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.snowball import SnowballStemmer
import unicodedata
import re
```

2.1 Introducción

Se podría suponer que el texto se que se analizará está bien escrito y tiene un formato adecuado para su procesamiento. Desafortunadamente, la realidad es que en la mayoría de aplicaciones el texto que se analiza tiene errores de ortográficos, errores de formato y además no es trivial identificar la unidad mínima de procesamiento que podría ser de manera natural, en el español, las palabras. Por este motivo, esta unidad trata técnicas comunes que se utilizan para normalizar el texto, esta normalización es un proceso previo al desarrollo de los algoritmos de PLN.

La Figura 2.1 esquematiza el procedimiento que se presenta en esta unidad, la idea es que se un texto pasa primeramente a un proceso de normalización (Sección 2.2 y Sección 2.3), para después ser segmentado (ver Sección 2.4) y el resultado es lo que se utiliza para modelar el lenguaje.

Las normalizaciones y segmentaciones descritas en esta unidad se basan principalmente en las utilizadas en los siguientes artículos científicos.

1. An automated text categorization framework based on hyperparameter optimization (Tellez et al. (2018))

Figura 2.1: Diagrama de Pre-procesamiento

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 Normalización de Texto Sintáctica

La descripción de las normalizaciones empieza presentando las que se puede aplicar a nivel de caracteres, sin la necesidad de conocer el significado de las palabras. También se agrupan en este conjunto aquellas transformaciones que se realizan mediante expresiones regulares o su búsqueda en una lista de palabras previamente definidas.

2.2.1 Entidades

La descripción de diferentes técnicas de normalización empieza con el manejo de entidades en el texto. Algunas entidades que se tratarán serán los nombres de usuario, números o URLs mencionados en un texto. Por otro lado están las acciones que se realizarán a las entidades encontradas, estas acciones corresponden a su borrado o remplazo por algún otro toquen.

2.2.1.1 Usuarios

En esta sección se trabajará con los nombres de usuarios que siguen el formato usado por Twitter. En un tuit, los nombres de usuarios son aquellas palabras que inician con el caracter @ y terminan con un espacio o caracter terminal. Las acciones que se realizarán con los nombres de usuario encontrados serán su borrado o reemplazo por una etiqueta en particular.

El procedimiento para encontrar los nombres de usuarios es mediante expresiones regulares, en particular se usa la expresión `@\S+`, tal y como se muestra en el siguiente ejemplo.

```
text = 'Hola @xx, @mm te está buscando'
re.sub(r"@\S+", "", text)
```

```
'Hola   te está buscando'
```

La segunda acción es reemplazar cada nombre de usuario por una etiqueta particular, en el siguiente ejemplo se reemplaza por la etiqueta `_usr`.

```
text = 'Hola @xx, @mm te está buscando'
re.sub(r"@\S+", "_usr", text)
```

```
'Hola _usr _usr te está buscando'
```

2.2.1.2 URL

Los ejemplos anteriores se pueden adaptar para manejar URL; solamente es necesario adecuar la expresión regular que identifica una URL. En el siguiente ejemplo se muestra como se pueden borrar las URLs que aparecen en un texto.

```
text = "puedes verificar que http://google.com esté funcionando"
re.sub(r"https?://\S+", "", text)
```

```
'puedes verificar que  esté funcionando'
```

2.2.1.3 Números

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

```
text = "acabamos de ganar 10 M"
re.sub(r"\d\d*\.?\d*|\d*\.\d\d*", "_num", text)
```

```
'acabamos de ganar _num M'
```

2.2.2 Ortografía

El siguiente bloque de normalizaciones agrupa aquellas modificaciones que se realizan a algún componente de tal manera que aunque impacta en su ortografía puede ser utilizado para reducir la dimensión y se ve reflejado en la complejidad del algoritmo.

2.2.2.1 Mayúsculas y Minúsculas

La primera de estas transformaciones es convertir todas los caracteres a minúsculas. Como se puede observar esta transformación hace que el vocabulario se reduzca, por ejemplo, las palabras *México* o *MÉXICO* son representados por la palabra *méxico*. Esta operación se puede realizar con la función `lower` tal y cómo se muestra a continuación.

```
text = "México"
text.lower()
```

```
'méxico'
```

2.2.2.2 Signos de Puntuación

Los signos de puntuación son necesarios para tareas como la generación de textos, pero existen otras aplicaciones donde los signos de puntuación tienen un efecto positivo en el rendimiento del algorithm, este es el caso de tareas de categorización de texto. El efecto que tiene el quitar los signos de puntuación es que el vocabulario se reduce. Los símbolos de puntuación se pueden remover teniendo una lista de los mismos, esta lista de signos de puntuación se encuentra en la variable `SKIP_SYMBOLS` y el siguiente código muestra un procedimiento para quitarlos.

```
text = "¡Hola! buenos días:"
output = ""
for x in text:
    if x in SKIP_SYMBOLS:
        continue
    output += x
output
```

```
'Hola buenos días'
```

2.2.2.3 Símbolos Diacríticos

Continuando con la misma idea de reducir el vocabulario, es común eliminar los símbolos diacríticos en las palabras. Esta transformación también tiene el objetivo de normalizar aquellos textos informales donde los símbolos diacríticos son usado con una menor frecuencia, en particular los acentos en el caso del español. Por ejemplo, es común encontrar la palabra *México* escrita como *Mexico*.

El siguiente código muestra un procedimiento para eliminar los símbolos diacríticos.

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

```
'Mexico'
```

2.3 Normalización Semántica

Las siguientes normalizaciones comparten el objetivo con las normalizaciones presentadas hasta este momento, el cual es la reducción del vocabulario; la diferencia es que las siguientes utilizan el significado o uso de la palabra.

2.3.1 Palabras Vacías

Las palabras vacías (*stop words*) son palabras utilizadas frecuentemente en el lenguaje, las cuales son necesarias para comunicación, pero no aportan información para discriminar un texto de acuerdo a su significado.

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.

Las palabras vacías se pueden guardar en un diccionario y el proceso de identificación consiste en buscar la existencia de la palabra en el diccionario. Una vez que la palabra analizada se encuentra en el diccionario, se procede a quitarla o cambiarla por un token particular. El proceso de borrado se muestra en el siguiente código.

```
lang = LangDependency('spanish')

text = '¡Buenos días! El día de hoy tendremos un día cálido.'
output = []
for word in text.split():
    if word.lower() in lang.stopwords[len(word)]:
        continue
    output.append(word)
```

```
output = " ".join(output)
output
```

```
'¡Buenos días! día hoy día cálido.'
```

2.3.2 Lematización y Reducción a la Raíz

La idea de lematización y reducción a la raíz (*stemming*) es transformar una palabra a su raíz mediante un proceso heurístico o morfológico. Por ejemplo, las palabras *jugando* o *jugaron* se transforman a la palabra *jugar*.

El siguiente código muestra el proceso de reducción a la raíz utilizando la clase `SnowballStemmer`.

```
stemmer = SnowballStemmer('spanish')

text = 'Estoy jugando futbol con mis amigos'
output = []
for word in text.split():
    w = stemmer.stem(word)
    output.append(w)
output = " ".join(output)
output
```

```
'estoy jug futbol con mis amig'
```

2.4 Segmentación

Una vez que el texto ha sido normalizado es necesario segmentarlo (*tokenize*) a sus componentes fundamentales, e.g., palabras o gramas de caracteres (q-grams) o de palabras (n-grams). Existen diferentes métodos para segmentar un texto, probablemente una de las más sencillas es asumir que una palabra está limitada entre dos espacios o signos de puntuación. Partiendo de el encontrar la palabra se empieza a generar los gramas de palabras, e.g., bigramas, o los gramas de caracteres si se desea solo generarlos a partir de las palabras.

2.4.1 Gramas de Palabras (n-grams)

El primer método de segmentación revisado es la creación de los gramas de palabras. El primer paso es encontrar las palabras las cuales se pueden encontrar mediante la función `split`; una vez que las palabras están definidas estás se pueden unir para generar los gramas de palabras del tamaño deseado, tal y como se muestra en el siguiente código.

```
text = 'Estoy jugando futbol con mis amigos'
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
```

```
['Estoy~jugando~futbol',
 'jugando~futbol~con',
 'futbol~con~mis',
 'con~mis~amigos']
```

2.4.2 Gramas de Caracteres (q-grams)

La segmentación de gramas de caracteres complementa los gramas de palabras. Los gramas de caracteres están definidos como la subcadena de longitud q. Este tipo de segmentación tiene la característica de que es agnóstica al lenguaje, es decir, se puede aplicar en cualquier idioma; contrastando, los gramas de palabras se pueden aplicar solo a los lenguajes que tienen definido el concepto de palabra, por ejemplo en el idioma chino las palabras no se pueden identificar como se pueden identificar en el español o inglés. La segunda característica importante es que ayuda en el problema de errores ortográficos, siguiendo una perspectiva de similitud aproximada.

El código para realizar los gramas de caracteres es similar a la presentada anteriormente, siendo la diferencia que el ciclo está por los caracteres en lugar de la palabras como se había realizado. El siguiente código muestra una implementación para realizar gramas de caracteres.

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

```
['Esto',
 'stoy',
 'toy ',
 'oy j',
 'y ju',
 ' jug',
 'juga',
 'ugan',
 'gand',
 'ando']
```

2.5 TextModel

Habiendo descrito

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

Paquetes usados

```
from microtc.utils import tweet_iterator, load_model, save_model
from b4msa.textmodel import TextModel
from EvoMSA.tests.test_base import TWEETS
from EvoMSA.utils import bootstrap_confidence_interval
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import recall_score, precision_score, f1_score
from sklearn.naive_bayes import MultinomialNB
from scipy.stats import norm, multinomial, multivariate_normal
from scipy.special import logsumexp
from collections import Counter
from matplotlib import pylab as plt
from os.path import join
import numpy as np
```

4.1 Introducción

Text Categorization is an NLP task that deals with creating algorithms capable of identifying the category of a text from a set of predefined categories. For example, sentiment analysis belongs to this task, and the aim is to detect the polarity (e.g., positive, neutral, or negative) of a text. Furthermore, different NLP tasks that initially seem unrelated to this problem can be formulated as a classification one such as question answering and sentence entailment, to mention a few.

Text Categorization can be tackled from different perspectives; the one followed here is to treat it as a supervised learning problem. As in any supervised learning problem, the starting point is a set of pairs, where the first element of the pair is the input and the second one corresponds to the output. Let $\mathcal{D} = \{(\text{text}_i, y_i) \mid i = 1, \dots, N\}$ where $y \in \{c_1, \dots c_K\}$ and text_i is a text.

4.2 Modelado Probabilistico (Distribución Categórica)

4.2.1 Problema Sintético

The description of Bayes' theorem continues with an example of a Categorical distribution. A Categorical distribution can simulate the drawn of K events that can be encoded as characters, and ℓ repetitions can be represented as a sequence of characters. Consequently, the distribution can illustrate the generation sequences associated with different classes, e.g., positive or negative.

The first step is to create the dataset. As done previously, two distributions are defined, one for each class; it can be observed that each distribution has different parameters. The second step is to sample these distributions; the distributions are sampled 1000 times with the following procedure. Each time, a random variable representing the number of outcomes taken from each distribution is drawn from a Normal $\mathcal{N}(15, 3)$ and stored in the variable `length`. The random variable indicates the number of outcomes for each Categorical distribution; the results are transformed into a sequence, associated to the label corresponding to the positive and negative class, and stored in the list `D`.

```
pos = multinomial(1, [0.20, 0.20, 0.35, 0.25])
neg = multinomial(1, [0.35, 0.20, 0.25, 0.20])
length = norm(loc=15, scale=3)
D = []
m = {k: chr(122 - k) for k in range(4)}
id2w = lambda x: " ".join([m[_] for _ in x.argmax(axis=1)])
for l in length.rvs(size=1000):
    D.append((id2w(pos.rvs(round(l))), 1))
    D.append((id2w(neg.rvs(round(l))), 0))
```

The following table shows four examples of this process; the first column contains the sequence, and the second the associated label.

Text	Label
x w x x z w y	positive
y w z z z x w	negative
z x x x z x z w x w	positive
x w z w y z z z z w	negative

As done previously, the first step is to compute the likelihood given that dataset; considering that the data comes from a Categorical distribution, the procedure to estimate the parameters is similar to the ones used to estimate the prior. The following code estimates the data

parameters corresponding to the positive class. It can be observed that the parameters estimated are similar to the ones used to generate the dataset.

```
D_pos = []
[D_pos.extend(data.split()) for data, k in D if k == 1]
words, l_pos = np.unique(D_pos, return_counts=True)
w2id = {v: k for k, v in enumerate(words)}
l_pos = l_pos / l_pos.sum()
l_pos
array([0.25489421, 0.33854064, 0.20773186, 0.1988333 ])
```

An equivalent procedure is performed to calculate the likelihood of the negative class.

```
D_neg = []
[D_neg.extend(data.split()) for data, k in D if k == 0]
_, l_neg = np.unique(D_neg, return_counts=True)
l_neg = l_neg / l_neg.sum()
```

The prior is estimated with the following code, equivalent to the one used on all the examples seen so far.

```
_, priors = np.unique([k for _, k in D], return_counts=True)
N = priors.sum()
prior_pos = priors[1] / N
prior_neg = priors[0] / N
```

Once the parameters have been identified, these can be used to predict the class of a given sequence. The first step is to compute the likelihood, e.g., $\mathbb{P}(\text{w w x z}|\ \mathcal{Y})$. It can be observed that the sequence needs to be transformed into tokens which can be done with the `split` method. Then, the token is converted into an index using the mapping `w2id`; once the index is retrieved, it can be used to obtain the parameter associated with the word. The likelihood is the product of all the probabilities; however, this product is computed in log space.

```
def likelihood(params, txt):
    params = np.log(params)
    _ = [params[w2id[x]] for x in txt.split()]
    tot = sum(_)
    return np.exp(tot)
```

The likelihood combined with the prior for all the classes produces the evidence, which subsequently is used to calculate the posterior distribution. The posterior is then used to predict the class for all the sequences in $\mathcal{D}$. The predictions are stored in the variable `hy`.

```
post_pos = [likelihood(l_pos, x) * prior_pos for x, _ in D]
post_neg = [likelihood(l_neg, x) * prior_neg for x, _ in D]
evidence = np.vstack([post_pos, post_neg]).sum(axis=0)
post_pos /= evidence
post_neg /= evidence
hy = np.where(post_pos > post_neg, 1, 0)
```

4.2.2 Clasificador de Texto

The approach followed on text categorization is to treat it as supervised learning problem where the starting point is a dataset $\mathcal{D} = \{(\text{text}_i, y_i) \mid i = 1, \ldots, N\}$ where $y \in \{c_1, \ldots c_K\}$ and text_i is a text. For example, the next code uses a toy sentiment analysis dataset with four classes: negative (N), neutral (NEU), absence of polarity (NONE), and positive (P).

```
D = [(x['text'], x['klass']) for x in tweet_iterator(TWEETS)]
```

As can be observed, $\mathcal{D}$ is equivalent to the one used in the Categorical Distribution example. The difference is that sequence of letters is changed with a sentence. Nonetheless, a feasible approach is to obtain the tokens using the `split` method. Another approach is to retrieve the tokens using a Tokenizer, as covered in the Text Normalization Section.

The following code uses the `TextModel` class to tokenize the text using words as the tokenizer; the tokenized text is stored in the variable `D`.

```
tm = TextModel(token_list=[-1])
tok = tm.tokenize
D = [(tok(x), y) for x, y in D]
```

Before estimating the likelihood parameters, it is needed to encode the tokens using an index; by doing it, it is possible to store the parameters in an array and compute everything `numpy` operations. The following code encodes each token with a unique index; the mapping is in the dictionary `w2id`.

```
words = set()
[words.update(x) for x, y in D]
w2id = {v: k for k, v in enumerate(words)}
```

Previously, the classes have been represented using natural numbers. The positive class has been associated with the number 1, whereas the negative class with 0. However, in this dataset, the classes are strings. It was decided to encode them as numbers to facilitate subsequent

operations. The encoding process can be performed simultaneously with the estimation of the prior of each class. Please note that the priors are stored using the logarithm in the variable `priors`.

```
uniq_labels, priors = np.unique([k for _, k in D], return_counts=True)
priors = np.log(priors / priors.sum())
uniq_labels = {str(v): k for k, v in enumerate(uniq_labels)}
```

It is time to estimate the likelihood parameters for each of the classes. It is assumed that the data comes from a Categorical distribution and that each token is independent. The likelihood parameters can be stored in a matrix (variable `l_tokens`) with K rows, each row contains the parameters of the class, and the number of columns corresponds to the vocabulary's size. The first step is to calculate the frequency of each token per class which can be done with the following code.

```
l_tokens = np.zeros((len(uniq_labels), len(w2id)))
for x, y in D:
    w = l_tokens[uniq_labels[y]]
    cnt = Counter(x)
    for i, v in cnt.items():
        w[w2id[i]] += v
```

The next step is to normalize the frequency. However, before normalizing it, it is being used a Laplace smoothing with a value 0.1. Therefore, the constant 0.1 is added to all the matrix elements. The next step is to normalize (second line), and finally, the parameters are stored using the logarithm.

```
l_tokens += 0.1
l_tokens = l_tokens / np.atleast_2d(l_tokens.sum(axis=1)).T
l_tokens = np.log(l_tokens)
```

4.2.2.1 Prediction

Once all the parameters have been estimated, it is time to use the model to classify any text. The following function computes the posterior distribution. The first step is to tokenize the text (second line) and compute the frequency of each token in the text. The frequency stored in the dictionary `cnt` is converted into the vector `x` using the mapping function `w2id`. The final step is to compute the product of the likelihood and the prior. The product is computed in log-space; thus, this is done using the likelihood and the prior sum. The last step is to compute the evidence and normalize the result; the evidence is computed with the function `logsumexp`.

```
def posterior(txt):
    x = np.zeros(len(w2id))
    cnt = Counter(tm.tokenize(txt))
    for i, v in cnt.items():
        try:
            x[w2id[i]] += v
        except KeyError:
            continue
    _ = (x * l_tokens).sum(axis=1) + priors
    l = np.exp(_ - logsumexp(_))
    return l
```

The posterior function can predict all the text in $\mathcal{D}$; the predictions are used to compute the model's accuracy. In order to compute the accuracy, the classes in $\mathcal{D}$ need to be transformed using the nomenclature of the likelihood matrix and priors vector; this is done with the `uniq_labels` dictionary (second line).

```
hy = np.array([posterior(x).argmax() for x, _ in D])
y = np.array([uniq_labels[y] for _, y in D])
(y == hy).mean()
0.974
```

4.2.2.2 Training

Solving supervised learning problems requires two phases; one is the training phase, and the other is the prediction. The posterior function handles the later phase, and it is missing to organize the code described in a training function. The following code describes the training function; it requires the dataset's parameters and an instance of `TextModel`.

```
def training(D, tm):
    tok = tm.tokenize
    D =[(tok(x), y) for x, y in D]
    words = set()
    [words.update(x) for x, y in D]
    w2id = {v: k for k, v in enumerate(words)}
    uniq_labels, priors = np.unique([k for _, k in D], return_counts=True)
    priors = np.log(priors / priors.sum())
    uniq_labels = {str(v): k for k, v in enumerate(uniq_labels)}
    l_tokens = np.zeros((len(uniq_labels), len(w2id)))
    for x, y in D:
        w = l_tokens[uniq_labels[y]]
```

```
        cnt = Counter(x)
        for i, v in cnt.items():
            w[w2id[i]] += v
    l_tokens += 0.1
    l_tokens = l_tokens / np.atleast_2d(l_tokens.sum(axis=1)).T
    l_tokens = np.log(l_tokens)
    return w2id, uniq_labels, l_tokens, priors
```

4.3 Modelado Vectorial

xxx

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.