Procesamiento de Lenguaje Natural

Eric S. Téllez Mario Graff

Tabla de contenidos

Prefacio

El curso trata de ser auto-contenido, es decir, no debería de ser necesario leer otras fuentes para poder entenderlo y realizar las actividades. De cualquier manera es importante comentar que el curso está basado en los siguientes libros de texto:

- Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Third Edition draft. Daniel Jurafsky and James H. Martin. pdf
- Introduction to machine learning, Third Edition. Ethem Alpaydin. MIT Press.
- An Introduction to Statistical Learning with Applications in R. Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. Springer Texts in Statistics.
- All of Statistics. A Concise Course in Statistical Inference. Larry Wasserman. MIT Press.
- An Introduction to the Bootstrap. Bradley Efron and Robert J. Tibshirani. Monographs on Statistics and Applied Probability 57. Springer-Science+Business Media.
- Understanding Machine Learning: From Theory to Algorithms. Shai Shalev-Shwartz and Shai Ben-David. Cambridge University Press.

Notación

La Tabla 1 muestra la notación que se seguirá en este documento.

Tabla 1: Notación

Símbolo	Significado
x	Variable usada comunmente como entrada
y	Variable usada comunmente como salida
$\mathbb{R}$	Números reales
$\mathbf{x}$	Vector Columna $\mathbf{x} \in \mathbb{R}^d$
d	Dimensión
$\mathbf{w} \cdot \mathbf{x}$	Producto punto donde $\mathbf{w}$ y $\mathbf{x} \in \mathbb{R}^d$
$\mathcal{D}$	Conjunto de datos
$\mathcal{T}$	Conjunto de entrenamiento
$\mathcal{V}$	Conjunto de validación
$\mathcal{G}$	Conjunto de prueba
N	Número de ejemplos
K	Número de clases
$\mathbb{P}(\cdot)$	Probabilidad
$\mathcal{X}, \mathcal{Y}$	Variables aleatorías
$\mathcal{N}(\mu, \sigma^2)$	Distribución Normal con parámetros μ y σ^2
$f_{\mathcal{X}}$	Función de densidad de probabilidad de $\mathcal{X}$
$\mathbb{1}(e)$	Función para indicar; 1 only if e is true
Ω	Espacio de búsqueda
$\mathbb{V}$	Varianza
$\mathbb{E}$	Esperanza

1 Introducción

El **objetivo** de la unidad es

2 Manejando Texto

El **objetivo** de la unidad es

Paquetes usados

```
from microtc.params import OPTION_GROUP, OPTION_DELETE, OPTION_NONE
from microtc.textmodel import SKIP_SYMBOLS
from b4msa.textmodel import TextModel
from b4msa.lang_dependency import LangDependency
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud as WC
from matplotlib import pylab as plt
import numpy as np
import unicodedata
import re
```

2.1 Normalización de Texto

In all the topics covered, the assumption is that the text is well-formatted and spaces nicely surround the words (tokens). However, this is not the general case, and the spelling errors and the procedure used to define the tokens strongly impact the algorithm's performance. Consequently, this part of the course is devoted to presenting standard techniques used to normalize the text and to transform the text into tokens.

The text normalization described are mainly the ones used in the following research words:

1. An automated text categorization framework based on hyperparameter optimization (Tellez et al. (2018))
2. A simple approach to multilingual polarity classification in Twitter (Tellez, Miranda-Jiménez, Graff, Moctezuma, Suárez, et al. (2017))
3. A case study of Spanish text transformations for twitter sentiment analysis (Tellez, Miranda-Jiménez, Graff, Moctezuma, Siordia, et al. (2017))

2.2 Entity

The journey of text normalization starts with handling different entities within a text; the entities could be the mentioned of a user in a tweet, the numbers, or the URL, to mention a few. The actions performed to the entities found are to delete them or replace them for a particular token.

2.2.1 Users

The first process is to deal with username following the format of Twitter. In a tweet, the mention of a user is identified with a string starting with the character @. The two actions could be to delete all the users' mentions or change them for a common label.

The procedure uses regular expressions to find the entities; for example, the following code can remove the users' mentions.

```
text = 'Hi @xx, @mm is talking about you.'
re.sub(r"@\S+", "", text)
```

```
'Hi   is talking about you.'
```

On the other hand, to replace the username with a shared label can be implemented with the following code, where the label is `_usr`

```
text = 'Hi @xx, @mm is talking about you.'
re.sub(r"@\S+", "_usr", text)
```

```
'Hi _usr _usr is talking about you.'
```

2.2.2 URL

The previous code can be adapted to handle URL; one only needs to define the regular expression to use; see the following code that removes all the appearances of the URL.

```
text = "go http://google.com, and find out"
re.sub(r"https?://\S+", "", text)
```

```
'go  and find out'
```

2.2.3 Numbers

The previous code can be modified to deal with numbers and replace the number found with a shared label such as `_num`.

```
text = "we have won 10 M"
re.sub(r"\d\d*\.?\d*|\d*\.\d\d*", "_num", text)
```

```
'we have won _num M'
```

2.3 Spelling

The next block of text normalization modifies the writing of the text, removing components that, for particular applications, can be ignored to reduce the vocabulary size, which impacts the complexity of the algorithm and could be reflected in an improvement in the performance.

2.3.1 Case sensitive

The first of these transformations is the conversion to lower case; transforming all the words to the lower case has the consequence that the vocabulary is reduced, e.g., the word Mexico and mexico would be considered the same token. This operation can be implemented with function `lower` as follows.

```
text = "Mexico"
text.lower()
```

```
'mexico'
```

2.3.2 Punctuation

The punctuation symbols are essential to natural language understanding and generation; however, for other applications, such as sentiment analysis or text categorization, its contribution is opaque by the increase in the vocabulary size. Consequently, its removal influences the vocabulary size, which sometimes has a positive result on the performance.

These symbols can be removed by traversing the string and skipping the punctuations.

```
text = "Hi! good morning,"
output = ""
```

```
for x in text:
    if x in SKIP_SYMBOLS:
        continue
    output += x
output
```

```
'Hi good morning'
```

2.3.3 Diacritic

Different languages use diacritic symbols, e.g., México; as expected, this has the consequence of increasing the vocabulary. On the other hand, in informal writing, the misuse of diacritic symbols is common; one particular way to handle this problem is to remove the diacritic symbols and treat them as the same word, e.g., México would be replaced by Mexico.

```
text = 'México'
output = ""
for x in unicodedata.normalize('NFD', text):
    o = ord(x)
    if 0x300 <= o and o <= 0x036F:
        continue
    output += x
output
```

```
'Mexico'
```

2.4 Semantic Normalizations

The next set of normalization techniques aims to reduce the vocabulary size using the meaning of the words to modify them or remove them from the text.

2.4.1 Stop words

The stop words are the most frequent words used in the language. These words are essential to communicate but are not so much on tasks where the aim is to discriminate texts according to their meaning.

The stop words can be stored in a dictionary, and then the process of removing them consists of traversing all the tokens from a text and then removing those in the dictionary. The process is exemplified with the following code.

```
lang = LangDependency('english')

text = 'Good morning! Today, we have a warm weather.'
output = []
for word in text.split():
    if word.lower() in lang.stopwords[len(word)]:
        continue
    output.append(word)
output = " ".join(output)
output
```

```
'Good morning! Today, warm weather.'
```

2.4.2 Stemmming and Lemmatization

The idea of stemming and lemmatization, seen as a normalization process, is to group different words based on their root; for example, the process would associate words like *playing*, *player*, *plays* with the token *play*.

Stemming treats the problem with fewer constraints than lemmatization, having as a consequence that the common word found cannot be the common root of the words; additionally, the algorithms do not consider the role of the word being processed in the sentence. On the other hand, a lemmatization algorithm obtains the root of the word considering the part of the speech of the processed word.

```
stemmer = PorterStemmer()

text = 'I like playing football'
output = []
for word in text.split():
    w = stemmer.stem(word)
    output.append(w)
output = " ".join(output)
output
```

```
'i like play footbal'
```

2.5 Tokenization

Once the text has been normalized, it is time to transform it into its fundamental elements, which could be words, bigrams, n-grams, substrings, or a combination of them; this process is known as tokenization. Different methods can be applied to tokenize a text, the one used is so far is to transform a text into a list of words where the word is surrounded by space or non-printable characters. The decision of which tokenizer to use depends on the application; for example, in order to generate text, it is crucial to learn the punctuation symbols, so these symbols are tokens. On the other hand, in the text categorization problem, where the task is to classify a text, it might be irrelevant to keep the order of the words.

2.5.1 n-grams

The first tokenizer review corresponds to transforming the text into words, bigrams, and in general, n-grams. The case of words is straightforward using the function `split`; once the words have been obtained, these can be combined to form an n-gram of any size, as shown below.

```
text = 'I like playing football on Saturday'
words = text.split()
n = 3
n_grams = []
for a in zip(*[words[i:] for i in range(n)]):
    n_grams.append("~".join(a))
n_grams
```

```
['I~like~playing',
 'like~playing~football',
 'playing~football~on',
 'football~on~Saturday']
```

2.5.2 q-grams

The q-gram tokenizer complements the n-grams one; it is defined as the substring of length q. The q-grams have two relevant features; the first one is that they are language agnostic consequently can be applied to any language, and the second is that they tackle the misspelling problem from an approximate matching perspective.

The code is equivalent to the one used to compute n-grams, being the difference that the iteration is on characters instead of words.

```
text = 'I like playing'
q = 4
q_grams = []
for a in zip(*[text[i:] for i in range(q)]):
    q_grams.append("".join(a))
q_grams
```

```
['I li',
 ' lik',
 'like',
 'ike ',
 'ke p',
 'e pl',
 ' pla',
 'play',
 'layi',
 'ayin',
 'ying']
```

2.6 TextModel

The class `TextModel` of the library B4MSA contains the text normalization and tokenizers described and can be used as follows.

The first step is to instantiate the class given the desired parameters. The Entity parameters have three options to delete (`OPTION_DELETE`) the entity, replace (`OPTION_GROUP`) it with a predefined token, or do not apply that operation (`OPTION_NONE`). These parameters are:

- usr_option
- url_option
- num_option

The class has three additional transformation which are:

- emo_option
- hashtag_option
- ent_option

The Spelling transformations can be triggered with the following keywords:

- lc
- del_punc

- del_diac

which corresponds to lower case, punctuation, and diacritic.

The Semantic normalizations are set up with the parameters:

- stopwords
- stemming

Finally, the tokenizer is configured with the `token_list` parameter, which has the following format; negative numbers indicate n-grams and positive numbers q-grams.

For example, the following code invokes the text normalization algorithm; the only difference is that spaces are replaced with `~`.

```
text = 'I like playing football with @mgraffg'
tm = TextModel(token_list=[-1, 3], lang='english',
               usr_option=OPTION_GROUP,
               stemming=True)
tm.text_transformations(text)
```

```
'~i~like~play~fotbal~with~_usr~'
```

On the other hand, the tokenizer is used as follows.

```
text = 'I like playing football with @mgraffg'
tm = TextModel(token_list=[-1, 5], lang='english',
               usr_option=OPTION_GROUP,
               stemming=True)
tm.tokenize(text)
```

```
['i',
 'like',
 'play',
 'fotbal',
 'with',
 '_usr',
 'q:~i~li',
 'q:i~lik',
 'q:~like',
 'q:like~',
 'q:ike~p',
 'q:ke~pl',
```

```
'q:e~pla',
'q:~play',
'q:play~',
'q:lay~f',
'q:ay~fo',
'q:y~fot',
'q:~fotb',
'q:fotba',
'q:otbal',
'q:tbal~',
'q:bal~w',
'q:al~wi',
'q:l~wit',
'q:~with',
'q:with~',
'q:ith~_',
'q:th~_u',
'q:h~_us',
'q:~_usr',
'q:_usr~']
```

It can be observed that all q-grams start with the prefix $q:$.

3 Modelado de Lenguaje

El **objetivo** de la unidad es

4 Clasificación de Texto

El **objetivo** de la unidad es

5 Representación de Texto

El **objetivo** de la unidad es

6 Mezcla de Modelos

El **objetivo** de la unidad es

7 Tareas de Clasificación de Texto

El **objetivo** de la unidad es

8 Bases de Conocimiento

El **objetivo** de la unidad es

9 Visualización

El **objetivo** de la unidad es

10 Conclusiones

El **objetivo** de la unidad es

Referencias

Tellez, Eric S., Sabino Miranda-Jiménez, Mario Graff, Daniela Moctezuma, Oscar S. Siordia, y Elio A. Villaseñor. 2017. «A case study of Spanish text transformations for twitter sentiment analysis». *Expert Systems with Applications* 81: 457-71. https://doi.org/https://doi.org/10.1016/j.eswa.2017.03.071.

Tellez, Eric S., Sabino Miranda-Jiménez, Mario Graff, Daniela Moctezuma, Ranyart R. Suárez, y Oscar S. Siordia. 2017. «A Simple Approach to Multilingual Polarity Classification in Twitter». *Pattern Recognition Letters*. https://doi.org/10.1016/j.patrec.2017.05.024.

Tellez, Eric S., Daniela Moctezuma, Sabino Miranda-Jiménez, y Mario Graff. 2018. «An automated text categorization framework based on hyperparameter optimization». *Knowledge-Based Systems* 149: 110-23. https://doi.org/10.1016/j.knosys.2018.03.003.