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 ma 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 do histórico de vendas de um supermercado onde cada venda tem um identificador único e os itens da venda.

Sendo $I = \{i_1, i_2, ..., i_m\}$ 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* desta como:

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

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.

O usuário fornece ao algoritmo o banco de dados, o *suporte mínimio* e *confiança mínimia* e obtém como saída tais regras de associação.

O algoritmo, primeiramente, gera itemsets de tamanho 1, ou seja, formado apenas por um item que tenha o suporte maior que o suporte mínimio e a cada iteração vai aumentando de forma unitária o tamanho dos itemsets gerados, que tenham o suporte maior que o suporte mínimio, 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 mínimio.

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

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

Por exemplo, suponha que $k = 1$ e tomemos a transação ('1','2','3'), portanto são geradoss três itemsets: ('1','2'),('1','3'),('2','3'). Cada itemset gerado é utilizado para gerar uma nova tupla, onde o itemset é utilizado como chave seguido de um segundo elemento que será o número

um, portanto considerando o exemplo acima o resultado do mesmo seria: (('1','2'),1),(('1','3'),1),(('2','3'),1).

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

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

Considerando o exemplo acima, suponha que ao final desta etapa, as tuplas tenham sido alteradas para (('1','2'),10),(('1','3'),8),(('2','3'),4), ou seja, o itemset ('1','2') apareceu em 10 transações, o itemset ('1','3') apareceu em 8 e o itemset ('2','3') apareceu em 4.

Tendo o número de transações em que cada itemset aparece, basta dividir o mesmo pelo número total de transações e assim obter o valor de suporte para cada itemset. Para isto é aplicado uma transformação *map* onde em cada tupla, o segundo elemento é divido pelo número total de transações, obtendo assim o valor de suporte, e em seguida é aplicado uma transformação *filter* onde são selecionados os itemsets que contém o valor de suporte maior que o valor de suporte mínimio.

A cada iteração é feita a poda, ou seja, é descartado das transações itens que não aparecem nos itemsets, evitando a geração desnecessária de candidatos.

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

2.2. Geração das regras de associação

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

É gerado um dicionário de dados com a transformação collectAsMap, ou seja, através de um itemset podemos acessar o valor de seu suporte. Atravês de uma transformação map são gerados os produtos cartesianos de todos os itemsets, ou seja, são gerados tuplas de tamanho dois onde ambos os

elementos são itemsets distintos. Tais produtos cartesianos são as possíveis regras de associações.

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

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

Antes de calcular a confiança e o lift da regra de associação, é necessário verificar se existe a chave $X \cup Y$ no dicionário, portanto é aplicado um transformação filter onde é descartada todas as tuplas que não tenham o terceiro elemento como chave do dicionário. Isto significa na prática que o itemset $X \cup Y$ não tem um valor de suporte maior que o valor de suporte mínimo.

Posteriormente é feito o cálculo da confiança e lift utilizando os valores de suporte do dicionário, que são acessador utilizando os itemset como chave, restando apenas a aplicação de uma transformação filter para selecionas as regras de associações que tiveram valores de confiança e lift maiores que os definidos pelo usuário.

3. Resultados experimentais

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

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

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

References

[1] Competiç ao Instacart Market Basket Analysis. https://www.kaggle.com/c/instacart-market-basket-analysis/data, 2018.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] Wikipedia. Apriori algorithm, https://en.wikipedia.org/wiki/Apriori_algorithm, 2018.