BACO
BAse de Co-Ocorrências

Luís Sarmento

BACO?

- O BACO é uma base de dados que guarda informação gerada a partir um processamento efectuado a um ou vários corpora.
- O objectivo:
  - Permitir pesquisar **rapidamente** grandes quantidades de texto com um determinado objectivo
  - Pensado de raiz para permitir uma vasta série de pesquisas que são **frequentes**

Motivação

- Há uma série de pesquisas são frequentes e que poderia ser interessante pré-processar / optimizar (especialmente para EC e afins)
- Temos cada vez mais corpora grandes (CP, WTP0#) que é util aproveitar, e que nos servem por um longo período de tempo.
- O aproveitamento requer experimentação
  - A experimentação só se torna possível se tivermos meios expeditos de pesquisa

Ferramentas

- Ferramentas habituais (ex: IMS-CWB) apesar de não exigirem trabalho de pré-processamento, são:
  - relativamente opacas
    - "não sabemos" o que fazem
  - Parecem limitadas para lidar com grandes corpora
    - BNC teve de ser dividido em 10 partes
  - Proprietárias
    - como distribuir corpora pesquisável (e multi-plataforma) codificado nessas ferramentas?
  - Não são flexiveis na produção de certos resultados ou no processamento de certas pesquisas:
    - n-gramas, colocações, co-ocorrências (ignorância minha?)...

SGBD's

- Os SGBD relacionais são:
  - uma tecnologia com quase 30 anos!
  - Muito difundida e bem compreendida
  - Eficiente, e conhecemos quando é que não são!
  - Há muitas soluções centradas em SGBD que permitem lidar com vastissimas quantidades de informação:
    - Então porque não corpora! (Mark Davies pioneiro?)
- Todas as SGBD permitem interface com linguagens de programação por isso devem ser vistas como uma adição: Perl + SQL

Esquema relacional

- A eficiência de pesquisa numa BD está essencialmente relacionada com o seu esquema relacional:
  - as tabelas e as suas relações
- Há muitos esquemas possiveis:
  - deve ser pensado em função dos objectivos
- Se bem pensado, poderemos armazenar um corpus numa BD para o poder pesquisar muito rapidamente:
- É necessário um pré-processamento intensivo:
  - No tratamento do corpus
  - Conversão do corpus para um esquema relacional apropriado
  - Na geração de indices na BD (cpu)
- Dá trabalho (o CQP não!) mas compensa se depois forem efectuadas muitas pesquisas.
  - alternativa seriam usar motores de pesquisa (Nuno Seco) mas não são optimizados para o tipo de pesquisas que poderemos querer fazer

O que é que eu posso querer pesquisar num corpus?

- Muita coisa (claro!):
  - Número de ocorrencias de uma palavra / n-grama
  - Que, quantos vizinhos tem uma palavra/sequência de palavras?
  - Quais as possiveis colocações em torno de uma palavra?
  - Que palavras aparecem entre outras duas?
  - Que, quantas frases respeitam um PL?
  - Que palavras ocorrem no contexto de outra?
  - ....
- Poderemos não querer saber directamente isso mas pode ser necessário para cálculo de uma medida...

O BACO

- Tenta fornecer métodos expeditos para cálculo destes valores!
- Pesquisas rápidas, ainda que:
  - antes de poder pesquisar tenha de
    - investir muito CPU
    - precise de muito disco
- A informação está armazenada num SGBD com muita redundância (por ex: o cqp não):
  - Esquema hiper-redundante
  - Mas temos normalmente muito mais disco
  - Se isso nos poupar tempo de pesquisa é optimo!
- SGBD: MySQL, teoricamente dos + rápidos

Uma abordagem ingénua

- Ponto de partida: corpus
- Pré-processamento:
  - atm + frs
  - "relacionalização"
- Esquema:
  - frase(id,[id_doc_origem,] texto,npals)
    - cada tuplo contem informação de uma frase
    - Poderemos ter uma tabela que guarda info do doc, de origem!
- Pesquisa de concordâncias:
  - Select * from frase where texto
    - Like "%sarmento%"
    - Regexp "sarmento"

Uma abordagem ingénua

- Complexidade: O(n)
- Mau!
- O MySQL tem de percorrer a totalidade das frases (tuplos da tabela) e tentar a regexp
- Mesmo assim mais rápido que o Perl a percorrer um ficheiro mas mesmo assim impraticável para corpus grandes (1 para 5)!:
  - 20% do WPT03 (15M f):
    - MySQL: entre 2 e 3 minutos (contagem apenas 30s)
    - Perl: mais de 10 minutos

Como será possível melhorar?

- Não explora a principal capacidade do um SGBD:
  - O índice interno B-tree
  - **Garante** tempos de pesquisa logaritmicos se a pesquisa for feita sobre um campo indexado:
    - **Encontramos um qq tuplo em log(N) (N = # de tuplos)**
- Mas não é possivel gerar um índice B-Tree directamente sobre um campo de texto (i.e. arbitráriamente grande)
- É necessário recorrer a uma técnica tradicional de IR:
  - Vamos criar o nosso próprio índice (invertido) que depois poderá ser indexado pelo mysql!

Usando IR

- Em IR existe uma estrutura chamada indíce invertido:
  - iv(atomo_id,documento_id,posicao)
- Tantos tuplos quanto palavras:
  - Normalmente, excluem-se tuplos de átomos demasiado frequentes (prep., artigos,etc): reduz a cerca de 50%
- Vamos adaptar para:
  - Iv'(atomo,frase_id,[posicao])
  - Posso gerar um indice B-Tree por iv'(atomo)
  - Passei a pesquisar iv em tempo log
  - Gero indice sobre frase(id)

Uma abordagem menos ingénua

- Novo esquema:
  - Frase(id,texto,npals) --- ibt(id)
  - Iv'(atomo,frase_id,[posicao]) --- ibt(atomo)
  - Pesquisar por "sarmento":
    - Procurar id das frases em iv para os quais o atomo seja "sarmento" --- tempo log (-> alguns segundos 8-15)
    - Para os id obtido procurar em frase os textos
    - tempo linear desprezável (cada frase << 0,1 s) porque a pesquisa é realizada sobre outro campo indexão
    - grande vantagem quando o match retorna poucos resultados
- Muito mais trabalho no pré-processamento.

Uma abordagem menos ingénua

- Mas e se eu quiser pesquisar "luis sarmento"?
  - Procurar no iv o id das frases para os quais o átomo seja "sarmento" --- tempo log (-> alguns segundos 8-15)
  - Se número resultados mais do que limite_max
    - Procurar no iv, o id das frases para os quais o átomo seja "luis" --- tempo log (-> alguns segundos...)
    - Intersectar conjunto: Perl – tempo desprezável usando uma hash
  - Para os id obtido procurar em frase os textos --- linear (< 1s)
  - Sobre as frases obtidas voltar a correr a regexp e retirar só os tuplos válidos
    - Perl: tempo línear
  - Para os dois passos anteriores, temos tempos lineares mas já trabalhamos sobre muito, muito menos frases!

Um pouco mais complicado

- Procurar um padrão como:
  - "PLN é uma disciplina de * porque"
  1. Excluir "é" "uma" "de" (demasiado frequentes)
  2. Excluir "*" e simbolos da regexp
  3. Pesquiso "PLN" no iv
  4. Pesquiso "disciplina" no iv
  5. Pesquiso "porque" no iv
  6. Interesectar os resultados
  7. Recolher as frases da tabela frases:
  8. Aplicar a regex inicial ao resultado
- Parece complicado de processar a querie
- Mas o tempo de pesquisa tende para logaritmico
  - é um ganho imenso sobre Perl simples (sobre CQP? Não sei)

Resumo desta parte

- Convertemos os corpus para formato relacional
  - Tabela de frases + ibt
  - índice invertido + ibt
  - Pré-processmento (algumas horas)
  - Podemos também gerar o dicionários
- Tornamos possível:
- Pesquisa sobre o texto usando regex:
  - Por fases com tempo logarítmicos
  - na totalidade do WPT03: estimativa 30s–1m

Mais capacidades de pesquisa

- Mas isto não resolve todos os problemas:
  - Quais a colocações da palavra p1?
  - Quais as palavras de comprimento n que podem aparecer entre p1 e p2?
- Podemos pesquisar as frases e calcular tudo com Perl
  - Ou podemos ter tudo já preparado podendo assim repetir as pesquisas
  - Em vez de "pesquisar" efectua-se uma "consulta"

Aumentar o nosso esquema

- Vamos criar uma tabelas de n_gramas
  - ainda que isso seja aumentar brutalmente a redundância dos dados!!
- Novas tabelas:
  - N_gramas_1(p1,n) → ibt(p1) : dicionário
  - N_gramas_2(p1,p2,n) → ibt(p1 e p2)
  - N_gramas_3(p1,p2,p3,n) → ibt(p1, p2 e p3)
  - N_gramas_4(p1,p2,p3,p4,n) → ibt(p1, p2, p3 e p4)
  - Mais que isto começa a complicar

Alterações a pré-processamento

- Durante o pre-processamento:
  - recolher o n-gramas
  - n=1 e n=2: o problema deve encaixar na RAM
  - n=3 e n=4: o problema não encaixa na RAM
    - Partir o problema em parte (ex: 10% corpus)
    - Carregar os tuplos para tabela temporária e agregar
    - Ou ordenação externa ou usar
- Lei de Zipf: há um "tecto" ao n-numero de n-gramas gerados (tuplos da tabela):
  - mesmo que o corpus aumenta para o dobro a possibilidade de novos n-grams vai sendo reduxida
    - quase só há alterações às contagens
- Gerar os ibt demora uns dias mas vai compensar!
- Pesquisas deste tipo ficam mais rápidas que no CQP

O que é que eu posso fazer mais agora!

1. Quais a palavras que normalmente aparecem depois de "célula" (possíveis modificadores):
   - Select p2,n from n_gramas_2 where p1 = "célula" [order by n]
   - Alguns segundos: 5 -10...
     - Usar IM para extrair pares mais interessantes.
     - Usar o dicionário gerado para para obter n(p1) e n(p2)
2. Pesquisar ngramas "o (\w+) é um"
   - Select p3,n from n_gramas_4 where p1 = "o" AND p3 = "é" AND p4="um"
   - Um bocado mais pesada mas ainda assim muito mais rápida que Perl ou CQP

"Leque"

- Quais a palavra semelhantes a "p2" usando janela de 1 + 1
  - Vizinhos = select p1,p2 from n_gramas_3 where p2="bola";
  - Já em Perl para cada par (v1,v3) de Vizinhos:
  - select p2 from n_gramas_3 where p1="v1" AND p3="v3";
  - Contar / Ponderar

Contagens

- Pesquisas que tenham como resultado apenas o valor (contagem, máximo, etc) são ainda mais rápidas:
  - MYSQL apenas mantém um registo
  - reduzida transferência interna de dados
  - Pode não ser necessário a leitura dos tuplos em disco (basta o índice)
- Quantos contextos vizinhos 1 + 2 pode ter a palavra professor:
  - select count(*) from n_gramas_4 where p2 = "professor" group by p1,p3,p4;
- Muito mais rápida que:
  - select * from n_gramas_4 where p2 = "professor" group by p1,p3,p4;
- Apesar da pesquisa (parecer) ser a mesma e (parecer) envolver os mesmo dados
- Há por isso vantagem em reduzir a informção de cada tuplo
- Para diminuir as leituras necessário (mais tuplos por leitura)
- Pode ser realizado com codificação numérica dos atomos usando um dicionário

Algo sobre a performance

- Grande parte do tempo da queries é gasto:
  - na leitura em disco
  - e na trasnferência de resultados
- Pesquisas que envolvam:
  - resultados com poucos tuplos (poucas leituras)
  - resultados que são apenas contagens
  - respostas "sim ou não" ou "existe algo?"
- são muito mais rápidas
- Saber transformar uma querie em qualquer uma das anteriores pode ser crucial...

Um último exemplo

- As operações de co-ocorrência anteriores ficaram limitada a uma janela pequena.
- Soluções:
  - Aumentar janela: n_gramas de 5 mais do que isso é começamos a gerar tabelas muito muito grande ~ 1 tuplo por palavra
  - Isso é possivel se se dividir o BACO por várias máquinas, cada uma com info de uma parte do corpus
- Mais simples:
  - Guardar qq co-ocorrencias num contexto de uma frase!

Um último exemplo

- Pré-processamento:
- Percorrer o corpus e para cada frase, guarder os pares de todas a palavras:
- Excluido algumas (artigos, prep, etc...)
  - O BACO é uma base que permite pesquisas
  - (BACO,base,1), (BACO, permite,1), (BACO, pesquisas,1) (Base, permite,1) (base, pesquisas,1)...
- Guardar tudo em:
  - Cooc (p1,p2,n) -> ibt(p1,p2,n)
- Pode ser necessário processar o problema por partes!

Um último exemplo

- Demonstração:
- Co-ocorrencas tiradas do 20% do WPT
- Filtragem prévia:
  - só perimitir formas que possam ser nome
  - Ignoramos os problemas dos homónimos
  - Mais de 14 milhões de tuplos!
  - Quase todas as pesquisas: < 1 segundo
  - Podemos aproveitar o efeito da cache

Conclusões

- É possível pesquisar corpora muito rapidamente usando um SGBD
  - Apesar de grande trabalho e tempo de pré-processamento
  - Apesa de ser necessário estabelecer estratégias de pesquisa próprias
- Vale a pena a criação de um recurso comum que seja versátil nas várias pesquisas que são frequentes