COMP8210 — Big Data Technologies

Week 9 Lecture 1: Practical Text Analytics

Diego Mollá

COMP8210 2020H2

Abstract

This lecture will be more practical and will introduce the most popular Python libraries for text analytics. Special emphasis will be given to sklearn because of its popularity, effectiveness, speed, and ease of use.

Update September 29, 2020

Contents

Reading

- Lecture notes.

1 Some APIs for Text Analytics

Web Demos

- Explosion AI: *https://explosion.ai/demos/*
- Analysis of tweets: *https://www.csc2.ncsu.edu/faculty/healey/tweet_viz/tweet_app/*
- Analysis of news: *https://developer.aylien.com/text-api-demo*
- Coronavirus news tracker: *https://coronavirus.aylien.com/*
- APIs and Demos: *http://text-processing.com/*
- . . .

Programming Libraries

- gensim *https://radimrehurek.com/gensim/*
- Spacy *https://spacy.io/*
- Natural Language Toolkit (NLTK) *https://www.nltk.org/*
- Scikit-Learn *http://scikit-learn.org/stable/*
- Keras *https://keras.io/*
- ...

Graphical Interfaces

Usually integrated in general machine learning tools

- RapidMiner *https://rapidminer.com/*
- Weka *https://www.cs.waikato.ac.nz/ml/weka/*
- SAS Enterprise Miner, SAS Viya *https://www.sas.com*
- ...

Cloud Services

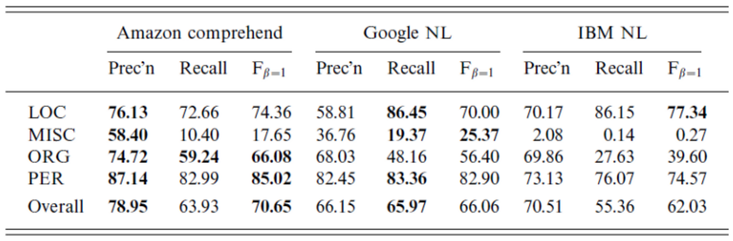
- Azure Machine Learning Studio *https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/text-analytics*
- Aylien *https://aylien.com/text-api/*
- ...

Comparison of Named Entity Recognition Systems

Text APIs compared in these two posts:

- *https://medium.com/@boab.dale/text-analytics-apis-part-1-the-bigger-players-3ce8a93577bd*
- *https://becominghuman.ai/text-analytics-apis-part-2-the-smaller-players-c9e608cf7810*

Table 2. *Results on the CoNLL shared task data; all values are percentages*



	Amazon comprehend			Google NL			IBM NL		
	Prec'n	Recall	$F_{\beta=1}$	Prec'n	Recall	$F_{\beta=1}$	Prec'n	Recall	$F_{\beta=1}$
LOC	**76.13**	72.66	74.36	58.81	**86.45**	70.00	70.17	86.15	**77.34**
MISC	**58.40**	10.40	17.65	36.76	**19.37**	**25.37**	2.08	0.14	0.27
ORG	**74.72**	**59.24**	**66.08**	68.03	48.16	56.40	69.86	27.63	39.60
PER	**87.14**	82.99	**85.02**	82.45	**83.36**	82.90	73.13	76.07	74.57
Overall	**78.95**	63.93	**70.65**	66.15	**65.97**	66.06	70.51	55.36	62.03

https://medium.com/@boab.dale/text-analytics-apis-part-1-the-bigger-players-3ce8a93577bd

2 Scikit-Learn

2.1 Vectorisation

Feature Extraction and Vectorizers

- Often we want to convert text into a vector.
- This process is called *feature extraction*.
- This way it can be fed to data analytics modules.
- Machines do not really understand text but they are good at numbers.

What we say to dogs
Okay, Ginger! I've had it! You stay out of the garbage! Understand, Ginger? Stay out of the garbage, or else!
What they hear
blah blah GINGER blah blah blah blah blah blah blah blah GINGER blah blah blah blah blah...

Vectorizers in Scikit-Learn
https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

CountVectorizer

- Each vector element represents a word in the vocabulary.
- The vector element indicates the count of the word in the document.

TfidfVectorizer

- Each vector element indicates the $tf.idf$ value of the word.

HashingVectorizer

- Uses the *hashing trick* to handle large text collections.
- Now, we do not know what word is mapped to what vector element.
- It can still give good results in practice, at the expense of interpretability.

A Simple Example

Training a CountVectorizer
The following example uses a small corpus with just 4 sentences to train a CountVectorizer. In the process, sklearn determines the vocabulary and assigns each element of the vector to one word from the vocabulary.

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> vectorizer = CountVectorizer()
>>> corpus = [
...     'This is the first document.',
...     'This is the second second document.',
...     'And the third one.',
...     'Is this the first document?',
... ]
>>> X = vectorizer.fit_transform(corpus)
```

Printing the vocabulary and the TfIdf matrix
In the following example, note how we need to use .toarray() to print the document matrix. This is because X is a sparse matrix. In the printed matrix, each row indicates one document, and each column indicates the counts of the word in each document.

```
>>> vectorizer.get_feature_names()
['and', 'document', 'first', 'is', 'one', 'second',
 'the', 'third', 'this'])

>>> X.toarray()
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 2, 1, 0, 1],
       [1, 0, 0, 0, 1, 0, 1, 1, 0],
       [0, 1, 1, 1, 0, 0, 1, 0, 1]])
```

Finding the document vector of new text
Only words in the vocabulary will be used to build the vectors. So it is important that the corpus used to train the vectoriser has a representative vocabulary.

```
>>> vectorizer.transform(
        ['Something completely new.']
    ).toarray()
array([[0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> vectorizer.transform(
        ['A document with repeated document word']
    ).toarray()
array([[0, 2, 0, 0, 0, 0, 0, 0, 0]])
```

Computing document similarities
Once we have converted document to vectors, we can compute document similarities, e.g. pairwise cosine similarities.

```
>>> from sklearn.metrics.pairwise import cosine_similarity
>>> cosine_similarity(X)
array([[1.        , 0.63245553, 0.2236068 , 1.        ],
       [0.63245553, 1.        , 0.1767767 , 0.63245553],
       [0.2236068 , 0.1767767 , 1.        , 0.2236068 ],
       [1.        , 0.63245553, 0.2236068 , 1.        ]])
```

Example with TfidfVectorizer

The TfidfVectorizer uses the same API as the CountVectorizer.

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
>>> X_tfidf = vectorizer.fit_transform(corpus)
>>> X_tfidf.toarray()
array([[0.        , 0.43877674, 0.54197657, 0.43877674, 0.        ,
        0.        , 0.35872874, 0.        , 0.43877674],
       [0.        , 0.27230147, 0.        , 0.27230147, 0.        ,
        0.85322574, 0.22262429, 0.        , 0.27230147],
       [0.55280532, 0.        , 0.        , 0.        , 0.55280532,
        0.        , 0.28847675, 0.55280532, 0.        ],
       [0.        , 0.43877674, 0.54197657, 0.43877674, 0.        ,
        0.        , 0.35872874, 0.        , 0.43877674]])
>>> cosine_similarity(X_tfidf)
array([[1.        , 0.43830038, 0.1034849 , 1.        ],
       [0.43830038, 1.        , 0.06422193, 0.43830038],
       [0.1034849 , 0.06422193, 1.        , 0.1034849 ],
       [1.        , 0.43830038, 0.1034849 , 1.        ]])
```

HashingVectorizer

- For large data sets, just keeping the large vocabulary and intermediate operations in memory can be too expensive.
- sklearn provides HashingVectorizer, which is like CountVectorizer but it does not keep track of a vocabulary.
- Instead, it uses the *hashing trick*:
  - Use a hash function to map a word to a number.
  - Normally, different words will map to different numbers.
  - But in large vocabularies, some words might map to the same number, creating collisions.
  - There is a tradeoff between speed (number of features) and precision/interpretability since now we do not know what word (or words) is represented in each column of the document matrix.

Example of Use of HashingVectorizer

This example defines a HashingVectorizer with 5 features. Normally we want a large number of features. The default is 2^{20}, roughly 1 million features.

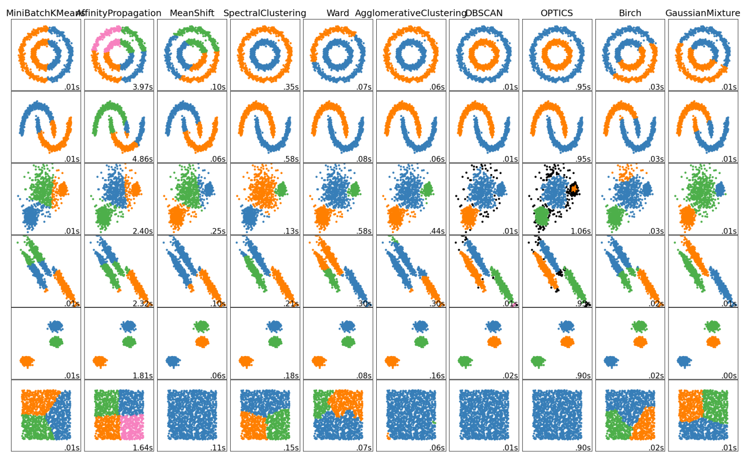
```
>>> from sklearn.feature_extraction.text import HashingVectorizer
>>> hv = HashingVectorizer(n_features=5)
>>> hv.transform(corpus).toarray()
array([[ 0.        , -0.57735027,  0.57735027, -0.57735027,  0.        ],
       [ 0.81649658,  0.        ,  0.40824829, -0.40824829,  0.        ],
       [-0.5        ,  0.5        ,  0.         , -0.5        , -0.5        ],
       [ 0.         , -0.57735027,  0.57735027, -0.57735027,  0.         ]])
>>> cosine_similarity(hv.transform(corpus))
array([[ 1.         ,  0.47140452,  0.         ,  1.         ],
       [ 0.47140452,  1.         , -0.20412415,  0.47140452],
       [ 0.         , -0.20412415,  1.         ,  0.         ],
       [ 1.         ,  0.47140452,  0.         ,  1.         ]])
```

2.2 Processing

Clustering
https://scikit-learn.org/stable/modules/clustering.html

- Clustering is an example of *unsupervised machine learning*.
  - We do not need to manually annotate the data.
  - The system learns to find some structure in the data.
- Clustering identifies groups ("clusters") of documents based on their similarity.
- Clustering needs to know how to compute the similarity between two documents.
  - Several similarity metrics are possible, e.g. Euclidean (distance), cosine (similarity), ...
  - *https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.pairwise*
- sklean provides many types of clustering algorithms.

Sklearn's Cluster Comparison chart



This chart illustrates the result of applying different clustering algorithms to several data sets. For the sake of illustration, each document in the data set is converted into a vector of two elements (so that it can be plotted as a dot in a 2-dimensional space).

An Example with KMeans Clustering
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...                   [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

An Example with Real Text

(see notebook for text clustering of tweets)

Topic Modelling

- Topic modelling is another example of unsupervised machine learning.
- Whereas clustering is a generic algorithm for many kinds of data, topic modelling is specific to text.
- Topic modelling can be used to convert documents to vectors.
- It can be also used to identify specific topics in a document.

Topic Modelling with Latent Dirichlet Allocation

- LDA is a *generative model*.
  - It assumes that each document has been generated by picking random words from a pre-defined set of topics.
- The task of LDA is to try to unravel:
  - For each document, what percentage of words come from each topic.
  - For each topic, what is the distribution of words.

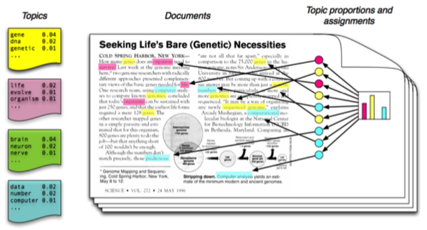


Figure source: Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, *55*(4), 77-84.

LDA in SKLearn

- SKLearn's LatentDirichletAllocation offers the same API as with other tools, plus some specialised methods.
  - *https://scikit-learn.org/stable/auto_examples/applications/plot_topics_extraction_with_nmf_lda.html*
- In addition, pyLADvis is a library that can be used to visualise the topics.
- (See related notebook)

Take-home Messages

- There are a wide range of text demos and APIs on the web. Explore them, use them for this unit's exercises and assignment.
- We have explored how to process text using Scikit-Learn but there are more tools available in Python and other programming languages.

What's Next

Week 10

- Visual Analytics.