desempenho dos meta-aprendizes tunados contra os meta-aprendizes não tunados todos os meta-aprendizes obtiveram um aumento de performance média de *f1-score* quando tunado, mas apenas no caso do MVS essa diferença se tornou significativa segundo o teste de hipótese, no caso do Naive Bayes a tunagem apenas tornou seu desempenho igual ao do algoritmo de linha de base o que mostra um sobreajuste do meta-aprendiz em relação à base de dados.

Referências bibliográficas

Mahamad Nabab Alam. Particle swarm optimization: Algorithm and its codes in matlab. *ResearchGate*, 8(1):10, 2016.

Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís Paulo F Garcia, Jefferson Tales Oliva, André CPLF de Carvalho, et al. Mfe: Towards reproducible meta-feature extraction. *J. Mach. Learn. Res.*, 21(111):1–5, 2020.

Jehad Ali, Rehanullah Khan, Nasir Ahmad, and Imran Maqsood. Random forests and decision trees. *International Journal of Computer Science Issues (IJCSI)*, 9(5):272, 2012.

Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53):370–418, 1763.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11): 27–34, 1996.

Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter. Openml-python: an extensible python api for openml. *arXiv:1911.02490*, 2019.

Zoubin Ghahramani. Unsupervised learning. In *Summer school on machine learning*, pages 72–112. Springer, 2003.

Marouane Hachimi, Georges Kaddoum, Ghyslain Gagnon, and Poulmanogo Illy. Multi-stage jamming attacks detection using deep learning combined with kernelized support vector machine in 5g cloud radio access networks. In *2020 international symposium on networks, computers and communications (ISNCC)*, pages 1–5. IEEE, 2020.

Guy S Handelman, Hong Kuan Kok, Ronil V Chandra, Amir H Razavi, Shiwei Huang, Mark Brooks, Michael J Lee, and Hamed Asadi. Peering into the black box of artificial intelligence: evaluation metrics of machine learning methods. *American Journal of Roentgenology*, 212(1):38–43, 2019.

Simon Haykin. Neural networks and learning machines. 2010.

Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, 29(4):329–337, 2015.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

S Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. **DOI** 10.1109/ICNN.1995.488968.

Hsuan-Tien Lin and Chih-Jen Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. *Neural Comput*, 3(1-32):16, 2003.

Rafael G Mantovani, André LD Rossi, Joaquin Vanschoren, Bernd Bischl, and André CPLF De Carvalho. Effectiveness of random search in svm hyper-parameter tuning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. Ieee, 2015.

Patrick E McKnight and Julius Najab. Mann-whitney u test. *The Corsini encyclopedia of psychology*, pages 1–1, 2010.

Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

Iqbal Muhammad and Zhu Yan. Supervised machine learning approaches: A survey. *ICTACT Journal on Soft Computing*, 5(3), 2015.

Vladimir Nasteski. An overview of the supervised machine learning methods. *Horizons. b*, 4: 51–62, 2017.

Marco Aurélio Cavalcanti Pacheco et al. Algoritmos genéticos: princípios e aplicações. *ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida*, 28, 1999.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

J. Ross Quinlan. Learning decision tree classifiers. *ACM Computing Surveys (CSUR)*, 28(1): 71–72, 1996.

Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

Adriano Rivolli, Luís PF Garcia, Carlos Soares, Joaquin Vanschoren, and André CPLF de Carvalho. Characterizing classification datasets: a study of meta-features for meta-learning. *arXiv preprint arXiv:1808.10406*, 2018.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. **DOI** 10.1147/rd.33.0210.

Axel Thevenot. Particle swarm optimization (pso) visually explained. https://towardsdatascience.com/particle-swarm-optimization-visually-explained-46289eeb2e14, December 2020.

Alan M Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009.