# Lista 03

Nome: Júlio César Gonzaga Ferreira Silva

1. Utilizando o algoritmo de Naive Bayes, qual a probabilidade de Jogar ou não Jogar, respectivamente para o seguinte registro:

Aparência = Chuva

Temperatura = Fria

Umidade = Normal

Ventando = Sim

A resposta é sim, deve-se jogar seguindo a probabilidade.

CODIGO FEITO:

import pandas as pd

from sklearn.naive\_bayes import GaussianNB

base\_tempo\_para\_jogar = pd.read\_csv("C:\\Users\\ferre\\Documents\\GitHub\\Inteligência Artificial\\Lista 2\\weather.nominal.csv")

base\_tempo\_para\_jogar

x\_risco\_tempo = base\_tempo\_para\_jogar.iloc[:, 0:4].values

x\_risco\_tempo

y\_jogar = base\_tempo\_para\_jogar.iloc[:,4].values

y\_jogar

from sklearn.preprocessing import LabelEncoder

label\_encoder\_outlook = LabelEncoder()

label\_encoder\_temperature = LabelEncoder()

label\_encoder\_humidity = LabelEncoder()

label\_encoder\_windy = LabelEncoder()

x\_risco\_tempo[:,0] = label\_encoder\_outlook.fit\_transform(x\_risco\_tempo[:,0])

x\_risco\_tempo[:,1] = label\_encoder\_temperature.fit\_transform(x\_risco\_tempo[:,1])

x\_risco\_tempo[:,2] = label\_encoder\_humidity.fit\_transform(x\_risco\_tempo[:,2])

x\_risco\_tempo[:,3] = label\_encoder\_windy.fit\_transform(x\_risco\_tempo[:,3])

x\_risco\_tempo

import pickle

with open('risco\_tempo.pkl', 'wb') as f:

    pickle.dump([x\_risco\_tempo, y\_jogar], f)

naive\_tempo = GaussianNB()

naive\_tempo.fit(x\_risco\_tempo, y\_jogar)

#Aparência = Chuva, Temperatura = Fria, Umidade = Normal, Ventando = Sim

#outlook = rainy, temperature = cool, humidity = Normal, windy = true

#outlook = 1, temperature = 0, humidity = 1, windy = 1

# Fazendo uma previsão com os dados fornecidos: outlook = rainy, temperature = cool, humidity = normal, windy = True

previsao = naive\_tempo.predict([[1, 0, 1, 1]])

1. Codigo Feito:

#Carregar bibliotecas

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

Carregamento das Bases Usadas ( treino, teste, resultados )

# Carregar bases de dados

dbTreino = pd.read\_csv('/content/sample\_data/train.csv')

dbTeste = pd.read\_csv('/content/sample\_data/test.csv')

dbResultados = pd.read\_csv('/content/sample\_data/gender\_submission.csv')

Tratamento dos Dados

# Verificar dados ausentes

dbTreino.isnull().sum()

# Tratando pela média

dbTreino['Age'].fillna(dbTreino['Age'].mean(), inplace=True)

dbTeste['Age'].fillna(dbTeste['Age'].mean(), inplace=True)

dbTreino['Fare'].fillna(dbTreino['Fare'].mean(), inplace=True)

dbTeste['Fare'].fillna(dbTeste['Fare'].mean(), inplace=True)

# Binarizar Atributo 'Sex' usando onehot

dbTreino = pd.get\_dummies(dbTreino, columns=['Sex'])

dbTeste = pd.get\_dummies(dbTeste, columns=['Sex'])

# Juntar atributos 'SibSp'e 'Parch' para 'FamilyOnBoard'

dbTreino['FamilyOnBoard'] = dbTreino['SibSp'] + dbTreino['Parch'] + 1

dbTeste['FamilyOnBoard'] = dbTeste['SibSp'] + dbTeste['Parch'] + 1

# Excluir atributo das bases de dados

dbTreino.drop(['SibSp', 'Parch'], axis=1, inplace=True)

dbTeste.drop(['SibSp', 'Parch'], axis=1, inplace=True)

Separação das Features

# Atributos para classificação

atributos = ['Pclass', 'Sex\_female', 'Sex\_male', 'Age', 'FamilyOnBoard', 'Fare']

# Atributo alvo (Survived)

classificacao = ['Survived']

# Definição de variáveis

X\_train = dbTreino[atributos]

y\_train = dbTreino[classificacao].values.ravel()

X\_test = dbTeste[atributos]

y\_test = dbResultados[classificacao].values.ravel()

Naive Bayes

# Definir os hiperparâmetros para busca

param\_dist = {

'var\_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4]

}

# Configurar o RandomizedSearchCV

random\_search = RandomizedSearchCV(

estimator=GaussianNB(),

param\_distributions=param\_dist,

cv=10,

verbose=1,

n\_jobs=5,

random\_state=42,

n\_iter=6

)

# Treinar o modelo com RandomizedSearchCV

random\_search.fit(X\_train, y\_train)

# Melhor modelo encontrado

best\_model = random\_search.best\_estimator\_

# Fazer previsões no conjunto de teste

y\_test\_pred = best\_model.predict(X\_test)

# Calcular e exibir as métricas de desempenho no conjunto de teste

class\_report\_test = classification\_report(y\_test, y\_test\_pred)

print("Relatório de classificação:\n")

print(class\_report\_test)

# Calcular a matriz de confusão

conf\_matrix = confusion\_matrix(y\_test, y\_test\_pred)

# Plotar a matriz de confusão

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=['Não Sobreviveu', 'Sobreviveu'],

yticklabels=['Não Sobreviveu', 'Sobreviveu'])

plt.title('Matriz de Confusão')

plt.show()