Algoritmo ID3 para Classificação de Campanhas de Marketing

Douglas Galetti Ribeiro

São Paulo, Brasil

Abstract

This project aims to implement a code able to classify a given information related to marketing campaigns using the algorithm ID3. A decision tree was generated for the campaigns used and information about their performance and the relationship between them and week days was retrieved. Also, the implemented code was tested using up to four processor threads to verify performance gain during its execution.

Keywords: ID3, Classification Tree, PySpark, Multithreading

1. Introdução

Empresas que trabalham com campanhas de *marketing* possuem a necessidade de entender o valor do negócio e se as estratégias utilizadas garantem os resultados esperados. Este trabalho tem por objetivo implementar um código que utilize o algoritmo classificador ID3 para avaliar uma base de dados de campanhas e seus desempenhos durante as semanas, dessa forma, espera-se obter uma árvore de decisão que possa classificar as campanhas de acordo com o desempenho semanal e indicar quais possuem menor ou maior potencial de aceite pelos usuários e em quais dias da semana.

2. Classificação de Dados Utilizando o Algoritmo ID3

Uma árvore de decisão é um modelo utilizado para classificar dados de maneira eficaz, sendo capaz de manipular dados categóricos e numéricos. Esse modelo realiza a classificação na forma de uma estrutura de árvore. Isso é realizado através da divisão de um conjunto de dados em subconjuntos cada vez menores ao mesmo tempo que uma árvore de decisão associada é

desenvolvida de forma incremental. O resultado final é uma árvore com nós de decisão e nós de folha. Um nó de decisão tem duas ou mais ramificações e o nó da folha representa uma classificação ou decisão [1].

O algoritmo de árvore de decisão ID3 (Iterative Dichotomiser 3) calcula a entropia e ganho de informação para construir uma árvore de decisão [2]. A árvore é construída de cima para baixo a partir de um nó raiz e envolve o particionamento dos dados em subconjuntos que contém instâncias com valores semelhantes. Se a amostra é completamente homogênea, a entropia é zero e se a amostra é dividida igualmente, tem entropia igual a um [1][3].

Para construir uma árvore de decisão, é necessário calcular dois tipos de entropia: (1) entropia usando a tabela de frequência de um atributo e (2) Entropia usando a tabela de frequência de dois atributos.

$$E(S) = \sum_{i=1}^{c} -P_i log_2 P_i \tag{1}$$

$$E(T, X) = \sum_{c \in X} P(c)E(c) \tag{2}$$

O ganho de informação é baseado na diminuição da entropia depois que um conjunto de dados é dividido em um atributo (3). Construir uma árvore de decisão é encontrar um atributo que retorna o maior ganho de informação, ou seja, os ramos mais homogêneos. Uma vez medida a entropia do conjunto podemos aplicar o ganho de informação para selecionar o melhor atributo para ser o primeiro nó de uma árvore [1][2].

$$GanhoInfo(T, X) = E(T) - E(T, X) \tag{3}$$

3. Metodologia

Foi realizado o pré-processamento de uma base de dados contendo as campanhas semanais, quantidade de impactos ao cliente, quantidade de produtos comprados entre outros. Em seguida, o classificador ID3 foi utilizado de forma a gerar uma árvore de decisões com a classificação de cada campanha segundo seu desempenho e outros atributos relevantes. O pré-processamento e a classificação foram implementados em linguagem Python, tendo o primeiro utilizado de recursos da ferramenta PySpark.

3.1. Pré-processamento de Dados

O algoritmo ID3 é responsável pela classificação de dados, mas estes dados precisam estar formatados de modo que o classificador entenda. Portanto, foi necessário a implementação de um algoritmo capaz de processar uma base de dados original que continha 33 milhões de linhas com colunas e valores em comuns. Esses dados foram importados a um RDD utilizando o PySpark e foi realizado um mapeamento e redução deles em grupos menores e também pré-classificação desses dados.

O arquivo pré-processado possui o nome da campanha, o dia da semana que foi executada, o alcance do impacto, alcance da aceitação ou compra do produto. Com base nesses dois últimos atributos, é feita a pré-classficação de desempenho em péssimo, ruim,bom, excelente. Após essa etapa, os dados processados são utilizados no classificador ID3.

Original: fbz_gld_bonif_suarenda_f2,710474562,2018-05-04 09:39:13, 2018-05-04 09:39:13,2018-05-04 09:39:22,,1,0,,,,,1,5,,5585987812007

Pré-processado: fbz_gld_bonif_suarenda_f2,fri,baixo,normal,RUIM

3.2. Algoritmo Classificador ID3

A criação da árvore de decisão requer o cálculo da entropia para um conjunto e, posteriormente, o cálculo do ganho de informação, este último utilizado para definir o nó de decisão e os nós de folhas[1][4]. O código implementado possui duas funções principais relacionadas ao algoritmo ID3:

calculoEntropiaParticao

classificador

A primeira função faz o cálculo da entropia de cada atributo; a segunda é a responsável pelos cálculos e chamadas em outras funções para obter o ganho de informação, encontrar o maior ganho da partição atual, escolher os atributos relevantes, fazer uma nova partição desses atributos e executar o mesmo processo para cada partição até que o ganho de informação seja zero ou não haja mais atributos a serem processados. Dessa forma, gera-se os nós de decisão da árvore, assim como os nós de folhas que correspondem a classificação dos atributos.

3.3. Árvore de Decisão

Conforme o algoritmo ID3 é executado, a árvore de decisão com as classificações dos atributos é gerada com seus ramos contendo os nós de decisão e os nós de classificação ou folhas. Após o término do classificador a árvore é exibida na tela:

> 'atributo': 'CAMPANHA', 'nodes': 'fbz_mov_marvel_f3': 'atributo': 'ACEITE', 'nodes': 'alto': 'atributo': 'EXCELENTE', 'normal': 'atributo': 'BOM', ...

3.4. Desempenho do Algoritmo

O algoritmo ID3 implementado foi executado utilizando PySpark. Foram executados testes utilizando uma, duas, três e quatro thread. O tempo de execução foi armazenado pela prórpia ferramenta (Table 1).

Threads	**Execução (min)**
1	5.3 min
2	5.0 min
3	4.9 min
4	4.5 min

Table 1: Número de threads e tempo execução do classificador

4. Resultados

O código implementado foi executado quatro vezes, utilizando diferentes quantidades de threads. O pré-processamento foi o gargalo do código, pois processou 33 milhões de linhas, enquanto que o classificador executou apenas as 32 linhas geradas pelo pré-processador devido ao *MapReduce* executado por este. A árvore gerada forneceu informações de interesse. Por exemplo, campanhas da Marvel sempre tinham desempenho ruim, enquanto que campanhas do StarWars tinham bons desempenhos. Campanhas como gamedom apresentou desempenho melhor em alguns dias da semana.

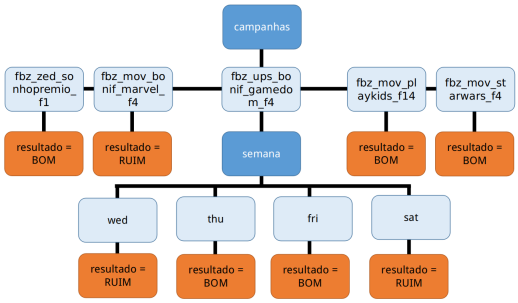

Figure 1: Resultado parcial da árvore de decisão para campanhas de marketing.

5. Conclusão

Com algoritmo de classificação ID3 foi possível classificar uma base de dados de tamanho considerável e obter informações relevantes sobre as campanhas de marketing realizadas. A árvore de decisão e a classificação dos atributos permitiram que as campanhas fossem classificadas com base no desempenho de aceitação (compra) do produto e impacto ao cliente, além de outros dados de interesse como o tipo de produto ou dia da semana que dado produto tem um desempenho de vendas considerado bom ou ruim. Essas informações permitiriam que uma empresa pudesse rever seus produtos ofertados e determinar novas estratégias de venda.

Em relação ao código, o uso de RDDs proporcionou paralelismo ao pré-processamento da base de dados, portanto foi observado que tempo de execução do código foi menor conforme mais threads utilizadas. O classificador ID3 não foi implementado usando RDD, porém, a base utilizada por este era muito pequena e seu uso não agregaria valor ao desempenho do mesmo.

[1] N. Shukla, Haskell Data Analysis Cookbook, Packt Publishing, 1st edition, 2014.

[2] S. Sayad, Decision Tree - Classification, http://www.saedsayad.com, 2018.

[3] P. Chiusano, B. Runar, Functional programming in Scala, Manning Publications, 2015.

[4] D. Wampler, Programação Funcional Para Desenvolvedores Java, O'Relly Novatec, 2012.

* Base utilizada para pré-processamento não está disponível ao público.