Trabalho Prático 2 - Processamento de Linguagem Natural

Hugo Araujo de Sousa

Processamento de Linguagem Natural (2017/2)
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)

hugosousa@dcc.ufmg.br

Resumo. *O objetivo desse trabalho é estudar a tarefa de Part-of-Speech (POS) tagging para a Língua Portuguesa. Para isso, utilizamos o corpus Mac-Morpho e comparamos o desempenho de dois modelos preditivos, um paramétrico e outro não-paramétrico.*

1. INTRODUÇÃO

Em Processamento de Linguagem Natural, uma vez que o principal objetivo é fazer com que os computadores entendam as linguagens naturais usadas pelos seres humanos, muitas vezes torna-se necessário reduzir a complexidade dessa tarefa ao quebrá-la em tarefas intermediárias. Uma dessas tarefas, que é particularmente útil na área de **parsing**, é a tarefa de **Part-of-speech tagging**, que se trata da identificação das classes gramaticais de cada uma das palavras presentes um corpus [Manning and Schütze 1999]. Esse problema não trata-se apenas da criação de um banco de dados que contenha a classe de cada palavra, uma vez que uma mesma palavra pode estar associada a múltiplas classes de acordo com seu contexto e posição em uma sentença.

Uma das abordagens que podem usadas para resolver esse problema é utilizar algum algoritmo de aprendizado de máquina supervisionado. Nesse trabalho, serão utilizados dois algoritmos de aprendizado para resolver o problema de POS tagging a fim de verificar e comparar a precisão e desempenho de cada um.

2. MODELAGEM

Dentro dos métodos de aprendizado de máquina supervisionada, existem os paramétricos e os não-paramétricos. A diferença entre os dois é que, nos métodos paramétricos, assume-se que os dados se organizam em algum modelo e então encontra-se valores apropriados do modelo a partir dos exemplos. Para abordar o problema de POS tagging, vamos utilizar um algoritmo de cada categoria, sendo **Naive Bayes** [McCallum and Nigam 1998] o paramétrico e o classificador de **Support Vector Machines (SVM)** [Hearst et al. 1998] o não-paramétrico.

Uma vez determinados os algoritmos a serem utilizados no processo de aprendizado, é necessário discutir o conjunto de dados de entrada e o mapeamento desses dados para a entrada dos algoritmos.

2.1. Corpus de Entrada

O conjunto de dados utilizado no trabalho como entrada dos algoritmos de aprendizado é o corpus Mac-Morpho [Aluísio et al. 2003]. O Mac-Morpho é um corpus de textos

escritos em Português Brasileiro, anotados com as classes gramaticais de cada palavra presente. Há, disponíveis para download gratuito, as seções de treinamento, validação e teste do corpus, que representam 76%, 4% e 20% do total do corpus, respectivamente.

Na página do corpus online [1] é possível fazer o download do mesmo, juntamente com o manual das anotações, que descreve todas as classes gramaticais utilizadas no corpus.

2.2. Extração de Features

Com o corpus de entrada em mãos, o próximo passo foi determinar como mapear esses dados para serem alimentados aos algoritmos de aprendizado de máquina. Para isso, a decisão foi trabalhar com features das palavras. Essas features são, como o nome indica, características obtidas através de cada palavra em si, além do seu contexto na sentença em que se encontra, isto é, posição absoluta e relativa às classes gramaticais. A lista das features utilizadas no trabalho é mostrada na Tabela 1.

Feature	Descrição
word	A própria palavra em si.
is_first	Booleano que indica se a palavra é a primeira da sentença.
is_last	Booleano que indica se a palavra é a primeira da sentença.
is_capitalized	Booleano que indica se a palavra começa com uma letra maiúscula.
is_all_caps	Booleano que indica se a palavra somente contém letras maiúsculas.
is_all_lower	Booleano que indica se a palavra contém somente letras minúsculas.
prefix-1	String com o primeiro caractere da palavra.
prefix-2	String com os dois primeiros caracteres da palavra.
prefix-3	String com os três primeiros caracteres da palavra.
suffix-1	String com o último caractere da palavra.
suffix-2	String com os dois últimos caracteres da palavra.
suffix-3	String com os três últimos caracteres da palavra.
prev_tag	String que representa a classe gramatical da palavra anterior à palavra atual.
next_tag	String que representa a classe gramatical da palavra seguinte à palavra atual.
has_hyphen	Booleano que indica se a palavra possui hífen.
is_numeric	Booleano que indica se a palavra é um número (dígitos).

Tabela 1. Features de palavras utilizadas no trabalho.

3. IMPLEMENTAÇÃO

O trabalho foi implementado utilizando a linguagem Python 3. Além disso, nenhum dos algoritmos de aprendizado de máquina foram implementados, sendo utilizada uma biblioteca que já continha tanto uma implementação do Naive Bayes quanto do SVM. Essa biblioteca é a Scikit-Learn [2].

[1] `http://nilc.icmc.usp.br/macmorpho`
[2] `http://scikit-learn.org/stable/`

3.1. Decisões de Implementação

Algumas decisões de implementação foram se mostrando necessárias ao longo do desenvolvimento do projeto. Estas são apresentadas a seguir:

- **Feature Hasher:** Para cada palavra do corpus, os features são coletados e armazenados em um dicionário. Para usar esses dados como entrada dos algoritmos de aprendizado, é necessário efetuar uma transformação para uma matriz de tipo Numpy Array [3]. O utilitário FeatureHasher da biblioteca Scikit-Learn se encarrega dessa tarefa, sendo muito útil para conjuntos de dados tão grandes como o corpus Mac-Morpho. Com ele, a lista que contém os dicionários que representam os features de cada palavra são convertidos para Numpy Array de forma eficiente tanto em tempo quanto espaço. No momento da instanciação de um objeto FeatureHasher, especifica-se o número de features no dicionário de features. No caso desse trabalho, esse número é igual a 17.
- **Naive Bayes Gaussiano:** A biblioteca Scikit-Learn fornece várias implementações do algoritmo de Naive Bayes. Dentre essas, duas foram testadas no trabalho: a implementação Multinomial e a Gaussiana. Comparando a precisão obtida com ambas, foi verificado que a segunda atinge maior precisão para o corpus e, portanto, foi usada no código final do projeto.

3.2. Execução do Projeto

Os arquivos do projeto encontram-se distribuídos em três pastas:

- **doc:** Contém os arquivos de documentação e manual de tags do corpus Mac-Morpho.
- **corpus:** Base de dados do corpus Mac-Morpho, dividido em treino, validação e teste.
- **src:** Código fonte do programa principal que lê o corpus, calcula as features de cada palavra e alimenta esses dados aos dois algoritmos de aprendizado, imprimindo os resultados de precisão para cada um deles.

  Para executar o programa, na pasta **src**, basta executar o comando:

  ./tp2.py [-h] [-s RSEED] train_file test_file validation_file

  Onde os argumentos do programa são:

- **Argumentos posicionais:**
  - train_file: Nome do arquivo de treino.
  - test_file: Nome do arquivo de teste.
  - validation_file: Nome do arquivo de validação.
- **Argumentos opcionais:**
  - -h ou –help: Exibe uma mensagem de ajuda e termina execução.
  - -s RSEED: Seta o número RSEED como semente de geração de números aleatórios para o algoritmo SVM.

4. RESULTADOS

Executando o programa com o corpus Mac-Morpho, uma das saídas obtidas (resultado do SVM varia com o número da seed de números aleatórios) é mostrada a seguir.

[3] www.numpy.org/

```
[+] Reading training file
[+] Reading validation file
[+] Reading test file

        NAIVE BAYES

[+] Validation precision: 53.87%
         PREP+ADV precision: 100.0%
         CUR precision: 99.12%
         PRO-KS precision: 82.51%
         ART precision: 81.75%
         KC precision: 78.32%
         PU precision: 77.27%
         PREP+ART precision: 67.15%
         PREP precision: 63.36%
         ADV-KS precision: 61.76%
         N precision: 55.62%
         NPROP precision: 51.43%
         PROPESS precision: 50.23%
         PDEN precision: 38.08%
         PREP+PROPESS precision: 36.84%
         KS precision: 36.06%
         PREP+PROADJ precision: 31.82%
         NUM precision: 30.89%
         ADV precision: 27.63%
         V precision: 25.16%
         PCP precision: 19.01%
         PREP+PROSUB precision: 16.67%
         ADJ precision: 8.03%
         PROADJ precision: 4.51%
         PROSUB precision: 1.53%
         PREP+PRO-KS precision: 0.0%
         IN precision: 0.0%

[+] Test precision: 52.3%
         PREP+ADV precision: 100.0%
         CUR precision: 96.96%
         PRO-KS precision: 83.69%
         ART precision: 79.79%
         PU precision: 75.51%
         KC precision: 74.31%
         PREP+ART precision: 66.01%
         PREP precision: 62.47%
         ADV-KS precision: 62.17%
         N precision: 57.05%
         NPROP precision: 49.67%
         PREP+PROPESS precision: 46.03%
         PROPESS precision: 40.3%
         KS precision: 37.55%
         PDEN precision: 34.98%
         NUM precision: 31.17%
         PREP+PROADJ precision: 29.13%
         ADV precision: 28.15%
         V precision: 23.54%
         PREP+PROSUB precision: 22.44%
```

```
          PCP precision: 20.52%
          ADJ precision: 8.79%
          PROADJ precision: 2.98%
          PROSUB precision: 1.98%
          PREP+PRO-KS precision: 0.0%
          IN precision: 0.0%

--------------------------------------------------------------------------------

         SVM

[+] Validation precision: 86.27%
          PU precision: 99.83%
          CUR precision: 99.12%
          PREP+ART precision: 98.56%
          ART precision: 97.93%
          KC precision: 96.12%
          PREP precision: 95.53%
          PROPESS precision: 93.69%
          PRO-KS precision: 91.93%
          NPROP precision: 87.94%
          N precision: 87.24%
          PROADJ precision: 81.14%
          PDEN precision: 79.5%
          NUM precision: 76.29%
          KS precision: 74.56%
          V precision: 71.19%
          PREP+PROADJ precision: 69.7%
          ADV precision: 69.45%
          ADV-KS precision: 64.71%
          PCP precision: 52.18%
          ADJ precision: 49.13%
          PROSUB precision: 47.51%
          PREP+PROSUB precision: 30.0%
          PREP+PROPESS precision: 10.53%
          PREP+PRO-KS precision: 0.0%
          IN precision: 0.0%
          PREP+ADV precision: 0.0%

[+] Test precision: 84.55%
          PU precision: 99.75%
          CUR precision: 99.32%
          PREP+ART precision: 98.14%
          ART precision: 97.8%
          KC precision: 96.01%
          PREP precision: 94.6%
          PRO-KS precision: 94.21%
          PROPESS precision: 90.54%
          N precision: 87.06%
          NPROP precision: 84.82%
          PREP+PROADJ precision: 76.7%
          PROADJ precision: 75.9%
          NUM precision: 75.8%
          KS precision: 75.41%
          PDEN precision: 73.53%
```

```
111             ADV-KS precision: 72.61%
112             ADV precision: 67.7%
113             V precision: 66.94%
114             PCP precision: 51.46%
115             PROSUB precision: 50.35%
116             PREP+ADV precision: 45.16%
117             ADJ precision: 44.61%
118             PREP+PROPESS precision: 34.92%
119             PREP+PROSUB precision: 30.77%
120             PREP+PRO-KS precision: 18.97%
121             IN precision: 0.0%
```

Um primeiro resultado interessante a observar é o aspecto de tempo de execução. Enquanto o Naive Bayes demora cerca de 30 segundos para executar com todos os arquivos do corpus Mac-Morpho, o SVM precisa de por volta de 7 horas para computar seus resultados.

Em relação à precisão de acerto dos algoritmos, vemos que as classes mais difíceis de serem previstas utilizando os mesmo foram as classes Interjeição e aquelas formadas pela combinação entre Preposições e outras classes gramaticais. Podemos a baixa acurácia da classe de interjeição a partir do fato de que as palavras dessa classe, além de geralmente serem muito curtas (sem informação de sufixos ou prefixos), também geralmente vêm sozinhas em uma frase - *"Ufa!"* -, não possuindo assim qualquer informação de contexto. Com esses fatores, essas palavras acabam produzindo features pouco expressivos no programa. Já para as preposições combinadas com outras classes, uma hipótese para tal precisão baixa é a de que padrões mais complicados (que envolvem uma sequência de tags) não são de fácil aprendizado para os algoritmos.

Uma outra observação diz respeito à precisão muito próxima de 100% para a classe de Símbolo de Moeda Corrente. Isso muito provavelmente se deve ao fato de que as palavras dessa classe são identificadas por serem numéricas e, como há um feature para identificar tal situação, os algoritmos não têm dificuldade em aprender esse padrão.

5. CONCLUSÃO

O trabalho em questão possibilitou um grande aprendizado, além de fixar aquele já adquirido durante as aulas da disciplina de Processamento de Linguagem Natural.

Dentro do aspecto do conhecimento adquirido com a disciplina, é importante destacar os métodos de aprendizado supervisionado, que aqui ilustram como é feito o processo de treinamento para uma tarefa da área de Processamento de Linguagem Natural. Foi muito interessante utilizar dois modelos, um paramétrico e outro não-paramétrico, obter seus resultados e poder compará-los a fim de verificar os padrões identificados por cada um e como o fato de serem paramétricos ou não afetam nesses padrões e resultados, buscando encontrar explicações para os mesmos.

Finalmente, ressalto a utilidade do trabalho para poder adquirir conhecimento sobre a biblioteca Scikit-Learn e sobre o corpus Mac-Morpho, creio que esses dois conhecimentos são de grande importância para qualquer estudante da área.

6. REFERÊNCIAS

[Aluísio et al. 2003] Aluísio, S., Pelizzoni, J., Marchi, A. R., de Oliveira, L., Manenti, R., and Marquiafável, V. (2003). *An Account of the Challenge of Tagging a Reference Corpus for Brazilian Portuguese*, pages 110–117. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Hearst et al. 1998] Hearst, M., Dumais, S., Osman, E., Platt, J., and Scholkopf, B. (1998). Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18 –28.

[Manning and Schütze 1999] Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

[McCallum and Nigam 1998] McCallum, A. and Nigam, K. (1998). A comparison of event models for naive bayes text classification. In *IN AAAI-98 WORKSHOP ON LEARNING FOR TEXT CATEGORIZATION*, pages 41–48. AAAI Press.