Tagging, sequences and dependecies

- Many NLP sub-tasks can be seen as tagging problems for constituents of a document. Examples: Part-of-Speech (PoS) tagging, Named Entity Recognition (NER), morpheme analysis, parsing, Semantic role tagging.
- Many such tagging tasks can be modelled as sequence learning problems.
- Tagging requires standard tagsets. For PoS two tagsets are common: the Penn tree bank tagset and the ICE (International Corpus of English) tagset. We will use the Penn tree bank tagset. NER has a much smaller set of tags. One common categorization is: Location, Person, Organization, Money, Percent, Date, Time, NA (not a tag). Often for each category there are 3 tags a start tag, a continue tag and an end tag e.g SP, CP, EP for start person, continue person, end person.

Tagging example: Word composition from morphemes

A **morpheme** is the smallest unit of text that carries meaning.

- A word is made up of one or more morphemes.
- The main morpheme is the root or stem. The others are affixes.
- Affixes can occur at the beginning prefix (e.g. un-able), at the end suffix (e.g. word-s), both beginning and end circumfix (e.g. ge-sag-t - German) or in between infix - no examples in English.

Used in stemming and other kinds of tagging.

Effect of morphemes on words

Morphemes behave in different ways in languages.

- **Inflection:** stem + morpheme (grammatical marker) e.g. plurals, past - stem + 's', stem + 'ed'. The word remains the same type as the stem and the morpheme performs some syntactic function like agreement - for number or gender.
- **Derivation:** stem + morpheme produces a word usually of a different class. Examples: refuse + 'al', beauty + 'ful', train + 'er'.
- **Compounding:** stem + stem. Examples: blue + berry, under + estimate, over + look, nation + wide.
- **Cliticization:** I + am = I'm, I + have = I've, namah + te = namaste (Sanskrit); French has many clitics - a common one is article + stem = clitic; la + église = l'eglise.

Concatenative and templatic morphology

- Most languages have concatenative morphology - that is morphemes are strung together to form a word.
- Some languages like the Semitic languages (e.g. Arabic, Hebrew) have templates (in addition to concatenation). E.g. in Hebrew the stem lmd (meaning learn/study) can use templates like CaCaC to produce lamad (he studied) or CiCeC - limed (he taught) or CuCaC - lumad (he was taught). The stem is a sequence of 3 consonants.

Morphological analysis

Input	**Output**
word	word+N+Sg
words	word+N+Pl
banks	bank+N+Pl
banks	bank+V
foxes	fox+N+Pl
coming	come+V+PrPl
caught	catch+V+Past
caught	catch+V+PaPl

Given the input in the first column a morphological analyser must produce the output in the second one.

Morphological analysis

Morphological analysis requires:

- Set of all stems.
- Morphological rules: defining order of the morphemes or templates for e.g. pluralization in English: stem+‘s’.
- Surface rendition/orthographic rules: how the text or surface form will be rendered e.g. for plurals fox+‘s’=foxes, mouse+‘s’=mice.

Approaches for morphological analysis:

- One way is to design finite state transducers or FSTs. Such a lexicon is called a finite state lexicon.
- Another is to use sequence to sequence learning and use statistical learning to do morphological analysis.
- It is viable to encode rules deterministically since we have a finite lexicon.

PoS tagging example

We use the Penn tree bank tagset.

Mohan saw the man with the telescope. gives

Mohan/NP saw/VBD the/DT man/NN with/RP the/DT telescope/NN.

Tagging example: NER

NER tagset uses the 7 class tagset (with start, continue, end markers for each class) with NA for not a tag. IOB (Inside, Outside, Beginning) tagset is also possible.

Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

Profits/NA soared/NA at/NA Boeing/SC Co./EC, easily/NA topping/NA forecasts/NA on/NA Wall/SL Street/EL, as/NA their/NA CEO/NA Alan/SP Mulally/EP announced/NA first/NA quarter/NA results/NA.

Tagging as supervised learning

- Tagging can be seen as a supervised learning problem. The learning set $\mathcal{L} = \{(s_1, y_1), \ldots, (s_n, y_n)\}$, where s_i is a sequence of words from a vocabulary $\mathcal{V}$, $s_i = (w_{i_1}, \ldots, w_{i_m})$, $w_{i_j} \in \mathcal{V}$ (s_i is a sentence in our context) and y_i is a sequence of labels, $y_i = (y_{i_1}, \ldots, y_{i_m})$ chosen from the label set $\mathcal{Y}$ - this can be any tagset - for example PoS or NER tagsets.
- The learning algorithm learns a model in the form of a function $f : \mathcal{S} \rightarrow \mathcal{Y}_{seq}$, where $\mathcal{S}$ is a set of finite sequences of words from $\mathcal{V}$ and $\mathcal{Y}_{seq}$ is a set of finite sequences of labels from $\mathcal{Y}$. Note that the mapping f maps sequences of length m to sequences of the same length.
- This learning problem is a special kind of learning problem called a *sequence* learning problem since the order of the elements is important.

Models for sequence learning

- One way to model the problem is to learn the distribution for $P(y|s)$ from the learning set $\mathcal{L}$, where $s \in \mathcal{S}$ and $y \in \mathcal{Y}_{seq}$. Given a test sequence s_t we predict the label sequence using

$$y_t = f(s_t) = \underset{y \in \mathcal{Y}_{seq}}{argmax}\, P(y|s_t)$$

- An alternative way, called the *generative* approach is to model $P(s, y)$ the joint distribution over pairs (s, y) using $\mathcal{L}$. Then we decompose $P(s, y) = P(y)P(s|y)$. Here $P(y)$ is the apriori probability of y and $P(s|y)$ is the probability of generating sequence s given that the underlying label sequence is y. Once we have $P(y)$ and $P(s|y)$ we can use Bayes rule and get: $P(y|s) = \frac{P(s|y)P(y)}{P(s)}$. $P(s) = \sum_{y \in \mathcal{Y}_{seq}} P(y)P(s|y)$ can be calculated by marginalizing over y.
- A third way is to model the sequence learning problem using a recurrent neural network (RNN) and using $\mathcal{L}$ to learn the parameters of the recurrent neural n/w.

To predict the label sequence y_t for a test sequence s_t Bayes rule can be used directly:

$$\begin{aligned} y_t = f(s_t) &= \underset{y \in \mathcal{Y}_{seq}}{argmax}\, P(y|s) \\ &= \underset{y \in \mathcal{Y}_{seq}}{argmax}\, \frac{P(s|y)P(y)}{P(s)} \\ &= \underset{y \in \mathcal{Y}_{seq}}{argmax}\, P(s|y)P(y) \end{aligned}$$

$P(s)$ `is the same for all` y `so it does not affect the argmax.`

- Models that decompose $P(s, y)$ into $P(s|y)$ and $P(y)$ are often called *noisy channel* models. Intuitively, y is generated from the distribution $P(y)$ and then s is generated using $P(s|y)$. $P(s|y)$ can be thought of as a noisy channel which given sequence y outputs sequence s.
- Using $\mathcal{L}$ we have to learn distributions $P(y)$ and $P(s|y)$. This is our model. Then we use the model to predict the label sequence y_t for an unknown sequence s_t using the `argmax` equation above.

Formal generative tagging model

For $n \geq 0$ let $\mathfrak{S}$ be the set of all sequence pairs $(w_1, \ldots, w_n; y_1, \ldots, y_n)$ where $w_i \in \mathcal{V}$, $y_i \in \mathcal{Y}$, $i = 1..n$. A **generative tagging model** is a function P such that:

1. For any $(w_1, \ldots, w_n; y_1, \ldots, y_n) \in \mathfrak{S}$, $P(w_1, \ldots, w_n; y_1, \ldots, y_n) \geq 0$.
2. $\sum_{(w_1,\ldots,w_n;y_1,\ldots,y_n)\in\mathfrak{S}} P(w_1, \ldots, w_n; y_1, \ldots, y_n) = 1$.

So, $P(\ldots)$ is a probability distribution over sequence pairs.

Given a generative tagging model P the function $f : \mathcal{S} \to \mathcal{Y}_{seq}$ mapping word sequences (or sentences) to tag sequences is given by:

$$f(w_1, \ldots, w_n) = \underset{(y_1,\ldots,y_n)\in\mathcal{Y}_{seq}}{argmax} \; P(w_1, \ldots, w_n; y_1, \ldots, y_n)$$

.

So, for any input seq $s \in \mathcal{S}$, $|s| = n$, the output sequence is the highest probablity tag sequence $(y_1, \ldots, y_n) \in \mathcal{Y}_{seq}$.

Questions for a general tagging model:

a) What is the nature of $P(w_1, \ldots, w_n; y_1, \ldots, y_n)$?

b) How do we estimate the parameters of the model?

c) How do we efficiently calculate:

$$\underset{(y_1,\ldots,y_n)\in\mathcal{Y}_{seq}}{argmax} \; P(w_1, \ldots, w_n; y_1, \ldots, y_n)$$

for any input $(w_1, \ldots, w_n)$?

Nature of $P(w_1, \ldots, w_n; y_1, \ldots, y_n)$

Factor P into a conditional probability and an apriori probability:

$$P(w_1, \ldots, w_n; y_1, \ldots, y_n) = P(w_1, \ldots, w_n | y_1, \ldots, y_n) \times P(y_1, \ldots, y_n)$$

If we use the general product (or chain) rule to write the joint distributions in terms of conditional probabilities then calculating all the conditional probabilities becomes intractable so we have to make independence assumptions.
We make two independence assumptions one each for the two factored probabilities.

Independence assumption 1

By chain rule

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n+1} P(y_i | y_1, \ldots, y_{i_1})$$

Assume a tag depends only on two immediate previous tags - y_i depends only on y_{i-1} and y_{i-2}. A second order Markov assumption. Introduce two new tags: START and STOP and assume that $y_{-1} = y_0 = START$ and $y_{n+1} = STOP$.
Any tag sequence looks like $START, START, y_1, \ldots, y_n, STOP$.
We get:

$$P(y_1, \ldots, y_{n+1}) = \prod_{i=1}^{n+1} P(y_i | y_{i-2}, y_{i-1})$$

where $y_{-1} = y_0 = START, \ y_{n+1} = STOP$

Independence assumption 2

The chain rule gives:

$$P(w_1, \ldots, w_n | y_1, \ldots, y_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1}, y_1, \ldots, y_n)$$

.

Assume that given y_i, w_i does not conditionally depend on any other word in the word sequence or any other tag in the tag sequence. Then:

$$P(w_1, \ldots, w_n | y_1, \ldots, y_n) = \prod_{i=1}^{n} P(w_i | y_i)$$

So, the joint distribution becomes:

$$P(w_1, \ldots, w_n, y_1, \ldots, y_{n+1}) = \prod_{i=1}^{n+1} P(y_i | y_{i-2}, y_{i-1}) \prod_{j=1}^{n} P(w_j | y_j)$$

The generative model can be conceptualized as follows:

Algorithm 0.1: GENERATIVEMODEL(p_1, p_2)

```
comment: $p_1 = P(y_i|y_{-2}, y_{-1})$ distribution, $p_2 = P(w_i|y_i)$ distribution
$i \leftarrow 1$
$y_0 = y_{-1} = START$
repeat
 $y_i \leftarrow$ Sample from $p_1$
 if ($y_i \neq STOP$)
   then $w_i \leftarrow$ Sample from $p_2$
until ($y_i == STOP$)
return ($w_1 \dots w_{i-1}$)
```

Estimating the probabilities

$$\begin{aligned} cnt(y_1, y_2, y_3) &\equiv \text{Number of times seq } (y_1, y_2, y_3) \text{ occurs in } \mathcal{L}. \\ cnt(y_1, y_2) &\equiv \text{Number of times seq } (y_1, y_2) \text{ occurs in } \mathcal{L}. \\ cnt(y) &\equiv \text{Number of times tag } y \text{ occurs in } \mathcal{L}. \\ cnt((y, w)) &\equiv \text{Number of times tag } y \text{ is paired with token } w \text{ in } \mathcal{L}. \end{aligned}$$

The maximum likelihood probability estimates are:

$$\begin{aligned} P(y_3|y_1, y_2) &= \frac{cnt(y_1, y_2, y_3)}{cnt(y_1, y_2)} \\ P(w|y) &= \frac{cnt((y, w))}{cnt(y)} \end{aligned}$$

Above gives estimates of the required parameter/probability values of the model.

Problems with probability estimation

The maximum likelihood estimates of probabilities has two problems:

- A token that occurs in a test sequence may not occur at all in $\mathcal{L}$. In this case the probability is 0.
- A token may occur very infrequently in $\mathcal{L}$. Then with high probability the calculated estimate may be very far from the actual probability.

Solution to estimation problem: Pseudo-words

Estimation problems are solved by using pseudo-word tags. $\mathcal{L}$ is tagged with pseudo-word tags where necessary before probabilitiy estimation.

Examples:

Pseudo-word	**Possible meaning**
twoDigitNo	2-digits year
fourDigitNo	4-digits year
capPeriod	Initial
allCaps	Abbreviation or company name
initCap	Name
other	Other words that do not fit

Pseudo-word lists are created based on application (e.g. PoS, NER) and also the nature of the text/domain.

Often final estimates are smoothed:

$P(y_3|y_1, y_2) = \alpha_1 P(y_3|y_2, y_1) + \alpha_2 P(y_3|y_2) + \alpha_3 P(y_3)$ with $\alpha_i \geq 0$ and $\alpha_1 + \alpha_2 + \alpha_3 = 1$.

Decoding

- Calculating $\underset{(y_1,\ldots,y_n)\in\mathcal{Y}_{seq}}{argmax} P(w_1,\ldots,w_n;y_1,\ldots,y_n)$ is called decoding.
- A naive calculation of the sequence $y_1,\ldots,y_{n+1}$ can take $O(\mathcal{Y}^n)$ time in the worst case which is intractable for large n.
- A dynamic programming algorithm called **Viterbi's algorithm** does it in polynomial time.

Viterbi algorithm

Calculate: $\underset{(y_1,\ldots,y_n)\in\mathcal{Y}_{seq}}{argmax} \; P(w_1,\ldots,w_n; y_1,\ldots,y_n)$ where

$$P(w_1,\ldots,w_n; y_1,\ldots,y_{n+1}) = \prod_{i=1}^{n+1} P(y_i|y_{i-2},y_{i-1}) \prod_{j=1}^{n} P(w_j|y_j)$$

Let

$$g(y_{-1},y_0,y_1,\ldots,y_k) = \prod_{i=1}^{k} P(y_i|y_{i-2},y_{i-1}) \prod_{j=1}^{k} P(w_j|y_j)$$

where $y_{-1} = y_0 = START$ and $y_{n+1} = STOP$. This is the same as the earlier equation but truncated at k. So,
$P(w_1,\ldots,w_n; y_1,\ldots,y_{n+1}) =$
$g(START, START, y_1,\ldots,y_n) \times P(STOP|y_{n-1},y_n)$
Let $\mathcal{Y}_k$ = set of tags allowed at position k in the sequence. So,

$$\mathcal{Y}_{-1} = \mathcal{Y}_0 = \{START\},\; \mathcal{Y}_k = \mathcal{Y} \text{ for } k \in \{1\ldots n\}$$

Let $S(k, u, v)$ = set of all tag seq. of length k that end in u, v. Define $\pi(k, u, v)$ = max. probability for any seq. of length k ending in u, v:

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

Recurrences:

Observation 1 $pi(0, START, START) = 1$
Observation 2 For $k \in \{1 \ldots n\}$, $u \in \mathcal{Y}_{k-1}$, $v \in \mathcal{Y}_k$

$$\pi(k, u, v) = \max_{z \in \mathcal{Y}_{k-2}} (\pi(k-1, z, u) \times P(v|z, u) \times P(w_k|v))$$

Observation 3
$$\max P(w_1, \ldots, w_n; y_1, \ldots y_n) = \underset{u \in \mathcal{Y}_{n-1}, v \in \mathcal{Y}_n}{\pi(k, u, v)} \times P(STOP|u, v)$$

Viterbi Algorithm

```
input  : Seq. $w_1, \ldots, w_n$, model parameters $P(y_i|y_{i-1}, y_{i-2})$, $P(w_i|y_i)$
output: Seq. $y_1, \ldots, y_n$
$\mathcal{Y}_{-1} = \mathcal{Y}_0 = \{START\}$, $define\ \mathcal{Y}_k$, $k = 1..n$, usually $\mathcal{Y}$.;
$\pi(0, START, START) = 1$;
for $k$=1 to $n$ do
    foreach $u \in \mathcal{Y}_{k-1}$, $v \in \mathcal{Y}_k$ do
        $\pi(k, u, v) = \max_{z \in \mathcal{Y}_{k-2}} \pi(k-1, z, u) \times P(v|z, u) \times P(w_k|v)$;
        $Bptr(k, u, v) = \mathrm{argmax}_{z \in \mathcal{Y}_{k-2}} \pi(k-1, z, u) \times P(v|z, u) \times P(w_k|v)$;
        /* Store $z$ for which $\pi(k, u, v)$ is max.                */
    end
end
$(y_n, y_{n-1}) = \mathrm{argmax}_{u \in \mathcal{Y}_{n-1}, v \in \mathcal{Y}_n} \pi(n.u, v) \times P(STOP|u, v)$;
for $k = n - 2..1$ do
    $y_k = Bptr((k + 2, y_{k+1}, y_{k+2})$;
end
return $(y_1, \ldots, y_n)$;
```

Algorithm 1: Viterbi algorithm with back pointers