Relatório de Tópicos Especiais em Sistemas Inteligentes 2

Game Of Thrones: Entidades Nomeadas, Relações e TF-IDF

Aluizio Lima Filho / DRE:
Christian da Silva Cabral Cardozo / DRE: 111476126

Índice

1. Pré-processamento da base de dados

Neste trabalho, a base de dados utilizada foi um conjunto de textos sobre os episódios de Game Of Thrones. Antes de realizar as rotinas relacionadas ao processamento de linguagem natural, se faz necessário um pré-processamento na base original. Este procedimento tem o objetivo de tornar a base de dados um conjunto de entrada viável para os algoritmos posteriores. Com a análise da base de dados do trabalho, foram realizados alguns procedimentos simples que geraram como saída os seguintes arquivos:

- ***ep_name.txt***: nome do episódio
- ***ep_number.txt***: número do episódio
- ***deaths.txt***: mortes no episódio
- ***clean_text.txt***: texto limpo, contendo apenas o conteúdo essencial sobre cada episódio da base de dados

A seguir, estão listados os passos executados para a obtenção dos arquivos citados.

Passo 1: Nome e número do episódio

O nome do episódio é obtido de forma trivial a partir do nome do arquivo original do episódio. O número do episódio é obtido a partir do texto original, buscando pela primeira combinação da seguinte expressão regular:

$$\backslash b[Ee]pisode\ [0-9][0-9]?$$

Passo 2: Mortes do episódio

Em cada episódio temos informações semi-estruturadas de quais personagens morreram. Para encontrar esta lista, buscamos no texto original pelo índice do texto em que se inicia a seção de mortes. A seção de mortes pode iniciar com qualquer um dos seguintes termos: *'Deaths'*, '*DeathsEdit*' ou '*Deaths Edit*'. A primeira linha vazia após o início da seção marca o seu final. Seguindo esta estratégia, extraímos os dados sobre

mortes por episódio em um arquivo, onde cada linha na seção identificada representa um evento de morte.

Passo 3: Texto pré-processado

Este é o passo mais importante no tratamento da base de dados, que resultará no texto utilizado pelos algoritmos de PLN. O texto “limpo” é obtido após uma série de regras que são aplicadas ao texto original. Os seguintes passos são executados:

1. Corte do texto inicial: todo o conteúdo que é anterior ao termo “Contents[show]” é descartado.
2. Remoção de linhas inúteis: as chamadas “linhas inúteis” são aquelas que não possuem conteúdo semântico e podem ser descartadas. São removidas do texto as linhas iguais às seguintes strings:
    - 'Contents[show]'
    - 'Plot'
    - 'PlotEdit'
    - 'Plot Edit'
    - 'Synopsis'
    - 'SynopsisEdit'
    - 'Summary'
    - 'SummaryEdit'
3. Corte do texto final: a parte final do texto também é removida. Para definir a partir de qual índice o conteúdo deverá ser descartado, procura-se por um dos termos da seguinte listagem:
    - 'Recap'
    - 'RecapEdit'
    - 'Appearances'
    - 'AppearancesEdit'
    - 'Appearances Edit'

4. Ajustes finais: são realizados ajustes finais para o tratamento do texto como substituições de “\n” por “.” (ponto final), substituição de seguidos “ ” (espaços) por apenas um, substituição de seguidos “.”(pontos finais) por apenas um e remoção das ocorrências da string “Edit”.

Após os quatro passos acima serem realizados, o texto de saída é salvo em um arquivo “clean_text.txt” para uso posterior.

2. Reconhecimento de Entidades Nomeadas

Para a extração de entidades nomeadas, cada episódio é processado buscando as entidades de forma local (por episódio). Depois das entidades locais serem identificadas, elas são adicionadas à um dicionário global de entidades já reconhecidas. O dicionário global é iniciado vazio e é preenchido a cada passo de interação nos episódios.

Além de um dicionário global de entidades, neste passo é gerado um texto “taggeado”(tagged_text.txt) para cada episódio, onde cada entidade nomeada identificada é marcada com uma tag <entity></entity>. A seguir, estão explicados os passos para a obtenção destes resultados iterando em cada arquivo.

Passo 1: Tokenizar por sentenças

Tokenizar o texto por sentenças utilizando o nltk.sent_tokenizer.

Passo 2: Análise da sentença

Para cada sentença, é analisado seu conteúdo em busca de entidades nomeadas. Para isto, a sentença é tokenizada usando o nltk.word_tokenize(). Depois, utilizamos um POS Tagger para taggear os tokens da sentença. O POS Tagger utilizado é fornecido pelo método nltk.pos_tag(). Por fim, é utilizado o nltk.ne_chunk() para a marcação de possíveis entidades nomeadas.

Passo 3: Extração das entidades da sentença

Dado o vetor de resultado do passo anterior, iteramos nele em busca das entidades nomeadas. O ne_chunk retorna um vetor onde cada elemento é do tipo:

- Tupla: caso o token não seja uma entidade nomeada
- nltk.tree.Tree: caso uma sequência de tokens seja uma entidade nomeada

Ao percorrer os elementos do vetor de resutado do ne_chunk, identificamos entidades nomeadas e atribuímos um id à elas. Uma entidade nomeada é identificada quando se encaixa em alguma das seguintes regras:

- Regra 1 (Caso geral): *Entidade Nomeada + (Entidade Nomeada | NNP)**
  - Onde NNP é uma classificação do POS Tagger.
  - A regra busca por uma entidade nomeada e possíveis sequências do mesmo tipo ou do tipo NNP.
  - Exceções: casos em que não são entidades nomeadas e por isso a regra 1 deixa de ser contemplada:
    - *'–', '–', '—', '[', '', 'imp', 'beyond', 'house', 'ser ser'*
- Regra 2 (Caso especial): *Entidade Nomeada + "'s" + "Watch"*
  - A regra busca pelo caso específico de uma string igual à "Night's Watch". Esta string não é contemplada pela regra 1

Passo 4: Adicionar no dicionário global de entidades

Quando temos uma lista de entidades nomeadas identificadas localmente por episódio, devemos verificar o dicionário global de entidades para cada uma delas. Devemos decidir se cada entidade identificada do episódio deverá ser agregada à alguma já existente no dicionário global (casos de correferência), ou se ela deverá ser uma nova entrada. Para isso, buscamos entidades candidatas no dicionário global. Entidades candidatas são aquelas que:

- Já possuem o termo em sua lista.

    - Exemplo:
        - Entidade local: “Jon”
        - Entidade candidata no dicionário global: [“Jon Snow”, “Jon”]
- Possui uma similaridade média entre a entidade local e seus termos maior ou igual à 0.7.
    - A similaridade é calculada da seguinte forma:

*sim(str1, str2) = Pertinencia(str1, str2)*0.4 + JaroWinkler(str1, str2)*0.6*

    - *JaroWinkler* se refere ao algoritmo de distância de Jaro-Winkler
    - *Pertinencia* se refere à um índice criado para balancear se uma string está contida na outra.
        - A pertinência será igual à 0 (zero) se nenhuma das strings é substring da outra.
        - A pertinência será igual à 0.5 (um) se uma das strings for substring da outra onde a substring é exatamente um termo e está presente no final da string maior (entidades com sobrenomes e sobrenomes isolados). Esse valor ajuda a balancear casos como Tywin Lannister e Lannister, Robert Baratheon e Baratheons.
        - A Pertinência será igual à 1 caso contrário. Esse valor ajuda a valorizar casos como Jon e Jon Snow, Arya e Arya Stark, etc.

Dado a lista de entidades candidatas, caso a lista esteja vazia, a entidade local será uma nova entrada no dicionário global de entidades. Caso haja entidades candidatas, aquela que possui maior frequência até o momento é escolhida para adicionar a entidade local à sua lista de termos.

Ao final do processamento de todos os episódios, temos uma lista de entidades em um dicionário global onde a chave é o id da entidade e o valor um objeto Entidade Nomeada que possui um id da classe pai. Estes resultados são salvos em um arquivo “entities.csv” e possui a seguinte estrutura: id, id_entidade_pai, termos. Além disso,

como já foi mencionado, para cada episódio é gerado um texto taggeado marcando as entidades nomeadas com a tag <entity></entity>, com os atributos id (id da entidade) e class (classe da entidade). O atributo class pode assumir os valores NE (Named Entity genérica) ou HSE (House - Casa ou Clã). Nesta seção foram identificadas 5431 entidades, que são representadas como 756 entidades distintas no dicionário global.

3. Extração de Relações

Com as entidades nomeadas devidamente identificadas pela seção anterior, podemos buscar relações entre elas nos textos. Para isso, usamos os textos taggeados previamente salvos e as classificações gramaticais dos termos que não são entidades nomeadas fornecidos pelo POS Tagger do NLTK.

As relações são identificadas a partir de 3 regras, onde o conteúdo entre as duas entidades forma uma relação:

- Regra 1: Entidade + IN + DT + Entidade
  - IN (preposição) e DT (determinante) são classificações gramaticais do POS Tagger.
  - Relação: IN + DT
  - Exemplo: Eddard on the Kingsroad
- Regra 2: Entidade + Sequência de Verbos + Entidade
  - Relação: Sequência de Verbos (sem vírgula entre eles)
  - Exemplo 1: Robert Baratheon arrives Winterfell
  - Exemplo 2: Will return to warn Castle Black
- Regra 3: Entidade + Classificação Central de 1 termo + Entidade
  - Relação: Classificação Central de 1 termo
  - Exemplo: Daenerys Targaryen marries Khal Drogo
    - Marries não é classificado como verbo pelo POS Tagger

Estas regras geram um conjunto de triplas que representam as relações e são salvas em um arquivo chamado “relationships.csv”. As relações distintas encontradas e

suas frequências são salvas em um arquivo chamado “keys_relationships.csv”. Foram encontradas 6595 triplas de relações e 2806 relações distintas.

4. TF-IDF

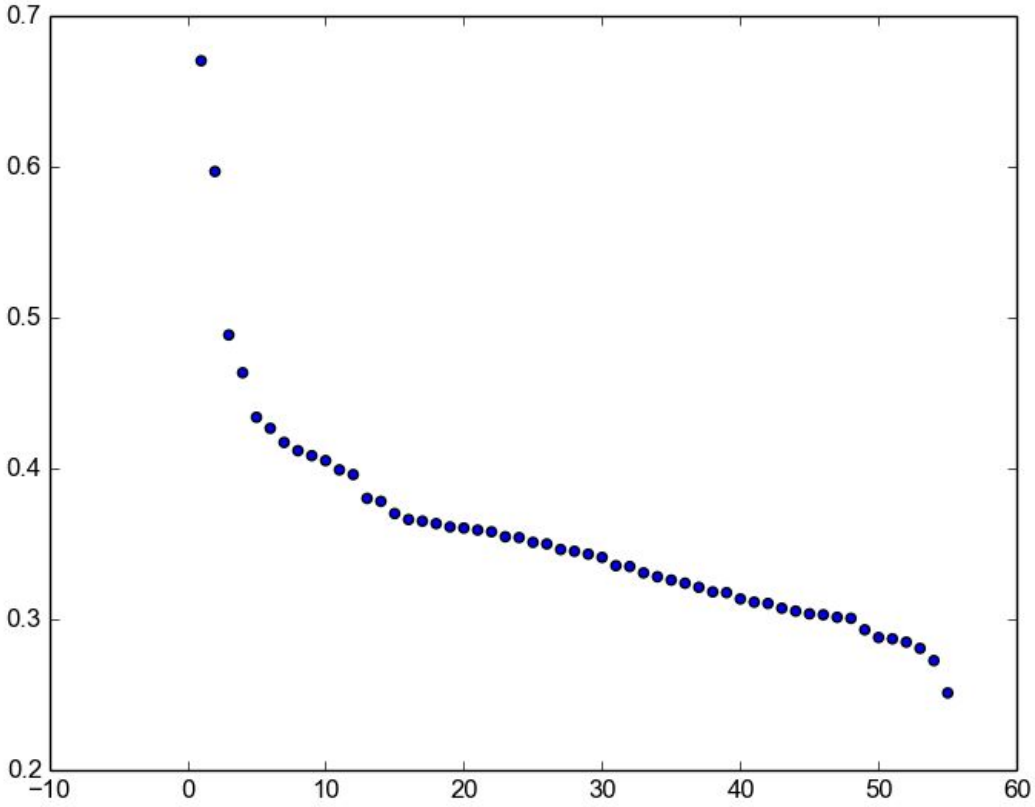
Para o cálculo do TF-IDF, optamos por implementar o algoritmo. Para calcular o TF-IDF, primeiro definimos o conjunto de tokens de cada episódio. Os tokens que são entidades nomeadas são identificadas como uma string “__id__num”, onde *num* é o identificador da entidade. Os tokens que não são entidades nomeadas, sofrem um processo de normalização:

1. Remove-se pontuações e mantém-se apenas dígitos e letras
2. Remove-se as Stop Words. A lista de Stop Words é a mesma utilizada pelo Scikit-learn para a língua inglesa[1].
3. Por fim, estes tokens passam pelo processo de Stemming. Utilizamos o Porter Stemmer do NLTK.

Ao final desses passos, foram encontrados no total 10424 tokens (onde 756 são entidades nomeadas). Com os tokens identificados, podemos calcular os valores de TF-IDF de cada token para cada documento (episódio). Cada episódio possui um dicionário TF-IDF que mantém apenas os tokens presentes no texto do episódio e seu score, reduzindo o espaço em disco ocupado. Estes dicionários são salvos em arquivo como “tfidf.csv” para cada pasta de episódio.

Em adicional, implementamos como seria a redução de dimensionalidade da matriz TF-IDF completa utilizando o método de SVD. A matriz completa de TF-IDF é uma matriz de dimensão 10424 x 55. Para essa matriz temos um total de 573320 células, onde 284825 são valores iguais à zero (uma matriz esparsa). Na implementação do SVD, escolhemos um valor K=30, onde K é o número de corte da decomposição da matriz. Este número foi escolhido baseado no gráfico K (abscissa) x Sigma (ordenada) a seguir e atrelado à experimentos empíricos:

[1] https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/stop_words.py



Para fins de comparação, a busca de documentos relevantes dado um conjunto de termos (query) foi implementada das duas formas: usando o resultado do SVD e pela matriz completa do TF-IDF.

Antes de buscar o documento mais relevante para a consulta, realizamos um pré-processamento na query. Primeiro, buscamos entidades nomeadas na query, substituindo suas ocorrências por "__id__num", onde *num* é o identificador da entidade nomeada. Em caso de empate, como por exemplo "Jon" poder pertencer às entidades "Jon Snow" e "Jon Arryn", as duas entidades são levadas em consideração na query para o cálculo, como se na prática o usuário houvesse informado as duas entidades nomeadas. Após marcar as entidades nomeadas na query, os tokens que sobram passam por um processo de normalização semelhante ao realizado para definir os tokens da tabela de TF-IDF: remoção de pontuação, transformação de todas as letras para minúsculas e Stemming usando o Porter Stemmer. O cálculo da similaridade é

feito utilizando a medida de cosseno entre os vetores de termos do documento e termos da query.

- Exemplo:

Suponha que a query seja igual à: “Battle of HARDHOME”. Esta batalha se dá no EP 08 da SE 05, e temos como resultado:

# Rank	TF-IDF Episode (Score)	SVD Episode (Score)
1	SE5 EP8: 0.87579589937	SE5 EP8: 0.156036884294
2	SE3 EP1: 0.932694443068	SE5 EP5: 0.245279537001
3	SE3 EP3: 0.947773983468	SE3 EP1: 0.246801204662
4	SE5 EP10: 0.958392269231	SE5 EP10: 0.379971556554
5	SE5 EP5: 0.966069487072	SE3 EP3: 0.379994993036

5. Código-fonte

O código-fonte do trabalho pode ser encontrado em:

- https://github.com/NLP-TESI/NamedEntitiesTESI

Como executar o programa:

- python3 main.py [opções]

Opções (devem ser declaradas em ordem):

- preprocess: pré-processa o texto
- find_ne: extrai entidades nomeadas e relações
- tfidf: calcular o tfidf
- query: ativa o modo query
- svd: ativa algoritmo SVD no modo query

Exemplo:

- python3 main.py preprocess find_ne tfidf query svd