Tutorial - WNN Short Doc Classifier

Introdução

Neste projeto, tratamos do problema de analisar um fluxo de tweets e filtrar aqueles que mencionam alguma entidade de interesse (empresa, pessoa, produto, etc.). O objetivo é monitorar tudo que é dito sobre uma entidade no Twitter. Algumas entidades são ambiguas pois outras entidades possuem nome simular ou mesmo idêntico. Por exemplo, considere o time de futebol *São Paulo Futebol Clube (SPFC)*. Em muitos tweets, o autor utiliza apenas o termo *São Paulo* para se referir ao time de futebol. Mas *São Paulo* é o nome de uma cidade, de um estado e de um apóstolo católico.

Tradicionalmente, este problema é resolvido manualmente através de uma espécie de "engenharia" de buscas, onde o usuário cria, iterativamente, restrições para tornar as buscas cada vez mais específicas. Usando novamente o exemplo acima do SPFC, o usuário poderia especificar que tweets com o termo *Dória* devem ser desconsiderados, pois provavelmente são relacionados à cidade de São Paulo, cujo prefeito se chama *João Dória*. Esta abordagem manual apresenta duas limitações relevantes: esforço alto e qualidade limitada. Para entidades *muito ambíguas*, o usuário precisa iterar muitas vezes até chegar a um resultado razoável. Além disto, depois de várias iterações, o resultado final nem sempre é satisfatório pois algumas entidades são muito difíceis de desambiguar.

O principal objetivo deste projeto é desenvolver um algoritmo baseado em *aprendizado de máquina* (AM) para resolver este problema de uma maneira mais eficaz e mais eficiente do que um ser humano. O algoritmo tem o propósito de classificar tweets de acordo com um contexto pré-definido, isto é, se o tweet referencia ou não uma entidade de interesse. Será explicado a seguir o passo a passo de utilização do algoritmo desenvolvido. O algoritmo foi avaliado em quatro entidades definidas pela equipe. Foram usados três clubes de futebol com nomes ambíguos: *São Paulo*, *Santos* e *Bahia*; e ainda a série de televisão Supernatural, a qual frequentemente é referenciada através de seus personagens principais cujos nomes são *Sam* e *Dean*.

Inicialmente, foram coletados vários tweets para cada entidade utilizando termos genéricos, a saber: "São Paulo", "Santos", "Bahia", "Sam" e "Dean". Os dois últimos termos são referentes à série Supernatural, enquanto os outros são relacionados aos respectivos clubes de futebol. Alguns voluntários anotaram manualmente diversos destes tweets com respeito à menção das entidades consideradas. O algoritmo de aprendizado de máquina necessita de uma quantidade de dados anotados (como relevante e não relevante) para aprender o padrão dos tweets relevantes à entidade em questão. Este processo de aprendizado é chamado *treinamento* e produz um modelo matemático que permite classificar tweets automaticamente como relevante e não relevante. Diversos experimentos foram realizados com o objetivo de desenvolver uma metodologia que permita o treinamento de um modelo eficaz usando o mínimo possível de dados anotados.

A metodologia desenvolvida inclui uma opção de pré-treinamento do modelo usando um conjunto de tweets anotados *automaticamente*. Neste caso, o objetivo é obter automaticamente um grande conjunto de tweets relevantes à entidade de interesse (exemplos positivos) e outro conjunto não relevante à entidade (exemplos negativos). Por exemplo, no caso do SPFC, pode-se usar o fluxo de tweets da conta oficial do clube (e de outras contas não oficiais mas específicas do SPFC) como fonte de exemplos positivos. Exemplos negativos podem ser gerados por buscas genéricas no Twitter. Mesmo que estas buscas retornem alguns tweets relevantes à entidade, a proporção destes será ínfima. Desta forma, o algoritmo pode ser pré-treinado nestes dados e o modelo aprendido pode ser salvo e usado para inicializar o treinamento posterior que usa os dados anotados manualmente (que estão disponíveis em quantidade bem menor).

Para realizar a validação dos quatro modelos treinados (um para cada entidade considerada), aplicamos a técnica de *holdout*. Esta técnica permite estimarmos com boa precisão a qualidade dos modelos treinados quando forem aplicados em dados desconhecidos. No holdout, os datasets anotados manualmente são divididos em três partes: treino, validação e teste. A partir dos dados originais, 10% são separados para validação, 10% para teste e os outros 80% para treino. Nos arquivos de exemplo fornecidos com este tutorial, a partição correspondente é indicada no nome de cada arquivo. O termo **train** é usado para as partições de treino, **dev** para validação e **test** para teste.

Código Fonte e Arquivos de Exemplo

O código necessário para aplicar o algoritmo desenvolvido está disponível no projeto **pln-ufms** no Bitbucket na tag **XXX**.

Na seção Download do repositório pln-ufms no Bitbucket, existe um arquivo compactado com exemplos de arquivos de entrada e saída para as quatro entidades tratadas neste projeto. O propósito destes arquivos é exemplificar o formato de entrada e saída dos scripts do projeto, facilitando a reprodução dos resultados obtidos e, principalmente, a aplicação do algoritmo proposto em novas entidades. Nos exemplos apresentados, usaremos o termo **`$examples`** para indicar o diretório base onde estão os arquivos de exemplo e o termo **`$src`** para indicar o diretório raiz do projeto **pln-ufms**.

Pré-Processamento

Antes de utilizar os tweets anotados manualmente ou automaticamente para treinar os modelos, é necessário aplicar um pré-processamento que normaliza alguns aspectos destes textos. Nesta seção, descrevemos a utilização do script de pré-processamento.

O script **`$src/experiments/shortdoc_class/pre_processing.py`** é responsável pelo pré-processamento. Sua execução é exemplificada pelo comando abaixo. O script recebe dois argumentos em linha de comando: o arquivo de entrada, contendo o texto a ser pré-processado; e o arquivo de saída, onde o texto pré-processado será salvo. Veja o exemplo abaixo para a entidade *São Paulo*.

```
python $src/experiments/shortdoc_class/pre_processing.py \
       $examples/SaoPaulo/Unchanged/Input/testSetSaoPaulo.txt \
       $examples/SaoPaulo/Unchanged/Input/testSetSaoPauloPreProcessed.txt
```

Para clarificar o pré-processamento, observe abaixo dois tweets antes do pré-processamento. O formato esperado de entrada é um tweet por linha, onde cada linha contém o texto do tweet seguido por caractere de tabulação e a classe de relevância (**`Sim`** para relevante e **`Não`** para não relevante).

```
BRASILEIRO SUB-20: Começou o 2º tempo de Coritiba x São Paulo #VamosSãoPaulo  Sim
Pela semifinal do Brasileiro, Sub-20 enfrenta Coritiba #VamosSãoPaulo https://t.co/G6Yg8kIU0U  Sim
```

Após o pré-processamento, temos a saída apresentada abaixo que segue o mesmo formato da entrada.

```
brasileiro sub - 00 : começou o 0º tempo de coritiba x são paulo ##HASHTAG##  Sim
pela semifinal do brasileiro , sub - 00 enfrenta coritiba ##HASHTAG## ##LINK##  Sim
```

É possível observar diversas alterações no texto de entrada. São elas:

- Os token (palavra e sinal de pontuação) são separados por espaço.
- Links são substituídos pelo token artificial **`##LINK##`**.
- Hashtags são substituídas pelo token artificial **`##HASHTAG##`**.
- Cada dígito é transformado no caractere 0 (zero).
- Todas as letras são substituídas por sua correspondente em minúsculo.

Utilização do Algoritmo

Antes de prosseguir, garanta que as dependências a seguir existam na máquina que rodará a ferramenta:

- **Numpy**
- **Theano**

Em seguida, serão descritos os passos para rodar a ferramenta sem pré-treino, com exemplo:

Inicialmente modificaremos os parâmetros do .json responsável por configurar a rede do nosso exemplo, presente no arquivo **`$examples/SaoPaulo/Unchanged/Input/wnn.json`**.

Os parâmetros de *train*, *test* e *dev* devem ser modificados de acordo com a localização destes em seus arquivos. O único parâmetro além desse a ser calibrado para obtimento de um melhor resultado, é o da Learning Rate, onde costumamos variar entre 0.02, 0.015, 0.01 e 0.005. No caso de nosso exemplo, a melhor learning rate obtida em testes foi a de 0.02, então manteremos esta para esse exemplo.

Tendo modificado o .json, iremos executar o comando em Python,ou em uma IDE,se preferir, onde executaremos a classe **`$src/experiments/shortdoc_class/wnn_shortdoc_class.py`** , que é a ferramenta em si.

Se o fizermos em um terminal,basta rodar a classe, e passar o .json do exemplo como parâmetro:

```
python $src/experiments/shortdoc_class/wnn_shortdoc_class.py \
       $examples/SaoPaulo/Unchanged/Input/wnn.json
```

Caso esteja utilizando uma IDE, configure na execução da classe e passe o path do .json de exemplo como parâmetro

Por fim, o resultado será gerado para analise posterior, no terminal, ou no console da IDE. Um exemplo de saída da ferramenta pode ser visto em **`$examples/SaoPaulo/Unchanged/Output/SaoPaulo.log`**

Utilização do Algoritmo com Pré-Treinamento

Nesta parte, teremos duas sub-seções , uma onde pré-treinaremos a rede, e a outra onde utilizaremos a rede pré-treinada para classificar os exemplos antigos.

Primeiramente, para pré-treinarmos a rede, além dos parâmetros já modificados previamente(incluindo os paths de acordo com a maquina que estará sendo rodada a ferramenta), precisamos de quatro novos parâmetros. Eles são :

- **"save_wordEmbedding"**
- **"save_conv"**
- **"save_hiddenLayer"**,
- **"save_softmax"**.

Em cada um deles, será passado o path com o local onde será salvo os dados da rede pré-treinada que carregaremos posteriormente. (Nota: apenas o primeiro parâmetro é salvo como um arquivo .txt, os outros três devem ser salvos como .npy).

Em seguida, como anteriormente, basta rodar a ferramenta em **`$src/experiments/shortdoc_class/wnn_shortdoc_class.py`** e passar o .json modificado como parâmetro. Segue exemplo:

```
python $examples/experiments/shortdoc_class/wnn_shortdoc_class.py \
       $examples/SaoPaulo/PreTrained/PreTrainBaseline/wnnToPreTrain.json
```

Os resultados serão então gerados a cada execução, e em nossos exemplos, irão para a pasta **`$examples/SaoPaulo/PreTrained/Savings/`**

Tendo obtido nossas redes pré-treinadas , teremos agora a segunda parte onde as carregaremos e as utilizaremos para os antigos datasets. Para isso, novamente iremos modificar o .json. Agora, modificamos novamente os paths, com antes, a learning rate, da mesma forma , adicionamos três novos parâmetros:

- **"load_conv"**
- **"load_hiddenLayer"**
- **"load_softmax"**

Um outro pequeno detalhe é que o parâmetro **"word_embedding"** passa a usar o **WordVector** salvo na etapa anterior , no nosso exemplo sendo **`$examples/SaoPaulo/PreTrained/Savings/SaoPauloWEmbedd.txt`** . A partir daqui , os parâmetros de save se tornam opcionais, caso seja de desejo salvar as redes para avaliações posteriores. No nosso exemplo, iremos retirá-los.

Seguindo adiante, os passos serão os mesmos do exemplo sem rede pré-treinada. Basta rodar a ferramenta passando o novo .json com os parâmetros **loads** citados acima e o de **wordEmbedding** modificados como parâmetro :

```
python $examples/experiments/shortdoc_class/wnn_shortdoc_class.py \
       $examples/SaoPaulo/PreTrained/Jsons/1000/wnnPreTrained.json
```

Por fim , o resultado será novamente gerado no terminal ou no console da IDE. Um exemplo de saída pode ser visto em `$examples/SaoPaulo/PreTrained/Output/1000/SaoPauloPT.log`

Script de Predição

Por fim, para fins de testar as redes treinadas, está disponível um script de predição na pasta **`$src/experiments/shortdoc_class/ShortDocEmbedding.py`**

Sua execução depende de dois argumentos. Um .json de configuração onde serão passados as redes treinadas no capítulo anterior.Novamente, os parâmetros que necessitam ser modificados são os quatro que foram salvos no pré-treinamento:

- **word_embedding**
- **load_conv**
- **load_hiddenLayer**
- **load_softmax**

Feito isso, o outro arquivo que será passado por parâmetro na execução do script é o arquivo de entrada, cujo conteúdo deve ser os tweets a serem preditos separados por uma quebra de linha cada.

Em seguida, executa-se o script com o seguinte comando:

```
python $src/experiments/shortdoc_class/ShortDocEmbedding.py \
       $examples/SaoPaulo/Script/wnnPrediction.json \
       $examples/SaoPaulo/Script/Input.txt
```

A saída será gerada em um arquivo .txt de nome Output.txt, na mesma pasta do script, com os tweets e suas respectivas predições separadas por uma tabulação. Um exemplo de entrada e saída da predição segue abaixo:

```
Dória faz certo em privatizar o carnaval de São Paulo
Com seleção e gols , Lucas Pratto tem início dos sonhos no São Paulo
```

Cujo resultado estará no arquivo **`$examples/SaoPaulo/Script/Output.txt`** *, vide exemplo:*

```
Dória faz certo em privatizar o carnaval de São Paulo    Não
Com seleção e gols , Lucas Pratto tem início dos sonhos no São Paulo    Sim
```