Análise de Desempenho de Diferentes Algoritmos Supervisionados de Machine Learning

Kevin Lourenço Tomé - 135065
Universidade Federal de São Paulo
UNIFESP - Campus SJC
São José dos Campos, São Paulo, Brasil
k.tome@unifesp.br

***Abstract*—This project aims to analyze the performance of different machine learning supervised algorithms submitted to a dataset with medicinal data to predict health accidents as strokes. (*Abstract*)**

Keywords—algoritmo, análise, supervisionado, desempenho, dataset, AVC, aprendizado de máquina

I. Introdução

A fim de analisar o desempenho de diferentes algoritmos, o dataset escolhido possui uma gama de atribuições e classificações extremamente úteis para que os algoritmos sejam treinados e testados. O healthcare_dataset_stroke_data é um dataset que contem dados médicos de diversos pacientes e, analisando todos atributos e parâmetros observados, o código vai prever se um paciente tem alguma probabilidade de desenvolver um AVC (Acidente Vascular Cerebral - "Cerebrovascular Accident").

O AVC será analisado devido a sua grande e, justificável, preocupação. De acordo com a Organização Mundial de AVC (World Stroke Organization), estatisticamente mais de 13 milhões de pessoas têm e terão um AVC a cada ano e 5,5 milhões dessas pessoas morrerão como resultado. O impacto do derrame pode ser de curto e longo prazo, dependendo de qual parte do cérebro será afetada e também da maneira que será tratado. Os sobreviventes de AVCs podem demonstrar deficiências variadas, incluindo problemas na fala e de mobilidade, bem como na maneira que sentem e pensam. A velocidade dos sintomas são percebidos e se inicia o tratamento pode ser a diferença que salva vidas e melhora e facilita os tratamentos de recuperação. [1]

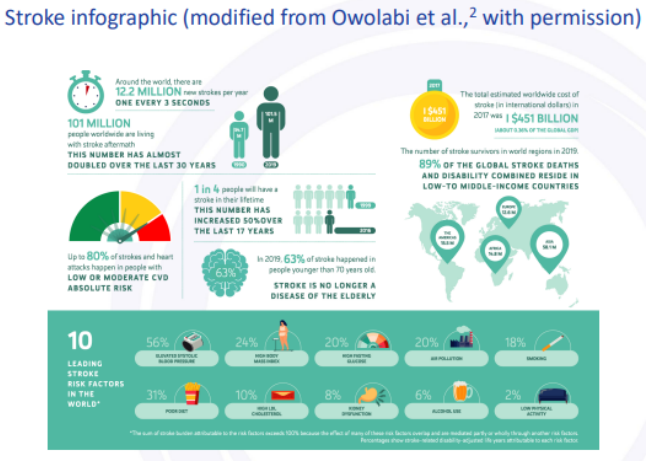

Fig 1. Infográfico relacionado a derrames no mundo. [2]

A partir da importância da rápida análise de possíveis derrames e do conjunto de dados apresentados no dataset, o objetivo é implementar três algoritmos supervisionados de machine learning e, através do tempo de execução e precisão de acertos, dizer qual algoritmo tem o melhor proveito em prever futuros AVCs.

II. Metodologia

A. Dataset

O dataset healthcare_dataset_stroke_data fornecido pelo Kaggle, com o título stroke prediction dataset [2], possui mais de 5 mil registros com as seguintes colunas :

- Identificação(id): Numérico
- Gênero(gender): Categórico.
  - Masculino(male)
  - Feminino(female)
  - Outro(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)
  - Sim(yes)
- 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.
- IMC(bmi): Numérico.
- Status de fumante(smoking_status): Categórico
  - Costumava fumar(formely_smoked)
  - Nunca fumou(never_smoked)
  - Fuma(smokes)
  - Inválido p/ paciênte(unknown)
- Status de AVC(stroke): Binário
  - 0 para não
  - 1 para sim

B. Implementação: Algoritmos

Os algoritmos escolhidos são os supervisionados de machine learning que envolvem o aprendizado de uma lógica definida (com ou sem uma heurística) a partir de exemplos de um banco de dados, ou seja, o algoritmo "aprende" a partir dos dados apresentados, e para conferir, armazenamos alguns dos dados para testes após o treinamento. Os algoritmos utilizados neste projeto são:

a) K-Nearest Neighbors:

O algoritmo KNN assume que objetos similares existem com uma certa proximidade. Em outras palavras, objetos similares estão próximos uns dos outros. [3]

Na prática, ao avaliar a similaridade dos dados, estamos medindo as distâncias entre eles, quanto menor a distância, mais similaridades.

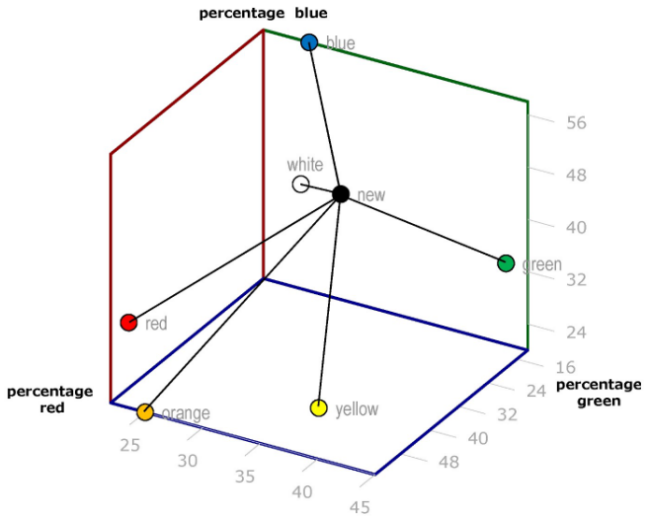

Fig 2. Novo objeto e sua distância no algoritmo KNN

A distância pode ser medida de várias formas como o método Euclidiano, Manhattan, Minkowski, dentre outros. O algoritmo implementado utiliza o método KNeighborsClassifier() da biblioteca SKLearn, e por padrão o método utilizado é o Minkowski, representada por: soma(|x - y|^p)^(1/p), onde o valor de p é 2. [5]

A vantagem é que é um algoritmo simples e fácil de implementar, não é necessário construir modelos complexos ou adicionar e modificar pontos extras no código por ser um código muito versátil podendo ser usado para classificação, regressão e até procura. Já a desvantagem é que o algoritmo fica consideravelmente mais lento com o aumento do número de dados e a complexidade do dataset.

Para um exemplo, considere que a base de dados possuem registros nas cores azul, verde e vermelho, em um gráfico KNN sob esse dataset, o resultado seria esse:

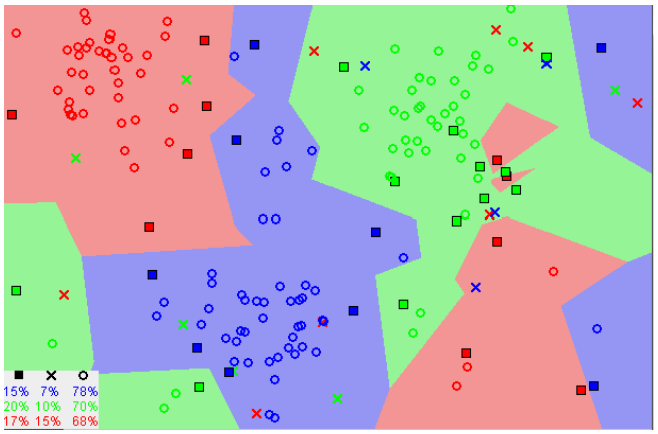

Fig 3. Imagem que mostra quão próximo objetos semelhantes ficam um do outro

A implementação deste código segue as seguintes etapas:

- Carregar e preparar o dataset;
- Iniciar K para o número definido de vizinhos;
- Agora, para cada dado
  - Calcular a distância do dado consultado com o dado atual;
  - Adicionar à distância e o index do dado em uma coleção;
- Ordenar a coleção de forma crescente (menor para maior);
- Escolher as primeiras K entradas da coleção já ordenada;
- Armazenar os dados das K entradas.

b) Naive Bayes:

O algoritmo Naive Bayes funciona como classificador e baseia-se na probabilidade de cada evento ocorrer, desconsiderando a correlação entre features. Por ter uma parte matemática relativamente simples, possui um bom desempenho e precisa de poucas observações para ter uma boa precisão.

Esse algoritmo utiliza o Teorema de Bayes com a hipótese de independência condicional, ou seja, a probabilidade de um evento ocorrer dado que um outro evento já ocorreu. [6]

Esse cálculo é representado pela seguinte fórmula:

$$P(c \mid X) = \frac{P(X \mid c)P(c)}{P(X)}$$

Fig 4. Fórmula do Teorema de Bayes

Onde:

- P(c | X): probabilidade da classe c dado o vetor X
- P(X | c): probabilidade do vetor X dado a classe c
- P(c): probabilidade a priori da classe c
- P(X): probabilidade a priori do vetor X

Para dados numéricos, como é o caso do dataset trabalhado, foi utilizado o Gaussian Naive Bayes, onde o algoritmo assume uma distribuição Gaussiana dos dados.

$$f(x) = \frac{e^{\left[-0{,}5 \cdot \left(\frac{x-\mu}{\sigma}\right)^2\right]}}{\sigma\sqrt{2\pi}}$$

Fig 5. Função de Gauss

c) Random Forest:

O algoritmo Random Forest usa a saída de uma árvore de múltipla decisão, criada de modo randômico, para chegar em um resultado único [8] onde cada árvore é utilizada na definição da classe, ou seja, utiliza o algoritmo de árvore de decisão já visto nos assuntos da aula.

Para um exemplo rápido, imagine que nosso conjunto de dados consiste nos números 0 e 1. Temos dois 1s e cinco 0s (1s e 0s são nossas classes) e desejamos separar as classes usando seus recursos. As características são de cor (vermelho e azul) e se a observação está sublinhada ou não. E assim o código as separa, fazendo "perguntas" em relação às suas diferenças e, assim, cria a árvore de decisão. [9]

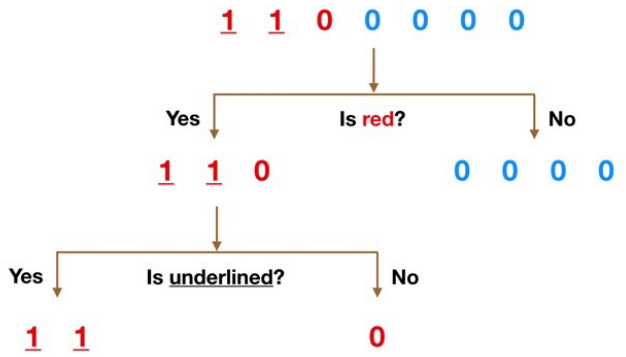

Fig 6. Exemplo simples de uma árvore de decisão

Para obter um resultado único e pelo método de misturar algumas estruturas, chega ao problema de aumentar o custo computacional, porém torna os resultados mais precisos. O algoritmo implementado utiliza o 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.

C. Implementação: Preparação do dataset

No código feito, a primeira etapa executada é o tratamento de dados, nela, prepara-se o conjunto de informações do dataset para que fiquem coerentes com os algoritmos.

Com passar da explicação, serão mostradas imagens do código que estão sendo explicadas, e nelas, serão vistos números que utilizei para organizar melhor as etapas do código e sua explicação que será o próximo tópico a ser dito.

O conjunto de dados é importado para um DataFrame da biblioteca Pandas, então, como tratamento padrão da biblioteca, são removidos as linhas que contém colunas vazias através do método "dropna".

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

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

Fig 7. Importação do dataset e a remoção dos valores nulos

Próximo passo, transforma-se os dados categóricos e numéricos através da função LabelEncoder da biblioteca sklearn e remove-se a coluna ID, já que mantê-la traria erros ao analisar o dataset já que não representa um atributo e nem dado a ser analisado

```
#(6). 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'])

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

Fig 8. Transformação dos dados categóricos e remoção da coluna Id

Já em relação a quantidade de dados da classe de AVC, temos um registro elevado de 0s (não tem AVC) em relação aos 1s (tem AVC), então também foi preciso um balanceamento dos dados. Para isso, foi utilizado o método upsampling [10], onde aumentamos os registros da classe de menor quantidade através do método resample da biblioteca sklearn. Além disso, os registros foram embaralhados e a coluna que identifica a classe foi eliminada.

```
#(8). Equilibra os dados, com o método Upsampling
no_stroke = df[df['stroke'] == ZERO]
stroke = df[df['stroke'] == ONE]
upsampling = resample(stroke, replace=True, n_samples=no_stroke.shape[ZERO])
df = pd.concat([no_stroke, upsampling])
df = shuffle(df)
```

Fig 9. Equilíbrio dos dados e remoção da coluna que identifica as classes

Por fim, separa-se 70% dos registros para treinar o algoritmo e 30% para testá-lo em relação a velocidade de compilação e precisão dos resultados, e assim, são executados os algoritmos.

```
#(9). Separa 70% dos dados para treino e 30% para teste
x_treino, x_teste, y_treino, y_teste = train_test_split(x, y, train_size=0.7, random_state=101)

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

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

Fig 10. Divisão dos dados de treino e teste e execução dos algoritmos

D. Implementação: Lógica e Código

Como dito anteriormente, o código foi numerado para facilitar reconhecer todas suas etapas e sua explicação, portanto aqui estão suas etapas enumeradas:

0. Variaveis Globais;
1. Construtor da Classe;
2. Executa os algoritmos;
    - 2-1. Verifica qual é o algoritmo
    - 2-2. Inicia as variáveis de análise
    - 2-3. Executa o algoritmo 100 vezes
        - 2-3-1. Recebe o tempo antes da execução
        - 2-3-2. Executa o algoritmo

2-3-3. Soma as precisões e os tempos de execução

2-4. Imprime os dados

2-5. Guarda os dados obtidos

3. Plota os gráficos

3-1. Gráfico em barra: algoritmo x precisão

3-2. Gráfico em barra: algoritmo x tempo

4. Importação do dataset

5. Remove valores nulos

6. Transforma os dados categóricos em numéricos

7. Remove coluna Id

8. Equilibra os dados, com o método upsampling

9. Separa os dados em 70% treino e 30% teste

10. Cria classe

11. Executa o algoritmo

Do ponto (4) até (11) já mostramos no segmento anterior, agora vou apresentar as etapas (0) até (3).

Segue as bibliotecas importadas necessárias para manter e rodas os algoritmos. Predominantemente, tempos as bibliotecas do sklearn que nos auxiliam nos algoritmos, tratamento, equilíbrio e análise dos dados importados do dataset e sua saída. Junto tempos as variáveis globais que usamos para facilitar a testagem das repetições e loops para aprendizagem e testagem dos algoritmos supervisionados.

```
import pandas as pd
import time
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle, resample
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier

#(0). Variáveis globais
global SIZE = 100
global ZERO = 0
global ONE = 1
```

Fig 11. Bibliotecas e variáveis globais

A seguir, o construtor _init_ da classe, que armazena todos os dados necessários para a análise final de cada algoritmo executado.

```
#(1). Construtor da classe
def __init__(self, x_treino, x_teste, y_treino, y_teste):
    self.x_treino = x_treino
    self.x_teste = x_teste
    self.y_treino = y_treino
    self.y_teste = y_teste
    self.algoritmos = []
    self.precisao = []
    self.tempo_execucao = []
```

Fig 12. Construtor da classe

Agora, o código executa os algoritmos selecionados através da chave que é as iniciais dos nomes dos algoritmos já mostrados. KNN(K-Nearest Neighbors), NB(Naive Bayes) e RF(Random Forest).

Após a execução, já iniciamos as variáveis de análise que vão armazenar as precisões e os tempos de execução necessários para a análise.

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

    #(2-1). Verifica qual é o algoritmo
    if algoritmo == 'KNN':
        nome_algoritmo = 'K-Nearest Neighbours'
        execucao = KNeighborsClassifier(n_neighbors=5)
    if algoritmo == 'NB':
        nome_algoritmo = 'Naive Bayes'
        execucao = GaussianNB()
    if algoritmo == 'RF':
        nome_algoritmo = 'Random Forest'
        execucao = RandomForestClassifier()

    #(2-2). Inicia variáveis
    precisao = ZERO
    tempo_execucao = ZERO
```

Fig 13. Executa o algoritmo e inicia variáveis de análise

Após as variáveis de precisão e de tempo de execução estiverem prontas, começa-se o loop que executa 100 vezes cada algoritmo armazenando as precisões e os tempos necessários para a análise.

```
#(2-3). Executa o algoritmo 100 vezes
for i in range(SIZE):
    #(2-3-1). Recebe o tempo antes da execução
    tempo_inicio = time.time()

    #(2-3-2). Executa o algoritmo
    execucao.fit(self.x_treino, self.y_treino)
    predict = execucao.predict(self.x_teste)

    #(2-3-3). Soma as precisões e os tempos de execução
    precisao += accuracy_score(self.y_teste, predict) * SIZE
    tempo_execucao += time.time() - tempo_inicio
```

Fig 14. Loop de execução e preenchimento das variáveis de análise

Após termos os dados de análise necessários, enviamos para a saída do código, além de armazenarmos para o plot dos gráficos finais.

```
#(2-4). Printa os dados
print(nome_algoritmo)
#>>Tempo médio das execuções
print("Tempo de execução: " + str(format(tempo_execucao / SIZE, ".3f")) + "s")
#>>Média das precisões
print("Precisão dos resultados: " + str(format(precisao / SIZE, ".3f")) + "%")
print(" ")

#(2-5). Guarda os dados obtidos
self.tempo_execucao.append(float(format(tempo_execucao / SIZE, ".3f")))
self.precisao.append(float(format(precisao / SIZE, ".3f")))
self.algoritmos.append(algoritmo)
```

Fig 15. Impressão dos dados e armazenamento para o plot dos gráficos

E finalmente, após armazenar todos os resultados de maneira organizada a fim de tê-las prontas para serem imprimidas, plotamos dois gráficos.

O primeiro gráfico apresenta o algoritmo em relação a sua precisão e o segundo gráfico apresenta o algoritmo em relação ao seu tempo de execução.

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

    #(3-1). Gráfico em barra: algoritmo x precisão
    fig = plt.figure()
    ax = fig.add_axes([ZERO, ZERO, ONE, ONE])
    ax.bar(self.algoritmos, self.precisao)
    ax.set_title('Precisão dos algoritmos')
    ax.set_xlabel('Algoritmos')
    ax.set_ylabel('Precisão (%)')
    plt.show()

    #(3-2). Gráfico em barra: algoritmo x tempo de execução
    fig = plt.figure()
    ax = fig.add_axes([ZERO, ZERO, ONE, ONE])
    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 16. Montagem e saída dos gráficos

III. Análise dos resultados

Agora para analisar os resultados adquiridos pela saída do algoritmo, precisamos listar qual foi o ambiente usado a fim de tornar os resultados coerentes também com a capacidade computacional do projeto junto ao critério de desenvolvimento, e assim, dissertar sobre o resultado.

A. Ambiente computacional

O ambiente computacional consiste em um notebook pessoal Dell Inspiron 7560,com as seguintes configurações:

a) Processador 7ª geração Intel Core i7 - 7500
b) Placa de vídeo NVIDIA GeForce 940MX 4GB
c) Disco rígido de 1TB com 128GB de SSD
d) Memória de 16GB
e) Sistema operacional de 64bits, procesador baseado em x64
f) Edição Windows 10 Home Single Language

B. Critério de desenvolvimento

Através das variáveis que armazenam o tempo de execução e precisão dos resultados, executar os algoritmos supervisionados de machine learning n vezes (n = 100) e dissertar sobre os resultados.

C. Resultados

Segue a saída do algoritmo:

```
K-Nearest Neighbors
Tempo de execucao: 0.072s
Precisão dos resultados: 93.298%

Naive Bayes
Tempo de execucao: 0.003s
Precisão dos resultados: 76.986%

Random Forest
Tempo de execucao: 0.435s
Precisão dos resultados: 99.624%
```

Fig 17. Saída do algoritmo

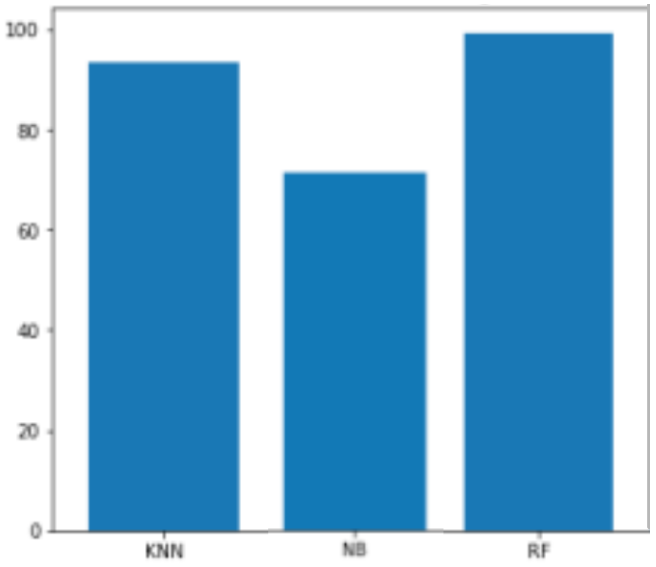

Fig 18. Gráfico em barras. Precisão por algoritmo

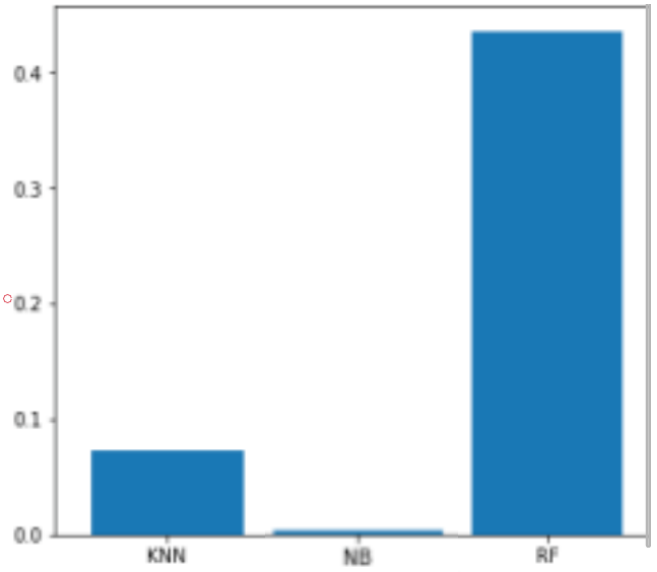

Fig 19. Gráfico em barra. Tempo por algoritmo.

Analisando a porcentagem de precisão de resultado, todos os algoritmos estudados tiveram uma boa precisão (acima de 75%). O algoritmo Random Forest (RF) teve a maior porcentagem de precisão com 99.624% e o algoritmo Naive Bayes (NB) teve a menor porcentagem de precisão com 76.986%.

Analisando o tempo de execução, a situação se inverteu. O algoritmo Naive Bayes (NB) teve o menor tempo de execução com 0.003s, e o algoritmo Random Forest (RF) teve o maior tempo de execução, demorando 0.435s para executar, 0.360s mais lento que o KNN e 0.432s mais lento que Naive Bayes.

Analisando os gráficos em barra, considerando apenas os valores de Precisão de Resultados x Tempo de Execução, o algoritmo K-Nearest Neighbors (KNN) teve os melhores resultados para ser uma ótima escolha. Semelhante ao Random Forest, com uma precisão de 93.298%, porém muito mais rápido, levando apenas 0.072s para executar.

Conclusão

O projeto tinha como proposta de analisar o desempenho de diferentes algoritmos supervisionado de machine learning: K-Nearest Neighbors, Naive Bayes e Random Forest, aplicados em um dataset focado a área da saúde que prevê a possibilidade de um paciente ter um AVC ou não, a partir dos atributos.

Depois, preparamos os dados, transformamos os dados que são categóricos e executamos 100 vezes os algoritmos.Colhido os resultados, analisamos os valores de forma numérica e representação gráfica. As conclusões foram coesas 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 K-Nearest Neighbors foi o mais completo, com uma precisão de apenas 6.326% abaixo da Random Forest, porém com um tempo de execução bem inferior.

References.

[1] Website da Organização Mundial de Derrame em: https://www.world-stroke.org

[2] Ficha Técnica Mundial de Derrame em: https://www.world-stroke.org/assets/downloads/WSO_Global_Stroke_Fact_Sheet.pdf

[3] Noções Básicas de Aprendizado de Máquina com o Algoritmo K-Nearest Neightbors em: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[4] KNN(K-Nearest Neighbors) #1 em: https://medium.com/brasil-ai/knn-k-nearest-neighbors-1-e140c82e9c4e

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

[6] Modelos de Predição | Naive Bayes em: https://medium.com/turing-talks/turing-talks-16-modelo-de-predição-naive-bayes-6a3e744e7986

[7] Aprendizado de Máquina pelo professor Alessandro L. Koerich em: http://www.eletrica.ufpr.br/ufpr2/professor/36/TE808/5-NaiveBayes-AM.pdf

[8] Random Forest IBM em: https://www.ibm.com/cloud/learn/random-forest

[9] Entendendo a Random Forest em: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[10] Como Lidar com Dados de Desequilíbrio e Pequenos Conjuntos de Treinamento no ML em: https://towardsdatascience.com/how-to-handle-imbalance-data-and-small-training-sets-in-ml-989f8053531d