Aprendizagem de Máquina: Atividade 07 − Pré-Processamento de Dados

Carlos Emmanuel Pereira Alves
Curso de Bacharelado em Ciência da Computação
Universidade Federal do Agreste de Pernambuco (UFAPE)
Garanhuns, Brasil
carlos.emmanuel.236@gmail.com

1) Converta os atributos numéricos para atributos categóricos da base Iris. archive.ics.uci.edu/ml/datasets/Iris
    a) Explique qual a conversão mais adequada para cada atributo.
    Como os atributos são numéricos vamos converter para categóricos, cada atributo vai ter 3 características possíveis e cada característica vai corresponder a um intervalo.
    Sepal Length e Petal Length: Short, Medium, Long.
    Sepal Width e Petal Width: Narrow, Medium, Wide.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer

cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

data = pd.read_csv('iris.data', header=None, names=cols)

est = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')

datat = pd.DataFrame(est.fit_transform(data[cols[:-1]]), columns=cols[:-1])
datat['class'] = data['class']

datat.head(n=-1)
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	0.0	1.0	0.0	0.0	Iris-setosa
1	0.0	1.0	0.0	0.0	Iris-setosa
2	0.0	1.0	0.0	0.0	Iris-setosa
3	0.0	1.0	0.0	0.0	Iris-setosa
4	0.0	1.0	0.0	0.0	Iris-setosa
...	...	...	...	...	...
144	2.0	1.0	2.0	2.0	Iris-virginica
145	2.0	1.0	2.0	2.0	Iris-virginica
146	1.0	0.0	2.0	2.0	Iris-virginica
147	1.0	1.0	2.0	2.0	Iris-virginica
148	1.0	1.0	2.0	2.0	Iris-virginica

149 rows × 5 columns

b) Realize um experimento de 100 repetições de Holdout 50/50 estratificado para calcular a taxa de acerto na base transformada utilizando o classificador 1-NN com distância de Hamming. Calcule o intervalo de confiança da acurácia.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

data = pd.read_csv('iris.data', header=None, names=cols)

est = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')

datat = pd.DataFrame(est.fit_transform(data[cols[:-1]]), columns=cols[:-1])
datat['class'] = data['class']

scores = []

for i in range(100):
  X_train, X_test, y_train, y_test = train_test_split(datat[cols[:-1]], datat['class'], stratify=datat['class'], test_size=0.5)

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="hamming")
  knn.fit(X_train, y_train)
  score = knn.score(X_test, y_test)
  scores = np.append(scores, score)

med = np.average(scores)
dev = scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print("Média:", med)
print("Desvio padrão:", dev)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.9554666666666667
Desvio padrão: 0.02248713212286332
Intervalo de confiança: 0.91139   1.0
```

c) Realize um teste de hipótese de sobreposição do intervalo de confiança comparando o resultado da letra anterior com a acurácia do 1-NN com distância Euclidiana na base Iris original.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

data = pd.read_csv('iris.data', header=None, names=cols)

scores = []

for i in range(100):
  X_train, X_test, y_train, y_test = train_test_split(data[cols[:-1]], data['class'], stratify=data['class'], test_size=0.5)

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train, y_train)
  score = knn.score(X_test, y_test)
  scores = np.append(scores, score)

med = np.average(scores)
dev = scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print("Média:", med)
print("Desvio padrão:", dev)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.9525333333333333
Desvio padrão: 0.014885638119416388
Intervalo de confiança: 0.92336   0.98171
```

Existe uma sobreposição de intervalos, então não podemos rejeitar o H_0, e não podemos afirmar se os classificadores possuem ou não taxas de acerto diferentes.

2) A base Heart Disease (hungarian) possui alguns valores de atributos omissos. Realize o experimento descrito abaixo utilizando o classificador 1-NN. Atenção: considere remover as colunas que apresentam um número muito grande de valores omissos (mais de 50% de valores omissos).
Primeiro verifiquei quais colunas apresentam um número muito grande de valores omissos e as retirei.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']

data = pd.read_csv('processed.hungarian.data', header=None, names=cols, sep=',')
data = data.replace('?', np.nan)

missing_values = data.isnull().sum()
total_rows = len(data)
missing_percentage = (missing_values / total_rows) * 100

for column in data.columns:
    print(f"Atributo: {column}")
    print(f"  Valores faltando: {missing_values[column]}")
    print(f"  Porcentagem de valores faltando: {missing_percentage[column]:.2f}%")
    print(f"  Mais de 50% faltando?: {missing_percentage[column] > 50}\n")
```

```
Atributo: age
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: sex
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: cp
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: trestbps
  Valores faltando: 1
  Porcentagem de valores faltando: 0.34%
  Mais de 50% faltando?: False

Atributo: chol
  Valores faltando: 23
  Porcentagem de valores faltando: 7.82%
  Mais de 50% faltando?: False

Atributo: fbs
  Valores faltando: 8
  Porcentagem de valores faltando: 2.72%
  Mais de 50% faltando?: False

Atributo: restecg
  Valores faltando: 1
  Porcentagem de valores faltando: 0.34%
  Mais de 50% faltando?: False
```

```
Atributo: thalach
  Valores faltando: 1
  Porcentagem de valores faltando: 0.34%
  Mais de 50% faltando?: False

Atributo: exang
  Valores faltando: 1
  Porcentagem de valores faltando: 0.34%
  Mais de 50% faltando?: False

Atributo: oldpeak
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: slope
  Valores faltando: 190
  Porcentagem de valores faltando: 64.63%
  Mais de 50% faltando?: True

Atributo: ca
  Valores faltando: 291
  Porcentagem de valores faltando: 98.98%
  Mais de 50% faltando?: True

Atributo: thal
  Valores faltando: 266
  Porcentagem de valores faltando: 90.48%
  Mais de 50% faltando?: True

Atributo: num
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False
```

Então, retirei as colunas slope, ca e thal.

a) Divida a base em treino (90%) e teste (10%) de forma estratificada. Determine como preencher os valores omissos dos atributos utilizando apenas o conjunto de treinamento.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']

data = pd.read_csv('processed.hungarian.data', header=None, names=cols, sep=',')
data = data.replace('?', np.nan)
data = data.drop(columns=['slope', 'ca', 'thal'])

cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'num']

X_train, X_test, y_train, y_test = train_test_split(data[cols[:-1]], data['num'], stratify=data['num'], test_size=0.1)
```

Vou utilizar o SimpleImputer para substituir os valores omissos, com a média.

b) Preencha os valores omissos no conjunto de treino.

```
imputer = SimpleImputer(strategy='mean', missing_values=np.nan)
imputer = imputer.fit(X_train)
X_train.iloc[:,:] = imputer.transform(X_train)

missing_values = X_train.isnull().sum()
print(f"Valores nulos no conjunto de treinamento:\n{missing_values}")
```

```
Valores nulos no conjunto de treinamento:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
dtype: int64
```

c) Preencha os valores omissos no conjunto de teste utilizando o método e os valores definidos para o conjunto de treino.

```
imputer2 = SimpleImputer(strategy='mean', missing_values=np.nan)
imputer2 = imputer2.fit(X_test)
X_test.iloc[:,:] = imputer2.transform(X_test)

missing_values = X_test.isnull().sum()
print(f"Valores nulos no conjunto de teste:\n{missing_values}")
```

```
Valores nulos no conjunto de teste:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
dtype: int64
```

d) Repita 30 vezes este experimento para calcular o intervalo de confiança para a taxa de acerto do classificador utilizando 100 repetições deste experimento.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import SimpleImputer

cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']

data = pd.read_csv('processed.hungarian.data', header=None, names=cols, sep=',')
data = data.replace('?', np.nan)
data = data.drop(columns=['slope', 'ca', 'thal'])

cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'num']

scores = []

for i in range(100):
  X_train, X_test, y_train, y_test = train_test_split(data[cols[:-1]], data['num'], stratify=data['num'], test_size=0.1)

  imputer = SimpleImputer(strategy='mean', missing_values=np.nan)
  X_train_transformed_df = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
  X_train = X_train_transformed_df

  imputer2 = SimpleImputer(strategy='mean', missing_values=np.nan)
  X_test_transformed_df = pd.DataFrame(imputer.fit_transform(X_test), columns=X_test.columns)
  X_test = X_test_transformed_df

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train, y_train)
  score = knn.score(X_test, y_test)
  scores = np.append(scores, score)

med = np.average(scores)
dev = scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print("Média:", med)
print("Desvio padrão:", dev)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.632
Desvio padrão: 0.08729515705034525
Intervalo de confiança: 0.4609   0.8031
```

e) Realize um teste de hipótese comparando o intervalo calculado anteriormente com o um intervalo de confiança calculado de forma similar removendo todas as colunas que apresentavam valores omissos.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import SimpleImputer

cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']

data = pd.read_csv('processed.hungarian.data', header=None, names=cols, sep=',')
data = data.replace('?', np.nan)
data = data.drop(columns=['trestbps', 'chol', 'fbs', 'thalach', 'exang', 'restecg', 'slope', 'ca', 'thal'])

cols = ['age', 'sex', 'cp', 'oldpeak', 'num']

scores = []

for i in range(100):
  X_train, X_test, y_train, y_test = train_test_split(data[cols[:-1]], data['num'], stratify=data['num'], test_size=0.1)

  imputer = SimpleImputer(strategy='mean', missing_values=np.nan)
  X_train_transformed_df = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
  X_train = X_train_transformed_df

  imputer2 = SimpleImputer(strategy='mean', missing_values=np.nan)
  X_test_transformed_df = pd.DataFrame(imputer.fit_transform(X_test), columns=X_test.columns)
  X_test = X_test_transformed_df

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train, y_train)
  score = knn.score(X_test, y_test)
  scores = np.append(scores, score)

med = np.average(scores)
dev = scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print("Média:", med)
print("Desvio padrão:", dev)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.7440000000000003
Desvio padrão: 0.07451770855903242
Intervalo de confiança: 0.59795   0.89005
```

Vemos que há sobreposição de intervalos, portanto não podemos rejeitar o H_0, e não podemos afirmar nada sobre a taxa de acerto dos classificadores.

3) Faça o mesmo da questão anterior para a base Adult. Se optar converter os atributos categóricos para numéricos, para cada valor de atributo categórico crie um novo atributo binário (substitua a coluna antiga por novas colunas). Nenhum atributo apresentou mais de 50% de omissão de valores, por isso não removi nenhuma coluna.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

cols = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'num']

data = pd.read_csv('adult.data', header=None, names=cols, sep=',')
data = data.applymap(lambda x: x.strip() if isinstance(x, str) else x)
data = data.replace('?', np.nan)

missing_values = data.isnull().sum()
total_rows = len(data)
missing_percentage = (missing_values / total_rows) * 100

for column in data.columns:
    print(f"Atributo: {column}")
    print(f"  Valores faltando: {missing_values[column]}")
    print(f"  Porcentagem de valores faltando: {missing_percentage[column]:.2f}%")
    print(f"  Mais de 50% faltando?: {missing_percentage[column] > 50}\n")
```

```
Atributo: age
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: workclass
  Valores faltando: 1836
  Porcentagem de valores faltando: 5.64%
  Mais de 50% faltando?: False

Atributo: fnlwgt
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: education
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: education-num
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: marital-status
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: occupation
  Valores faltando: 1843
  Porcentagem de valores faltando: 5.66%
  Mais de 50% faltando?: False

Atributo: relationship
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: race
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: sex
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: capital-gain
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: capital-loss
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: hours-per-week
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False

Atributo: native-country
  Valores faltando: 583
  Porcentagem de valores faltando: 1.79%
  Mais de 50% faltando?: False

Atributo: num
  Valores faltando: 0
  Porcentagem de valores faltando: 0.00%
  Mais de 50% faltando?: False
```

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

cols = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race',
        'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'class']

data = pd.read_csv('adult.data', header=None, names=cols, sep=',')
data = data.applymap(lambda x: x.strip() if isinstance(x, str) else x)
data = data.replace('?', np.nan)

scores = []

for i in range(100):
  X_train, X_test, y_train, y_test = train_test_split(data[cols[:-1]], data['class'], stratify=data['class'], test_size=0.1)

  imputer = SimpleImputer(strategy='most_frequent', missing_values=np.nan)
  X_train_transformed_df = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
  X_train = X_train_transformed_df

  X_test_transformed_df = pd.DataFrame(imputer.fit_transform(X_test), columns=X_test.columns)
  X_test = X_test_transformed_df

  transformer = make_column_transformer(
      (OneHotEncoder(handle_unknown='ignore'),
       ['workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-country']),
      remainder='passthrough',
      verbose_feature_names_out=False)
  X_train_transformed = transformer.fit_transform(X_train)
  X_test_transformed = transformer.transform(X_test)

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train_transformed, y_train)
  score = knn.score(X_test_transformed, y_test)
  scores = np.append(scores, score)

med = np.average(scores)
dev = scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print("Média:", med)
print("Desvio padrão:", dev)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.7323508341009107
Desvio padrão: 0.007395532335479652
Intervalo de confiança: 0.71786   0.74685
```

Retirando todas as colunas com valores omissos:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

cols = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race',
        'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'class']

data = pd.read_csv('adult.data', header=None, names=cols, sep=',')
data = data.applymap(lambda x: x.strip() if isinstance(x, str) else x)
data = data.replace('?', np.nan)
data = data.drop(columns=['workclass', 'occupation', 'native-country'])

cols = ['age', 'fnlwgt', 'education', 'education-num', 'marital-status', 'relationship', 'race',
        'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'class']

scores = []

for i in range(100):
  X_train, X_test, y_train, y_test = train_test_split(data[cols[:-1]], data['class'], stratify=data['class'], test_size=0.1)

  transformer = make_column_transformer(
      (OneHotEncoder(handle_unknown='ignore'),
       ['education', 'marital-status', 'relationship', 'race', 'sex']),
      remainder='passthrough',
      verbose_feature_names_out=False)
  X_train_transformed = transformer.fit_transform(X_train)
  X_test_transformed = transformer.transform(X_test)

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train_transformed, y_train)
  score = knn.score(X_test_transformed, y_test)
  scores = np.append(scores, score)

med = np.average(scores)
dev = scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print("Média:", med)
print("Desvio padrão:", dev)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.7342646607307338
Desvio padrão: 0.00642996047644995
Intervalo de confiança: 0.72166   0.74687
```

Vemos que existe sobreposição de intervalos, portanto não podemos rejeitar o H_0, e não podemos afirmar nada sobre a taxa de acerto dos classificadores.

4) Utilizando a base de dados Wine https://archive.ics.uci.edu/ml/datasets/wine, para cada um dos casos abaixo, realize 100 repetições de Holdout 50/50 e calcule o intervalo de confiança da acurácia utilizando o classificador 1-NN com distância Euclidiana.
    a) Com todas as características ajustadas para o intervalo [0,1].

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

cols = ['class', 'alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids',
        'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'OD280/OD315_of_diluted_wines', 'proline']

data = pd.read_csv('wine.data', header=None, names=cols)

#ajustado para [0, 1]

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(data[cols[1:]])

data_scores = []

for i in range(100):
  data_train_x, data_test_x, data_train_y, data_test_y = train_test_split(scaled_data, data['class'], test_size=0.5)

  knn = KNeighborsClassifier(n_neighbors = 1, weights="distance", metric="euclidean")
  knn.fit(data_train_x, data_train_y)

  predict = knn.predict(data_test_x)
  data_score = knn.score(data_test_x, data_test_y)
  data_scores = np.append(data_scores, data_score)

med = np.average(data_scores)
dev = data_scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print(f'Intervalo de confiança da taxa de acerto ajustado para [0, 1]: {confidence_interval_n}   {confidence_interval_p}')
```

```
Intervalo de confiança da taxa de acerto ajustado para [0, 1]: 0.90509   0.98862
```

b) Com todas as características ajustadas para ter média zero e desvio padrão igual a um.

```python
#padronizado

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data[cols[1:]])

data_scores = []

for i in range(100):
  data_train_x, data_test_x, data_train_y, data_test_y = train_test_split(scaled_data, data['class'], test_size=0.5)

  knn = KNeighborsClassifier(n_neighbors = 1, weights="distance", metric="euclidean")
  knn.fit(data_train_x, data_train_y)

  predict = knn.predict(data_test_x)
  data_score = knn.score(data_test_x, data_test_y)
  data_scores = np.append(data_scores, data_score)

med = np.average(data_scores)
dev = data_scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print(f'Intervalo de confiança da taxa de acerto padronizado: {confidence_interval_n}   {confidence_interval_p}')
```

```
Intervalo de confiança da taxa de acerto padronizado: 0.90623   0.9922
```

5) Realize testes de hipótese por sobreposição dos intervalos de confiança comparando os pré-processamentos realizados na questão anterior com a base de dados original.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

cols = ['class', 'alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids',
        'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'OD280/OD315_of_diluted_wines', 'proline']

data = pd.read_csv('wine.data', header=None, names=cols)

#original
data_scores = []

for i in range(100):
  data_train_x, data_test_x, data_train_y, data_test_y = train_test_split(data[cols[1:]], data['class'], test_size=0.5)

  knn = KNeighborsClassifier(n_neighbors = 1, weights="distance", metric="euclidean")
  knn.fit(data_train_x, data_train_y)

  predict = knn.predict(data_test_x)
  data_score = knn.score(data_test_x, data_test_y)
  data_scores = np.append(data_scores, data_score)

med = np.average(data_scores)
dev = data_scores.std()

confidence_interval_n = round(med - (1.96 * dev), 5)
confidence_interval_p = round(med + (1.96 * dev), 5)

print(f'Intervalo de confiança da taxa de acerto: {confidence_interval_n}   {confidence_interval_p}')
```

```
Intervalo de confiança da taxa de acerto: 0.63361   0.7983
```

Comparando a base original com a ajustada, não existe sobreposição dos intervalos, por isso rejeita-se H_0, e concluímos que a base ajustada tem a maior acurácia.

Agora comparando a base original com a padronizada, vemos que também não existe sobreposição dos intervalos, então rejeitamos o H_0, e concluímos que a base padronizada tem a maior acurácia.