Part-of-Speech Tagging

- Task definition
  - Part-of-speech tags
  - Task specification
  - Why is POS tagging difficult
- Methods
  - Transformation-based learning approach [Brill 93]
  - Hidden Markov Models

Why is POS Tagging Hard?

- Ambiguity
  - He will race/VB the car.
  - When will the race/NOUN end?
  - The boat floated/VBD down the river.
  - The boat floated/VBN down the river sank.
- Average of ~2 parts of speech for each word
- The number of tags used by different systems varies a lot. Some systems use < 20 tags, while others use > 400.

Among Easiest of NLP Problems

- State of the art methods achieve ~97% accuracy.
- Simple heuristics can go a long way.
  - ~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

Part-of-Speech Tagging Task

- Assign the correct part of speech (word class) to each word in a document
  - “The DT planet NN Jupiter NNP and/CC its/PRP moons NNS are/VBP in/IN effect/NN a DT mini-solar JJ system/NN ./,” and/CC Jupiter/NNP itself/PRP is/VBZ often/RB called/VBN a DT star/NN that/IN never/RB caught/VBN fire/NN ./”
- Needed as an initial processing step for a number of language technology applications
  - Information extraction
  - Answer extraction in QA
  - Base step in identifying syntactic phrases for IR systems
  - Critical for word-sense disambiguation (WordNet apps)
- ...

Part-of-Speech Tagging

- Task definition
  - Part-of-speech tags
  - Task specification
  - Why is POS tagging difficult
- Methods
  - Transformation-based learning approach [Brill 93]
  - Hidden Markov Models
Transformation-Based Learning

- Machine learning technique
  - For acquiring simple default heuristics and rules for special cases
  - Rules are learned by iteratively collecting errors and generating rules to correct them.
- Requires a large (training) corpus of manually tagged text

TBL: Top-Level Algorithm

Unannotated Text

Initial State

Annotated Text

Learns an ordered list of transformations (i.e. rewrite rules)

Learning Algorithm: Greedy Search

- Specify
  - An initial state annotator
  - Space of allowable transformations
  - Objective function for comparing corpus to truth
- Algorithm
  - 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

Rewrite Rules

- Rule
  - Change modal to noun, if preceding word is a determiner.
- Example
  - Determiner: the, a, an, this, that …
  - Modals: can, will, would, may, might…followed by the main verb
  - The/det can/modal rusted/verb ...
  - The/det can/noun rusted/verb ...

Transformation Templates

- Change tag A to B when:
  - preceding/following word is tagged Z
  - word two before/after is tagged Z
  - one of the two preceding/following words is tagged Z
  - one of the three preceding/following words is tagged Z
  - preceding word is tagged Z and following word is tagged W
  - preceding/following word is tagged Z and word two before/after is tagged W
Generating Transformations

- Apply the initial tagger and compile types of tagging errors. Each type of error is of the form:
  - <incorrect tag, desired tag, # of occurrences>

- 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.

Example

- Suppose that the initial tagger mistags 159 words as verbs when they should have been nouns.

- Produces 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 template is applied to the corpus, it corrects 98 of the 159 errors. But it also creates 18 new errors. Error reduction is 98-18=80.

Learned Rules

1. NN → VB if the previous tag is TO
   I wanted/to win/NN → VB a Subaru WRX…
2. VBP → VB if one of the prev-3 tags is MD
   The food might/MĐ vanish/VBP → VB from sight.
3. NN → VB if one of prev-2 tags is MD
   I might/MD not reply/NN → VB
4. VB → NN if one of the prev-2 tags is DT
5. VBD → VBN if one of the prev-3 tags is VBZ
6. VBN → VBD if one of the previous tag is PRP

Tagging New Text

- The resulting tagger consists of two phases:
  - Use the initial tagger to tag all the text
  - 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.

Evaluation

- Training: 600,000 words from the Penn Treebank WSJ corpus
- Testing: separate 150,000 words from PTB
- Assumes all possible tags for all test set words are known.
- 97.0% accuracy
- Tagger learned 378 rules.

Problems?

- Not lexicalized
  - Transformations are 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...
  - Add additional templates
  - E.g. when the preceding/following word is w...
- Tagger achieves 97.2% accuracy
  - First 200 rules achieved 97.0%
  - First 100 rules achieved 96.8%
  - Learns 447 rules
- Unknown words
Transformation-Based Learning

- Part-of-speech tagging
  [Brill 1995; Ramshaw & Marcus 1994]
- Prepositional phrase attachment
  [Brill & Resnik 1995]
- Syntactic parsing
  [Brill 1994]
- Noun phrase chunking
  [Ramshaw & Marcus 1995, 1999]
- Context-sensitive spelling correction
  [Mangu & Brill 1997]
- Dialogue act tagging
  [Samuel et al. 1998]

Part-of-Speech Tagging

- Part-of-Speech Tagging
  - Part-of-speech tags
  - Task specification
  - Why is POS tagging difficult
- Methods
  - Transformation-based learning approach [Brill 93]
  - Hidden Markov Models
- Named Entity Recognition

Hidden Markov Models

- Application to POS tagging:
  - View POS tagging as a sequence of word classification tasks
  - Goal: Train an HMM to label every word with one of the POS tags.
- What is a HMM?
  - Hidden Markov Model (HMM) represents a process of generating the word and tag sequence
  - Probabilistic model
  - Probability for each word and tag sequence
  - Predict most likely tag sequence for a given word sequence

States and Transitions

- States
  - Think about as nodes of a graph
  - One for each POS tag
  - Special start state (and maybe end state)
- Transitions
  - Think about as directed edges in a graph
  - Edges have transition probabilities
- Output
  - Each state also produces a word of the sequence
  - Sentence is generated by a walk through the graph

Probabilistic Model

- Starting state s0
  - Specifies where the sequence starts
- Transition probability P(Sk|Sk-1)
  - Probability that one state succeeds another
  - Matrix of size #states * #states
- Emission probability P(Wt|St)
  - Probability that word is generated in this state
  - Matrix of size #states * #words

=> Every word + state sequence has a probability P(W,S)

\[
P(W_1,\ldots,W_n, S, s_1, s_2, \ldots, s_n) = \prod_{i=1}^n P(W_i | s_i)P(s_i | s_{i-1})
\]

HMM Inference Type I: Evaluation

- Question: What is the probability of an output sequence given an HMM
  - Given fully specified HMM: s0, P(W|S), P(S|S)
  - Find for a given w1,\ldots,wn

\[
P(W_1,\ldots,W_n) = \sum_{\{s_1,\ldots,s_n\}} \prod_{i=1}^n P(W_i | s_i)P(s_i | s_{i-1})
\]

- Naïve algorithm exponential runtime; “forward” algorithm linear in length of sequence
- Language model
- Example: classify sequences as question vs. answer sentence.
**HMM Inference Type II: Decoding**

- **Question:** What is the most likely state sequence given an output sequence
  - Given fully specified HMM: \( x_0, P(W_t|S_t), P(S_t|S_{t-1}) \)
  - Find

\[
\max P(x_0 \rightarrow x_n | x_t, w_1, ..., w_n) = \max_{x_0 \rightarrow x_n} \left\{ \prod_{i=1}^{n} P(w_i | x_i) P(x_i | x_{i-1}) \right\}
\]

- "Viterbi" algorithm has runtime linear in length of sequence
- Example: find the most likely tag sequence for a given sequence of words

**Experimental Results**

<table>
<thead>
<tr>
<th>Tagger</th>
<th>Accuracy</th>
<th>Training time</th>
<th>Prediction time</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>96.80%</td>
<td>20 sec</td>
<td>18,000 words/s</td>
</tr>
<tr>
<td>TBL</td>
<td>96.47%</td>
<td>9 days</td>
<td>750 words/s</td>
</tr>
</tbody>
</table>

- **Experiment setup**
  - WSJ Corpus
  - Trigram HMM model
  - Lexicalized
  - from [Pla and Molina, 2001]

**Estimating the Probabilities**

- **Given:** Fully observed data
  - Pairs of word sequence with their state sequence
- **Estimating transition probabilities** \( P(S_t|S_{t-1}) \)

\[
P(S_t | S_{t-1}) = \frac{\# of TimesStateAFollowsStateB}{\# of TimesStateBOccurs}
\]

- **Estimating mission probabilities** \( P(W_t|S_t) \)

\[
P(w_t | S_t) = \frac{\# of TimesWordIsObservedInStateB}{\# of TimesStateBOccurs}
\]

- **Smoothing the estimates**
  - Laplace smoothing -> uniform prior
  - See naïve Bayes for text classification
- **Partially observed data:** Expectation Maximization (EM)

**HMM’s for POS Tagging**

- **Design HMM structure (vanilla)**
  - States: one state per POS tag
  - Transitions: fully connected
  - Emissions: all words observed in training corpus
- **Estimate probabilities**
  - Use corpus, e.g. Treebank
  - Smoothing
  - Unseen words?
- **Tagging new sentences**
  - Use Viterbi to find most likely tag sequence