Lista 03

Nome: Júlio César Gonzaga Ferreira Silva

01) 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]])
```

02) 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()
```