Natural Language Processing

Adam Richards

Galvanize, Inc

Updated: 4. Mai 2017

Objectives

Morning

- Introduction
- Processing (vectorization, TF-IDF)
- Document Similarity
- Basic Usage Examples

Afternoon

- Naive Bayes
- Laplace Smoothing
- Example of Multi-class Classification

the packages

This is the one lecture where Python3 is not optional

Ensure conda is up to date

```
conda update --all
```

To install nltk

```
import nltk
nltk.download('all')
```

To install

```
conda install spacy
python -m spacy.en.download
```

Both spaCy and NLTK have large dictionaries (>500MB) so get started now

What is NLP?

- Conversational Agents
  - Siri, Cortana, Google Now, Alexa
  - Talking to your car
  - Communicating with robots
- Machine Translation
  - Google Translate
  - Google's Neural Machine Translation
- Speech Recognition, Speech Synthesis
- Lexical Semantics, Sentiment Analysis
- Dialogue Systems, Question Answering

Conversation Agent Lecture - Dan Jurafsky

Challenges

Ambiguity

- 'Court to Try Shooting Defendant'
- 'Hospitals are sued by seven foot doctors'

What does it mean when we say: 'I made her duck'

- I cooked waterfowl for her
- I cooked waterfowl belonging to her
- I created the (papier mache?) duck she owns
- I caused her to quickly lower her head or body
- I waved my magic wand and turned her into undifferentiated waterfowl

This problem of determining which sense was meant by a specific word is formally known as word sense disambiguation
Other challenges include:

- Part of speech tagging
- Syntactic disambiguation (The I made her duck example)

Speech and Language Processing (Jurafsky and Martin)

Knowledge of language

- Phonetics & Phonology (linguistic sounds)
- Morphology (meaningful components of words)
- Semantics (meaning)
- Pragmatics (meaning wrt goals and intentions)
- Discourse (linguistic units larger than a single utterance)

NLP vocab

- A collection of documents is a corpus
- Each document is a collection of tokens
- Each token is a...?
- ... word
  - Every word? what about 'the' and 'a'?
  - There are different versions of words too

Banks working with that bank on the east bank were banking on a banker's strike

Blocks of *n* words are referred to as n-grams

- little
- little boy
- little boy blue
- little boy blue and the man on the moon

Things to consider

Part of Speech tagging

- Stopwords
- Sentence Segmentation

And more

- N-grams
- Normalization
- New York, POTUS, LDAP ...
- Stemming - the process of reducing words to their word stem, base or root form
- Lemmatization - removes inflectional endings only and to return the base or dictionary form of a word

Stemming and lemmatization both aim to

car, cars, car's, cars' $\Rightarrow$ car

Text Processing

A typical processing procedure might look like:

1. Lower all of your text (although you could do this depending on the POS)
2. Strip out misc. spacing and punctuation
3. Remove stop words (careful they may be domain or use-case specific)
4. Stem/Lemmatize our text
5. Part-Of-Speech Tagging
6. Expand feature matrix with N-grams

Stop words are words which have no real meaning but make the sentence grammatically correct. Words like 'I', 'me', 'my', 'you'...
Scikit-Learn's contains 318 words for the English set of stop words.

What does it do?

Lets say we have some text where each line is a document

```
Oh, the thinks you can think if only you try.
If you try, you can think up a guff going by
And what would you do if you met a jaboo?
```

```
from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS
STOPLIST = ENGLISH_STOP_WORDS
processed = [lemmatize_string(doc, STOPLIST) for doc in corpus]
```

```
['oh think think try',
'try think guff',
'meet jaboo']
```

You will see the lemmatize string function in a few slides

Making text machine consumable

Convert our corpus of text data into some form of numeric matrix representation.

In a Term-Frequency matrix each column of the matrix is a word and each row is a document. Each cell therein contains the count of that word in a document.
Each of the following lines can be a document

```
oh,the,thinks,you,can,think,if,only,you,try
if,you,try,you,can,think,up,a,guff,going,by
and,what,would,you,do,if,you,met,a,jaboo
```

jaboo	if	try	can	you	guff	think	going	would	up	only	thinks	met	oh	by	what	do
0	1	1	1	2	0	1	0	0	0	1	1	0	1	0	0	0
0	1	1	1	2	1	1	1	0	1	0	0	0	0	1	0	0
1	1	0	0	2	0	0	0	1	0	0	0	1	0	0	1	1

Removed stopwords: 'the','a','and'

What problems do you see with this approach?

- If documents are of a different length?
- May have over or underrepresentation issues due to terms with high frequency

One solution is to normalize the term counts by the length of a document which would alleviate some of this problems.
L2 Normalization is the default in 'sklearn'

$$tf(t,d) = \frac{f_{t,d}}{\sqrt{\sum_{i \in V} (f_{i,d})^2}} \tag{1}$$

But we can go further and have the value associated with a document-term be a measure of the importance in relation to the rest of the corpus (TF-IDF)

Term Frequency and Inverse Document Frequency

We need two pieces to calculate TF-IDF

(1) How apparently important was this token in this document?

- Term Frequency: $|t|$ in this doc

(2) How common is this term in general?

- Document Frequency:

$$\frac{|\text{docs containing t}|}{|\text{docs}|} \tag{2}$$

- Inverse the Document Frequency (idf): inverse the previous term and log it (if you want). Then add 1 to the denominator (to avoid divide-by-zero):

$$idf(t, D) : \log\left(\frac{|\text{docs}|}{1 + |\text{docs containing t}|}\right) \tag{3}$$

- TF-IDF is then calculated by multiplying the Term-Frequency by the Inverse-Document-Frequency

The log scale is used so terms that occurs 10 times more than another are not 10 times more important. The 1 term on the bottom is known as a smoothing constant and is there to ensure that we don't have a zero in the denominator.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

How do you adjust thresholds for inclusion/exclusion?

- `max_df` - Can either be absolute counts or a number between 0 and 1 indicating a proportion. Specifies words which should be excluded due to appearing in more than a given number of documents.
- `min_df` - Can either be absolute counts or a between 0 and 1 indicating a proportion. Specifies words which should be excluded due to appearing in less than a given number of documents.
- `max_features` - Specifies the number of features to include in the resulting matrix. If not None, build a vocabulary that only considers the top max_features ordered by term frequency across the corpus.

```
from sklearn.feature_extraction.text import TfidfVectorizer

c_train = ['Here is my corpus of text it says stuff and things',
'Here is some other document']
c_test = ['Yet another document',
'This time to test on']

tfidf = TfidfVectorizer()
tfidf.fit(c_train)
test_arr = tfidf.transform(c_test).todense()

# Print out the feature names
print(tfidf.get_feature_names())
```

Now that we have a matrix representation of our corpus, how should we go about comparing documents to identify those which are most similar to one another?

There are some metrics we already know about.

Cosine Similarity

$$similarity = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4}$$

Euclidean Distance

$$d(A, B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2} \tag{5}$$

```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import euclidean_distances
```

sapCy

The spaCy package is an industrial-strength Natural Language Processing tool in Python. spaCy can be used to perform lemmatization, part-of-speech tagging, sentence extraction, entity extraction, and more; all while excelling at large-scale information extraction tasks. Leveraging the power of Cython, spaCy is the fastest syntactic parser in the world and is capable of parsing over 13,000 words per minute.

https://spacy.io/docs/api#benchmarks

spaCy - lemmatization

```
nlp = spacy.load('en')
doc = nlp("And what would you do if you met a jaboo?")
lemmatized_tokens = [token.lemma_ for token in doc]
print(lemmatized_tokens)
```

```
['and', 'what', 'would', 'you', 'do', 'if', 'you', 'meet', 'a', 'jaboo',
      '?']
```

spaCy visualization tool

spaCy - sentences

```
def sentence_list(doc):
    ''' Extract sentences from a spaCy document object'''

    sents = []
    # The sents attribute returns spans which have indices into the
     original
    # spacy.tokens.Doc. Each index value represents a token
    for span in doc.sents:
        sent = ''.join(doc[i].string for i in range(span.start,
                        span.end)).strip()
        sents.append(sent)
    return sents
```

```
import sys,spacy
from string import punctuation
from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS

if not 'nlp' in locals():
    print("Loading English Module...")
    nlp = spacy.load('en')

def lemmatize_string(doc, stop_words):
    if sys.version_info.major == 3:
        PUNCT_DICT = {ord(punc): None for punc in punctuation}
        doc = doc.translate(PUNCT_DICT)
    else:
        # spaCy expects a unicode object
        doc = unicode(doc.translate(None, punctuation))

    # Run the doc through spaCy
    doc = nlp(doc)

    # Lemmatize and lower text
    tokens = [token.lemma_.lower() for token in doc]

    return ' '.join(w for w in tokens if w not in stop_words)
```

word2vec

- Group of related models that are used to produce word embeddings
- Typically they are two-layer neural networks that are trained to reconstruct linguistic contexts of word
- Input is a large corpus of text and output is a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.
- Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space

NIPS word2vec paper

Extract Similar Words Using Vector Representation

```
import numpy as np

def get_similar_words(wrd, top_n=10):
    token = nlp(wrd)[0]
    if not token.has_vector:
        raise ValueError("{} doesn't have a vector representation".
     format(wrd))

    cosine = lambda v1, v2: np.dot(v1, v2) / (norm(v1) * norm(v2))

    # Gather all words spaCy has vector representations for
    all_words = list(w for w in nlp.vocab if w.has_vector
                     and w.orth_.islower() and w.lower_ != token.lower_)

    # Sort by similarity to token
    all_words.sort(key=lambda w: cosine(w.vector, token.vector))
    all_words.reverse()

    print("Top {} most similar words to {}:".format(top_n, token))
    for word in all_words[:top_n]:
        print(word.orth_, '\t', cosine(word.vector, token.vector))
```

Review

- What are stop words and where do we get some?
- What does it mean to stem or lemmatize your text?
- What is an *n*-gram and why might they help?
- Term frequency is useful buy what are some of the issues?
- What does TF-IDF mean and what does it do?
- Can you name a metric to measure the similarity between two documents?
- BONUS: Can someone explain word2vec?

Objectives

Morning

- ✓ Introduction
- ✓ Processing (vectorization, TF-IDF)
- ✓ Document Similarity
- ✓ Basic Usage Examples

Afternoon

- Naive Bayes
- Laplace Smoothing
- Example of Multi-class Classification

Bayes Theorem (again)

Bayes' Theorem allows us to switch around the events X and Y in a $P(X|Y)$ situation, provided we know certain other probabilities.

Derivation:

$$P(A, B) = P(B, A) \tag{6}$$

$$P(A|B) * P(B) = P(B|A) * P(A) \tag{7}$$

$$P(A|B) * P(B) = P(A, B) \tag{8}$$

$$P(A|B) = P(B|A) * P(A)/P(B) \tag{9}$$

$$\tag{10}$$

In English:

$$\text{Posterior} = \frac{\text{Prior} * \text{Likelihood}}{\text{Evidence normalizing constant}} \tag{11}$$

Naive Bayes classifiers

Naive Bayes classifiers are considered naive because we assume that **all words in the string are assumed independent from one another**

While this clearly isn't true, they still perform remarkably well and historically were deployed as spam classifiers in the 90's. Naive Bayes handles cases where our number of features vastly outnumber our data points (i.e. we have more words than documents). These methods are also computationally efficient in that we just have to calculate sums.

Let's say we have some arbitrary document come in, $(w_1, ..., w_n)$, and we would like to calculate the probability that it was from the sports section. In other words we would like to calculate...

$$P(y_c|w_1, ..., w_n) = P(y_c) \prod_{i=1}^{n} P(w_i|y_c)$$

where...

$$P(y_c) = \frac{\sum y == y_c}{|D|}$$

$$P(w_i|y_c) = \frac{count(w_{D,i}|y_c) + 1}{\sum_{w \in V}[count(w|y_c) + 1]} \quad (12)$$

$$= \frac{count(w_{D,i}|y_c) + 1}{\sum_{w \in V}[count(w|y_c)] + |V|} \quad (13)$$

Laplace Smoothing

$$P(y_c|w_{d,1},...,w_{d,n}) = P(y_c)\prod_{i=1}^{n} P(w_{d,i}|y_c) \quad (14)$$

$$log(P(y_c|w_{d,1},...,w_{d,n})) = log(P(y_c)) + \sum_{i=1}^{n} log(P(w_{d,i}|y_c)) \quad (15)$$

Why do we add 1 to the numerator and denominator?

This is called Laplace Smoothing and serves to remove the possibility of having a 0 in the denominator or the numerator, both of which would break our calculation.

Example

Resources

- Some materials came from Erich Wellinger and Dan Wiesenthal
- sklearn working with text tutorial
- Intro to NLP with spaCy tutorial
- spaCy makes it easy to use word2vec
- spaCy deep learning example

References I