Big Data Mining & Inteligência Artificial

Charles Guimarães Cavalcante – RM 334409
Luan Nonato Figueiredo – RM 334325
Rodrigo Rossi de Lima Cano – RM 333927

Modelos de machine learning

Utilizando a base Titanic faça uma análise utilizando um dos modelos de machine learning aprendidos.

1) Dicionário de dados:

Variável	Definição	Valores
Survived	Sobreviveu	0 = Não, 1 = Sim
Pclass	Classe da passagem	1 = 1ª classe, 2 = 2ª classe, 3 = 3ª classe
Name	Nome	
Sex	Sexo	male = masculino, female = feminino
Age	Idade em anos	valores de 0,42 a 80
Siblings/Spouses Aboard	Número de irmãos/cônjuge abordo	valores de 0 a 8
Parents/Children Aboard	Número de pais/filhos abordo	valores de 0 a 6
Fare	Valor pago pela passagem	valore de 0 a 512,3292

2) Bibliotecas utilizadas:

```
import pandas as pd
import numpy as np
import seaborn as sns
import graphviz
import matplotlib.pyplot as plt
from sklearn import tree, metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

3) Leitura do arquivo:

```
df = pd.read_csv('titanic.csv')
df.head()
```

Output com a amostra dos dados:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500

4) Descrição dos dados:

```
df.describe()
```

Output com a descrição dos dados:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
count	887.000000	887.000000	887.000000	887.000000	887.000000	887.00000
mean	0.385569	2.305524	29.471443	0.525366	0.383315	32.30542
std	0.487004	0.836662	14.121908	1.104669	0.807466	49.78204
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	0.000000	2.000000	20.250000	0.000000	0.000000	7.92500
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.13750
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

```
df.info()
```

Output com informações sobre os dados:

```
RangeIndex: 887 entries, 0 to 886
Data columns (total 8 columns):
Survived                  887 non-null int64
Pclass                    887 non-null int64
Name                      887 non-null object
Sex                       887 non-null object
Age                       887 non-null float64
Siblings/Spouses Aboard   887 non-null int64
Parents/Children Aboard   887 non-null int64
Fare                      887 non-null float64
dtypes: float64(2), int64(4), object(2)
memory usage: 55.6+ KB
```

O data frame não contém nenhum dado nulo, não será necessário tratamento para ajustar dados nulos.

5) Preparação:

a) Substituição da variável “Sex” por 1 e 0.

```
df['Sex Category'] = df['Sex']
df['Sex Category'].replace(['female', 'male'], [0, 1], inplace=True)
df['Sex Category'].value_counts()
```

```
1    573
0    314
Name: Sex Category, dtype: int64
```

b) Categorização da variável “Age”, o desvio padrão é 14.12, arredondamos para 16 e separamos em categorias de 16 em 16.

```
df['Age'].std()
```

```
14.121908405462555
```

```
df['Age Category'] = pd.cut(df['Age'], bins=[0,16,32,48,64,80], labels=[0,1,2,3,4])
df['Age Category'].value_counts(sort=False)
```

```
0    114
1    453
2    230
3     77
4     13
Name: Age Category, dtype: int64
```

c) Categorização da variável “Fare”, separamos pelos quartis, 7.92, 14.45 e 31.13.

```
df['Fare'].quantile([0.25, 0.5, 0.75])
```

```
0.25     7.9250
0.50    14.4542
0.75    31.1375
Name: Fare, dtype: float64
```

```
df['Fare Category'] = pd.cut(df['Fare'],bins=[-np.inf,7.92,14.45,31.13,np.inf],
labels=[0,1,2,3])
df['Fare Category'].value_counts(sort=False)
```

```
0    220
1    217
2    228
3    222
Name: Fare Category, dtype: int64
```

d) Criação da variável “Family”, somando as variáveis “Siblings/Spouses Aboard” e “Parents/Children Aboard”.

```
df['Family'] = df['Siblings/Spouses Aboard'] + df['Parents/Children Aboard']
df['Family'].value_counts(sort=False)
```

```
0      533
1      161
2      102
3       29
4       15
5       22
6       12
7        6
10       7
Name: Family, dtype: int64
```

e) Categorização da variável “Family”, utilizando 1 para qualquer quantidade e 0 para nenhum.

```
df['Family Category'] = df['Family'].apply(lambda x: 1 if x > 0 else 0)
df['Family Category'].value_counts()
```

```
0    533
1    354
Name: Family Category, dtype: int64
```

f) Separação da base com os dados que serão utilizados.

```
data =df[['Pclass','Sex Category','Age Category','Fare Category','Family Category']]
data.columns = ['Pclass', 'Sex', 'Age', 'Fare', 'Family']
data
```

	Pclass	Sex	Age	Fare	Family
0	3	1	1	0	1
1	1	0	2	3	1
2	3	0	1	1	0
3	1	0	2	3	1
4	3	1	2	1	0
...	...	...	...	...	...
882	2	1	1	1	0
883	1	0	1	2	0
884	3	0	0	2	1
885	1	1	1	2	0
886	3	1	1	0	0

887 rows × 5 columns

g) Variável target.

```
target = df['Survived']
target
```

```
0      0
1      1
2      1
3      1
4      0
      ..
882    0
883    1
884    0
885    1
886    0
Name: Survived, Length: 887, dtype: int64
```

h) Separação da base em treino e teste utilizando o **train_test_split**, foi utilizado o *seed* (random_state=41) para que o script retorne os mesmos resultados quando reproduzido:

```
train_data, test_data, train_target, test_target = train_test_split(data, target, te
st_size=0.2, random_state=41)
print('base de treino: ', len(train_data))
print('base de teste: ', len(test_data))
```

```
base de treino:  709
base de teste:  178
```

6) Árvore de Decisão:

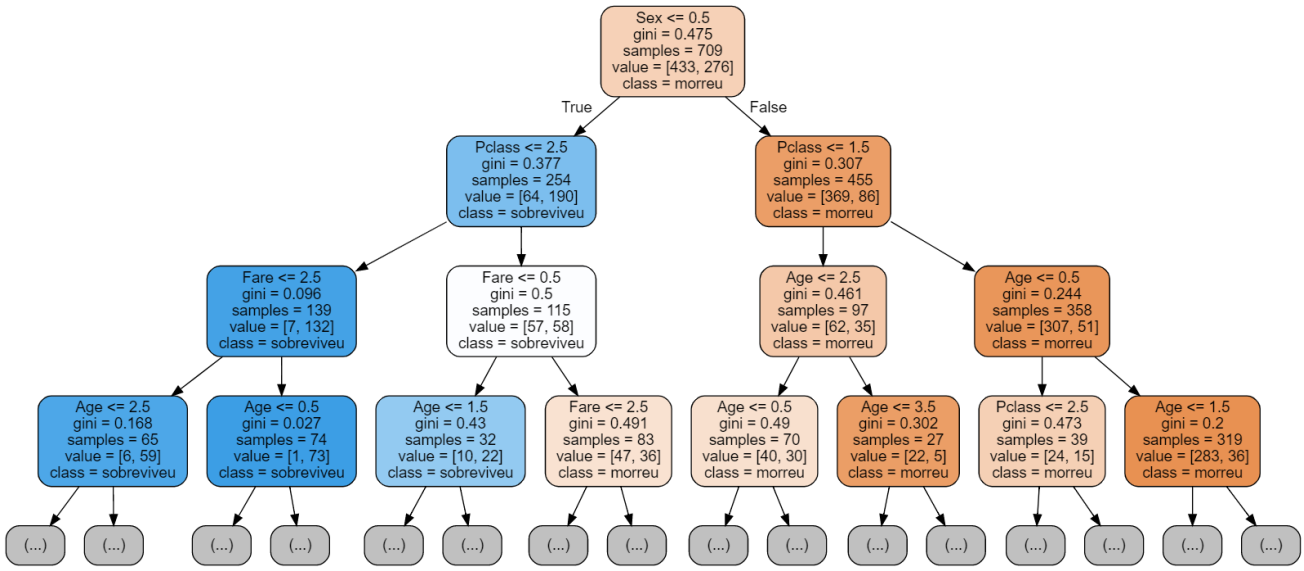
a) Criação da árvore de decisão com a base de treino:

```
dtc = tree.DecisionTreeClassifier()
dtc = dtc.fit(train_data, train_target)
```

b) Exibição do gráfico árvore de decisão, como a árvore ficou muito complexa, vamos limitar a 3 níveis para visualização:

```
graph_data = tree.export_graphviz(dtc, out_file=None,
                                  feature_names=data.columns,
                                  class_names=['morreu', 'sobreviveu'],
                                  filled=True, rounded=True, max_depth=3)

graph = graphviz.Source(graph_data)
graph
```



c) Predição:

```
predicted = dtc.predict(test_data)
predicted
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1])
```

d) Medindo a acurácia da predição:

```
acuracia = metrics.accuracy_score(test_target, predicted)
print("Acurácia: %.2f%%\n" % (acuracia*100))
```

```
Acurácia: 81.46%
```

e) Testando a árvore de decisão 1.000 vezes, com diferentes separações de base de treino e de teste. A média de acurácia da predição nos 1.000 testes foi de 80%, mesmo com tamanhos diferentes de base de teste (foram testadas com 10%, 20%, 30% e 40%), o resultado foi sempre em torno de 80%.

```
acuracias = []

for i in range(1000):
  train_data, test_data, train_target, test_target = train_test_split(data, target,
test_size=0.2)
  dtc.fit(train_data, train_target)
  predicted = dtc.predict(test_data)
  acuracias.append(metrics.accuracy_score(test_target, predicted))

print("Acurácia: %.2f%%\n" % (np.mean(acuracias)*100))
```

```
Acurácia: 80.48%
```

7) Random Forest:

a) Criação da árvore de decisão com a base de treino:

```
rfc = RandomForestClassifier(n_estimators=1000)
rfc.fit(train_data, train_target)
```

b) Predição:

```
predicted = rfc.predict(test_data)
predicted
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 1])
```

c) Medindo a acurácia da predição:

```
acuracia = metrics.accuracy_score(test_target, predicted)
print("Acurácia: %.2f%%\n" % (acuracia*100))
```

```
Acurácia: 80.90%
```

8) KNN - K-Nearest Neighbors:

a) Criação do modelo KNN com a base de treino:

```
knn = KNeighborsClassifier()
knn.fit(train_data, train_target)
```

b) Predição:

```
predicted = knn.predict(test_data)
predicted
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 1])
```

c) Medindo a acurácia da predição:

```
acuracia = metrics.accuracy_score(test_target, predicted)
print("Acurácia: %.2f%%\n" % (acuracia*100))
```

```
Acurácia: 83.15%
```

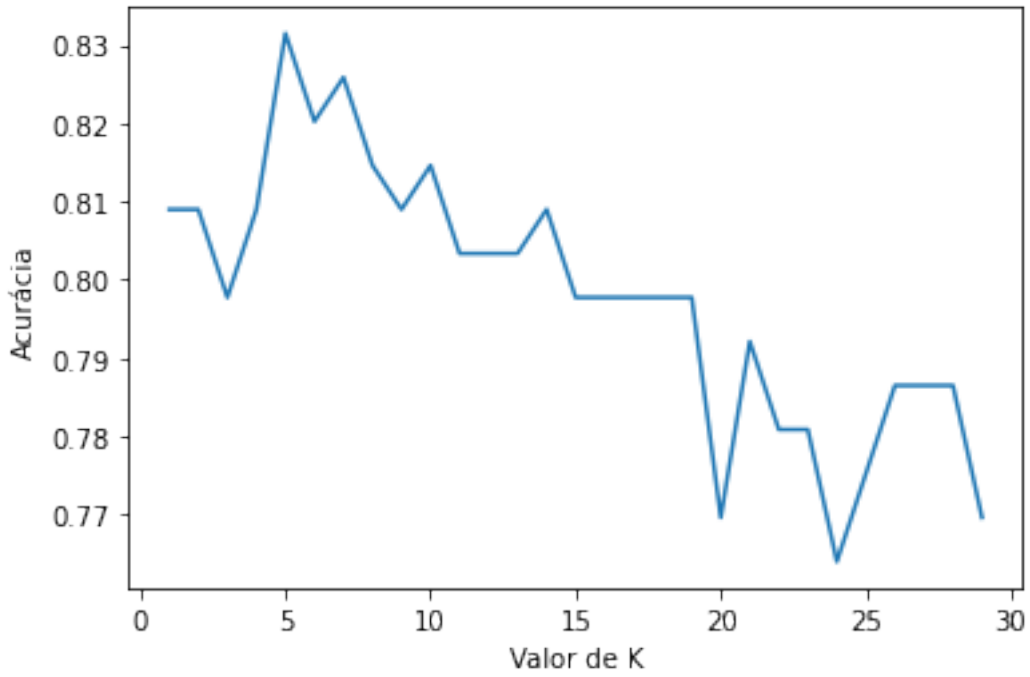
d) Testando o KNN com diferentes valores para k, foram testados de 1 a 30:

```
kscores = range(1, 30)
scores = []

for k in kscores:
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(train_data, train_target)
  predicted = knn.predict(test_data)
  score = metrics.accuracy_score(test_target, predicted)
  scores.append(score)
```

e) Exibindo o resultado em um gráfico:

```
plt.plot(kscores, scores)
plt.xlabel("Valor de K")
plt.ylabel("Acurácia")
```



O melhor resultado foi com k=5, que é o valor padrão da função `KNeighborsClassifier`, portanto a melhor acurácia de 83.15% foi com k=5.

f) Testando a KNN 1.000 vezes, com diferentes separações de base de treino e de teste. A média de acurácia da predição nos 1.000 testes foi de 79,8%.

```
acuracias = []

knn = KNeighborsClassifier(n_neighbors=5)

for i in range(1000):
  train_data, test_data, train_target, test_target = train_test_split(data, target,
test_size=0.2)
  knn.fit(train_data, train_target)
  predicted = knn.predict(test_data)
  acuracias.append(metrics.accuracy_score(test_target, predicted))

print("Acurácia: %.2f%%\n" % (np.mean(acuracias)*100))
```

```
Acurácia: 79.86%
```

9) Conclusão:

Foram testados dois modelos de *machine learning*: **Árvore de Decisão** (*Decision Tree*) e **KNN** (**K-Nearest Neighbors**).

Para os dois modelos foram utilizadas as mesmas bases de treino e de teste. Também foram realizados testes com bases randômicas 1.000 vezes. Também foi utilizado o Random Forest para testar 1.000 árvores de decisão.

Os resultados variaram pouco nos dois modelos, a árvore de decisão teve acurácia de 81,46% e o KNN 83,15%. Porém nos testes randômicos os resultados foram 80% e 79,8% respectivamente.

Ambos os modelos tiveram bom desempenho e poderiam ser utilizados para predição.