

Language Understanding Systems

Sequence Labeling with WFSM & LM

Evgeny A. Stepanov

SISL, DISI, UniTN & VUI, Inc.

`evgeny.stepanov@unitn.it`

Outline

- 1 Sequence Labeling
- 2 POS-Tagging
- 3 Exercise
- 4 Solution

Section 1

Sequence Labeling

Sequence Labeling: *Definition*

Classification

Assignment of a categorical **label** to a **new observation** based on the categories in the **training data**.

Sequence Labeling

Assignment of a categorical **label** to each member of a **sequence** of observed values

- Can be treated as a set of independent **classification** tasks, one per member of the sequence;
- Performance is generally improved by making the optimal label for a given element **dependent on the choices of nearby elements**;

Sequence Labeling: *NLP Tasks*

- Part-of-Speech Tagging
- Chunking
- Named Entity Recognition
- Semantic Role Labeling
- Frame Net Parsing
- Dependency Parsing
- Discourse Parsing
- **Spoken Language Understanding**
- ...

Sequence Labeling: *NLP Tasks*

- Part-of-Speech Tagging
- Chunking
- Named Entity Recognition
- Semantic Role Labeling
- Frame Net Parsing
- Dependency Parsing
- Discourse Parsing
- Spoken Language Understanding
- ...

Section 2

POS-Tagging

Part-of-Speech Tagging

POS-Tags: Noun, Verb, etc.

the/DT cat/NN is/VBZ fat/JJ

The general setting:

- Create ‘training’ and ‘testing’ corpora by POS-tagging a certain amount of text by hand
- ‘Train’ POS-tagging model to extract generalizations from the annotated corpus
- Use the trained POS-tagger too annotate new texts

Markov Model Tagging

Given word sequence w_1, \dots, w_n find the most probable tag sequence t_1, \dots, t_n

$$t_1, \dots, t_n = \arg \max_{t_1, \dots, t_n} P(t_1, \dots, t_n | w_1, \dots, w_n) \quad (1)$$

Bayes's Law:

$$t_1, \dots, t_n = \arg \max_{t_1, \dots, t_n} \frac{P(w_1, \dots, w_n | t_1, \dots, t_n) P(t_1, \dots, t_n)}{P(w_1, \dots, w_n)} \quad (2)$$

Probability of a word sequence is the same for all tags, thus:

$$t_1, \dots, t_n = \arg \max_{t_1, \dots, t_n} P(w_1, \dots, w_n | t_1, \dots, t_n) P(t_1, \dots, t_n) \quad (3)$$

Simplifying Assumptions

- Probability of a word only depends on its own tag, not tags of other words in sentence:

$$P(w_1, \dots, w_n | t_1, \dots, t_n) \approx P(w_1 | t_1) P(w_2 | t_2) \dots P(w_n | t_n) \quad (4)$$

- The (first-order) Markov assumption (bigram):

$$P(t_1, \dots, t_n) \approx P(t_1 | t_0) P(t_2 | t_1) \dots P(t_n | t_{n-1}) \quad (5)$$

- Second-order Markov assumption would correspond to trigram model

Simplifying Assumptions

$$t_1, \dots, t_n = \arg \max_{t_1, \dots, t_n} P(t_1, \dots, t_n | w_1, \dots, w_n) \quad (6)$$

$$t_1, \dots, t_n = \arg \max_{t_1, \dots, t_n} \prod_{i=1}^n P(w_i | t_i) P(t_i | t_{i-1}) \quad (7)$$

- $P(w_i | t_i)$ – probability of seeing current word given the current tag
- $P(t_i | t_{i-1})$ – probability of seeing the current tag given the tag we just saw

Training from Data

- $$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \quad (8)$$

- $$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \quad (9)$$

Section 3

Exercise

Unigram POS-Tagger

Develop simple **unigram**-based **POS-Tagger**

$$t_1, \dots, t_n = \arg \max_{t_1, \dots, t_n} \prod_{i=1}^n P(w_i | t_i) P(t_i) \quad (10)$$

$$P(w_1 | t_1) * P(w_2 | t_2) * \dots * P(w_n | t_n) \quad (11)$$

Data

train.pos.txt

- Create lexicon
- Create transducer & compile it using OpenFST
- Take care of *unknown words*: all tags are equally probable
- POS-tag the test set: test.txt

Ngram POS-Tagger

Extend the unigram POS-Tagger to

$$t_1, \dots, t_n = \arg \max_{t_1, \dots, t_n} \prod_{i=1}^n P(w_i | t_i) P(t_i | t_{i-1}) \quad (12)$$

Data

train.pos.txt

- Create lexicon
- Create *LM* using OpenGRM
- POS-tag the test set: test.txt

Section 4

Solution

Calculating $P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$

Create count file for $C(t_i)$, from 2 column tab-separated token-per-line format: 1st column - tokens, 2nd column POS-tags

```
cat $data | cut -f 2 |\ # get 2nd column
sed '/^ *$/d' |\       # remove empty lines
sort | uniq -c |\      # unique list of POS with counts
sed 's/^ */g' |\       # remove leading spaces
awk '{OFS="\t"; print $2,$1}' # swap columns
                             # & use tab for separator
```

The last step is optional, but convenient. Redirect output to some file, e.g. *POS.counts*

Calculating $P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$

Similarly, create count file for $C(t_i, w_i)$, from 2 column tab-separated token-per-line format: 1st column - tokens, 2nd column POS-tags

```
cat $data | \           # we need both columns
sed '/^ */d' | \        # remove empty lines
sort | uniq -c | \       # unique list of word-POS counts
sed 's/^ */g' | \       # remove leading spaces
awk '{OFS="\t"; print $2,$3,$1}' # swap columns
                                # & use tab for separator
```

The last step is optional, but convenient. Redirect output to some file, e.g. *TOK_POS.counts*

Calculating $P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$

Calculate probabilities $P(w_i|t_i)$ using both count files

```
while read token pos count
do
    # get pos counts
    poscount=$(grep "^$pos\t" POS.counts | cut -f 2)
    # calculate probability
    prob=$(echo "$count / $poscount" | bc -l)
    # print token, pos-tag & probability
    echo -e "$token\t$pos\t$prob"
done < TOK_POS.counts
```

Redirect output to some file, e.g. *TOK_POS.probs*

Note: Some POS-tags end with \$, which is ‘end-of-line’ symbol in RegEx. This might lead to errors. Using grep as above gets the desired behavior.

[**Advice:** Use proper scripting language]

Building Transducer

- Create lexicon file that contains both words and POS-tags.
- Since word probabilities depends only on tokens, single state transducer is enough.
- Either add 0 0 in-front of every line in TOK_POS.probs, or modify the code to do so.

...

```
# -e to interpret \t
# -n to not print new line
echo -en "0\t0\t"
# print token, pos-tag & probability
echo -e "$token\t$pos\t$prob"
```

...

- Then add the final state '0' as
echo '0' >> TOK_POS.probs
- Compile using `fstcompile`

Issues to Solve

- By default FST wights are *costs* – the lower the better; *probability* – the higher the better.
- Unknown words & their $P(w_i|t_i)$

Weights

Instead of raw probabilities use negative log probabilities

$x = -\ln(x)$:

```
prob=$(echo "-l($count / $poscount)" | bc -l)
```

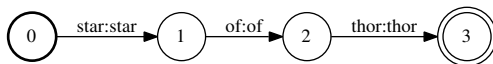
Issues to Solve: Unknown words

- All tags have equal probability to generate an unknown word (or an unknown word has equal probability for all tags).
- We have 38 tags.
- Conditional probabilities must sum to 1.
- Thus: $P(< unk > | t_i) = 1/38$
- Build another transducer for unknown words, compile it, and make a union with the other word transducer.

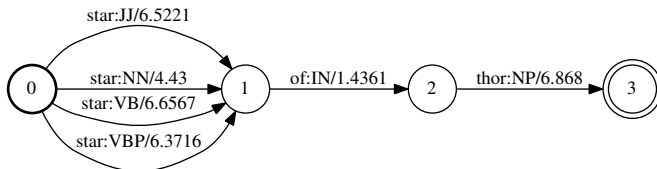
```
prob=$(echo "-l(1/38)" | bc -l)
while read pos count
do
    echo -e "0\t0\t<unk>\t$pos\t$prob"
done < POS.counts
echo "0"
```

Tagging

- ① Represent input sentence “star of thor” as FSA/FST:



- ② Compose it with our tagger to get



- ③ use `fstshortestpath` to get the ‘best’ tag sequence (NN,IN,NP)

Second Term: $P(t_i|t_{i-1})$

Convert token-per-line format into POS-tag sentence-per-line format

```
cat $data | cut -f 2 |\ # get 2nd column
# replace empty line with some special symbol (#)
sed 's/^ *$/#/g' |\
tr '\n' ' ' |\          # replace '\n' with space
tr '#' '\n' |\          # replace '#' with '\n'
sed 's/^ *///g;s/ *///g' # clean redundant spaces
```

Train Language model on this data

```
farcompilestrings --symbols=lex.txt
                  --unknown_symbol='<unk>'
                  $data > data.far
ngramcount --order=3
            --require_symbols=false
            data.far > pos.cnt
ngrammake --method=witten_bell pos.cnt > pos.lm
```


Using All Together

Compose sentence FSA, POS-tagger FST and the POS-LM in a sequence:

$$sent_{FSA/FST} \circ tagger_{FST} \circ pos.lm_{FSA/FST}$$

```
fstcompose sent.fsa pos-tagger.fst |\
fstcompose - pos.lm |\
fstrmepsilon |\
fstshortestpath
```

