Regular Expression Matching

A well-defined matching language
(used in Emacs, grep, sed, perl, libg++ etc).

- `.` matches any character (except newline)
- `X*` zero or more `X`, `X+` one or more `X`
- `[A-Z]*` zero or more capital letters
- `\\(fish\\|chip\\)` matches fish or chip
- `-?[0-9]+` matches an integer
- `[A-Z][a-z]+` matches a capitalized alphabetic string

This is exactly the libg++ Regex language
Festival offers an interface through
`(string-matches ATOM REGEX)`

Decision Trees (CART)

A method for choosing a class, or number based on features (of a stream item).
– (QUESTION YESTREE NOTREE)

```
((name is "the")
 ((n.num_syllables > 1)
 ((0))
 ((1)))
 ((name in ("a" "an"))
 ((0))
 ((1))))
```

Operators: is, matches, in, `<`, `>`, `=`
`wagon` can automatically build trees from data

building CART trees

Requires set of samples with features:
P f0 f1 f2 f3 f4 ...
P may be class or float
`wagon` finds best (local) possible question that minimises the distortion in each partition.
CART is ideal for

- ☐ mixed features
- ☐ no clear general principle
- ☐ a reasonable amount of data
- ☐ can't (sort of) deal with very large classes

Homographs

Words with same written form but different pronunciation

- □ Different part of speech: project
- □ Semantic difference: bass, tear
- □ Proper names: Nice, Begin, Said
- □ Roman Numerals: Chapter II, James II
- □ Numbers: years, days, quantifiers, phone numbers
- □ Some symbols: 5-3, high/low, usr/local

How common are they?

– Numbers: email 2.57% novels 0.00013%
– POS/hgs: WSJ 7.6%

Homograph disambiguation (Yarowsky)

Same tokens with different pronunciation

- ☐ Identify particular class of homographs
  - – e.g. numbers, roman numerals, “St”.
- ☐ Find instances in large db with context
- ☐ Train decision mechanism to find most distinguished feature

Homograph disambiguation: example

Roman numerals: as cardinals, ordinals, letter

Henry V: Part I Act II Scene XI: Mr X is I believe, V I Lenin, and not Charles I.

- ☐ Extract examples with context features
- ☐ Label examples with correct class:
  – king, number, letter
- ☐ Build decision tree (CART) to predict class

Features

```
class: n(umber) l(etter) c(entury) t(imes)
rex rex_names section_name num_digits p.num_digits n.num_digits
pp.cap p.cap n.cap nn.cap


n II 0 0 0 11 7 2 3 7 0 0 1 1
n III 0 0 0 3 4 3 3 5 0 0 1 1
c VII 1 0 0 4 9 3 3 3 1 1 0 0
n V 0 0 1 3 4 1 1 2 0 1 0 1
n VII 0 0 1 2 4 3 1 2 0 1 0 1
...
```

```
     ((p.lisp_tok_rex_names is 0)
      ((lisp_num_digits is 5)
       ((number))
       ((lisp_num_digits is 4)
        ((number))
        ((nn.lisp_num_digits is 13)
...
                  ((nn.lisp_num_digits is 2)
                   ((letter))
                   ((n.cap is 0) ((letter)) ((number)))))))))
...
```

Homograph disambiguation: example

Example data features:
– surrounding words, capitalization, “king-like”, “section-like”

class	ord	let	card	times	total	correct	percent
ord	133	0	15	0	148	133/148	89.865
let	3	40	9	0	52	40/52	76.923
card	7	6	533	0	546	533/546	97.619
times	0	2	1	1	4	1/4	25.000

707/750 94.267% correct

Homograph disambiguation

But it still fails on many obscure (?) cases

- ☐ William B. Gates III.
- ☐ Meet Joe Black II.
- ☐ The madness of King George III.
- ☐ He’s a nice chap. I met him last year.

How many homographs are there?

Very few actually, ...

axes bass Begin bathing bathed bow Celtic close cretan Dr executor jan jean lead live lives Nice No Reading row St Said sat sewer sun tear us wed wind windier windiest windy winds winding windily wound Number num/num num-num Roman_numerals

Plus *many* POS homographs

Letter Sequences?

Letter sequences as words, letters, abbreviations (or mix)

- □ IBM, CIA, PCMCIA, PhD
- □ NASA, NATO, RAM
- □ etc, Pitts, SqH, Pitts. Int. Air.
- □ CDROM, DRAM, WinNT, SunOS, awblack

Require lexicon, or "pronouncability oracle".
Hueristic: captialization and vowels

Alphabetic tag sub-classification

NSW tag **t** for alphabetic observations **o**
NATO: ASWD, **PCMCIA**: LSEQ, **frplc**: EXPN

- ☐
$$p(\mathbf{t}|\mathbf{o}) = \frac{p_t(\mathbf{o}|\mathbf{t})p(\mathbf{t})}{p(\mathbf{o})}$$
  where $\mathbf{t} \in [ASWD, LSEQ, EXPN]$.
- ☐ $p_t(\mathbf{o}|\mathbf{t})$ estimated by a letter trigram model
$$p_t(\mathbf{o}|\mathbf{t}) = \prod_{i=1}^{N} p(l_i|l_{i-1}, l_{i-2}),$$
- ☐ $p(\mathbf{t})$ prior from data or uniform
- ☐ normalized by
$$p(\mathbf{o}) = \sum_t p_t(\mathbf{o}|\mathbf{t})p(\mathbf{t})$$

Token to Words

Exanding identifyied token to words

- ☐ numbers+type = word list
- ☐ homographs+type = words
- ☐ symbols broken down and pronounced
- ☐ unknown words: as word or letter sequence

```
(define (token_to_words token name)
  (cond
   ((string-matches name "[0-9]+'s") ;; e.g. 1950's
     (item.set_feat token "token_pos" "year")
     (append
      (builtin_english_token_to_words token (string-before name "'s"))
      (list '((name "'s")(pos nnp)))))
   ((string-matches name "[0-9]+-[0-9]+")
    ;; e.g.  12-14
    ;; split into two numbers
    ;; identify type of one number (ordinal/cardinal)
    ;; expand woth ''to'' between them
   )
   ....
   (t
     ;; just a simply word
     (builtin_english_token_towords token name))))
```

Example token rule

for “$120 million”

```
(define (token_to_words token name)
 (cond
  ((and (string-matches name "\\$[0-9,]+\\(\\.[0-9]+\\)?")
        (string-matches (item.feat token "n.name")
                        ".*illion.?"))
   (append
    (english_token_to_words token (string-after name "$"))
    (list
     (item.feat token "n.name"))))
  ((and (string-matches (item.feat token "p.name")
                        "\\$[0-9,]+\\(\\.[0-9]+\\)?")
        (string-matches name ".*illion.?"))
   (list "dollars"))
  (t
   (english_token_to_words token name))))
```

Festival front-end

- ☐ Tokenize string of chars
- ☐ Chunk into utterance sized chunks
- ☐ Identify token types (homographs, numbers etc)
- ☐ Expand tokens with token to word rules

Mistakes in Festival

Some identifiable mistakes

- ☐ CMU, LTI as a word
- ☐ Breaking words down, “ttools”, “MCIMail”
- ☐ Homographs, EST, MT
- ☐ Roman numerals XXXV
- ☐ missing punctuation clues for phrasing
- ☐ missing layout cues for phrasing

Exercises for 20th March

1. * Add a token to word rule, to say money values with two places after the point properly.
2. * Add a token to word rule to say numbers in dates as ordinals (first, second, third, etc.) rather than cardinals. Also add a token to word rule to dates of the form ”11/06/97” in their full form rather than number slash number ...
3. Build a text mode for reading Latex, HTML, syslog messages, machine use summary or such like.

Hints for 20th March

You will need to add a new definition for `token_to_words` by convention save the existing one and call that for things that don't match what you are looking for. Thus your file will look something like

```
(set! previous_token_to_words token_to_words)

(define (token_to_words token name)
   (cond
     ;; condition to recognize money tokens
     ;; return list of words
    (t
     (previous_token_to_words token name))))
```

The actual condition and return list of words is similar to the treatment of email addresses described above.
The regular expression for money is probably

```
"$[0-9]+\\.[0-9][0-9]"
```

Hints for 20th March

Take the above and add to it. The rule is probably: if the token is a two digit number and the succeeding token's name is a month name (or abbreviation of a month name) then return the word, first, second etc.
You will need a new function to relate the digits to the pronunciation. If you don't know how to do that in Scheme the following will work

```
(define (num-to-ordinal num)
"Returns the ordinal in words for num (up to 40)."
(cdr
 (cdr
  (assoc (parse-number num)
   '((1 first) (2 second) (3 third) ...
     (39 thirty-ninth) (40 fortieth))))))
```

Once you decide on the condition, remember that you need to return a **list** of words.