

**2)** def naive\_bayes\_probability(aparencia, temperatura, umidade, ventando):

# Probabilidades a priori

prob\_jogar = 0.8229 # P(jogar)

prob\_nao\_jogar = 0.1771 # P(não jogar)

# Probabilidades condicionais para jogar

prob\_aparencia\_jogar = {

'sol': 3/9,

'nublado': 4/9,

'chuva': 3/9

}

prob\_temperatura\_jogar = {

'quente': 2/9,

'agradavel': 4/9,

'fria': 3/9

}

prob\_umidade\_jogar = {

'alta': 3/9,

'normal': 6/9

}

prob\_ventando\_jogar = {

'sim': 3/9,

'não': 6/9

}

# Probabilidades condicionais para não jogar

prob\_aparencia\_nao\_jogar = {

'sol': 2/5,

'nublado': 0/5,

'chuva': 2/5

}

prob\_temperatura\_nao\_jogar = {

'quente': 2/5,

'agradavel': 2/5,

'fria': 1/5

}

prob\_umidade\_nao\_jogar = {

'alta': 4/5,

'normal': 1/5

}

prob\_ventando\_nao\_jogar = {

'sim': 3/5,

'não': 2/5

}

probabilidade\_jogar = (

prob\_aparencia\_jogar[aparencia]

\* prob\_temperatura\_jogar[temperatura]

\* prob\_umidade\_jogar[umidade]

\* prob\_ventando\_jogar[ventando]

\* prob\_jogar

)

probabilidade\_nao\_jogar = (

prob\_aparencia\_nao\_jogar[aparencia]

\* prob\_temperatura\_nao\_jogar[temperatura]

\* prob\_umidade\_nao\_jogar[umidade]

\* prob\_ventando\_nao\_jogar[ventando]

\* prob\_nao\_jogar

)

# Normalizando as probabilidades para obter a probabilidade posterior

normalizacao = probabilidade\_jogar + probabilidade\_nao\_jogar

probabilidade\_jogar\_posterior = probabilidade\_jogar / normalizacao

probabilidade\_nao\_jogar\_posterior = probabilidade\_nao\_jogar / normalizacao

return probabilidade\_jogar\_posterior, probabilidade\_nao\_jogar\_posterior

# Exemplo de uso:

aparencia = 'sol'

temperatura = 'quente'

umidade = 'alta'

ventando = 'sim'

prob\_jogar, prob\_nao\_jogar = naive\_bayes\_probability(aparencia, temperatura, umidade, ventando)

print(f'Probabilidade de jogar: {prob\_jogar:.4f}')

print(f'Probabilidade de não jogar: {prob\_nao\_jogar:.4f}')

**3) Implementar Random Forest:**

rf\_model = RandomForestClassifier(random\_state=42)

# Treinar o modelo

rf\_model.fit(X\_train, y\_train)

### **Implementar Naive Bayes e Árvore de Decisão:**

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

### **Comparar os Modelos:**

# Prever os resultados nos dados de teste

rf\_pred = rf\_model.predict(X\_test)

nb\_pred = nb\_model.predict(X\_test)

dt\_pred = dt\_model.predict(X\_test)

# Calcular acurácia dos modelos

rf\_accuracy = accuracy\_score(y\_test, rf\_pred)

nb\_accuracy = accuracy\_score(y\_test, nb\_pred)

dt\_accuracy = accuracy\_score(y\_test, dt\_pred)

print(f'Acurácia do Random Forest: {rf\_accuracy:.4f}')

print(f'Acurácia do Naive Bayes: {nb\_accuracy:.4f}')

print(f'Acurácia da Árvore de Decisão: {dt\_accuracy:.4f}')

### **Ajustar Hiperparâmetros com RandomSearch:**

# Definir os hiperparâmetros a serem testados

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

rf\_random = RandomizedSearchCV(estimator=rf\_model, param\_distributions=param\_grid,

n\_iter=100, cv=3, verbose=2, random\_state=42, n\_jobs=-1)

rf\_random.fit(X\_train, y\_train)

print("Melhores hiperparâmetros encontrados:")

print(rf\_random.best\_params\_)

best\_rf\_model = rf\_random.best\_estimator\_

# Avaliar o modelo ajustado nos dados de teste

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

best\_rf\_accuracy = accuracy\_score(y\_test, best\_rf\_pred)

print(f'Acurácia do Random Forest com melhores hiperparâmetros: {best\_rf\_accuracy:.4f}')

**4)**

Utilizando-se o algoritmo Apriori, um suporte mínimo aceitável de 0.3 e confiança de 0.8, o número de ItensSets 1, 2, 3 e de regras a partir desta base de dados são: 2.

**5)** Após rodar o código, podemos deduzir que: ele calculará as probabilidades de jogar e não jogar com base nas condições específicas de aparencia='sol', temperatura='quente', umidade='alta' e ventando='sim'.

**6)**

data = {

'Transação': [1, 2, 3, 4, 5],

'Itens': [['A', 'B', 'D'],

['B', 'C', 'E'],

['A', 'B', 'C', 'E'],

['A', 'E'],

['B', 'D']]

}

df = pd.DataFrame(data)

df\_encoded = df['Itens'].apply(lambda x: pd.Series([1] \* len(x), index=x)).fillna(0)

frequent\_itemsets = apriori(df\_encoded, min\_support=0.4, use\_colnames=True)

print("Conjuntos frequentes com seus suportes:")

print(frequent\_itemsets)

**7)**

def naive\_bayes\_association\_rules(aparencia, temperatura, umidade, ventando):

prob\_jogar = 0.8229 # P(jogar)

prob\_nao\_jogar = 0.1771 # P(não jogar)

prob\_aparencia\_jogar = {

'sol': 3/9,

'nublado': 4/9,

'chuva': 3/9

}

prob\_temperatura\_jogar = {

'quente': 2/9,

'agradavel': 4/9,

'fria': 3/9

}

prob\_umidade\_jogar = {

'alta': 3/9,

'normal': 6/9

}

prob\_ventando\_jogar = {

'sim': 3/9,

'não': 6/9

}

prob\_aparencia\_nao\_jogar = {

'sol': 2/5,

'nublado': 0/5,

'chuva': 2/5

}

prob\_temperatura\_nao\_jogar = {

'quente': 2/5,

'agradavel': 2/5,

'fria': 1/5

}

prob\_umidade\_nao\_jogar = {

'alta': 4/5,

'normal': 1/5

}

prob\_ventando\_nao\_jogar = {

'sim': 3/5,

'não': 2/5

}

prob\_condicional\_jogar = (

prob\_aparencia\_jogar[aparencia]

\* prob\_temperatura\_jogar[temperatura]

\* prob\_umidade\_jogar[umidade]

\* prob\_ventando\_jogar[ventando]

)

prob\_condicional\_nao\_jogar = (

prob\_aparencia\_nao\_jogar[aparencia]

\* prob\_temperatura\_nao\_jogar[temperatura]

\* prob\_umidade\_nao\_jogar[umidade]

\* prob\_ventando\_nao\_jogar[ventando]

)

prob\_X = (prob\_conditional\_jogar \* prob\_jogar) + (prob\_conditional\_nao\_jogar \* prob\_nao\_jogar)

prob\_associations = {}

products = {

'álcool': prob\_aparencia\_nao\_jogar['sol'] \* prob\_temperatura\_nao\_jogar['quente'] \* prob\_umidade\_nao\_jogar['alta'] \* prob\_ventando\_nao\_jogar['sim'],

'detergente': prob\_aparencia\_nao\_jogar['sol'] \* prob\_temperatura\_nao\_jogar['agradavel'] \* prob\_umidade\_nao\_jogar['normal'] \* prob\_ventando\_nao\_jogar['não'],

'arroz': prob\_aparencia\_nao\_jogar['nublado'] \* prob\_temperatura\_nao\_jogar['agradavel'] \* prob\_umidade\_nao\_jogar['normal'] \* prob\_ventando\_nao\_jogar['sim']

}

for product, prob in products.items():

prob\_associations[product] = prob / prob\_X

return prob\_associations

# Exemplo de uso:

aparencia = 'sol'

temperatura = 'quente'

umidade = 'alta'

ventando = 'sim'

associations = naive\_bayes\_association\_rules(aparencia, temperatura, umidade, ventando)

# Imprimir as regras de associação

for product, prob in associations.items():

print(f"Quem não leva {product}, leva: {prob:.4f}")

**8) Exemplo das regras:**

# Exemplo de dados:

data = {'ID': [1, 2, 3, 4, 5],

'Items': [['A', 'B', 'D'],

['B', 'C'],

['A', 'C', 'D'],

['A', 'B'],

['B', 'D']]}

df = pd.DataFrame(data)

# Transformar a coluna 'Items' em one-hot encoding

df\_encoded = df['Items'].str.join('|').str.get\_dummies()

# Aplicar o algoritmo Apriori para encontrar itemsets frequentes

frequent\_itemsets = apriori(df\_encoded, min\_support=0.2, use\_colnames=True)

# Gerar regras de associação

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

print(rules)