# 13. NATURAL LANGUAGE PROCESSING

# LEARNING OBJECTIVES

- NLP Processing
- Feature extraction
- Pipeline in scikit
- Hashing Trick

# REVIEW OF LESSON 12

# LAST LESSON REVIEW

- Random Forests
- SVM

# NLP

# HUGE DOMAIN

- Classify documents,
- Detect (depression, ...)
- Translate, Summarize, ....
- Named entity recognition
- Survey analysis
- Automatic Speech Recognition (Siri, Alexa, ...)
- Unsupervised: infer sentiment or topics, find associations and links, summarize

# FEATURE EXTRACTION

From raw text (html) to vectors used by ML libraries

- Tokenization
- Counting occurences of words / frequencies
- Numerical representation of documents

# TOKENIZE

## TOKENS

Tokenization is the process of breaking a stream of text up into syllables, letters, words, n-grams, phrases, symbols, or other meaningful elements called tokens. The list of tokens becomes input for further processing such as parsing or text mining.

## N-GRAMS

Contiguous sequence of n items from a given sequence of text or speech

- unigram
- bigram
- n-gram

# VECTORIZING TEXT

First we need to transform the raw text into vectors of numerical values

Extract numerical features from text

- tokenizing strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
- counting the occurrences of tokens in each document.
- weighting with diminishing importance tokens that occur in the majority of samples / documents.

# TEXT FEATURE EXTRACTION

Features and samples are defined as follows:

- each individual token occurrence frequency is treated as a feature.
- the vector of all the token frequencies for a given document is considered a multivariate sample.
- A corpus of documents can thus be represented by
  - a matrix with one row per document
  - one column per token (e.g. word) occurring in the corpus.

# BAG OF WORDS

Documents are described by word occurrences while completely ignoring the relative position information of the words in the document.

This approach (tokenization, counting and normalization) is called the **Bag of Words or Bag of n-grams representation**.

# NLP VOCABULARY

- **Corpus**: ensemble of documents
- **Token**: the element, the word, the atom
- **Unigrams, bi-grams, n-grams**: sequence of 1, 2 or n words taken as the basic element
- **Stopwords**: small words that are discarded as not meaningful: *a, an, the, my, get, ...*

# LEMMATIZATION AND STEMMING

Lemmatization and stemming are special cases of normalization. They identify a canonical representative for a set of related word forms. The purpose of both stemming and lemmatization is to reduce morphological variation

'to walk' may appear as 'walk', 'walked', 'walks', 'walking'.

## STEMMING

Crude direct approach that chops off the endings of the word to limit variations

'walk', 'walked', 'walks', 'walking' => walk

NLTK Stemmer

# LEMMATIZATION

Returns the closest meaningful *root* word.

In NLTK

```
wordnet_lemmatizer.lemmatize('is', pos='v')
'be'
wordnet_lemmatizer.lemmatize('are', pos='v')
'be'

wordnet_lemmatizer.lemmatize('are', pos='n')
'are'
```

NLTK Lemmatizer

# POS - TAGGING

Given a text, assigns roles to each word: noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection.

Demo

## LIBRARIES

- NLTK
- Spacy (POS)
- Gensim (Latent Dirichlet Allocation - Topic Modeling)

and

- Scikit

# DEMO 1: VECTORIZING A TEXT

## Vectoring a text

## NORMALIZING: TF-IDF

What if you have several documents and a word is very frequent in just one of them. It's relative frequency should not be important.

TF-IDF: term frequency–inverse document frequency

**TF:**

- Boolean *frequency*: $f_{t,d} = 1$ if t occurs in d and 0 otherwise;
- Logarithmically scaled frequency: $f_{t,d} = 1 + log f_{t,d}$, or zero if $f_{t,d} = 0$ is zero;
- Agmented frequency, to prevent a bias towards longer documents, e.g. raw frequency divided by the maximum raw frequency of any term in the document:

$$\mathrm{tf}(t, d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max f_{t',d} : t' \in d}$$

# TF-IDF

## IDF

**inverse document frequency** is a measure of how much information the word provides, that is, whether the term is common or rare across all documents

IDF = the total number of documents divided by the number of documents containing the term, and then taking the logarithm of that quotient.

$$\mathrm{idf}(t, D) = \log \frac{N}{1 + |d \in D : t \in d|}\Bigg|$$

- N: total number of documents in the corpus $N = |D|$
- $|d \in D : t \in d|$ : number of documents with term $t$ $\mathrm{tf}(t, d) \neq 0|$.

## TF-IDF

Then tf–idf is calculated as

$$\mathrm{tf} - \mathrm{idf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

### IN SCIKIT

TfidfVectorizer

# HASHING TRICK

- Documents don't have the same length
- You may not have all the documents available (streaming)
- Too many words => too many dimensions
- highly dimensional very sparse input matrix.

=> Dimensionality reduction with **The Hashing Trick**

# HASHING TRICK

Ex: Transform any sentences into value 0 to 99

Simple Hash function:

- *a* to 1, *b* to 2, *c* to 3 and so on, up to *z* being 26
- result modulo 100

Sentence: "Beware the hobby that eats"

```
* (beware) 2 + 5 + 23 + 1 + 18 + 5 +
* (the) 20 + 8 + 5 +
* (hobby) 8 + 15 + 2 + 2 + 25 +
* (that) 20 + 8 + 1 + 20 +
* (eats) 5 + 1 + 20 + 19
* = 233
```

result = 33

# HASHING TRICK

Good hash function

- Uniformity: translate your input into each number in its output range with same probability
- Cascading: a small change in your input data will cause a big change in the output

=> limit collisions => lose interpretability

- How do you build a language model with a million dimensions?
- HashingVectorizer

## INSTALL A FEW THINGS

- conda install BeautifulSoup4
- nltk

# 1ST LAB PREDICTING SENTIMENT

## N2 Imdb Reviews

- Get the data
- Clean up the text: remove punctuation, html markup, numbers, stop words, tokenize and return one long paragraph per document
- Process all the reviews, store into an array
- Train a RF
- Assess on the test set
- AUC curve

Use scikit CountVectorizer and Try to improve the score

# PIPELINE IN SCIKIT

The ability to chain operations

For instance:

- HashingVectorizer
- Classifier

Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must be ‘transforms’, that is, they must implement fit and transform methods. The final estimator only needs to implement fit.

Example

# 2ND LAB: CLASSIFICATION

Classification on the twenty Newsgroup dataset

# SIMILARITY BETWEEN DOCUMENTS

## COSINE SIMILARITY

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

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

ex in scikit learn

# CLUSTERING

- with HashingVectorizer
- Silhouette Coefficient

http://scikit-learn.org/stable/auto_examples/text/document_clustering.html

# LINKS

- Standford NLP Group
- Kaggle - Bag of Words
- How do you build a language model with a million dimensions?

# QUESTIONS