Implementação do algoritmo Apriori no Spark

Pedro Henrique Parreira

Algoritmo Apriori

É um algoritmo para a mineração de conjunto de itens frequentes e aprendizado de regras de associação em bancos de dados de transações.

Um subconjunto de itens é chamado de *itemset,* e tem associado a ele um *suporte,* que é definido como a porcentagem de transações no qual ele aparece.

O algoritmo cria regras de associação do tipo A → B, onde a mesma indica uma relação interessante entre os dois conjuntos.

Cada regra de associação possui associada a ela um grau de *confiança,* que é definida como:

$$Conf(A \rightarrow B) = \frac{suporte(A \cup B)}{suporte(A)}$$

E também um Lift, que é definido como:

$$Lift(A \rightarrow B) = \frac{Suporte(A \cup B)}{Suporte(A)Suporte(B)}$$

É informado ao algoritmo um *suporte mínimo*, *Lift mínimo* e uma *confiança mínima* que são utilizados para selecionar os itemsets e regras de confiança geradas.

Algoritmo Apriori

Inicialmente o algoritmo gera os itemsets e posteriormente as regras de associação.

Na fase de geração dos itemsets, o algoritmo gera itemsets com *k* elementos, sendo *k* começando por um e sendo incrementado a cada iteração, e seleciona apenas os que tiveram um suporte maior ou igual ao suporte mínimo. Ou seja, em uma transação com *n* elementos são geradas $\binom{n}{k}$ combinação de k elementos.

A cada iteração é feito o processo de poda para evitar combinações desnecessárias, onde é descartado das transações itens que não apareceram nos itemsets que tiveram um suporte maior ou igual ao suporte mínimo.

Posteriormente é feita a geração das regras de associação do tipo A → B, onde A e B são itemsets gerados na fase anterior, que não tem nenhum elemento em comum e a união dos dois tenha o valor de suporte maior ou igual ao suporte mínimo.

E finalmente através do suporte calculado na fase anterior, é calculado a confiança e o lift destas regras de associação e são selecionadas as que tiveram a confiança e lift maiores ou igual as mínimas definidas.

Implementação no Spark

Fase de geração dos itemsets:

```
# Definindo função que irá gerar as combinações dos elementos
# Ex: (A,B,C) para k = 2 => (A,B),(A,C),(B,C)
def gerarCombinacoes(k, elementos):
    x = list(combinations(elementos, k))
    return ((tuple(x[i]), 1) for i in range(len(x)))

# Definindo função recebe um RDD contendo os dados e retorna uma com os candidatos que
# tiverem o suporte maior que o definido pelo usuário.
def gerarCandidatos(k,rddDados,totalTransacao,suporteMinimo):
    return  rddDados.flatMap(lambda x: gerarCombinacoes(k,x))\
                    .reduceByKey(lambda x, y: y + x) \
                    .map(lambda p: (p[0],round((p[1]/(totalTransacao*1.0)),6)))\
                    .filter(lambda xy: xy[1] >= suporteMinimo)
```

Cada iteração k, são geradas tuplas de acordo com o número de itens de cada transação, com a combinação gerada como chave e o valor “1” como segundo elemento e após a redução é obtido o número de transações que cada combinação aparece.

É calculado o suporte para cada combinação e selecionando apenas os que tiveram o suporte maior ou igual ao suporte mínimo.

Implementação no Spark

Fase de geração das regras de associação:

```
associacoes = candidatosTotais.flatMap(lambda x: [(tuple(x), tuple(b)) for b in elementos])\
                              .filter(lambda x: any(t in x[0] for t in x[1]) == False)\
                              .map(lambda x: (x[0],x[1],tuple(sorted(set().union(x[0],x[1])))),0))\
                              .filter(lambda x: ( (x[2] in dicionario.keys()) ) == True)\
                              .map(lambda x: (x[0],x[1],
                                       ((dicionario[x[2]])/(dicionario[x[0]]*1.0)),
                                       ((dicionario[x[2]])/(dicionario[x[0]]*dicionario[x[1]]*1.0)) ))\
                              .filter(lambda x: x[2] > confiancaMinimo) \
                              .filter(lambda x: x[3] > liftMinimo).collect()
```

Dos itemsets gerados na fase anterior é extraído um dicionário contendo os itemsets como chave e o valor de suporte dos mesmo, e uma lista de elementos contendo todos os itemsets.

É feito o produto cartesiano entre os itemsets da fase anterior, e são descartados os que tiverem algum elemento em comum, em seguida é gerado a união entre os dois itemsets das tuplas e é selecionado as tuplas onde a união entre os dois itemsets existe como chave no dicionário.

Finalmente é calculado a confiança e o lift usando os valores de suporte através e são selecionadas as regras com os valores maiores que os mínimos definidos.

Resultados Experimentais

Exemplo de saída do algoritmo implemento no Spark

De	Para	Confiança	Lift
('40706',)	('13176',)	0.197611	1.67426
('45007',)	('13176',)	0.215911	1.82931
('30391',)	('13176',)	0.267856	2.26941
('30391',)	('47209',)	0.206111	3.10239

Foi utilizado o dataset da competição Instacart Market Basket Analysis do Kaggle.

O arquivo possui 564 mb de tamanho e um total de 3.214.875 transações.

Em sua versão não distribuída o tempo de execução total foi de 927,7 segundos.

Em sua versão distribuída, para 4 threads, o tempo de execução total foi de 587,6 segundos.