Uma Abordagem para Tratamento de Queries Nulas Unitárias em Sistemas de Recuperação de Informação

Giovanni Ap. S. Oliveira

Departamento de Ciência da Computação, Instituto de Matemática
Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, BR
giovanni@ufrj.br

Abstract. A correção ortográfica é essencial no processamento de texto e desempenha um papel importante no procedimento de reescrita de queries nulas, ou seja, que não retornam resultados. A dinamicidade da linguagem torna quase impossível manter um dicionário de correção atualizado com o dialeto coloquial. Nesse trabalho é apresentado um método para a correção de queries nulas unitárias que obtém seu dicionário a partir dos tokens presentes no corpus. Utiliza-se uma métrica de distância para encontrar termos presentes no dicionário que não distem mais que δ do termo a ser corrigido. Finalmente, classifica-se cada um dos candidatos baseado nos resultados gerados por um ranqueador probabilístico, escolhendo a sugestão que maximiza o índice dos resultados retornados.

Keywords: Correção ortográfica · Construção de dicionário · Geração de multi-candidatos · Modelo de ranqueamento

1 Introdução

O tratamento de queries nulas tem sido amplamente utilizado em sistemas de recuperação de informação[1][2]. Podemos analisar esse tratamento divindindo-o em dois componentes principais: o verificador e o corretor de escrita. Basicamente, o verificador de valida a query a nível de palavra, identificando possíveis erros gramaticais. Fica a cargo do corretor de escrita elaborar queries alternativas e selecionar a melhor opção. Na Figura 1 é possível observar a dinâmica de interação entre os estágios citados anteriormente. Adicionalmente, há um estágio final opcional que é constituído de um controlador de qualidade do resultado, onde são descritas condições mínimas para que umas sugestão seja considerada aceitável.

As técnicas empregadas para gerar as queries alternativas e ranqueá-las, entretanto, podem variar consideravelmente entre os sistemas de tratamento de queries nulas. As diferenças nas abordagens desses sistemas se dão principalmente devido a diferença das naturezas do conteúdo indexado. Muitos sistemas também podem apresentar estratégias e heurísticas para contemplar o tipo de entrada da query (e.g.: há sistemas dedicados a tratar queries nulas geradas

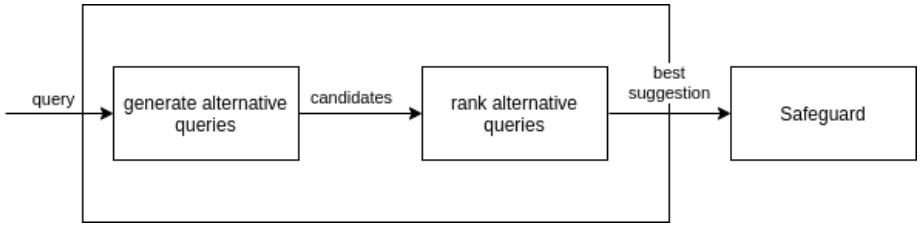

Fig. 1. Fluxograma representando o processo genérico de um gerador de sugestões

por reconhecimento de voz[2] ou digitadas através de teclados de dispositivos móveis[3]).

Nesse trabalho exploraremos estratégias voltadas para a correção de queries nulas compostas por uma única palavra. Diferente das principais referencias bibliográficas adotadas, o modelo de ranqueamento desse sistema não procura maximizar a similaridade textual ou semântica entre a query original e a modificada. Ao invés disso, usaremos os valores dos ranks gerados por um analisador probabilístico Okapi BM25 Tradicional [4] combinado com um analisador booleano para escolher a melhor sugestão de substituição. A geração de sugestões é realizada com base em um dicionário contendo os termos do corpus e a comparação de distância de Levenshtein entre termos da query e do dicionário.

Para validar a solução, utilizaremos como queries, palavras com erros ortográficos comuns na língua inglesa[7]. Com a palavra correta conhecida a priori, podemos computar a taxa das queries que foram modificadas de maneira correta, permitindo medir a eficiência do corretor.

2 Trabalhos Relacionados

A reescrita de queries é um tópico que tem sido amplamente estudado pela comunidade científica e pela indústria de Recuperação da Informação [1][2]. Naturalmente, há diferentes técnicas que podem ser concatenadas para compor um pipeline de correção [1]. Cada uma dessas técnicas pode representar um objeto de pesquisa em si e a composição do pipeline pode variar para contemplar os diferentes Sistema de Recuperação de Informação (SRIs).

Uma etapa comum a todos geradores de sugestão é a etapa de correção e geração de sugestões [1]. Dado um dicionário de termos, encontra-se o conjunto de correções possíveis. Nesse nível há uma escolha a ser feita: utilizar uma, apenas uma parte ou todas as sugestões geradas para constituir as alternativas. Essa etapa da análise também pode se dividir em mais de um estágio onde é executada uma varredura por tipos diferentes de erros (e.g.: erro único em palavra, erros em mais de uma palavra, erros por ausência ou presença indevida do carácter de espaço, que podem aparecer combinados).

Uma categoria a parte dos SRIs é constituída por técnicas de análise voltadas para o processamento de queries de e-commerce, onde geralmente as strings de busca são menores e mais sensíveis ao contexto [2]. Esse trabalho será validado

utilizando um conjunto de dados com nomes de livros e, desse modo, compartilha de características comuns a sistemas com esse propósito.

O tipo de entrada também pode exercer influência no comportamento do analisador e do corretor [3]. Um dos erros mais comuns em queries geradas por aplicações de busca, principalmente em smartphones, é chamado de fat finger typing error, que é quando o usuário desintencionalmente pressiona uma tecla adjacente à desejada[8]. A elaboração de alternativas nesse caso pode se valer da disposição física das teclas para gerar sugestões que corrijam a query. Há também conjuntos de técnicas para realizar tratamento específico de queries geradas por processamento de voz e fala[2]. Esse tipo de problema demanda a utilização de técnicas voltadas para similaridades fonéticas. Nesses casos um possível complicador é a necessidade de utilizar diferentes regras de correção baseadas no idioma e dialeto local.

Dentro dos buscadores destinados à recuperação de produtos de um e-commerce, é comum incluir critérios de geração e seleção de queries alternativas que fazem uso dos dados gerados pelo feedback da aplicação. Usuários logados podem ter sugestões direcionadas ao seu perfil e histórico de buscas e compras. Ações de outros usuários que resultaram em uma compra, por exemplo, também podem ser levados em consideração na formulação e análise dos resultados.

Outros passos do analisador podem conter estratégias que se valham das similaridades textuais. A análise de similaridade textual pode se dar através de modelos probabilísticos (como a distância de Jaccardi) ou determinísticos (como a distância de Levenshtein[6]). Estratégias como a indexação dos documentos em n-gramas e a análise probabilística de coocorrência de termos também podem ser empregadas nesse contexto. Quando há a disponibilidade ou possibilidade de extração de metadados dos elementos indexados, é possível explorar técnicas de medição de similaridade semântica para avaliar a proximidade da query original com a query sugerida.

Por fim, há um grande número de trabalhos na atualidade que buscam se valer de resultados obtidos através de análises baseadas em redes neurais ou, de forma mais geral, que fazem uso de algum recurso de aprendizado de máquina para formular e classificar queries alternativas.

3 Projeto e Implementação

3.1 Especificação em Alto Nível

Para iniciar a descrição da proposta de implementação, analisamos o processo em alto nível. A Figura 2 representa cada um dos estágios principais de processamento através de um fluxograma que contempla os diferentes tipos de entrada que podem ser submetidas na aplicação.

O primeiro tipo consiste nas queries que originalmente apresentam resultado não nulo. Nesse caso, a query passa por um processo de ranqueamento, filtragem (para selecionar os resultados da que serão mostrados ao usuário) e renderização do view que interfaceia diretamente com o usuário apresentando os resultados da

busca. Esse tratamento é comum aos demais tipos de query que serão descritos subsequentemente.

Caso a busca usando a query original tenha retornado resultado nulo, passamos para o próximo estágio onde é realizada uma verificação buscando por palavras não contidas no dicionário. Caso algum termo nessas condições seja identificado, buscamos no dicionário por outros termos com pequena distância textual. Feito isso para cada um dos erros, são geradas as queries alternativas baseadas em cada possível combinação dos termos sugeridos. Essas alternativas tem sua relevância medida e a sugestão com melhor valor é utilizada para substituir a query original, seguida dos mesmos passos de filtragem e renderização do view descrito no item anterior. Caso mesmo após a correção, o número de resultados retornados seja nulo, a query é encaminhada para ser tratada no próximo estágio.

Nesse caso, temos as queries nulas que foram encaminhadas pelo passo anterior, e tiveram sua ortografia corrigida, ou queries que originalmente apresentam uma grafia compatível com o dicionário, mas cuja execução trás um resultado nulo. Em qualquer um dos casos, o procedimento é o mesmo: para cada termo da query, levantamos o conjunto de palavras do dicionário que distam até determinado valor da palavra original. Assim como no caso das sugestões geradas por correções ortográficas, nesse passo são avaliadas todas as queries geradas pela combinação das sugestões. Se foi possível encontrar uma query superior entre as propostas, temos que essa query substitui a original e é encaminhada para ter seu resultado filtrado e renderizado. Caso nenhuma query alternativa traga resultados, o programa classifica a query original como definitivamente nula e retorna um resultado vazio.

3.2 Detalhamento dos Processos

Nessa subseção detalhamos cada um dos processos que compõem a implementação do sistema.

retrieve document scores A abordagem para calcular os valores de relevância de cada documento para uma dada query segue padrões bem conhecidos e utilizados em sistemas de recuperação de informação. Em um momento anterior ao da busca, a base de documentos é varrida para tokenização dos documentos e geração do índice invertido que fica pronto para ser parseado facilmente e utilizado pela aplicação principal. Usando o índice invertido, calcula-se o rank dos documentos através do modelo probabilístico Okapi BM25. Virtualmente, podemos considerar que temos um analisador booleano rodando em paralelo e que o resultado de documentos retornado pela busca é composto pela interseção entre os resultados obtidos em cada um dos analisadores. Na prática, o que fazemos é garantir que o conjunto de tokens da query está contido no conjunto de tokens de cada um dos documentos retornados. Nesse pipeline, o analisador booleano faz o papel de controlador de qualidade do resultado retornado.

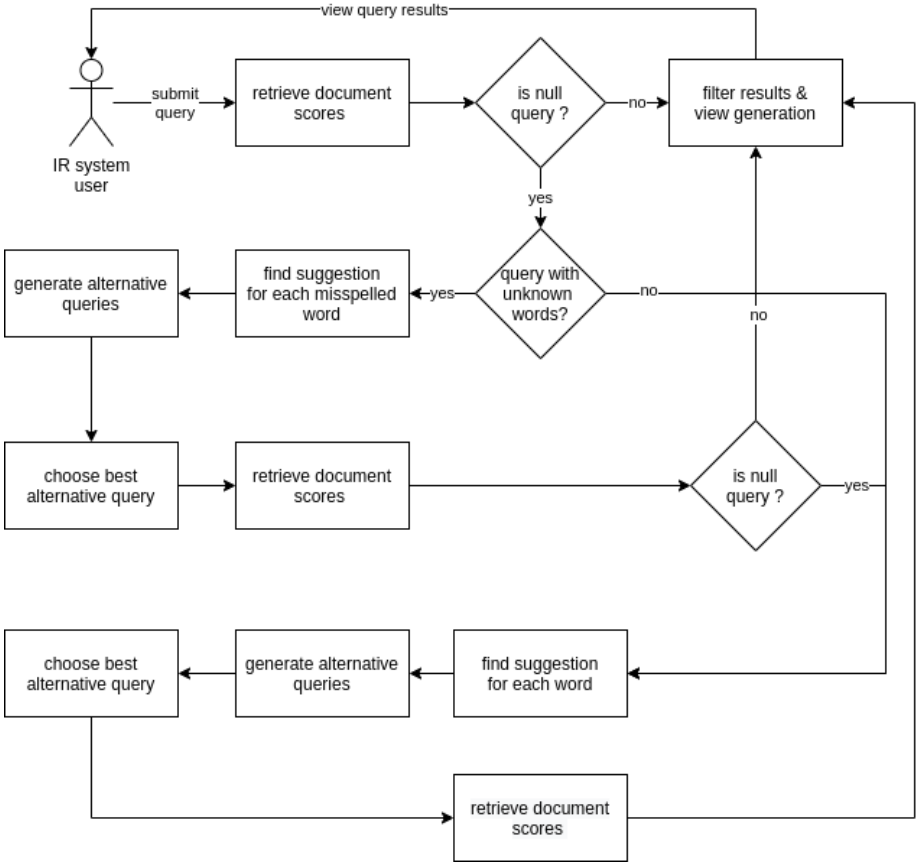

Fig. 2. Fluxograma representando os estágios de processamento de uma query, em alto nível.

find suggestion for each misspelled word Na etapa de pré-processamento descrita anteriormente, em adição ao índice invertido, salvamos também o dicionário que é o conjunto formado pela união dos tokens presentes em cada documento. Esse dicionário é utilizado para identificar erros na escrita. Nesse passo, uma palavra tem sua grafia considerada incorreta se não está presente no dicionário. Para cada palavra nessas condições, utilizamos uma métrica baseada na distância de Levenshtein para identificar possíveis sugestões de correção.

generate alternative queries Para cada uma das palavras corrigidas no passo anterior, há um conjunto de sugestões de substituição. As queries alternativas são obtidas através do resultado do produto cartesiano entre os conjuntos de sugestão, concatenados com cada uma das palavras não corrigidas da query origina..

choose best alternative A escolha da melhor entre um conjunto de queries alternativas pode se tornar tão complexa quanto se queira. Procuramos maximizar a métrica de acurácia do corretor. Nesse trabalho utilizaremos uma métrica que chamaremos de RA (relevância acumulada).

Definition 1. *Seja D o corpus sobre o qual opera o SRI e Q um conjunto de queries. Para uma dada query $q_i \in Q$ e uma função de ranqueamento Rank, temos:*

$Rank: D \to \mathbb{R}$, define a relevância acumulada da query q_i como:

$$RA(q_i) := \sum_{d \in D} Rank(d)$$

toma-se dentre as queries a que apresenta maior valor de RA. Escolheu-se essa métrica pela sua justificativa intuitiva e implementação simples.

find suggestion for each word Segue o mesmo princípio que a geração de sugestões para palavras com erro de escrita, mas nesse caso, para cada palavra da query, é levantado um conjunto de alternativas que deve incluir o termo original.

3.3 Implementação

Para implementar o modelo descrito na subseção anterior, utilizamos a linguagem de programação Python na versão 3.

Para gerar os tokens tanto dos documentos preprocessados do corpus, quanto da query em tempo de execução da aplicação, utilizamos o lexer disponível no pacote NLTK.

O resultado da fase de pré-processamento, contendo o índice invertido e o dicionário de termos, foi codificado em texto usando o formato JSON.

Para ranqueamento das queries utilizou-se o pacote BM25Okapi, combinado com um filtro baseado no modelo booleano.

A verificação de ortografia e a geração de sugestões com base na distância de Levenshtein é feita através do uso das funções disponível no pacote SpellCheck, configurado para operar com o dicionário gerado na fase de pré-processamento.

4 Experimentação

4.1 Objetivo

O objetivo dessa experimentação é medir a capacidade de correção de erros ortográficos por parte do analisador descrito na seção anterior. Como a fase de ranqueamento é realizada por dois modelos consolidados, a saber, modelo probabilístico Okapi BM25 e booleano, não será objetivo desse experimento avaliar o conjunto de documentos retornado, mas a corretude da query após ser processada.

4.2 Base de Testes

Mediante a impossibilidade de encontrar um conjunto de dados aberto, principalmente devido ao caráter comercial da informação, utilizou-se como base de testes a lista de erros comuns de escrita da língua inglesa, disponível na Wikipédia na versão formatada para máquinas[7].

A base consta com 4285 termos com erros ortográficos, ao grafia correta esperada. Em alguns casos, um termo errado pode endereçar a mais de uma possível correção. Como o conjunto constituído pelas correções com endereçamento único contém 4078 itens, entende-se que ele compõe massa de teste suficiente e será utilizado como base de testes ao invés do conjunto de dados original.

4.3 Execução

Para executar os testes de forma automatizada, utilizamos um script Python que varre a lista de erros ortográficos comuns, aplica no corretor e verifica se cada uma das queries foi modificada para o valor esperado (grafia correta do termo).

O número de queries corrigidas com sucesso nos permite extrair um parâmetro que mede a eficiência do corretor como a razão entre queries corrigidas com sucesso e número total de queries submetidas. Adicionalmente, cada correção diferente do esperado é salva em um arquivo de log que nos permitirá fazer uma análise qualitativa dos casos onde houve falha.

Também usamos um stemmer em um segundo teste para remover do grupo de erros os casos em que houve casamento entre o stem da query corrigida e o stem da query esperada.

4.4 Resultados Quantitativos

Das 4078 queries submetidas, observou-se que 2647 queries foram corrigidas conforme o esperado. Isso dá ao algoritmo uma acurácia de $\frac{2647}{4078} \approx 65\%$.

Quando comparamos os resultados obtidos com os esperados a nível semântico, através do uso do stemmer, há um aumento significativo na acurácia. De 2647, passamos para 2953 acertos que implica em uma taxa de $\frac{2953}{4078} \approx 72\%$, um aumento de 7% e relação à medição anterior.

4.5 Resultados Qualitativos

Dentre os erros de correção identificados na experimentação, identifica-se que os principais motivos que fizeram o algoritmo falhar foram:

1. A presença de termos no dicionário com distância de Levenshtein igual a do termo a ser corrigido, mas com semânticas distintas;
2. Queries incorretas e que cuja correção se traduz em mais de um termo.

5 Comparação de Resultados

Ao comparar os resultados obtidos na Subseção 4.4 com a principal referência comparada[1], nota-se que o método desenvolvido possui desempenho razoável.

Na Figura 3 são exibidos os percentuais de acerto para dois métodos desenvolvidos na referência (THD SC v1 e THC SC v2), e o corretor de escrita do Lucene (Lucene SC). Tomamos os valores da coluna "Overall" (que representam a média ponderada da acurácia para os tipos de erros ortográficos discretizados no artigo), para cada método temos, respectivamente, os valores de 77.71%, 80.32% e 65.26%.

Embora haja distorções ao compararmos os métodos com conjunto de dados, query e casos de teste distintos, nos apoiaremos nesses valores para realizar uma análise comparativa.

6 Conclusões

Conforme foi discorrido na Seção "Comparação de Resultados", a acurácia do corretor desenvolvido nesse trabalho é compatível com valores de corretores comerciais. Ainda que os valores tenham sido obtidos processando um conjunto de dados e buscas diferentes, acredita-se que a comparação seja uma estimativa válida.

Dado que nenhum método linguístico foi aplicado na lógica de geração de sugestões, entende-se que o nosso corretor seja capaz de operar com diversos idiomas, mediante ao carregamento de um corpus com documentos do idioma em questão.

Não aplicar nenhum método linguístico torna a implementação mais simples e geral. Entretanto, a não utilização desses métodos impede que sugestões baseadas em análise fonética, por exemplo, possam ser geradas como possíveis sugestões.

Pelo fato de o escopo desse trabalho se limitar à analise de queries unitárias, não fazemos uso de informações de correlação entre as palavras, que também poderia ser útil na geração de sugestões.

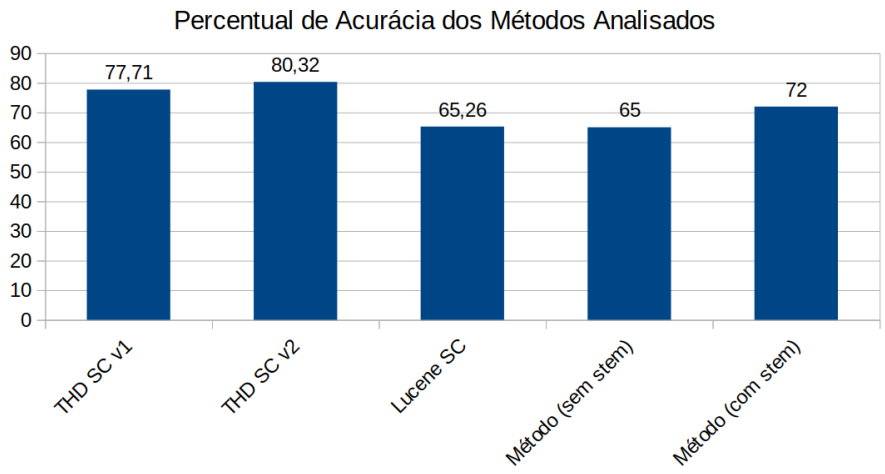

Fig. 3. Uma análise superficial nos permite identificar que a acurácia do modelo proposto nesse artigo é compatível com a acurácia de outros corretores experimentais (THDs) e comercial (Lucene SC).

O gerador de sugestões também apresentou limitações ao avaliarmos que uma parte dos erros de correção observados se deram pelo fato de que um termo pode ser corrigido para mais de um termo. Como o corretor não apresentou tratamento para esses casos, perdeu-se um número considerável de sugestões possíveis.

Por fim, o fator que mais prejudicou a correção foi a divergência entre a distância de Levenshtein e a distância intuitiva entre os termos. Para o propósito desse trabalho, necessita-se de uma métrica de distância entre termos mais apropriada.

7 Trabalhos Futuros

Para que o analisador seja capaz de atender a maior parte das aplicações reais, é necessário que seu escopo seja ampliado de queries unitárias para queries com mais de um termo.

A introdução de queries compostas torna ainda mais vantajosa a adoção de técnicas que explorem a correlação entre os tokens do corpus. Uma abordagem possível é indexar o conjunto de dados como n-gramas e utilizar os índices para gerar sugestões.

Um problema que é atenuado pela geração de n-gramas, mas que também precisa de um tratamento especial é a geração de sugestões que considerem erros gerados por ausência ou presença indevida do carácter de separação dos tokens (e.g.: carácter de espaço).

Com algum grau de dificuldade maior, poderíamos gerar um conjunto de heurísticas que se valham das similaridades fonéticas e na grafia dos termos corrigidos.

A performance pode ser melhorada pré-computando as sugestões possíveis para cada um dos termos do dicionário. Desse modo, só é necessário computar em tempo de execução as sugestões para termos não presentes no dicionário.

Se a aplicação tornar possível a obtenção de dados de feedback de usuários reais, é possível utilizar uma rede neural para aprimorar o sistema de recuperação de informação (usando os pesos da rede neural para, por exemplo, ponderar a influência de diversos métodos possíveis de ranqueamento de sugestão).

References

1. Wang, C., Zhao, R.: Multi-Candidate Ranking Algorithm Based Spell Correction. eCOM@SIGIR 2019
2. Gamzu, I., Haikin, M., Halabi, N.: Query Rewriting for Voice Shopping Null Queries. SIGIR@2020
3. Leiva, L.-A., Sahami A., Catalá A., Henze N., Schmidt A.: Error Auto-Correction Mechanisms on Tiny QWERTY Soft Keyboards
4. Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M.-M., Gatford, M.: Okapi at TREC-3. Overview of the Third Text REtrieval Conference (TREC-3)
5. Christopher, D.-M., Raghavan, P., Schütze, H.: An Introduction to Information Retrieval, Cambridge University Press, 2009, p. 233
6. Levenshtein, V.-I., "Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady. 10 (8): 707–710
7. Wikipedia:Lists of common misspellings/For machines. `https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings/For_machines`. Acessado em 01/11/2020 às 16:15
8. Fat Finger Error. `https://www.investopedia.com/terms/f/fat-finger-error.asp`. Acessado em 01/11/2020 às 16:20