TP2: Implementação do Algoritmo de Boosting

Tasso Augusto Tomaz Pimenta 2021072198

Table of contents

List of Tables

Apresentação

Explicação do codigo feito

Cross-validation

Essa função basicamente separa a data em k partes, para usar a cross validation

```
import pandas as pd
from math import log, e, log2

def splits(k, df):
    splits = []
```

```
    for i in range(k):
        splits.append(df[(len(df)//k)*(i):(len(df)//k)*(i+1)])
    return splits
```

Modelo de classficação fraco

Aqui eu crie o decision stumps via codigo mesmo para ser o separador fraco, um detalhe é que os pesos aqui representam a quantidade de dados, então se um dado tiver menos pesos do que o outro o grau de impureza será menor e portando o retorno do tronco também

```
#df = pd.read_csv('tic+tac+toe+endgame/tic-tac-toe.data', sep=',')
#df = df.sample(frac=1)#.reset_index(drop=True)
#cross = splits(5, df)
#train = pd.concat(cross[0:4])
#test  = pd.concat(cross[4:5])
#w1 = [1/len(train)]*len(train)
#t = tree(w,train)
#w = pd.Series(w1, index=train.index)
def tree(w, df):
    impurity = {}
    gini     = {}
    for col in df.columns[:-1]:
        positive = {'x': 0, 'o': 0, 'b': 0}
        negative = {'x': 0, 'o': 0, 'b': 0}
        _gini    = {'x': 0, 'o': 0, 'b': 0}
        _impurity = 0.0
        for i in df.index:
            value = df.loc[i, col]
            if value == 'x':
                if df.loc[i, 'x-win'] == 'positive':
                    positive[value] += w[i]
                else:
                    negative[value] += w[i]
            if value == 'o':
                if df.loc[i, 'x-win'] == 'positive':
                    positive[value] += w[i]
                else:
                    negative[value] += w[i]
            if value == 'b':
                if df.loc[i, 'x-win'] == 'positive':
```

```
                    positive[value] += w[i]
                else:
                    negative[value] += w[i]
        #print(col)
        #for value in ('x', 'o', 'b'):
        #    print(value, positive[value], negative[value])
        for value in ('x', 'o', 'b'):
            _gini[value] = 1 - pow((positive[value] / (positive[value] + negative[value])),
                            - pow((negative[value] / (positive[value] + negative[value])),
        for value in ('x', 'o', 'b'):
            _impurity += ((positive[value] + negative[value]) * _gini[value])#somatorio de
        impurity[col] = _impurity
        gini [col]    = _gini
    #print(impurity)
    #print(gini)
    best_col = min(impurity, key=impurity.get)
    #print([best_col, min(gini[best_col], key=gini[best_col].get)])
    return [best_col, min(gini[best_col], key=gini[best_col].get)]
```

classificador linear

basicamente o classificador identifica se o valor da entrada é o mesmo que o valor da Decision stunp, definido na função tree

```
class H():
    def __init__(self, result: list):
        self.col = result[0]
        self.val = result[1]
    def classifier(self, x:list):
        if x[self.col] == self.val:
            return 1
        else:
            return -1
```

Saida

Aqui é simplismente uma conversão do valor da string pra 1 ou -1

```
def y(x):
        if x['x-win']   == 'positive': return 1
        elif x['x-win'] == 'negative': return -1
        else:   print(f"Erro na posição {i}")
```

Modelo bosting

Separação dos dados

separando a parte do treino e a do test os dados foram embaralhados para ter uma separação melhor

```
df = pd.read_csv('tic+tac+toe+endgame/tic-tac-toe.data', sep=',')
df = df.sample(frac=1)#.reset_index(drop=True)
cross = splits(5, df)
train = pd.concat(cross[0:4])
test  = pd.concat(cross[4:5])
w1 = [1/len(train)]*len(train)
w = pd.Series(w1, index=train.index)
```

Aqui Está a definição do modelo, criei um loop de 50 iterações pois o resultado não muda se aumentat a quantidade de classificadores Eu mudei a atualização dos pesos, tirando o negativo do classificador obtive resultados melhores

```
alphas = []
classificadores = []
erro_empirico = []
decision = {}
it = 0
while it < 500:
    error = 0.0
    emp_error = 0# zerar o erro para refazer o calculo

    stump = tree(w,train)
    decision[it] = stump
    h = H(stump)

    for index, row in test.iterrows():#calculo do erro do separador
        if h.classifier(row) == -1: error += 1
```

```
    error = error/(len(test))
    alpha = log((1-error)/error)/2#caclulo do alpha de acordo com o erro do classificador

    for index, row in train.iterrows():#atualização dos pesos
        w[index] = w[index]*pow(e,alpha*y(row)*h.classifier(row))
    w = w/sum(w) #normatização de w

    classificadores.append(h)#adiciona o classificador a uma lista de classficadores
    alphas.append(alpha)     # adiciona o alpha a uma lista de alphas

    for index, row in test.iterrows():
        classification = 0.0
        for h in range(len(classificadores)):
            classification += classificadores[h].classifier(row)*alphas[h]
        if classification > 0: sgn = 1
        else: sgn= -1
        if sgn != y(row): emp_error +=1

    emp_error = emp_error/len(test)
    erro_empirico.append(emp_error)
    it += 1
```

Algumas escolhas da Decision stump

```
import random

it_ale = random.sample(range(it), 5)
for i in it_ale:
    print(i, decision[i])
```

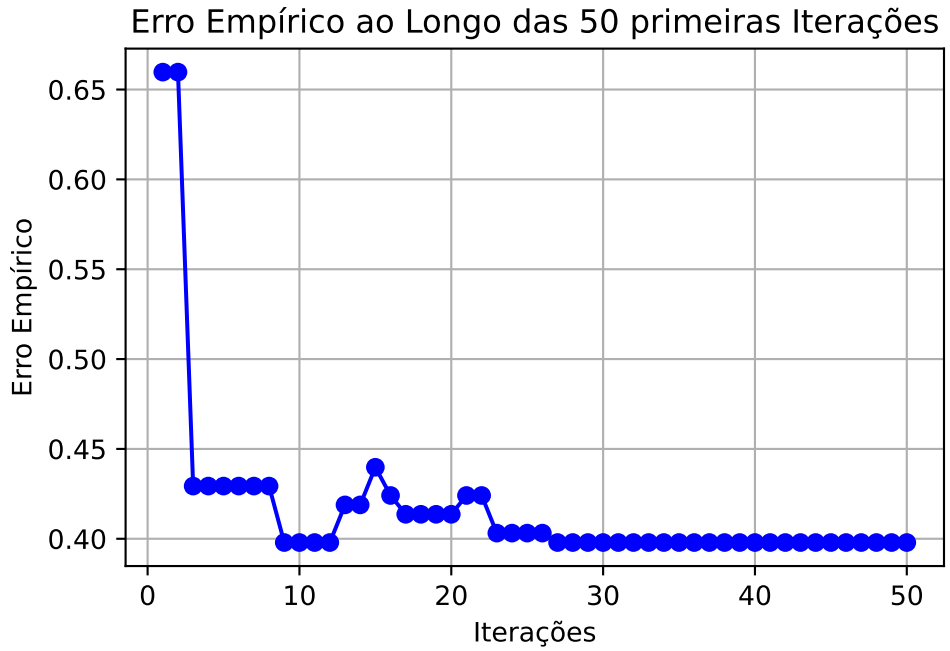
```
405 ['7', 'b']
83 ['9', 'x']
482 ['2', 'o']
116 ['7', 'x']
420 ['7', 'b']
```

Basicamente a função, começa recebendo um w inicial, e uma data de treino ela decide qual a melhor separação para os dados e seus pesos, testa o classificador, armazena seu erro, e calcula o alpha, atualiza os pesos ai finalmente adiciona o separador no classificador final com o alpha dentro da função signal, e por fim testa o erro total desse classificador ponderado # Grafico erro empirico por iterações Eu criei um grafico para mostrar os resultados, coloquei somente

50 iterações o resultado não melhorava com mais, o modelo ficou travado em 30% de erro mais ou menos, não consegui melhorar,a conclusão é que para jogo da velha boosting não funiona tão bem assim, chega em um ponto em que o resultado não melhora mais, tentei adicionar mais iterações para ver se o erro diminui porém ele semmpre trava na mesma % de erro por fim eu testei com 500 iterações para ver se em algum momento ele melhorava

```
import matplotlib.pyplot as plt

# Crie um gráfico de linha
plt.plot(range(1, len(erro_empirico[:50]) + 1), erro_empirico[:50], marker='o', linestyle=
plt.xlabel('Iterações')
plt.ylabel('Erro Empírico')
plt.title('Erro Empírico ao Longo das 50 primeiras Iterações')
plt.grid(True)
plt.show()
```




```
import matplotlib.pyplot as plt

# Crie um gráfico de linha
plt.plot(range(1, len(erro_empirico) + 1), erro_empirico, marker='o', linestyle='-', color
plt.xlabel('Iterações')
```

```
plt.ylabel('Erro Empírico')
plt.title('Erro Empírico ao Longo das 500 Iterações')
plt.grid(True)
plt.show()
```