Comparação de Desempenho de Algoritmos de Aprendizado de Máquinas Supervisionados

Vitor Galioti Martini - 135543
Universidade Federal de São Paulo, UNIFESP
São José dos Campos, São Paulo, Brasil
galioti.martini@unifesp.br

***Abstract*— This project aims to analyze the performance of different machine learning algorithms submitted to a dataset with clinical data that predict stroke events. (*Abstract*)**

Keywords*— *aprendizado de maquinas, dataset, algoritmo, desempenho, AVC, acidente vascular cerebral

I. INTRODUÇÃO

De acordo com a Organização Mundial da Saúde (OMS), dados recentes mostram que o AVC (acidente vascular cerebral) é a 2° complicação de saúde que mais causas mortes no mundo, responsável por aproximadamente 11% do total de mortes até 2019. [1]

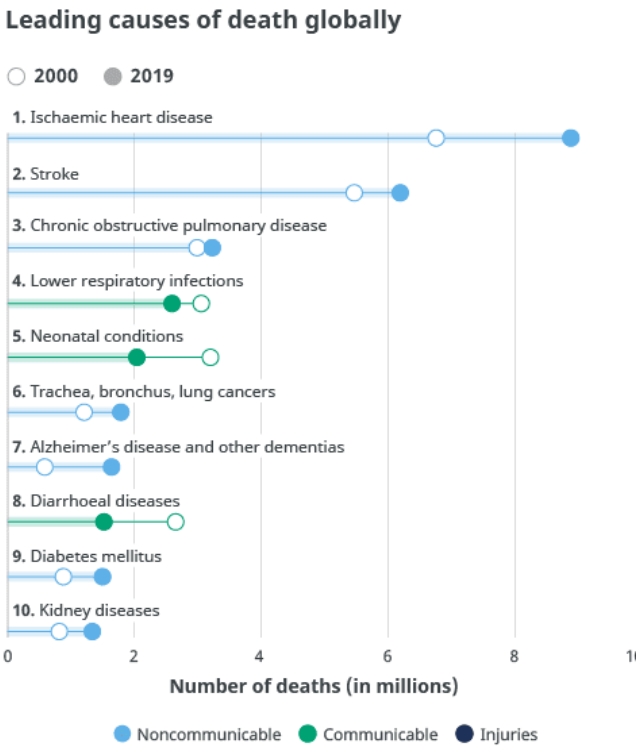


Fig 1. Principais causas de morte no mundo.

Tendo em vista essa situação, o dataset utilizado busca prever se um paciente tem probabilidade de desenvolver AVC baseado nos parâmetros observados.

A partir desse conjunto de dados, o objetivo do projeto é implementar quatro algoritmos de aprendizado de máquina supervisionados e analisar seus resultados e desempenhos, em específico a precisão de acertos e o tempo de execução.

II. METODOLOGIA

O projeto visa analisar o desempenho de 4 algoritmos de aprendizado de máquina aplicados no dataset "Stroke Prediction" [2], e verificar qual deles tem o melhor desempenho.

A. *Dataset*

O dataset, contém 5110 registros, e suas colunas são:

- ID: Identificador único da linha
- Gênero (gender): Categórico. Masculino (male), feminino (female) e outros (other)
- Idade (age): Numérico
- Hipertensão (hypertension): Binário. 0 para não, 1 para sim.
- Doença cardíaca (heart_disease): Binário. 0 para não, 1 para sim.
- Já foi casado (ever_married): Binário. Não (no), Yes (sim)
- Tipo de trabalho (work_type): Categórico. Criança (children), Funcionário público (govt_job), Nunca trabalhou (never_worked), Setor privado (private), Trabalhador autônomo (self_employed)
- Tipo de residência (Residence_type): Categórico. Urbano (urban), rural (Rural).
- Média de glucose (avg_glucose_level): Numérico. Média de glucose no sangue.
- IMC (BMI): Número. Índice de massa corporal
- Status de fumante (smoking_status): Categórico. Costumava fumar (formerly smoked), Nunca fumou (never smoked), Fuma (smokes), Inválido para o paciente (Unknown).

	A
1	id,gender,age,hypertension,heart_disease,ever_married,work_type,Residence_type,avg_glucose_level,bmi,smoking_status,stroke
2	9046,Male,67,0,1,Yes,Private,Urban,228.69,36.6,formerly smoked,1
3	51676,Female,61,0,0,Yes,Self-employed,Rural,202.21,N/A,never smoked,1
4	31112,Male,80,0,1,Yes,Private,Rural,105.92,32.5,never smoked,1
5	60182,Female,49,0,0,Yes,Private,Urban,171.23,34.4,smokes,1
6	1665,Female,79,1,0,Yes,Self-employed,Rural,174.12,24,never smoked,1

Fig 2. Cinco primeiras linhas do dataset analisado.

B. *Tratamento de dados*

No código implementado, a primeira função executada é a "trata_dados", nela, trata-se as informações do dataset para que fiquem coerentes com os algoritmos. O conjunto de dados é importado para um DataFrame da biblioteca Pandas, então são removidas as linhas que contém colunas vazias através do método "dropna":

```
# Importa dataset
df = pd.read_csv('healthcare-dataset-stroke-data.csv')

# Remove valores nulos
df.dropna(inplace=True)
```

Fig 3. Importação do dataset e remoção de valores nulos.

Depois disso, transforma-se os dados categóricos em numéricos através da função LabelEncoder da biblioteca

sklearn e remove-se a coluna ID, já que esta não representa um atributo a ser analisado.

```
# Transforma os dados categóricos em numéricos
le = LabelEncoder()
le.fit(df['smoking_status'])
df['smoking_status'] = le.transform(df['smoking_status'])
le.fit(df['work_type'])
df['work_type'] = le.transform(df['work_type'])
le.fit(df['Residence_type'])
df['Residence_type'] = le.transform(df['Residence_type'])
le.fit(df['ever_married'])
df['ever_married'] = le.transform(df['ever_married'])
le.fit(df['gender'])
df['gender'] = le.transform(df['gender'])

# Remove coluna ID
df.drop(['id'], axis=1)
```

Fig 4. Transformando os dados categóricos.

Como tem-se uma quantidade muito elevada de registros da classe que não tem AVC (stroke = 0) em relação a classe que tem (stroke = 1), faz-se necessário um balanceamento dos dados amostrais. Para isso, foi utilizado o método upsampling [3], onde a aumentamos os registros da classe minoritária através do método resample da biblioteca sklearn. Além disso, embaralha-se os registros e elimina-se a coluna que identifica a classe.

```
# Como a classe 0 aparece muito mais que a 1, vamos equilibrar reamostrando os dados com o método Upsampling,
# aumentando os registros da classe 1
no_stroke = df[df['stroke'] == 0]
stroke = df[df['stroke'] == 1]
upsampling = resample(stroke, replace=True, n_samples=no_stroke.shape[0])
df = pd.concat([no_stroke, upsampling])
df = shuffle(df)

x = df.drop(['stroke'], axis=1)
y = df['stroke']
```

Fig 5. Método upsampling

Por fim, separa-se 70% dos dados para treino e 30% para teste, e são executados os algoritmos.

```
# Separa 70% dos dados para treino e o restante para teste
x_treino, x_teste, y_treino, y_teste = train_test_split(x, y, train_size=0.7, random_state=101)

# Cria classe
d = DataSet(x_treino, x_teste, y_treino, y_teste)

# Executa o algoritmos
d.executar_algoritmo('KNN')
d.executar_algoritmo('DT')
d.executar_algoritmo('RF')
d.executar_algoritmo('NB')
d.plotar_graficos()
```

Fig 6. Prepara os dados de treino e teste e chama a execução dos algoritmos.

C. Algoritmos utilizados

Os algoritmos de aprendizado supervisionado envolvem o aprendizado de uma função a partir de exemplos de sua entrada e saída. Os algoritmos utilizados nesse projeto são:

a) K-Nearest Neighbours: Este é um algoritmo de classificação onde o aprendizado é baseado "no quão similar" um registro é dos seus K vizinhos mais próximos, definindo assim a qual classe pertence.

Na prática, ao avaliar a similaridade dos registros, estamos medindo as distâncias entre eles, quanto menor a distância, mais similares eles são. A distância entre os pontos pode ser medida utilizando a métrica Euclidiana, de Hamming, Manhattan, Minkowski, dentre outras. O código implementado aplica o método KNeighborsClassifier da biblioteca SKLearn, e por padrão a distância utilizada é a Minkowski, representada por: soma(|x - y|^p)^(1/p), onde o valor padrão de p é 2 [4].

Para um exemplo [5], considere a base de dados abaixo, onde os registros são classificados nas cores azul e vermelho:

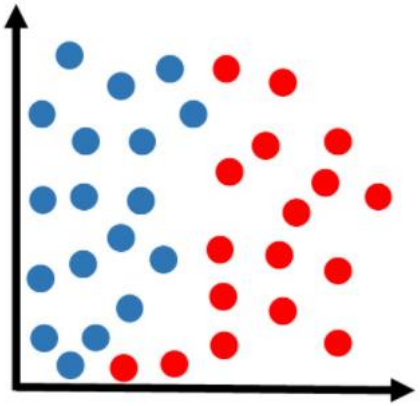


Fig 7. Dataset explicativo para KNN.

Inserida uma amostra ainda não classificada, representada pela cor verde, o objetivo é definir a qual classe essa pertence.

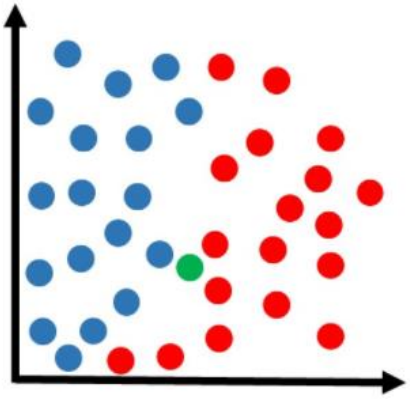


Fig 8. Registro a se classificar.

Considerando K = 5, é analisada a classe dos 5 vizinhos mais próximos. A classe do novo registro será definida de acordo com a majoritária a sua volta. Para o exemplo, como dois dos vizinhos são vermelhos e três azuis, então a classe do novo registro é azul:

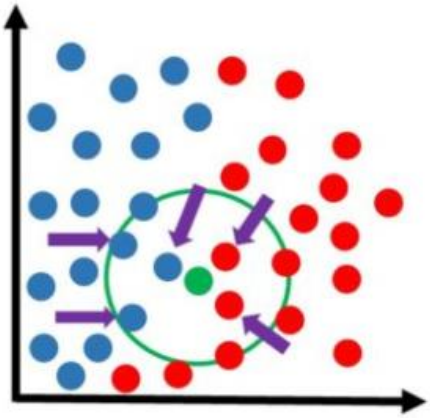


Fig 9. Cinco vizinhos mais próximos.

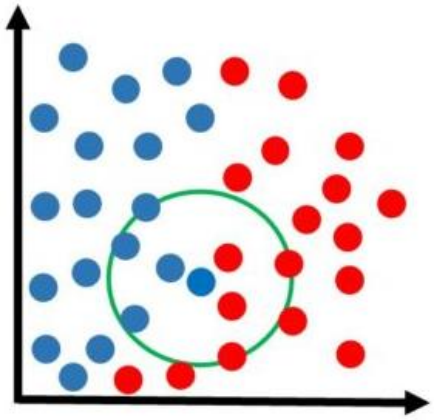


Fig 10. Definida a classe do novo registro.

Não existe um valor ideal para K, esse varia de acordo com o dataset implementado. Porém existem algumas considerações a se fazer:

- K ser um inteiro ímpar: Como o objetivo é escolher a classe baseado nos registros mais próximos, escolher um valor par pode ocasionar um empate.
- Tomar um valor de K muito pequeno torna a classificação sensível às regiões mais próximas, podendo ocorrer um overfitting, isto é, o algoritmo não tem capacidade de generalização e fica tendencioso aos parâmetros dos dados de teste. [6]
- Escolher um K grande ocorre o contrário, a classificação fica menos tendenciosa, porém está sujeita a problemas de underfitting, isto é, o algoritmo não consegue ser treinado e não tem um desempenho satisfatório.

b) Decision Tree: Na árvore de decisão é criado um fluxograma com diversos nós de decisão, e o resultado desses nós criará novos caminhos, os ramos. Os nós de decisão serão perguntas binárias e de acordo com a resposta, serão tomados caminhos diferentes. [7]

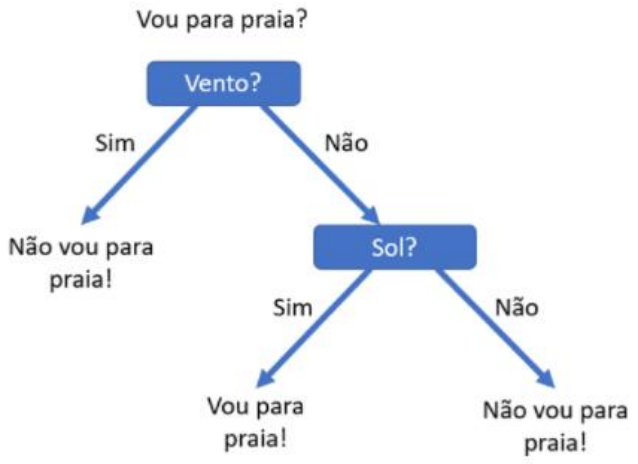


Fig 11. Exemplo de árvore de decisão.

Nas árvores de decisão, temos as variáveis target, que representam a classe à qual o registro irá pertencer, e as preditoras, que são os atributos daquele registro. De acordo com os valores representados pelas variáveis preditoras, o caminho será construído até definir um valor para a variável target.

Os métodos de previsão identificam as variáveis preditoras que possuem maior relação com a variável target, colocando-as no topo da árvore, em seus nós principais.

O algoritmo de árvore de decisão implementado utilizado método DecisionTreeClassifier() da biblioteca SKLearn, implementando o método de previsão "Índice GINI". A variável preditora com o menor índice Gini será a escolhida para o nó principal da árvore, pois um baixo valor do índice indica maior ordem na distribuição dos dados.

c) Random Forest: O algoritmo da floresta aleatória cria diversas árvores de decisão de modo randômico, onde cada árvore é utilizada na definição da classe. Este método utiliza o algoritmo de árvore de decisão visto anteriormente. [8]

O algoritmo Random Forest utiliza do método ensemble. Este método combina diversos modelos diferentes para obter um resultado único, o que aumenta o custo computacional, porém torna os resultados mais precisos. O algoritmo de floresta aleatória implementado utilizado método RandomForestClassifier() da biblioteca SKLearn, este cria 100 modelos de árvores de decisão, e dentre elas, escolhe a classe do registro baseado no modelo que mais se repete.

d) Naive Bayes: Este algoritmo utiliza o Teorema de Bayes com a hipótese de independência entre atributos, que por sua vez é uma fórmula matemática que calcula a probabilidade condicional, isto é, a probabilidade de um evento ocorrer dado que um outro evento já ocorreu. Esse cálculo é representado pela seguinte equação[9]:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Fig 12. Probabilidade condicional.

Onde:

- P(B|A): probabilidade de B ocorrer dado que A ocorreu
- P(A): probabilidade de A ocorrer
- P(B): probabilidade de B ocorrer

Para dados numéricos, primeiro deve-se discretizar os dados, como é o caso do dataset trabalhado, e então utiliza-se o Gaussian Naive Bayes, onde o algoritmo assume uma distribuição Gaussiana dos dados.

$$PDF(x, \bar{x}, \sigma) = \frac{1}{\sqrt{2 \times \pi} \times \sigma} \times e^{-\left(\frac{(\infty - \bar{x})^2}{2 \times \sigma^2}\right)}$$

Fig 13. Função de Gauss.

D. Implementação

Dentro do método "trata_dados", visto anteriormente, é chamado o procedimento "executar_algoritmo" para cada um dos algoritmos analisados. Esse método recebe como parâmetro a abreviação do algoritmo que será executado e seu objetivo é imprimir os dados de análise e guardar essas informações para posteriormente plotar os gráficos.

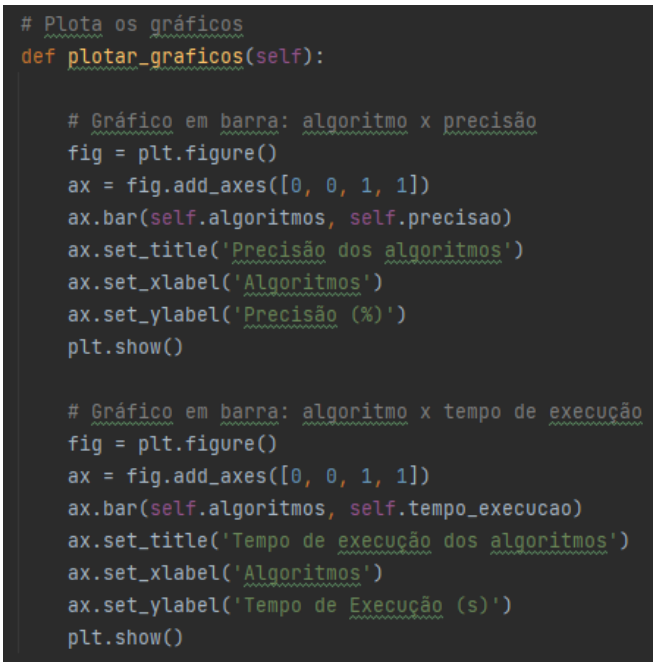
```
# Executa o algoritmo
def executar_algoritmo(self, algoritmo):

    # Verifica qual é o algoritmo
    if algoritmo == 'KNN':
        nome_algoritmo = 'K-Nearest Neighbours'
        execucao = KNeighborsClassifier(n_neighbors=5)
    if algoritmo == 'DT':
        nome_algoritmo = 'Decision Tree'
        execucao = DecisionTreeClassifier()
    if algoritmo == 'RF':
        nome_algoritmo = 'Random Forest'
        execucao = RandomForestClassifier()
    if algoritmo == 'NB':
        nome_algoritmo = 'Naive Bayes'
        execucao = GaussianNB()

    # Inicia variáveis
    precisao = 0
    tempo_execucao = 0

    # Executa o algoritmo 100 vezes
    for i in range(100):
        # Recebe o tempo antes da execução
        tempo_inicio = time.time()

        # Executa o algoritmo
        execucao.fit(self.x_treino, self.y_treino)
        predict = execucao.predict(self.x_teste)

        # Soma as precisões e os tempos de execução
        precisao += accuracy_score(self.y_teste, predict) * 100
        tempo_execucao += time.time() - tempo_inicio

        # Printa os dados
        print(nome_algoritmo)
        print("Precisão: " + str(format(precisao / 100, ".4f")) + "%")  # Média das precisões
        print("Tempo de execução: " + str(format(tempo_execucao / 100, ".4f")) + "s")  # Tempo médio das execuções
        print(" ")

        # Guarda os dados obtidos
        self.precisao.append(float(format(precisao / 100, ".4f")))
        self.tempo_execucao.append(float(format(tempo_execucao / 100, ".4f")))
        self.algoritmos.append(algoritmo)
```

Fig 14. Método executar_algoritmo.

O método segue o seguinte fluxo de operações:

1. Verifica qual algoritmo será executado e o define através da biblioteca SKLearn.
2. Inicia as variáveis de precisão e tempo de execução com 0.
3. Executa um loop de 100 iterações.
4. Dentro do loop, guarda a hora que foi chamado na variável “tempo_inicio”.
5. Treina o algoritmo e faz as previsões.
6. Adiciona a precisão e o tempo de execução nas variáveis definidas anteriormente.
7. Encerra o loop.
8. Exibe os dados para o usuário. Nome do algoritmo, média de precisão e tempo médio de execução.
9. Armazena as informações nas listas da classe DataSet.

Depois da coleta dos dados, é chamado o método “plotar_graficos”, conforme demonstrado na figura de número 6.



```
# Plota os gráficos
def plotar_graficos(self):

    # Gráfico em barra: algoritmo x precisão
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1])
    ax.bar(self.algoritmos, self.precisao)
    ax.set_title('Precisão dos algoritmos')
    ax.set_xlabel('Algoritmos')
    ax.set_ylabel('Precisão (%)')
    plt.show()

    # Gráfico em barra: algoritmo x tempo de execução
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1])
    ax.bar(self.algoritmos, self.tempo_execucao)
    ax.set_title('Tempo de execução dos algoritmos')
    ax.set_xlabel('Algoritmos')
    ax.set_ylabel('Tempo de Execução (s)')
    plt.show()
```

Fig 15. Método plotar_graficos.

Esse método tem como objetivo plotar dois gráficos em barra para analisar a precisão dos algoritmos e seu tempo de execução. É utilizada a biblioteca matplotlib.pyplot. No eixo X dos dois gráficos temos os algoritmos e no Y a precisão e o tempo de execução, respectivamente.

III. ANÁLISE EXPERIMENTAL

A. *Dados tratados*

Depois de discretizado, o dataset ficou da seguinte forma:

gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
1	41.0	0	0	1	3	0	62.93	26.1	3
0	14.0	0	0	0	4	0	57.93	30.9	0
0	75.0	0	0	1	2	0	226.73	43.7	2
0	78.0	0	0	1	2	0	154.75	17.6	2
0	23.0	0	0	0	2	0	112.30	26.6	0

Fig 16. Cinco primeiras linhas do dataset.

Note que os campos inicialmente textuais, como “work_type” e “smoking_status”, agora recebem valores numéricos, decorrente da função LabelEncoder() mencionada na figura 4.

B. *Configuração do Algoritmo e do ambiente computacional*

A linguagem do código desenvolvido é Python 3, e a máquina onde os algoritmos foram executados apresenta as configurações a seguir:

- AMD Ryzen 7 3800X 8-Core Processor 3.89 GHz
- 16 GB Memória RAM
- SO Windows 10 Home 64bits

.

C. *Critérios de Análise*

Executar cada um dos algoritmos 100 vezes e analisar a média de precisão e tempo de execução.

D. *Resultados e discussões*

Os resultados obtidos foram os seguites:

```
K-Nearest Neighbours
Precisão: 93.2979%
Tempo de execução: 0.0723s

Decision Tree
Precisão: 97.2195%
Tempo de execução: 0.0151s

Random Forest
Precisão: 99.6241%
Tempo de execução: 0.4347s

Naive Bayes
Precisão: 76.9858%
Tempo de execução: 0.0032s
```

Fig 17. Precisão e tempo de execução de cada um dos algoritmos.

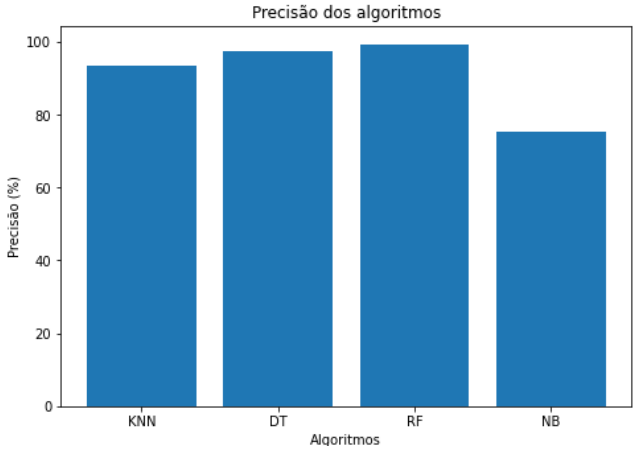


Fig 18. Gráfico em barras. Algoritmos x Precisão.

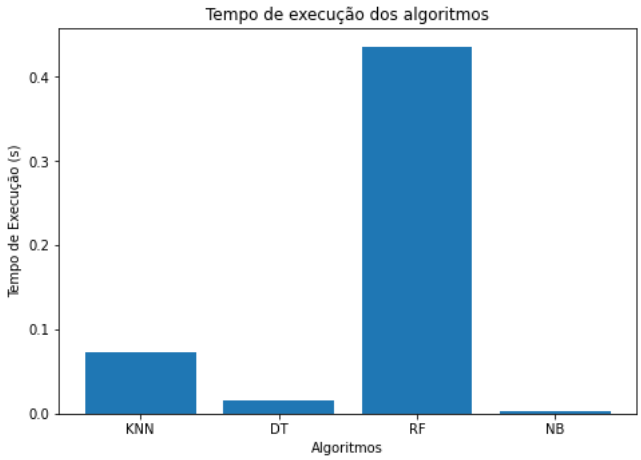


Fig 19. Gráfico em barras. Algoritmos x Tempo de execução.

Quanto a precisão, todos os algoritmos estudados tiverem precisão acima de 75%. O algoritmo com maior precisão foi o Random Forest (RF), com 99.6241% e o com menor foi o Naive Bayes (NB), com 76.9858%.

Já quanto ao tempo de execução, a situação foi contrária. Naive Bayes foi disparado o mais rápido, executando em 0.0032s, e o Random Forest, por ser mais robusto e exigir mais demanda computacional, o mais lento, demorando 0.4347s para executar.

Analisando os gráficos, se levarmos em consideração os valores de Precisão x Tempo de Execução, o algoritmo de árvore de decisão (DT) foi o que melhor se sobressaiu, podendo ser uma ótima escolha. Bem próximo do Random Forest, com 97.2195% de precisão, porém bem mais rápido, levando apenas 0.0151s para executar. Como apresentado anteriormente, o algoritmo Random Forest implementa árvores de decisões, o que justifica a similaridade na porcentagem de precisão e a sua demora no tempo de execução.

IV. CONCLUSÕES

A proposta do trabalho era analisar o desempenho de quatro algoritmos de aprendizado de máquina supervisionado, K-Nearest Neighbours, Árvore de Decisão, Floresta Aleatória e Naive Bayes, aplicados em um dataset que, a partir dos atributos dos registros, prevê se um paciente é propenso a ter um AVC ou não.

Depois de tratados os dados, discretizando os valores categóricos, cada algoritmo foi executado 100 vezes. Colhido os resultados, foi analisado os valores de forma numérica e representação gráfica. As conclusões foram satisfatórias e dentro do esperado.

O algoritmo Random Forest foi o com maior precisão, porém maior tempo de execução. Já o Naive Bayes apresentou o resultado oposto, com menor precisão e maior tempo de execução. E o algoritmo de árvore de decisão foi o mais completo, com uma precisão de apenas 3% abaixo da Floresta Aleatória, porém com um tempo de execução bem inferior.

REFÊNCIAS

[1] Organização Mundial da Saúde (OMS), “The top 10 causes of death”. Dísponivel em: https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death

[2] “fedesoriano”, “Stroke Prediction Dataset”. Disponível em: https://www.kaggle.com/fedesoriano/stroke-prediction-dataset

[3] Ege Hoşgüngör, “How to Handle Imbalance Data and Small Training Sets in ML”. Disponível em: https://towardsdatascience.com/how-to-handle-imbalance-data-and-small-training-sets-in-ml-989f8053531d

[4] sklearn.neighbors.KNeighborsClassifier. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[5] “O que é e como funciona o algoritmo KNN?”. Disponível em: https://didatica.tech/o-que-e-e-como-funciona-o-algoritmo-knn/

[6] “Underfitting e Overfitting”. Disponível em: https://didatica.tech/underfitting-e-overfitting/

[7] “Como funciona o algoritmo Árvore de Decisão”. Disponível em: https://didatica.tech/como-funciona-o-algoritmo-arvore-de-decisao/

[8] “O que é e como funciona o algoritmo RandomForest”. Disponível em: https://didatica.tech/o-que-e-e-como-funciona-o-algoritmo-randomforest/

“Teorema de Bayes: saiba o que é e aprenda a utilizar“. Disponível em: https://www.voitto.com.br/blog/artigo/teorema-de-bayes