Applied Data Science

Session 33_34: Processing Text

Dr. Soharab Hossain Shaikh

1

Processing Raw Text

The source of data could be many. We may read text from a web-page or from a CSS feed or from a local disk file etc.

This raw text may contain text in many different format (ASCII, Unicode format etc.). With Unicode the number of alpha-numeric characters that can be represented becomes really huge as it has support for not just English but many other foreign languages.

This collected text needs to be processed in order to extract suitable information from it.

Processing this text will involve extraction sentences, extracting words; finding the part-of-speech of the words written in a sentence, recognizing named entities, disambiguating polysemous words etc. – all these tasks will help understand the meaning or semantics of the sentence.

2

Processing Raw Text

NAME	DESCRIPTION
Tokenization	Segmenting text into words, punctuations marks etc.
Part-of-speech (POS) **Tagging**	Assigning word types to tokens, like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Lemmatization	Assigning the base forms of words. For example, the lemma of "was" is "be", and the lemma of "rats" is "rat".
Sentence Boundary Detection (SBD)	Finding and segmenting individual sentences.
Named Entity Recognition (NER)	Labelling named "real-world" objects, like persons, companies or locations.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.

3

Tokenization

Tokenization is the process by which a body of text is divided into smaller parts called tokens.

NLP is used for building applications such as text classification, sentimental analysis, language translation, intelligent chatbot etc.

> It is important to identify pattern in the text. Tokens are very useful for finding such patterns.

> Tokenization is considered as a base step for stemming and lemmatization.

Tokenization can be done at the word level (e.g. extracting individual words from a given sentence.).

Or, it can be performed at the sentence level (e.g. extracting individual sentences from a large textual paragraph/document etc.)

4

Tokenization

Tokenization is the process by which a body of text is divided into smaller parts called tokens.

Sentence Tokenization: Different sentences can be tokenized from a text paragraph
Word Tokenization: words can be tokenized from a given sentence

NLP is used for building applications such as
> text classification
> sentimental analysis
> language translation
> intelligent chatbot etc.

It is important to identify pattern in the text.
Tokens are very useful for finding such patterns.

Tokenization is considered as a preprocessing step for other NLP tasks e.g. stemming, lemmatization, named-entity recognition etc.

5

Tokenization

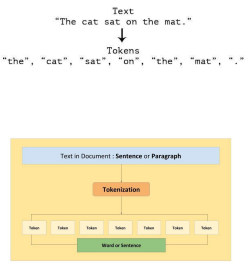
Word Tokenization from a sentence:
After tokenizing the words from the given sentence
" **This is a beautiful day.**

We may generate a list of the words as tokens as follows:
['This', 'is', 'a', 'beautiful', 'day', '.']

Sentence Tokenization from a sentence:
After tokenizing the sentences from the following paragraph
"This is a beautiful day. Sun is shining bright. Use an umbrella."

We may generate a list of the sentences as tokens as follows:

['This is a beautiful day', 'Sun is shining bright', 'Use an umbrella']



6

String Processing with Regular Expression

Regular expressions give us a more powerful and flexible method for describing the character patterns we are interested in.

Basic Regular Expression Meta-Characters, Including Wildcards, Ranges and Closures

Operator	Behavior
.	Wildcard, matches any character
^abc	Matches some pattern *abc* at the start of a string
abc$	Matches some pattern *abc* at the end of a string
[abc]	Matches one of a set of characters
[a-Za-0]	Matches one of a range of characters
ed|ing|s	Matches one of the specified strings (disjunction)
*	Zero or more of previous item, e.g. a*, [a-z]* (also known as *Kleene Closure*)
+	One or more of previous item, e.g. a+, [a-z]+
?	Zero or one of the previous item (i.e. optional), e.g. a?, [a-z]?
{n}	Exactly *n* repeats where n is a non-negative integer
{n,}	At least *n* repeats
{,n}	No more than *n* repeats
{m,n}	At least *m* and no more than *n* repeats
a(b|c)+	Parentheses that indicate the scope of the operators

7

String Processing with Regular Expression

Regular expressions can also be used for finding word stems, tokenized text etc.

Example:
We can define patterns using regular expression (R.E) as follows:

A R.E pattern: **"^.*(ing|ly|ment)$"** can be used to search for words ending with the substring "ing" or "ly" or "ment".

Explanation:
^ denotes that there should be some characters at the beginning of the word.
.* denote that there could be many characters (including none)
(ing|ly|ment) denotes that we are taking about any of the strings containing the substring "ing", "ly" or"ment".
$ denotes we are looking for such a substring at the end of the word (i.e. words ending with these patterns).

8

Stop Words - Removal

Stop Words in a Language
> have little meaning - no discriminating power
> high frequency -> large proportion of the index

```
> stopwords("english")
  [1] "i"          "me"         "my"         "myself"     "we"
  [6] "our"        "ours"       "ourselves"  "you"        "your"
 [11] "yours"      "yourself"   "yourselves" "he"         "him"
 [16] "his"        "himself"    "she"        "her"        "hers"
 [21] "herself"    "it"         "its"        "itself"     "they"
 [26] "them"       "their"      "theirs"     "themselves" "what"
 [31] "which"      "who"        "whom"       "this"       "that"
 [36] "these"      "those"      "am"         "is"         "are"
 [41] "was"        "were"       "be"         "been"       "being"
 [46] "have"       "has"        "had"        "having"     "do"
```

9

Normalizing Text

Text normalization can be done by converting a text from upper/mixed case to lower case.
After this process, the distinction between the word "The" and "the" is ignored.

Often, we want to go further than this, and strip off any affixes, a task known as **stemming.**

A further step is to make sure that the resulting form is a known word in a dictionary, a task known as **lemmatization**.

10

Stemming and Lemmatization – Text Normalization

Stemming : Reduction of word-forms to stems. Typically done by suffix stripping plus some extra steps and checks
studies -> studi
troubled -> troubl
changing -> chang

Lemmatization - linguistically proper way of normalization
Transformation of a word-form into a linguistically valid base form, called the lemma
nouns -> singular nominative form; verbs -> infinitive form; adjectives -> singular, nominative, masculine, indefinitive, positive form

studies -> study
troubled -> trouble
cries -> cry
A much more difficult task than stemming, especially for morphologically complex languages, for which you basically need:
> a morphological dictionary that maps word-forms to lemmas
> a machine learning model, trained on a large number of word-lemma pairs

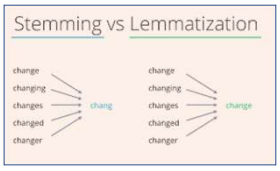
11

Normalizing Text with Stemming

Result of applying the Snowball stemmer on the following list of words:
['run','runner','running','ran','runs','easily','fairly']

Produces the following results (conversion of the words are shown below):

run → run
runner → runner
running → run
ran → ran
runs --> run
easily → easili
fairly → fair



12

Normalizing Text with Lemmatization

Result of applying lemmatization on the following list of words:
[studies, studying, cries, cry]

Produces the following results (conversion of the words are shown below):

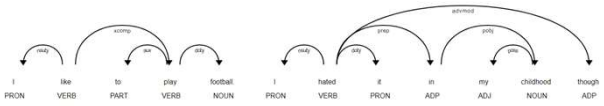
studies → study
studying → studying
cries → cry
cry -> cry

Stemming	Lemmatization
adjustable → adjust	was → (to) be
formality → formaliti	better → good
formaliti → formal	meeting → meeting
airliner → airlin ⚠	

13

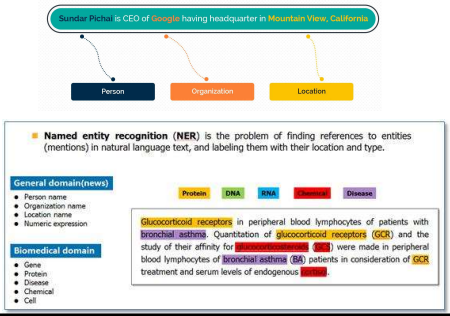
Part-of-Speech (POS) Tagging

I like to play football. I hated it in my childhood though.



14

Named Entity Recognition (NER)



15

Text - Feature Extraction

Raw text is not suitable for further analysis by any machine learning models.

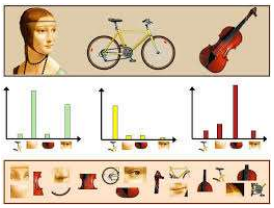
We need to represent a body of text with some numeric vectors so that these vectors can be further fed to a machine learning models for processing.

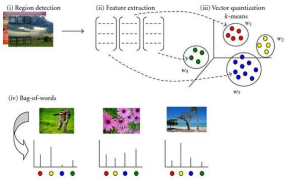
We are going to discuss two techniques for converting a sentence/document into a numeric vector –

> Bag of Words
> TF-IDF

16

Bag-of-Words (BoW) in Computer Vision

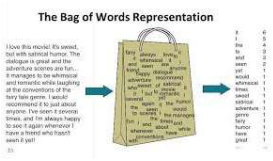




17

Bag-of-Words (BoW)

- **Bag of Words (BoW)** refers to the representation of text which describes the presence of words within the text data.
- The intuition behind this is that two similar text fields will contain similar kind of words and will therefore have a similar bag of words.
- Further, that from the text alone we can learn something about the meaning of the document.



18

Bag-of-Words (BoW)

Let us consider two strings/documents:

1: "i am happy to see you"
2: "the happy prince looked around"

The list of unique words/vocabulary are the following:
am, around, happy, i, looked, prince, see, you, the, to

A binary feature vector can be formed for each of the documents as follows:

	am	**around**	**happy**	**i**	**looked**	**prince**	**see**	**you**	**the**	**to**
1:	1	0	1	1	0	0	1	1	0	1
2:	0	1	1	0	1	1	0	0	1	0

Here 1 represents that the word is **present** in the string/document and 0 means the word is **absent**.

19

Bag-of-Words (BoW)

Similarly if a word is present more than once in a document, the 1 can be replaced with the **actual count** of the word in the document.

Example:

If we have a third string/document
3: "the happy prince looked around to see you happy"

	am	**around**	**happy**	**i**	**looked**	**prince**	**see**	**you**	**the**	**to**
3:	0	1	2	1	1	1	1	1	1	1

20

Bag-of-Words (BoW)

It is evident that the dimension of this feature vector will be equal to the number of unique words in the text (strings or documents).

For a very large document there are chances that many words present in one document will not be present in the other.
Therefore, the feature vectors for the documents will hold the values 0' sin most of the places.

To overcome this problem, normally the BoW vectors are represented as sparse vectors, reducing the dimensions of the vector representations of the documents.

We generally perform stop-word-removal before finding BoW.

21

Bag-of-Words (BoW)

Advantages:

1.Very simple to understand and implement.

Disadvantages:

1. Bag of words leads to a high dimensional feature vector due to large size of Vocabulary, V. It leads to a highly sparse vectors as there is nonzero value in dimensions corresponding to words that occur in the sentence.

2. Bag of words doesn't leverage co-occurrence statistics between words. In other words, it assumes all words are independent of each other.

When we want to capture more **context** (what word appeared after some other word) and not just co-occurrence in the same document.

Sometimes **bag of bigrams** are used to capture some context, though they are very expensive. More generic form of bigrams is **n-grams** based model.

22

TF-IDF

Term Frequency (TF)

In Term Frequency (TF), we count the number of words occurred in each document.

The main issue with this Term Frequency is that it will give more weight to longer documents.

Term frequency is basically the output of the BoW model.

Inverse Document Frequency (IDF)

IDF (Inverse Document Frequency) measures the amount of information a given word provides across a set of the documents. Defined as follows:

IDF(W) = log(#(documents)/ #(documents containing word W))

IDF is the logarithmically scaled inverse ratio of the number of documents that contain the word and the total number of documents.

23

TF-IDF

Term Frequency (tf): gives us the frequency of the word in each document in the corpus. It is the ratio of number of times the word appears in a document compared to the total number of words in that document. It increases as the number of occurrences of that word within the document increases. Each document has its own tf.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse Document Frequency (idf): used to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score. It is given by the equation below.

$$idf(w) = log(\frac{N}{df_t})$$

Combining these two we come up with the **TF-IDF** score (w) for a word in a document in the corpus. It is the product of tf and idf:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

24

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) normalizes the document term matrix.

It is the **product of TF and IDF.**

A word with a high tf-idf score in a given document suggests that the word occurs most of the times in that document and must be absent in the other documents. Therefore, in the context of that given document, the word must be a signature word.

On the other hand, tfidf score of a word that occurs in all/most of the documents will be low implying that the word is not so important.

25

TF-IDF Example

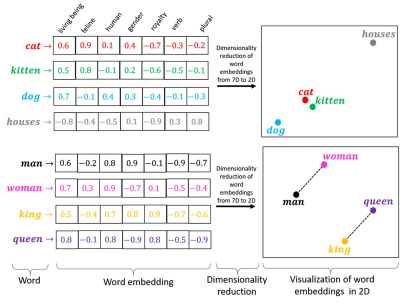
Let's take an example to get a clearer understanding.
Sentence 1 : The car is driven on the road.
Sentence 2: The truck is driven on the highway.
In this example, each sentence is a separate document.
We will now calculate the TF-IDF for the above two documents, which represent our corpus.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

From the above table, we can see that TF-IDF of common words was zero, which shows they are not significant. On the other hand, the TF-IDF of "car" , "truck", "road", and "highway" are non-zero. These words have more significance.

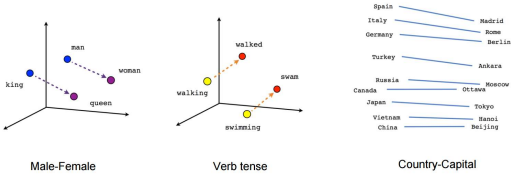
26

Word Embedding – Deep Learning



27

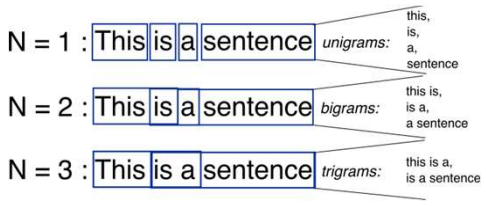
Word Embedding – Deep Learning



Interested readers may further explore Word2Vec, GloVe, Doc2Vec and the Gensim library.

28

N-Gram



29

Similarity Score – Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

The cosine of 0° is 1, and it is less than 1 for any angle in the interval (0,π] radians.

It is thus a judgment of orientation and not magnitude:

> two vectors with the same orientation have a cosine similarity of 1

> two vectors oriented at 90° relative to each other have a similarity of 0

> two vectors diametrically opposed have a similarity of -1

30

Similarity Score – Cosine Similarity

The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

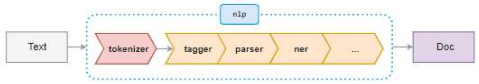
$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos\theta$$

Given two vectors of attributes, A and B, the cosine similarity, cos(θ), is represented using a dot product and magnitude as

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

31

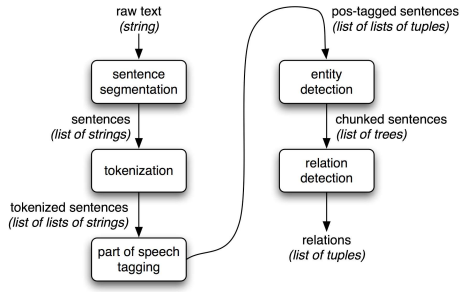
Typical Text Processing Pipeline



The traditional text processing pipeline consists of a tagger, a parser and an entity recognizer.
Each pipeline component returns a processed document, which is then passed on to the next component.

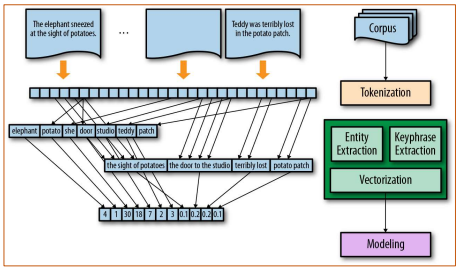
32

Extraction of Information from Text



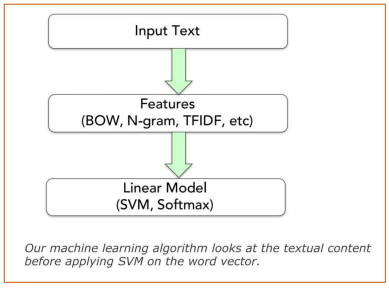
33

Key Steps in an NLP Application



34

Text Processing with ML



35

Approaching a Text Classification Problem

36

Text Classification

Classification is the task of choosing the correct **class label** for a given input.

In basic classification tasks, each input is considered in isolation from all other inputs, and the set of labels is defined in advance.

Text classification refers to the problem of deciding the output class labels for a collection of textual documents/articles/files etc.

Some examples of text classification tasks are:

> Deciding whether an email is spam or not.

> Deciding what the topic of a news article is, from a fixed list of topic areas such as "sports," "technology," and "politics."

> Deciding whether a given occurrence of the word ***bank*** is used to refer to a river bank, a financial institution, the act of tilting to the side, or the act of depositing something in a financial institution.

37

Supervised Classification

In supervised classification the data/features as well as the desired output labels are known.

The input data is a pair (X_i , y_i). Where X_i represents the *i-th* input data/observation and y_i denotes the corresponding desired output label/category/class.

In general X_i is a multidimensional feature vector

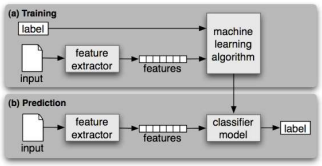
$X_i = [x_{1i} , x_{2i} .... x_{ni}]$

A machine learning model is trained by feeding the input data as well as the desired output labels.

After many iterations of training, the model becomes able to figure out what output label will be given to a new data.

38

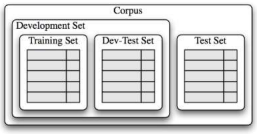
Supervised Classification



- (a) **During training**, a feature extractor is used to convert each input value to a feature set. These feature sets, which capture the basic information about each input. Pairs of feature sets and labels are fed into the machine learning algorithm to generate a model.
- (b) **During prediction**, the same feature extractor is used to convert unseen inputs to feature sets. These feature sets are then fed into the model, which generates predicted labels.

39

Data Splitting



Organization of corpus data for training supervised classifiers.

The corpus data is divided into two sets: the development set and the test set.

The development set is often further subdivided into a ***training set*** and a dev-test set (***validation set***).

40

Libraries

- **NLTK (Natural Language Toolkit)**
- *https://www.nltk.org/*
- **Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.**
- *http://www.nltk.org/book/*

- **spaCy**
- *https://spacy.io/*

41

NLTK – Tokenize and POS Tag Words

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

42

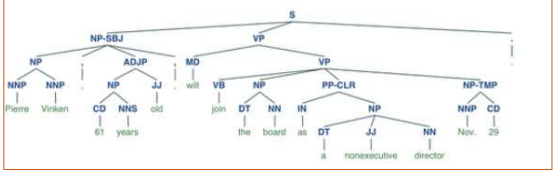
NLTK – NER on POS Tagged Words

```
>>> entities = nltk.chunk.ne_chunk(tagged)
>>> entities
Tree('S', [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'),
          ('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN'),
      Tree('PERSON', [('Arthur', 'NNP')]),
          ('did', 'VBD'), ("n't", 'RB'), ('feel', 'VB'),
          ('very', 'RB'), ('good', 'JJ'), ('.', '.')])
```

43

NLTK – Parse Tree

```
>>> from nltk.corpus import treebank
>>> t = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> t.draw()
```



44

45

Working demos on the following topics will be explained in the next session

> How to tokenize a text?
> How to perform pattern matching with Regular Expression?
> How to remove stop words from a text document?
> How to perform POS tagging and NER?
> How to extract features from a text document?
> How to perform Text Classification with Supervised Learning?
> How can document categorization be performed with an Unsupervised approach?

46

To be continued in the next session…..

47