Comparative Analysis of Machine Learning Algorithms in Predicting Smoking and Drinking Behaviors

December 5, 2023

1 Trabalho final Sistemas Inteligentes

Vitor Albuquerque de Paula

[1]:
```
# Etapa 1: Carregar e Preparar o Dataset

import pandas as pd

# Caminho do dataset
dataset_path = './smoking_driking_dataset_Ver01.csv'  # Ajuste conforme o␣
 ↪caminho real

# Tentando carregar o dataset
try:
    data = pd.read_csv(dataset_path)
    # Exibindo as primeiras linhas do dataset para verificação
    display_data = data.head()
except Exception as e:
    display_data = str(e)

display_data
```

[1]:

	sex	age	height	weight	waistline	sight_left	sight_right	hear_left	\
0	Male	35	170	75	90.0	1.0	1.0	1.0	
1	Male	30	180	80	89.0	0.9	1.2	1.0	
2	Male	40	165	75	91.0	1.2	1.5	1.0	
3	Male	50	175	80	91.0	1.5	1.2	1.0	
4	Male	50	165	60	80.0	1.0	1.2	1.0	

	hear_right	SBP	…	LDL_chole	triglyceride	hemoglobin	urine_protein	\
0	1.0	120.0	…	126.0	92.0	17.1	1.0	
1	1.0	130.0	…	148.0	121.0	15.8	1.0	
2	1.0	120.0	…	74.0	104.0	15.8	1.0	
3	1.0	145.0	…	104.0	106.0	17.6	1.0	
4	1.0	138.0	…	117.0	104.0	13.8	1.0	

```
   serum_creatinine  SGOT_AST  SGOT_ALT  gamma_GTP  SMK_stat_type_cd  DRK_YN
0               1.0      21.0      35.0       40.0               1.0       Y
1               0.9      20.0      36.0       27.0               3.0       N
2               0.9      47.0      32.0       68.0               1.0       N
3               1.1      29.0      34.0       18.0               1.0       N
4               0.8      19.0      12.0       25.0               1.0       N

[5 rows x 24 columns]
```

[2]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Imputação dos valores faltantes
imputer = SimpleImputer(strategy='mean')
numeric_columns = data.select_dtypes(include=[float, int]).columns
numeric_columns = numeric_columns.drop(['SMK_stat_type_cd'])  # Exclua 'DRK_YN' se ela já for categórica
data[numeric_columns] = imputer.fit_transform(data[numeric_columns])

# Normalizar os dados numéricos
scaler = StandardScaler()
data[numeric_columns] = scaler.fit_transform(data[numeric_columns])

# Codificação de variáveis categóricas
label_encoders = {}
categorical_columns = ['sex', 'SMK_stat_type_cd', 'DRK_YN']  # Adicione outras colunas categóricas se necessário
for col in categorical_columns:
    if col in data.columns:
        le = LabelEncoder()
        data[col] = le.fit_transform(data[col])
        label_encoders[col] = le

# Visualizando os dados após o pré-processamento
print(data.head())
```

```
   sex       age    height    weight  waistline  sight_left  sight_right  \
0    1 -0.889514  0.835874  0.936210   0.739781    0.031629     0.035668
1    1 -1.242090  1.913117  1.335755   0.655395   -0.133401     0.366370
2    1 -0.536938  0.297252  0.936210   0.824167    0.361690     0.862423
3    1  0.168215  1.374495  1.335755   0.824167    0.856782     0.366370
4    1  0.168215  0.297252 -0.262425  -0.104078    0.031629     0.366370

   hear_left  hear_right       SBP  …  LDL_chole  triglyceride  hemoglobin  \
0  -0.180329   -0.177296 -0.167261  …   0.361643     -0.392788    1.810919
1  -0.180329   -0.177296  0.520349  …   0.975435     -0.109022    0.990693
2  -0.180329   -0.177296 -0.167261  …  -1.089136     -0.275368    0.990693
3  -0.180329   -0.177296  1.551763  …  -0.252148     -0.255798    2.126391
```

```
4  -0.180329   -0.177296  1.070436  …   0.110547     -0.275368   -0.271194

   urine_protein  serum_creatinine  SGOT_AST  SGOT_ALT  gamma_GTP  \
0       -0.21526          0.290374 -0.212371  0.351404   0.056791
1       -0.21526          0.082270 -0.254936  0.389415  -0.201022
2       -0.21526          0.082270  0.894324  0.237373   0.612081
3       -0.21526          0.498477  0.128151  0.313394  -0.379508
4       -0.21526         -0.125833 -0.297501 -0.522835  -0.240685

   SMK_stat_type_cd  DRK_YN
0                 0       1
1                 2       0
2                 0       0
3                 0       0
4                 0       0

[5 rows x 24 columns]
```

[3]:
```python
from sklearn.model_selection import train_test_split

# Dividir os dados para prever 'SMK_stat_type_cd'
X = data.drop(['SMK_stat_type_cd', 'DRK_YN'], axis=1)
y_smk = data['SMK_stat_type_cd']
X_train_smk, X_test_smk, y_train_smk, y_test_smk = train_test_split(X, y_smk,␣
 ↪test_size=0.2, random_state=42)

# Dividir os dados para prever 'DRK_YN'
y_drk = data['DRK_YN']
X_train_drk, X_test_drk, y_train_drk, y_test_drk = train_test_split(X, y_drk,␣
 ↪test_size=0.2, random_state=42)

import numpy as np

X_train_smk = np.array(X_train_smk)
y_train_smk = np.array(y_train_smk)
X_test_smk = np.array(X_test_smk)
y_test_smk = np.array(y_test_smk)

X_train_drk = np.array(X_train_drk)
y_train_drk = np.array(y_train_drk)
X_test_drk = np.array(X_test_drk)
y_test_drk = np.array(y_test_drk)


# Verificando valores únicos para DRK_YN
print("Valores únicos em DRK_YN:", data['DRK_YN'].unique())
```

```python
# Verificando o balanceamento das classes
print("Contagem de classes em DRK_YN:")
print(data['DRK_YN'].value_counts())

# Verificando tipos de dados
print("\nTipos de dados:")
print(data.dtypes)

# Verificando dados faltantes
print("\nDados faltantes por coluna:")
print(data.isnull().sum())
```

```
Valores únicos em DRK_YN: [1 0]
Contagem de classes em DRK_YN:
DRK_YN
0    495858
1    495488
Name: count, dtype: int64

Tipos de dados:
sex                   int32
age                 float64
height              float64
weight              float64
waistline           float64
sight_left          float64
sight_right         float64
hear_left           float64
hear_right          float64
SBP                 float64
DBP                 float64
BLDS                float64
tot_chole           float64
HDL_chole           float64
LDL_chole           float64
triglyceride        float64
hemoglobin          float64
urine_protein       float64
serum_creatinine    float64
SGOT_AST            float64
SGOT_ALT            float64
gamma_GTP           float64
SMK_stat_type_cd      int64
DRK_YN                int32
dtype: object

Dados faltantes por coluna:
sex                 0
```

```
age                  0
height               0
weight               0
waistline            0
sight_left           0
sight_right          0
hear_left            0
hear_right           0
SBP                  0
DBP                  0
BLDS                 0
tot_chole            0
HDL_chole            0
LDL_chole            0
triglyceride         0
hemoglobin           0
urine_protein        0
serum_creatinine     0
SGOT_AST             0
SGOT_ALT             0
gamma_GTP            0
SMK_stat_type_cd     0
DRK_YN               0
dtype: int64
```

[4]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC

def apply_knn(X_train, y_train, X_test, y_test):
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)
    return knn.score(X_test, y_test)

def apply_random_forest(X_train, y_train, X_test, y_test):
    rf = RandomForestClassifier(n_estimators=100, verbose=True)
    rf.fit(X_train, y_train)
    return rf.score(X_test, y_test)

def apply_mlp(X_train, y_train, X_test, y_test):
    mlp = MLPClassifier(hidden_layer_sizes=(100,), verbose=True,␣
 ↪early_stopping=True)
    mlp.fit(X_train, y_train)
    return mlp.score(X_test, y_test)

def apply_svm(X_train, y_train, X_test, y_test):
```

```
    svm = SVC()
    svm.fit(X_train, y_train)
    return svm.score(X_test, y_test)
```

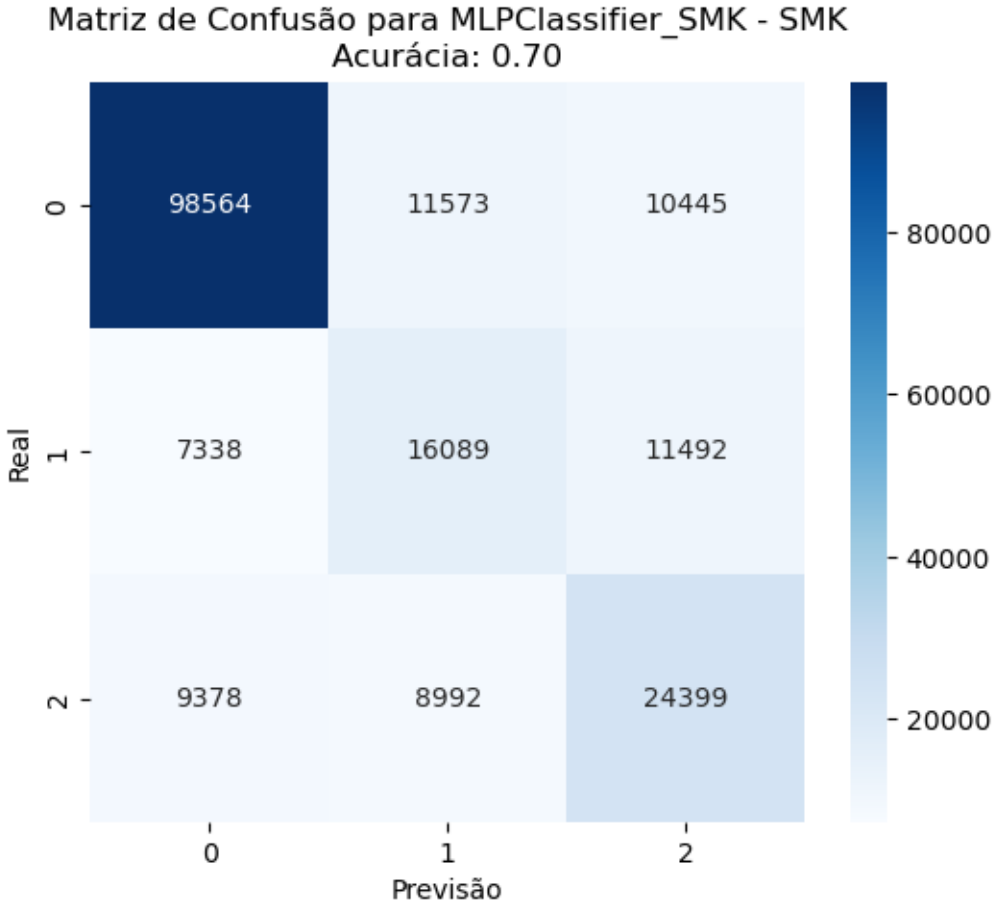
[8]:
```
from concurrent.futures import ThreadPoolExecutor, as_completed
import logging
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Configurando o logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -␣
 ↪%(message)s')

def train_and_evaluate(model, X_train, y_train, X_test, y_test, model_name,␣
 ↪dataset_name):
    logging.info(f"Iniciando treinamento do modelo {model_name} para o dataset␣
 ↪{dataset_name}")
    model.fit(X_train, y_train)
    accuracy = model.score(X_test, y_test)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    logging.info(f"Treinamento concluído para o modelo {model_name} no dataset␣
 ↪{dataset_name}")
    return model_name, dataset_name, accuracy, cm

# Definindo os modelos
models = [
    (KNeighborsClassifier(n_neighbors=5), "KNeighborsClassifier_SMK"),
    (RandomForestClassifier(n_estimators=100), "RandomForestClassifier_SMK"),
    (MLPClassifier(hidden_layer_sizes=(64,16,4), verbose=True,␣
 ↪early_stopping=True), "MLPClassifier_SMK"),
    #(SVC(), "SVC")
]

# Criando tarefas para treinamento
tasks = []
for model, model_name in models:
    tasks.append((model, X_train_smk, y_train_smk, X_test_smk, y_test_smk,␣
 ↪model_name, "SMK"))

# Executando treinamento em paralelo
results = []
with ThreadPoolExecutor(max_workers=len(models)) as executor:
    future_to_task = {executor.submit(train_and_evaluate, *task): task for task␣
 ↪in tasks}
```

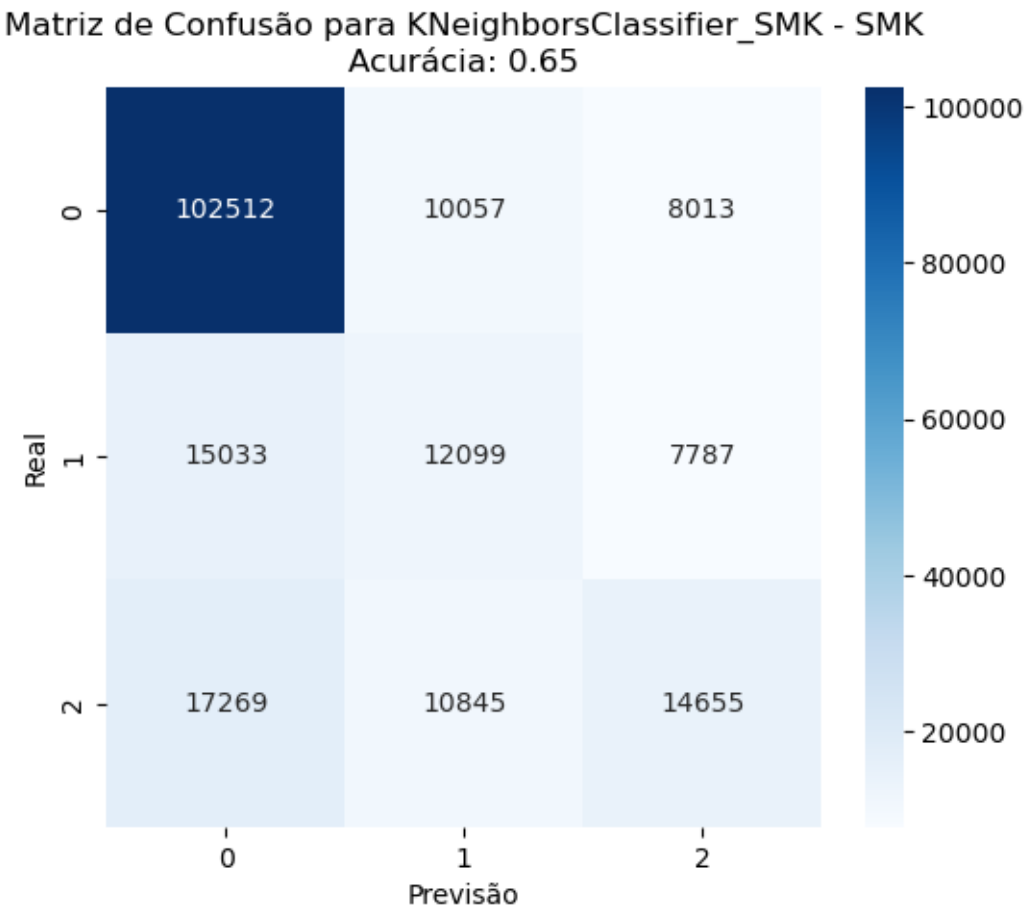
```
    for future in as_completed(future_to_task):
        results.append(future.result())

# Plotando as matrizes de confusão
for model_name, dataset_name, accuracy, cm in results:
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Matriz de Confusão para {model_name} - {dataset_name}\nAcurácia:
 ↪ {accuracy:.2f}")
    plt.xlabel('Previsão')
    plt.ylabel('Real')
    plt.show()
```

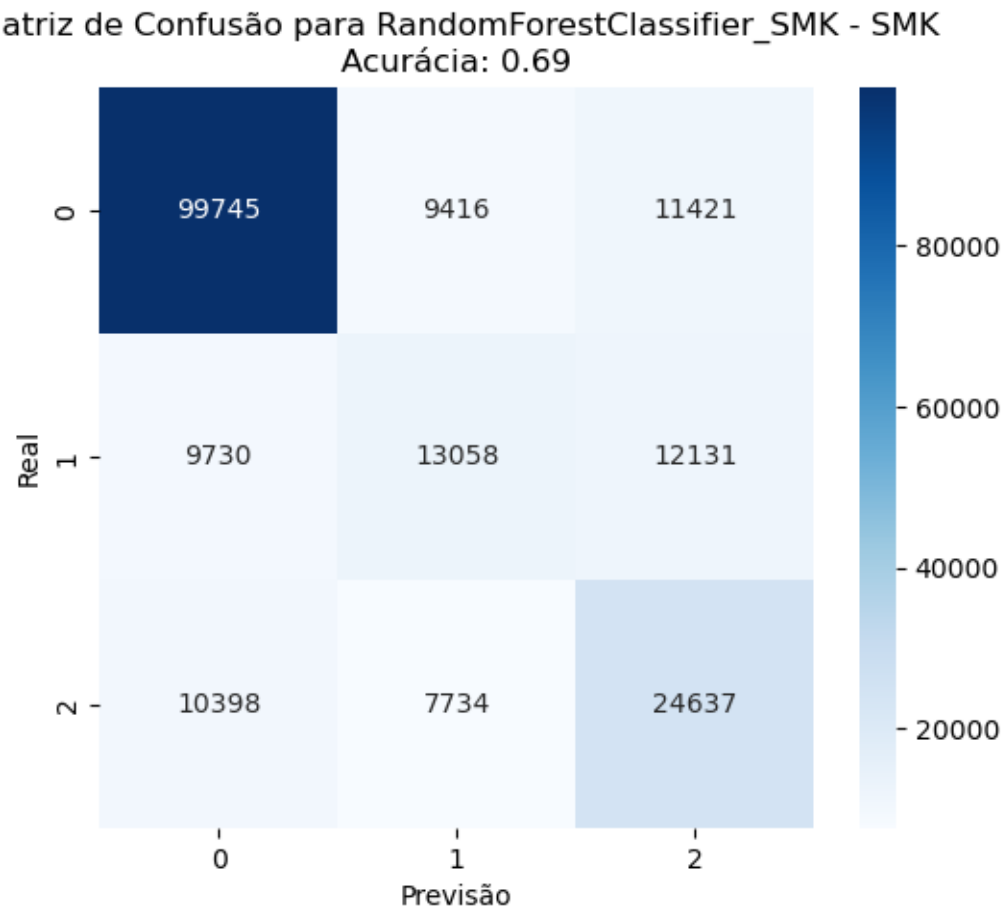
```
2023-12-05 19:13:17,028 - INFO - Iniciando treinamento do modelo
KNeighborsClassifier_SMK para o dataset SMK
2023-12-05 19:13:17,032 - INFO - Iniciando treinamento do modelo
RandomForestClassifier_SMK para o dataset SMK
2023-12-05 19:13:17,037 - INFO - Iniciando treinamento do modelo
MLPClassifier_SMK para o dataset SMK

Iteration 1, loss = 0.66513535
Validation score: 0.693511
Iteration 2, loss = 0.65147124
Validation score: 0.697609
Iteration 3, loss = 0.64913372
Validation score: 0.697571
Iteration 4, loss = 0.64812207
Validation score: 0.699904
Iteration 5, loss = 0.64730937
Validation score: 0.700093
Iteration 6, loss = 0.64678203
Validation score: 0.700119
Iteration 7, loss = 0.64639153
Validation score: 0.699337
Iteration 8, loss = 0.64597520
Validation score: 0.700018
Iteration 9, loss = 0.64577521
Validation score: 0.700661
Iteration 10, loss = 0.64556216
Validation score: 0.700182
Iteration 11, loss = 0.64534725
Validation score: 0.699299
Iteration 12, loss = 0.64520492
Validation score: 0.700509
Iteration 13, loss = 0.64498087
Validation score: 0.700535
Iteration 14, loss = 0.64490875
Validation score: 0.699173
```

```
Iteration 15, loss = 0.64474973
Validation score: 0.698517
Iteration 16, loss = 0.64476749
Validation score: 0.700081
Iteration 17, loss = 0.64462767
Validation score: 0.700711
Iteration 18, loss = 0.64446855
Validation score: 0.700459
Iteration 19, loss = 0.64442430
Validation score: 0.700068
Iteration 20, loss = 0.64433100
Validation score: 0.700030
Validation score did not improve more than tol=0.000100 for 10 consecutive
epochs. Stopping.
```

```
2023-12-05 19:17:45,372 - INFO - Treinamento concluído para o modelo
MLPClassifier_SMK no dataset SMK
2023-12-05 19:22:57,954 - INFO - Treinamento concluído para o modelo
KNeighborsClassifier_SMK no dataset SMK
2023-12-05 19:26:07,210 - INFO - Treinamento concluído para o modelo
RandomForestClassifier_SMK no dataset SMK
```




Matriz de Confusão para KNeighborsClassifier_SMK - SMK
Acurácia: 0.65
102512
10057
8013
15033
12099
7787
17269
10845
14655
Real
Previsão
0
1
2
100000
80000
60000
40000
20000

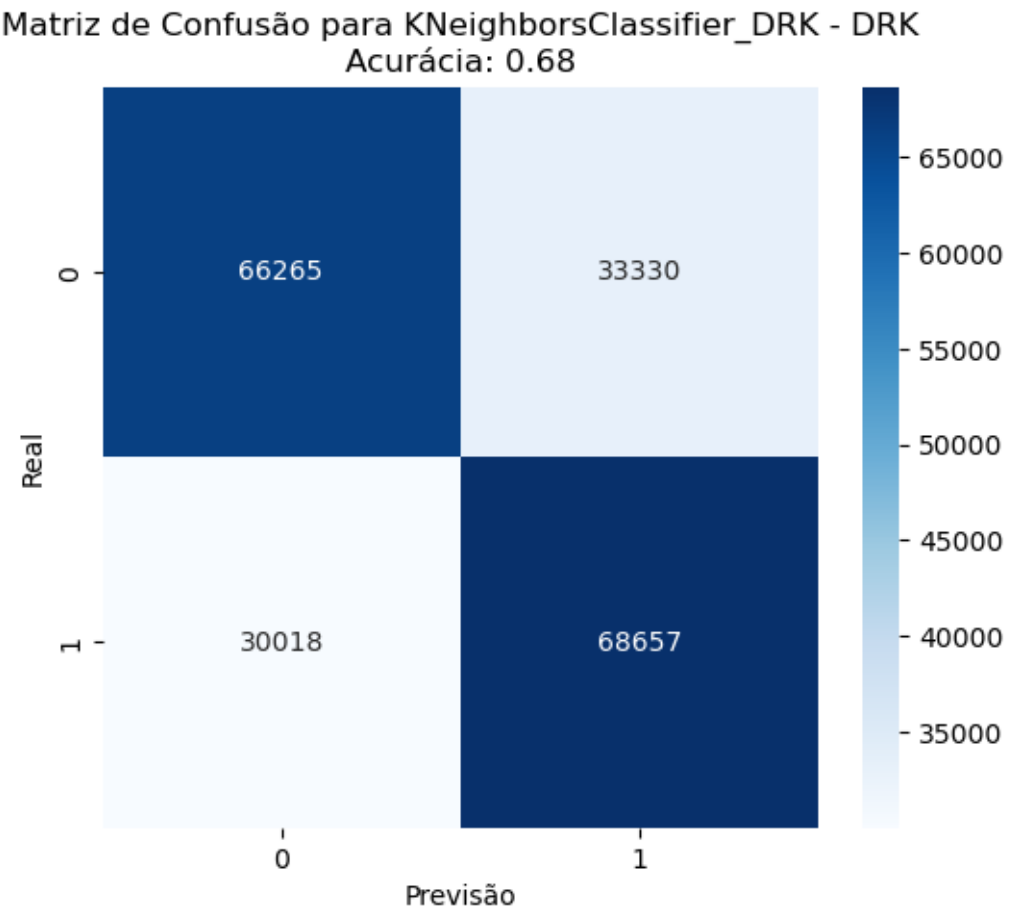


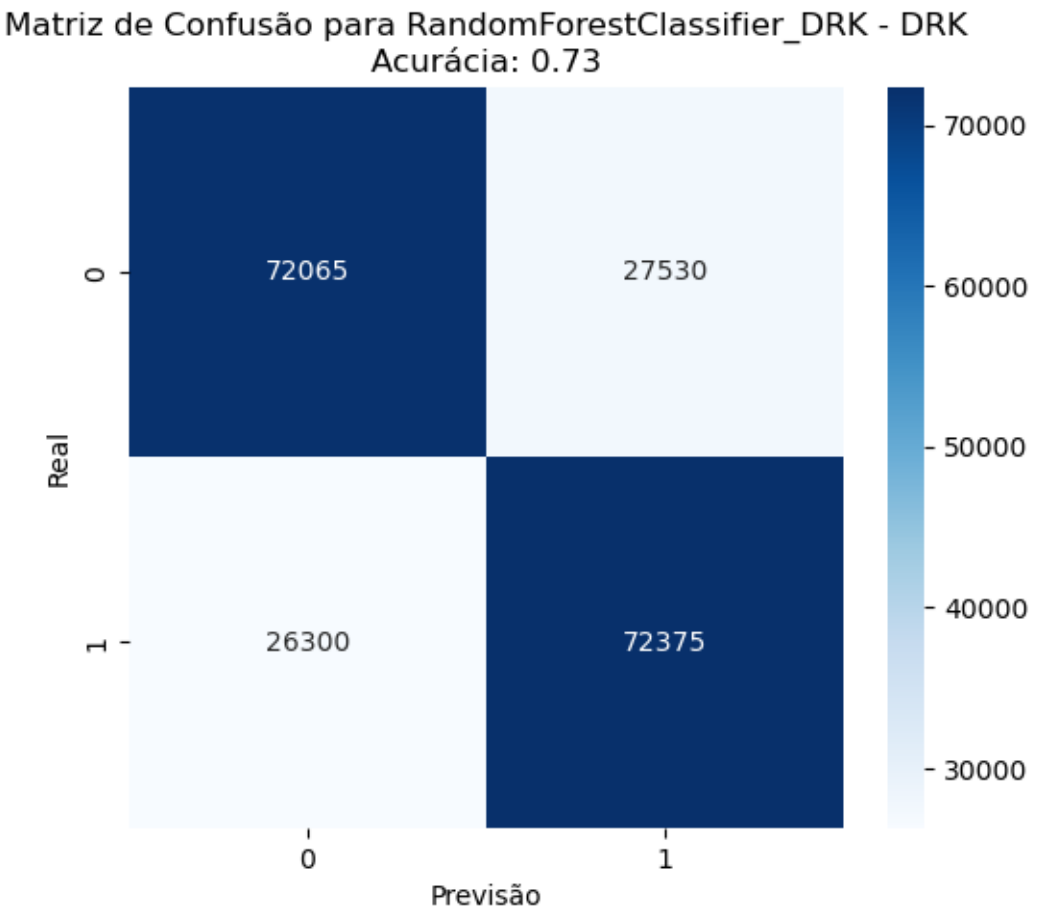
[9]:
```
from concurrent.futures import ThreadPoolExecutor, as_completed
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

def train_and_evaluate(model, X_train, y_train, X_test, y_test, model_name):
    print(f"Iniciando treinamento do modelo {model_name} para o dataset DRK")
    model.fit(X_train, y_train)
    accuracy = model.score(X_test, y_test)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(f"Treinamento concluído para o modelo {model_name} no dataset DRK")
    return model_name, accuracy, cm

# Definindo os modelos
models = [
    (KNeighborsClassifier(n_neighbors=5), "KNeighborsClassifier_DRK"),
    (RandomForestClassifier(n_estimators=100), "RandomForestClassifier_DRK"),
```

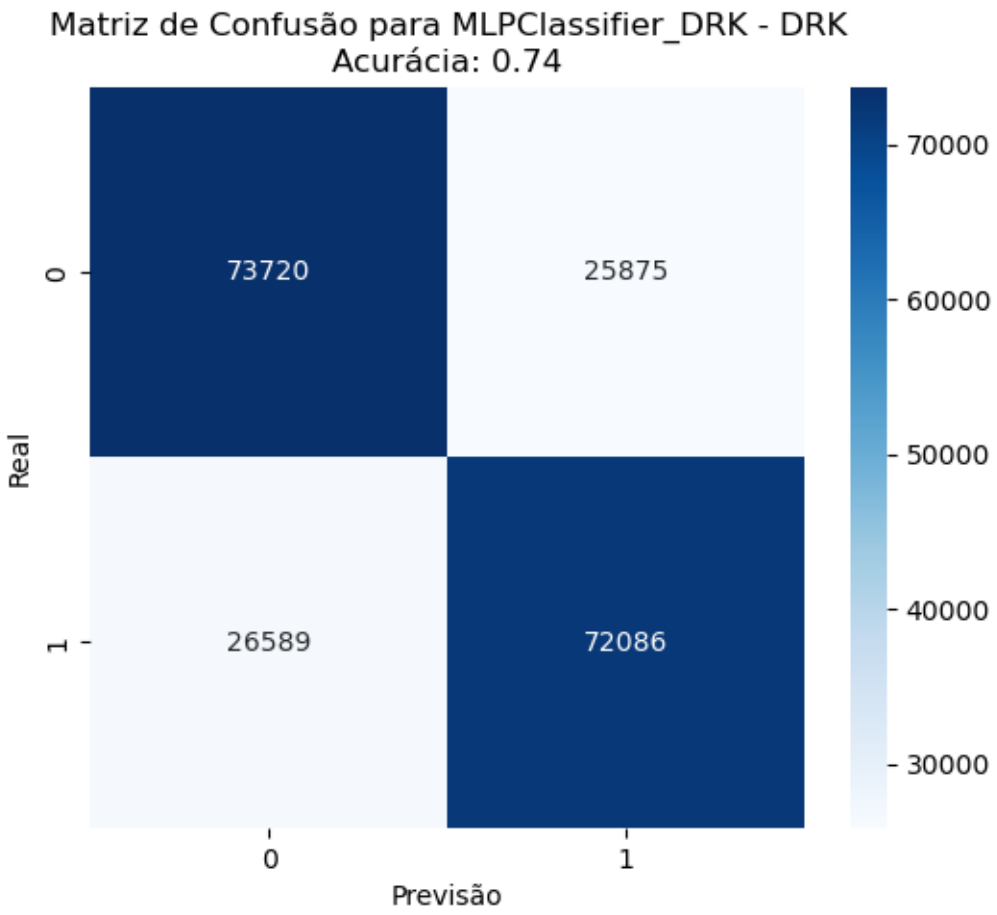
```
    (MLPClassifier(hidden_layer_sizes=(64, 16, 4), max_iter=1000),␣
 ↪"MLPClassifier_DRK"),
    #(SVC(), "SVC")
]

# Executando treinamento em paralelo
results = []
with ThreadPoolExecutor(max_workers=len(models)) as executor:
    future_to_model = {executor.submit(train_and_evaluate, model, X_train_drk,␣
 ↪y_train_drk, X_test_drk, y_test_drk, model_name): model_name for model,␣
 ↪model_name in models}
    for future in as_completed(future_to_model):
        results.append(future.result())
```

```
Iniciando treinamento do modelo KNeighborsClassifier_DRK para o dataset DRK
Iniciando treinamento do modelo RandomForestClassifier_DRK para o dataset DRK
Iniciando treinamento do modelo MLPClassifier_DRK para o dataset DRK
Treinamento concluído para o modelo KNeighborsClassifier_DRK no dataset DRK
Treinamento concluído para o modelo RandomForestClassifier_DRK no dataset DRK
Treinamento concluído para o modelo MLPClassifier_DRK no dataset DRK
```

[10]:
```
# Plotando as matrizes de confusão
for model_name, accuracy, cm in results:
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Matriz de Confusão para {model_name} - DRK\nAcurácia: {accuracy:
 ↪.2f}")
    plt.xlabel('Previsão')
    plt.ylabel('Real')
    plt.show()
```


Matriz de Confusão para KNeighborsClassifier_DRK - DRK
Acurácia: 0.68
66265
33330
30018
68657
Real
Previsão
0
1
0
1
65000
60000
55000
50000
45000
40000
35000


Matriz de Confusão para RandomForestClassifier_DRK - DRK
Acurácia: 0.73
72065
27530
26300
72375
Real
0
1
Previsão
0
1
70000
60000
50000
40000
30000


Matriz de Confusão para MLPClassifier_DRK - DRK
Acurácia: 0.74
73720
25875
26589
72086
Real
Previsão
0
1
70000
60000
50000
40000
30000

[ ]: