Words

Notes on Natural Language Processing

Chia-Ping Chen

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan ROC

Introduction

- words: the building blocks of human languages
- emphasis on computational aspects
- modeling targets
  - spelling
  - pronunciation
  - morphology
- modeling tools
  - finite-state automata and transducers
  - hidden Markov models
  - n-gram models

Pattern Matching

- Suppose we want to search a collection of texts for the following “meaningful” strings
  - woodchuck, woodchucks, Woodchuck(s)
  - $199, $24.99, and other dollar amounts
  - variable identifiers in c programming language
- all modeled by **regular expressions**

Regular Expression

- a tool for string search, used in editing applications such as `emacs, vi, perl`, and others
- a specification of a pattern
- all strings conforming to such a pattern (a regular expression) constitute a set of strings, i.e., a language

Regular Expression Search

- requires a regular expression and a **corpus** (text collection)
- searching the corpus for the pattern dictated by the given regular expression
- For example, the `grep` utility returns the lines in text file(s) containing the pattern.

Basic Patterns

- a sequence of plain symbols: `/urgl/`
- disjunction: `/[wW]oodchuck/`
- range: `[0-9]`, `[a-zA-Z]`
- negation by caret (used with opening `[`): `[^A-Z]`
- 0 or 1 occurrence: `/[wW]oodchucks?/`
- 0 more occurrence: `/ba*/`
- 1 more occurrence: `/ba+/` = `/baa*/`
- wildcard: `/beg.n/` matches `begin` and `began`

Anchors

- the start of a line: `/^The/`
- the end of a line: `/^The dog\.$/`
- word boundary: `/\bthe\b/`
- non-boundary: `/\Bthe\B/`

Precedence

- disjunction of strings: `/dog|cat/` meaning?
- parenthesis: `/pupp(y|ies)/`
- Operator precedence hierarchy
  1. parenthesis: `( )`
  2. counters: `* + ?`
  3. sequences and anchors: `the`, `^my`, `end$`
  4. disjunction: `|`

An Example

Suppose we want to find the occurrences of `the`.

- The first thought is to use `/the/`.
- But we want `The` as well. So we use `/[tT]he/`.
- But we don't want `other`. So we use `/\b[tT]he\b/`.
- But we want also `_the_`. So we use

  `/[^a-zA-Z][tT]he[^a-zA-Z]/`.

- Still, we are missing those beginning a line. So we use

  `/(^|[^a-zA-Z])[tT]he[^a-zA-Z]/`.

A More Complex Example

“any PC faster than 2 GHz, and less than $1000”

- `/$[0-9]+\.[0-9][0-9]/`: $199.99 but not $199
- `/$[0-9]+(\.[0-9][0-9])?/`
- `/\b[2-9](\.[0-9]+)?\b*(GHz|[Gg]igahertz)/`

Alias

- \d: any digit, same as `[0-9]`
- \D: any non-digit, same as `[^0-9]`
- \w: alphanumeric/underscore, `[a-zA-Z0-9_]`
- \W: non-alphanumeric/underscore, `[^a-zA-Z0-9_]`
- \s: whitespace
- \S: non-whitespace

Numbers of Occurrences

- `{n}`: exactly n occurrences of the last expression
- `{n,m}`
- `{n,}`: at least n occurrences of the last expression

Special Characters

- \\`*`: literal `*`
- \\`.`: literal `.`
- \\?: literal ?
  - Note `*`, `.`, `?` have non-literal meanings.
- \\n: new-line
- \\`t`: tab

Registers

- Parenthesis is used to delimit pattern.
- A matched pattern is stored in a number variable \n, e.g., \1 is the matched instance of the first pattern.
- examples
  - put brackets around the first woodchuck

    ```
    s/(woodchuck)/<\1>/
    ```

  - used with pattern matching

    ```
    /the (.*)er they (.*), the \1er they \2/
    ```

ELIZA

- This system responds by text substitution according to certain rules written by regular expressions.
- A session goes like this.
  - U: Men are all alike
  - E: IN WHAT WAY (`s/.* all .*/IN WHAT WAY/`)
  - U: They're always bugging us about something
  - E: CAN YOU THINK OF A SPECIFIC EXAMPLE
  - U: Well, my boy friend made me come here
  - E: YOUR BOYFRIEND MADE YOU COME HERE

Finite-State Automata

- After studying regular expressions, we now turn our attention to **finite-state automata** (FSA).
- Both regular expressions and finite-state automata describes the same set of languages.
- Specifically, if $L = L(M)$ for some FSA M, then $L = L(R)$ for some regular expression R. The converse is also true.
- They are said to be **equivalent**.

Components of an FSA

- A finite-state automata, say A, is characterized by a 5-tuple $\{Q, \Sigma, \delta, q_0, F\}$
  - Q: a finite set of states
  - Σ: a finite set of input symbols
  - $\delta(q, a) \ \forall a \in \Sigma, q \in Q$: state transition function
  - q_0: initial state
  - F: a set of final states

Mechanism of an FSA

- A begins with state q_0.
- Given an input string w, it processes w from left to right, and make state transitions.
- Specifically, suppose the current input symbol is a and the current state is q, then the next state q' is determined by δ
$$q' = \delta(q, a).$$
- When the end of input string is processed, the last state, say p, decides whether w is accepted.

The Language of an FSA

- A string w is said to be **accepted**, or **recognized**, by A if the last state is in the set of final states, $p \in F$.
- The **language** of A is the set of strings accepted by A, denoted by $L(A)$.

An Example: Sheeptalk

The sounds of sheep can be modeled by

$$L = \{\texttt{baa!,baaa!, baaaa!,...}\}.$$

- L can be described by a regular expression `/baa+!/`.
- L can be described by a FSA, where
  - $Q : \{q_0, q_1, \ldots, q_4\}$
  - $\Sigma : \{b, a, !\}$
  - $q_0 : q_0$
  - $F : \{q_4\}$
  - $\delta(q, i)$
- Figure 2.10

Dollar Amounts

- Suppose we want an FSA to model dollar amounts (in English), such as `ten cents, three dollars, one dollar thirty-five cents`, etc.
- It is more convenient to use words as symbols instead of letters.
  - Fig 2.15: accepting the numbers 1 to 99
  - Fig 2.16: including dollars and cents
- For larger amounts, we need to deal with hundred, thousand, million, and so on.
- You get the ideas.

Non-Deterministic FSA’s

- In the definition of FSA, the *state transition function* δ is “complete”, and the value is a single state.
  - Given an input string, there is only one path in the graph of the FSA.
- In a non-deterministic FSA (NFSA), there are *multiple choices* of next state given current state and input symbol. That is, the value of δ is a *set* of states.
- We also allow “spontaneous” state transition, one without consuming any input symbol.
  - multiple paths in the graph for a given input string

The Language of NFSA

- Given an NFSA N and an input string w, w is said to be accepted by N if any one of the state sequences ends in a final state.
- The set of strings accepted by N is the language of N, denoted by $L(N)$.
- The sheeptalk can be recognized by NFSA (Figures 2.17, 2.18).

Acceptance/Rejection by Search

- Whether w is accepted by N is a **search** problem.
  - acceptance = there exists one path ending in a final state
  - rejection = looking at every candidate and none can be found
- To systematically exhaust all possibilities
  - at a point of multiple choices, the search proceeds with one and save the alternatives for later.
  - when no further progress can be made, the search backs up to a previous point of choice.

Search State

- For NFSA, at a point of choice, we need to store
  - the state
  - the input position
- The combination of state and input position is called a **search-state**.
- We look for a search-state with a final state and the end position to accept.

Order of Search

- depth-first search
  - next search-state: the most recent
  - search agenda: a stack
  - It is commonly referred to as a depth-first search, or Last-In-First-Out (LIFO).
- breadth-first search
  - next search-state: the least recent
  - search agenda: a queue
  - It is commonly referred to as a breadth-first search, or First-In-First-Out (FIFO).

Issues

- depth-first search
  - infinite loop
- breadth-first search
  - often use an enormous amount of memory

Recursive Definition

- The set of regular languages represented by regular expressions over an alphabet Σ can be defined by
  - $\emptyset$ is a regular expression for $\emptyset$
  - $\forall a \in \Sigma \cup \{\epsilon\}, L(a) = \{a\}$
  - If E, F are regular expressions
    - $E + F$ is a regular expression and $L(E + F) = L(E) \cup L(F)$
    - EF is a regular expression and $L(EF) = L(E)L(F)$
    - E^* is a regular expression and $L(E^*) = (L(E))^*$
    - (E) is a regular expression and $L((E)) = L(E)$

Closure Properties

- intersection

$$L_1 \cap L_2.$$

- complementation

$$\overline{L} = \Sigma^* - L.$$

- reversal

$$L^R = \{w \mid w^R \in L\}.$$

- difference (from intersection and complementation)

$$L_1 - L_2 = L_1 \cap \overline{L_2}.$$

Morphemes

- **morpheme**: minimal bearing unit of **meaning**
- for instance
  - `fox`: one morpheme `fox`
  - `cats`: two morphemes `cat` and `-s`
- in the previous example
  - `cats` is called the **surface form**
  - `NOUN-cat` + `PL-s` is called the **lexical form**

Morphological Parsing

- given the surface form, find the lexical form

$$\texttt{foxes} \Rightarrow \texttt{NOUN-fox} + \texttt{PL-es}$$

- For the nouns, there are quite a few rules in converting a singular noun to its plural,

  `book, fox, goose`.

  Similarly for the verbs.

- A noun and a verb may have the same surface form, further complicating the problem.

Stems and Affixes

- two broad classes of morphemes
  - **stems**
  - **affixes**
    - **prefixes**: preceding a stem, `undo`
    - **suffixes**: following, `eats`
    - **infixes**: inserting inside
    - **circumfixes**: wrapping around, `gesagt`
- A surface form may have more than one affixes,

  `unlikely, rewrites`.

Inflection and Derivation

- A surface form and its stem may be in the same or different **part of speech (POS)**.
- Based on this, there are two types of morphological transformation
  - **inflection**: a stem is transformed into a surface form of the *same* POS
  - **derivation**: a stem is transformed into a surface form of a *different* POS

Inflectional Morphology

- English noun inflection
  - affix for plural: `-s(cats)`, `-es(brushes)`, `-en(oxen)`
  - affix for possessive: `-'s(cat's)`, `-'(cats')`
- English verbal inflection
  - regular: `-s(sends), -es(watches), -ing(watching), -ed(watched)`
  - irregular: `eat/ate, cut/cut`

Derivational Morphology

- Nouns can be derived from adjectives or verbs, e.g., `-ation(computerization)`, `-ee(appointee)`, `-er(killer)`, `-ness(fuzziness)`, `-ity(reality)`.
- Adjectives can be derived from nouns or verbs, e.g., `-al(national)`, `-able(doable)`, `-less(clueless)`.

Morphological Parsing

- example of morphological parsing

$$\begin{cases} \texttt{cats} \Rightarrow \texttt{cat +N+PL}, \\ \texttt{geese} \Rightarrow \texttt{goose+N+PL}, \\ \texttt{gooses} \Rightarrow \texttt{goose +V+3SG}. \end{cases}$$

- Specifically, a word in its surface form is parsed into its stem and a set of **morphological features**, such as
  - POS
  - plurality
  - 3SG
  - present participle, past participle, past tense, etc

Components of a Parser

- **lexicon**: the lists of stems (with POS) and affixes
- **morphotactics**: a model (rules) of morpheme ordering in a word
  - to draw an analogy, syntax is a model of word ordering in a sentence
- **orthographic rules**: spelling rules when morphemes combine
  - e.g., `y`→`ie` when `city + PL`

Nominal Inflection

- lexicon
  - `reg-noun` (cat, dog)
  - `-s` (for plural).
  - `irreg-sg-noun` (goose, mouse)
  - `irreg-pl-noun` (geese, mice)
- FSA (Fig 3.3)

Verbal Inflection

- FSA (Fig 3.4)
- There are three classes for verb stems
  - `reg-verb-stem` (walk, impeach)
  - `irreg-verb-stem` (cut, sing)
  - `irreg-past-verb-form` (caught, sang)
- There are also four affix classes
  - `-ed` (for past tense)
  - `-ed` (for past participle)
  - `-ing` (for present participle)
  - `-s` (for 3SG)

Adjectives

- FSA (Fig 3.5)
  - lexicon: stems (such as `big, cool`), prefix (`un-`), and suffix (`-er,-est,-ly`)
  - morphotactics: a stem, optional prefix, optional suffix
- However, it also accepts something like *unbig* ...
  - need to separate stems which allow prefix from those which do not allow prefix ...

Alphabet Set

- The alphabet (symbol set) of a morphological FSA contains “morphemes” so far.
- The FSA can be *expanded* so that the new alphabet contains *letters*.
- Fig 3.7 gives an toy example for noun FSA
  - the label of any path that starts at the initial state and ends at a final state is a noun
  - not exact, for example, the wrong word `foxs` is accepted

Finite-State Transducer

- A **finite-state transducer (FST)** maps an input string to an output string.
- A deterministic FST is defined by
  - Q: a finite state set
  - Σ: input alphabet
  - Δ: **output alphabet**
  - q_0: the start state
  - F: the set of final states
  - $\delta(q, w)$: transition function, returning a state in Q
  - $\sigma(q, w)$: **output function**, returning a string in Δ^*
- For FST as a morphological parser, the input is a string of morphemes and the output is a string of letters.

Inversion and Composition

- The **inversion** of an FST T is another FST, T^{-1}, such that

$$T(w) = u \;\Rightarrow\; T^{-1}(u) = w, \;\forall\, w$$

- The **composition** of two FSTs T_1 and T_2 is an FST, $T_1 \circ T_2$, such that

$$\begin{cases} T_1(x) = y \\ T_2(y) = z \end{cases} \quad \Rightarrow \; T_1 \circ T_2(x) = z, \;\forall\, x, y, z$$

One can also write

$$T_1 \circ T_2(x) = T_2(T_1(x)).$$

Apply FST to M-Parsing

- the two levels in m-parsing
  - lexical level: concatenation of morphemes
  - surface level: concatenation of letters
- the two **tapes** of a m-parsing FST
  - (upper) lexical tape: for the concatenation of morphemes
  - (lower) surface tape: for the concatenation of letters

T_{num}

- an FST for singular/plural inflection (Fig. 3.13)
  - based on Fig. 3.3
  - add morphological features, `+N, +SG, +PL`
  - some features are mapped to the empty string, morpheme boundary marker # or word boundary marker $^\wedge$, as they do not produce any segment on the surface tape
- Note in Fig. 3.13, input symbol is above a transition while output symbol is below; default input/output pair $a : a$ is simply denoted by by a.

T_{lex}

- Fig. 3.13 needs to be expanded into letters, resulting in Fig. 3.14 for the lexicon immediately above.
- This FST is denoted by T_{lex}.
- For example, we have

  $$\texttt{c:c a:a t:t +N:}\epsilon\ \texttt{+PL:}^\wedge\texttt{s\#}$$

  The output string is not quite the surface form.
- We put the output string in an intermediate tape.

Orthographic Rules

- We need yet another transduction process from the intermediate tape to the surface tape.
- Here we focus on the application of orthographic rules.
- an example for such a rule is the e-insertion.
  - rule notation

$$\epsilon \rightarrow e / \begin{bmatrix} x \\ s \\ z \end{bmatrix} \wedge \_\_ \, s\#$$

  - implemented by an FST T_e (Fig. 3.17)

Putting It Together

- There maybe other rules, each may be implemented by an FST.
- The entire picture is given in Fig. 3.19.
- As an example, the transduction of `foxes` is shown in Fig. 3.20.

Language Models

- To develop **computational linguistics**, mathematical models for natural languages are required.
- A **language model** gives a probability to a sentence.
  - In speech recognition, this is combined with **acoustic model** score to decide optimal hypothesis.
  - In machine translation, this is combined with **translation model** score to decide optimal translation.
- Here we introduce the n-**gram** language models.

Word Prediction

- A language model can help us in **word prediction**, the task of guessing next word. For example,

  `I'd like to make a collect ...`

- WP can help saving significant typing efforts in certain situations
  - disabled users
  - Chinese input
  - mobile devices

n-Gram

- An n-gram is an n-word sequence. For example,
  - a **bigram** is a sequence of two words;
  - a **trigram** is a sequence of three words;
  - a **unigram** is a single word.
- For clarity, we also define the following terms
  - a **type** is a distinct sequence (n-gram);
  - a **token** is an instance of a type.
- Thus, in the string

  `reading a good book is a good thing`

  there are 6 unigram types and 8 unigram tokens.

The Probability of A Sentence

- What is the probability of a sentence, say

$$S = w_1, \ldots, w_i?$$

- Applying the **chain rule of probability**,

$$Pr(S = w_1, \ldots, w_i) = Pr(w_1)Pr(w_2|w_1)\ldots Pr(w_i|w_1, \ldots, w_{i-1}).$$

- If we have all the probabilities, we can compute the probability of S.
- However, the size of the probability table

$$Pr(w_i|w_1, \ldots, w_{i-1})$$

grows exponentially with i.

n-Gram Language Model

- An n-gram language model is based on the following approximation (or assumption), that
  - *the probability of a word following an* ($i-1$)*-gram depends at most on the last* ($n-1$)*-gram*.
- That is,

$$p(w_i = w | w_{1:i-1}) = p(w_i = w | w_{i-n+1:i-1}).$$

Note the notation $w_{m:n} = w_m, w_{m+1}, \ldots, w_n$.

How to Get Probability?

- Probability is based on *counting*.
- The basic idea is that of **relative frequency**.
- It is reasonable to assume

$$Pr(w_2|w_1) = \frac{o(w_1 w_2)}{o(w_1)}$$

  where $o(t)$ is the number of occurrences of type t.
- To obtain these numbers (counts), we need a collection of text, a.k.a. **text corpus**.

Text Corpus and LM

- Using one text corpus leads to an n-gram probability, while using another corpus leads to another.
- Put in another way, the language model is highly dependent on the text corpus.
- exemplar text corpora
  - **Switchboard**: telephone conversations, $20k$ types, $2.4m$ tokens
  - **Brown corpus**: written texts, $60k$ types, $1m$ tokens

Data Sets

- The data used to learn the language models and the data to evaluate the learned models should be disjoint.
  - The learning data is called the **training set**.
  - The evaluating data is called the **test set**.
- Sometimes we need extra set to learn certain additional parameters. That is called the **held-out set**.
- Sometimes we have no test set but need one for evaluation. That is called the development test set, or **devset**.

Random Sentence Generation

- We can generate random sentences from the n-gram models estimated from the work of Shakespeare.
- randomly generated samples, Fig. 4.3
  - unigram
  - bigram
  - trigram
- The larger n, the more *coherent* are the generated sentences.
- Using n-grams trained by the **Wall Street Journal** corpus results in Fig. 4.4.
- Differnt training sets leads to different language models, generating different kinds of sentences.

LM Evaluation

- **extrinsic evaluation**: apply LM in an application, to see the performance of that application
  - e.g., measuring word error rate in speech recognition
- **intrinsic evaluation**: use a measure that is independent of any application
  - e.g., perplexity

Perplexity

- Given a LM, say p, the perplexity of a text

$$W = w_1, \ldots, w_N$$

  is defined by

$$\text{PPL}(W) = p(W)^{-\frac{1}{N}}.$$

- average branching factor of next word
- For n-grams, often a larger n leads to a lower perplexity without *data paucity*.

Data Paucity

- The data paucity problem, a.k.a., the 0-occurrence problem, can be stated as follows.
  - Some n-grams may not occur in a given corpus and the estimated probability is 0.
  - If the test set contain any n-gram instance that does not occur in the training set, then the probability is 0.
- Indeed, as n gets large, the majority of n-gram types will have 0 counts with any realistic corpus.

LM Smoothing

- trade-off of n in n-grams
  - large n for accurate model, but non-robust parameter estimation
  - small n for robust parameter estimation, but inaccurate model
- **Smoothing** can hedge this problem.
  - A large n can be adopted, while some probability mass is re-distributed to the n-grams with 0 or very low counts.

Laplace Smoothing

- a.k.a. add-1 smoothing
- add 1 to every n-gram count; $0 \rightarrow 1$
- The smoothed probability is

$$\begin{cases} p^*(w) = \frac{o(w)+1}{\sum_{w'} o(w')+V}, & \text{unigram smoothing} \\ p^*(w|w_0) = \frac{o(w_0 w)+1}{\sum_{w'} o(w_0 w')+V}, & \text{bigram smoothing} \end{cases}$$

- The same idea can be generalized to add-$\delta, (\delta > 0)$. For type t,

$$p^*(t) = \frac{o(t) + \delta}{N + \delta V}.$$

Discounting

- In add-1 smoothing, we are effectively changing the count of type t to be

$$c^*(t) = (o(t) + 1)\frac{N}{N + V},$$

and

$$p^*(t) = \frac{c^*(t)}{N}.$$

- The actual number of occurrences is $o(t)$, so we also say that each occurrence is **discounted** by

$$d(t) = \frac{c^*(t)}{o(t)}.$$

Example

- Fig. 4.1: bigram counts and unigram counts
- Fig. 4.2: bigram probability before smoothing
- Fig. 4.5: adding 1 to every bigram count in
- Fig. 4.7: $c^*(t)$ (multiplied by $\frac{N}{N+V}$)
- Fig. 4.6: add-1 smoothed bigram probability

$$p^*(w|v) = \frac{o(vw) + 1}{o(v) + V}.$$

- large discount when $V >> N$.

Good-Turing Discounting

- described by Good, who credits Turing with the idea
- **frequency of frequency**: the number of types (first frequency) with a certain count (second frequency)

$$N_c = \sum_{t:o(t)=c} 1.$$

- The new count c^* (the count of a type which occurs c times in the training data) is replaced by

$$c \longrightarrow c^* = (c+1)\frac{N_{c+1}}{N_c}.$$

Good-Turing Discounting

- The total count is conserved, since

$$\sum_{c=0}^{\infty} c^* N_c = \sum_{c=0}^{\infty} (c+1) N_{c+1} = \sum_{c'=1}^{\infty} (c') N_{c'} = \sum_{c=0}^{\infty} c N_c.$$

- Suppose this total count is C. The probability is for a type t with $o(t) = c$ is simply

$$P^*(t) = \frac{c^*}{C}$$

- Specifically, for the set of all unseen types U,

$$P^*(U) = \frac{N_1}{C}.$$

Example

- Fig. 4.8: GTD from two different corpora
- The count for $c \geq 1$ is reasonably discounted.
- Same idea can be applied to all low-count types, not just the 0-count types.
- Contrarily, we can also assume that $o(t) > k$ is reliable for a threshold k.

n-Gram Hierarchy

- We have seen how to deal with the 0-occurrence problem with smoothing/discounting schemes.
- Another line of approach is to use the probability of a lower n-gram.
  - If the count of uvw is zero, we approximate $p(w|uv)$ by $p(w|v)$.
  - If the count of vw is also zero, we approximate by $p(w)$.
- This is called n-gram hierarchy. We introduce schemes of
  - **(deleted) interpolation**
  - **backoff**

Interpolation

- linear combination of n-gram probabilities of progressive n's
- For example, for trigrams,

$$\widehat{p}(w|uv) = \lambda_1 p(w|uv) + \lambda_2 p(w|v) + \lambda_3 p(w), \;\; \sum_i \lambda_i = 1.$$

- λ_i is often learned from a **held-out corpus**.
- λ_i's may depend on u, v.

Backoff

- In the *interpolation* scheme, the lower-order n-grams are always used.
- In the *backoff* scheme, a lower-order n-gram is used only when the higher orders fail.
- For example, for a simple trigram backoff

$$\widehat{p}(w|uv) = \begin{cases} p(w|uv), & o(uvw) > 0, \\ \alpha\widehat{p}(w|v), & o(uvw) = 0. \end{cases}$$

  - Note that those $p(w|uv)$ with $o(uvw) > 0$ alone add up to 1, so $\widehat{p}$ is not a probability.

Normalization Constant

- We can use a discount scheme, such as Good-Turing,

$$p(w|uv) \rightarrow \tilde{p}(w|uv).$$

- α is set so that the total probability is 1. Define the set $A = \{w|o(uvw) > 0\}$ with non-zero trigram count.

$$\begin{aligned} 1 &= (\sum_{w' \in A} + \sum_{w' \notin A}) \,\widehat{p}(w'|uv) = \sum_{w' \in A} \tilde{p}(w'|uv) + \sum_{w' \notin A} \alpha \widehat{p}(w'|v) \\ &= \sum_{w' \in A} \tilde{p}(w'|uv) + \alpha(1 - \sum_{w' \in A} \widehat{p}(w'|v)) \\ \Rightarrow \alpha &= \frac{1 - \sum_{w' \in A} \tilde{p}(w'|uv)}{1 - \sum_{w' \in A} \widehat{p}(w'|v)}. \end{aligned}$$

- Fig. 4.9

Class-Based n-Grams

- The probability depends on the previous POS tags.
- More specifically, the conditional probability of the next word is

$$\begin{aligned} Pr(w_{1:i}) &= \sum_{c_{1:i}} Pr(w_{1:i}|c_{1:i}) Pr(c_{1:i}) \\ &= \sum_{c_{1:i}} \prod_j Pr(w_j|c_j) \prod_j Pr(c_j|c_{j-n+1:j-1}) \end{aligned}$$

- more reliable estimation of parameters as

$$|C| \;<<\; |V|$$

Part-of-Speech

- The significance of **parts-of-speech** (a.k.a. **POS, word classes, lexical tags**) is the large amount of information they give about a word and its neighbors.
- In fact, grammatical rules are written in terms of word classes.
- The POS tag of a word in a sentence, e.g. `noun`, `preposition`, indicates its syntactical role.
  - The same word may be labeled by a different tag in a different context. For example, *book*.

English Word Classes

- closed: POS categories with fixed members
  - `preposition, pronoun`
  - small sets
  - function words
- open: POS categories with non-fixed members
  - `noun, verb`
  - large sets
  - content words

Noun

- a POS tag often given to words for people, places, things
  - *friend, restaurant, desk*
- also given to verb-like word according to the syntax
  - His *writing* is good.
- given to abstraction as well
  - *quality, generalization*
- miscellaneous
  - proper nouns, *Chen, Kaohsiung*
  - mass nouns vs. count nouns

Verbs

- used to refer to actions and processes
  - *paint, count, deal*
- morphological forms of a verb (*sing*)
  - `3rd-person-sg` (*sings*)
  - `progressive` (*singing*)
  - `past` (*sang*)
  - `past-participle` (*sung*)

Adjectives and Adverbs

- `adjectives`: describing properties or qualities
  - color (*pink/white*), size (*large/small*), age (*old/young*), etc.
- `adverbs`: modifying/specializing something

  John walked home extremely slowly yesterday.

  - locative adverbs (*home, there*)
  - degree adverbs (*much, rarely, extremely*)
  - manner adverbs (*quickly, slowly*)
  - temporal adverbs (*now, tomorrow, yesterday*)

Closed Classes

- prepositions
- determiners
- pronouns
- conjunctions
- auxiliary verbs
- particles
- numberals

Prepositions and Particles

- `preposition`: used to indicate spatial or temporal relation
  - <u>under</u> the table, <u>before</u> noon
- `particle`: combined with a verb to form a larger unit called **phrasal verb**
  - call <u>off</u>, give <u>up</u>, give <u>in</u>
- Fig. 5.1 lists prepositions and particles.

Determiners

- occurring with `nouns`, marking the beginning of a noun phrase

  a book, the book, this book

- `articles`: a subtype of `determiners`, consisting of *a, an, the*
- extremely frequent in English

Conjunctions

- used to join two phrases, clauses or sentences
  - coordinating conjunctions: joining two elements of equal status (*and, or, but*)
  - subordinating conjunctions: one of the element is of an embedded status (*that*)
- Fig. 5.3 lists conjunctions of English.

Miscellaneous Classes

- `pronouns`: *I, my, who*
- `auxiliary verbs`: *be, have/do, can/may*
- `interjections`: *oh, ah, hey, man, alas*
- `negatives`: *no, not*
- `politeness markers`: *please*
- `greetings`: *hello*
- `existential there`: *there*

POS Tagsets

- varies with application
- Penn Treebank tagset: see Fig. 5.6
- Each word token has a tag.

  The/`DT` grand/`JJ` jury/`NN` commented/`VBD` on/`IN`

  a/`DT` number/`NN` of/`IN` other/`JJ` topics/`NNS` ./`.`

POS Tagging

- the process of **assigning a POS tag** to each word token in a corpus
  - Often a punctuation mark is considered a token.
- We are primarily interested in **automatic** taggers.

Ambiguity

- POS tagging is non-trivial due to **ambiguity**.
- Suppose the input word string is

  Book that flight.

  - *Book*: noun? verb?
  - *that*: determiner? conjunction? pronoun?
- In Brown corpus, 11.5% word types are ambiguous, while 40% of word tokens are ambiguous.
  - most word types are unambiguous
  - but many common words are ambiguous

Tagging Algorithms

- simple tagger: assigning to a word token its most frequent tag
  - actually not too bad
- **rule-based tagger**: using a set of hand-written disambiguation rules
- **stochastic tagger**: computing the probability of POS tags in a sentence
  - HMM
  - log-linear model

Stochastic POS Tagging

- Bayes criterion

$$T^* = \arg\max_T P(T|W).$$

  - $T = t_1^n$: tag sequence
  - $W = w_1^n$: word (token) sequence

- T^* minimizes the probability of sequence-level error, over all candidates T,

$$1 - P(T|W).$$

Stochastic POS Tagging

- Note that T^* minimizes the expected number of sequence-level errors, as

$$E(N_s = I_{T \neq R}) = E(I_{T \neq R}) = (1 - P(T|W)).$$

- In contrast, the total number of tag-level errors is

$$N_t = \sum_{i=1}^{l} I_{t_i \neq r_i}.$$

- To minimize $E(N_t)$, the criterion should be

$$t_i^* = \arg\max_t P(t_i = t|W).$$

- Stringing t_i^* together does not necessarily yields T^*!

Prior and Likelihood

- We can re-write the posterior probability as

$$P(T|W) = \frac{P(T,W)}{P(W)} = \frac{P(W|T)P(T)}{P(W)} \propto P(W|T)P(T)$$

  - $P(T)$ is called the **prior probability** of T
  - $P(W|T)$ is called the **likelihood** of W for given T
- Note

$$\arg\max_T P(T|W) = \arg\max_T P(T)P(W|T),$$

since $P(W)$ does not vary with T.

HMM Assumptions

- HMM (of order k) makes two assumptions
  - the probability of a word token, given its POS tag, is independent of any other things

$$P(w_1^n|t_1^n) = \prod_{i=1}^{n} P(w_i|t_i)$$

  - the probability of a POS tag, given its $(k-1)$ previous tags, is independent of any other things

$$P(t_1^n) = P(t_1^{k-1}) \prod_{i=k}^{n} P(t_i|t_{i-k+1}^{i-1})$$

Bigram HMM Taggers

- Suppose we have a bigram HMM tagger.
- The joint probability of (T, W) is

$$p(W|T)p(T) = p(t_1)p(w_1|t_1)\prod_i p(t_i|t_{i-1})p(w_i|t_i).$$

- With the neighboring fixed, the optimal tag t_i depends on t_{i-1}, t_{i+1}, and w_i,

$$t_i^* = \arg\max_{t_i} p(w_i|t_i)p(t_i|t_{i-1})p(t_{i+1}|t_i).$$

An Example

- Consider the sentence W

Secretariat is expected to race tomorrow.

- Fig. 5.12 illustrates possible 2 state sequences for the given W.
- We know that *race* is a verb.
- Using bigram HMM, we need to compare

$$p(\textbf{race}|VB)\ p(VB|TO)\ P(NR|VB),$$

$$p(\textbf{race}|NN)\ p(NN|TO)\ P(NR|NN).$$

Decoding HMM

- Given W, find T.
- Viterbi decoding
  - initialization, recursion, and termination

$$v_0(0) = 1;$$
$$v_m(j) = \max_i v_{m-1}(i) a_{ij} b_j(w_m);$$
$$v_{n+1}(f) = \max_i v_n(i) a_{if};$$

    where m is the index of input position, state 0 and f are initial and final states.
  - back-trace
- Fig. 5.18 depicts the algorithm.

Confusion Matrix

- Suppose there are N classes. The **confusion matrix** or the **contingency table** C is an $N \times N$ matrix, where C_{ij} is the number of instances of class i classified as class j.
- a useful tool for error analysis for classification problems
- can also be used to estimate (error) probability (each row can become a probability function)

Noisy Channel Model

- **Spelling** can be modeled as mapping from one (intended) string of symbols to another (spelled).
- One can imagine a channel that has the correct word as input and the observed string of letters as output.
- The channel is noisy, so input and output may differ (*typo*).
- The same idea, called **noisy channel model**, can be applied to model any input/output relations.

Spelling Error Patterns

- single-error mis-spellings
  - **insertion**: the $\rightarrow$ ther
  - **deletion**: the $\rightarrow$ th
  - **substitution**: the $\rightarrow$ thw
  - **transposition**: the $\rightarrow$ hte
- For typing, they occur due to *typographic* or *cognitive* factors.
- For OCR (optical character recognition), there are other error types, e.g. space deletion/insertion and multiple substitution.

Optimal Solution

- We are given observation O and want to decide an optimal $\widehat{w}$.
- The probability of error when deciding w, $P_e(w)$, is

$$1 - P(w|O).$$

- $\widehat{w}$ that minimizes $P_e(w)$ must maximize $P(w|O)$, so

$$\widehat{w} = \arg\max_w P(w|O) = \arg\max_w \frac{P(O,w)}{P(O)} = \arg\max_w P(O|w)P(w).$$

- We call $P(O|w)$ *likelihood* and $P(w)$ *prior*.

Example

- Consider a non-word error `acress`.
- actress? cress? caress? access? assess? ...
- Which is the most likely original word?
- How do we decide?
- What do we need to decide?

Single-Error Assumption

- To simplify the problem, we suppose typo t differs from the correct spelling c by a single-error mis-spelling.
- The Bayes decision rule is

$$\widehat{c} = \arg\max_{c} P(t|c)P(c).$$

- To decide $\widehat{c}$, we need two probabilities.
  - $P(c)$
  - $P(t|c)$
- The problem now is how to estimate $P(c), P(t|c)$.

Kernighan's Idea

- Kernighan proposed to approximate $P(acress|across)$ by $S(e|o)$, the probability of substituting o by e.
- To estimate $S(e|o)$, we use a corpus of spelling errors and count the number of times o is typed as e.
- Let the error counts be stored in a matrix $sub[o, e]$.
- Define the counts of o to be the sum

$$count[o] = \sum_{\alpha} sub[o, \alpha].$$

- Then

$$S(e|o) = \frac{sub[o, e]}{count[o]}.$$

Confusion Matrices for Single Errors

- $del[x, y]$: the count that xy is typed as x
- $ins[x, y]$: the count that x is typed as xy
- $sub[x, y]$: the count that x is typed as y
- $trans[x, y]$: the count that xy is typed as yx
- the single-error probabilities can be estimated by counts of a labelled corpus

$$\begin{cases} D(x|xy) = \frac{del[x,y]}{count[xy]}, & \text{deletion} \\ I(xy|x) = \frac{ins[x,y]}{count[x]}, & \text{insertion} \\ S(y|x) = \frac{sub[x,y]}{count[x]}, & \text{substitution} \\ T(yx|xy) = \frac{trans[x,y]}{count[xy]}, & \text{transposition} \end{cases}$$

Candidate List and Probability

- For each position p, we check if t can results from a real word c and a single-error mis-spelling.
- If so, we approximate $P(t|c)$ by

$$P(t|c) = \begin{cases} D(t_{p-1}|t_{p-1}c_p), & \text{deletion of } c_p \\ I(t_{p-1}t_p|t_{p-1}), & \text{insertion after } c_{p-1} \\ S(t_p|c_p), & \text{substitution of } c_p \text{ by } t_p \\ T(t_{p+1}t_p|t_pt_{p+1}), & \text{transposition at } p : p+1 \end{cases}$$

- Fig. 5.25 provides the details.

Markov Chain

- A **Markov chain** is an FSA with the following properties
  - state transition has a weight, representing transition probability
  - input sequence = state sequence
- So a Markov chain is a kind of **weighted finite-state automata (WFSA)**, specified by
  - a state set $Q = \{q_1, \ldots, q_N\}$
  - transition probabilities, $A = a_{ij}$, with

$$\sum_j a_{ij} = 1, \; \forall\, i$$

  - q_0, q_F, the start and end states

Markov Chain Assumptions

- A **Markov model** is based on the following assumptions
  - **conditional independence assumption**

$$Pr(s_t|s_{t-1}, s_{t-2}, \ldots, s_1) = Pr(s_t|s_{t-1})$$

  - **time-invariance assumption**

$$Pr(s_t = q_j|s_{t-1} = q_i) = a_{ij} \quad \forall t$$

- We sometimes single out $a_{0i} = \pi_i$, to represent the **initial probability**.

An Example

- A starter W of New York Yankees
  - state set: $Q = \{1 = \text{A}, 2 = \text{B}, 3 = \text{T}\}$.
  - uniform initial probability
  - transition probability

$$A = \begin{bmatrix} 0.6 & 0.1 & 0.3 \\ 0.4 & 0.3 & 0.3 \\ 0.5 & 0.3 & 0.2 \end{bmatrix}$$

- What is the probability that W leaves with leads for the first 10 games he started, i.e. state = AAAAAAAAAA?

Hidden States and Observations

- The performance of a starting pitcher is best measured by the state **when he leaves the field**, A/B/T.
- However, the win/loss result **when the game is over** is often easier to track.
- Suppose we have no data of the states but have the win/loss record.
  - A/B/T are called the *hidden states*
  - the win/loss records are called the *observations*
- It is clear hear that an observation depend on the corresponding hidden state.
- The above model constitutes a **hidden Markov model**.

HMM

- An HMM is specified by
  - a state set $Q = \{q_1, \ldots, q_N\}$
  - an **observation sequence** $o_1, \ldots, o_T$.
  - transition probabilities

$$A = a_{ij} = Pr(s_{t+1} = q_j | s_t = q_i), \quad \sum_j a_{ij} = 1, \ \forall\ i$$

  - **observation likelihood**,

$$B = b_i(o) = p(o | s = q_i).$$

  - q_0, q_F, the start and end states
- Again, in some literature $a_{0.}$ is denoted by π.

Example: Ice Cream

- Fig. 6.3 gives an HMM which models **the weather** and **the number of ice creams eaten by Jason**.
- The weather is **hidden**, the number is observed.
- With this example, we introduce the **fundamental problems in HMM**.
  - **likelihood computation**
  - **decoding**
  - **parameter estimation**

Likelihood Computation

- **likelihood computation**: computing the likelihood of a given observation
  - What is the probability that Jason eats $(2, 2, 2)$ ice creams in three days?
- We introduce the **forward algorithm** and the **backward algorithm** for this problem.

Decoding

- **decoding**: finding the most-likely state sequence for a given observation
  - Suppose in three days, Jason eat $(2, 2, 2)$ ice creams. What is the most likely weather sequence?
- We introduce the **Viterbi algorithm** for this problem.

Parameter Estimation

- **parameter estimation**: estimating probability parameters A, B from data observation, say a long observation sequence or a set of sequences
- Note the state sequence is unknown.
- We introduce the **EM algorithm** for this problem.
  - The EM algorithm is based on the computation of the posterior probabilities of (hidden) states, which can be computed by the **forward-backward** algorithm.

Forward Probability

- We have the initial probability π, the transition probability a_{ij}, and the observation likelihood $b_i(o)$.
- Define the **forward probability**, denote by $\alpha_j(t)$, to be the **joint probability** that the hidden state at time t is q_j and the observation sequence from 1 to t is $o_1, \ldots, o_t$,

$$\alpha_j(t) = p(o_1, \ldots, o_t, s_t = q_j).$$

Forward Algorithm

- Recall q_0 and q_F are the start and end states. The **forward algorithm** evaluates $\alpha_j(t)$ progressively.
  - initialization
$$\alpha_j(1) = a_{0j} b_j(o_1)$$
  - recursion
$$\alpha_j(t) = \sum_{i=1}^{N} \alpha_i(t-1) a_{ij} b_j(o_t)$$
  - termination
$$\alpha_F(T) = \sum_{i=1}^{N} \alpha_i(T) a_{iF}$$

Backward Probability

- Similarly, we can define the **backward probability**, denote by $\beta_i(t)$, to be

$$\beta_i(t) = p(o_{t+1}, \ldots, o_T | s_t = q_i).$$

- In other words, it is the **conditional probability** of the observation sequence from $t+1$ to T is $o_{t+1}, \ldots, o_T$, given $s_t = q_i$.

Backward Algorithm

- The **backward algorithm** evaluates $\beta_i(t)$ in the reverse direction.
  - initialization

$$\beta_i(T) = a_{iF}$$

  - recursion

$$\beta_i(t) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_j(t+1)$$

  - termination

$$\beta_0(1) = \sum_{j=1}^{N} a_{0j} b_j(o_1) \beta_j(1)$$

Likelihood

- If we multiply $\alpha_j(t)$ and $\beta_j(t)$ for the same state q_j at any time t, we get

$$\alpha_j(t)\beta_j(t) = p(o_1, \ldots, o_t, s_t = q_j)p(o_{t+1}, \ldots, o_T | s_t = q_j)$$
$$= p(o_1, \ldots, o_T, s_t = q_j) = p(O, s_t = q_j).$$

- The likelihood of the observation sequence is

$$p(O) = \sum_j p(O, s_t = q_j) = \sum_j \alpha_j(t)\beta_j(t).$$

- Alternatively, the likelihood is also given by the forward probability or backward probability alone,

$$p(O) = \alpha_F(T) = \beta_0(1).$$

Viterbi Probability

- We define the **Viterbi probability**, $v_j(t)$, to be the maximum joint probability of $o_1, \ldots, o_t$ and a state sequence $s_1, \ldots, s_t$ with $s_t = q_j$.
- That is,

$$v_j(t) = \max_{i_1,\ldots,i_{t-1}} p(s_1 = q_{i_1}, \ldots, s_{t-1} = q_{i_{t-1}}, s_t = q_j, o_1, \ldots, o_t).$$

- Note the relation

$$\begin{aligned} v_j(t) &= \max_{i_1,\ldots,i_{t-1}} p(s_1 = q_{i_1}, \ldots, s_{t-1} = q_{i_{t-1}}, s_t = q_j, o_{1:t}) \\ &= \max_{i_{t-1}} \left[\max_{i_1,\ldots,i_{t-2}} p(\ldots, s_{t-2} = q_{i_{t-2}}, s_{t-1} = q_{i_{t-1}}, o_{1:t-1}) \right] a_{i_{t-1}j} b_j(o_t) \\ &= \max_i v_i(t-1) a_{ij} b_j(o_t) \end{aligned}$$

Viterbi Algorithm

- $v_j(t)$ can be computed by the **Viterbi algorithm**.
  - initialization

$$v_j(1) = p(s_1 = q_j, o_1) = a_{0j} b_j(o_1)$$

  - recursion

$$v_j(t) = \max_i v_i(t-1) a_{ij} b_j(o_t)$$

  - termination

$$v_F(T) = \max_i v_i(T) a_{iF}$$

- Note its resemblance to the forward algorithm.

$\mathcal{Q}$ Function

- We need the following auxiliary function

$$\mathcal{Q}(\lambda, \lambda_0) = E[\log p(S, O|\lambda)] = \sum_{\mathbf{s}} p(S = \mathbf{s}|O, \lambda_0) \log p(S = \mathbf{s}, O|\lambda)$$

  - λ_0: current set of model parameters
  - O: observed data
  - S: hidden variable (a sequence of states)
- The function $\mathcal{Q}$ is maximized with respect to the unknown λ to yield a new set of model parameters.

Data Likelihood and $\mathcal{Q}$

- Note

$$
\begin{aligned}
&\mathcal{Q}(\lambda, \lambda_0) - \mathcal{Q}(\lambda_0, \lambda_0) \\
&= \sum_{\mathbf{s}} \left[p(S = \mathbf{s}|O, \lambda_0) \log p(S = \mathbf{s}, O|\lambda) - p(S = \mathbf{s}|O, \lambda_0) \log p(S = \mathbf{s}, O|\lambda_0) \right] \\
&= \sum_{\mathbf{s}} p(S = \mathbf{s}|O, \lambda_0) \left[\log p(O|\lambda) + \log p(S = \mathbf{s}|O, \lambda) \right] \\
&\quad - \sum_{\mathbf{s}} p(S = \mathbf{s}|O, \lambda_0) \left[\log p(O|\lambda_0) + \log p(S = \mathbf{s}|O, \lambda_0) \right] \\
&= \log p(O|\lambda) - \log p(O|\lambda_0) - \sum_{\mathbf{s}} p(S = \mathbf{s}|O, \lambda_0) \log \frac{p(S = \mathbf{s}|O, \lambda_0)}{p(S = \mathbf{s}|O, \lambda)} \\
&= \log p(O|\lambda) - \log p(O|\lambda_0) - D(p_0||p_\lambda).
\end{aligned}
$$

- $D(p_0||p_\lambda)$ is called the relative entropy between p_0 and p_λ. It is always non-negative.

EM Algorithm

- Suppose $\mathcal{Q}(\lambda, \lambda_0)$ is maximized by λ^*, meaning

$$\mathcal{Q}(\lambda^*, \lambda_0) \geq \mathcal{Q}(\lambda_0, \lambda_0).$$

- We have

$$\begin{aligned} &\log p(O|\lambda^*) - \log p(O|\lambda_0) \\ &= \mathcal{Q}(\lambda^*, \lambda_0) - \mathcal{Q}(\lambda_0, \lambda_0) + D(p_0||p_\lambda) \\ &\geq \mathcal{Q}(\lambda^*, \lambda_0) - \mathcal{Q}(\lambda_0, \lambda_0) \\ &\geq 0. \end{aligned}$$

- That is, maximizing the expected log likelihood $\mathcal{Q}$ increases the data likelihood.

$\mathcal{Q}$ Function for HMM

- From the independence assumption of HMM, we have

$$\begin{aligned} p(S, O) &= p(S)p(O|S) \\ &= p(s_1) \prod_{t=2}^{T} p(s_t|s_{t-1}) \prod_{t=1}^{T} p(o_t|s_t). \end{aligned}$$

- Taking the logarithm as required in the $\mathcal{Q}$ function

$$\log p(S, O) = \log p(s_1) + \sum_{t=2}^{T} \log p(s_t|s_{t-1}) + \sum_{t=1}^{T} \log p(o_t|s_t)$$

Putting Together

$$
\begin{aligned}
\mathcal{Q}(\lambda, \lambda_0) &= \sum_{\mathbf{s}} p(\mathbf{s}|O, \lambda_0) \log p(\mathbf{s}, O|\lambda) \\
&= \sum_{\mathbf{s}} p(\mathbf{s}|O, \lambda_0) \log p(s_1|\lambda) + \sum_{\mathbf{s}} p(\mathbf{s}|O, \lambda_0) \sum_{t=1}^{T} \log p(o_t|s_t, \lambda) \\
&\quad + \sum_{\mathbf{s}} p(\mathbf{s}|O, \lambda_0) \sum_{t=2}^{T} \log p(s_t|s_{t-1}, \lambda) \\
&= \sum_{i=1}^{N} p(s_1 = q_i|O, \lambda_0) \log \pi_i + \sum_{t=1}^{T} \sum_{i=1}^{N} p(s_t = q_i|O, \lambda_0) \log b_i(o_t) \\
&\quad + \sum_{t=2}^{T} \sum_{i=1}^{N} \sum_{j=1}^{N} p(s_{t-1} = q_i, s_t = q_j|O, \lambda_0) \log a_{ij}
\end{aligned}
$$

Posterior Probability

- posterior probability (given O) of $s_t = q_i$

$$\gamma_i(t) = p(s_t = q_i | O) = \frac{\alpha_i(t)\beta_i(t)}{\sum_j \alpha_j(t)\beta_j(t)}.$$

- joint probability of $s_t = q_i, s_{t+1} = q_j$ and O

$$p(s_t = i, s_{t+1} = j, O) = \alpha_i(t) a_{ij} b_j(o_{t+1}) \beta_j(t+1).$$

- posterior probability of $\xi_{ij}(t) = p(s_t = q_i, s_{t+1} = q_j | O)$

$$\xi_{ij}(t) = \frac{p(s_t = q_i, s_{t+1} = q_j, O)}{p(O)} = \frac{\alpha_i(t) a_{ij} b_j(o_{t+1}) \beta_j(t+1)}{\alpha_F(T)}.$$

State Occupancy

- If I is a Bernoulli random variable, then

$$E(I) = p(I = 1).$$

- Thus, given the observation sequence O, the expected number of occupancy in state q_i is

$$\sum_{t=1}^{T} \gamma_i(t)$$

State Transition

- For the same reason, $\xi_{ij}(t)$ is expected number of transition from state q_i to state q_j, at time t.
- For the entire sequence, the expected number of transitions from state q_i to q_j is

$$\sum_{t=1}^{T} \xi_{ij}(t).$$

Parameters in Markov Chains

- initial probability

$$\pi_i = \gamma_i(1).$$

- transition probability

$$a_{ij} = \frac{\sum_t \xi_{ij}(t)}{\sum_t \gamma_i(t)}$$

Parameters in Gaussians

- We note without proof that the parameters of Gaussian observation likelihood are updated by
  - mean

$$\mu_i = \frac{\sum_t \gamma_i(t) o_t}{\sum_t \gamma_i(t)}$$

  - covariance

$$\sigma_i^2 = \frac{\sum_t \gamma_i(t)(o_t - \mu_i)(o_t - \mu_i)'}{\sum_t \gamma_i(t)}$$

Maximum Entropy Models

- This is another commonly used probability model in speech and language processing.
- We will cover this subject in the following order
  - **linear regression**
  - **logistic regression**
  - **classification based on logistic regression**
  - **maximum entropy models**

Regression and Classification

- We discuss two distinguishable classes of problems.
  - When the output is real-valued, the task is called **regression**.
  - When the output is discrete, the task is called **classification**.
- The input of a regression or classification system is often called **features**, representing important **quantities** or **attributes** of raw input data.

Linear Regression

- A **linear regression** is the special case of regression where the input-output function is **linear**.
- In the simplest case, there is a single input x, and the output y is related to x by

$$\hat{y} = b + wx.$$

  w is called **weight** and b is called **bias**.
- With multiple features, $f_1, \ldots, f_n$, we have

$$\hat{y} = b + \sum_{i=1}^{n} w_i f_i$$

Vectorial Form

- Define the extended vectors of feature and weight

$$\mathbf{f} = (f_0, \ldots, f_n) = (1, \ldots, f_n)$$
$$\mathbf{w} = (w_0, \ldots, w_n) = (b, \ldots, w_n).$$

- The linear regression can be written as a **dot product**

$$\hat{y} = \mathbf{w} \cdot \mathbf{f}.$$

Learning Weights

- Suppose we have a data set of M instances $\{(y, \mathbf{f})\}_i^M$.
- Suppose the **learning criterion** is to minimize the **sum of squared errors**

$$\sum_{i=1}^{M} (\hat{y}_i - y_i)^2.$$

- It can be shown that the optimal weight is given by

$$\mathbf{w}^* = (F^T F)^{-1} F^T \mathbf{y},$$

where F is the matrix whose ith column is $\mathbf{f}_i$, and $\mathbf{y}$ is the vector whose ith component is y_i.

Classification

- Next we turn to a classification problem.
- Given feature $\mathbf{f}$, we want to decide the class of the data x represented by $\mathbf{f}$.
- In a classification problem, we are often content with a **probability distribution** of different classes given $\mathbf{f}$.
- However, $\mathbf{w} \cdot \mathbf{f}$ cannot be a probability since the range is unbounded.

Odds

- Consider a two-class classification problem

$$\begin{cases} y = 1, & \text{if } x \text{ is in class } 1 \\ y = 0, & \text{otherwise} \end{cases}$$

- The **odds** of $y = 1$, for a given x, is defined by

$$\frac{p(y = 1|x)}{p(y = 0|x)} = \frac{p(y = 1|x)}{1 - p(y = 1|x)}.$$

- The range of an odds is from 0 to ∞ ...

Logit Function

- We can apply a logarithm to an odds. The range of **log odds** is from $-\infty$ to ∞.
- Then we can equate log odds to a linear regression of features

$$\log \frac{p(y=1|x)}{1-p(y=1|x)} = \mathbf{w} \cdot \mathbf{f}.$$

- The lhs function is called the **logit function**

$$\text{logit}(s) = \log \frac{s}{1-s}.$$

Logistic Function

- Solving $\text{logit}(p) = \mathbf{w} \cdot \mathbf{f}$ for p, we get

$$\text{logit}(p) = \log \frac{p}{1-p} = \mathbf{w} \cdot \mathbf{f} \;\Rightarrow\; p = \frac{e^{\mathbf{w} \cdot \mathbf{f}}}{1 + e^{\mathbf{w} \cdot \mathbf{f}}} = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{f}}}$$

- Define the **logistic function**

$$\text{logistic}(t) = \frac{1}{1 + e^{-t}}.$$

- We have

$$\text{logit}(p) = t \;\Rightarrow\; p = \text{logistic}(t).$$

Logistic Regression

- Putting all together, the probability is given by

$$p(y = 1|x) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{f}}} = \text{logistic}(\mathbf{w} \cdot \mathbf{f}).$$

- It is called a **logistic regression**.

Decision Boundary

- Given x, we decide it class based on comparing the probabilities $p(y=1|x)$ and $p(y=0|x)$.
- From the logistic function, we can see that

$$\begin{cases} \mathbf{w}\cdot\mathbf{f} > 0 \;\Rightarrow\; p(y=1|x) \;>\; \frac{1}{2} \;>\; p(y=0|x), \\ \mathbf{w}\cdot\mathbf{f} < 0 \;\Rightarrow\; p(y=1|x) \;<\; \frac{1}{2} \;<\; p(y=0|x). \end{cases}$$

- So the decision boundary is a hyperplane in the feature space, decided by $\mathbf{w}$.

A Different Perspective

- In logistic regression, the posterior probability of sample x being in class 1 is given by

$$p(y=1|x) = \frac{e^{\mathbf{w}\cdot\mathbf{f}}}{1+e^{\mathbf{w}\cdot\mathbf{f}}} \;\propto\; e^{\mathbf{w_1}\cdot\mathbf{f}}$$
$$p(y=0|x) = e^{0} \;\propto\; e^{\mathbf{w_0}\cdot\mathbf{f}}.$$

- The log probability of a class, say c, is proportional to a class-dependent weighted sum of features

$$\mathbf{w_c}\cdot\mathbf{f} = \sum_i w_{ci} f_i.$$

$\mathbf{w_c}$ represents the weights for class c.

Multiple Classes

- We can use the same idea towards multi-class classification problem.
- Specifically, we can assume

$$p(y = c|x) \propto e^{\mathbf{w_c} \cdot \mathbf{f}} = \frac{1}{Z} e^{\mathbf{w_c} \cdot \mathbf{f}}.$$

- Z is a normalization constant to make total probability 1, so

$$\sum_c p(y = c|x) = 1 \Rightarrow Z = \sum_c e^{\mathbf{w_c} \cdot \mathbf{f}}.$$

For example, the two-class problem has $Z = 1 + e^{\mathbf{w} \cdot \mathbf{f}}$.

Stochastic Modeling

- In the problem of **stochastic modeling**, we want to construct a model to best characterize a **random process**.
- We have at our disposal some **sample data** from this random process.
  - baseball managers
  - stock brokers
  - nlp researchers
- We construct a model to “fit” the sample data. Put in another way, we exploit data to decide the model.

Initial Model

- Suppose we want to model the translation of the word **in** in English to one of 5 candidates in French

dans, en, à, au cours de, pendant

- The only knowledge we know about the probabilities is that they sum to 1.
- Without further information, it is reasonable (and *wise*) to assume **uniform** probability for each candidate, i.e.,

$$p(\textit{dans}) = \cdots = \frac{1}{5}.$$

Knowledge From Sample

- Suppose we have a sample of translation results by a human expert translator on the translation of **in**.
- Suppose in this sample, the expert chooses *dans* or *en* 30% of the time.
- If we think this is an important piece of information, we may require that our model has this *feature*.
- We now have two *constraints* for the probabilities, i.e.,

$$p(\textit{dans}) + p(\textit{en}) = p_1 + p_2 = \frac{3}{10}$$

$$p_1 + p_2 + p_3 + p_4 + p_5 = 1.$$

More Knowledge From Sample

- Suppose in this sample, we further find out that *dans* or *à* are chosen 50% of the time.
- We may require the model to satisfy

$$p(\textit{dans}) + p(\textit{en}) = p_1 + p_2 = \frac{3}{10}$$
$$p(\textit{dans}) + p(\textit{à}) = p_1 + p_3 = \frac{1}{2}$$
$$p_1 + p_2 + p_3 + p_4 + p_5 = 1.$$

- Now it is not clear how p_i can be “uniformized”. We need a mathematical measure for “uniformness”.

Entropy

- Suppose Y is a discrete **random variable** with probability

$$p(y) = Pr(Y = y),\ y \in \mathcal{Y}.$$

- The **entropy** of Y is defined by

$$H(Y) = -\sum_{y \in \mathcal{Y}} p(y) \log p(y).$$

- $H(Y)$ is really a function of $p(y)$, not Y.
- If the base-2 logarithm is used, the unit of entropy is **bit**.
- For example, the entropy of a fair coin toss is 1 bit.

Entropy Bounds

- Assuming $\mathcal{Y} = \{y_1, \ldots, y_K\}$, we can write the entropy $H(Y)$ as a function of the variables $(p_1, p_2, \ldots, p_K)$

$$H(Y) = -\sum_{i=1}^{K} p_i \log p_i.$$

- The probabilities must satisfy

$$p_i \geq 0, \quad \sum_i p_i = 1.$$

- It can be proved that

$$0 \leq H(Y) \leq \log K.$$

Uniformness and Entropy

- It can be shown that the maximum entropy is achieved when the distribution is uniform

$$p_i = \frac{1}{K} = \text{constant.}$$

- So we have a mathematical measure of “uniformness”: the larger the entropy, the more uniform the distribution.

Conditional Entropy

- The entropy of a conditional probability is defined in the same way as entropy

$$H(Y|X=x) = -\sum_{y} p(y|x) \log p(y|x).$$

- The **conditional entropy** of a random variable Y given a random variable X is defined by

$$\begin{aligned} H(Y|X) &= -\sum_{x,y} p(x,y) \log p(y|x) \\ &= -\sum_{x,y} p(x) p(y|x) \log p(y|x). \end{aligned}$$

Training Data

- Let the training sample data set be

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}.$$

- Here x_i is the ith phrase containing **in** and y_i is the translation of that **in**.
- x represents the **contextual information** relevant to the classification.

Empirical Distribution

- The **empirical distribution** of (x, y) is based on the sample set

$$\tilde{p}(x, y) = \frac{1}{N} \sum_{i=1}^{N} I_{(x_i, y_i) = (x, y)}.$$

- I_s is the **indicator function** for statement s

$$I_s = \begin{cases} 1, & S \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

- Note $\tilde{p}(x, y)$ is just the relative frequency.

Statistics of Data

- We first decide certain statistics of the sample data set are important in constructing the stochastic model.
- For example, we have employed two such statistics, the number of instances of *ans* or *en*, as well as the number of instances of *dans* or *à*.
- Certain contextual information x may be important, such as
  - if **April** is the word following **in**, then the translation of **in** is *en* 90% of the time.

Feature Functions

- We can define a binary-valued function

$$f(x,y) = \begin{cases} 1, & y = \textit{en} \text{ and } \textbf{April} \text{ follows } \textbf{in} \text{ in } x \\ 0, & \text{otherwise.} \end{cases}$$

- $f(x,y)$ is called a **feature function** or just **feature**.
- The expected value of $f(x,y)$ with respect to $\tilde{p}(x,y)$ is just the relative frequency in the sample set.

Expected Values

- $f(x, y)$ is an **indicator function**.
- The expected value of f with respect to the empirical distribution is

$$\tilde{p}(f) = \sum_{x,y} \tilde{p}(x, y) f(x, y).$$

- The expected value of f with respect to a model $p(y|x)$ is

$$p(f) = \sum_{x,y} \tilde{p}(x) p(y|x) f(x, y).$$

Constraint Equations

- For feature f, we request the expected value of f with respect to the model and to the empirical distribution to **agree**, i.e.,

$$\tilde{p}(f) = p(f).$$

- This equation is called a **constraint equation**, or simply **constraint**.
- We can see there is one constraint for each feature.

Feasible Set

- Let P be the set of all probability distributions $p(y|x)$.
- Suppose we are given n features f_i. We would like our model p to be in a set C

$$C = \{p \in P \mid p(f_i) = \tilde{p}(f_i),\ i = 1, \ldots, n\}.$$

- Let C_i be the set of distributions satisfying the ith constraint. Then

$$C = C_1 \cap C_2 \cap \cdots \cap C_n,$$

- C is called the **feasible set**. It is often *infinite*.

Maximum Entropy Principle

- Among all feasible models, we wan to decide an optimal one.
- The **conditional entropy** of a model $p = p(y|x)$ is

$$H(p) = -\sum_{x,y} \tilde{p}(x) p(y|x) \log p(y|x).$$

$\tilde{p}(x)$ is the empirical distribution of x.

- The **maximum entropy principle** is to choose the model with the maximum conditional entropy

$$p_* = \arg\max_{p \in C} H(p).$$

Lagrange Multipliers

- For each feature f_i, we introduce a **Lagrange multiplier** λ_i. Then we define the **Lagrangian**

$$\Lambda(p,\lambda) = H(p) + \sum_i \lambda_i (p(f_i) - \tilde{p}(f_i)).$$

- The unconstrained optimization of $\Lambda(p,\lambda)$ and the constrained optimization of $H(p)$ have the same optimal **value**, i.e.,

$$\max_\lambda \max_p \Lambda(p,\lambda) = \max_{p \in C} H(p).$$

Dual Function

- For a given λ,
  - let p_λ be the model p that $\Lambda(p, \lambda)$ achieves maximum, i.e.,

$$p_\lambda = \arg\max_{p \in P} \ \Lambda(p, \lambda)$$

  - let $\Psi(\lambda)$ be the value

$$\Psi(\lambda) = \Lambda(p_\lambda, \lambda)$$

- $\Psi(\lambda)$ is called the **dual function**.

Kuhn-Tucker Theorem

- The **dual problem** is defined to be the unconstrained optimization problem

$$\lambda^* = \arg\max_{\lambda} \; \Psi(\lambda).$$

- Kuhn-Tucker
  - The dual problem has the same value as the primal problem.
  - Suppose λ^* is the solution of the dual problem, then the solution of the primal problem is

$$p_* = p_{\lambda^*}.$$

Solving p_λ

- Equating the derivative of Λ with respect to $p(y|x)$ to 0, one can show that, for a given λ,

$$p_\lambda(y|x) \propto e^{\sum_i \lambda_i f_i(x,y)} = \frac{1}{Z_\lambda(x)} e^{\sum_i \lambda_i f_i(x,y)}$$

- The normalizing Z is given by

$$Z_\lambda(x) = \sum_y e^{\sum_i \lambda_i f_i(x,y)}$$

- Note that this is a log-linear model.