Natural Language Processing

Lecture 1

Spring 2023

RowanUniversity

Instructor Introduction

- Me: Alexander Kalinowski (please, call me Alex!), Pronouns: he/him
- BA Mathematics, MS Data Science, PhD Information Science
- Industry experience: start-ups, finance, animal health
- Research experience: NLP and knowledge graph information fusion
- South Jersey native, happy to be teaching close to home!

Student Introduction

- Name and pronouns
- Where and what you studied for undergrad
- Your current program at Rowan
- Why you are signed up for this course

NLP Example

Figure: Sunspring

Why is NLP Hard?

- Ambiguity
- Evolution
- Culture and bias
- Dynamics

Course Objectives

- Describe the methods for preprocessing and cleaning text data
- Implement feature extraction methods over natural language
- Utilize NLP features in other ML pipelines
- Familiarize with modern NLP architectures, such as Transformers
- Build comfortability reading state-of-the-art NLP research

Grading Schedule

- Four homework assignments
  - Week 1: HW 2 assigned
  - Week 5: HW 1 collected (10% HW grade)
  - Week 7: Project Proposal due
  - Week 10: HW 2 in-class assignment (10% HW grade)
  - Week 16: Final project presentations
- Total HW (20%)
- Course project (40%)
- Presentations and contribution to course (40%)

Textbooks

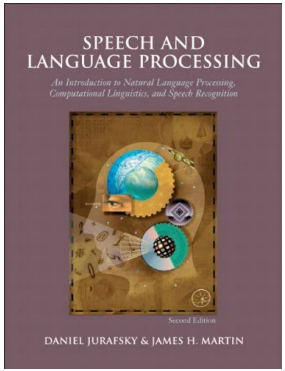

Figure: Speech and Language Processing

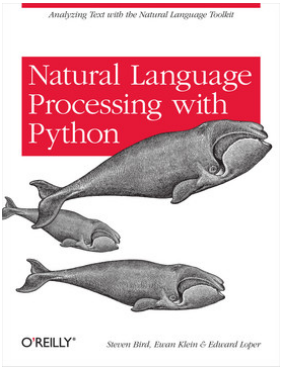

Figure: NLTK Textbook

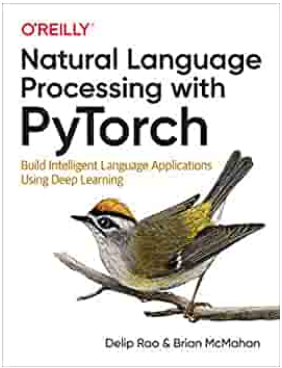

Figure: NLP in PyTorch

Coding Exercises

- All coding exercises in this course will be performed in Python
- Coding philosophy: notebooks do not lead to good practices
- Scripts, classes and functions are better!
- Anaconda environments will be used, all will be included in the course github
- Proper code formatting and documentation is a must: see PEP8

Open Source Packages for NLP

- NLTK: natural language toolkit, good for small-scale linguistic analysis
- Gensim: topic modelling and advanced models
- PyTorch: general deep learning framework, overtaking Tensorflow/Keras
- Huggingface: nice APIs for large-scale, pre-trained language models
- IaMaN: experimental framework using analytical optimization [in development]

Questions?

Processing Text Data

Objectives

- Regular expressions for text parsing
- Brief aside on automata and FSA/FST [raise your hand if you remember these]
- Tokenization, normalization, stemming, BPE
- String distance calculations
- **TL;DR: cleaning text**

Terminology

Definition

A **character** is the atomic-unit of text, a printable symbol having phonetic or pictographic meaning and usually forming part of a word of text, depicting a numeral, or expressing grammatical punctuation.

Definition

A **token** is a sequence of characters that represents a fundamental linguistic unit.

Definition

A **corpus** is a collection of natural language documents. The plural of corpus is **corpora**. There are many generally available free/licensed corpora that are used in natural language processing. Those that are leveraged by many researchers to compare and contrast results are called **benchmark datasets**.

Regular Expressions

Definition

A **regular expression** is a *formal language* for describing strings of data.

Example

$[A - Z]$ matches an upper case letter.

Example

$[a - z]$ matches a lower case letter.

Example

$[0 - 9]$ matches a digit.

Formal Languages

Definition

In automata theory, a **formal language** is a set of strings of symbols drawn from a finite alphabet. A formal language can be specified either by a set of rules (such as regular expressions or a context-free grammar) that generates the language, or by a formal machine that accepts (recognizes) the language.

Theorem

A language is regular if and only if (iff) some regular expression describes it.

Corollary

Regular expressions and finite automata are equivalent expressions of information. (aka for every regex, you can draw a finite automata! hence, regex, NLP and theory of computation are all intertwined!!)

Practical Regex

```
def remove_pii(_text):
    """ A function to strip the documents of the CFPB scrubbed PII.
    :param _text: An individual document.
    :return: The same document without the masking strings.
    """
    return re.sub(r'\s*X+', '', _text)
```

Figure: Regex for PII

Tokenization

- Dissect sentences from natural language corpora
- Split resulting sentences into individial tokens
- Further pre-process tokens into ‘atomic’ units

Exercise

I called the doctors' office to get an update, but Dr. Ham wasn't available. This is related to a crazy car accident. Can you ask him to call me at 447-700-7000? I'm in urgent need of care by a capable, caring, caregiver.

Tokens and Types

Definition

A **token** is a sequence of characters that represents a fundamental linguistic unit.

Definition

A **type** is the set of all unique tokens, also known as the **vocabulary**. In Python, if we have a list of tokens T, the vocabulary is simply $V = list(set(T))$.

Tokens and Types

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{½})$

	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Figure: Comparison of token and vocabulary sizes

Issues in Tokenization

- Dr. Ham $\rightarrow$ is Dr. the end of a sentence?
- doctors' office $\rightarrow$ is s' a valid token?
- New York City $\rightarrow$ $\{New, York, City\}$ or New_York_City?
- Agglutinative languages form words through the combination of smaller morphemes to express compound ideas:
  - German: Lebensversicherungsgesellschaftsangestellter
  - English: Life insurance company employee

Compression and BPE

Another option for text tokenization

Instead of

- white-space segmentation
- single-character segmentation

Use the data to tell us how to tokenize.

Subword tokenization (because tokens can be parts of words as well as whole words)

Compression and BPE

Subword tokenization

Three common algorithms:

- **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
- **Unigram language modeling tokenization** (Kudo, 2018)
- **WordPiece** (Schuster and Nakajima, 2012)

All have 2 parts:

- A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
- A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Compression and BPE

Byte Pair Encoding (BPE) token learner

Let vocabulary be the set of all individual characters

= {A, B, C, D,…, a, b, c, d….}

Repeat:

- Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
- Add a new merged symbol 'AB' to the vocabulary
- Replace every adjacent 'A' 'B' in the corpus with 'AB'.

Until *k* merges have been done.

Compression and BPE

BPE token learner

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _	
6 n e w e r _	
3 w i d e r _	
2 n e w _	

Merge e r to er

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2 l o w e s t _	
6 n e w er _	
3 w i d er _	
2 n e w _	

Probabilistic Language Modelling

Objectives

- Unsmoothed N-gram models
- Perplexity
- Simple smoothing methods
- **TL;DR: imputing probabilities from text**

LMs

- Goal: generate the probability of a sequence of tokens
- $P(S) = P(w_1, w_2, \ldots, w_n)$
- OR, given some partial sequence of words, predict the next word
- $P(w_5|w_1, w_2, w_3, w_4)$
- Either way, we are generating a **language model**

Probability Estimation

- How do we compute such probabilities?
- Answer: estimate them from *relative frequency counts*
- Example $P(that|its\ water\ is\ so\ transparent)$
- Simple solution: use a large corpus and count:
  - C_d number of times we see the phrase *its water is so transparent*
  - C_n number of times we see the phrase *its water is so transparent that*
- The relative frequency is then $\frac{C_n}{C_d}$
- Any immediate issues you can see?

Joint Probabilities

- We can also ask a LM: what is the probability of a given sequence S?
- Could estimate this from how often the sequence S appears as a ratio to the count of *all other sequences of* $len(S)$ (this would be a big ask!)
- Instead, leverage the **chain rule of probability**
- $p(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1})$
- Equivalent to the product notation $\Pi_{k=1}^{n} P(w_k|w_1^{k-1})$
- These probabilities of individual words are called unigrams

From Unigram to Bigram

- Chain rule helps, but not much... still need to do many, many counts of long, long sequences!
- Relax the requirement of needing the entire sequence history and instead rely on a smaller substring to *estimate* these probabilities
- Easiest way: let the probability of a word depend only on the conditional probability of the previous word using **the Markov Assumption**
- $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$

Unigram Example

Simplest case: Unigram model

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

```
fifth, an, of, futures, the, an, incorporated, a,
a, the, inflation, most, dollars, quarter, in, is,
mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
```

Bigram Example

Bigram model

- Condition on the previous word:

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november

N-gram Example

N-gram models

We can extend to trigrams, 4-grams, 5-grams

In general this is an insufficient model of language

- because language has **long-distance dependencies**:

  "The computer which I had just put into the machine room on the fifth floor crashed."

But we can often get away with N-gram models

Evaluating N-gram Models

Evaluation: How good is our model?

Does our language model prefer good sentences to bad ones?

- Assign higher probability to “real” or “frequently observed” sentences
  - Than “ungrammatical” or “rarely observed” sentences?

We train parameters of our model on a **training set**.

We test the model’s performance on data we haven’t seen.

- A **test set** is an unseen dataset that is different from our training set, totally unused.
- An **evaluation metric** tells us how well our model does on the test set.

Perplexity

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

Perplexity Demo

Lower perplexity = better model

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Overfitting N-grams

The perils of overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus

- In real life, it often doesn't
- We need to train robust models that generalize!
- One kind of generalization: Zeros!
  - Things that don't ever occur in the training set
    - But occur in the test set

Zero Bigrams

Zero probability bigrams

Bigrams with zero probability

- mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity (can't divide by 0)!

Laplace smoothing

Add-one estimation

Also called Laplace smoothing

Pretend we saw each word one more time than we did

Just add one to all the counts!

MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

With/Without Smoothing

Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Other options

Backoff and Interpolation

Sometimes it helps to use **less** context

- Condition on less context for contexts you haven't learned much about

Backoff:

- use trigram if you have good evidence,
- otherwise bigram, otherwise unigram

Interpolation:

- mix unigram, bigram, trigram

Interpolation works better