Aprendizagem de Máquina: Atividade 06 − 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) Nesta questão você deve utilizar a base Student Performance, archive.ics.uci.edu/ml/datasets/Student+Performance (ver arquivo student-mat.csv no student.zip). Assuma a última coluna (G3, que representa a nota final de cada estudante) como classe. Pode utilizar biblioteca.
    a) Explique qual a forma mais adequada para converter todos os atributos da base para numéricos (exceto a classe).
    Primeiro vamos separar os dados que já são numéricos e não precisam ser convertidos, que são: age, Medu, Fedu, traveltime, studytime, failures, famrel, freetime, goout, Dalc, Walc, health, absences, G1, G2.
    Agora com os atributos restantes vamos fazer a conversão para os que são binários e não-binários.
    Binários (nominais e ordinais): school, sex, address, famsize, Pstatus, schoolsup, famsup, paid, activities, nursery, higher, internet, romantic.
    Não binário (nominais): Mjob, Fjob, reason, guardian.

    b) Converta todos os atributos da base para numéricos.

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from google.colab.data_table import DataTable

DataTable.max_columns = 50
pd.set_option('display.max_columns', None)

cols = ['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
    'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 'schoolsup',
    'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel',
    'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences', 'G1', 'G2', 'G3']

binary = ['school', 'sex', 'address', 'famsize', 'Pstatus', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic']

data = pd.read_csv('student-mat.csv', header=0, names=cols, sep= ';')

for i in range(len(binary)):
  le = LabelEncoder()
  data[binary[i]] = le.fit_transform(data[binary[i]])

transformer = make_column_transformer((OneHotEncoder(), ['Mjob', 'Fjob', 'reason', 'guardian']), remainder='passthrough', verbose_feature_names_out=False)
transformed = transformer.fit_transform(data)
transformed_data = pd.DataFrame(transformed, columns=transformer.get_feature_names_out())

transformed_data.head(n=10)
```

	Mjob_at_home	Mjob_health	Mjob_other	Mjob_services	Mjob_teacher	Fjob_at_home	Fjob_health	Fjob_other	Fjob_services	Fjob_teacher	reason_course	reason_home	reason_other	reason_reputation	guardian_father	guardian_mother	guardian_other	school
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0
6	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
7	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
8	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
9	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0

n_mother	guardian_other	school	sex	age	address	famsize	Pstatus	Medu	Fedu	traveltime	studytime	failures	schoolsup	famsup	paid	activities	nursery	higher	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
1.0	0.0	0.0	0.0	18.0	1.0	0.0	0.0	4.0	4.0	2.0	2.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	4.0	3.0	4.0	1.0	1.0	3.0	6.0	5.0	6.0	6.0
0.0	0.0	0.0	0.0	17.0	1.0	0.0	1.0	1.0	1.0	1.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	5.0	3.0	3.0	1.0	1.0	3.0	4.0	5.0	5.0	6.0
1.0	0.0	0.0	0.0	15.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	4.0	3.0	2.0	2.0	3.0	3.0	10.0	7.0	8.0	10.0
1.0	0.0	0.0	0.0	15.0	1.0	0.0	1.0	4.0	2.0	1.0	3.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	2.0	2.0	1.0	1.0	5.0	2.0	15.0	14.0	15.0
0.0	0.0	0.0	0.0	16.0	1.0	0.0	1.0	3.0	3.0	1.0	2.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	4.0	3.0	2.0	1.0	2.0	5.0	4.0	6.0	10.0	10.0
1.0	0.0	0.0	1.0	16.0	1.0	1.0	1.0	4.0	3.0	1.0	2.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	5.0	4.0	2.0	1.0	2.0	5.0	10.0	15.0	15.0	15.0
1.0	0.0	0.0	1.0	16.0	1.0	1.0	1.0	2.0	2.0	1.0	2.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	4.0	4.0	4.0	1.0	1.0	3.0	0.0	12.0	12.0	11.0
1.0	0.0	0.0	0.0	17.0	1.0	0.0	0.0	4.0	4.0	2.0	2.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	4.0	1.0	4.0	1.0	1.0	1.0	6.0	6.0	5.0	6.0
1.0	0.0	0.0	1.0	15.0	1.0	1.0	0.0	3.0	2.0	1.0	2.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	4.0	2.0	2.0	1.0	1.0	1.0	0.0	16.0	18.0	19.0
1.0	0.0	0.0	1.0	15.0	1.0	0.0	1.0	3.0	4.0	1.0	2.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	5.0	5.0	1.0	1.0	1.0	5.0	0.0	14.0	15.0	15.0

c) Calcule o intervalo de confiança do RMSE para o 100 repetições de holdout 50/50 utilizando o classificador 1-NN com distância Euclidiana.

```
cols = [ 'Mjob_at_home',  'Mjob_health',  'Mjob_other',  'Mjob_services',  'Mjob_teacher',
         'Fjob_at_home',  'Fjob_health',  'Fjob_other',  'Fjob_services',  'Fjob_teacher',
         'reason_course',  'reason_home',  'reason_other',  'reason_reputation',
         'guardian_father',  'guardian_mother',  'guardian_other',
         'school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
          'traveltime', 'studytime', 'failures', 'schoolsup',
          'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel',
          'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences', 'G1', 'G2', 'G3']

rmse_list = []

for i in range(100):
  X_train, X_test, y_train, y_test = train_test_split(transformed_data[cols[:-1]], transformed_data['G3'], test_size=0.5)

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train, y_train)
  predict = knn.predict(X_test)

  rmse = mean_squared_error(y_test, predict, squared=False)
  rmse_list = np.append(rmse_list, rmse)

mean = np.mean(rmse_list)
dev = np.std(rmse_list)

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

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

```
Média do RMSE: 2.4295607474620673
Desvio padrão do RMSE: 0.2037829818323642
Intervalo de confiança do RMSE: 2.03015   2.82898
```

d) Faça o teste de hipótese comparando o intervalo de confiança da letra anterior com o intervalo de confiança pareado do RMSE utilizando apenas a características originalmente numéricas (removendo as características originalmente categóricas).

2) Utilize a base Student Performance (ver arquivo student-mat.csv no student.zip) archive.ics.uci.edu/ml/datasets/Student+Performance. Assuma a última coluna (G3, que representa a nota nal de cada estudante) como classe.

a) Converta a coluna da classe (atributo numérico) para uma variável categórica binária. Após esta conversão é possível realizar as tarefas a seguir.

Converti utilizando o critério de que 14 é a nota mínima exigida para ser aprovado então se G3 < 14; G3 = Failed; e se G3 > 13; G3 = Approved. Utilizei o código da questão 1 letra B, e adicionei o código abaixo.

```
transformed_data['G3'] = pd.cut(x=transformed_data['G3'],
                        bins=[-1, 13, 20],
                        labels=['Failed ', 'Approved'])

transformed_data.head()
```

G1	G2	G3
5.0	6.0	Failed
5.0	5.0	Failed
7.0	8.0	Failed
15.0	14.0	Approved
6.0	10.0	Failed
15.0	15.0	Approved
12.0	12.0	Failed
6.0	5.0	Failed
16.0	18.0	Approved
14.0	15.0	Approved

b) Calcule o intervalo de confiança da acurácia para o 100 repetições de holdout 50/50 utilizando o classificador 1-NN com distância Euclidiana.

```
cols = [ 'Mjob_at_home',  'Mjob_health',  'Mjob_other',  'Mjob_services',  'Mjob_teacher',
         'Fjob_at_home',  'Fjob_health',  'Fjob_other',  'Fjob_services',  'Fjob_teacher',
         'reason_course',  'reason_home',  'reason_other',  'reason_reputation',
         'guardian_father',  'guardian_mother',  'guardian_other',
         'school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
          'traveltime', 'studytime', 'failures', 'schoolsup',
          'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel',
          'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences', 'G1', 'G2', 'G3']

scores = []

for i in range(100):
  X_train, X_test, y_train, y_test = train_test_split(transformed_data[cols[:-1]], transformed_data['G3'], 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: 0.8968686868686868
Desvio padrão: 0.016768589484712344
Intervalo de confiança: 0.864   0.92974
```

3) Utilizando a base Forest Fires. archive.ics.uci.edu/ml/datasets/Forest+Fires
   a) Explique qual a forma mais adequada para converter todos os atributos da base para numéricos (exceto a classe).
      Os únicos atributos que precisam ser convertidos são: month e day. A forma mais adequada vai ser a utilização de dados cíclicos, por se tratar de dias da semana e do mês.

   b) Converta todos os atributos da base para numéricos.

```
import pandas as pd
import numpy as np
import math
from google.colab.data_table import DataTable

DAYS = {
  'sun': 1,
  'mon': 2,
  'tue': 3,
  'wed': 4,
  'thu': 5,
  'fri': 6,
  'sat': 7,
}

MONTHS = {
  'jan': 1,
  'feb': 2,
  'mar': 3,
  'apr': 4,
  'may': 5,
  'jun': 6,
  'jul': 7,
  'aug': 8,
  'sep': 9,
  'oct': 10,
  'nov': 11,
  'dec': 12,
}

cols = ['X', 'Y', 'month', 'day', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain', 'area']

data = pd.read_csv('forestfires.csv', header=0, names=cols, sep= ',')

month_cos_all = []
month_sin_all = []
day_cos_all = []
day_sin_all = []

for i, row in data.iterrows():

  month = MONTHS[row['month']]

  month_cos = round(math.cos(2 * math.pi * month / 12), 2)
  month_cos_all = np.append(month_cos_all, month_cos)

  month_sin = round(math.sin(2 * math.pi * month / 12), 2)
  month_sin_all = np.append(month_sin_all, month_sin)

  day = DAYS[row['day']]

  day_cos = round(math.cos(2 * math.pi * day / 7), 2)
  day_cos_all = np.append(day_cos_all, day_cos)

  day_sin = round(math.sin(2 * math.pi * day / 7), 2)
  day_sin_all =  np.append(day_sin_all, day_sin)

data['month_cos'] = month_cos_all
data['month_sin'] = month_sin_all
data['day_cos'] = day_cos_all
data['day_sin'] = day_sin_all

data.drop(['month', 'day'], inplace=True, axis=1)
data.head(n=10)
```

	X	Y	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	month_cos	month_sin	day_cos	day_sin
0	7	5	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0	0.0	1.00	0.62	-0.78
1	7	4	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0	0.5	-0.87	-0.90	0.43
2	7	4	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0	0.5	-0.87	1.00	-0.00
3	8	6	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0	0.0	1.00	0.62	-0.78
4	8	6	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0	0.0	1.00	0.62	0.78
5	8	6	92.3	85.3	488.0	14.7	22.2	29	5.4	0.0	0.0	-0.5	-0.87	0.62	0.78
6	8	6	92.3	88.9	495.6	8.5	24.1	27	3.1	0.0	0.0	-0.5	-0.87	-0.22	0.97
7	8	6	91.5	145.4	608.2	10.7	8.0	86	2.2	0.0	0.0	-0.5	-0.87	-0.22	0.97
8	8	6	91.0	129.5	692.6	7.0	13.1	63	5.4	0.0	0.0	-0.0	-1.00	-0.90	0.43
9	7	5	92.5	88.0	698.6	7.1	22.8	40	4.0	0.0	0.0	-0.0	-1.00	1.00	-0.00

c) Calcule o intervalo de confiança do RMSE para o 100 repetições de holdout 50/50 utilizando o classificador 1-NN com distância Euclidiana.

```
rmse_list = []
cols = ['X', 'Y', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH',
        'wind', 'rain', 'month_cos', 'month_sin', 'day_cos', 'day_sin', 'area']
data = data.reindex(columns=cols)

le = LabelEncoder()
data['area'] = le.fit_transform(data['area'])

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

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train, y_train)
  predict = knn.predict(X_test)

  rmse = mean_squared_error(y_test, predict, squared=False)
  rmse_list = np.append(rmse_list, rmse)

med = np.average(rmse_list)
dev = rmse_list.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: 112.27787730760838
Desvio padrão: 4.017962194121639
Intervalo de confiança: 104.40267   120.15308
```

4) Utilizando a base Forest Fires. Realize uma transformação na variável que representa a classe do problema. Cada valor da classe deve ser substituído pelo valor do seu logaritmo na base b:

$$y'_i = log_b(y_i + 1),$$

em que y'_i e y_i são, respectivamente, valor transformado e o valor original da classe do exemplo i e b é a base do logaritmo. Calcule o intervalo de confiança do RMSE para o 100 repetições de holdout 50/50 utilizando o classificador 1-NN com distância Euclidiana.

Utilizei o código da Questão 3 Letra B adicionado do seguinte:

```
rmse_list = []
cols = ['X', 'Y', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH',
        'wind', 'rain', 'month_cos', 'month_sin', 'day_cos', 'day_sin', 'area']
data = data.reindex(columns=cols)

data['area'] = np.log(data['area'] + 1) / np.log(10)

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

  knn = KNeighborsRegressor(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train, y_train)
  predict = knn.predict(X_test)

  rmse = mean_squared_error(y_test, predict, squared=False)
  rmse_list = np.append(rmse_list, rmse)

med = np.mean(rmse_list)
dev = np.std(rmse_list)

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

print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Intervalo de confiança: 0.79495   0.91284
```

5) A transformação inversa em relação a questão anterior é

$$y_i = b^{y'_i} - 1.$$

Refaça a questão anterior calculando a transformação inversa após a classificação e antes de calcular o RMSE. O objetivo desta questão é utilizar o valor transformado da classe para fazer a previsão e calcular o RMSE utilizando a escala original dos dados.

```
rmse_list = []
cols = ['X', 'Y', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH',
        'wind', 'rain', 'month_cos', 'month_sin', 'day_cos', 'day_sin', 'area']
data = data.reindex(columns=cols)

data['area'] = np.log(data['area'] + 1) / np.log(10)

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

  knn = KNeighborsRegressor(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(X_train, y_train)
  predict = knn.predict(X_test)

  y_test = (10 ** y_test) - 1
  predict = (10 ** predict) - 1

  rmse = mean_squared_error(y_test, predict, squared=False)
  rmse_list = np.append(rmse_list, rmse)

med = np.mean(rmse_list)
dev = np.std(rmse_list)

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

print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Intervalo de confiança: 54.12053   128.80049
```

6) Utilizando a base Car Evaluation. archive.ics.uci.edu/ml/datasets/Car+Evaluation
    a) Crie uma versão da base assumindo que todos os atributos são ordinais.

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

cols = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']

buying = ['low', 'med', 'high', 'vhigh']
maint = ['low', 'med', 'high', 'vhigh']
doors = ['2', '3', '4', '5more']
persons = ['2', '4', 'more']
lug_boot = ['small', 'med', 'big']
safety = ['low', 'med', 'high']
classification = ['unacc', 'acc', 'good', 'vgood']

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

enc = OrdinalEncoder(categories=[buying, maint, doors, persons, lug_boot, safety, classification])
data = pd.DataFrame(enc.fit_transform(data), columns=cols)

data.head()
```

	buying	maint	doors	persons	lug_boot	safety	class
0	3.0	3.0	0.0	0.0	0.0	0.0	0.0
1	3.0	3.0	0.0	0.0	0.0	1.0	0.0
2	3.0	3.0	0.0	0.0	0.0	2.0	0.0
3	3.0	3.0	0.0	0.0	1.0	0.0	0.0
4	3.0	3.0	0.0	0.0	1.0	1.0	0.0

b) Crie uma versão da base assumindo que todos os atributos NÃO são ordinais

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

cols = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']

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

transformer = make_column_transformer((OneHotEncoder(), ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']), remainder='passthrough', verbose_feature_names_out=False)
transformed = transformer.fit_transform(data)
transformed_data = pd.DataFrame(transformed, columns=transformer.get_feature_names_out())

transformed_data.head(n=10)
```

	buying_high	buying_low	buying_med	buying_vhigh	maint_high	maint_low	maint_med	maint_vhigh	doors_2	doors_3	...	persons_2	persons_4	persons_more	lug_boot_big	lug_boot_med
0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	0.0	1.0
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	0.0	1.0
6	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	1.0	0.0
7	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	1.0	0.0
8	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	1.0	0.0
9	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	1.0	0.0	0.0	0.0

c) Calcule, para cada versão da base, de forma pareada, os intervalos de confiança da acurácia para o 100 repetições de holdout 50/50 utilizando o classificador 1-NN com distância Euclidiana. Faça o teste de hipótese comparando os intervalos de confiança para verificar se há diferença significativa da acurácia entre os dois casos.
Como mostrado abaixo, o 0 está fora do intervalo, então rejeitamos o H_0, ou seja, os classificadores não tem o mesmo erro. Analisando os dados acima vemos que, a base ordinal tem uma taxa de acerto significativamente maior.

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

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)

cols = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']

buying = ['low', 'med', 'high', 'vhigh']
maint = ['low', 'med', 'high', 'vhigh']
doors = ['2', '3', '4', '5more']
persons = ['2', '4', 'more']
lug_boot = ['small', 'med', 'big']
safety = ['low', 'med', 'high']
classification = ['unacc', 'acc', 'good', 'vgood']

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

enc = OrdinalEncoder(categories=[buying, maint, doors, persons, lug_boot, safety, classification])
data1 = pd.DataFrame(enc.fit_transform(data), columns=cols)

transformer = make_column_transformer((OneHotEncoder(), ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']), remainder='passthrough', verbose_feature_names_out=False)
transformed = transformer.fit_transform(data)
data2 = pd.DataFrame(transformed, columns=transformer.get_feature_names_out())

cols_alt = transformer.get_feature_names_out()

data_x = data1[cols[:-1]]
data_y = data1['class']

data_alt_x = data2[cols_alt[:-1]]
data_alt_y = data2['class']

data_scores = []
data_alt_scores = []

for i in range(100):
  train_X, test_X, train_y, test_y = train_test_split(data_x, data_y, test_size=0.5)

  knn = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn.fit(train_X, train_y)

  data_score = knn.score(test_X, test_y)
  data_scores = np.append(data_scores, data_score)

  alt_train_X, alt_test_X, alt_train_y, alt_test_y = train_test_split(data_alt_x, data_alt_y, test_size=0.5)

  knn_alt = KNeighborsClassifier(n_neighbors=1, weights="distance", metric="euclidean")
  knn_alt.fit(alt_train_X, alt_train_y)

  data_alt_score = knn_alt.score(alt_test_X, alt_test_y)
  data_alt_scores = np.append(data_alt_scores, data_alt_score)
```

```python
print(f'Diferença das 100 taxas de acerto:\n{data_scores - data_alt_scores}\n')

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

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

print(f'Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}\n')

dif = {'Ordinal': data_scores,
       'Não Ordinal': data_alt_scores,
       'Diferença': data_scores - data_alt_scores}

df = pd.DataFrame(dif)
print(df)
```

```
Diferença das 100 taxas de acerto:
[0.05092593 0.05092593 0.08912037 0.05902778 0.04513889 0.01041667
 0.05671296 0.06712963 0.02662037 0.06944444 0.03703704 0.04976852
 0.05208333 0.06481481 0.07407407 0.07175926 0.05787037 0.05439815
 0.04861111 0.05555556 0.03703704 0.06597222 0.07291667 0.01967593
 0.02893519 0.07407407 0.04976852 0.04513889 0.06944444 0.09606481
 0.04513889 0.06134259 0.01851852 0.0625     0.04282407 0.03587963
 0.08796296 0.07060185 0.05787037 0.04166667 0.03356481 0.08333333
 0.07407407 0.03587963 0.03125    0.09143519 0.0625     0.05902778
 0.04282407 0.04976852 0.03703704 0.03935185 0.05555556 0.0162037
 0.05902778 0.05902778 0.03009259 0.06481481 0.08333333 0.04050926
 0.04166667 0.02314815 0.05555556 0.0462963  0.05439815 0.0462963
 0.02777778 0.05787037 0.08564815 0.08333333 0.06134259 0.0787037
 0.05092593 0.05902778 0.08333333 0.06134259 0.04861111 0.04861111
 0.06018519 0.06828704 0.07060185 0.03356481 0.04050926 0.06018519
 0.06712963 0.02430556 0.06481481 0.05555556 0.0787037  0.04282407
 0.04398148 0.05092593 0.03472222 0.0462963  0.05787037 0.0474537
 0.07523148 0.03240741 0.04513889 0.04398148]
```

```
Intervalo de confiança: 0.01858   0.08906
```

	Ordinal	Não Ordinal	Diferença
0	0.815972	0.765046	0.050926
1	0.834491	0.783565	0.050926
2	0.859954	0.770833	0.089120
3	0.826389	0.767361	0.059028
4	0.824074	0.778935	0.045139
5	0.812500	0.802083	0.010417
6	0.842593	0.785880	0.056713
7	0.849537	0.782407	0.067130
8	0.829861	0.803241	0.026620
9	0.848380	0.778935	0.069444
10	0.810185	0.773148	0.037037
11	0.825231	0.775463	0.049769
12	0.834491	0.782407	0.052083
13	0.832176	0.767361	0.064815
14	0.828704	0.754630	0.074074
15	0.843750	0.771991	0.071759
16	0.836806	0.778935	0.057870
17	0.822917	0.768519	0.054398
18	0.827546	0.778935	0.048611
19	0.841435	0.785880	0.055556
20	0.827546	0.790509	0.037037
21	0.832176	0.766204	0.065972
22	0.829861	0.756944	0.072917
23	0.788194	0.768519	0.019676
24	0.810185	0.781250	0.028935
25	0.842593	0.768519	0.074074
26	0.822917	0.773148	0.049769
27	0.827546	0.782407	0.045139
28	0.849537	0.780093	0.069444
29	0.859954	0.763889	0.096065
30	0.832176	0.787037	0.045139
31	0.832176	0.770833	0.061343
32	0.787037	0.768519	0.018519
33	0.847222	0.784722	0.062500
34	0.811343	0.768519	0.042824
35	0.842593	0.806713	0.035880
36	0.841435	0.753472	0.087963
37	0.843750	0.773148	0.070602
38	0.827546	0.769676	0.057870
39	0.833333	0.791667	0.041667
40	0.824074	0.790509	0.033565
41	0.820602	0.737269	0.083333
42	0.842593	0.768519	0.074074
43	0.819444	0.783565	0.035880
44	0.818287	0.787037	0.031250
45	0.831019	0.739583	0.091435
46	0.836806	0.774306	0.062500
47	0.832176	0.773148	0.059028
48	0.807870	0.765046	0.042824
49	0.827546	0.777778	0.049769
50	0.810185	0.773148	0.037037
51	0.819444	0.780093	0.039352
52	0.825231	0.769676	0.055556
53	0.813657	0.797454	0.016204
54	0.835648	0.776620	0.059028
55	0.815972	0.756944	0.059028
56	0.828704	0.798611	0.030093
57	0.846065	0.781250	0.064815
58	0.866898	0.783565	0.083333
59	0.825231	0.784722	0.040509
60	0.829861	0.788194	0.041667
61	0.820602	0.797454	0.023148
62	0.839120	0.783565	0.055556
63	0.817130	0.770833	0.046296
64	0.829861	0.775463	0.054398
65	0.828704	0.782407	0.046296
66	0.817130	0.789352	0.027778
67	0.833333	0.775463	0.057870
68	0.856481	0.770833	0.085648
69	0.839120	0.755787	0.083333
70	0.834491	0.773148	0.061343
71	0.841435	0.762731	0.078704
72	0.828704	0.777778	0.050926
73	0.819444	0.760417	0.059028
74	0.832176	0.748843	0.083333
75	0.844907	0.783565	0.061343
76	0.832176	0.783565	0.048611
77	0.825231	0.776620	0.048611
78	0.826389	0.766204	0.060185
79	0.844907	0.776620	0.068287
80	0.850694	0.780093	0.070602
81	0.817130	0.783565	0.033565
82	0.820602	0.780093	0.040509
83	0.832176	0.771991	0.060185
84	0.826389	0.759259	0.067130
85	0.821759	0.797454	0.024306
86	0.846065	0.781250	0.064815
87	0.817130	0.761574	0.055556
88	0.834491	0.755787	0.078704
89	0.811343	0.768519	0.042824
90	0.804398	0.760417	0.043981
91	0.829861	0.778935	0.050926
92	0.805556	0.770833	0.034722
93	0.818287	0.771991	0.046296
94	0.848380	0.790509	0.057870
95	0.849537	0.802083	0.047454
96	0.847222	0.771991	0.075231
97	0.811343	0.778935	0.032407
98	0.814815	0.769676	0.045139
99	0.822917	0.778935	0.043981

d) Faça o teste de hipótese comparando os intervalos de confiança da letra anterior com o intervalo de confiança pareado da acurácia calculado utilizando as características originais utilizando o 1-NN com distância de Hamming.
   * hamming para letra (b)
   * código termômetro (binário mantendo as características ordinais)