Aprendizagem de Máquina IF699

Modelo de classificação do comportamento de bando

Implementação de um modelo de classificação para o entendimento da percepção humana do comportamento de bando utilizando a metodologia CRISP-DM

Escrito por:

José Danilo Silva do Carmo (jdsc),

Ednelson Luan Lourenço Cavalcanti (ellc).

Sumário

1. Business Understanding

1.1. Objetivo do projeto

O comportamento de enxame é um fenômeno natural que ocorre em muitas espécies, como pássaros, peixes e insetos. O entendimento desse comportamento é essencial em diversas áreas, incluindo robótica, inteligência artificial e até mesmo urbanismo. A capacidade humana de reconhecer padrões do comportamento de bandos, sem necessariamente entender as nuances, é uma habilidade que desejamos replicar em modelos computacionais.

Este projeto tem o objetivo de desenvolver um modelo, que a partir de dados da classificação humana do comportamento de enxame, seja capaz de discernir se um conjunto de dados representando o movimento de *boids* está se movendo em bando ou não. Ao incorporar a percepção humana na classificação, buscamos insights sobre como os seres humanos interpretam e categorizam o comportamento de enxames.

1.2. Critérios de sucesso

O critério sucesso de negócio é obter um modelo com pelo menos 70% de acurácia em identificar *boids* que estão se movendo em bando ou não, levando em consideração a interpretabilidade dos modelos utilizados, de forma a responder as seguintes questões: “O modelo consegue reconhecer se há um comportamento de movimento em bando acima do limiar estabelecido (70%)?” e “Quais são as informações que os humanos consideram mais relevantes para classificar um movimento em bando?”.

1.3. Avaliação da situação

O termo "boid" é derivado da palavra "bird-oid", criado por Craig Reynolds. Esses agentes simulam comportamentos coletivos de animais seguindo três regras fundamentais: separação, alinhamento e coesão. O dataset utilizado será o Swarm Behavior, que foi construído a partir de uma enquete online por UNSW, Australia, com 24017 instâncias e 2400 features, que a partir de vídeos, tinha o objetivo de classificar humanamente se um boid está em bando, alinhado ou agrupado.

Ao focar na percepção humana, concentramo-nos na tarefa de classificar se um boid está ou não em bando. Esse enfoque simplificado permitirá uma análise mais profunda da capacidade dos modelos em replicar a intuição humana, destacando padrões sutis que podem ser imperceptíveis em uma análise mais ampla.

Cada instância do dataset denota um enxame de 200 boides, onde cada atributo representa características de um boide, como a posição, vetor de velocidade, vetor de alinhamento, vetor de separação, vetor de coesão, número de boids vizinhos em um raio de alinhamento/coesão e o número de boids vizinhos em um raio de separação.

1.4. Plano de projeto

O plano de projeto seguirá a estrutura do CRISP-DM (Cross-Industry Standard Process for Data Mining), dividido em fases distintas: entendimento do negócio, entendimento dos dados, preparação dos dados, modelagem, avaliação e implantação.

Durante a fase de entendimento dos dados, além da análise exploratória padrão, focaremos em entender como as características específicas dos boids impactam na percepção humana de movimento em bando. A preparação dos dados incluirá transformações necessárias para criar features relevantes para a classificação, mantendo a fidelidade ao contexto original.

Na fase de modelagem, exploraremos uma variedade de algoritmos, desde métodos tradicionais até abordagens de aprendizado profundo. A interpretabilidade dos modelos será uma consideração central, buscando entender não apenas a performance, mas também como os modelos tomam decisões.

O projeto será conduzido em um ambiente colaborativo, permitindo a execução assíncrona e paralela dos algoritmos. Utilizaremos as bibliotecas do sklearn no Jupyter Notebook, seja no Colab ou localmente. A documentação detalhada garantirá a rastreabilidade e reprodutibilidade do projeto, facilitando futuras iterações e análises.

2. Exploratory data analysis

2.1. Descrição dos dados

Vamos utilizar os dados proveniente da pesquisa online realizada pela Universidade de Nova Gales do Sul (UNSW), Austrália, chamado de Swarm Behavior, no qual possui três categorias de dados: “Flocking - Not Flocking” (se os *boids* estão se movendo em bando com a mesma velocidade), “Aligned - Not Aligned” (se os *boids* estão se movendo na mesma direção) e ‘Grouped - Not Grouped’ (se os *boids* estão juntos). Nós iremos utilizar apenas o conjunto de dados “Flocking.csv”, que se refere a “Flocking - Not Flocking”, uma vez que nosso objetivo é reconhecer o movimento de bando.

O dataset possui 24016 instâncias de 2400 atributos, no qual cada atributo se refere às seguintes informaçeos de um *boid*: posição (xm, ym), vetor de velocidade (xVeln, yVeln), vetor de alinhamento (xAm, yAm), vetor de separação (xSm, ySm), vetor de coesão (xCm, yCm), número de *boids* no raio de Alinhamento/Coesão (nACm) e número de boids no raio do separação (nSm). No qual cada informação é repetida para todos os m *boids*, onde $m = 1, \ldots, 200$. Por fim, existe o atributo que determina a classificação humana binária se os *boids* estão andando em bando ou não (1 para sim, e 0 para não).

Devido à natureza intrínseca dos dados, a análise focada na variável x1, não pode ignorar as demais variáveis xN, e assim por diante. A atenção dedicada a todas as features xN durante esta fase inicial da análise contribui para a identificação de padrões, relações e comportamentos multifacetados presentes no conjunto de dados. Essa compreensão abrangente estabelece as bases para decisões subsequentes no processo de modelagem e inferência, fortalecendo a integridade e robustez da análise de dados conduzida.

2.2. Quallidade dos dados

Em busca de observar os tipos de dados, foi utilizado o método `info()` no *DataFrame* de dados, é observado que existem 3 tipos de dados `float64(1999)`, `int64(401)`, `object(1)`, sendo que não deveria haver nenhum tipo object.

```
# replace field that's entirely space (or empty) with NaN
data = data.replace(r'^\s*$', np.nan, regex=True)
data.isna().sum().sum()
```

```
1
```

Figura 1: Contagem de valores não numéricos

A figura 1 mostra que foi descoberto que isso foi causado por causa de um valor incompleto. Para resolver, foi removido as últimas duas instâncias, de forma a remover a instância com x1 incompleto que tem a classe 1 e a penúltima instância que tinha classe 0, para equilibrar a quantidade de classes (1 e 0), ficando 12007 instâncias para cada classe.

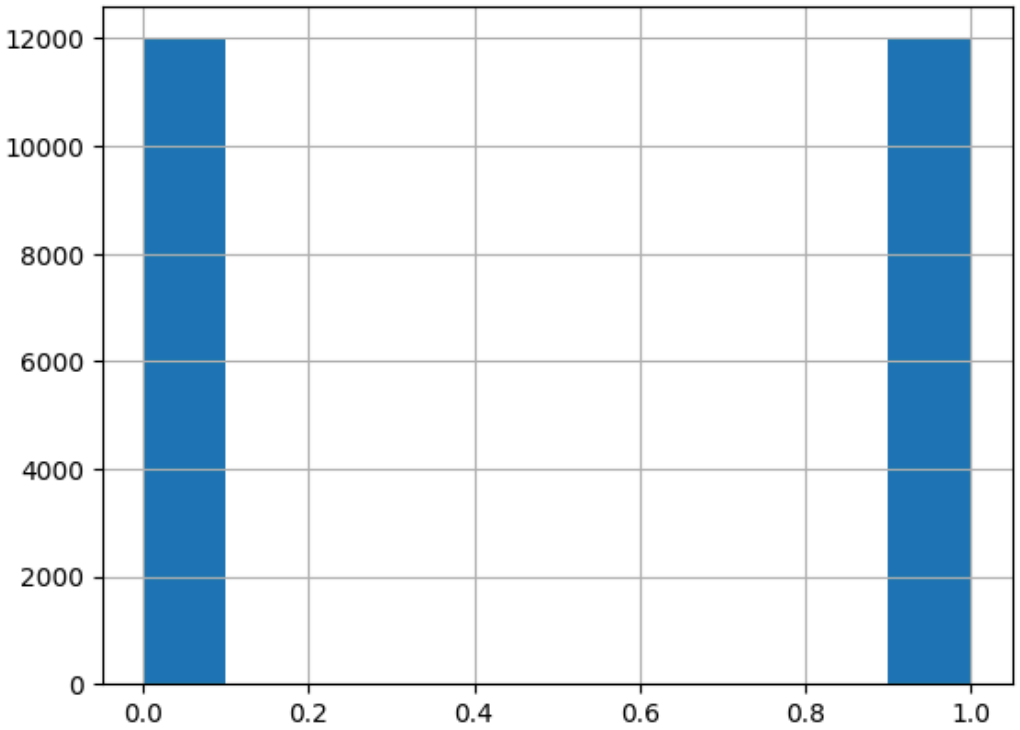


Figura 2: Histograma do atributo classe

Por fim, após a modificação, foi verificado se existe algum valor nulo nos dados e se havia algum valor duplicado, através da linha de código a seguir, mostrando que todos os dados estão preenchidos e não estão duplicados:

```
print("Exists some null value: ", data.isnull().sum().any())
print("Exists some duplicated value: ", len(data[data.duplicated()]) > 0)
```

```
Exists some null value:  False
Exists some duplicated value:  False
```

Figura 3: Verificação de valores nulos

2.3. Exploração dos dados

Uma vez que as informações de cada *boid* está disposto nas features, então para analisar cada uma das features foi agregado em um novo *DataFrame* no qual, por exemplo, a coluna “x” é a concatenação de x1…x200, e assim por diante:

```
columns = [x[:-1] for x in data.columns[:12]]
aggr_data = pd.DataFrame(columns=columns)

for x in columns:
    aggr_data[x] = np.concatenate(tuple(data[x + str(y)] for y in range(1, 201)))

print(f"Transforming from {data.shape} to {aggr_data.shape}")
```

```
Transforming from (24014, 2401) to (4802800, 12)
```

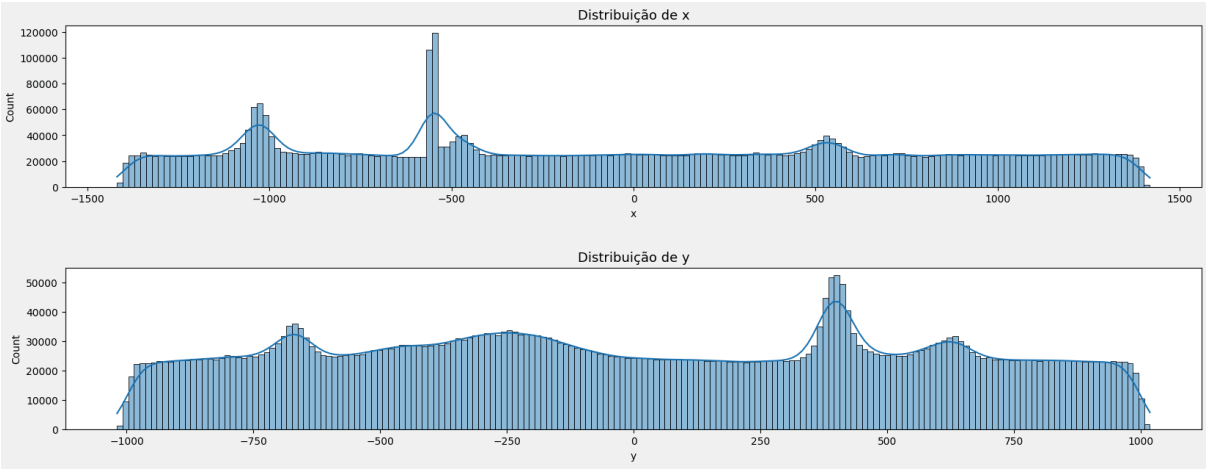
Figura 4: Agregação das features

A partir disso, é possível obter a distribuição de cada uma das features, onde, quanto a localização foi observado que a coordenada x varia entre -1418.25 e 1417.68, enquanto a y varia entre -1018.92 e 1018.16, onde a mediana deles são -55.94 e -6.44 respectivamente. No caso da velocidade, o vetor x varia entre -19.95 e 19.83 enquanto o y varia entre -19.99 e 19.78. É importante ressaltar que essas informações apenas determinam o estado inicial da simulação do comportamento dos *boids* e podem ser visualizadas na tabela a seguir:

	x	y	xVel	yVel
Mediana	-55.94	-6.44	-1.40	0.22
Desvio padrão	799.57	561.48	5.93	7.69
Mínimo	-1418.25	-1018.92	-19.95	-19.99
Máximo	1417.68	1018.16	19.83	19.78

Tabela 1: Coordenadas iniciais e vetor de velocidade

A figura a seguir mostra a distribuição das coordenadas inicias (x, y). Podemos notar que os valores -545.66 e 393.77 são os mais recorrentes em (x, y), com uma frequência de 214 e 71 ocorrências respectivamente. Enquanto 0.02 e 7.17 são os valores mais recorrentes em xVel e yVel, com a frequência de 12333 e 3925 respectivamente:



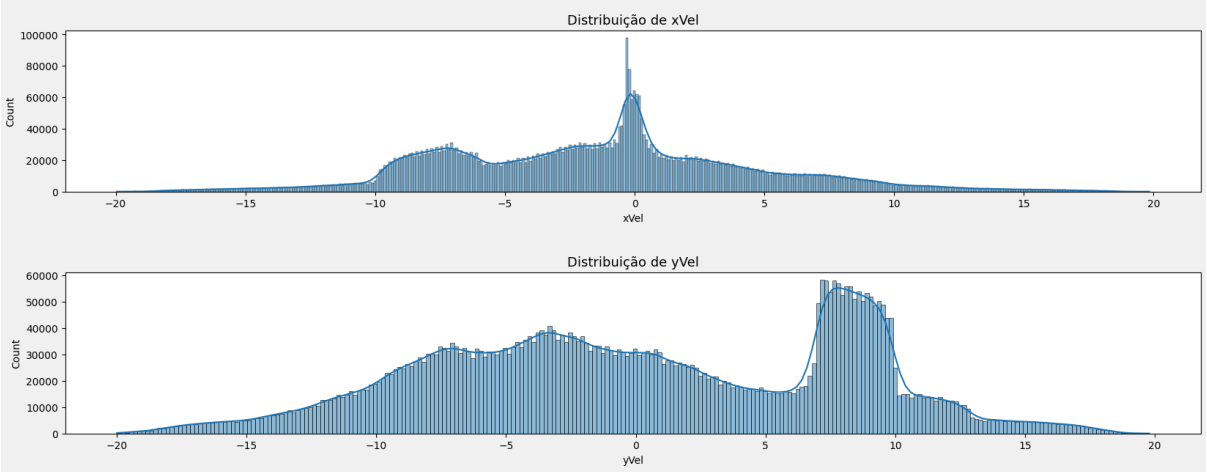


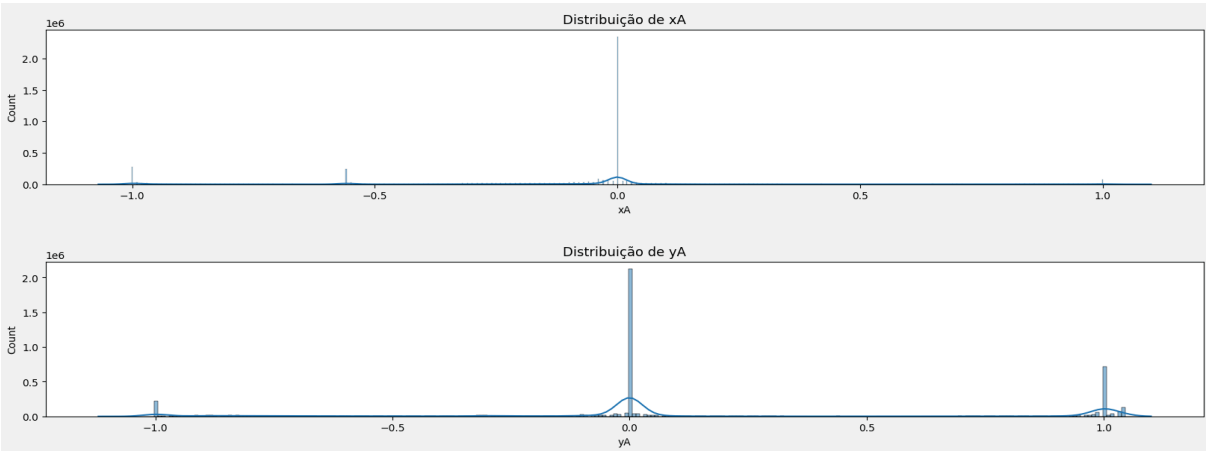
Figura 5: Distribuição de X, Y, xVel e yVel

No caso das informações do comportamento do movimentos dos *boids* (sepração, alinhamento e coesão), os vetores de separação variam entre (-1037197.42, -1048459.42) e (3692911.2, 4191195.27), os vetores de alinhamento variam entre (-1.07, -1.12) e (1.1, 1.1) e os de coesão variam de (-2.68, -2.68) até (2.68, 2.68), como mostrado na tabela a seguir:

	xS	yS	xA	yA	xC	yC
Mediana	12.48	20.13	-0.11	0.12	-0.05	0.08
Desvio padrão	6007.75	7691.21	0.38	0.58	0.55	0.61
Mínimo	-1037197.42	-1048459.42	-1.07	-1.12	-2.68	-2.68
Máximo	3692911.2	4191195.27	1.1	1.1	2.68	2.68

Tabela 2: Vetores de separação, alinhamento e coesão

Quanto a distribuição de separação, alinhamento e coesão é possível ver que os valores com maior recorrência são todos o valor 0, como mostrado na figura a seguir:



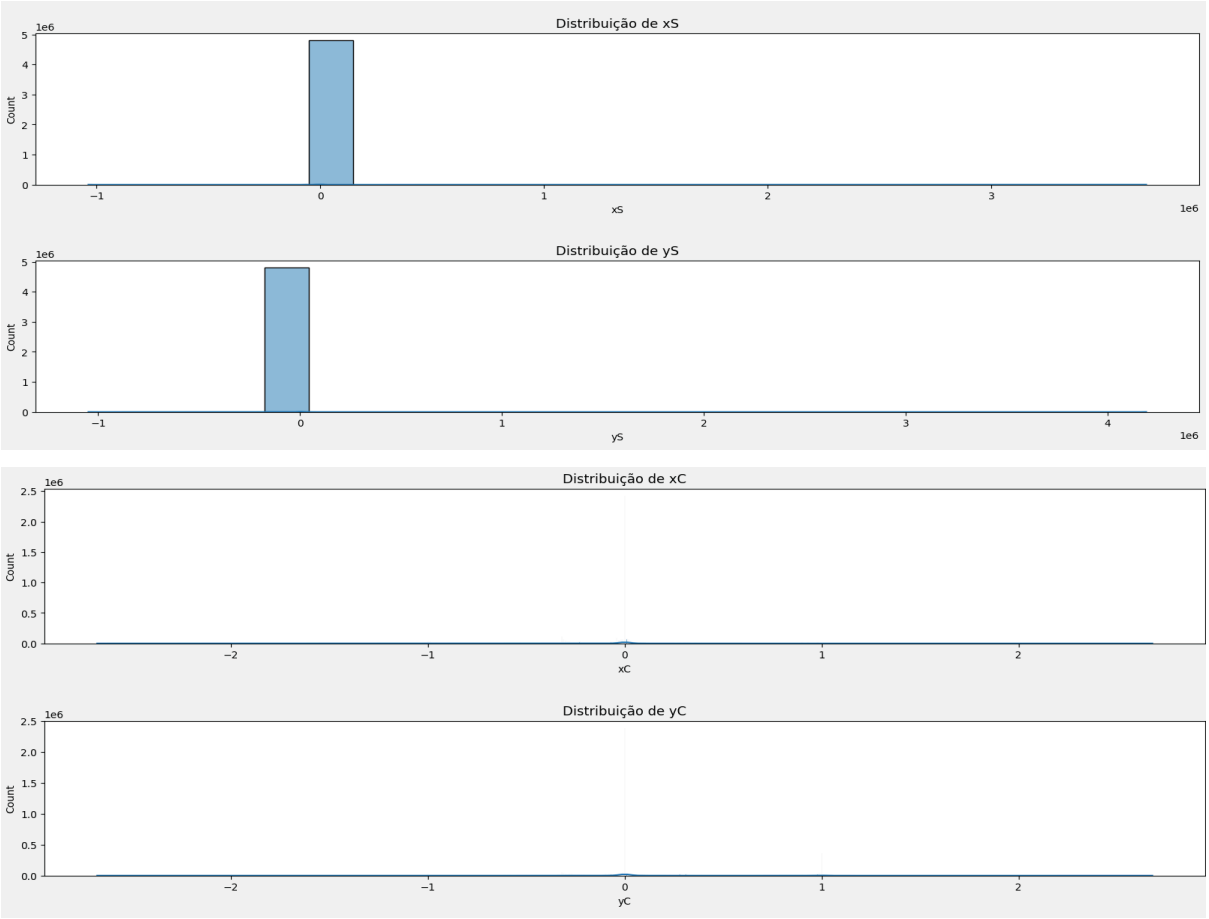


Figura 6: Distribuição de xA, yA, xS, yS, xC, yC

Por fim, as informações de número de *boids* no raio de Alinhamento/Coesão (nACm) e número de boids no raio do separação (nSm) variam de 0 até 171 e 108 respectivamente, possuindo uma mediana de 23.68 e 1.78 *boids* respectivamente. A distribuição deles se encontra na figura a seguir:

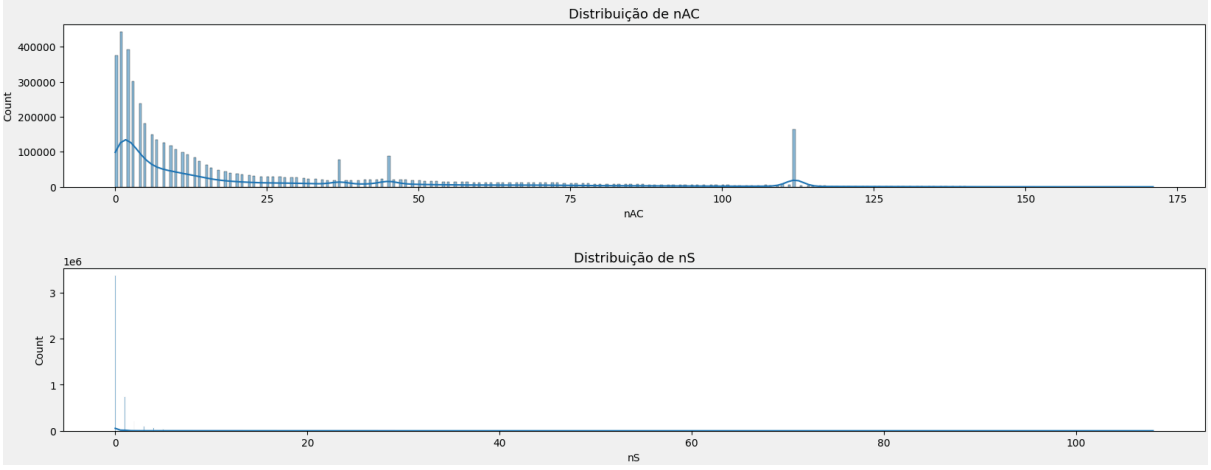


Figura 7: Distribuição de nAC e nS

Por fim, na próxima figura é possível ver o gráfico boxplot de cada uma das features, onde é perceptível que metade das features possuem muitos outliers: xVel (vetor x de velocidade), xA (vetor x de alinhamento), xC (vetor x de coesão), yC (vetor y de coesão), nAC

(número de *boids* no ráio de Alinhamento/Coesão) e nS (número de *boids* no raio de separação).

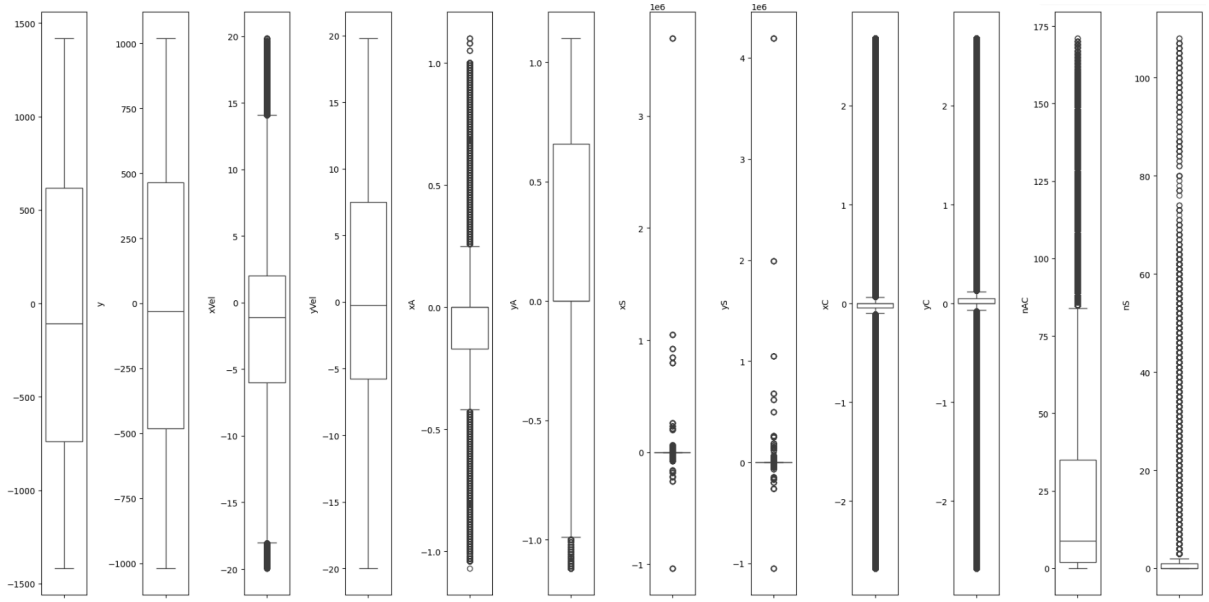


Figura 8: Boxplot das features x, y, xVel, yVel, xA, yA, xS, yS, xC, yC, nAC e nS

3. Data preparation

3.1. Lidando com outliers

Como foi visto na etapa anterior, nós temos algumas features com outliers, porém lidar com os outliers, seja os removendo ou aplicando a winsorização (substituir outliers com o valor mais próximo de dados que não é considerado um outlier), seria equivalente a remover *boids* ou modificar o comportamento de alguns deles (como por exemplo, sua velocidade ou alinhamento). Por isso não foi removido os outliers, pois prejudica a nossa proposta de solução, removendo a variabilidade dos dados, uma vez que o nosso modelo deve determinar o resultado independente do comportamento dos outliers.

3.2. Lidando com valores categóricos

Como foi mostrado na etapa anterior, não possuímos quaisquer valores categóricos, apenas valores numéricos, de forma que tratamentos como generalização dos dados ou estratégias como One-Hot Encoding não serão necessárias.

3.3. Normalização e padronização dos dados

Uma vez que os dados variam em diferentes escalas, então para a realização das próximas análises dos dados, é necessário realizar uma normalização ou padronização dos dados. Através de estatística descritiva, podemos observar que a média e o desvio padrão dos dados não possuem proximidade, o que serve de constatação que o conjunto de dados não se ajusta adequadamente a uma curva gaussiana, assim, a melhor abordagem é pela normalização dos dados, em vez da padronização.

Essa normalização será útil para se poder entender algumas análises na próxima etapa. O processo de normalização dos dados os transforma de forma que sigam uma distribuição mais próxima da uniforme, com média zero e desvio padrão igual a um. E foi implementada utilizando o MinMaxScaler como demonstrado a seguir:

```
from sklearn.preprocessing import MinMaxScaler

scale_norm = MinMaxScaler()
normalised_aggr = scale_norm.fit_transform(aggr_data)
```

Figura 9: Aplicação da normalização dos dados

3.4. Seleção das features

A partir dos dados normalizados, foi realizado um esforço de verificar se existe alguma feature que deve ser removida do dataset, através da ajuda do gráfico de violino e análise correlação entre as features. O gráfico de violino é semelhante ao boxplot, no qual mostra a distribuição dos dados após agrupar por uma ou mais features, porém mostrando uma representação da densidade dos dados. No nosso caso, estamos plotando o gráfico de forma vertical e dividindo a distribuição dos dados por classe. Dessa forma, se o gráfico apresentar apenas uma linha horizontal, significa que os dados são todos constantes, já se apresentar uma linha vertical, significa que estão muito dispersos.

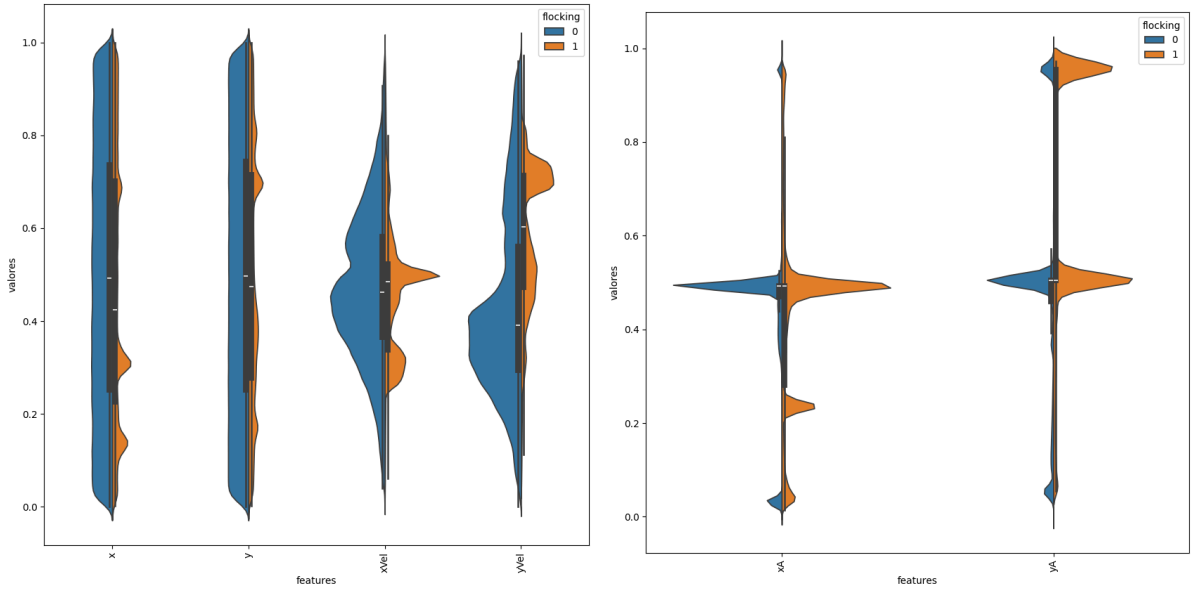


Figura 10: Gráfico de violino de x, y, xVel, yVel, xA e yA

Analisando o gráfico de violino da feature xS e yS em relação às demais features, percebemos que apesar dessas features possuírem o maior desvio padrão e maior range de dados, de acordo com a análise exploratória, ainda assim, a sua distribuição é agrupada demais em um determinado ponto, corroborando com o gráfico boxplot da etapa anterior:

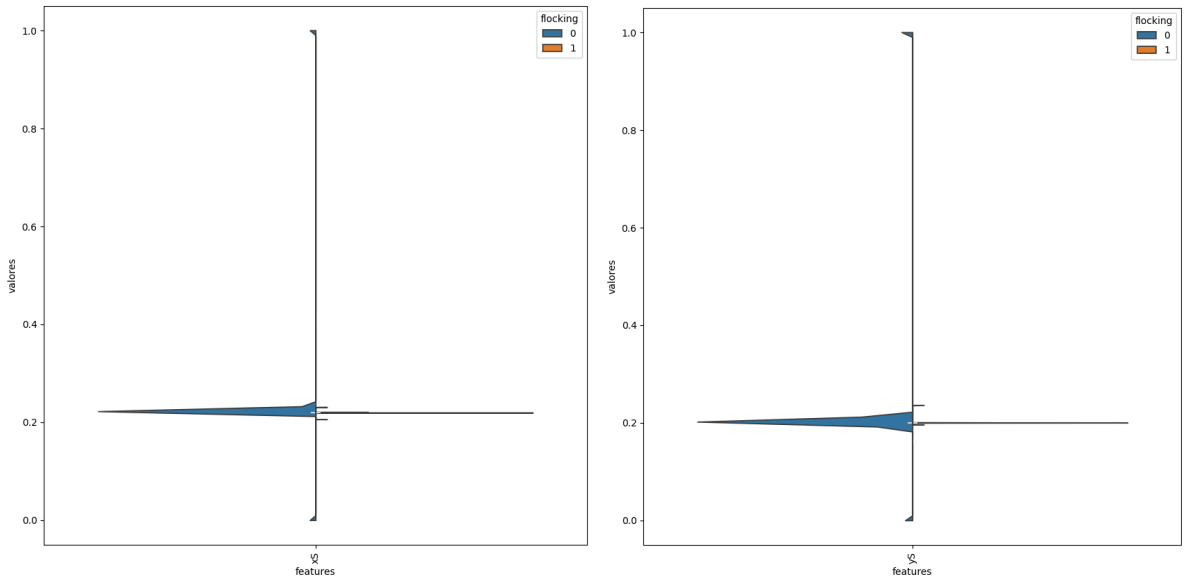


Figura 11: Gráfico de violino de xS e yS

Dessa forma, apesar dessa feature ser parte da lógica da movimentação dos *boids*, uma vez que essa feature possui muitos outliers, e a maioria dos seus dados se encontram próximo de zero ou da mediana, então optamos por remover essa feature, que irá resultar em uma redução de 400 features das 2400 features do dataset, uma vez que cada instância possui 200 features. Por fim, foi feito uma análise de correlação das features, com um limiar de corte de 90%, porém as features com maior correlação foi de 60%:

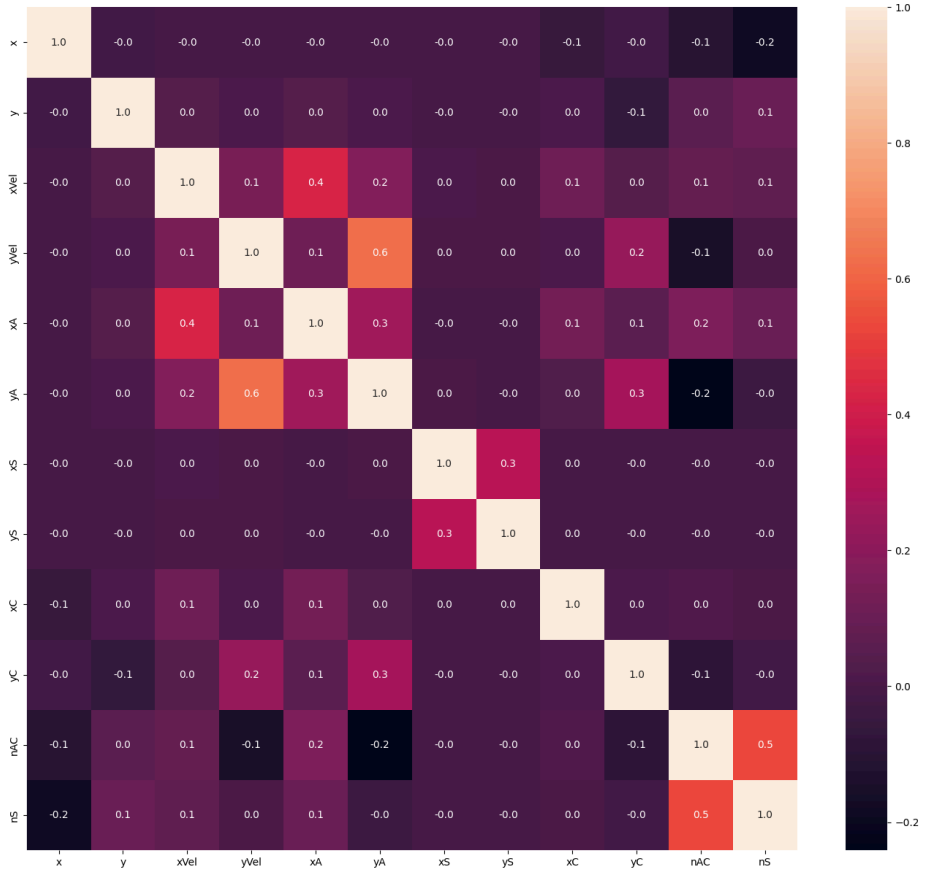


Figura 12: Gráfico de correlação entre as features

Ademais, foi realizado uma seleção de features por teste univariado, que aplica testes estatísticos para encontrar relações entre a variável de saída e cada variável de entrada isoladamente, no qual cada teste é realizados uma variável de entrada por vez. Foi utilizado uma tarefa de classificação utilizando a pontuação f-classif e um p-value para cada variável, o código pode ser visto abaixo:

```
from sklearn.feature_selection import SelectKBest, f_classif

feature_selector = SelectKBest(f_classif, k = "all")
fit = feature_selector.fit(x_df, y)

scores = pd.DataFrame(fit.scores_)
p_values = pd.DataFrame(fit.pvalues_)
input_variable_names = pd.DataFrame(x_df.columns)
summary_stats = pd.concat([input_variable_names, p_values, scores], axis = 1)
summary_stats.columns = ["input_variable", "p_value", "f_score"]
summary_stats.sort_values(by = "p_value", inplace = True)

p_value_threshold = 0.05
score_threshold = 5

selected_variables = summary_stats.loc[(summary_stats["f_score"] >= score_threshold) &
                                       (summary_stats["p_value"] <= p_value_threshold)]
selected_variables = selected_variables["input_variable"].tolist()
selected_variables
```

```
['x', 'xVel', 'yVel', 'xA', 'yA', 'xC', 'yC', 'nAC', 'nS', 'yS', 'xS']
```

Figura 13: Código de seleção de features por teste univariado

Uma vez que o gráfico de violino e a distribuição dos dados das features xS e yS, são de pouca variação, então escolhemos remove-las. Da mesma forma, a seleção por teste univariado removeu a feature y, que pode ter pouco impacto uma vez que possuímos a feature nS e nAC para determinar a quantidade de boids ao redor, logo, a posição dos boids não deve afetar tanto. Dessa forma, foram removidos as features y, xS e yS do dataset, reduzindo 600 features do dataset.

3.5. Eliminação de features

Tendo em vista que o nosso dataset possui muitas features correlacionadas (xN, yN e demais features) optamos por implementar o método Recursive Feature Elimination with Cross-Validation (RFECV). A escolha se fundamenta na eficácia comprovada do RFECV em selecionar características relevantes de maneira sistemática. Ao iniciar com todas as variáveis e, de forma recursiva e validação cruzada, seleciona as features a partir da sua importância, dada pela acurácia no estimador escolhido (SVC linear).

A inclusão do Cross-Validation no RFECV proporciona uma avaliação mais abrangente, dividindo os dados em conjuntos para treinar e validar modelos iterativamente. Para que possa acelerar o processamento, nós testamos a remoção de 9 em 9 features, uma vez que das 12 foram removidas 3. Como se pode ver na imagem 14, o número de features com maior acurácia foi 114 features, que são informações dos vetores de alinhamento, coesão e velocidade e separação entre os *boids*.

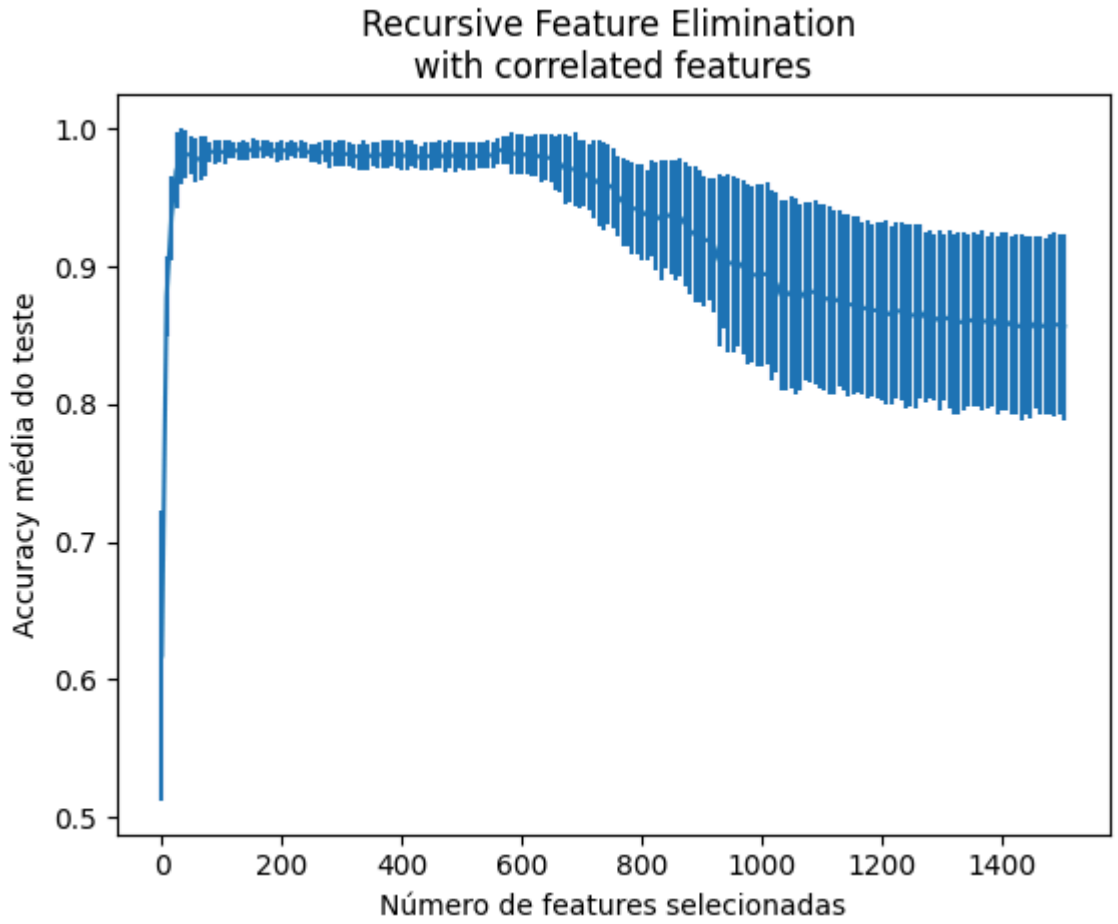


Figura 14: Redução de dimensionalidade com RFECV

4. Modeling

4.1 Estratégia de busca de hiperparâmetros

Na busca pelos melhores hiperparâmetros, empregamos uma abordagem de busca aleatória (Random Search) de um grande espaço amostral de hiperparâmetros, selecionando até 25 combinações de parâmaetros, no qual cada combinação é avaliada por meio de validação cruzada estratificada pela métrica de “accuracy”, utilizando a divisão de 10 folds.

Após a busca de hiperparâmetros, é realizado uma etapa de diminuição do espaço amostral dos hiperparâmetros, no qual são considerados “melhores parâemtros” aqueles que possuem desempenho maior que a metade dos selecionados, e os demais como “piores parâmetros”, então são removidos os parâmetros presentes nos “piores parâmetros” que não estão presentes nos “melhores parâmetros”. Então é executado novamente a busca com o novo espaço amostral de hiperparâmetros.

Por fim, essa etapa é realizada 3 vezes, mesclando os resultados de todas as buscas realizadas, e selecionado os parâmetros com maior acurácia. Essas etapas são realizadas para a busca de hiper parâmetros de cada um dos modelos, de forma a escolher os melhores hiperparâmetros no menor tempo possível.

4.2 Espaço de hiperparâmetros

Nesta seção será mostrado o espaço de parâmetros no qual cada busca foi realizada, assim como quais são esses parâmetros, para cada um dos modelos. Após cada uma das análises, vamos mostrar o gráfico de barras que exibe cada uma das três tentativas das buscas de parâmetros ordenada por sua pontuação, no qual cada tentativa é separadas por uma cor. Abaixo, será exibido todas as tentativas juntas ordenadas por sua pontuação, para evidenciar a distribuição de acurácia da busca de parâmetros por completo.

A **Árvore de Decisão**, teve os parâmetros otimizados a partir de um espaço amostral de 128000 hiperparâmetros que são: critério de avaliação da divisão, estratégia de divisão dos nós, máximo de profundidade da árvore, número mínimo de amostras necessárias para dividir um nó interno, número mínimo de amostras para divisão e profundidade máxima foram otimizados, número mínimo de amostras necessárias para ser um nó folha, número máximo de recursos a serem considerados para a divisão, número máximo de folhas da árvore, mínimo de diminuição da impureza que é necessário para uma divisão e pesos de classe para ajustar a importância de cada classe.

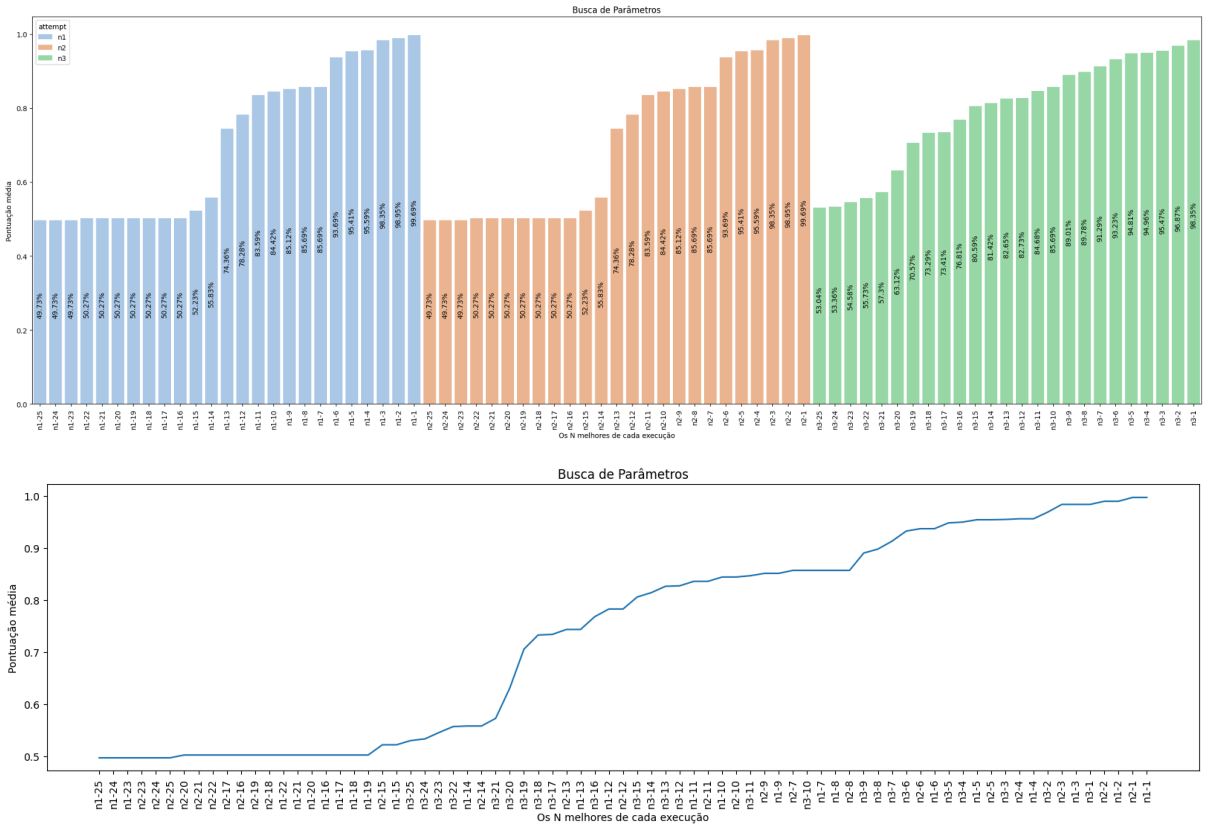


Figura 15: Acurácia da busca de parâmetros da árvore de decisão

No **SVM** (Máquinas de Vetores de Suporte), teve os parâmetros escolhidos a partir de um espaço amostral de 57600 hiperparâmetros, que são: parâmetro de regularização, tipo de kernel a ser usado no algoritmo, grau do kernel polinomial, parâmetro do kernel (gamma), termo independente (kernel polinomial ousigmoidal), se o algoritmo deve usar a técnica de enxugamento, se a probabilidade deve ser calculada, tolerância para a otimização, pesos das classes e máximo de iterações.

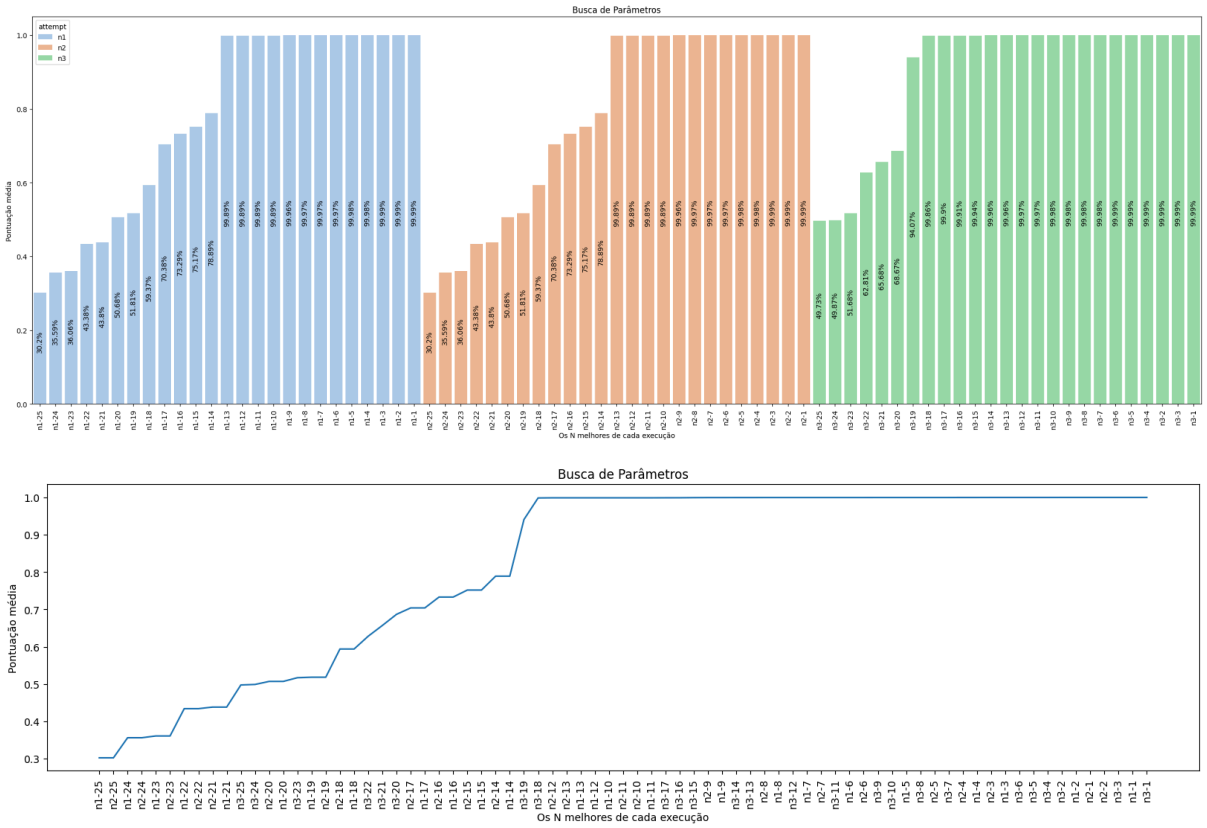


Figura 16: Acurácia da busca de parâmetros do SVM

No **KNN** (K vizinhos mais próximos), teve os parâmetros escolhidos a partir de um espaço amostral de 2560 hiperparâmetros ajustamos os parâmetros do número de vizinhos a serem considerados, tipo de peso a ser usado na votação, algoritmo a ser usado para a busca de vizinhos mais próximos, tamanho da folha (leaf_size) caso o algoritmo for BallTree ou KDTree, parâmetro de potência (p) para a métrica Minkowski e métrica a ser usada para a computação de distância.

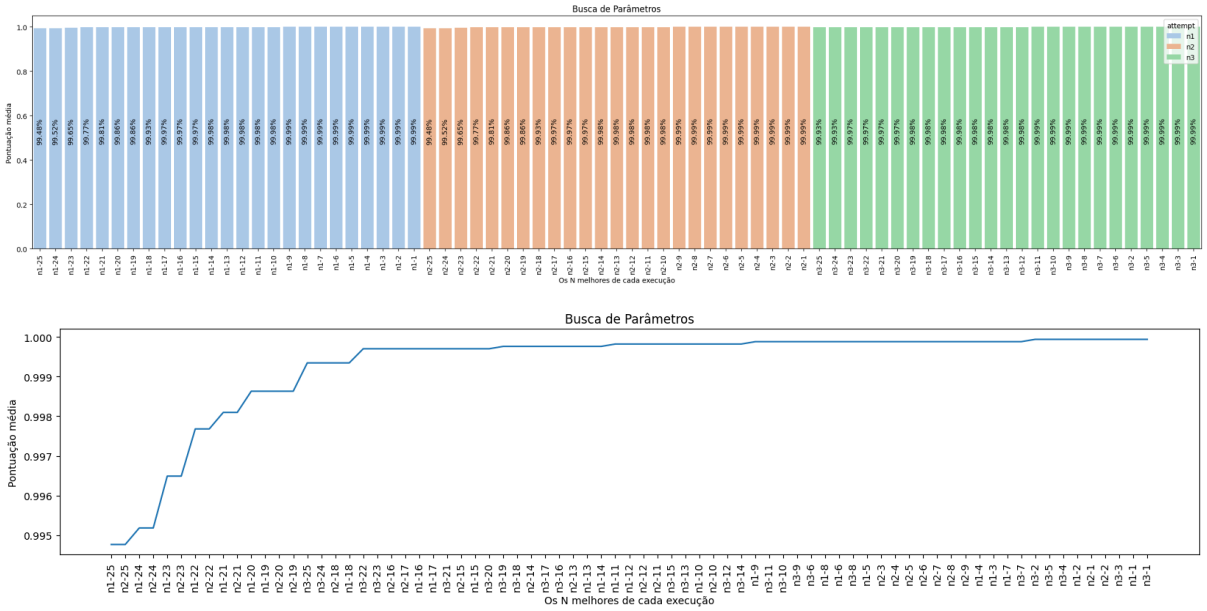


Figura 17: Acurácia da busca de parâmetros do KNN

Para o **MLP** foram otimizados parâmetros a partir de um espaço amostral de 95551488 combinações de hiper-parâmetros, a partir dos seguintes parâmetros: Número de neurônios em cada camada oculta, função de ativação, algorítimo de otimização, parâmetro de regularização, número de amostras por lote, taxa de aprendizado, máximo de iterações para o algoritmo de otimização, indicação se os exemplos devem ser embaralhados antes de cada época, tolerância de otimização, indicação se deve reutilizar a solução da chamada anterior como inicialização, momento para o gradiente descendente, indicação se deve usar o momento de Nesterov, indicação se deve usar o early stopping, proporção do conjunto a ser utilizada como validação para o early stopping e os parâmetros para o algoritmo 'adam'.

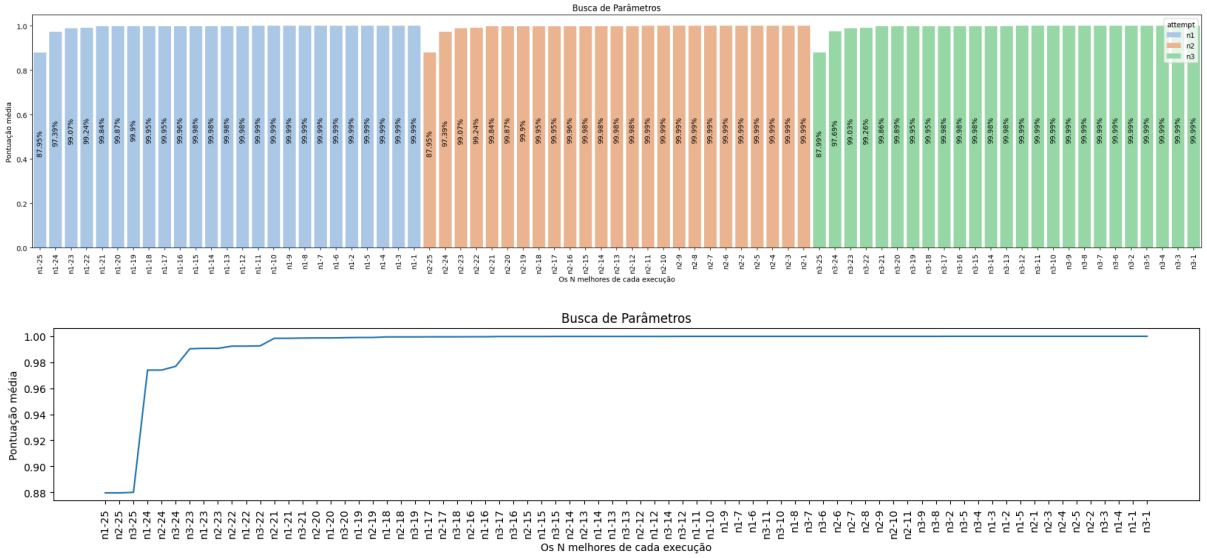
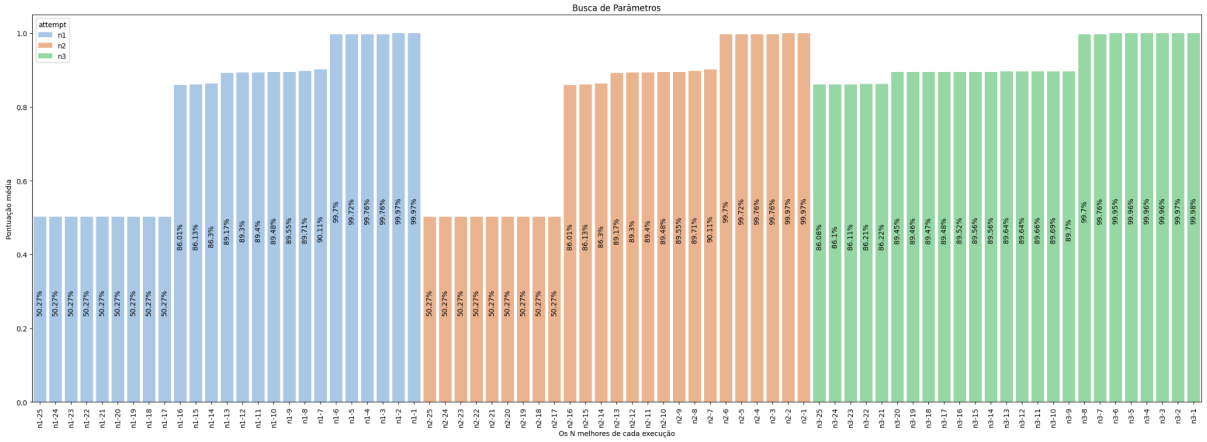


Figura 18: Acurácia da busca de parâmetros do MLP

Para o classificador **Random Forest** (modelo de floresta aleatória) foram otimizados parâmetros a partir de um espaço amostral de 82080000 combinações de hiper-parâmetros, a partir dos seguintes parâmetros: Número de árvores na floresta, profundidade máxima de cada árvore, número mínimo de amostras para a divisão do nó, número mínimo de amostras para ser um nó folha, número máximo de features a serem considerados na divisão, número máximo de folhas da árvore, mínimo de diminuição da impureza para divisão, indicação se deve usar amostras fora da bolsa para estimar a pontuação e indicação se deve reutilizar a solução anterior para a inicialização.



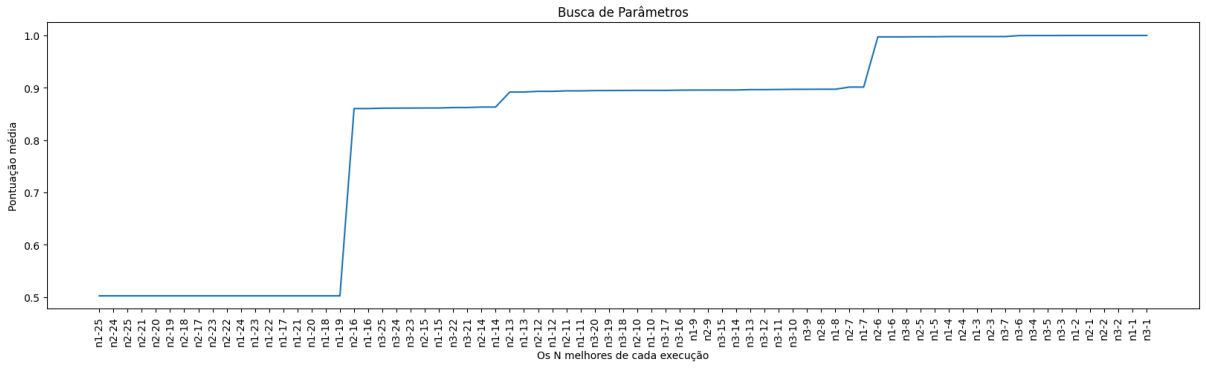


Figura 19: Acurácia da busca de parâmetros do RandomForest

No caso do LVQ (Learning Vector Quantization) foi necessário utilizar a normalização do MinMaxScaler antes de cada treino através de um pipeline, justamente por causa da natureza do algoritmo. Feito esse pipeline, o modelo foi treinado em um espaço de apenas 40 parâmetros que são: A métrica do cálculo de distância, o algoritmo de ativação e qual é o algoritmo de otimização que será utilizado.

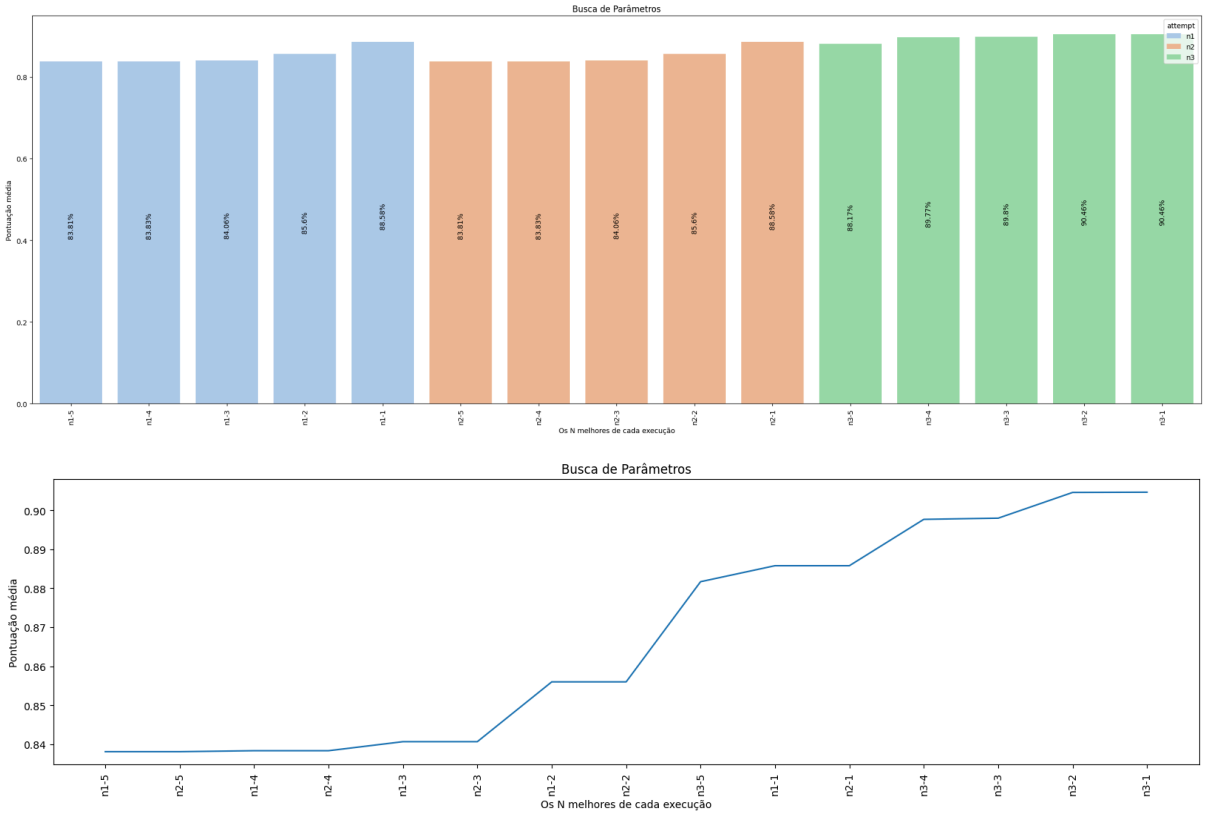


Figura 20: Acurácia da busca de parâmetros do LVQ

Além destes modelos, foi também treinado e avaliados um **comitê heterogêneo** e um **comitê de redes neurais**, que não foi necessário realizar uma nova busca de parâmetros, pois basta utilizar os melhores modelos obtidos para compor cada um desses comitês. No caso do comitê heterogêneo foi utilizado o melhor modelo da árvore de decisão, SVM e KNN, enquanto o comitê de redes neurais foi treinado com os 3 melhores modelos MLP. Cada um desses comitês foram treinados e foram obtidos a acurácia média dos mesmos através da validação cruzada com 10 folds.

4.3 Melhores hiperparâmetros

Uma realizado a busca de parâmetros, foi obtido o modelo mais otimizado, ou seja, o modelo com os parâmetros que resultaram na maior acurácia média. Nesta seção, média para cada um dos algoritmos, será mostrado a acurácia do melhor modelo encontrado, assim como quais são os parâmetros este modelo está utilizando.

A **Árvore de Decisão**, com hiperparâmetros otimizados, obteve a acurácia de 99.69%. Os melhores parâmetros encontrados são: `class_weight: 'balanced', criterion: 'gini', max_depth: 10, max_features: None, max_leaf_nodes: 30, min_impurity_decrease: 0.0, min_samples_leaf: 3, min_samples_split: 6, splitter: 'best'`. O **SVM** conseguiu uma acurácia média de 99.99%, mesmo não conseguindo convergir, uma vez que foi limitado pelo parâmetro de máximo de iterações, os melhores hiperparâmetros foram: `tol: 0.0001, shrinking: True, probability: False, max_iter: 100, kernel: 'rbf', gamma: 'scale', degree: 3, coef0: 0.2, class_weight: 'balanced', C: 100`.

O **KNN** alcançou uma acurácia de 99.99% com os seguintes parâmetros: `weights: 'uniform', p: 3, n_neighbors: 3, metric: 'manhattan', leaf_size: 30, algorithm: 'brute'`. O classificador **MLP** obteve uma acurácia de 99,99% a partir dos seguintes parâmetros otimizados: `warm_start: False, validation_fraction: 0.2, tol: 0.001, solver: 'lbfgs', shuffle: False, momentum: 0.99, nesterovs_momentum: False, hidden_layer_sizes: (150,), max_iter: 200, learning_rate: 'adaptive', epsilon: 1e-08, early_stopping: True, beta_2: 0.999, beta_1: 0.9, batch_size: 200, alpha: 0.001, activation: 'tanh'`.

O melhor modelo do algoritmo de **RandomForest** alcançou uma acurácia média de 99.97% com os seguintes parâmetros: `max_depth: 15, max_features: None, max_leaf_nodes: None, min_impurity_decrease: 0.0, min_samples_leaf: 2, oob_score: False, min_samples_split: 2, n_estimators: 491, warm_start: True`. Por fim, o melhor modelo do algoritmo **LVQ** alcançou uma acurácia média de 90.46% com os seguintes parâmetros: `activation_type: 'swish', solver_type: 'bgd', distance_type: 'squared-euclidean'`.

O **comitê heterogêneo** obteve uma acurácia média de 99,99% com o melhor modelo da árvore de decisão, SVM e KNN que já foram descritos anteriormente. Por fim, o **comitê de redes neurais** obteve uma acurácia média de 100%, no qual além de ser composto pelo melhor modelo MLP já descrito anteriormente, ainda possuia os seguintes modelos: `MLP(warm_start: True, validation_fraction: 0.15, tol: 0.001, solver: 'lbfgs', momentum: 0.99, max_iter: 400, hidden_layer_sizes: (50, 50), epsilon: 1e-07, early_stopping: True, beta_1: 0.99, batch_size: 200, activation: 'identity')` e `MLP(solver: 'lbfgs', shuffle: False, tol: 0.001, validation_fraction: 0.2, max_iter: 600, learning_rate: 'adaptive', hidden_layer_sizes: (200, 200), epsilon: 1e-07, beta_1: 0.99, alpha: 0.1, activation: 'logistic')`.

5. Evalutation

Foi realizado a validação cruzada com os modelos com os melhores parâmetros obtidos na etapa anterior, para cada um dos algoritmos, e realizado testes estatísticos no qual é comparado a distirbuição de cada um dos resultados. É importante ressaltar que essas conclusões são específicas para este conjunto de dados e para a métrica de avaliação utilizada. Outros conjuntos de dados ou métricas podem resultar em conclusões diferentes.

5.1 Teste de Kruskal-Wallis

Foi realizado o teste de Kruskal-Wallis para verificar a distribuição dos resultados. O teste de Kruskal-Wallis rejeitou a hipótese nula de que todas as amostras têm a mesma distribuição. A maioria das comparações entre pares de modelos resultou em valores p muito baixos ($p < 0.05$), indicando diferenças significativas entre as distribuições desses modelos. Isso significa que há evidências estatísticas para sugerir que pelo menos uma das amostras tem uma distribuição diferente das outras.

```
Distribuições diferentes (reject H0)

Comparison stats 63.43611250303086
Comparação Árvore | SVM: KruskalResult(statistic=15.754560530679925, pvalue=7.211388256866276e-05)
Comparação Árvore | KNN: KruskalResult(statistic=15.754560530679925, pvalue=7.211388256866276e-05)
Comparação Árvore | RedeNeural: KruskalResult(statistic=15.754560530679925, pvalue=7.211388256866276e-05)
Comparação Árvore | RandomForest: KruskalResult(statistic=14.413586097946286, pvalue=0.0001467398156898854)
Comparação Árvore | LVQ: KruskalResult(statistic=14.339622641509425, pvalue=0.00015261860208363206)
Comparação Árvore | ComiteHet: KruskalResult(statistic=15.754560530679925, pvalue=7.211388256866276e-05)
Comparação Árvore | ComiteRede: KruskalResult(statistic=15.754560530679925, pvalue=7.211388256866276e-05)
Comparação SVM | KNN: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação SVM | RedeNeural: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação SVM | RandomForest: KruskalResult(statistic=1.5486902927580706, pvalue=0.21332889331334104)
Comparação SVM | LVQ: KruskalResult(statistic=15.715467328370545, pvalue=7.361969459771538e-05)
Comparação SVM | ComiteHet: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação SVM | ComiteRede: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação KNN | RedeNeural: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação KNN | RandomForest: KruskalResult(statistic=1.5486902927580706, pvalue=0.21332889331334104)
Comparação KNN | LVQ: KruskalResult(statistic=15.715467328370545, pvalue=7.361969459771538e-05)
Comparação KNN | ComiteHet: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação KNN | ComiteRede: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação RedeNeural | RandomForest: KruskalResult(statistic=1.5486902927580706, pvalue=0.21332889331334104)
Comparação RedeNeural | LVQ: KruskalResult(statistic=15.715467328370545, pvalue=7.361969459771538e-05)
Comparação RedeNeural | ComiteHet: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação RedeNeural | ComiteRede: KruskalResult(statistic=0.0, pvalue=1.0)
Comparação RandomForest | LVQ: KruskalResult(statistic=14.937106918238985, pvalue=0.00011115505168600113)
Comparação RandomForest | ComiteHet: KruskalResult(statistic=1.5486902927580706, pvalue=0.21332889331334104)
Comparação RandomForest | ComiteRede: KruskalResult(statistic=1.5486902927580706, pvalue=0.21332889331334104)
Comparação LVQ | ComiteHet: KruskalResult(statistic=15.715467328370545, pvalue=7.361969459771538e-05)
Comparação LVQ | ComiteRede: KruskalResult(statistic=15.715467328370545, pvalue=7.361969459771538e-05)
Comparação ComiteHet | ComiteRede: KruskalResult(statistic=0.0, pvalue=1.0)
```

Figura 21: Resultado do Teste de Kruskal

As comparações entre pares de modelos, como Árvore de Decisão (Arvore) versus SVM, Árvore de Decisão versus KNN, Árvore de Decisão versus Redes Neurais (RedeNeural), entre outros, mostram valores p baixos, o que sugere diferenças significativas em seus desempenhos. No entanto, algumas comparações, como SVM versus KNN, RandomForest versus Rede Neural, e outras, resultaram em valores p altos ($p > 0.05$), indicando que não há diferenças significativas entre os resultados desses modelos.

Algumas comparações também resultaram em valores p significativos entre o LVQ e outros modelos, indicando que o desempenho do LVQ pode ser estatisticamente diferente de outros modelos. Além disso, como o teste positivo foi igual a 1 para o resultado entre KNN, SVM e Comitê Heterogêneo, não podemos afirmar que há diferenças estatísticas significativas no desempenho dos três. Essa descoberta sugere que, para este conjunto de dados em particular e para a métrica de avaliação escolhida, os três modelos têm desempenhos bastante similares.

5.2 Teste de Tukey

Tendo em vista que o Teste de Kruskal obteve resultados poucos satisfatórios, decidimos por realizar o teste de Tukey, onde cada modelo é comparado com os outros em pares. Nesse caso, cada modelo é representado por um grupo, seguindo a seguinte ordem: Árvore de decisão, SVM, KNN, MLP, Random Forest, LVQ, Comitê heterogêneo e comitê de redes neurais. Na análise, a coluna "meandiff" mostra a diferença média entre os grupos, enquanto o valor "p-adj" é ajustado para corrigir a comparação múltipla:

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
0	1	0.0032	0.2245	-0.0008	0.0071	False
0	2	0.0032	0.2245	-0.0008	0.0071	False
0	3	0.0032	0.2245	-0.0008	0.0071	False
0	4	0.0029	0.3162	-0.0011	0.0069	False
0	5	-0.0917	0.0	-0.0957	-0.0877	True
0	6	0.0032	0.2245	-0.0008	0.0071	False
0	7	0.0032	0.2245	-0.0008	0.0071	False
1	2	0.0	1.0	-0.004	0.004	False
1	3	0.0	1.0	-0.004	0.004	False
1	4	-0.0002	1.0	-0.0042	0.0037	False
1	5	-0.0948	0.0	-0.0988	-0.0908	True
1	6	0.0	1.0	-0.004	0.004	False
1	7	0.0	1.0	-0.004	0.004	False
2	3	0.0	1.0	-0.004	0.004	False
2	4	-0.0002	1.0	-0.0042	0.0037	False
2	5	-0.0948	0.0	-0.0988	-0.0908	True
2	6	0.0	1.0	-0.004	0.004	False
2	7	0.0	1.0	-0.004	0.004	False
3	4	-0.0002	1.0	-0.0042	0.0037	False
3	5	-0.0948	0.0	-0.0988	-0.0908	True
3	6	0.0	1.0	-0.004	0.004	False
3	7	0.0	1.0	-0.004	0.004	False
4	5	-0.0946	0.0	-0.0986	-0.0906	True
4	6	0.0002	1.0	-0.0037	0.0042	False
4	7	0.0002	1.0	-0.0037	0.0042	False
5	6	0.0948	0.0	0.0908	0.0988	True
5	7	0.0948	0.0	0.0908	0.0988	True
6	7	0.0	1.0	-0.004	0.004	False

Figura 22: Resultados do Teste de Tukey

Na saída, o valor "reject" indica se a diferença é significativa ou não. A análise revela que, com base nos resultados, existem diferenças significativas particularmente envolvendo o LVQ (group5), para com todos os demais modelos, mostrando que a comparação que será feito a seguir não deve-se considerar o LVQ.

5.3 Comparação dos resultados

Os resultados da avaliação de desempenho dos modelos de aprendizado de máquina indicam que a maioria dos modelos obteve altas taxas de acurácia média, superiores a 99%. Especificamente, os modelos Árvore de Decisão, SVM, KNN e Rede Neural alcançaram uma acurácia média de 99.99%, enquanto o modelo Random Forest obteve uma acurácia média de 99.97%. O Comitê Heterogêneo e o Comitê de Redes Neurais também alcançaram uma acurácia média de 99.99%, indicando consistência em seu desempenho. Abaixo é exibido a comparação em forma de boxplot para cada um dos modelos:

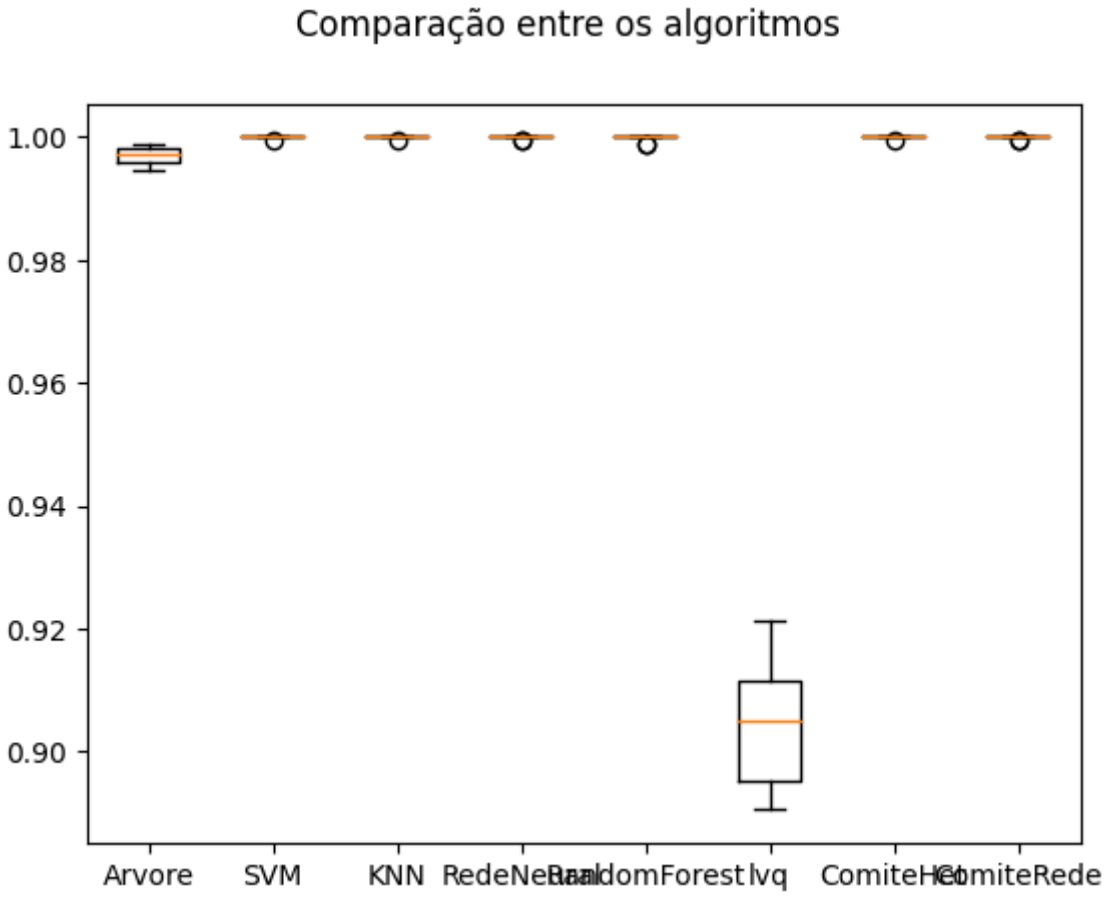


Figura 23: Resultado do Teste de Kruskal

O modelo LVQ, embora tenha apresentado uma acurácia média ligeiramente inferior de 90.51%, ainda demonstrou um desempenho competitivo, mas como vimos na etapa anterior, possui uma distribuição diferente das demais, não possibilitando a comparação. Estes resultados sugerem que os modelos avaliados são capazes de realizar previsões com alta precisão em relação aos dados de treinamento fornecidos. Abaixo é mostrado a matriz de confusão para cada um dos algoritmos calculados.

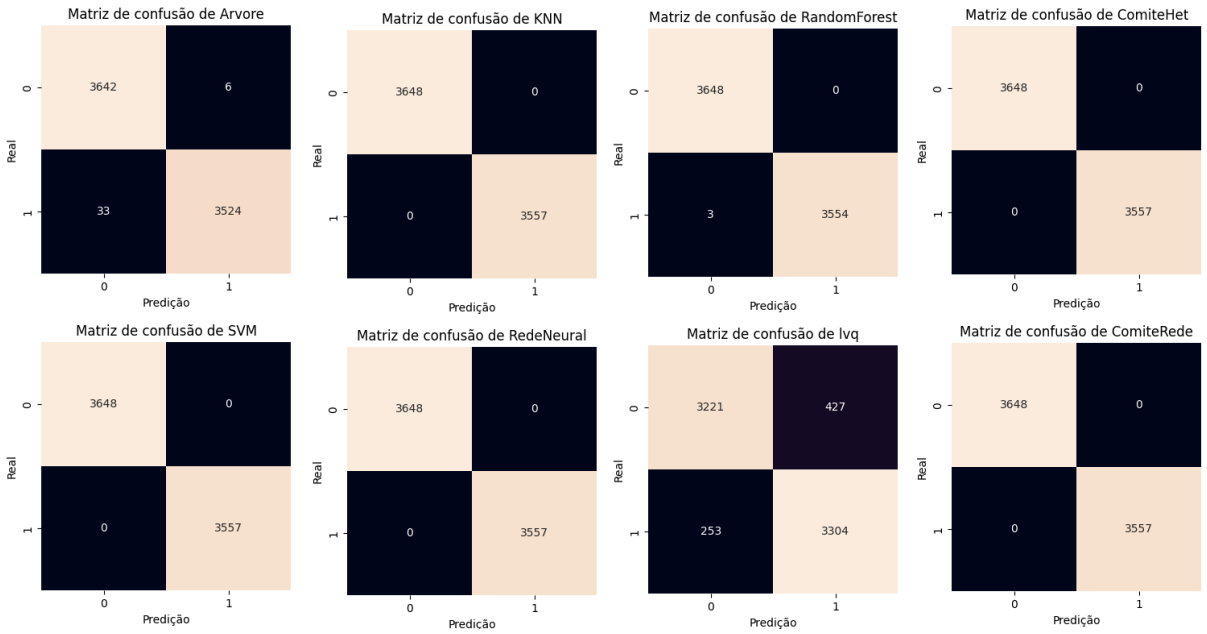


Figura 24: Matriz de confusao de cada algoritmo