Universidade Federal do Agreste de Pernambuco

Bacharelado em Ciências da Computação

Prof. Tiago Buarque A. de Carvalho

Aprendizagem de Máquina:

Pré-processamento de Dados

Aluno: Vinícius Santos de Almeida

- 1. a) A melhor técnica de conversão de atributos categóricos para numéricos vai depender do tipo do atributo. Segue abaixo a melhor forma de lidar com cada um dos atributos da base:
  - **Binário, seja ordinal ou nominal**, todos os atributos abaixo se caracterizam como atributos binários, ou seja, podem assumir apenas dois valores na base e por isso podem ser convertidos para para forma binário {0,1} ou {-1, 1}:
    - school
    - sex
    - address
    - famsize
    - pstatus
    - schoolsup
    - famsup
    - paid
    - activities
    - nursery
    - higher
    - internet
    - romantic
  - **Não binário nominal**, todos os atributos abaixo podem ter seus valores convertidos em N colunas, dependendo da quantidade nominal de valores que o atributo pode assumir:
    - Mjob
    - Fjob
    - reason
    - guardian
  - As seguinte colunas já estão em formato numérico.
    - age
    - Medu
    - Fedu
    - traveltime
    - studytime
    - failures
    - famrel
    - freetime

    - goout
    - Dalc
    - Walc
    - health
    - absences
    - G1
    - G2
    - G3

- 1. b)
    - Converti para numéricos todos os atributos da questão **1(a)** listados em Binário e Não binário nominal, os demais atributos já estavam em formato numérico.
    - Ver arquivo `data/questao_1_b_export.csv` com a base com todos seus atributos em numéricos.
    - Implementação:

```
from utils import load_data, export_data

class BinaryConverter():

    def __init__(self, data, columns):
        self.data = data
        self.columns = columns

    def values_to_binary(self):
        possible_values = {}
        for key in self.columns:
            values = set()
            for r in self.data:
                values.add(r[key])

            possible_values[key] = values

        new_values = {}
        for key, value in possible_values.items():
            new_values[key] = dict(zip(value, (0,1)))

        return new_values

    def convert_data(self):
        b_values = self.values_to_binary()

        for index, row in enumerate(self.data):
            row_converted = row
            for key, value in row.items():
                if key in self.columns:
                    row_converted[key] = b_values[key][value]
                else:
                    row_converted[key] = value
            self.data[index] = row_converted
```

```
        return self.data

class NonBinaryConverter():

    def __init__(self, data, columns):
        self.data = data
        self.columns = columns
        print()

    def generate_columns(self):
        columns = []
        for column in self.data[0]:
            columns.append(column)
        return columns

    def new_columns(self):
        new_columns = {}
        for key in self.columns:
            values = set()
            for r in self.data:
                values.add(r[key])

            new_columns[key] = values

        return new_columns

    def convert_data(self):
        new_columns = self.new_columns()

        new_data = []
        for index, row in enumerate(self.data):
            row_converted = {i: v for i, v in row.items()}
            for key, value in row.items():
                if key in self.columns:
                    for column in new_columns[key]:
                        new_col_name = f'{key}_{column}'
                        if value == column:
                            row_converted[new_col_name] = 1
                        else:
                            row_converted[new_col_name] = 0
            for column in self.columns:
                del row_converted[column]
            new_data.append(row_converted)

        self.data = new_data
        return self.data

def main():
    data, _ = load_data('./data/student-mat.csv')

    binary_converter = BinaryConverter(data,
['sex','school','address','famsize','Pstatus','schoolsup','famsup
','paid','activities','nursery','higher','internet','romantic'])
    data = binary_converter.convert_data()
```

```
    nonbinary_converter = NonBinaryConverter(data,
['Mjob','Fjob','reason','guardian'])
    data = nonbinary_converter.convert_data()

    print('Base com atributos convertidos para numéricos: ')
    print(tuple(data[0].keys()))
    [print(tuple(row.values())) for row in data]
    export_data('./data/questao_1_b_export.csv', data,
nonbinary_converter.generate_columns())

if __name__ == '__main__':
    main()
```

- 1. c) Para converter a coluna G3 para valores binários, é necessário usar a técnica limiar para conversão de atributos numéricos para categóricos, para isso vou assumir dois intervalos:
  - `x >= 14 && x <= 20`, aluno na média ou acima dela.
  - `x >= 0 && x < 14`, aluno abaixo da média.
  - Os resultados da conversão podem ser conferidos em `data/questao_1_c_export.csv`.
  - Implementação:

```
    from utils import load_data, export_data

    class BinaryCategorizer():

        def __init__(self, data: list[dict], columns: list,
intervals: tuple[tuple]):
            '''
                - interval: dict = {'value1': 0, 'value2': 0,
'value3': 1, 'value4': 1}
            '''
            self.data = data
            self.columns = columns
            self.intervals = intervals

        def categorize(self, value):
            return self.intervals[value]

        def convert_data(self):
            categorized = []
            for row in self.data:
                new_row = {k:v for k,v in row.items()}
                for col in self.columns:
                    new_row[col] = self.categorize(new_row[col])
                categorized.append(new_row)
            self.data = categorized
            return self.data

    def main():
        data, columns =
load_data('./data/questao_1_b_export.csv')
```

```
        intervals = {str(v): (0 if v < 14 else 1) for v in
range(0,21)}
        binary_categorizer = BinaryCategorizer(data, ['G3'],
intervals)
        data = binary_categorizer.convert_data()
        print(tuple(data[0].keys()))
        [print(tuple(row.values())) for row in data]
        export_data('./data/questao_1_c_export.csv', data,
columns)

    if __name__ == '__main__':
        main()
```

- 1. d) O intervalo de confiança é `[0.86, 0.93]`.
    - Implementação do classificador:

```
import numpy as np

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support as
score
from sklearn.metrics import accuracy_score


class KNeighborsKFoldClassifier():

    def __init__(self, X, y, n_neighbors=1):
        self.X = X
        self.y = y
        self.n_neightbors = n_neighbors

    def load_data(self, filename):
        self.X = []
        self.y = []
        with open(filename, 'r') as file:
            for row in file.readlines():
                row_list = row.rstrip('\n').split(',')
                self.X.append([float(x) for x in row_list[:4]])
                self.y.append(row_list[4])
        self.X = np.array(self.X)
        self.y = np.ravel(self.y)

    def classify(self):
        self.accuracy_score = np.array([])
        for i in range(100):
            X_train, X_test, y_train, y_test =
train_test_split(self.X, self.y, test_size=.5)
            knn = KNeighborsClassifier(n_neighbors=1,
weights="distance", metric="euclidean")
```

```
            knn.fit(X_train, y_train)

            predicts = knn.predict(X_test)
            accuracy = accuracy_score(y_test, predicts)
            self.accuracy_score = np.append(self.accuracy_score,
accuracy)

    def intervalo_confianca(self):
        media = np.average(self.accuracy_score)
        desvio = self.accuracy_score.std()
        intervalo_confianca = (media-(1.96*desvio), media+
(1.96*desvio))
        return intervalo_confianca
```

- Implementação da questão:

```
import numpy as np

from my_kneighbors_kfold_classifier import
KNeighborsKFoldClassifier
from utils import load_data

data, columns = load_data('./data/questao_1_c_export.csv')
X = np.array([tuple([int(x) for x in list(row.values())[:28]]) +
tuple([int(x) for x in list(row.values())[29:46]]) for row in
data])
y = np.ravel([int(row['G3']) for row in data])
knn_kfold = KNeighborsKFoldClassifier(X, y)
knn_kfold.classify()
print(f'Intervalo de confiança:
{knn_kfold.intervalo_confianca()}')
```

- 2. a)
  - **Dados cíclicos**, pois são dados que representam informações cíclicas, que vão e se repetem sem existir um valor inicial ou um valor final, como dias da semana, meses do ano, horas do dia, minutos da hora etc. segue abaixo os atributos que podem ser convertidos para numéricos por essa técnica:
    - month
    - day
  - Os demais atributos, listados abaixo, já estão no formato numérico:
    - X
    - Y
    - FFMC
    - DMC
    - DC
    - ISI
    - temp
    - RH

    - wind
    - rain
    - area
- 2. b)
    - O base de dados convertida assim como solicitado está em `data/questao_2_b_export.csv`.
    - Implementação:

```
import numpy as np
import math

from utils import export_data, load_data

WEEK_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,
}

class CyclicConverter():

    def __init__(self, data, columns):
        self.data = data
        self.columns = columns

    def order_cyclic_table(self):
        for col in self.columns:
            if col == 'month':
                ordered = {}
                for month in self.cyclic_table[col]:
                    ordered[month] = MONTHS[month]
                self.cyclic_table[col] = ordered
            if col == 'day':
```

```
                ordered = {}
                for day in self.cyclic_table[col]:
                    ordered[day] = WEEK_DAYS[day]
                self.cyclic_table[col] = ordered

    def create_cyclic_table(self):
        self.cyclic_table = {}
        for col in self.columns:
            possible_values = list(set(row[col] for row in
self.data))
            self.cyclic_table[col] = possible_values

        self.order_cyclic_table()
        return self.cyclic_table

    def convert_data(self):
        self.create_cyclic_table()

        for index, row in enumerate(self.data):
            new_row = {k:v for k,v in row.items()}
            for key, value in row.items():
                if key in self.columns:
                    new_row[key] = (

CyclicConverter.get_sin_attr(self.cyclic_table[key][value],
len(self.cyclic_table[key])),

CyclicConverter.get_cos_attr(self.cyclic_table[key][value],
len(self.cyclic_table[key])))
            self.data[index] = new_row
        return self.data

    @staticmethod
    def get_sin_attr(index, length):
        return round(math.sin(2 * math.pi * index / length),2)

    @staticmethod
    def get_cos_attr(index, length):
        return round(math.cos(2 * math.pi * index / length),2)

def main():
    data, columns = load_data('./data/forestfires.csv', sep=',')
    cyclic_converter = CyclicConverter(data, ['month', 'day'])
    data = cyclic_converter.convert_data()

    print(tuple(data[0].keys()))
    [print(tuple(row.values())) for row in data]
    export_data('./data/questao_2_b_export.csv', data, columns)


if __name__ == '__main__':
    main()
```