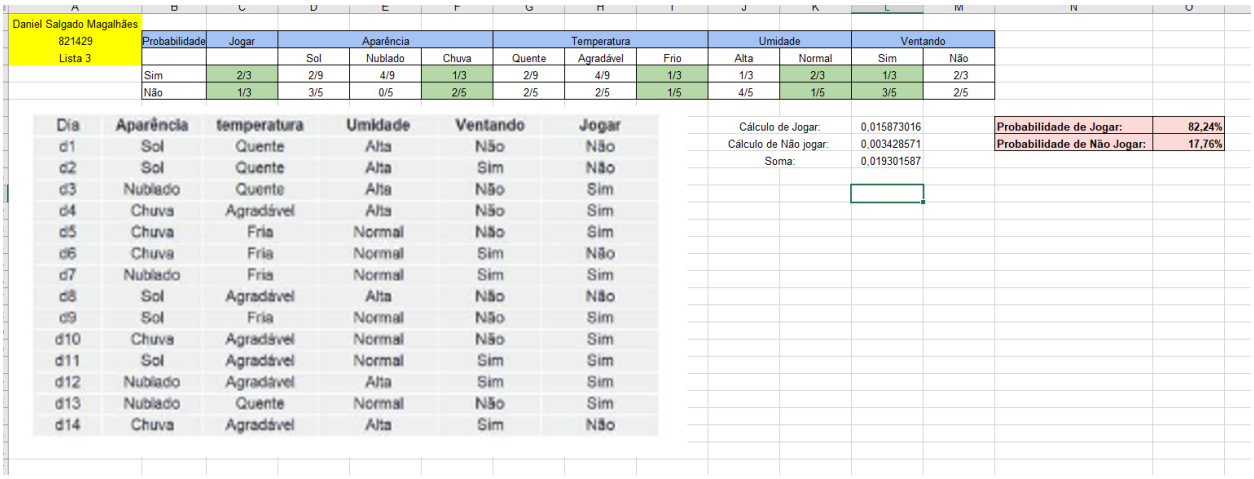
Lista 3 IA

Daniel Salgado Magalhães - 821429

Questão 1

Os valores em verde são os valores utilizados para o cálculo, tanto na linha de jogar quanto na linha de não jogar. Os valores em rosa são as porcentagens obtidas após a utilização do algortimo de Naive Bayes, utilizando da multiplicação de cada fator dividido pela soma dos dois cálculos.



Daniel Salgado Magalhães
821429
Lista 3

Probabilidade	Jogar	Aparência			Temperatura			Umidade		Ventando	
		Sol	Nublado	Chuva	Quente	Agradável	Frio	Alta	Normal	Sim	Não
Sim	2/3	2/9	4/9	1/3	2/9	4/9	1/3	1/3	2/3	1/3	2/3
Não	1/3	3/5	0/5	2/5	2/5	2/5	1/5	4/5	1/5	3/5	2/5

Dia	Aparência	temperatura	Umidade	Ventando	Jogar
d1	Sol	Quente	Alta	Não	Não
d2	Sol	Quente	Alta	Sim	Não
d3	Nublado	Quente	Alta	Não	Sim
d4	Chuva	Agradável	Alta	Não	Sim
d5	Chuva	Fria	Normal	Não	Sim
d6	Chuva	Fria	Normal	Sim	Não
d7	Nublado	Fria	Normal	Sim	Sim
d8	Sol	Agradável	Alta	Não	Não
d9	Sol	Fria	Normal	Não	Sim
d10	Chuva	Agradável	Normal	Não	Sim
d11	Sol	Agradável	Normal	Sim	Sim
d12	Nublado	Agradável	Alta	Sim	Sim
d13	Nublado	Quente	Normal	Não	Sim
d14	Chuva	Agradável	Alta	Sim	Não

Cálculo de Jogar:	0,015873016
Cálculo de Não jogar:	0,003428571
Soma:	0,019301587

Probabilidade de Jogar:	**82,24%**
Probabilidade de Não Jogar:	**17,76%**

Questão 2

Código usado no google colab para tentar rodar os dados no zip TITANIC

Implemente os métodos de Random Forest, Naive e Árvore de decisão para classificar o passageiro em Survived e Not Survived

```
# Importando as bibliotecas necessárias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
```

```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Lendo o arquivo CSV
data = pd.read_csv('/content/sample_data/test.csv')

# Separando as características (X) e o rótulo (y)
X = data.iloc[:, :-1]  # Todas as colunas, exceto a última
y = data.iloc[:, -1]   # Última coluna

# Verificando se existem valores NaN no dataset
print("Colunas com valores ausentes e suas quantidades antes do
print(X.isna().sum())

# Identificando colunas categóricas (strings)
categorical_columns = X.select_dtypes(include=['object']).colum
numerical_columns = X.select_dtypes(exclude=['object']).columns

# Criando os pipelines para processar dados categóricos e numéri
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Imp
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse=Fal
])

numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean'))  # Imputando a m
])

# Aplicando os transformers às colunas categóricas e numéricas
preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_transformer, numerical_columns),
    ('cat', categorical_transformer, categorical_columns)
])

# Dividindo os dados em treinamento e teste
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_s

# Aplicando o preprocessor nos dados de treinamento
X_train_processed = preprocessor.fit_transform(X_train)

# Verificando se ainda há NaNs após o pré-processamento
print(f"Número de valores NaN após o pré-processamento no conju

# Aplicando o preprocessor nos dados de teste
X_test_processed = preprocessor.transform(X_test)

# Verificando se ainda há NaNs após o pré-processamento no conju
print(f"Número de valores NaN após o pré-processamento no conju

# Criando o modelo Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state

# Treinando o modelo Random Forest
rf_model.fit(X_train_processed, y_train)

# Fazendo previsões com Random Forest
y_pred_rf = rf_model.predict(X_test_processed)

# Calculando e imprimindo a acurácia do Random Forest
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Acurácia do Random Forest: {accuracy_rf * 100:.2f}%")

# Criando e treinando o modelo Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train_processed, y_train)

# Fazendo previsões com Naive Bayes
y_pred_nb = nb_model.predict(X_test_processed)

# Calculando e imprimindo a acurácia do Naive Bayes
accuracy_nb = accuracy_score(y_test, y_pred_nb)
```

```
print(f"Acurácia do Naive Bayes: {accuracy_nb * 100:.2f}%")

# Criando e treinando o modelo de Árvore de Decisão
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_processed, y_train)

# Fazendo previsões com a Árvore de Decisão
y_pred_dt = dt_model.predict(X_test_processed)

# Calculando e imprimindo a acurácia da Árvore de Decisão
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Acurácia da Árvore de Decisão: {accuracy_dt * 100:.2f}%"
```

2 - Ajuste dos hiperparâmetros e inicialização dentro do código

```
# Ajuste os hiperparâmetros, utilizando o RandomizedSearchCV
param_dist = {
    'n_estimators': np.arange(50, 300, 50)
    'max_depth': [None] + list(np.arange(5, 30, 5)),
     'min_samples_split': np.arange(2, 10), # Min de amostras pa
    'min_samples_leaf': np.arange(1, 5) # Min de amostras por fo
    'bootstrap': [True, False]
}

# Inicializa o algoritmo de Random Forest com a variável rf_mode
rf_model = RandomForestClassifier(random_state=42)
random_search = RandomizedSearchCV(rf_model, param_distributions
random_search.fit(X_train, y_train)


# Ele usa o Best Random Forest Model para colocar na variável o
best_rf_model = random_search.best_estimator_
```