Solução Lista 02

Nome: Julia Xavier
E-mail: julia.xavier@aluno.ufabc.edu.br
Nome: Leonardo Bernardes Lerio
E-mail: leonardo.lerio@aluno.ufabc.edu.br

21 outubro, 2023

Exercício 01

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
```

semente para os resultados serem sempre os mesmos

```
np.random.seed(1234)
```

treino

```
x_train = np.random.uniform(-1, 1, size=(50,))
eq_train = x_train**2 - 2 * x_train**3 + 1
y_train = eq_train + np.random.normal(0, 0.5, size=(50,))
```

teste

```
x_test = np.random.uniform(-1, 1, size=(100,))
eq_test = x_test**2 - 2 * x_test**3 + 1
y_test = eq_test + np.random.normal(0, 0.5, size=(100,))
```

```
train_data = pd.DataFrame({"x": x_train, "y": y_train})
test_data = pd.DataFrame({"x": x_test, "y": y_test})
```

extrai variaveis explicativas e resposta

```
x_train_rl = train_data[["x"]]
y_train_rl = train_data["y"]
```

gera modelo de regressão linear

```python
rl = LinearRegression()
rl.fit(x_train_rl, y_train_rl)
```

```
## LinearRegression()
```

```python
y_pred_rl = rl.predict(test_data[['x']])
```

calcula o erro médio quadrado da regressão linear

```python
erro_rl = mean_squared_error(y_test, y_pred_rl)
```

cria o modelo kNN com k = 1

```python
knn_1 = KNeighborsRegressor(n_neighbors=1)
knn_1.fit(x_train_rl, y_train_rl)
```

```
## KNeighborsRegressor(n_neighbors=1)
```

```python
y_pred_knn_1 = knn_1.predict(test_data[['x']])
```

calcula o erro médio quadrado do kNN = 1

```python
erro_knn_1 = mean_squared_error(y_test, y_pred_knn_1)
```

cria o modelo kNN com k = 10

```python
knn_10 = KNeighborsRegressor(n_neighbors=10)
knn_10.fit(x_train_rl, y_train_rl)
```

```
## KNeighborsRegressor(n_neighbors=10)
```

```python
y_pred_knn_10 = knn_10.predict(test_data[['x']])
```

calcula o erro médio quadrado com kNN = 10

```python
erro_knn_10 = mean_squared_error(y_test, y_pred_knn_10)
```

gera os valores dos erros

```python
print("Erro do modelo Regressão Linear: ", erro_rl)
```

```
## Erro do modelo Regressão Linear:  0.4319639183452123
```

```python
print("Erro do modelo kNN com k=1: ", erro_knn_1)
```

```
## Erro do modelo kNN com k=1:  0.4354264975858965
```

```
print("Erro do modelo kNN com k=10: ", erro_knn_10)
```

```
## Erro do modelo kNN com k=10:  0.3509510141218841
```

resultado final

```
if erro_rl < erro_knn_1 and erro_rl < erro_knn_10:
    print("A rl tem menor erro, tem baixo viés, mas pode ter alta variância")
elif erro_knn_1 < erro_rl and erro_knn_1 < erro_knn_10:
    print("O kNN com k=1 tem o menor erro, tem baixo viés, mas pode ter alta variância")
else:
    print("O kNN com k=10 tem o menor erro, tem baixo viés, mas pode ter alta variância")
```

```
## O kNN com k=10 tem o menor erro, tem baixo viés, mas pode ter alta variância
```

Exercício 02

```
import pandas as pd
import seaborn as sns
```

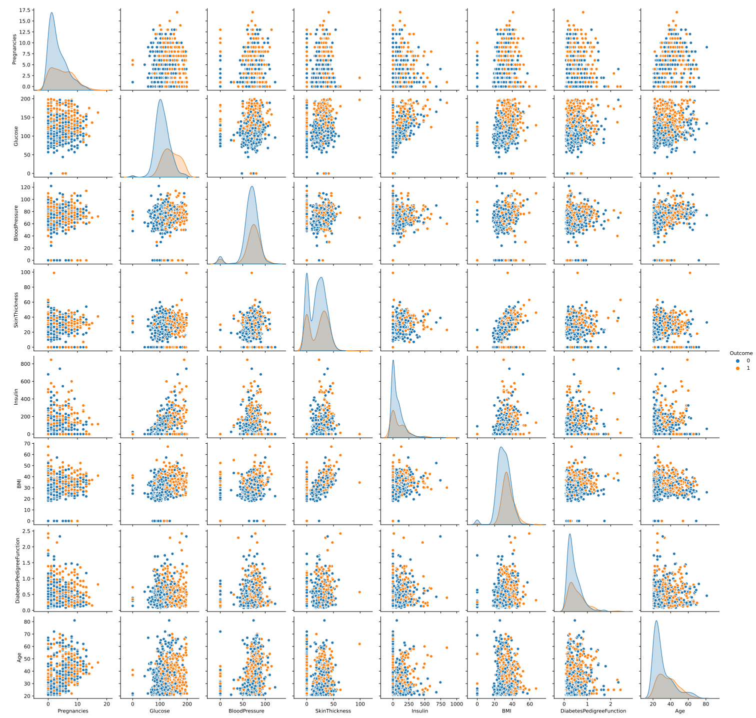
```
data = pd.read_csv('diabetes.csv')
```

```
data.head()
```

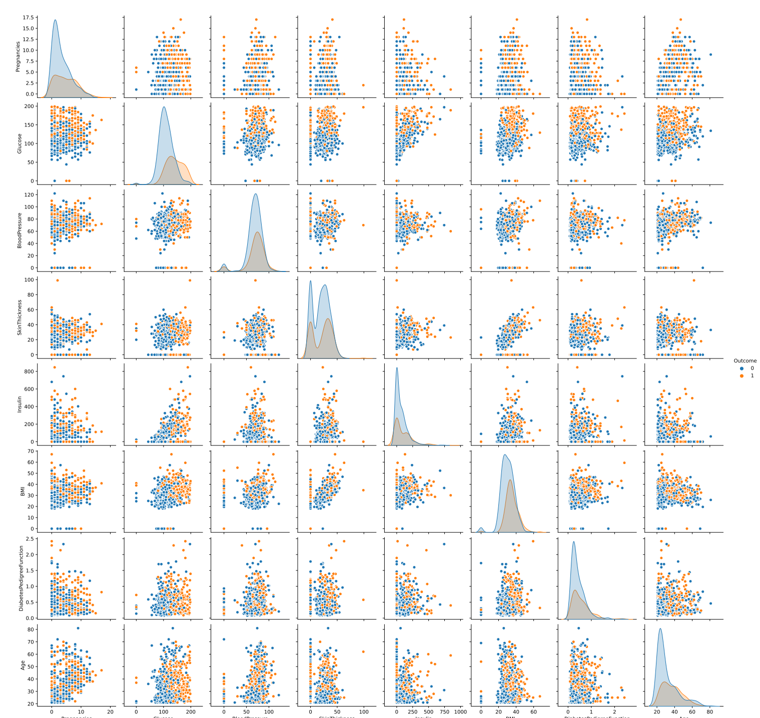
```
##    Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
## 0            6      148             72  ...                     0.627   50        1
## 1            1       85             66  ...                     0.351   31        0
## 2            8      183             64  ...                     0.672   32        1
## 3            1       89             66  ...                     0.167   21        0
## 4            0      137             40  ...                     2.288   33        1
##
## [5 rows x 9 columns]
```

```
resposta = 'Outcome'
```

```
sns.pairplot(data, hue = resposta)
```




```python
sns.pairplot(data, hue = resposta)
```



Exercício 03

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

Leitura dos dados

```
data = pd.read_csv('diabetes.csv')
data.head()
```

```
##    Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
## 0            6      148             72  ...                     0.627   50        1
## 1            1       85             66  ...                     0.351   31        0
## 2            8      183             64  ...                     0.672   32        1
## 3            1       89             66  ...                     0.167   21        0
## 4            0      137             40  ...                     2.288   33        1
##
## [5 rows x 9 columns]
```

Separação dos dados e dos resultados (X e Y)

```
X = data[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunc
Y = data['Outcome']
```

Separação para treinamento e teste (0.2 de teste e 0.8 de treinamento)

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Classificação por KNN usando numeros de vizinhos de 1 a 49

```
max = 0
max_n = 0
for x in range(1,50):
  knn = KNeighborsClassifier(n_neighbors=x)
  knn.fit(X_train, y_train)
  y_pred = knn.predict(X_test)
  accuracy = accuracy_score(y_test, y_pred)
  print("Accuracy:", accuracy)
  if accuracy > max:
    max = accuracy
    max_n = x
```

```
## KNeighborsClassifier(n_neighbors=1)
## Accuracy: 0.6753246753246753
## KNeighborsClassifier(n_neighbors=2)
## Accuracy: 0.7012987012987013
## KNeighborsClassifier(n_neighbors=3)
## Accuracy: 0.6493506493506493
## KNeighborsClassifier(n_neighbors=4)
## Accuracy: 0.7077922077922078
## KNeighborsClassifier()
```

```
## Accuracy: 0.6623376623376623
## KNeighborsClassifier(n_neighbors=6)
## Accuracy: 0.7272727272727273
## KNeighborsClassifier(n_neighbors=7)
## Accuracy: 0.6883116883116883
## KNeighborsClassifier(n_neighbors=8)
## Accuracy: 0.7467532467532467
## KNeighborsClassifier(n_neighbors=9)
## Accuracy: 0.7207792207792207
## KNeighborsClassifier(n_neighbors=10)
## Accuracy: 0.7662337662337663
## KNeighborsClassifier(n_neighbors=11)
## Accuracy: 0.7337662337662337
## KNeighborsClassifier(n_neighbors=12)
## Accuracy: 0.7792207792207793
## KNeighborsClassifier(n_neighbors=13)
## Accuracy: 0.7727272727272727
## KNeighborsClassifier(n_neighbors=14)
## Accuracy: 0.7727272727272727
## KNeighborsClassifier(n_neighbors=15)
## Accuracy: 0.7597402597402597
## KNeighborsClassifier(n_neighbors=16)
## Accuracy: 0.7792207792207793
## KNeighborsClassifier(n_neighbors=17)
## Accuracy: 0.7727272727272727
## KNeighborsClassifier(n_neighbors=18)
## Accuracy: 0.7597402597402597
## KNeighborsClassifier(n_neighbors=19)
## Accuracy: 0.7532467532467533
## KNeighborsClassifier(n_neighbors=20)
## Accuracy: 0.7272727272727273
## KNeighborsClassifier(n_neighbors=21)
## Accuracy: 0.7402597402597403
## KNeighborsClassifier(n_neighbors=22)
## Accuracy: 0.7402597402597403
## KNeighborsClassifier(n_neighbors=23)
## Accuracy: 0.7142857142857143
## KNeighborsClassifier(n_neighbors=24)
## Accuracy: 0.7467532467532467
## KNeighborsClassifier(n_neighbors=25)
## Accuracy: 0.7142857142857143
## KNeighborsClassifier(n_neighbors=26)
## Accuracy: 0.7402597402597403
## KNeighborsClassifier(n_neighbors=27)
## Accuracy: 0.7272727272727273
## KNeighborsClassifier(n_neighbors=28)
## Accuracy: 0.7337662337662337
## KNeighborsClassifier(n_neighbors=29)
## Accuracy: 0.7142857142857143
## KNeighborsClassifier(n_neighbors=30)
## Accuracy: 0.7077922077922078
## KNeighborsClassifier(n_neighbors=31)
## Accuracy: 0.7207792207792207
## KNeighborsClassifier(n_neighbors=32)
```

```
## Accuracy: 0.7402597402597403
## KNeighborsClassifier(n_neighbors=33)
## Accuracy: 0.7467532467532467
## KNeighborsClassifier(n_neighbors=34)
## Accuracy: 0.7597402597402597
## KNeighborsClassifier(n_neighbors=35)
## Accuracy: 0.7532467532467533
## KNeighborsClassifier(n_neighbors=36)
## Accuracy: 0.7467532467532467
## KNeighborsClassifier(n_neighbors=37)
## Accuracy: 0.7402597402597403
## KNeighborsClassifier(n_neighbors=38)
## Accuracy: 0.7272727272727273
## KNeighborsClassifier(n_neighbors=39)
## Accuracy: 0.7207792207792207
## KNeighborsClassifier(n_neighbors=40)
## Accuracy: 0.7402597402597403
## KNeighborsClassifier(n_neighbors=41)
## Accuracy: 0.7337662337662337
## KNeighborsClassifier(n_neighbors=42)
## Accuracy: 0.7337662337662337
## KNeighborsClassifier(n_neighbors=43)
## Accuracy: 0.7207792207792207
## KNeighborsClassifier(n_neighbors=44)
## Accuracy: 0.7272727272727273
## KNeighborsClassifier(n_neighbors=45)
## Accuracy: 0.7337662337662337
## KNeighborsClassifier(n_neighbors=46)
## Accuracy: 0.7272727272727273
## KNeighborsClassifier(n_neighbors=47)
## Accuracy: 0.7337662337662337
## KNeighborsClassifier(n_neighbors=48)
## Accuracy: 0.7272727272727273
## KNeighborsClassifier(n_neighbors=49)
## Accuracy: 0.7272727272727273
```

Resultado do melhor numero de vizinhos

```
print("Max Accuracy:", max)
```

```
## Max Accuracy: 0.7792207792207793
```

```
print("Melhor número de vizinhos:", max_n)
```

```
## Melhor número de vizinhos: 12
```

Exercício 04

```
import pandas as pd
from sklearn.model_selection import cross_val_score, KFold
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from scipy.stats import sem
import matplotlib.pyplot as plt
import numpy as np
```

Leitura dos dados

```
data = pd.read_csv('diabetes.csv')
data.head()
```

```
##    Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
## 0            6      148             72  ...                     0.627   50        1
## 1            1       85             66  ...                     0.351   31        0
## 2            8      183             64  ...                     0.672   32        1
## 3            1       89             66  ...                     0.167   21        0
## 4            0      137             40  ...                     2.288   33        1
##
## [5 rows x 9 columns]
```

Separação X e Y

```
X = data[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunc
Y = data['Outcome']
```

Classficação com validação cruzada de 10 folds e de k entre 1 e 50

```
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

errors = []
k_values = np.arange(1, 50, 2)

min = 999
best_k = 0
for k in k_values:
  knn = KNeighborsClassifier(n_neighbors=k)
  scores = cross_val_score(knn, X, Y, cv=kfold)
  standard_error = sem(scores)
  print("Standard Error:", standard_error)
  errors.append(standard_error)
  if standard_error < min:
    min = standard_error
    best_k = k
```

```
## Standard Error: 0.020458782559503078
## Standard Error: 0.015497978233600128
## Standard Error: 0.017580527617296643
## Standard Error: 0.01290783751233132
## Standard Error: 0.019188891067122595
## Standard Error: 0.01649202776541329
## Standard Error: 0.015502499542873133
```

```
## Standard Error: 0.012570724543021232
## Standard Error: 0.014971031491597663
## Standard Error: 0.013669785237766272
## Standard Error: 0.015156931862807223
## Standard Error: 0.016630184423659048
## Standard Error: 0.013072254064019278
## Standard Error: 0.01411425786132047
## Standard Error: 0.016430071665733514
## Standard Error: 0.01806772821577492
## Standard Error: 0.016802395984795812
## Standard Error: 0.01459404844033075
## Standard Error: 0.01598854351338908
## Standard Error: 0.017043459380689647
## Standard Error: 0.01648609326340709
## Standard Error: 0.013173193691165241
## Standard Error: 0.011886127478262341
## Standard Error: 0.01150236581611677
## Standard Error: 0.014183386540867282
```

Melhor K

```
print("Min error:", min)
```

```
## Min error: 0.01150236581611677
```

```
print("Melhor número de vizinhos:", best_k)
```

```
## Melhor número de vizinhos: 47
```

Tabela

```
plt.plot(1 / k_values, errors, marker='o')
plt.xlabel('1/k')
plt.ylabel('Erro de Classificação')
plt.title('Erro vs 1/k')
plt.grid(True)
plt.show()
```