UNIVERSIDADE SÃO JUDAS TADEU
UC: INTELIGÊNCIA ARTIFICIAL

Gabriely Camile de Oliveira Martins – RA: 823155179
Camilo Caetano Pinheiro de Oliveira – RA: 822158928
Eliabe Trajano de Araújo – RA: 125111372575
Rafael Henrique Koppe – RA: 202013205

APLICAÇÃO DE ALGORITMOS DE INTELIGÊNCIA ARTIFICIAL EM DADOS DO CENSUS DOS ESTADOS UNIDOS

Random Forest e Regressão Logística

SÃO PAULO
2023

Gabriely Camile de Oliveira Martins – RA: 823155179
Camilo Caetano Pinheiro de Oliveira – RA: 82212184
Eliabe Trajano de Araújo – RA: 125111372575
Rafael Henrique Koppe – RA: 202013205

APLICAÇÃO DE ALGORITMOS DE INTELIGÊNCIA ARTIFICIAL EM DADOS DO CENSUS DOS ESTADOS UNIDOS

Random Forest e Regressão Logística

Trabalho apresentado para a unidade curricular de inteligência artificial

Professores:
Cleber Silva
Vinicius Cassol

SÃO PAULO
2023

Sumário

Base de Dados

O que é o Census Data?

Census Data (dados do censo) são informações coletadas com o objetivo de obter uma ampla gama de informações demográficas, socioeconômicas e geográficas sobre a população dos Estados Unidos.
Sobre os dados de treino e teste utilizados:
A base de dados utilizada contém dados relacionados à saúde de habitantes dos Estados Unidos.

Métodos de IA

Random Forest

O algoritmo Random Forest é um método de aprendizado de máquina que combina várias árvores de decisão individuais para realizar classificação ou regressão. Ele foi proposto por Leo Breiman e Adele Cutler em 2001. A Random Forest é uma técnica popular devido à sua capacidade de lidar com problemas complexos, como dados de alta dimensionalidade, variáveis correlacionadas e outliers.
A Random Forest opera criando uma coleção (ou floresta) de árvores de decisão. Cada árvore é construída a partir de uma amostra aleatória dos dados de treinamento, e em cada divisão da árvore, apenas um subconjunto aleatório das variáveis de entrada é considerado. Isso ajuda a evitar o overfitting (sobreajuste) e aumenta a diversidade das árvores na floresta.
Para realizar uma previsão usando uma Random Forest, cada árvore individual é percorrida e a previsão é obtida a partir da votação majoritária (no caso de classificação) ou da média (no caso de regressão) das previsões das árvores.
A Random Forest tem várias vantagens, como a capacidade de lidar com grandes conjuntos de dados, detectar interações entre variáveis e fornecer uma medida de importância das variáveis. Além disso, ela é menos sensível ao overfitting do que uma única árvore de decisão.

Regressão Logística

A regressão logística é um algoritmo de aprendizado de máquina usado para realizar análise de classificação binária, ou seja, para prever a probabilidade de uma variável de resposta pertencer a uma das duas classes possíveis. É uma técnica amplamente utilizada em diversos campos, como ciência de dados, medicina, finanças e ciências sociais.

Ao contrário da regressão linear, que é usada para prever valores contínuos, a regressão logística é adequada para problemas de classificação em que a variável de resposta é binária (por exemplo, verdadeiro/falso, positivo/negativo, sim/não). A regressão logística estima a probabilidade de um evento ocorrer, com base em um conjunto de variáveis independentes (também chamadas de variáveis preditoras ou de entrada).

O modelo de regressão logística aplica a função logit à combinação linear das variáveis independentes. A função logit transforma a probabilidade de um evento ocorrer em uma escala logarítmica, mapeando-a para um valor que varia de menos infinito a mais infinito. Em seguida, a função logística, também conhecida como função sigmóide, é aplicada ao valor logit para mapeá-lo em uma probabilidade que varia de 0 a 1.

Durante o treinamento do modelo de regressão logística, são estimados coeficientes para cada variável independente, de forma a maximizar a verossimilhança dos dados observados. Esses coeficientes são usados para calcular as probabilidades previstas para novos exemplos.

Existem várias abordagens para estimar os coeficientes da regressão logística, como o método de máxima verossimilhança ou o método de mínimos quadrados ponderados. O modelo final pode ser avaliado usando várias métricas, como a acurácia, a curva ROC e a precisão e recall.

Amostra dos Dados:

male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	0
0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	95.0	76.0	0
1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	75.0	70.0	0
0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	65.0	103.0	1
0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	85.0	85.0	0
0	43	2.0	0	0.0	0.0	0	1	0	228.0	180.0	110.0	30.30	77.0	99.0	0
0	63	1.0	0	0.0	0.0	0	0	0	205.0	138.0	71.0	33.11	60.0	85.0	1
0	45	2.0	1	20.0	0.0	0	0	0	313.0	100.0	71.0	21.68	79.0	78.0	0
1	52	1.0	0	0.0	0.0	0	1	0	260.0	141.5	89.0	26.36	76.0	79.0	0
1	43	1.0	1	30.0	0.0	0	1	0	225.0	162.0	107.0	23.61	93.0	88.0	0

Variáveis:

- male: Sexo (1 = Masculino, 0 = Feminino);
- age: Idade;
- education: Nível de educação (1 = Analfabeto, 2 = Ensino Fundamental, 3 = Ensino Médio, 4 = Ensino Superior);
- currentSmoker: Fumante atual (1 = Sim, 0 = Não);
- cigsPerDay: Cigarros por dia;
- BPMeds: Uso de medicamentos para pressão arterial (1 = Sim, 0 = Não);
- prevalentStroke: Histórico de AVC (1 = Sim, 0 = Não);
- prevalentHyp: Histórico de hipertensão (1 = Sim, 0 = Não);
- diabetes: Diabetes (1 = Sim, 0 = Não);
- totChol: Colesterol total;
- sysBP: Pressão arterial sistólica;
- diaBP: Pressão arterial diastólica;
- BMI: Índice de massa corporal;
- heartRate: Frequência cardíaca;
- glucose: Glicose;
- TenYearCHD: Risco de doença coronariana em 10 anos (1 = Sim, 0 = Não)

Variável Alvo/Variável de Saída: Diabetes

Implementação dos algoritmos de ML usando Python

Este script é uma implementação em Python de um pipeline de aprendizado de máquina (ML) para tarefas de classificação. Ele realiza análise de dados, pré-processamento de dados, treinamento e avaliação de modelos. O script utiliza várias bibliotecas e módulos, como pandas, numpy, matplotlib, seaborn, scikit-learn, warnings, ipywidgets e IPython.display.

Bibliotecas e Módulos

O script importa as seguintes bibliotecas e módulos:

- pandas (importado como pd): Fornece estruturas de dados e ferramentas de análise de dados.
- numpy (importado como np): Oferece suporte a matrizes e matrizes multidimensionais grandes, juntamente com uma coleção de funções matemáticas para operar nessas matrizes.
- matplotlib.pyplot (importado como plt): Fornece funcionalidades de plotagem para criar visualizações.
- seaborn (importado como sns): Melhora a aparência visual dos gráficos do matplotlib e fornece gráficos estatísticos adicionais.
- sklearn.ensemble.RandomForestClassifier: Uma classe para construir classificadores de floresta aleatória.
- sklearn.linear_model.LogisticRegression: Uma classe para construir modelos de regressão logística.
- sklearn.metrics.accuracy_score: Uma função para calcular a pontuação de precisão da classificação.
- sklearn.metrics.confusion_matrix: Uma função para calcular a matriz de confusão.
- sklearn.model_selection.train_test_split: Uma função para dividir os dados em conjuntos de treinamento e teste.
- sklearn.preprocessing.StandardScaler: Uma classe para padronizar as características, removendo a média e escalonando para a variância unitária.

- sklearn.impute.SimpleImputer: Uma classe para lidar com valores ausentes por imputação.
- warnings: Um módulo para gerenciar mensagens de aviso durante a execução do script.
- ipywidgets: Uma biblioteca para widgets HTML interativos em notebooks Jupyter.
- IPython.display: Um módulo para gerenciar a exibição de tipos de mídia ricos em notebooks Jupyter.

Carregamento de Dados

O script carrega os dados de um arquivo CSV especificado pela variável file_path. Ele usa a função pd.read_csv() da biblioteca pandas para ler o arquivo CSV em um DataFrame do pandas. Se o arquivo não for encontrado, será lançado um FileNotFoundError.

Classes Personalizadas

O script define duas classes personalizadas derivadas das classes LogisticRegression e RandomForestClassifier do scikit-learn. Essas classes personalizadas, CustomLogisticRegression e CustomRandomForestClassifier, substituem o método fit() para suprimir os UserWarnings durante o treinamento do modelo.

Interação com o Usuário

O script utiliza ipywidgets para fornecer interação com o usuário por meio de menus suspensos e campos de entrada. O usuário pode selecionar a variável alvo, escolher uma estratégia de tratamento de dados ausentes e especificar a porcentagem de dados de treinamento.

Análise de Dados

Quando o usuário clica no botão "Analyze", o script executa a função analyze_button_clicked(). Dentro dessa função, a função analyze_data() é definida e chamada.

A função analyze_data() realiza as seguintes etapas:

1. Define X e y com base na variável alvo selecionada.
2. Exibe estatísticas gerais para o conjunto de dados usando o método describe() do DataFrame.
3. Exibe estatísticas para a variável alvo usando o método value_counts().
4. Divide os dados em conjuntos de treinamento e teste usando train_test_split().
5. Trata valores ausentes com base na estratégia selecionada:
    - Se a estratégia for "remove", remove linhas com valores ausentes tanto nos conjuntos de treinamento quanto nos de teste.
    - Se a estratégia for qualquer outro valor ("mean", "median", "most_frequent" ou "constant"), usa um SimpleImputer para preencher os valores ausentes com a estratégia escolhida.
6. Padroniza os dados usando um StandardScaler.
7. Cria instâncias das classes CustomLogisticRegression e CustomRandomForestClassifier.
8. Define a função train_and_evaluate_model() para treinar e avaliar os modelos.
9. Treina e avalia os modelos de regressão logística e floresta aleatória.
10. Exibe os scores de precisão e as matrizes de confusão para ambos os modelos.
11. Exibe um gráfico de barras mostrando a importância das variáveis para o modelo de floresta aleatória.
12. Exibe um mapa de calor da matriz de correlação para os dados de treinamento.

Script Python

1º Importa as bibliotecas necessárias:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import warnings
import ipywidgets as widgets
from IPython.display import display
```

2º Baixa e Carrega os dados de um arquivo CSV hospedado no Google Drive:

```
# URL do link de compartilhamento do Google Drive
url = 'https://drive.google.com/uc?id=1zN2FLFTboKAD7eu_9Xh1PCVbAb18jCas'

# Faz o download do arquivo usando o gdown
output = 'DadosDoCensus.csv'
gdown.download(url, output, quiet=False)

# Lê o arquivo CSV com o Pandas
df = pd.read_csv(output)
```

3º Define uma função para determinar se uma variável é binária:

```
# Function to determine if the variable is binary
def is_binary_variable(column):
    unique_values = column.unique()
    return len(unique_values) == 2 and all(value in [0, 1] for value in unique_values)
```

4º Define duas classes personalizadas (CustomLogisticRegression e CustomRandomForestClassifier) que herdam de classes do sklearn e sobrescrevem o método fit para ignorar avisos:

```
class CustomLogisticRegression(LogisticRegression):
    def fit(self, X, y):
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=UserWarning)
            super().fit(X, y)

class CustomRandomForestClassifier(RandomForestClassifier):
    def fit(self, X, y):
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=UserWarning)
            super().fit(X, y)
```

5º Cria widgets para selecionar a variável alvo, a estratégia de tratamento de dados ausentes e a porcentagem de dados de treinamento:

```
# Select the target variable
target_variable = widgets.Dropdown(options=df.columns.tolist(), description='Target Variable:')
display(target_variable)

# Select the missing data handling strategy
imputer_strategy = widgets.Dropdown(
    options=['mean', 'median', 'most_frequent', 'constant', 'remove'],
    description='Missing Data Handling Strategy:'
)
display(imputer_strategy)

# Input field for entering the percentage of training data
train_data_percentage = widgets.FloatText(value=70, description='Training Data Percentage (%):')
display(train_data_percentage)
```

6º Define uma função que será chamada quando o botão "Analyze" for clicado. Esta função realiza a análise dos dados:

```
def analyze_button_clicked(b):
    # Display the target variable
    print(f"Target Variable: {target_variable.value}")
```

```
    analyze_data()

analyze_button = widgets.Button(description="Analyze")
analyze_button.on_click(analyze_button_clicked)
display(analyze_button)
```

Dentro da função **analyze_button_clicked**, as etapas de análise de dados são realizadas:

7º Exibe a variável alvo selecionada:

```
print(f"Target Variable: {target_variable.value}")
```

8º Define X e y com base na variável alvo selecionada:

```
X = df.drop(columns=[target_variable.value])
y = df[target_variable.value]
```

9º Exibe estatísticas gerais do conjunto de dados:

```
print("Dataset Statistics:")
display(df.describe(include='all'))
print()
```

10º Exibe estatísticas da variável alvo:

```
print("Target Variable Statistics:")
display(y.value_counts())
print()
```

11º Divide os dados em conjuntos de treinamento e teste:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=1 - train_data_percentage.value / 100, random_state=42
)
```

12º Verifica a estratégia selecionada para tratar os dados ausentes. Se a estratégia for "remove", remove as linhas com valores ausentes. Caso contrário, preenche os valores ausentes usando um imputer (preenchedor):

```
if imputer_strategy.value == 'remove':
    # Remove rows with missing values
    original_shape = X_train.shape[0]
    X_train = X_train.dropna()
    y_train = y_train[X_train.index]
    X_test = X_test.dropna()
    y_test = y_test[X_test.index]
    removed_rows = original_shape - X_train.shape[0]
    print(f"Removed {removed_rows} rows with missing values.")
    print(f"Total rows: {original_shape}")
    print(f"Remaining rows: {X_train.shape[0]}")
    print(f"Impact on analysis: {(removed_rows / original_shape) * 100}%")
    print()
    # Update the number of training and testing rows
    training_rows = X_train.shape[0]
    testing_rows = X_test.shape[0]
    print(f"Number of Training Rows: {training_rows}")
    print(f"Number of Testing Rows: {testing_rows}")
    print()

else:
    # Fill missing values using an imputer
    if imputer_strategy.value == 'mean':
        imputer = SimpleImputer(strategy='mean')
    elif imputer_strategy.value == 'median':
        imputer = SimpleImputer(strategy='median')
    elif imputer_strategy.value == 'most_frequent':
        imputer = SimpleImputer(strategy='most_frequent')
    elif imputer_strategy.value == 'constant':
        imputer = SimpleImputer(strategy='constant', fill_value=0)  # Replace 0 with the desired constant value
    X_train = imputer.fit_transform(X_train)
    X_test = imputer.transform(X_test)
```

13º Padroniza os dados usando um StandardScaler:

```
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X.columns)
X_test = pd.DataFrame(scaler.transform(X_test), columns=X.columns)
```

14º Cria instâncias dos modelos CustomLogisticRegression e CustomRandomForestClassifier e define uma função para treinar e avaliar os modelos:

```
logistic_regression_model = CustomLogisticRegression()
random_forest_model = CustomRandomForestClassifier(n_estimators=100, random_state=42)

# Function to train and evaluate the models
def train_and_evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    return accuracy, cm
```

15º Treina e avalia os modelos:

```
logistic_regression_accuracy, logistic_regression_cm = train_and_evaluate_model(
    logistic_regression_model, X_train, X_test, y_train, y_test
)
random_forest_accuracy, random_forest_cm = train_and_evaluate_model(
    random_forest_model, X_train, X_test, y_train, y_test
)
```

16º Exibe os resultados de acurácia e as matrizes de confusão dos modelos:

```
print("Logistic Regression Accuracy:", logistic_regression_accuracy)
print("Logistic Regression Confusion Matrix:")
plt.figure()
sns.heatmap(logistic_regression_cm, annot=True, fmt='d', cmap='Blues')
plt.title("Logistic Regression Confusion Matrix")
plt.show()
print()
print("Random Forest Accuracy:", random_forest_accuracy)
print("Random Forest Confusion Matrix:")
plt.figure()
sns.heatmap(random_forest_cm, annot=True, fmt='d', cmap='Blues')
plt.title("Random Forest Confusion Matrix")
plt.show()
print()
```

17º Exibe o gráfico de importância das variáveis:

```
random_forest_model.fit(X_train, y_train)
importances = random_forest_model.feature_importances_
indices = np.argsort(importances)[::-1]
plt.figure(figsize=(10, 6))
plt.bar(range(X_train.shape[1]), importances[indices])
plt.xticks(range(X_train.shape[1]), X_train.columns[indices], rotation=90)
plt.title("Variable Importance")
plt.tight_layout()
plt.show()
```

18º Exibe a matriz de correlação:

```
correlation_matrix = X_train.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.tight_layout()
plt.show()
```

19º Chama a função analyze_data() para realizar a análise quando o botão "Analyze" for clicado:

```
analyze_data()
```

Resultados Obtidos

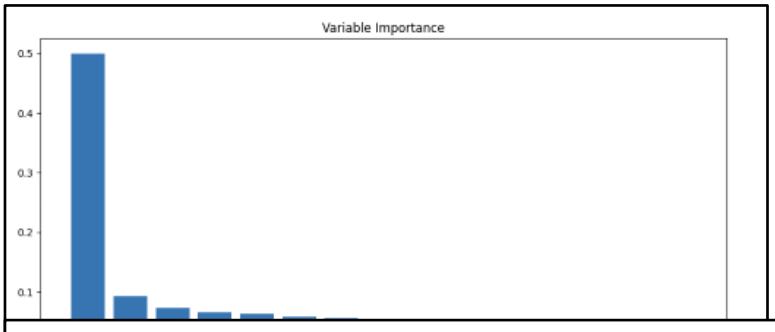
Variável Alvo/Variável de Saída: Diabetes

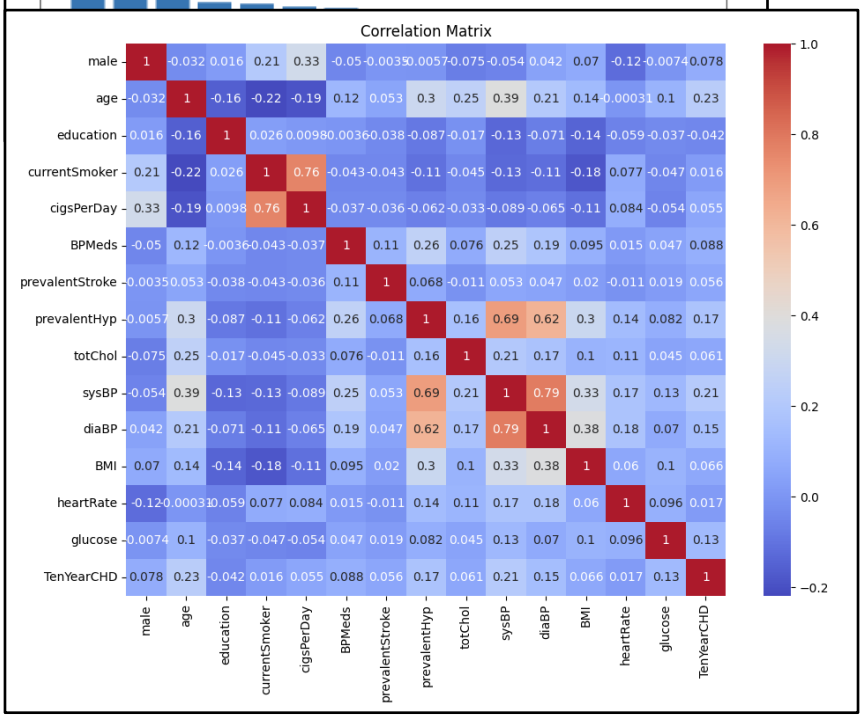
```
Number of Training Rows: 2966
Number of Testing Rows: 1272

Logistic Regression Accuracy: 0.9858490566037735
```

```
Random Forest Accuracy: 0.985062893081761
```

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
count	4238.000	4238.000	4133.000	4238.000	4209.000	4185.000	4238.000	4238.000	4238.000	4188.000	4238.000	4238.000	4219.000	4237.000	3850.000	4238.000
mean	0.429	49.585	1.979	0.494	9.003	0.030	0.006	0.311	0.026	236.722	132.352	82.893	25.802	75.879	81.967	0.152
std	0.495	8.572	1.020	0.500	11.920	0.170	0.077	0.463	0.158	44.590	22.038	11.911	4.080	12.027	23.960	0.359
min	0.000	32.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	107.000	83.500	48.000	15.540	44.000	40.000	0.000
25%	0.000	42.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	206.000	117.000	75.000	23.070	68.000	71.000	0.000
50%	0.000	49.000	2.000	0.000	0.000	0.000	0.000	0.000	0.000	234.000	128.000	82.000	25.400	75.000	78.000	0.000
75%	1.000	56.000	3.000	1.000	20.000	0.000	0.000	1.000	0.000	263.000	144.000	89.875	28.040	83.000	87.000	0.000
max	1.000	70.000	4.000	1.000	70.000	1.000	1.000	1.000	1.000	696.000	295.000	142.500	56.800	143.000	394.000	1.000





Referências Bibliográficas

Base de dados Census: https://data.census.gov/

Random Forest: Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32

Regressão Logística: Hosmer Jr., D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied logistic regression. John Wiley & Sons