Jim Martin

**NLP** is the study what goes into getting computers to perform **useful** and **interesting** tasks involving human language.

**Computational linguistics** is the use of computer models to provide **insight** into the nature of human language

More specifically, it's about the **algorithms** used to process language, the **rationale** for those algorithms, and the relevant **facts about human language** that allow those algorithms to work.

Why important? An enormous amount of text is available in machine readable form: newspapers, web pages, medical record.

Advance in speech recognition make conversational agents a key form of human-machine interaction

Much Of human-human interaction is now mediated by computers and is therefore **potentially available for analysis**.

NLP Application:

Information extraction:

2008 United Stock Crash: when news about United, if bad news then sell

Dialog Agents:

Combine speech recognition and synthesis to accomplish some restricted task

Chatbots

Machine Translation:

Fully automatic translation of texts between languages is one of the oldest applications in Computer Science.

MT has gone from a niche academic curiosity to a robust commercial enterprise, Google Translation

Input/Outputs: take in speech/text as input and generate language as output. For IE system, producing some form of structured representation as output.

How? All of these applications operate by **discovering** ways to exploit regularities underlying human languages.

**3 general ways of viewing NLP problems:**

**Classifying texts according to a given set of categories (Text/sequence classification)**

**Viewing texts as sequences and labeling the elements of the sequence (Token labeling/classification)**

**Structure building: taking in texts and building structured representation from them (graphs and trees)**

Machine learning: Supervised machine learning approaches, and neural networks in particular, are the dominant way to build NLP systems.

Why NLP so hard? Anne Hathaway Effect

**Ambiguity**: I made her duck

I cooked waterfowl for her benefit to eat

I cooked waterfowl belonging to her

I created the duck she owns

I caused her to quickly lower her upper body

I waved my magic wand and turned her into undifferentiated waterfowl

Lexical category: duck can be a noun or verb; her can be possess

5 assignments 40%

Shared programming task 15%

3 quizzes 40%

Participation 5% Pizaa

Words

In modern NLP systems, words are the fundamental bridge from linguistic form to underlying meaning

Managing works and their meaning will be fundamental problem that we'll have to address

Counting -> Priority -> Probability

Type: an entry in a vocabulary

Token: an instance of a type in some text

Lemmatization: reduce inflected forms to base forms

Morphology is the study of the ways that words are built up from smaller units called morphemes

We can divide morphemes into two classes

Stems: The core meaning-bearing unit

Affixes: Bits and pieces that adhere to stems to change their meanings and grammatical function

Inflectional morphology: has the same word class as the original; Different from the original; transparently related to the original

Derivational morphology is the messy stuff that no one ever taught you

How many morphemes per word? Isolating vs. Synthetic; isolating is one morpheme per word

How hard is it to find morpheme boundaries? Agglutinating(concatenates) vs. fusional (change their form)

fromtransformersimportBertTokenizer,BertModel

tokenizer=BertTokenizer.from\_pretrained('bert-base-uncased')

tokenizer.get\_vocab()

sequence="AdamismyfavoriteTAincsci-5832,RehanisokIguess"

tokenizer.tokenize(sequence)

['adam',

'is',

'my',

'favorite',

'ta',

'in',

'cs',

'##ci',

'-',

'58',

'##32',

',',

're',

'##han',

'is',

'ok',

'i',

'guess']

#Foreachtokeninthevocabulary...

fortokenintokenizer.vocab.keys():

#Recordanysingle-charactertokens.

iflen(token)==1:

one\_chars.append(token)

#Ifit'sasubword...

iflen(token)>=2andtoken[0:1]=='#':

#Tallyallsubwords

num\_subwords+=1

#Tallyifit'sanumber.

iftoken.isdigit():

count+=1

Chapters 2, 3, 4, 5 and 8

Chapter 2: Skip  Section 2.5

Chapter 3: Skip 3.6 and 3.8

Chapter 4: Skip 4.9

Chapter 5: Skip 5.5 and 5.8

Chapter 8: Skip 8.5 and 8.7

Calculate derivate = -0.31

So gradient = -0.31 \* [ 3,2,1,3,0,4.15] input

New\_weights = orignial weights + gradient

We also want to update bias

Add 0.1 to weights and add [1] to input

Batch training

Process each example in the training set and accumulate the gradients

Do a single update

Repeat

Minibatch training

Select N examples and proceed as with batch training

Update after each mini-batch

N is chosen to maximize parallelism

Adjust the learning rate during training.

Bootstrap test: sample with replacement

“Paired tests” are a common approach used in NLP

Compare two sets of observations in which each observation in one set can be paired with an observation in another.

For example, when looking at systems A and B on the same test set, we can compare item for item the performance of system A and B

We have 10,000 test sets x(i) and a threshold of .01

And in 47 of the test sets we find that δ(x(i)) ≥ 2δ(x)

The resulting p-value is .0047

This is smaller than .01, indicating δ (x) is indeed sufficiently surprising

And we reject the null hypothesis and conclude A is better than B

Vocabulary

**Corpora**:

Tokens = N = number of instances or tokens in a corpus 20000-30000

Types = V = vocabulary = set of unique types. 3500

**Heaps' Law**: the vocabulary grows faster than the square root of the corpus size

the rate of growth of the vocabulary tails off as the corpus grows but never completely flattens out.

The size of a vocabulary grows without bound as a function of corpus size.

Tokenization: Language issues: noun compounding; no spaces between words; word segmentation

Deal with out-of-vocabulary (OOV) terms and still keep the vocabulary size reasonable

**Subword tokenization:** -est, -er;

Three common algorithms:

Byte-Pair Encoding (BPE)

Unigram Language modeling tokenization

WordPieces

All have 2 parts:

A LEARNER that takes a raw training corpus and induces a vocabulary consisting of tokens (words and subwords)

A token Segmenter that takes an input and tokenizes it according to a vocabulary.

Words present the vocabulary are left alone (unsegmented)

OOV words are broken into optimal sequences of words and subwords.

BPE:

ABCD…,abcd…,0-9,etc.

Choose two symbols that are most frequently adjacent in training corpus (concatenating)

add 'ab' to the vocabulary

Replace every ab in the corpus with AB

Until k merges have been done

* 1. low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new
  2. low \_

lowest \_

lowest\_

newer\_

newer\_

newer\_

wider\_

* 1. l o w \_

l o w e s t \_

l o w e s t \_

n e w e r \_

n e w e r \_

n e w e r \_

w i d e r \_

* 1. \_\_,d,e,I,l,n,o,r,s,t,w
  2. \_\_,d,e,I,l,n,o,r,s,t,w,er
  3. \_\_,d,e,I,l,n,o,r,s,t,w,er, er\_\_
  4. \_\_,d,e,I,l,n,o,r,s,t,w,er, er\_\_, ne
  5. \_\_,d,e,I,l,n,o,r,s,t,w,er, er\_\_, ne, new
  6. \_\_,d,e,I,l,n,o,r,s,t,w,er, er\_\_, ne, new, lo
  7. \_\_,d,e,I,l,n,o,r,s,t,w,er, er\_\_, ne, new, lo, low
  8. \_\_,d,e,I,l,n,o,r,s,t,w,er, er\_\_, ne, new, lo, low, newer\_\_
  9. \_\_,d,e,I,l,n,o,r,s,t,w,er, er\_\_, ne, new, lo, low, newer\_\_, low\_\_

With BPE (and other subword approaches) there are never any out of vocabulary words. Every word can be decomposed into a sequence of known vocabulary items (sequences of words and subwords, or worst case, characters).

BERT Vocabulary 30k entries, generated from Books corpus and an english wikipedia dump

Zero counts:

Zipf's law (long tail phenomenon):

A small number of events occur with high frequency

A large number of events occur with low frequency

We can quickly collect high frequency but wait long time get low frequency

Our estimate are sparse

Laplace Smoothing:

Add-one smoothing, just add one to all the counts

P(W) = c + 1 / N+V

However, P(to|want) from 0.66 to 0.26; c(want to) from 608 to 238

Laplace still used for pilot studies; in document classification; in domains where the number of zeros isn't so huge

Determining how many species occupy a particular area (how many types)

Determining how many individuals of a given species are living in each area (tokens)

Backoff

Using lower order N-grams when counts are lacking for higher-order N-grams

Interpolation

Mixing unigram, bigram, trigram probabilities

Discounting

Stealing from the rich

We set the lambdas use a held-out corpus

Absolute discounting, subtract a small fixed amount from all the observed counts

Split training data in half; get bigram counts from the first half of data; observe how often on average

P\_absolutediscounting(wi|wi-1) = c()-d/c() + interpolation\_weight \* unigram\_prob

Caveats.

Smoothing N-gram counts is directed at N-grams consisting of words that have occurred.

Dealing with unknown words (OOV) is a separate issue.

The more data you have the less important the choice of smoothing method

Text classification (Naïve bayes classifier)

Cmap = argmaxP(xi|c)P(c)

Bag of words assumption: sequence doesn’t matter

Features are just word occurrences

Conditional independence: assume the feature probabilities are independent given the class c

Naive bayes text classifiers use all sorts of additional features, not just the words

A picture containing text, clock

Description automatically generated

A screen with numbers and letters on it

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with low confidence

Precision = tp/tp+fp ; recall = tp/tp+fn; accuracy = tp+tn/tp+fp+tn+fn

Combined measure: F, weighted harmonic mean, F1 = 2PR / P + R; P for precision, r for recall

If we have more than one class, how do we combine multiple performance measures into one quantity?

Macro-averaging: Compute performance for each class, then average.

Micro-averaging: Collect decisions for all classes, compute contingency table, evaluate.

|  |  |  |
| --- | --- | --- |
|  | Truth: yes | Truth: no |
| Classifier: yes | 10 | 10 |
| Classifier: no | 10 | 970 |
|  | Truth: yes | Truth: no |
| Classifier: yes | 90 | 10 |
| Classifier: no | 10 | 890 |
|  | Truth: yes | Truth: no |
| Classifier: yes | 100 | 20 |
| Classifier: no | 20 | 1860 |

Macroaveraged precision: (0.5 + 0.9)/2 = 0.7. 10/20 + 90/100

Microaveraged precision: 100/120 = .83.

Sequence labeling involves classifying each token in a longer sequence

Part-of-speech tagging

Assigning the correct word class to each word in sentence.

Named entity recognition

Find all instances of various kinds of entities in an input.

Three sources of evidence

~~Semantics~~

Morphological evidence

walk, walking, walked, walks

Probably a verb

Distributional evidence

The crash, A crash, Two crashes, The big crash…

Probably a noun since nouns follow determiners and adjectives

POS Tagging:

The process of assigning a part of speech or lexical class marker to each word in a text.

Often a useful first step in an NLP pipeline.

Knowing the part of speech of the words in an input is a valuable signal for further processing

Fast and accurate taggers are widely available for many languages

WORD tag

the DET

koala NOUN

put VERB

the DET

keys NOUN

on ADP

the DET

table NOUN

Words can have more than one part of speech: back

The back door = ADJ

On my back = NOUN

Win the voters back = ADV

Promised to back the bill = VERB

The POS tagging problem is to determine the tag for a particular instance of a word in context

Usually for a sentence

词性 词义

Note this is distinct from the task of identifying which sense of a word is being used given a particular part of speech. That’s called word sense disambiguation.

“backed” is a verb in both of these examples

“… backed the car into a pole”

“… backed the wrong candidate”

So there are 47k words in the vocab. 38k of them only have one POS. 6700 have 2. Why is this so hard?

The words that do have more than one tag are also the most frequently occurring ones

-Rule-based tagging

-Probabilistic sequence models

HMM (Hidden Markov Model) tagging

Conditional Random Fields

Neural sequence models

How do we make this operational? How to compute this value? Two steps

Use Bayes rule to transform this equation into a new set of equations

Use independence assumptions to make computing these tractable

Tag transition probabilities p(ti|ti-1)

Word likelihood probabilities p(wi|ti).

Secretariat/NNP is/VBZ expected/VBN to/TO race/VB tomorrow/NR

People/NNS continue/VB to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/IN outer/JJ space/NN

P(NN|TO) = .00047

P(VB|TO) = .83

P(race|NN) = .00057

P(race|VB) = .00012

P(NR|VB) = .0027

P(NR|NN) = .0012

P(VB|TO)P(NR|VB)P(race|VB) = .00000027

P(NN|TO)P(NR|NN)P(race|NN)=.00000000032

So we (correctly) choose the verb tag for “race”

If there are 30 or so tags in the Penn set

And the average sentence is around 20 words...

How many tag sequences do we have to enumerate to argmax over?

* + 30 ^ 20

What we’ve just described is called a Hidden Markov Model (HMM)

This is an example of a generative model.

There is a hidden underlying generator of observable events

The hidden generator can be modeled as a network of states and transitions

We want to infer the underlying state sequence given the observed event sequence

The hidden process is the unseen process of part of speech sequences

Modeled as states and transitions between states

The observations are the words that are emitted for each POS

Modeled as emissions from states

Given this HMM framework there are 3 classic problems that we can pose

Given an observation sequence and a model, what is the probability of the sequence?

Computing likelihood, Forward

Given an observation sequence and a model, what is the most likely state sequence?

Decoding, Viterbi

Given an observation sequence, find the best model parameters

Forward-Backward

If you initialize byte-pair encoding with a vocabulary of size of 9 and then run the algorithm for 8 iterations, how large is the resulting vocabulary?

Correct. The vocab starts at 9 and adds 1 for each iteration.

The size of a vocabulary grows without bound as a function of corpus size.

Lemmatization and stemming are both methods for reducing surface word forms to underlying forms for some computational or linguistic purpose.

P(some | and) = 1 when we see and first next is some

<s> some are low </s> = 0.5 \* 1 \* 0.25 \* 1 = 0.125

Assume a 2-class document classification setting (e.g. spam detection) where you are provided with a training set (positive and negative examples) and you are using naive Bayes.

True or False:  In this application, the size of the vocabulary |V|  is based on the number of word types in the corpus as a whole  (both positive and negative examples).

Correct Answer

True

Assume you have a 3-way authorship classification problem that you are addressing with logistic regression.  Let's call the classes John, James, and Alex. During training you encounter a training document from class James with the feature vector [-3, 1, 4, 1].   With the current set of weights, the model returns the  softmax vector [0.2, 0.4, 0.4] over the classes John, James, and Alex respectively.

Give the gradient (vector of partial derivatives of the loss) for the James class that would be generated during training for this example using cross-entropy loss. Express your answer as a vector of values as in [  ,  ,  ,  ]

 1.8

 -0.6

 -2.4

 -0.6

Correct answer - predict/computed answer \* input X

1 - 0.4 = 0.6 , -0.6 \* vector =

Which of the following best describes why English part-of-speech tagging is hard?

Because the most frequently occurring words in English have multiple parts of speech.

A Matrix

|  |  |  |
| --- | --- | --- |
| From/To | Q1 | Q2 |
| Q1 | 0.6 | 0.4 |
| Q2 | 0.5 | 0.5 |

B Matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | a | b | c |
| Q1 | 0.2 | 0.4 | 0.4 |
| Q2 | 0.5 | 0.4 | 0.1 |

Pi

|  |  |
| --- | --- |
| Q1 | 0.8 |
| Q2 | 0.2 |

Consider the preceding HMM setup, and an observation sequence of length 2  consisting of "c c".

Complete the final column of the Viterbi table given below

|  |  |  |
| --- | --- | --- |
| Time | 1 | 2 |
| Q1 | 0.32 = 0.8 \* 0.4 | Max(q1-q1, q2-q1)0.0768 = p(q1) \*p(q1c) \* p(q1-q1) \* p(q1c) = 0.8\*0.4\*0.6\*0.4 = 0.0768 |
| Q2 | 0.02 = 0.2 \* 0.1 | 0.0128=max(q1-q2, q2-q2) = 0.32 \* p(q1-q2) \* p(q2c) = 0.32 \* 0.4 \* 0.1 = 0.0128 |
| Observation | c | c |

Lecture 10

Named Entity Recognition

Task, encoding, conditional random fields

Information extraction

Extraction of useful information from text

Text analytics

Shallow semantics

Ordinary newswire text is often used in typical examples

Find and classify all the named entities in a text

Colorado Rockies

Find means identify the exact span of mention

Classify means determine the category of the entity

Span/Type Ambiguity:

Louis Armstrong New International Airport

NER as sequence labeling

Per-word tagging task and use any sequence labeling tenique

BIO tagging inside outside Begin;

This word is inside a span; outside a span of interest; begins a span

1. Tokenizing
2. Sorting
3. Counting: mering upper and lower case; merging instances and getting counts; sorting the counts

Word prediction: compute a probability distribution over that vocabulary given the preceding words

It called probabilistic language model

Basic idea: choose the option with the highest probability as assigned by a language model

We are concerned with the probability of the outcome of discrete events

The sentence is the event

The sample space is the space of all possible sentences

We'd like to assign a probability to that event

P(the| its water is so transparent that) = Count(its water is so transparent that the) / count(its water is so transparent that)

According to Google, 1320/1420 = 0.9296

We are likely to get a lot of 0 counts

We first use the chain rule for the probability and then apply useful independence assumption

P(A^B) = P(A|B)P(B)

**The Chain rule:**

P(its water was so transparent)=

P(its)\*

P(water|its)\*

P(was|its water)\*

P(so|its water was)\*

P(transparent|its water was so)

Independence assumption: independent of its earlier history

Lizard |The other day I was walking along and saw a = lizard | a = lizard | saw a

Markov Assumption: replace each component in the product with an approximation; it depends only upon the present state

Maximum likelihood estimate MLE: P(wi | wi-1) = count(wi-1,wi)/count(wi-1)

Bigram



Table

Description automatically generated

P(want | I) = 827/2533 = 0.3265

N-gram

Autoregressive generation: sample a random bigram then sample a new random bigram but the prefix w match the suffix of the first bigram chosen

The generated texts from the higher-order models surely sound better

Extrinsic evaluation: A/B testing

Intrinsic evaluation:

Training set vs test set

Perplexity: a measure is the notion of surprise. The probability of a test set, as normalized by the number of words. Minimize perplexity is the same as maximizing probability of a test set

When we look at the test data, we run into words that we have not seen before. So non-subword solution: created a fixed lexicon, changed into UNK, collect counts, at test time use UNK counts for any word not seen in training

Multiplying a bunch of really small probabilities is a bad idea which cause underflow so do everything in log space, adding is faster than multiplying



Scoring: we block spam if the score gearter then threshold

Formulate the score as a probability P(class|document)

Softmax classification: it gives us the probability distribution of x belonging to each class

Federalist example: P(Jay |doc) = 3/8 = 0.375 ; P(Madison|doc) = 0.125; P(**hamilton** |doc) = 0.5;

Softmax = Score / all score = exp(wx+b)/sum exp(wx+b)

Shape

Description automatically generated with medium confidence

Shape

Description automatically generated with medium confidence

Shape

Description automatically generated with medium confidence

The kind of features used in NLP-oriented ML-based classifier systems are

Easily extracted from a text

Hand-crafted based on domain knowledge and data analysis

And we have lots and lots of them

Text

Description automatically generated

Text, letter

Description automatically generated

To learn the weights, we need some measure of how well (or badly) we’re doing with a current set of weights.

We’ll call that measure a Loss Function

The lower the loss the better we’re doing

We want to minimize the loss

What we want is a function that tells us how well our model is doing.

And does it in a way that can be used to guide the training process

Turns out that raw probabilities are the opposite of what we want for a loss.

We want bad performance to have high loss and good performance to have low loss.

So, we’ll take the negative of the log of the probability assigned to the correct answer as the loss

We’ll learning by starting with a random set of weights and then iteratively updating those weights to lower this cost.

The derivative of the loss function with respect to the weights tells us the direction and magnitude of change we should make to each weight.

Of course, in real applications we have many features/weights not just one.

So, we need the vector of the partial derivatives of the loss wrt the weights

Call that the gradient

In practice, that can be slow to converge because the algorithm can either be taking steps

That are too small and hence take us too long to get where we’re going

Or too large which leads us to overshoot the target and wander around too much

Fortunately, you don’t have to worry about this. Lots of packages available where you need to specify L and L’ and you’re done.

NER named entity recognition

CRF conditional random fields

Find and classify all the named entities in a text

Find means identify the exact span of the mention

Classify means determine the category of the entity

People (Type) - PER (Tag)

Type ambiguity: Washington can be person, place, organization

Instead of tag each element of a sequence we use span

BIO encoding: inside a span? Outside a span? Begins a span?

NER is works well from features

Entities such as organziation, person, time, date, number, money ….

NER avaiable from Standford Core NLP, Spacy

HMM Hidden Markov model

states = ('Rainy', 'Sunny')  
   
observations = ('walk', 'shop', 'clean')  
   
start\_probability = {'Rainy': 0.6, 'Sunny': 0.4}  
   
transition\_probability = {  
 'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},  
 'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},  
 }  
   
emission\_probability = {  
 'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},  
 'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},  
 }

Viterbi algorithm

A picture containing text, receipt

Description automatically generated