UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE PÓS GRADUAÇÃO EM GESTÃO E CIÊNCIAS DE DADOS

APRENDIZADO SUPERVISIONADO

FÁBIO JOSÉ DO NASCIMENTO

CUIABÁ – MT
2024

Problema 1: Estimativa de Salário, base de dados emprego.csv

Tabela_01: Base de dados emprego.csv

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	NaN
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0

Os parâmetros que foram usados e tiveram os melhores desempenho nos métodos foram; ssc_p, hsc_p, degree_t, etest_p e mba_p, onde o target é **salary**.

Tabela_02: Resultados obtidos usando os métodos, Regressão Linear, Random Forest e RNA Regressão.

Método: Regressão Linear

Métricas de Treino	
MSE:	7150937341.05
RMSE:	84563.22
MAE:	53390.00
R2:	0.18

Métricas de Teste	
MSE:	9795753997.82
RMSE:	98973.50
MAE:	67637.43
R2:	-0.16

Real vs Predição		
Nº Resg.	Real	Predição
178	350000.00	341043.11
74	336000.00	354627.30
203	260000.00	314951.02
28	350000.00	308256.14
145	400000.00	271649.79
20	265000.00	265493.07
112	250000.00	277506.03
48	250000.00	316283.11
117	240000.00	323664.09
15	200000.00	302463.13

Método: Random Forest

Métricas de Treino	
MSE:	7765761179.50
RMSE:	88123.56
MAE:	52871.10
R2:	0.11

Métricas de Teste	
MSE:	6035617942.27
RMSE:	77689.23
MAE:	58836.39
R2:	0.023474845740221295

Real vs Predição		
Nº Resg.	Real	Predição
178	350.000,00	349.254,00
74	336.000,00	331.136,00
203	260.000,00	316.457,00
28	350.000,00	277.598,00
145	400.000,00	281.450,00
20	265.000,00	258.601,00
112	250.000,00	276.946,00
48	250.000,00	289.306,00
117	240.000,00	315.901,00
15	200.000,00	294.217,00

Método: RNA Regressão

Métricas de Treino	
MSE:	12896306528.83
RMSE:	113561.90
MAE:	68206.71
R2:	-0.3206789447123237

Métricas de Teste	
MSE:	8706670966.76
RMSE:	93309.54
MAE:	67089.04
R2:	-0.03

Real vs Predição		
Nº Resg.	Real	Predição
178	350.000,00	257.025,00
74	336.000,00	270.416,00
203	260.000,00	176.447,00
28	350.000,00	295.607,00
145	400.000,00	304.952,00
20	265.000,00	182.648,00
112	250.000,00	162.398,00
48	250.000,00	224.998,00
117	240.000,00	325.356,00
15	200.000,00	272.825,00

Com base nas métricas de teste que tem maior peso para decisão de qual método usar. O método **Random Forest** é a melhor opção no geral, considerando que possui os menores valores de MSE e RMSE, indicando uma performance mais consistente, testei com varias combinações de atributos onde melhor performance foi essa nos três métodos utilizados.

Problema 2: Identificação de faixa de preço, base de dados celulares.csv

Tabela_03: Base de dados celulares.csv

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	842	0	2.2	0	1	0	7	0.6	188	2	2	20	756	2549	9	7	19	0	0	1	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	6	905	1988	2631	17	3	7	1	1	0	2
2	563	1	0.5	1	2	1	41	0.9	145	5	6	1263	1716	2603	11	2	9	1	1	0	2
3	615	1	2.5	0	0	0	10	0.8	131	6	9	1216	1786	2769	16	8	11	1	0	0	2
4	1821	1	1.2	0	13	1	44	0.6	141	2	14	1208	1212	1411	8	2	15	1	1	0	1

Ao contrário do primeiro problema o próprio algoritmo faz as correlações e define quais parâmetros usar para melhor desempenho e acurácia, assim obtendo melhores resultados.

Tabela_04: Resultados obtidos usando os métodos, Regressão Logística, Árvore de Decisão e RNA Classificação.

Método: Regressão Logística

	Precision	Recall	F1-Score	Support
0	1.00	0.94	0.97	105
1	0.93	1.00	0.96	91
2	0.99	0.97	0.98	92
3	0.98	0.99	0.99	112
Accuracy			0.97	400
Macro AVG	0.97	0.98	0.97	400
Weighted AVG	0.98	0.97	0.98	400

Real vs Predição		
Nº Resg.	Real	Predição
1860	0	0
353	2	2
1333	1	1
905	3	3
1289	1	1
1273	1	1
938	2	2
1731	0	0
65	3	3
1323	1	1

Método: Árvore Decisão

	Precision	Recall	F1-Score	Support
0	0.93	0.88	0.90	105
1	0.76	0.86	0.80	91
2	0.77	0.75	0.76	92
3	0.90	0.87	0.88	112
Accuracy			0.84	400
Macro AVG	0.84	0.84	0.84	400
Weighted AVG	0.84	0.84	0.84	400

Real vs Predição		
Nº Resg.	Real	Predição
1860	0	0
353	2	2
1333	1	1
905	3	3
1289	1	1
1273	1	1
938	2	1
1731	0	0
65	3	2
1323	1	1

Método: RNA Classificação

	Precision	Recall	F1-Score	Support
0	0.97	0.94	0.95	151
1	0.90	0.95	0.92	146
2	0.97	0.90	0.93	148
3	0.95	0.99	0.97	155
Accuracy			0.94	600
Macro AVG	0.95	0.94	0.94	600
Weighted AVG	0.95	0.94	0.94	600

Real vs Predição		
Nº Resg.	Real	Predição
1860	0	0
353	2	2
1333	1	1
905	3	3
1289	1	1
1273	1	1
938	2	2
1731	0	0
65	3	3
1323	1	1

Com base nas métricas de teste que tem maior peso para decisão de qual método usar. O método **Regressão Logística** é a melhor opção no geral alcançando 94% de acurácia, um excelente resultado para o método. O método **RNA Classificação** também obteve um bom resultado com 94% de acurácia.

Algoritmo Regressão Linear

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import matplotlib.pyplot as plt

# Carregar o dataset
file_path = 'c:/emprego.csv'
data = pd.read_csv(file_path)

# Remover linhas com valores nulos
data = data.dropna(subset=['salary'])

# Selecionar variáveis independentes e a variável dependente
X = data.drop(columns=['sl_no', 'salary', 'status'])
y = data['salary']

# One-hot encoding para variáveis categóricas
X = pd.get_dummies(X, drop_first=True)

# Dividir os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Treinar o modelo de regressão linear
model = LinearRegression()
model.fit(X_train, y_train)

# Fazer previsões
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Avaliar o modelo no conjunto de treino
mse_train = mean_squared_error(y_train, y_pred_train)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)

# Avaliar o modelo no conjunto de teste
mse_test = mean_squared_error(y_test, y_pred_test)
rmse_test = np.sqrt(mse_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
r2_test = r2_score(y_test, y_pred_test)

# Imprimir métricas de treino
print('Métricas de Treino:')
print(f'MSE: {mse_train:.2f}')
print(f'RMSE: {rmse_train:.2f}')
print(f'MAE: {mae_train:.2f}')
print(f'R2: {r2_train:.2f}')
```

```
# Imprimir métricas de teste
print('\nMétricas de Teste:')
print(f'MSE: {mse_test:.2f}')
print(f'RMSE: {rmse_test:.2f}')
print(f'MAE: {mae_test:.2f}')
print(f'R2: {r2_test:.2f}')

# Real vs Previsões de teste
predictions_df = pd.DataFrame({
    'Real': y_test,
    'Predição': y_pred_test
}).head(10)

print('\nPrimeiras 10 Previsões de Teste:')
print(predictions_df)
```

Algoritmo Random Forest

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor

# Carregar o dataset
file_path = 'c:/emprego.csv'
data = pd.read_csv(file_path)

# Remover linhas com valores nulos
data = data.dropna(subset=['salary'])

# Selecionar variáveis independentes e a variável dependente
X = data.drop(columns=['sl_no', 'salary', 'status'])
y = data['salary']

# One-hot encoding para variáveis categóricas
X = pd.get_dummies(X, drop_first=True)

# Dividir os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Treinar um modelo de Random Forest para avaliar a importância das características
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Obter a importância das características
feature_importances = rf_model.feature_importances_

# Criar um DataFrame para visualizar as importâncias
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Plotar Características Relevantes
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Características Relevantes')
plt.show()

# As 6 Características mais Relevantes
top_features = feature_importance_df['Feature'].head(6).values
X_train_top = X_train[top_features]
X_test_top = X_test[top_features]

# Treinar o modelo de regressão linear
model = LinearRegression()
```

```
model.fit(X_train_top, y_train)

# Fazer previsões
y_pred_train = model.predict(X_train_top)
y_pred_test = model.predict(X_test_top)

# Avaliar o modelo no conjunto de treino
mse_train = mean_squared_error(y_train, y_pred_train)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)

# Avaliar o modelo no conjunto de teste
mse_test = mean_squared_error(y_test, y_pred_test)
rmse_test = np.sqrt(mse_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
r2_test = r2_score(y_test, y_pred_test)

# Imprimir métricas de treino
print('Métricas de Treino:')
print(f'MSE: {mse_train:.2f}')
print(f'RMSE: {rmse_train:.2f}')
print(f'MAE: {mae_train:.2f}')
print(f'R2: {r2_train:.2f}')

# Imprimir métricas de teste
print('\nMétricas de Teste:')
print(f'MSE: {mse_test:.2f}')
print(f'RMSE: {rmse_test:.2f}')
print(f'MAE: {mae_test:.2f}')
print(f'R2: {r2_test:.2f}')

# Real vs Previsões de teste
predictions_df = pd.DataFrame({
    'Real': y_test,
    'Predição': y_pred_test
}).head(10)

print('\nPrimeiras 10 Previsões de Teste:')
print(predictions_df)
```

Algoritmo RNA Regressão

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler

# Carregar dados
df = pd.read_csv('c:/emprego.csv')
df.dropna(inplace=True)

x = df[['ssc_p', 'hsc_p', 'degree_p', 'etest_p', 'mba_p']]
y = df['salary']

# Normalizar os dados
scaler = MinMaxScaler()
x = scaler.fit_transform(x)

# Dividir os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

# Definir o modelo
modelo = MLPRegressor(random_state=20)

# Definir a grade de parâmetros
param_grid = {
    'hidden_layer_sizes': [(10, 5, 5), (50, 30, 10), (100,)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam'],
    'learning_rate': ['constant', 'adaptive'],
    'learning_rate_init': [0.001, 0.01, 0.1],
    'max_iter': [300, 500, 1000]
}

# Configurar o Grid Search
grid_search = GridSearchCV(estimator=modelo, param_grid=param_grid, cv=3, verbose=2,
n_jobs=-1, scoring='r2')

# Treinar o modelo usando Grid Search
grid_search.fit(X_train, y_train)

# Mostrar os melhores parâmetros
print("Melhores parâmetros encontrados: ", grid_search.best_params_)

# Avaliar o modelo com os melhores parâmetros
best_model = grid_search.best_estimator_

plt.plot(best_model.loss_curve_)
plt.xlabel('Época')
plt.ylabel('Loss')

print('Dados de treinamento')
```

```
predicao = best_model.predict(X_train)

print('R2:', metrics.r2_score(y_train, predicao))
print('MAE:', metrics.mean_absolute_error(y_train, predicao))
print('MSE:', metrics.mean_squared_error(y_train, predicao))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, predicao)))

print('Dados de teste')

predicao1 = best_model.predict(X_test)

print('R2:', metrics.r2_score(y_test, predicao1))
print('MAE:', metrics.mean_absolute_error(y_test, predicao1))
print('MSE:', metrics.mean_squared_error(y_test, predicao1))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predicao1)))

# Real vs Previsões de teste
predictions_df = pd.DataFrame({
    'Real': y_test,
    'Predição': predicao1
}).head(10)

print('\nPrimeiras 10 Previsões de Teste:')
print(predictions_df)
```

Algoritmo Regressão Logística

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Carregar a planilha
file_path = 'c:/celulares.csv'
df = pd.read_csv(file_path)

# Separar as características (X) e a variável alvo (y)
X = df.drop(columns=['price_range'])
y = df['price_range']

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalizar as características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Treinar o modelo de regressão logística com validação cruzada para ajustar os
hiperparâmetros
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['lbfgs', 'liblinear', 'saga']
}
log_reg = LogisticRegression(max_iter=1000)
grid_search = GridSearchCV(log_reg, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Melhor modelo
best_model = grid_search.best_estimator_

# Avaliar a acurácia no conjunto de teste
y_pred = best_model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)

print("Melhor modelo:", best_model)
#print("Acurácia no conjunto de teste:", accuracy)
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))

# Real vs Previsões de teste
predictions_df = pd.DataFrame({
    'Real': y_test,
    'Predição': y_pred
}).head(10)
```

Algoritmo Árvore Decisão

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Carregar a planilha
file_path = 'c:/celulares.csv'
df = pd.read_csv(file_path)

# Separar as características (X) e a variável alvo (y)
X = df.drop(columns=['price_range'])
y = df['price_range']

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalizar as características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Treinar o modelo de árvore de decisão com validação cruzada para ajustar os hiperparâmetros
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5, 10]
}
tree = DecisionTreeClassifier()
grid_search = GridSearchCV(tree, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Melhor modelo
best_tree_model = grid_search.best_estimator_

# Avaliar a acurácia no conjunto de teste
y_pred_tree = best_tree_model.predict(X_test_scaled)
accuracy_tree = accuracy_score(y_test, y_pred_tree)

print("Melhor modelo de árvore de decisão:", best_tree_model)
#print("Acurácia no conjunto de teste:", accuracy_tree)
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred_tree))

# Real vs Previsões de teste
predictions_df = pd.DataFrame({
    'Real': y_test,
    'Predição': y_pred_tree
}).head(10)

print('\nPrimeiras 10 Previsões de Teste:')
print(predictions_df)
```

Algoritmo RNA Classificação

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Carregar os dados
file_path = 'c:/celulares.csv'
df = pd.read_csv(file_path)

# Visualizar as primeiras linhas do dataframe
print(df.head())

# Definir as features (X) e o target (y)
X = df.drop(columns=['price_range'])
y = df['price_range']

# Dividir os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Normalizar os dados
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Definir o modelo
modelo = MLPClassifier(random_state=42, max_iter=300)

# Definir a grade de parâmetros
param_grid = {
    'hidden_layer_sizes': [(50, 50, 50), (100,), (150, 100, 50)],
    'activation': ['tanh', 'relu'],
    'solver': ['adam', 'sgd'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}

# Configurar o Grid Search
grid_search = GridSearchCV(estimator=modelo, param_grid=param_grid, cv=3, verbose=2,
n_jobs=-1, scoring='accuracy')

# Treinar o modelo usando Grid Search
grid_search.fit(X_train, y_train)

# Mostrar os melhores parâmetros
print("Melhores parâmetros encontrados: ", grid_search.best_params_)

# Avaliar o modelo com os melhores parâmetros
best_model = grid_search.best_estimator_
y_pred_train = best_model.predict(X_train)
y_pred_test = best_model.predict(X_test)

from sklearn.metrics import classification_report
```

```
print(classification_report(y_test,y_pred_test,))

# Real vs Previsões de teste
predictions_df = pd.DataFrame({
    'Real': y_test,
    'Predição': y_pred_test
}).head(10)

print('\nPrimeiras 10 Previsões de Teste:')
print(predictions_df)
```