PROJETO DISCIPLINA

APLICAÇÃO DO ALGORITMO K-NN

PABLO LUIZ LEON – UNIVERSIDADE FEDERAL - ABC

ALGORITMO K-NN

```
1  inicialização:
2      Preparar conjunto de dados de entrada e saída
3      Informar o valor de k;
4  para cada nova amostra faça
5      Calcular distância para todas as amostras
6      Determinar o conjunto das k's distâncias mais próximas
7      O rótulo com mais representantes no conjunto dos k's
8      vizinhos será o escolhido
9  fim para
10 retornar: conjunto de rótulos de classificação
```

PREPARAÇÃO DOS DADOS

```
def criarDados(RDD_Base, tamanho_Lista):
    """Cria uma RDD onde:
    id: Identificador
    lista: quantidade de itens do RDD base como itens de lista
    valor da lista + 1: valor posterior do RDD base

    Args:
        RDD_Base: Base de Dados.
        tamanho_list: qunantidade de valores para cada lista de cada item

    Returns:
        RDD novo contedo: id / lista de valores / valor posterior da lista de dados do RDD de or
    """

    #Cria indicie das informações
    RDD_DadosComIndicie = RDDDadosSeparados.map(lambda x: x[1]).zipWithIndex()

    #Cria tupla (chave, valor)
    DadosListaRDD = RDD_DadosComIndicie.map(lambda x : [(i,(float(x[1]), float(x[0]))) for i in
```

```
#Pega todo o RDD e transforma em uma lista única de itens
ListaUnicaTuplasRDD = DadosListaRDD.flatMap(lambda x : x)

#Aplica GroupByKey para agrupar os dados pela chave e criar a lista de valores para cada chave
DadosAgrupadosPorChave = ListaUnicaTuplasRDD.groupByKey().map(lambda x: (x[0], list(x[1])))
#Ordena os dados por chave de indice
DadosAgrupadosPorChaveOrdenados = DadosAgrupadosPorChave.sortBy(lambda x: x[0])

#for x in DadosAgrupadosPorChaveOrdenados.collect():
#    print(x)

#Remove as chaves que não fazem parte da estrutura final dos dados transformados.
RDD_DadosLimpos = DadosAgrupadosPorChaveOrdenados.filter(lambda x: tamanho_Lista < len(x[1]))

#for x in Teste.collect():
#    print(x)
#Monta os dados baseado na estrutura proposta (id, lista, valor)
DadosFinal = RDD_DadosLimpos.map(lambda x: (x[0],
                                            [x[1][i][1] for i in range(0,int(tamanho_Lista))],
                                            x[1][int(tamanho_Lista)][1]))
```

CÁLCULO DA DISTÂNCIA

```
def DistanciaEuclediana(RDDDados, DadosBase):
    """Calcula a Distancia Euclediana entre dois pontos:

    distancia = ((x1 - x2)^2 + (y1 - y2)^2)^2;

    Args:
        RDD de Dados (id, lista de valores de distancia, valor).
        Lista com os valors de base para calculo

    Returns:
        RDD com a distancia euclediana calculada para todos os pontos de análise
    """

    RDD_Pre_Euclediana = RDDDados.map(lambda x:(x[0],
                                    [np.power((DadosBase[i] -  x[1][i]),2) for i in range (0, len(
                                    x[2])
                         )

    RDDEucledianaFinal = (RDD_Pre_Euclediana.map(lambda x: (x[0],np.sqrt(sum(x[1])),x[2]))
                          .sortBy(lambda x: x[1]))


    return RDDEucledianaFinal
```

K-NN

```
def kNN(k, RDDTreino, Dados):
    """Aplica o Algoritmo de kNN - K Nearst Neirbohrs:

    Args:
        k ==> Amostra de K próximo.
        RDDTreino ==> (id, lista de valores de distancia, valor).
        Dados ==> Dado de base para calculo de kNN (Ultimo valor da série de Dados)

    Returns:
        Valor predito para o kNN indicado.
    """
    #Calcula Distancia Euclediana dos pontos Retorna um RDD com os pontos calculados por In
    RDDDadosCalculados = DistanciaEuclediana(RDDTreino, Dados)

    #Ordena os dados Calculados para discubrir a distancia menor (Ordenação pelos valores d
    RDDDadosCalculadosOrd = RDDDadosCalculados.sortBy(lambda x: x[1])

    #for x in RDDDadosCalculadosOrd.collect():
    #    print(x)

    valor_k_Dados = RDDDadosCalculadosOrd.take(2)
    #Pega o Identificador para filtrar os dados
    pFiltro = valor_k_Dados[1][0]
```

```
#pega o último item do RDD de valores
RDDlastItem = RDDTreino.filter(lambda x: x[2] == 0)


#pega o Valor predito pelo kNN
pChangeValue = valor_k_Dados[1][2]

#Cria um RDD com o valor da chave a ser predito colocando o valor da predição.
pRDDInvertido = RDDlastItem.map(lambda x: (x[0],x[1], float(x[2]+pChangeValue)))

#Aplica Join dos RDD para devolver o valor com indice da predição calculada.
RDDUnion = RDDTreino.union(pRDDInvertido)

#for x in RDDUnion.collect():
#    print(x)

#Aplica filtro para excluir chave de registro que tem valor zero.
```

SEQUENCIAL / PARALELISMO

```
141 # lê o arquivo com os dados e carrega em um RDD
142 DadosRDD1 = (sc.textFile(arquivo, 8).collect())
143
144 DadosRDD = sc.parallelize(DadosRDD1,8)
145 #Separa os dados do Arquivo (ativo, Data e Valor)
146 RDDDadosSeparados = DadosRDD.map(lambda x: x.split(";"))
147
```

RESULTADOS

- Mesma massa de dados, (Amostra pequena):
  - Tempo no Sequencial: 2m56s
  - Tempo no Paralelizado: 1m52s (36% mais eficiente).

- O Algoritmo não foi alterado com profundidade, mas há outros RDD, que poderiam ser paralelizados, principalmente durante a transformação dos dados (preparação) que tornaria o processamento mais eficiente.