

Uma versão distribuída do algoritmo Apriori em PySpark

Pedro Henrique Parreira
Santo André, São Paulo, Brasil

Abstract

O algoritmo apriori tem um custo computacional bastante elevado devido as inúmeras combinações de elementos que ele gera. Quando aplicado a um banco de dados bastante volumoso pode se tornar um grande desafio por necessitar de um grande volume de memória principal para armazenar e processar todas as combinações. Uma das possíveis saídas para tal problema é o uso da computação distribuída. Neste artigo iremos analisar uma versão distribuída do algoritmo apriori aplicada a uma grande base de dados e compará-la com a sua versão não distribuída.

Keywords: Apriori, Mineração de dados, PySpark,

1. Algoritmo Apriori

Um banco de dados de transações é formado por duas colunas, a primeira contendo o identificador da transação e a segunda os itens desta transação. Um exemplo clássico de um banco de dados de transações é o histórico de vendas de um supermercado onde cada venda tem um identificador único e os itens da venda.

Sendo $I = \{i_1, i_2, \dots, i_m\}$ o conjunto dos itens de um banco de dados de transações, definimos um *itemset* S como $S \subseteq I$, ou seja, um itemset é um subconjunto de itens do banco de dados. Para todo itemset associamos um *suporte* que é definido como a porcentagem das transações em que o itemset é encontrado.

O objetivo do algoritmo apriori é gerar regras de associação entre os itemsets, sob a forma $A \rightarrow B$, onde tal regra indica uma relação interessante entre tais itemsets. E para toda regra de associação medimos a *confiança* como:

15

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

Ou seja, a confiança de uma regra de associação é dada através da divisão entre a porcentagem de transações em que aparece o itemset $A \cup B$ e a porcentagem de transações em que aparece o itemset A .

E é associado também um *lift* para cada regra de associação, que é definido como:

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

O usuário fornece ao algoritmo o banco de dados, o *suporte mínimo*, *lift mínimo* e uma *confiança mínima* e obtém como saída tais regras de associação.

O algoritmo primeiramente gera itemsets de tamanho um, ou seja, formados apenas por um item que tenha o suporte maior ou igual ao suporte mínimo e a cada iteração vai aumentando de forma unitária o tamanho dos itemsets gerados e verificando quais tem o suporte maior ou igual ao suporte mínimo, até não ser mais possível gerar itemsets para um determinado tamanho.

A cada iteração é feito o processo de poda que consiste em descartar das transações itens que não aparecem em nenhum itemset, pois tais itens possuem um suporte menor que o suporte mínimo.

Ao final, os itemsets selecionados são usados para gerar as regras de associações, fazendo a combinação de tais itemsets, verificando quais associações possuem confiança e lifts maiores ou iguais as mínimas definida pelo usuário e usando os suportes que foram gerados na fase anterior.

2. Versão distribuída do algoritmo apriori

A implementação da versão distribuída do algoritmo consiste em duas etapas: a geração dos itemsets e a geração das regras de associação.

2.1. Geração dos itemsets

Cada transação é representada como uma tupla onde cada elemento da tupla é um item da transação. Como por exemplo ('1','2','3'), está tupla representa uma transação com três itens.

A geração dos itemsets começa com os de tamanho um e a cada nova iteração o tamanho é aumentado de forma unitária. Para um dado momento,

51 seja k o tamanho dos itemsets que vão ser gerados, então para cada transação
52 são realizadas $\binom{N}{k}$ combinações simples de tamanho k , onde N é o número
53 de itens da transação em questão. Cada combinação é definido como um
54 itemset.

55 Por exemplo, suponha que $k = 1$ e tomemos a transação ('1','2','3'),
56 portanto são gerados três itemsets: ('1','2'), ('1','3'), ('2','3'). Cada item-
57 set gerado é utilizado para gerar uma nova tupla, onde o itemset é uti-
58 lizado como chave seguido de um segundo elemento que será o número
59 um, portanto considerando o exemplo acima o resultado do mesmo seria:
60 (('1','2'),1), (('1','3'),1), (('2','3'),1).

61 A implementação da geração das tuplas contendo os itemsets e o número
62 um é feita com o uso da transformação *flatMap*. Ao final desta etapa temos,
63 para cada transação, os candidatos que poderam ser utilizados na geração
64 das regras de associação restando apenas verificar quais tem o suporte maior
65 que o suporte mínimo.

66 Para calcular o valor do suporte para todos os candidatos aplicamos a
67 transformação *reduceByKey*, com o operador de soma, e ao final obtemos
68 tuplas contendo como primeiro elemento o itemset e o segundo a quantidade
69 de transações em que tal itemset aparece.

70 Tendo o número de transações em que cada itemset aparece, basta dividir
71 o mesmo pelo número total de transações e assim obter o valor de suporte
72 para cada itemset. Para isto é aplicado uma transformação *map* onde em cada
73 tupla, o segundo elemento é dividido pelo número total de transações, obtendo
74 assim o valor de suporte, e em seguida é aplicado uma transformação *filter*
75 onde são selecionados os itemsets que contém o valor de suporte maior ou
76 igual ao valor de suporte mínimo.

77 A cada iteração é feita a poda, ou seja, é descartado das transações itens
78 que não aparecem nos itemsets, evitando a geração desnecessária de can-
79 didatos.

80 A etapa de geração dos itemsets é encerrada quando não for possível obter
81 nenhum itemset para aquele determinado tamanho, restando apenas a etapa
82 de geração das regras de associação.

83 2.2. Geração das regras de associação

84 Ao final da primeira etapa temos os itemsets e seus respectivos valores de
85 suporte, nesta última etapa são gerada as regras de associação que tenham o
86 valor de confiança e lift maiores que os definido pelo usuário.

87 É gerado um dicionário de dados com a transformação `collectAsMap`, ou
88 seja, através de um itemset podemos acessar o valor de seu suporte. Através
89 de uma transformação `map` são gerados os produtos cartesianos de todos
90 os itemsets, ou seja, são gerados tuplas de tamanho dois onde ambos os
91 elementos são itemsets distintos. Tais produtos cartesianos são as possíveis
92 regras de associações.

93 É aplicado posteriormente uma transformação `filter` onde é descartados
94 as tuplas em que existe algum elemento em comum entre os dois itemsets. A
95 tupla (X, Y) , é um exemplo de tupla gerada onde X e Y são os itemsets.

96 Posteriormente é necessário gerar um terceiro elemento $X \cup Y$, para cada
97 tupla que será utilizado para acessar o dicionário e calcular a confiança e o
98 lift da regra de associação $X \rightarrow Y$, assim é obtido no final deste processo
99 uma tupla com três elementos que no caso do exemplo citado acima seria,
100 $(X, Y, X \cup Y)$.

101 Antes de calcular a confiança e o lift da regra de associação, é necessário
102 verificar se existe a chave $X \cup Y$ no dicionário, portanto é aplicado um trans-
103 formação `filter` onde é descartada todas as tuplas que não tenham o terceiro
104 elemento como chave do dicionário. Isto significa na prática que o itemset
105 $X \cup Y$ não tem um valor de suporte maior ou igual ao valor de suporte
106 mínimo.

107 Posteriormente é feito o cálculo da confiança e lift utilizando os valores de
108 suporte do dicionário, que são acessados utilizando os itemset como chave,
109 restando apenas a aplicação de uma transformação `filter` para selecionar as
110 regras de associações que tiveram valores de confiança e lift maiores que os
111 definidos pelo usuário.

112 3. Resultados experimentais

113 Para os experimentos foi utilizado a base de dados da competição *Instacart Market Basket Analysis* do Kaggle, mais especificamente o arquivo
114 *order_products_prior.csv* [1]. Tal arquivo possui 3.214.875 de transações.

115 Em sua versão não distribuída o algoritmo foi executado totalmente em
116 15 minutos e 30 segundos. A sua versão distribuída para quatro threads foi
117 executada totalmente em 10 minutos, ou seja, a sua versão distribuída levou
118 cerca de 35 % a menos.

119 Como o esperado verificamos que o algoritmo, em sua versão distribuída,
120 obteve um ganho bastante interessante de tempo de processamento do que a
121 sua versão não distribuída.
122

123

124 **References**

- 125 [1] Competição Instacart Market Basket Analysis. [https://www.kaggle.](https://www.kaggle.com/c/instacart-market-basket-analysis/data)
126 [com/c/instacart-market-basket-analysis/data](https://www.kaggle.com/c/instacart-market-basket-analysis/data), 2018.
- 127 [2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining
128 association rules in large databases. In *Proceedings of the 20th Interna-*
129 *tional Conference on Very Large Data Bases*, VLDB '94, pages 487–499,
130 San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.
- 131 [3] Wikipedia. Apriori algorithm, [https://en.wikipedia.org/wiki/](https://en.wikipedia.org/wiki/Apriori_algorithm)
132 [Apriori_algorithm](https://en.wikipedia.org/wiki/Apriori_algorithm), 2018.