Part-of-speech tagging

- Each word must be assigned its correct part-of-speech, such as *noun*, *verb*, *adjective*, or *adverb*.
  He will *race* the car.
  When will the *race* end?
  The boat *floated* down the river sank.

- The number of tags used by different systems varies a lot. Some systems use < 20 tags, while others use > 400!
- Stochastic (bayesian) taggers: P $(Tag_i|Tag_{i-1}Tag_{i-2})$
- Achieve good results: 95-97% accuracy. But require enormous tables of statistics, do not represent intuitive rules, and adding improvements to the tagger is difficult.

Slide CS478–1

Transformation-based error-driven learning

- Simple heuristics can go a long way. You can get ~90% accuracy just by choosing the most frequent tag for a word!
- But defining the rules for special cases can be time-consuming, difficult, and prone to errors and omissions.
- *Transformation-based error-driven learning* is a technique for acquiring simple default heuristics and rules for special cases automatically. Rules are learned by iteratively collecting errors and generating rules to correct them.
- Requires a large (training) corpus of manually tagged text.

Slide CS478–2

Transformation-based error-driven learning

Slide CS478–3

Learning Algorithm: Greedy Search

- specify an initial tagger
- specify the allowable transformations
- provide an objective function for comparing corpus to the gold standard

ITERATE
  TRY EACH POSSIBLE TRANSFORMATION
  CHOOSE THE ONE WITH THE BEST SCORE
  ADD TO LIST OF TRANSFORMATIONS
  UPDATE THE TRAINING CORPUS
Until no transformation improves performance

Slide CS478–4

Possible Initial Taggers

- assign random tags
- label all as NOUN's
- ***assign most likely tag***
- use output of n-gram tagger

Slide CS478–5

Transformations: Patch Templates

- The tagger learns **transformations** to correct its mistakes.
- Before learning begins, a set of **transformation templates** are defined for the problem. Each template represents one type of rule that can be learned.
- Each rule (patch) learned by the system is one instantiation of a template. Most templates have many possible instantiations, so one template can spawn multiple rules!
- To generate a good set of templates, you must have a good understanding of what information is required to solve your problem!

Slide CS478–6

Templates for POS Tagging

Change tag **a** to tag **b** when:

1. the preceding/following word is tagged z
2. the word two before/after is tagged z
3. one of the two preceding/following words is tagged z
4. one of the three preceding/following words is tagged z
5. the preceding word is tagged z and the following word is tagged w
6. the preceding/following word is tagged z and the word two before/after is tagged w

Slide CS478–7

Generating Transformations

- Apply the initial tagger and compile types of tagging errors. Each type of error is of the form: $< tag_a, tag_b, number >$
  tag_a/tag_b : the incorrect/desired tag
  $number$: is the # of times this mistagging occurred
- For each error type, instantiate all templates to generate candidate transformations.
- Apply each candidate transformation to the corpus and count the number of corrections and errors that it produces. Save the transformation that yields the greatest improvement.
- Stop when no transformation can reduce the error rate by a predetermined threshold.

Slide CS478–8

An Example

Suppose that the initial tagger mistags 159 words as verbs when they should have been nouns.This would produce the error triple:

$$< verb, noun, 159 >$$

Suppose template #3 is instantiated as the rule:

> *change the tag from verb to noun if one of the two preceding words is tagged as a determiner.*

When this patch is applied to the corpus, it corrects 98 of the 159 new errors. But it also creates 18 new errors! The error reduction for this patch would be computed as 98-18=80.

Slide CS478–9

The learned transformations

	From	To	Condition	Value
1.	**nn**	**vb**	**prev-tag**	**to**
	Ex: I wanted to win/nn a car.			
2.	**vbp**	**vb**	**prev-1-or-2-or3-tag**	**md**
	Ex: The food might vanish/vbp from sight.			
3.	**nn**	**vb**	**prev1-or-2-tag**	**md**
	Ex: I would not run/nn for office.			

nn=singular noun, *vb*=verb base form, *to*=infinitive to, *vbp*=verb non-3rd pers sing pres, *vbz*=verb 3rd pers sing pres, *md*=modal, *dt*=determiner, *pos*=possessive, *vbd*= verb past tense, *vbn*= verb, past participle

Slide CS478–10

The learned transformations

4.	**vb**	**nn**	**prev-1-or-2-tag**	**dt**
	Ex: I went to the store/vb.			
5.	**vbd**	**vbn**	**prev-1-or-2-or-3-tag**	**vbz**
	Ex: He has gone/vbd to the store.			
10.	**pos**	**vbz**	**prev-1-tag**	**prp**
	Ex: He 's going to the store.			

nn=singular noun, *vb*=verb base form, *to*=infinitive to, *vbp*=verb non-3rd pers sing pres, *vbz*=verb 3rd pers sing pres, *md*=modal, *dt*=determiner, *pos*=possessive, *vbd*= verb past tense, *vbn*= verb, past participle

Slide CS478–11

Tagging new text

The resulting tagger consists of two phases:

1. Use the initial tagger to tag all the text.
2. Apply each transformation, in order, to the corpus to correct some of the errors.

The order of the transformations is very important!

For example, it is possible for a word's tag to change several times as different transformations are applied. In fact, a word's tag could thrash back and forth between the same two tags.

Slide CS478–12

Evaluation

- The tagger was trained on 600,000 words from the Penn Treebank WSJ Corpus.
- Tested on a separate 150,000 word test set.
- Assumes all possible tags for all test set words are known.
- 97.0% accuracy
- The tagger learned 378 rules.
- A stochastic tagger trained on 1 million words from the same corpus achieved 96.7% accuracy. The stochastic tagger had to learn and store 10,000 contextual probabilities.

Slide CS478–13

Lexicalizing the tagger

The original set of transformations were entirely tag-based.

No specific words were used in the rules.

But certain phrases and lexicalized expressions can yield idiosyncratic tag sequences, so allowing the rules to look for specific words should help.

5 new lexical patch templates were added:

Change tag **a** to tag **b** when:

1. the preceding/following word is w
2. the word two before/after is w
3. one of the two preceding/following words is w
4. the current word is w and the preceding/following word is x
5. the current word is w and the preceding/following word is tagged z

Slide CS478–14

Examples of Lexicalized Transformations

change tag from preposition to adverb if the word two positions to the right is "as".

This rule handles expressions such as "His son is as tall as Karl Malone."

"as" is most often tagged as a preposition, so the initial tagger would produce the sequence:

as/preposition tall/adjective as/preposition

But the Penn Treebank style guide says that the first occurrence of *as* in this expression should be an adverb.

Slide CS478–15

Results for Lexicalized Transformations

- The tagger was trained on 600,000 words from the Penn Treebank Tagged Corpus, and tested on a separate 150,000 word test set.
- The tagger learned 447 rules and achieved 97.2% accuracy.
- The first 200 rules achieved 97.0%.
- The first 100 rules achieved 96.8%.

Slide CS478–16

Unknown Word Tagger

Initial tagger: proper noun if capitalized, common noun otherwise.

Change the tag of an unknown word from **a** to **b** if:

1. deleting the prefix x results in a word ($| x | \leq 4$)
2. the first (1,2,3,4) characters of the word are x
3. the last (1,2,3,4) characters of the word are x
4. adding the character string x as a prefix results in a word ($| x | \leq 4$)
5. adding the character string x as a suffix results in a word ($| x | \leq 4$)
6. word w ever appears immediatley to the left/right of the word
7. character z appears in the word

Slide CS478–17

Example transformations to tag unknown words

Change the tag:

- from *noun* to *plural noun* if the word has suffix **-s**
- from *noun* to *number* if the word has character **.**
- from *noun* to *adjective* if the word has character **-**
- from *noun* to *past participle* if the word has suffix **-ed**
- from *noun* to *gerund* or *present participle* if the word has suffix **-ing**
- to *adjective* if adding the suffix **-ly** results in a word
- to *adverb* if the word has suffix **-ly**
- from *noun* to *number* if the word **$** appears immediately to the left
- from *noun* to *adjective* if the word has suffix **-al**
- from *noun* to *base verb* if the word **would** appears immediately to the left

Slide CS478–18

Results for tagging unknown words

- The templates for unknown words were evaluated using 1.1 million words of the Penn Treebank Corpus: 950,000 for training and 150,000 for testing.
- 600,000 words were used to learn 267 contextual rules. 350,000 words were used to learn 148 unknown word rules.
- Unknown word accuracy was 85% on the test corpus. Overall accuracy was 96.5%.
- A stochastic tagger achieved 85% accuracy for unknown words on the same corpus, but uses over 1,000 parameters and 10^8 probabilities for this task. The TBL tagger used only 148 easily understandable rules.

Producing multiple tags

- Sometimes it is useful to generate several possible tags and let another module decide which one is applicable in the current context. Brill calls this *k-best tagging*.
- The transformation-based learner can be modified to produce multiple tags by adding each new tag suggested by a patch instead of changing the original tag.
- The result is a set of possible tags for each word, where each tag was produced by either the initial tagger or a learned patch.
- The goal is to improve the potential accuracy (the correct tag is among the proposed tags) without letting the average number of tags/word explode.

Results for k-best tagging

- The original one-tag-per-word tagger was applied first, then the k-best tagger was applied. A set of k-best transformations was learned using a separate 240,000 word corpus.

- As a baseline, k-best tagging was done by:
  - for known words, collecting all tags seen with that word in the training corpus
  - for unknown words, collecting the five most likely tags for all unknown words

- The baseline tagger achieved 99% accuracy, with 2.28 tags/word on average. The rule-based tagger achieved 99% accuracy, with 1.43 tags/word on average.

Slide CS478–21

Results for k-best tagging

Accuracy was tested after every 50 learned rules were acquired:

#rules	*accuracy*	*avg # tags/word*
0	96.5	1.00
50	96.9	1.02
100	97.4	1.04
150	97.9	1.10
200	98.4	1.19
250	99.1	1.50

Slide CS478–22

Summary

Bad: relies heavily on a large, manually annotated training corpus, which may not be practical for some applications.

Bad: relies on reasonable default heuristics to get things started, and a set of templates that seem appropriate for the problem at hand.

Good: learns rules that are easily understandable.

Good: allows rules to be easily acquired for different domains or genres.

Good: the patch set is relatively small and does not require a lot of memory.

Transformation-based learning has also been applied to problems in PP attachment and syntactic parsing.