NLP Programming Tutorial 5 - Part of Speech Tagging with Hidden Markov Models

Graham Neubig
Nara Institute of Science and Technology (NAIST)
Part of Speech (POS) Tagging

- Given a sentence $X$, predict its part of speech sequence $Y$

```
Natural language processing (NLP) is a field of computer science
```

- A type of “structured” prediction, from two weeks ago
- How can we do this? Any ideas?
Many Answers!

- **Pointwise prediction**: predict each word individually with a classifier (e.g. perceptron, tool: KyTea)

- **Generative sequence models**: todays topic! (e.g. Hidden Markov Model, tool: ChaSen)

- **Discriminative sequence models**: predict whole sequence with a classifier (e.g. CRF, structured perceptron, tool: MeCab, Stanford Tagger)

Natural language processing (NLP) is a field of computer science.
Probabilistic Model for Tagging

- “Find the most probable tag sequence, given the sentence”

\[
\text{argmax } P \left( Y \mid X \right)
\]

- Any ideas?
Generative Sequence Model

- First decompose probability using Bayes' law

\[
\arg\max_Y P(Y|X) = \arg\max_Y \frac{P(X|Y)P(Y)}{P(X)} = \arg\max_Y P(X|Y)P(Y)
\]

Model of word/POS interactions
“natural” is probably a JJ

Model of POS/POS interactions
NN comes after DET

- Also sometimes called the “noisy-channel model”
Hidden Markov Models
Hidden Markov Models (HMMs) for POS Tagging

- POS → POS transition probabilities
  - Like a bigram model!
- POS → Word emission probabilities

\[
P(Y) \approx \prod_{i=1}^{l+1} P_T(y_i|y_{i-1})
\]

\[
P(X|Y) \approx \prod_{1}^{l} P_E(x_i|y_i)
\]
Learning Markov Models (with tags)

- Count the number of occurrences in the corpus and

```
 natural language processing ( nlp ) is ...
```

```
<s> JJ NN NN LRB NN RRB VB ... </s>
```

```
c(JJ → natural)++ c(NN → language)++ ...
```

```
c(<s> JJ)++ c(JJ NN)++ ...
```

- Divide by context to get probability

\[
P_T(LRB|NN) = \frac{c(NN LRB)}{c(NN)} = \frac{1}{3}
\]

\[
P_E(\text{language}|NN) = \frac{c(NN \rightarrow \text{language})}{c(NN)} = \frac{1}{3}
\]
Training Algorithm

# Input data format is “natural_JJ language_NN …”

make a map emit, transition, context

for each line in file

previous = “<s>”
context[previous]++

split line into wordtags with “ “

for each wordtag in wordtags

split wordtag into word, tag with “_”
transition[previous] += tag++ # Count the transition
context[tag]++ # Count the context
emit[tag] += “word”++ # Count the emission
previous = tag

transition[previous] += “</s>”++ # Print the transition probabilities

for each key, value in transition

split key into previous, word with “ “
print “T”, key, value/context[previous]

# Do the same thing for emission probabilities with “E”
Note: Smoothing

- In bigram model, we smoothed probabilities

\[
P_{\text{LM}}(w_i|w_{i-1}) = \lambda P_{\text{ML}}(w_i|w_{i-1}) + (1-\lambda) P_{\text{LM}}(w_i)
\]

- HMM transition prob.: there are not many tags, so smoothing is not necessary

\[
P_T(y_i|y_{i-1}) = P_{\text{ML}}(y_i|y_{i-1})
\]

- HMM emission prob.: smooth for unknown words

\[
P_{\text{E}}(x_i|y_i) = \lambda P_{\text{ML}}(x_i|y_i) + (1-\lambda) 1/N
\]
Finding POS Tags
Finding POS Tags with Markov Models

- Use the **Viterbi algorithm** again!!

  I told you I was important!!

- What does our graph look like?
Finding POS Tags with Markov Models

• What does our graph look like? Answer:

```
natural  language  processing  (  nlp  )
```

```
0:<S> → 1:NN → 2:NN → 3:NN → 4:NN → 5:NN → 6:NN
```

…  …  …  …  …  …
Finding POS Tags with Markov Models

- The best path is our POS sequence

```
0:<S>
1:NN
1:JJ
1:VB
1:LRB
1:RRB

1:NN
2:NN
2:JJ
2:VB
2:LRB
2:RRB

3:NN
3:J
3:VB
3:LRB
3:RRB

4:NN
4:JJ
4:VB
4:LRB
4:RRB

5:NN
5:JJ
5:VB
5:LRB
5:RRB

6:NN
6:JJ
6:VB
6:LRB
6:RRB

<s>   JJ   NN   NN   LRB   NN   RRB
```
Remember: Viterbi Algorithm Steps

- **Forward step**, calculate the best path to a node
  - Find the path to each node with the **lowest negative log probability**
- **Backward step**, reproduce the path
  - This is easy, almost the same as word segmentation
Forward Step: Part 1

- First, calculate transition from <S> and emission of the first word for every POS

```
natural

0:<S> → 1:NN
best_score[“1 NN”] = -log P_T(NN|<S>) + -log P_E(natural | NN)

1:JJ
best_score[“1 JJ”] = -log P_T(JJ|<S>) + -log P_E(natural | JJ)

1:VB
best_score[“1 VB”] = -log P_T(VB|<S>) + -log P_E(natural | VB)

1:LRB
best_score[“1 LRB”] = -log P_T(LRB|<S>) + -log P_E(natural | LRB)

1:RRB
best_score[“1 RRB”] = -log P_T(RRB|<S>) + -log P_E(natural | RRB)
...```
Forward Step: Middle Parts

- For middle words, calculate the minimum score for all possible previous POS tags

```
best_score["2 NN"] = min(
    best_score["1 NN"] + -log P_T(NN|NN) + -log P_E(language | NN),
    best_score["1 JJ"] + -log P_T(NN|JJ) + -log P_E(language | NN),
    best_score["1 VB"] + -log P_T(NN|VB) + -log P_E(language | NN),
    best_score["1 LRB"] + -log P_T(NN|LRB) + -log P_E(language | NN),
    best_score["1 RRB"] + -log P_T(NN|RRB) + -log P_E(language | NN),
    ...
)
best_score["2 JJ"] = min(
    best_score["1 NN"] + -log P_T(JJ|NN) + -log P_E(language | JJ),
    best_score["1 JJ"] + -log P_T(JJ|JJ) + -log P_E(language | JJ),
    best_score["1 VB"] + -log P_T(JJ|VB) + -log P_E(language | JJ),
    ...
)
```
Forward Step: Final Part

- Finish up the sentence with the sentence final symbol

```
science

I:NN
I:JJ
I:VB
I:LRB
I:RRB

best_score["I+1 </S>"] = min(
  best_score["I NN"] + -log PT(</S>|NN),
  best_score["I JJ"] + -log PT(</S>|JJ),
  best_score["I VB"] + -log PT(</S>|VB),
  best_score["I LRB"] + -log PT(</S>|LRB),
  best_score["I NN"] + -log PT(</S>|RRB),
  ...
)
```
Implementation: Model Loading

**make** a map for transition, emission, possible_tags

for each line in model_file
    split line into type, context, word, prob
    possible_tags[context] = 1  # We use this to
    # enumerate all tags

    *if* type = “T”
        transition[“context word”] = prob
    *else*
        emission[“context word”] = prob
Implementation: Forward Step

split line into words
I = length(words)
make maps best_score, best_edge
best_score[“0 <s>”] = 0  # Start with <s>
best_edge[“0 <s>”] = NULL
for i in 0 … I-1:
    for each prev in keys of possible_tags
        for each next in keys of possible_tags
            if best_score[“i prev”] and transition[“prev next”] exist
                score = best_score[“i prev”] +
                    -log P_T(next|prev) + -log P_E(word[i]|next)
            if best_score[“i+1 next”] is new or > score
                best_score[“i+1 next”] = score
                best_edge[“i+1 next”] = “i prev”
# Finally, do the same for </s>
Implementation: Backward Step

tag = [ ]
next_edge = best_edge[ "I <s>" ]
while next_edge != "0 <s>"
    # Add the substring for this edge to the words
    split next_edge into position, tag
    append tag to tags
    next_edge = best_edge[ next_edge ]
tag.reverse()
join tags into a string and print
Exercise
Exercise

- **Write** train-hmm and test-hmm
- **Test** the program
  - Input: test/05-{train,test}-input.txt
  - Answer: test/05-{train,test}-answer.txt
- **Train** an HMM model on data/wiki-en-train.norm_pos and **run** the program on data/wiki-en-test.norm
- **Measure** the accuracy of your tagging with script/gradeapos.pl data/wiki-en-test.pos my_answer.pos
- **Report** the accuracy
- **Challenge**: think of a way to improve accuracy
Thank You!