Analisando Entidades Nomeadas em textos - TESI2

Carlos Eduardo S. Martins e Patricia S. Ghiraldelli

Novembro de 2016

Contents

1 Apresentação **3**

2 Pré processamento do texto **3**

3 Geração de entidades nomeadas **3**
3.1 Reconhecendo entidades nomeadas 3
3.1.1 A gramática 4
3.1.2 Aprimorando entidades 4
3.2 Criando equivalências de entidades 5
3.2.1 Escolhendo entidades chaves 5
3.2.2 Classificando entidades a partir das chaves 5
3.2.3 Juntando entidades parecidas 6

4 Obtendo relações entre entidades **6**

5 Extraindo informações dos textos **7**

1 Apresentação

O trabalho tem como entrada textos referentes aos episódios da série Game of Thrones disponibilizados pelo professor.

O objetivo é analisar o texto e extrair informações através de Processamento de Linguagem Natural.

2 Pré processamento do texto

Todos os arquivos dos episódios disponíveis vinham com diversas informações que não seriam úteis para a extração de dados sobre os mesmos. Dentre elas estavam os comentários dos diretores dos episódios sobre a atuação dos atores.

A fim de evitar processamento desnecessário foi desenvolvido um programa para limpar os texto e disponibilizar uma nova versão dos mesmos.

Tal versão possui apenas a parte "central" do documento, contendo a descrição do episódio.

O arquivo se chama *clean_text.py* e espera como parâmetro o caminho para a pasta raiz de todos os episódios. Como saída ele gera uma pasta *cleaned_episodes* com a estrutura das temporadas dos arquivos já limpos.

3 Geração de entidades nomeadas

A partir da pasta *cleaned_episodes* gerada, iremos analisar o texto de cada episódio e extrair o que considerarmos entidades nomeadas do mesmo. Ao final iremos possuir todas as entidades encontradas em todos os episódios.

A fim de contornar o problema de muitos espaços entre parágrafos, separamos o texto em linhas e para cada uma delas realizaremos um procedimento, que será detalhado passo-a-passo nas próximas subseções.

Utilizaremos a biblioteca **NLTK** para auxílio nos passos para se encontrar as entidades.

3.1 Reconhecendo entidades nomeadas

Inicialmente utilizaremos a funções de *tokenização* que irão delimitar as estruturas do texto, como frases(*nltk.sent_tokenize*) e palavras(*nltk.word_tokenize*). Esse passo é importante pois o texto original contém estruturas como quebra de linha e espaços em branco que não serão relevantes para nós.

Após isso, precisaremos distinguir as classes de cada palavra para futuramente podermos classificá-las como uma entidade nomeada para o texto. Com isso, utilizaremos a função *nltk.pos_tag* para determinar a classe de cada palavra, sendo ela um verbo, um determinador, um nome prórprio, entre outros.

Entretanto, o taggeamento genérico do **NLTK** não é capaz de distinguir o que é relevante para o universo específico de Game of Thrones. Por isso, criamos nossa própria gramática com regras que combinam tags do **NLTK** para criar nossas próprias tags específicas para nossos textos.

3.1.1 A gramática

Para classificar uma entidade nomeada usamos tais regras de gramática em ordem de classificação:

1. Nome próprio com pronomes possessivos para cobrir casos como "Night's Watch":

   NE_POS_NE: $\{< NNP \| NNPS > + < POS >< NNP \| NNPS >\}$

2. Sequência de nomes próprios seguido de preposição antes de mais nomes próprios, para cobrir casos como "Sir Eddard Stark of Winterfell" :

   NE_IN_NE: $\{< NNP \| NNPS > + < IN >< NNP \| NNPS > +\}$

3. Preposição seguida de um ou meis nomes próprios para casar com nomes de lugares:

   PLACE: $\{< IN > + < NNP \| NNPS > + < POS > * < NNP \| NNPS > *\}$

4. Sequência de nomes próprios no singular ou plural:

   SIMPLE_NE: $\{< NNP \| NNPS > +\}$

A partir dessas novas regras utlilizaremos o *nltk.regexParser*, juntamente com o *regex_parser.parse*, que nos retornará uma árvore sintática de todos os elementos da linha e suas respectivas tags, tornando possível acessar todos os que possuem as tags criadas em nossa gramática.

3.1.2 Aprimorando entidades

Durante a execução deste trabalho nos deparamos com diversos problemas que precisaram ser corrigidos para o melhor funcionamento do programa, que iremos expô-los a seguir.

- **Caracteres indesejados:** Percebemos que algumas entidades possuíam caracteres indesejados em sua estrutura, por exemplo: "Jon Snow(", ou então espaços em branco, ou até mesmo caracteres que o ascii não conseguia decodificar. Para isso, foi criada uma função chamada *clear_entity* que é responsável por remover tais caracteres da entidade encontrada.

- **Stop Words:** Percebemos que existem algumas palavras antes de nomes próprios que não acrescentam para quem a pessoa representa, como por exemplo: "Sir", "Lord", "of", "The". Portanto, criamos uma lista de palavras que não iremos considerar.

- **Entidades que começam com palavras de letra minúscula** têm tais palavras excluídas de sua formação, pois as mesmas não possuem valor de significado. Como exemplo tivemos: "on Greyjoy", "in Tiwyn". Dessa forma, excluimos as palavras iniciais e só ficamos com as de letras maiúsculas (Greyjoy e Tiwyn), pois realmente são entidades nomeadas.

- **Entidades com palavras minúsculas no meio:** Retiramos a de letra minúscula. Tal tratamento é feito pois percebemos que nesses casos as palavras do meio estavam separando duas entidades nomeadas. Por exemplo: "Daenerys from Robert Baratheon" e "Lannisters since Eddard Stark". Dessa forma excluimos as palavras do meio e separamos o resto em 2 entidades nomeadas diferentes.

3.2 Criando equivalências de entidades

Ao longo do trabalho foi apontada a necessidade de associarmos as entidades que pertencem a mesma denominação. A partir disso, estruturamos um dicionário onde a chave será a entidade com maior destaque e seus valores são as demais que fazem referência, como detalharemos nas próximas seções.

3.2.1 Escolhendo entidades chaves

Para iniciar a população das chaves do tal dicionário de entidades foram seguidas determinadas regras:

- **Não serão consideradas como entidades** aquelas que já foram classificadas como entidade chave do dicionário; palavras únicas que começam com letra minúscula (excluindo casos estranhos encontrados, como: "refore", "on", "nn"); palavras vazias e palavras que estão na lista a serem ignoradas (entidades pontuais que não significavam nada e removemos na força bruta, pois eram poucas. São elas: 'Hizdahr zo Loraq', 'Yezzan zo Qaggaz', 'Kraznys mo Nakloz').
- **Serão chaves do dicionário de entidades** aquelas que possuem 2 palavras e foram consideradas *SIMPLE_NE* pela nossa gramática definida na seção 3.1.1, pois esse cenário é o que possui maior chance de encontrarmos um nome e sobrenome.
- **Serão salvas para futura classificação** aquelas que não passaram em nenhuma regra anterior.

3.2.2 Classificando entidades a partir das chaves

Após todo o processamento dos textos da entrada, reconhecimento de todas as entidades nomeadas do texto e inicial população das chaves do dicionário de entidades, é necessário adicionar ao dicionário as demais entidades que não possuem classificação ainda.

Para isso, seguimos tais regras:

- **Serão classificadas como chaves obrigatoriamente** aquelas que começam com palavras que estão na lista de palavras iniciais a considerar. São elas: 'Battle', 'Master' e 'Castle'. Pois são entidades

nomeadas que possuem palavras que podem ser sobrenomes de outras, mas certamente não referenciam à entidade de uma pessoa ou família diretamente.

- **Serão classificadas como valores de chaves existentes** aquelas que possuirem similaridade com a chave. A função de verificação de similaridade entre 2 entidades irá verificar se uma está completamente contida na outra ou se a metade das palavras de uma +1 estão completamente contidas na outra.

  Isso nos deixa com alguns problemas, por exemplo: "Benjen Stark" como chave e "Star" como valor.
- **Para as demais ainda não classificadas** continuaremos a compará-las com as chaves do dicionário, agora que mais entidades foram adicionadas devido à primeira regra citada acima. Porém, se ainda não a adicionarmos como valor, a mesma será adicionada como chave do dicionário.

3.2.3 Juntando entidades parecidas

Percebemos que muitas entidades eram iguais, porém não estavam sendo consideradas similares por causa de erros de digitação ou até mesmo palavras no plural. Para contornar tal problema criamos uma função que concatena entidades similares.

O principal motor dessa função é a distância de *Levenshtein* que nos retorna um número de distância calculado pela quantidade de letras que fazem com que as palavras sejam diferentes uma da outra.

Como observado em nosso set de entidades a maior distância que poderia ocorrer derivada de erros de digitação ou plural poderia ser 1, como exemplo: "White walker" e "white walkers"; "Westeros" e "Westoros".

Entretanto, isso acarretaria problemas com entidades que certamente são diferentes ou a distância seria maior que 1, como exemplo: "Aemon Targaryen" e "Aegon Targaryen", "Daenarys" e "Daenerys Targaryen". Para tais casos (como eram poucos), foram adicionadas restrições explícitas para serem considerados similares ou não.

4 Obtendo relações entre entidades

A partir do reconhecimento das entidades detalhado na seção 3 gostaríamos de extrair as relações entre as entidades que aparecem nos textos, a fim de analisar como uma se comporta perante a outra e entender a dinâmica da śerie.

Para realizar tal tarefa utilizamos o raciocínio da seção 3.1.1 para gerarmos uma árvore sintática onde todas as relações possuem uma tag chamada **RELATION**.

Para classificar uma relação entre entidades usamos tal regra de gramática em ordem de classificação:

- Named Entity já marcada após o processamento da gramática descrita em 3.1.1.

  NE: $< NE\_POS\_NE \| NE\_IN\_NE \| PLACE \| SIMPLE\_NE >$

- Relação será tudo o que possui uma ou mais entidades nomeados com um ou mais verbos entre elas, podendo ou não ser precedido ou sucedido de determinadores e preposições ou conjunções.

  RELATION: $< NE > + < IN >? < DT >? < V.* > + < IN >? < DT >? < NE > +$

Utilizando as mesmas funções de regex e pos_tag obteremos uma árvore sintática com todas as relações tageadas. A partir disso é gerado um arquivo csv com as triplas de relações (entidade1, relação, entidade2).

5 Extraindo informações dos textos

Numa outra frente do trabalho gostaríamos de fazer consultas aos textos a fim de descobrir quais possuem mais informações sobre um determinado assunto.

Gostaríamos de, dada uma consulta, descobrir quais são os documentos dos episódios que são mais relevantes para a mesma. De forma mais simples, gostaríamos de poder dizer quais documentos melhor se relacionam com a consulta.

Para fazer isso utilizamos o valor **TF-IDF** (*Term Frequency - Inverse Document Frequency*), que é o cálculo da importância de um texto para uma certa palavra. No nosso caso, temos como entrada uma consulta que pode ter uma ou mais palavras, utilizamos a soma dos valores de **TF-IDF** de cada palavra para um certo texto.

Iniciamos nossa implementação desse cálculo percorrendo os textos dos episódios e contanto quantas vezes cada palavra ocorre. Utilizamos o **NLTK** para dividir o texto em frases e em palavras. Usamos também a mesma biblioteca para remover *stopwords*. Removemos também pontuações de cada uma das frases.

Dessa forma, nossa primeira matriz é formada por 9136 linhas e 55 colunas. Cada linha representa o número de vezes que certa palavra aparece em cada um dos episódios. As colunas são os respectivos documentos de texto dos episódios.

Em seguida calculamos o valor de *inverse document frequency* de cada palavra, levando em consideração a quantidade total de documentos e o número de documentos em que cada palavra ocorre. Com isso podemos calcular o tf-idf como nos foi ensinado em sala de aula. Na nossa representação cada linha desta última matriz corresponde a uma palavra. Os valores das colunas correspondem ao tf-idf da mesma em cada um dos documentos em que ela aparece.

Por fim bastou-se criar uma função responsável por receber uma consulta e de fato executar a busca pelos documentos que mais se assemelhavam à mesma. Esta função percorre uma lista com o nome de todos os documentos e, para cada um deles, soma o valor de tf-idf de cada uma das palavras da consulta. Montamos assim uma tupla onde o primeiro elemento é o documento e o segundo

a soma. Após ordenar os valores encontrados, retornamos como resultado da consulta os 5 documentos com maior score.

Não achamos necessário o uso da técnica SVD para a redução da dimensão da matriz, pois todo o processo se deu de forma computacionalmente simples. Isso se dá pelo fato de aplicarmos a consulta nos documentos pré-processados, o que reduziu significativamente o número de palavras totais.

Executando o programa para os textos na sua forma bruta a tarefa realmente se mostrou mais onerosa, demorando mais para terminar e gastando mais recursos do computador. A matriz teve dimensão 17777x55, quase o dobro de linhas do que vimos anteriormente.

Quanto ao resultado, as duas execuções retornaram quase os mesmos documentos e com valores diferentes. Como pode ser visto abaixo, ao executar para os textos sem pré-processamento obtivemos uma queda pela metade dos scores que também apareceram na rodada com textos pré-processados.

- Com textos pré-processados:
    1. season 1 - the wolf and the lion - 32.79494015554535
    2. season 1 - winter is coming - 22.391846304587254
    3. season 1 - you win or you die - 20.280242279769332
    4. season 1 - cripples, bastards and broken things - 19.601978029901094
    5. season 1 - lord snow - 15.531088807986613
- Com textos brutos:
    1. season 1 - the wolf and the lion - 13.808739477685481
    2. season 1 - winter is coming - 12.945693260330138
    3. season 1 - a golden crown - 11.423670182995943
    4. season 1 - the kingsroad - 11.033953431855917
    5. season 1 - you win or you die - 10.644236680715892