Machine Learning 520
Advanced Machine Learning

Lesson 7: Natural Language Processing

Today's Agenda

- Bag of words
- Term frequency inverse document frequency
- Jaccard distance
- Information retrieval
- Sentiment analysis
- Topic modeling

Learning Objectives

- Use a bag-of-words representation for text.
- Use term frequency - inverse document frequency values to represent text
- Use Jaccard distance to measure the similarity between two documents.
- Use TF-IDF to retrieve the most relevant documents for a query.
- Apply classification models for sentiment analysis.
- Use topic model predictions as additional features for classification.

Overview of NLP

- Human **natural language** is a commonly encountered **unstructured data** type
- NLP seeks to extract information from and model human natural language
- Various estimates of what percent of ML solutions use NLP methods Take estimates with a grain of salt
- About 70% of web data is natural language
- In 2019 Gartner group claims 50% of analytical queries involve NLP
- NLP is part of daily life for many people

Motivation: Harnessing Big Text Data

- Internet
- News
- Literature
- Blogs applications
- Email
- Twitter
- Personal assistants
- Etc.

Knowledge

Many

Gaining Intuition:

I'm given a task. What approach will work?

Overview of NLP

- In this lesson we focus on **natural language processing** NLP attempts to model or process natural language
  - The goal of NLP is classification, prediction, and simple generation
  - The machine does not actually have any understanding of meaning
  - NLP is a form of **weak AI**
- We will not address **natural language understanding**NLU is a form of **strong AI**
  - For strong AI, the machine must actually **understands the meaning** of what is being said and what it is saying
  - NLU is the basis of the famous **Touring test**
  - NLU is a research frontier

Why is NLP hard?

- Human language is imprecise
- Computer languages have strict grammar and semantics
- Human languages use loose grammars and ambiguous semantics, at best
- Example; I ask an intelligent agent ‘do you know what time it is?’, what answer do I expect?
  - Yes. –The agent does know the time
  - It is 10:45. –The answer I wanted
- Example; But, if I say, ‘Tell me what time it is.’, the answer is unambiguous
  - A human would not notice a semantic ambiguity between these cases

Main approaches in NLP

- **Rule-based methods**
  - Regular expressions
  - Context free grammar

- **Probabilistic modelling and machine learning**
  - Linear classifiers
  - Likelihood maximization

- **Deep Learning**

- CNN, RNN, LSTM, etc.

Examples (rule-based methods)

> Semantic slot filling

> Context free grammar

> Parsing

Examples (traditional ML based methods)

- Training corpus (with some markups)

- Feature engineering
  - Is the word capitalized?
  - What are the previous words?
  - etc…

- Define a Probabilistic graphical model

- Conditional Random Field: p(tags|words)

Some useful libraries and tools

- NLTK
  - Small but useful datasets with markup
  - Preprocessing tools: normalization, tokenazition, etc.
  - Pre-trained models for parsing, POS-tagging, etc.
- spaCy: python and cython library for NLP
- Gensim: python library for text analysis (word embedding, topic modelling, etc.)

Text classification

- Example: sentiment analysis
  - Input: text of review
  - Output: class of sentiment (e.g. positive vs negative)

- Positive example:
  - The movie was great with likeable characters
  - The mayor has done a splendid job in revitalizing the downtown

- Negative example:
  - The worst performance by Actor John Doe, no wonder the film bombed
  - Don’t buy this product, run for your wallets!

Organization of text data

- We organize text data in a corpus
- A corpus is a collection of documents
- What constitutes a document?
- Most any unit of text can be a document for NLP
    - An entire book, a chapter from a book, a paragraph, or a single sentence
    - Email, text message, or tweet
    - Blog post
    - Tweet
    - Movie review
    - A contract
    - Instructions to assemble my new office chair
    - …

Text preprocessing

- What is text?
  - Sequence of words

- What is a word?
  - A single distinct meaningful element of speech or writing, used with others (or sometimes alone) to form a sentence and typically shown with a space on either side when written or printed (Ref: Oxford dictionary).

Input: The dog ate the fish, then took a nap.

Output: The dog ate the fish then took a nap

Text preprocessing

- German has some compound words which are written without space

rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz

that's 63 letters long meaning "the law for the delegation of monitoring beef labeling.”

- In Chinese there are no spaces at all!

如果你能看懂这句话那么你的中文很厉害！

Text Preparation

- Text data is messy
- Considerable preprocessing typically required
- Be careful! Make sure you do not remove the parts you need!
- Normalize text
- Stop words are removed
- Stemming to represent forms of a word as same token
- Limit vocabulary to remove bias from rare words
- Document is tokenized

Text Preparation

- Text Normalize prevents creation of spurious tokens
- Example; for sentence:
  - Is this the right way to solve the problem?
  - We do not want a token ‘problem?’
- Steps of text normalization can include
  - All characters to lower case –‘Shovel’ and ‘shovel’ are same token
  - Remove punctuation
  - Remove numbers and special characters
- Several possible pitfalls
  - Should the punctuation be removed in words like ‘can’t’?
  - Some numbers can be important tokens
  - How should dates be handled?

Tokenization

- Process that split an input sequence into tokens

- Tokens are useful unit for semantic processing (can be a word, sentence, paragraph, etc.)

Example:

```
>>> from nltk.tokenize import word_tokenize
>>> s = '''Good muffins cost $3.88\nin New York.  Please buy me
... two of them.\n\nThanks.'''
>>> word_tokenize(s)
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.',
'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```

Token normalization

- Process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens
  - Dog, dogs → dog
  - Wife, wives → wife
- **Stemming**
  - A process of removing and replacing suffixes to get to the root form of the word, called a *stem*
- **Lemmatization**
  - return the base or dictionary form of a word, which is known as the *lemma*

Example (stemmer)

- Rule

Rule			Example		
SSES	☾	SS	caresses	☾	caress
IES	☾	I	ties	☾	ti
SS	☾	SS	caress	☾	caress
S	☾		cats	☾	cat

```
from nltk.stem.porter import PorterStemmer

example_words = ["maximum","presumably","pythoning","provision","ear"]
ps = PorterStemmer()
for w in example_words:
    print(ps.stem(w))
```

```
maximum
presum
python
provis
ear
```

Problem: fails on irregular forms & produces non words

Example (lemmatization)

- WordNet lemmatizer
  - Uses the WordNet database to lookup lemmas
  - churches ☾ church   dogs ☾ dog
  - Wolves ☾ wolf   maximum ☾ maximum

```python
from nltk.stem import WordNetLemmatizer

example_words = ["churches","dogs","wolves","maximum"]
wordnet_lemmatizer = WordNetLemmatizer()

for w in example_words:
    print(wordnet_lemmatizer.lemmatize(w))
```

```
church
dog
wolf
maximum
```

Problem: not all words are reduced

Stemming Vs. Lemmatization

When to use what!

- If you need speed, then use stemming
- If you have time, then use lemmatizers as it scans a corpus
- If you are building a language application, then use lemmatization to ensure match to root forms.

Tokens to features

- Most common ways to extract numerical features from text content
  - **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.
  - **normalizing** and weighting with diminishing importance tokens that occur in the majority of samples / documents.

Ref: http://scikit-learn.org/stable/modules/feature_extraction.html

Tokens to features

- Most common ways to extract numerical features from text content
  - **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.
  - **normalizing** and weighting with diminishing importance tokens that occur in the majority of samples / documents.

- In this scheme, features and samples are defined as follows:
  - each **individual token occurrence frequency** (normalized or not) 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 and one column per token (e.g. word) occurring in the corpus.

Tokens to features

Great instructor
Not a great instructor
Do not like

great	instructor	not	a	do	like
1	1	0	0	0	0
1	1	1	1	0	0
0	0	1	0	1	1

- **Limitations**:
  - Word orders is lost hence BOW cannot capture phrases and multi-word expressions
  - BOW model doesn't account for potential misspellings or word derivations.

Tokens to features

➢ N-grams

- Instead of building a simple collection of unigrams (n=1), one might prefer a collection of bigrams (n=2), where occurrences of pairs of consecutive words are counted.

Great instructor
Not a great instructor
Do not like

→

Great instructor	instructor	Do not	a	Not like	…
1	1	0	0	0	…
1	1	0	1	0	…
0	0	1	0	1	…

```python
#Example character 2-gram representation
from sklearn.feature_extraction.text import CountVectorizer

ngram_vectorizer = CountVectorizer(analyzer='char_wb', ngram_range=(2, 2))
counts = ngram_vectorizer.fit_transform(['words', 'wprds'])
#ngram_vectorizer.get_feature_names() == ([' w', 'ds', 'or', 'pr', 'rd', 's ', 'wo', 'wp'])
counts.toarray().astype(int)
```

```
array([[1, 1, 1, 0, 1, 1, 1, 0],
       [1, 1, 0, 1, 1, 1, 0, 1]])
```

Tokens to features

Reduce the number of features in n-grams

- High frequency n-grams (e.g. stop words)
- Low frequency n-grams (e.g. typos)
- Medium frequency n-grams are usually good n-grams

Problem: If we were to feed the direct count data directly to a classifier very frequent terms would shadow the frequencies of rarer yet more interesting terms. Therefore we need to re-weight the count features into floating point values suitable for usage by a classifier

Tokens to features

- Text Frequency: Word frequency or TF
  - Count how many times a word appears in a single document.
  - Terms that occur in fewer documents are more descriptive and may contain more information (Rarity matters).
- Inverse Document Frequency (IDF)
  - Inverse of the proportion of documents containing term in the whole collection.
  - #documents / #documents with word might be too severe.
    - A word appearing twice instead of once shouldn’t have twice the impact

$$IDF = \log\left(\frac{¿\,Documents}{¿\,Documents\,wit\,h\,Word}\right)$$

- Maybe we should tie these together:
- TF-IDF: Rare terms in whole collection that appear frequently in some documents maybe very important!
  - Multiply these two

How to Use TF-IDF

$$TF-IDF=\log\left(\frac{\text{¿}\,Documents}{\text{¿}\,Documents\,with\,Word}\right)\times f(Word)$$

- Finding important words to describe the document collections or subgroups of collections.
- Using the count of important words as a feature in a model.
- Using the distribution of a document’s TF-IDF values.
  - Characterize writing styles
  - Comparing authors
  - Determining original authors
  - Finding plagiarism

Example

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

text = ["good instructor", "not a good instructor", "do not like", "I like him", "good one"]
tfidf = TfidfVectorizer(min_df=2, max_df=5, ngram_range=(1,2))
features = tfidf.fit_transform(text)
pd.DataFrame(features.todense(),
            columns = tfidf.get_feature_names())
```

	good	good instructor	instructor	like	not
0	0.506204	0.609818	0.609818	0.000000	0.000000
1	0.432183	0.520646	0.520646	0.000000	0.520646
2	0.000000	0.000000	0.000000	0.707107	0.707107
3	0.000000	0.000000	0.000000	1.000000	0.000000
4	1.000000	0.000000	0.000000	0.000000	0.000000

Document classification

- Document classification is a common NLP application
- Many applications of document classification
- Organize articles; sport, business, local, weather,…
- Organize books; romance, mystery, business, science,…
- Detect SPAM emails
- Determine sentiment in reviews and social media
- Detect language a document is written in
- …

Notebook Time

Language modeling

Assign a probability to a sentence

Why?

- Machine translation (e.g. P(the **average** probability) > P(the **mean** probability))
- Spell correction (e.g. P(I'm **enjoying** this class) > P(I'm **enjoiing** this class))
- Speech recognition (e.g. P(I'm gonna take a walk)> P(am gonna take a wok)
- Handwriting recognition
- Etc.

Language modeling

- Compute the probability of a sentence or sequence of words
- Compute the probability of an upcoming word
- A model that computes either of these or is called a **language model**.

Example

Toy corpus

- This is the house that Jack built.
- This is the malt
- That lay in the house that Jack built.
- This is the rat,
- That ate the malt
- That lay in the house that Jack built.
- This is the cat,
- That killed the rat,
- That ate the malt
- That lay in the house that Jack built.

What is P(house | this is the) ?

Example

Language modeling intuition

- Predict probability of a sequence of words
  -)

- Chain rule:

- Markov assumption:

Bigram language model

- For

- Example:
  - This is the malt
  - That lay in the house that Jack built

Bigram language model summarized

- Define the model

- Estimate the probabilities:

N-gram language model summarized

- Define the model

How to train n-gram models

- Log-likelihood maximization:

- Estimates for parameters:

- Where is the length of the train cor pus.

Example Generated Shakespeare

- Accuracy increases as N increases
  - Train various N-gram models and then use each to generate random sentences.
- Example Corpus: Complete works of Shakespeare
  - Unigram: To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have.
  - Bigram: What means, sir. I confess she? Then all sorts, he is trim, captain.
  - Trigram: Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ‘tis done.
  - Quadrigram: King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch.

Evaluating Language Models

- The best might depend on how much data you have
  - Bigrams might not be enough
  - 8-grams might never occur

- **Extrinsic evaluation**:
  - Best evaluation for comparing model A and B
    - Put each model in a task(e.g. machine translation, spelling corrector, etc.)
    - Run the task, get an accuracy for A and for B (e.g. how many misspelled words corrected properly, how many words translated correctly)
    - Compare accuracy for A and B

Evaluating Language Models

- **Extrinsic evaluation**:
  - Time-consuming (can take days, weeks, etc.)

  - How to address that?
    - Use **intrinsic evaluation**: **perplexity**
    - **Perplexity** is the inverse of the probability of the test set (as assigned by the language model), normalized by the number of word tokens in the test set.

Evaluating Language Models

Likelihood:

Perplexity: