Relatório

Duodécimo
duodecimo@gmail.com

Resumo

Em 26 de Outubro de 2018 recebi do Leoncio Cruz uma proposta de tarefa. A idéia inicialmente seria verificar o desempenho da plataforma Apache Spark [1] no processamento de uma árvore de decisão treinada à partir de um arquivo para um sistema de QA.

Ao testar o desempenho do Spark na tarefa, verifiquei que não era satisfatório. O Leoncio havia proposto uma meta de processamento em no máximo 25 minutos, e os códigos que testei iam além de dezenas de horas.

Propus então utilizar a biblioteca Numpy, com a linguagem Python, conseguindo um tempo de processamento de 2 a 4 minutos, implementando basicamente um algoritmo descrito pelo Leoncio.

Na sexta feira, 30 de Novembro de 2018, tivemos uma conferência remota em que o Leôncio me apresentou o prof. Antonio Juarez Alencar. Ele colocou uma interessante meta de melhorar o resultado de forma sucessiva, sem buscar de início o melhor resultado possível.

Existem detalhes muito importantes no caminho do desenvolvimento do programa que realiza esta tarefa. Obtive alguns resultados promissores. Vou relatá-los no último capítulo.

I. Introdução

A tarefa é criar uma árvore de decisão a partir de dados disponibilizados. A árvore é armazenada, e a posteriori, indica a resposta a ser dada para uma mensagem.

II. Processando a árvore

a. Entrada de dados

Os dados são obtidos do arquivo Training.csv, com 74,9 MB, consistindo de 11720 linhas e 3193 colunas. A primeira linha é o cabeçalho: Traz o termo 'Answer' na primeira coluna seguida de 3192 palavras extraídas de mensagens. As demais linhas trazem na primeira coluna um valor numérico que codifica uma resposta, seguida de valores binários em cada coluna, significando a presença ou não da palavra no cabeçalho da coluna.

Resposta	**palavra$_1$**	**...**	**palavra$_n$**
r_1	{0,1}	{0,1}	{0,1}
r_2	{0,1}	{0,1}	{0,1}
⋮	⋮	⋮	⋮
r_m	{0,1}	{0,1}	{0,1}

b. Árvore

Cada nó da árvore é rotulado com uma palavra. A lógica do processamento, a partir do nó raiz, é dividir o conjunto de mensagens, enviando para a esquerda as mensagens sem a palavra rótulo do nó, e para a direita as que possuem a palavra. Um nó também envia para os filhos um conjunto de palavras, sem a palavra que foi escolhida para seu rótulo. O nó raiz recebe um conjunto completo de palavras. As folhas da árvore não possuem rótulos com palavras. Armazenam uma resposta. As folhas são atingidas quando o processamento de um nó produz:

- um conjunto de mensagens que possuem a mesma resposta.
- um conjunto de mensagens vazio (e nesse caso a resposta é desconhecida).
- Um conjunto de palavras vazio.

O ponto vital do processamento de uma árvore é a função de escolha da palavra que vai rotular o nó, que podemos chamar de escolha da melhor palavra.

c. Função de escolha da melhor palavra

Este é um ponto em que ainda tenho dúvidas.

Ao considerar o funcionamento da árvore, me parece que a classificação da mensagem é uma função definida em relação às mensagens do conjunto de treino, ou seja, qualquer mensagem que tenha um mesmo conjunto $\mathbf{\Gamma}$ de palavras, deve ser classificada com a mesma resposta. Além disso, se possuir um conjunto de palavras maior do que $\mathbf{\Gamma}$, mas se o processamento da árvore ao final do conjunto $\mathbf{\Gamma}$ leva a uma folha, é classificada também com a resposta daquela folha.

E neste caso, a ordem de $\mathbf{\Gamma}$ é comutativa. Isto indica, para mim, que a melhor palavra, a cada processamento de um nó, a princípio é a que cause a maior divisão de mensagens, contra a degeneração da árvore binária.

Nos próximos três subcapítulos vou expor os três métodos que utilizei para a função de escolher a melhor palavra. Denominei-os:

- máxima divisão
- gini de uma resposta
- gini total

> **Patologia de uma folha.**
>
> Podemos discutir aqui uma patologia da resposta de uma folha. Se a folha for atingida por mais de uma mensagem, e que haja duas ou mais respostas diferentes entre elas, é preciso escolher uma das respostas, em detrimento das outras. Um critério de escolha pode ser a que estiver no maior número de mensagens. Mas, o que importa aqui, é a consequência da patologia: As mensagens que tiverem respostas diferentes da escolhida, se submetidas posteriormente à classificação da árvore, serão classificadas de forma errada.
>
> Uma causa para a patologia da folha é o fato de haver mensagens com exatamente o mesmo conjunto de palavras mas com respostas diferentes.

d. Método máxima divisão

Ao buscar uma maneira de, primeiramente dividir ao máximo a árvore a cada nó processado (evitar a degeneração da árvore é a melhor forma de acelerar o processamento) e além disso, fazer isso com a menor complexidade possível (também um fator que acelera o processamento), apliquei o seguinte procedimento:

Dada a matriz de valores mensagens x palavras, onde m é o número de mensagens e n o número de palavras, obtemos a palavra que aparece no maior número de mensagens:

$$max(i)\sum_{i=1}^{n}\sum_{j=1}^{m} valor_{ij}$$

Simplesmente assim, buscando diretamente a maior divisão do conjunto.

Se valer a teoria de que a ordem das palavras na árvore é comutativa, esta promete ser uma forma eficiente de agir.

e. Método gini de uma resposta

Buscando uma palavra com maior qualidade de divisão da árvore, escolhemos inicialmente a resposta com maior número de ocorrências.

Dada a primeira coluna da matriz de mensagens, respostas, onde m é a quantidade de respostas únicas, e n a quantidade de mensagens, obtemos a resposta que aparece no maior número de mensagens:

$$max(quantidade)\sum_{i=1}^{m}\sum_{j=1}^{n} resposta_i, quantidade_{ij}$$

Com a resposta escolhida, passamos a calcular o distúrbio gini:

1) Primeiro do conjunto (mensagens com relação a resposta escolhida):

$$Gini_{(t)} = 1 - \sum_{i=1}^{m} [\,p(i|t)]^2 = 1 - [(\frac{qm_r}{qmt})^2 + (\frac{qm_{\neq r}}{qmt})^2\,] \quad ,$$

onde qm_r é a quantidade de mensagens com a resposta r, $qm_{\neq r}$ é a quantidade de mensagens com resposta diferente da resposta r, e qmt é a quantidade total de mensagens.

2) para cada palavra pl:

2.1) Para cada mensagem com a palavra pl:

$$Gini_{(pl)} = 1 - \sum_{i=1}^{m} [\,p(i|t)]^2 = 1 - [(\frac{qm_{pl_r}}{qm_{pl}})^2 + (\frac{qm_{pl_{\neq r}}}{qm_{pl}})^2\,]$$

onde

$qm_{pl_r} = mensagens\ com\ a\ palavra\ avaliada\ e\ a\ resposta\ escolhida$

$qm_{pl_{\neq r}} = mensagens\ com\ a\ palavra\ avaliada\ e\ resposta\ diferente\ da\ escolhida$

$qm_{pl} = mensagens\ com\ a\ palavra\ avaliada$

2.2) para cada mensagem sem a palavra pl:

$$Gini_{(\not{p}l)} = 1 - \sum_{i=1}^{m} [\, p(i|t)]^2 = 1 - [(\frac{qm_{\not{p}l_r}}{qm_{\not{p}l}})^2 + (\frac{qm_{\not{p}l_r}}{qm_{\not{p}l}})^2]$$

onde

$qm_{\not{p}l_r} = mensagens\ sem\ a\ palavra\ avaliada\ ,\ com\ a\ resposta\ escolhida$

$qm_{\not{p}l_r} = mensagens\ sem\ a\ palavra\ avaliada\ e\ resposta\ diferente\ da\ escolhida$

$qm_{\not{p}l} = mensagens\ sem\ a\ palavra\ avaliada$

3) Calculo do gini da palavra:

$$Gini_{(p)} = Gini(o) - [Gini_p \times (\frac{qm_{pl_r}}{qmt}) + Gini_{\not{p}} \times \frac{qm_{\not{p}l_r}}{qmt}]$$

finalmente, escolhemos a palavra com o maior gini.

f. Método gini total

Vou adicionar a fórmula do cálculo depois, pois, com veremos adiante nos capitulo 3, resultados, ele sempre perde para o gini de uma resposta.

III. Resultados

Creio que obtive resultados potencialmente interessantes, e, seguindo a orientação do prof. Juarez, de procurar sempre capitalizar qualquer ganho em relação aos resultados anteriores, vou relatá-los aqui, para que uma avaliação possa ser feita pelo Leoncio.

Os comentários serão feitos no próximo capítulo.

Para a primeira comparação utilizei divisão entre treino e teste com relação 9:1, em cima dos dados originais. Os resultados melhoraram após eu utilizar o artifício (comum em ml) de inflar os dados de treino (i.e.: duplicar as linhas e embaralhar). A melhora obtida será demonstrada no final.

Os três conjuntos de resultados a seguir foram gerados com semente randômica setada para zero, garantindo que diante do particionamento do conjunto entre treino e teste, sempre foram utilizados os mesmos dados.

a. Método máxima divisão

uso de memoria(GB): 0.4960746765136719

total de nós : 11524
total de folhas: 11525
total na árvore: 23049
total de mensagens em O: 10547
Tempo de processamento: 0:00:19.143506

a) método utilizado para escolher a melhor palavra: máxima divisão
utilizando os dados de treinamento (espera-se overfitting)
testadas: 10547
acertos: 10546
erros: 1
sem resposta: 0
acuidade: 99.9905186308903
Tempo de processamento: 0:00:32.169621

b) método utilizado para escolher a melhor palavra: máxima divisão
utilizando os dados de teste, 10 % dos dados originais separados aleatoriamente
testadas: 1172
acertos: 391
erros: 770
sem resposta: 11
acuidade: 33.67786391042205
Tempo de processamento: 0:00:34.057785

b. Método gini de uma resposta

uso de memoria(GB): 0.46793365478515625
total de nós : 8466
total de folhas: 8467
total na árvore: 16933
total de mensagens em O: 10547
Tempo de processamento: 0:01:37.356381

a) método utilizado para escolher a melhor palavra: gini de uma resposta
utilizando os dados de treinamento (espera-se overfitting)
testadas: 10547

acertos: 10546

erros: 1

sem resposta: 0

acuidade: 99.9905186308903

Tempo de processamento: 0:02:29.808409

b) método utilizado para escolher a melhor palavra: gini de uma resposta

utilizando os dados de teste, 10 % dos dados originais separados aleatoriamente

testadas: 1172

acertos: 585

erros: 586

sem resposta: 1

acuidade: 49.95730145175064

Tempo de processamento: 0:02:36.498980

c. Método do gini total

uso de memoria(GB): 0.4856109619140625

total de nós : 11864

total de folhas: 11865

total na árvore: 23729

total de mensagens em O: 10547

Tempo de processamento: 0:04:24.760246

a) método utilizado para escolher a melhor palavra: gini total

utilizando os dados de treinamento (espera-se overfitting)

testadas: 10547

acertos: 10546

erros: 1

sem resposta: 0

acuidade: 99.9905186308903

Tempo de processamento: 0:04:38.646693

b) método utilizado para escolher a melhor palavra: gini total

utilizando os dados de teste, 10 % dos dados originais separados aleatoriamente

testadas: 1172

acertos: 396

erros: 775

sem resposta: 1

acuidade: 33.81725021349274

Tempo de processamento: 0:04:40.674178

d. Método do gini de uma resposta, com duplicação do treino

uso de memoria(GB): 0.7586326599121094

total de nós : 7430

total de folhas: 7431

total na árvore: 14861

total de mensagens em O: 24609

Tempo de processamento: 0:04:06.542892

a) método utilizado para escolher a melhor palavra: gini de uma resposta

utilizando os dados de treinamento (espera-se overfitting)

observação: os dados de treino, após o particionamento, foram inflados 2 vezes.

testadas: 24609

acertos: 24606

erros: 3

sem resposta: 0

acuidade: 99.98780933804706

Tempo de processamento: 0:05:59.974699

b) método utilizado para escolher a melhor palavra: gini de uma resposta

utilizando os dados de teste 70.0 % dos dados originais separados aleatoriamente

testadas: 3516

acertos: 2977

erros: 539

sem resposta: 0

acuidade: 84.67007963594995

Tempo de processamento: 0:06:16.786425

IV. Conclusões:

Podemos comparar os melhores resultados obtidos.

Método	**Testando com os próprios dados de treinamento**	**Testando com dados de teste separados antes do teinamento**	**Tempo total (treinamento e teste).**
Gini de uma resposta	Acuidade: **99,98% ***	Acuidade: **49,95%**	0:02:36.498980
Gini de uma resposta com duplicação de dados de treino.	Acuidade: **99,98% ***	Acuidade: **84,67%**	0:06:16.786425

*** Existem nos dados de treino três mensagens com as mesmas palavras, duas com uma resposta, e a outra com uma resposta diferente. O programa atribui à folha a resposta com maior número de ocorrências, causando sempre um erro de resposta, que impede atingir os 100% de acertos esperado.**

O fenômeno mais notável nos resultados foi o efeito da duplicação dos dados de treino, elevando os acertos do gini de uma resposta de **49,95%** para **84,67%**. Creio que isso é devido ao fato de a expansão da árvore, no primeiro caso, parar em alguns nós antes de esgotar todas as palavras, por ter as mensagens esgotadas. Havendo mais mensagens, mesmo iguais, o processamento prosseguiu mais profundamente, melhorando a classificação. Por outro lado, isso aumentou de 2 para 6 minutos o tempo total de processamento e teste.

Acho que provavelmente se pudermos obter uma relação maior de linhas de mensagens contra o número de palavras, a classificação pode ficar melhor do que a obtida pela inflação das mensagens.

Todo o código desenvolvido está disponível em https://github.com/duodecimo/spk, na pasta python3.

Para facilitar testes como os acima existem parâmetros configuráveis na função main do programa qasDecTreeNR.py.

A razão da utilização da linguagem Python foi para utilizar a biblioteca Numpy, que oferece facilidades e eficiência para paralelizar operações com vetores, nos moldes do Octave ou do MathLab.

Na verdade, em minha opinião, com as dimensões do conjunto de treino fornecido, para acelerar o processamento (com um grande número de colunas) a vetorização é o método mais indicado.

Podemos tirar vantagem de aplicações como o Apache Spark quando tivermos um grande número de linhas para processar.

No meu caso, o que mais sacrificou minha máquina foi inflar o número de linhas: como só tenho 4G de memória, com grande parte normalmente ocupada, quanto tentei triplicar, a máquina começou a utilizar memória cache.

Existe uma outra biblioteca para o Python, Pandas, que também oferece vetorização de operações com matrizes. Para poucos dados (como os nossos), ela é mais lenta do que Numpy, mas, segundo pesquisei, ela tem desempenho melhor diante de quantidades maiores de operações. Além disso, e mais importante, ela pode ser utilizada diretamente no Apache Spark.