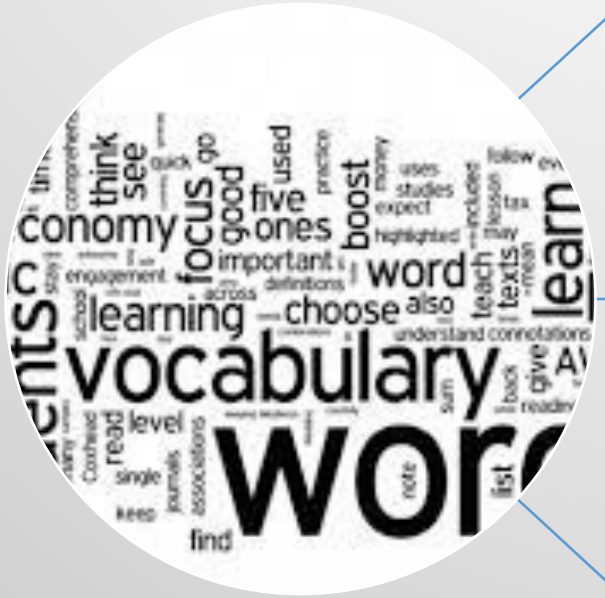
# Natural Language Processing

Dr. Karen Mazidi

# Part Two: Words




Topics

- POS tagging
- Types/history of POS taggers
- TextBlob
- spaCy

Quizzes

- Q Chp 6: POS Tagging

Homework

- Work on Guessing Game

# Approaches to POS tagging

1. Rule-based tagging
2. HMM and the Viterbi algorithm
3. Statistical approaches
4. ML approaches

# POS part-of-speech tagging

Approach 1: Rule-based systems

# 1. Rules-based POS tagging

- Begin with list of possible tags for a word, based on a lexicon, ex: sense [NN, VB]
- Assign words using morphology, ex: word ends in ‘est’ is probably a JJS superlative adjective
- These systems were brittle, even with thousands of rules

# POS part-of-speech tagging

Approach 2: HMMs and the Viterbi algorithm

# 2. HMM and the Viterbi algorithm

- Use a hidden Markov model to learn transition probabilities; ex: ‘the’ is usually followed by an adjective or a noun
- A sliding bigram window (plus start and end tags)

```
<s> John is a handsome man. </s>
<s> NNP  VBZ DT JJ NN . </s>
```

- The Viterbi algorithm uses dynamic programming to find the most probable transitions from one token to another (see Viterbi.pdf)
- First, a training corpus of tagged language is used to find counts/probabilities of tokens, bigrams, possibly trigrams (sequences of 1, 2, 3 successive words)

# The tagging problem

- Model pairs of sequences:
  - Let x represent tokens in the sequence
  - Let y represent POS tags (plus two start tags, one stop tag)

- This is challenging because
  - ambiguity
  - some words may not be in training data

# Trigram HMM

- V is the vocabulary (set of words) in the training corpus
- K is the set of tags, plus STOP and * for start
- Parameter q(s|u, v) is the probability of seeing tag s after seeing tags [u, v]
- Parameter e(x|s) is the probability of seeing x paired with s
- The joint probability of our tokens and tags is:

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i)$$

- Where $y^0$, $y^1$ = * and $y^{n+1}$ = STOP
- n is the number of tokens in the sentence and i is the number of tags

# Example

- The dog laughs (the x sequence)
- * * DT NN VBZ STOP (the y sequence)
- n = 3

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = q(\mathtt{D}|*,*) \times q(\mathtt{N}|*,\mathtt{D}) \times q(\mathtt{V}|\mathtt{D},\mathtt{N}) \times q(\mathtt{STOP}|\mathtt{N},\mathtt{V}) \times e(\textit{the}|\mathtt{D}) \times e(\textit{dog}|\mathtt{N}) \times e(\textit{laughs}|\mathtt{V})$$

- The q sequences is the prior learned from the training data
- The e sequence is the conditional probability that the token sequence has this tag sequence

# Independence assumption

- To make computation easier, we make an independence assumption that the tag of the current word depends on the tags of the previous two words, not the entire sequence

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i)$$

# Finding parameters

- MLE maximum likelihood estimates are obtained by counting occurrences in the training data

Given these definitions, the *maximum-likelihood* estimates are

$$q(s|u,v) = \frac{c(u,v,s)}{c(u,v)}$$

and

$$e(x|s) = \frac{c(s \rightsquigarrow x)}{c(s)}$$

For example, we would have the estimates

$$q(\mathtt{N}|\mathtt{V},\mathtt{D}) = \frac{c(\mathtt{V},\mathtt{D},\mathtt{N})}{c(\mathtt{V},\mathtt{D})}$$

and

$$e(dog|\mathtt{N}) = \frac{c(\mathtt{N} \rightsquigarrow dog)}{c(\mathtt{N})}$$

- Smoothing will be added to avoid 0 counts

# Search space

- Find the most likely tag sequence $\arg \max_{y_1 \cdots y_{n+1}} p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$
- For p(x, y) $p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i)$
- A brute force search would be intractable. A sentence with 10 tokens, using the 36 Penn tags would give a search space of $36^{10}$
- dynamic programming is used to solve the problem

# Viterbi algorithm

- Find most likely sequence of hidden states in a recursive, memoized algorithm
- We do this in chunks, filling in a table with “so far” values
- Chunk r considers only the first k terms:

$$r(y_{-1}, y_0, y_1, \ldots, y_k) = \prod_{i=1}^{k} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{k} e(x_i | y_i)$$

- pi is the max probability for sequence of length k ending in bigram [u, v]

$$\pi(k, u, v) = \max_{\langle y_{-1}, y_0, y_1, \ldots, y_k \rangle \in S(k,u,v)} r(y_{-1}, y_0, y_1, \ldots, y_k)$$

# Starting and recursing

- pi(k, u, v) base case

$$\pi(0, *, *) = 1$$

- The recursive part:

$$\pi(k, u, v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

# The Viterbi algorithm

**Input:** a sentence $x_1 \dots x_n$, parameters $q(s|u,v)$ and $e(x|s)$.
**Definitions:** Define $\mathcal{K}$ to be the set of possible tags. Define $\mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$, and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \dots n$.
**Initialization:** Set $\pi(0, *, *) = 1$.
**Algorithm:**

- For $k = 1 \dots n$,
  - For $u \in \mathcal{K}_{k-1}, v \in \mathcal{K}_k$,

$$\pi(k, u, v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

- **Return** $\max_{u \in \mathcal{K}_{n-1}, v \in \mathcal{K}_n} (\pi(n, u, v) \times q(\text{STOP}|u, v))$

# Viterbi with back pointers

- back pointers w are stored at each step
- at the end, unravel the back pointers to find the highest prob sequence
- Runs in time $O(n|K|^3)$

**Input:** a sentence $x_1 \dots x_n$, parameters $q(s|u,v)$ and $e(x|s)$.
**Definitions:** Define $\mathcal{K}$ to be the set of possible tags. Define $\mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$, and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \dots n$.
**Initialization:** Set $\pi(0, *, *) = 1$.
**Algorithm:**

- For $k = 1 \dots n$,
  - For $u \in \mathcal{K}_{k-1}, v \in \mathcal{K}_k$,

$$\pi(k,u,v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

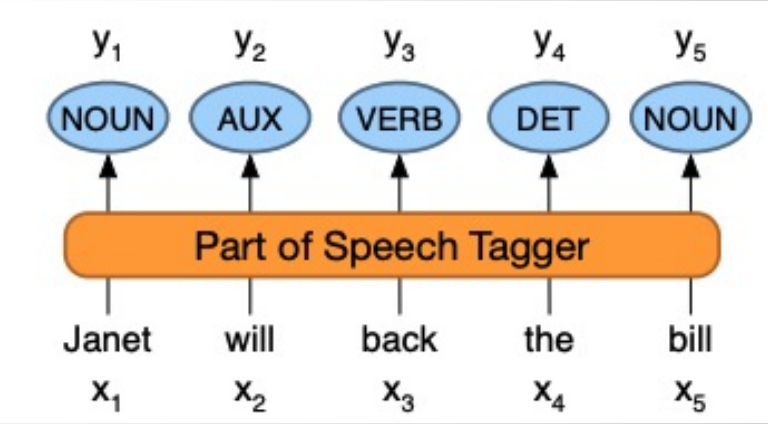
$$bp(k,u,v) = \arg\max_{w \in \mathcal{K}_{k-2}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

- Set $(y_{n-1}, y_n) = \arg\max_{u \in \mathcal{K}_{n-1}, v \in \mathcal{K}_n} (\pi(n,u,v) \times q(\text{STOP}|u,v))$
- For $k = (n-2) \dots 1$,

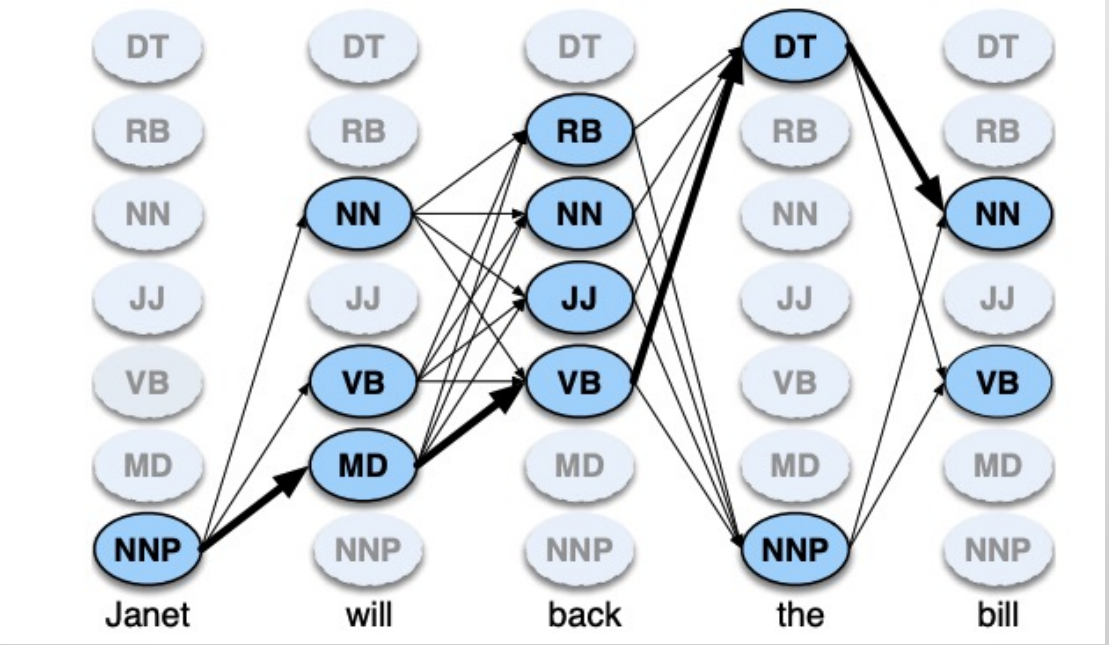
$$y_k = bp(k+2, y_{k+1}, y_{k+2})$$

- **Return** the tag sequence $y_1 \dots y_n$

# Example




- “lattice” data table
- Possible tags q in blue (each has probability p)
- Correct tags in bold path



# POS part-of-speech tagging

Approach 3: Statistical approaches

# Statistical approaches

- 1992 Eric Brill “Brill tagger”
- Takes up a fraction of memory size of n-gram HMM taggers
- Accuracy around 95%
- Rules are linguistically interpretable

1. Initially assigns each tag to its most likely tag according to a tagged corpus
2. Tags words according to last 3 letters (92% accuracy)
3. Patches:

1. If a word is tagged as **a** AND it is in context **C**, then change tag to **b**
2. If a word is tagged as **a** AND it has lexical property **P**, then change tag to **b**
3. IF a word is tagged as **a** AND a nearby word has lexical property **P**, then change the tag to **b**

# Brill tagger: example rules

- (1) Replace NN with VB when the previous word is TO
- (2) Replace TO with IN when the next tag is NNS
- Sample phrase:

```
to increase grants to states for …
```

- Assign most likely tags:

```
TO NN       NNS    TO NNS    IN …
```

- Apply Rule 1:

```
TO VB       NNS    TO NNS    IN …
```

- Apply Rule 2:

```
TO VB       NNS    IN NNS    IN ...
```

# NLTK Brill Demo

```
# Natural Language Toolkit: code_brill_demo

 >>> from nltk.tbl import demo as brill_demo
 >>> brill_demo.demo()
 Training Brill tagger on 80 sentences...
 Finding initial useful rules...
     Found 6555 useful rules.

            B      |
    S   F   r   O  |        Score = Fixed - Broken
    c   i   o   t  |  R     Fixed = num tags changed incorrect -> correct
    o   x   k   h  |  u     Broken = num tags changed correct -> incorrect
    r   e   e   e  |  l     Other = num tags changed incorrect -> incorrect
    e   d   n   r  |  e
------------------+-------------------------------------------------------
   12  13   1   4  | NN -> VB if the tag of the preceding word is 'TO'
    8   9   1  23  | NN -> VBD if the tag of the following word is 'DT'
    8   8   0   9  | NN -> VBD if the tag of the preceding word is 'NNS'
    6   9   3  16  | NN -> NNP if the tag of words i-2...i-1 is '-NONE-'
    5   8   3   6  | NN -> NNP if the tag of the following word is 'NNP'
    5   6   1   0  | NN -> NNP if the text of words i-2...i-1 is 'like'
    5   5   0   3  | NN -> VBN if the text of the following word is '*-1'
```

# POS part-of-speech tagging

Approach 4: Machine learning approaches

# Machine learning approaches

- Matthew Honnibal (create of spaCy) has a blog post where he described a simplified pos tagger using an averaged perceptron model and a greedy approach
- https://explosion.ai/blog/part-of-speech-pos-tagger-in-python
- Honnibal's tagger, written in about 200 lines of Python, achieved near 97% accuracy, which is competitive with state of the art taggers.

# Honnibal tagger

- Each token will have a set of features, including prefix, suffix, surrounding tags, and surrounding words.
- During training iterations, predictions are made for the tag, and weights are incremented if the predictions are correct, and decremented if they are not.
- At each iteration, average weights are used instead of final weights, so that later examples in the training don’t skew the weights.

# NLTK tagger

- pos_tag(list of tokens)
- Tagger works best on an input list of sentences
- See Chp 5 of nltk book for description of these taggers:
  - N-gram tagging
  - Brill tagger
- The perceptron tagger seems to be the default
- https://www.nltk.org/_modules/nltk/tag.html
- Code here: https://www.nltk.org/_modules/nltk/tag/perceptron.html

# Software for POS tagging

NLTK

TextBlob

SpaCy

# Tag Set

See end of Chapter 6

| Tag | Meaning | Example |
|---|---|---|
| CC | coordinating conjunction | but |
| CD | cardinal number | two |
| DT | determiner | the |
| EX | existential | there |
| FW | foreign word | ciao |
| IN | preposition | on |
| JJ | adjective | big |
| JJR | comparative adjective | bigger |
| JJS | superlative adjective | biggest |
| LS | list marker | A. |
| MD | modal | may |
| NN | noun | car |
| NNS | plural noun | cars |
| NNP | proper noun | Mary |
| NNPS | plural proper noun | Marys |
| PDT | predeterminer | *both* Marys |
| POS | possessive | Mary*'s* |
| PRP | personal pronoun | she |
| PRP$ | possessive pronoun | hers |
| RB | adverb | badly |
| RBR | comparative adverb | worse |
| RBS | superlative adverb | worst |
| RP | particle | give *up* |
| SYM | symbol | $ |
| TO | infinitive to | *to* be |
| UH | interjection | ugh |
| VB | lexical verb | run |
| VBD | past tense verb | ran |
| VBG | gerund or present participle | running |
| VBN | past particple | ran |
| VBP | singular present, not 3rd person | run |
| VBZ | singular present, 3rd person | runs |
| WDT | wh-determiner | which |
| WP | wh-pronoun | who |
| WP$ | possessive wh-pronoun | whose |
| WRB | wh-adverb | when |

Table 6.1: Part of Speech Tags

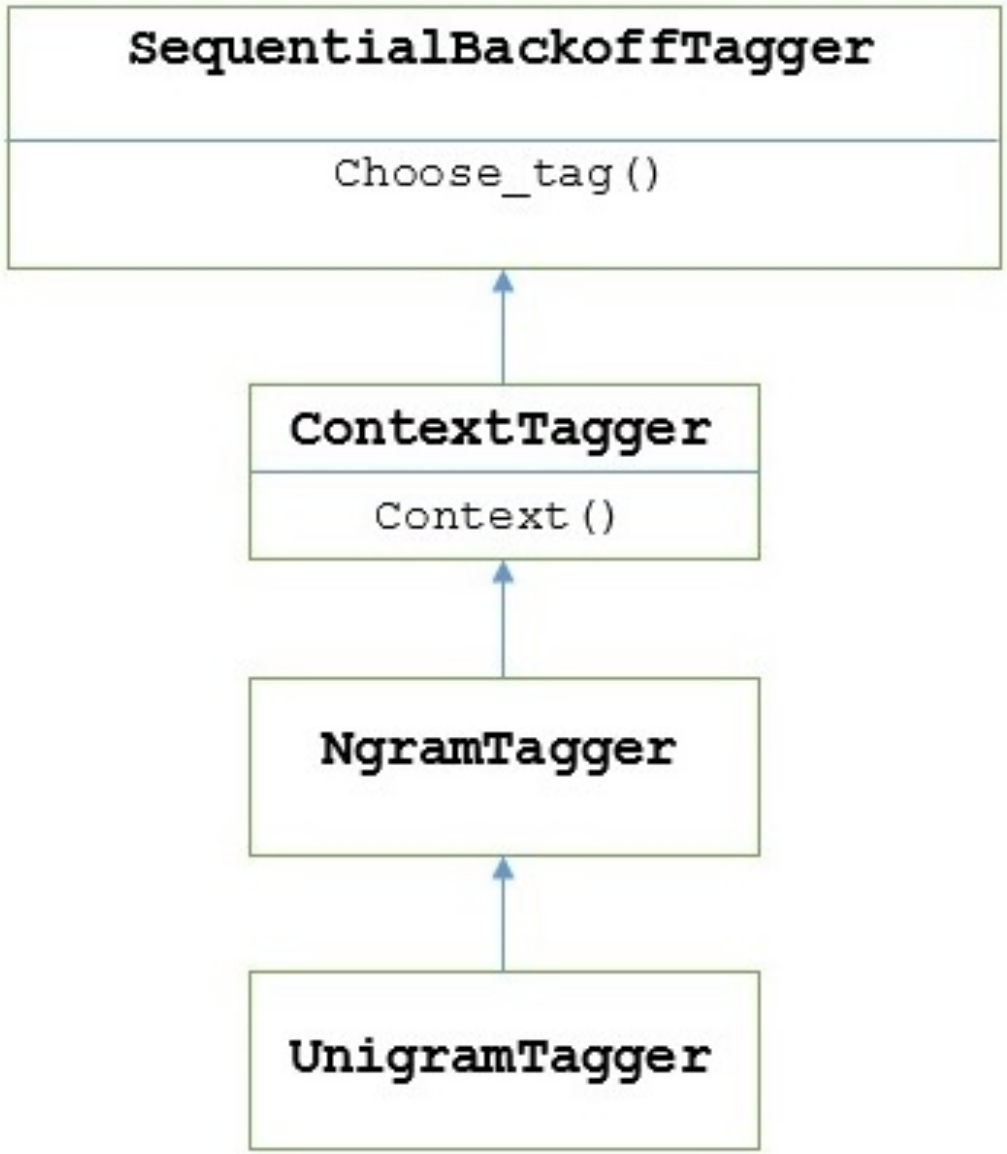
# POS Tagging NLTK

**Code 6.1.1 — NLTK.** POS Tags.

```
import nltk
from nltk import word_tokenize
text = "This is a sentence. This is another one."
tokens = word_tokenize(text)
tags = nltk.pos_tag(tokens)
print(tags)
[('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('sentence', 'NN'),
('.', '.'), ('This', 'DT'), ('is', 'VBZ'), ('another', 'DT'),
 ('one', 'NN'), ('.', '.')]
```

# NLTK Sequential POS Taggers

More taggers: https://www.nltk.org/api/nltk.tag.html



# TextBlob

- NLP API compatible with NLTK and in some parts built on top of it
- If you would like to check it out:
  - Install with pip/pip3, then download corpora

```
$pip3 install textblob
$python3 -m textblob.download_corpora
```

# The Blob

- Convert raw text to a TextBlob object

**Code 6.3.1 — Create a TextBlob object.** From a raw string

```
from textblob import TextBlob
# convert raw text to a TextBlob object
blob = TextBlob(text)
```

- See the Book or GitHub for examples

# spaCy

- https://spacy.io/
- Now version 3.0
- Install with pip/pip3 on mac, windows or linux
  - Hardware: CPU or GPU
  - Language options extensive, very international
  - See notebook in GitHub

# Python Code Examples
# In-class coding

- GitHub:
  - POS_NLTK
  - SpaCy
  - TextBlob


# Essential points to note

- POS tagging has taken different approaches over the years
- State of the art is near 98%

https://aclweb.org/aclwiki/POS_Tagging_(State_of_the_art)

# To Do

- Quiz Chapter 5 POS tagging
- Work on Guessing Game

# Next topic

WordNet