PCS 3438

LISTA DE EXERCÍCIOS - Prof. Eduardo Raul Hruschka

Dupla

Nome	Nº USP	E-mail
Fernanda Monteiro Lopes	9793158	fernanda.monteiro.lopes@usp.br
Tiago DAgostino	3759732	tiago.dagostino@usp.br

Questão 1

O Algoritmo Naive Bayes Gaussiano é um classificador que presume que os atributos (x0, x1,...,x99) são independentes, ou seja, o valor de um atributo não influencia no valor de outro atributo, dada a informação da classe (target). Como os atributos utilizados são contínuos, pode-se presumir uma distribuição Gaussiana para estimar as probabilidades.

Para a avaliação do classificador, foi utilizada a Validação cruzada Holdout, que consiste em separar a base de dados em duas partes: uma para treinamento do classificador, e outra parte para a validação. Após treinado o algoritmo, verifica-se a acurácia com que ele classifica os dados em cada parte da base (Porcentagem da base em que o algoritmo acertou a classificação). Em geral, a acurácia na base de treinamento será maior que na base de validação.

Para os dados contidos no arquivo class01.csv, foram obtidos os seguintes resultados:

- **Acurácia na base de treino:** 68.92782782782756%
- **Acurácia na base de validação:** 64.30000000000062%

Questão 2

No algoritmo K-NearestNeighbors (KNN), os exemplos correspondem a pontos no Rn. Um exemplo é classificado de acordo com seus *k* vizinhos mais próximos (neste caso, k = 10), de acordo com uma medida de distância, neste caso, a distância Euclidiana

Para a avaliação do classificador, utilizou-se a Validação cruzada k-fold, onde k é o numero de subconjuntos iguais em que a base de dados é dividida. Nesta exercício, a base é dividia em 5 folds contendo 20% dos dados em cada. Neste método de avaliação, são utilizadas 4 pastas para treino e 1 para validação. Este processo é realizado k vezes, alternando o subconjunto de teste.

Para os dados contidos no arquivo class02.csv, foi obtido o seguinte resultado:

- **Acurácia na base de validação:** 64.30000000000062%

Questão 3

O LASSO é um método de regularização que reduz a possibilidade de overfit do modelo de regressão. Ele penaliza os coeficientes com um alto grau de correlação entre si, de acordo com o seu valor absoluto (soma dos valores dos estimadores) minimizando o erro quadrático. Isso é feito até que o coeficiente convirja para zero, onde o atributo é eliminado, o que reduzir a dimensionalidade do modelo.

Para a avaliação deste modelo, utilizou-se a metodologia Leave-One-Out. Neste método de avaliação, é deixado apenas 1 exemplo da base (de tamanho n) para a fase de teste, enquanto todos os outros exemplos (n-1) são utilizados para o treinamento do modelo. Este processo é realizado n vezes, alternando o exemplo utilizado para teste

Para os dados contidos no arquivo reg01.csv, foram obtidos os seguintes resultados:

- **RMSE na base de treino:** 19.220259837710353
- **RMSE na base de validação:** 15.465218791702433

Questão 4

A Árvore de regressão serve para categorizar variáveis em função de uma variável dependente.

Para os dados contidos no arquivo reg02.csv, foram obtidos os seguintes resultados:

- **MAE na base de treino:** 0.0 (o resultado foi obtido pois não foi feito poda na árvore, causando overfit)
- **MAE na base de validação:** 48.50298797044502

Questão 5

Verdadeiro	Quando ajustamos um modelo linear, geralmente supomos que os erros tem distribuição normal e são independentes e identicamente distribuídos (i.i.d.).
Verdadeiro	Quando ajustamos um modelo de regressão, podemos utilizar os valores

	preditos e os resíduos do modelo para avaliar se o modelo se adequa bem aos dados.
Verdadeiro	O coeficiente de determinação (r 2) indica, em termos percentuais, quanto da variabilidade da variável resposta é explicada pelas covariáveis do modelo.
Falso	Os modelos de regressão não são afetados por observações atípicas (outliers) e valores faltantes.
Verdadeiro	Considerando um modelo de regressão simples, temos que o coeficiente associado à covariável representa o grau de inclinação da reta.
Verdadeiro	Para efetuar regressão com o algoritmo KNN, pode-se fazer uma votação simples dos valores dos k vizinhos encontrados.
Verdadeiro	Para melhor desempenho da árvore de regressão, pode-se utilizar regressões lineares em suas folhas para previsão do valor final.
Verdadeiro	No algoritmo Random Forest para regressão, o valor predito é obtido pela média dos valores encontrados em cada árvore.

Códigos

QUESTÃO 1

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

Naive Bayes Gaussiano + metodologia holdout.
Resultado:
-- Naive Bayes + Holdout --
Dividindo 1000 linhas em treino=350 and teste=650 rows
Holdout - Acuracia do train set: 76.0%
Holdout - Acuracia do test set: 62.30769230769231%
-- Leave One Out --
Leave one out - Acuracia no Treino: 68.92782782782756%
Leave one out - Acuracia no Teste: 64.30000000000062%
"""
import csv
import random
import math
```

```
def loadCsv(filename):
    lines = csv.reader(open(filename, "r"))
    #tirar o cabecalho
    header = next(lines, None)
    dataset = list(lines)
    #passar tudo pra float
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]

    return dataset

def divDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    copy = list(dataset)

    return [copy[:trainSize], copy[trainSize:]]

def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated


def media(numbers):
    return float(sum(numbers))/float(len(numbers))


def variancia(numbers):
    avg = media(numbers)
    variancia = sum([pow(x-avg, 2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variancia)

def summarize(dataset):
    summaries = [(media(attribute), variancia(attribute))
                 for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries

def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries

def calcGaussiana(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean, 2)/(2*math.pow(stdev, 2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calcProbPorClass(summaries, inputVector):
    prob = {}
```

```
    for classValue, classSummaries in summaries.items():
        prob[classValue] = 1
        for i in range(len(classSummaries)):
            media, var = classSummaries[i]
            x = inputVector[i]
            prob[classValue] *= calcGaussiana(x, media, var)
    return prob


def predict(summaries, inputVector):
    probs = calcProbPorClass(summaries, inputVector)
    bestLabel = None
    bestProb = -1

    for (classValue, probability) in probs.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel


def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions

def getAccuracy(testSet, predictions):
    correct = 0

    for i in range(len(testSet)):

        if testSet[i][-1] == predictions[i]:
            correct += 1
            #print(testSet[i])
            #print(testSet[i][-1])
            #print(predictions[i])
    return (correct/float(len(testSet))) * 100.0

def main():
    print('-- Naive Bayes + Holdout --')
    #Carregando a Base
    dataset = loadCsv('class01.csv')
    #print(dataset[1][1])

    #dividindo a Base
    treinoSet, testSet = divDataset(dataset, 0.35)
    print(('Dividindo {0} linhas em treino={1} and teste={2} rows').format(
        len(dataset), len(treinoSet), len(testSet)))

    # modelo
    summaries = summarizeByClass(treinoSet)
    #print(summaries[0][1][1])

    # resultados no treinoset
    trainset_predictions = getPredictions(summaries, treinoSet)
```

```
    trainset_accuracy = getAccuracy(treinoSet, trainset_predictions)
    print(('Holdout - Acuracia do train set: {0}%').format(trainset_accuracy))

    # resultados no teste
    testset_predictions = getPredictions(summaries, testSet)
    testset_accuracy = getAccuracy(testSet, testset_predictions)
    print(('Holdout - Acuracia do test set: {0}%').format(testset_accuracy))


main()
```

QUESTÃO 2

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  3 18:24:18 2019

@author: Tiago

Accuracy: 84.66666666666667%
83.86666666666666
"""

# Example of kNN implemented from Scratch in Python

import csv
import math
import operator

def loadDataset(filename, split, pad):
    if not (0<=pad<10) and type(pad)!='int':
        raise ValueError('pad must be int between 0 and 9')
    with open(filename, 'r') as csvfile:

        lines = csv.reader(csvfile)
        header = next(lines)
        dataset = list(lines)
        trainSize = int(split*len(dataset))
        foldsize = int(len(dataset)/10)

        trainset = dataset[:pad*foldsize] + dataset[(pad+1)*foldsize:]
        testset = dataset[pad*foldsize:(pad+1)*foldsize]
        print(len(trainset))
        print(len(testset))
        return [trainset, testset]

def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow(float(instance1[x]) - float(instance2[x]), 2)
    return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
    distances = []
```

```
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0

def main():
    # prepare data
    split = 0.90
    k = 10
    nfold =10
    total_accuracy = 0

    for i in range(nfold):

        trainingSet, testSet = loadDataset('class02.csv', split,pad=i)
        # generate predictions
        predictions=[]

        for x in range(len(testSet)):
            neighbors = getNeighbors(trainingSet, testSet[x], k)
            result = getResponse(neighbors)
            predictions.append(result)

        accuracy = getAccuracy(testSet, predictions)
        print('Accuracy: ' + repr(accuracy) + '%')
        total_accuracy += accuracy/nfold
    print(total_accuracy)


main()
```

QUESTÃO 3

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  3 19:39:16 2019

@author: Tiago

-- Leave One Out --
Leave one out - MSE on test set: 15.465218791702433
Leave one out - MSE on train set: 19.220259837710353
"""

from sklearn.linear_model import Lasso

import sklearn as sk
import csv
import math
import numpy as np


lines = csv.reader(open("reg01.csv", "r"),delimiter=',')
header = next(lines)
dataset = list(lines)

X_data_temp = [[eval(x) for x in t] for t in dataset]
#print(X_data_temp)
X_data = [X_data_temp[i][:10] for i in range(len(X_data_temp))]
#print(X_data)
y_data = [eval(t[-1]) for t in dataset]
#print(y_data)


print('-- Leave One Out --')
MSEtrainset_accuracy=0
MSEtestset_accuracy=0
for i in range(len(dataset)):
    X_test = np.array(X_data[i]).reshape(-1,1).T
    y_test = np.array(y_data[i]).reshape(-1,1)
    X_train = np.array(X_data[:i]+X_data[i+1:])
    y_train = np.array(y_data[:i]+y_data[i+1:]).reshape(-1,1)

    las = Lasso(alpha=1)
    las.fit(X_train, y_train)

    MSEtestset_accuracy += (math.sqrt(sk.metrics.mean_squared_error(y_test, las.predict(X_test))))
    MSEtrainset_accuracy += (math.sqrt(sk.metrics.mean_squared_error(y_train, las.predict(X_train))))


print(('Leave one out - MSE on test set: {0}').format(np.mean(MSEtestset_accuracy)))
print(('Leave one out - MSE on train set: {0}').format(np.mean(MSEtrainset_accuracy)))
```

QUESTÃO 4

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 11 09:52:36 2019

@author: Tiago

Mean Absolute Error: 51.02853074153567
Mean Squared Error: 4434.737895080578
Root Mean Squared Error: 66.59382775513492
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv('reg02.csv')

#print(dataset.head())
#print(dataset.describe())

X = dataset.drop('target', axis=1)
y = dataset['target']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
y_pred_train = regressor.predict(X_train)

df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
print(df)

from sklearn import metrics
print('Teste')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print('Treino')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_train, y_pred_train))
print('Mean Squared Error:', metrics.mean_squared_error(y_train, y_pred_train))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_train, y_pred_train)))
```