CS4248 Cheatsheet by Ming Ann

REGular EXpressions

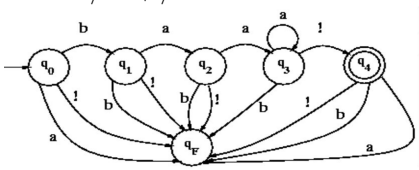
Patterns used to specify text strings to search texts
Operator precedence: () > *+?{} > ^$ > |

RE	String matched
*	0 or more instances of the prev. string
+	1 or more instances of the prev. string
?	exactly 0 or 1 instance of the prev. string
{n}	n instances of the prev. string
{n, m}	from n to m instances of the prev. string
{n, }	at least n instances of the prev. string
.	one instance of any character
*	an asterisk "*"
\.	a period "."
\?	a question mark
\n	a newline
\t	a tab
/[A-Z]/	an uppercase letter
/[a-z]/	a lowercase letter
/[0-9]/	a single digit
[^A-Z]	not an uppercase letter
[e^]	either "e" or "^"
a^b	the pattern "a^b"
\d	any digit
\D	any non-digit
\s	white space (space, tab)
\S	non-whitespace

`re.search('i.', 'uninteresting')` returns a `Match` object with `span=(2,4)` (index 4 exclusive) and `match='in'`

Finite State Automata

For RE: /baa+!/



	Input		
State	b	a	!
0	1	0	0
1	0	2	0
2	0	3	0
3	0	3	4
4:	0	0	0

Byte Pair Encoding

Add a special '_' end-of-word symbol before space
Initialise vocab to the set of all individual characters
Repeat:
- Choose the two most frequently adjacent symbols in the training corpus
- Add a new merged symbol to the vocab
- Replace every adjacent pair with the new merged symbol

On the test data, run each merge learned: greedily, in the order they were learned

Spelling Errors

Types of errors: insertion, deletion, substitution, transposition
Use **Bayesian Classification** to choose the most probable class, given the observation o

$$\hat{c} = \arg\max_{c \in C} \{P(o|c) \cdot P(c)\}$$

Minimum Edit Distance

Cost of operation:
- Insertion & Deletion: 1
- Substitution: 2 if the two characters are different, otherwise 0

Noisy Channel Mode

Estimating P(o|c)

$$P(o|c) = \begin{cases} \frac{sub[m,l]}{C(l)} & \text{if substitution} \\ \frac{trans[k,l]}{C(kl)} & \text{if transposition} \\ \frac{ins[l,m]}{C(l)} & \text{if insertion} \\ \frac{sub[k,l]}{C(kl)} & \text{if deletion} \end{cases}$$

n-gram

Maximum Likelihood Estimate (MLE)

$$P(w_n|w_1, w_2, ...w_{n-1}) = \frac{C(w_n|w_1, w_2, ..., w_n)}{C(w_n|w_1, w_2, ..., w_{n-1})}$$

Perplexity

A better LM better predicts the test data, assigning a high probability to the test data, and has lower perplexity.

$$PP = m(w_1, ..., w_n)^{-\frac{1}{n}} = \frac{1}{\sqrt[n]{\prod_{k=1}^{n} P(w_k|w_{k-1})}} \text{ if bigram LM}$$

Smoothing

$$P(c) = \frac{C(c) + \lambda}{N + B \cdot \lambda}$$

λ : small positive number
C(c): Frequency of c in training corpus
N: Total no. of c's
B: No. of distinct c's

Add-One Smoothing for Bigram

Smoothed bigram count: $C^*(w_0 w)$

$$P(w|w_0) = \frac{C(w_0 w) + 1}{C(w_0) + V} = \frac{C^*(w_0 w)}{C(w_0)}$$

$$\text{discount } d = \frac{C^*(w_0 w)}{C(w_0 w)} = \frac{C(w_0 w) + 1}{C(w_0 w)} \times \frac{C(w_0)}{C(w_0) + V}$$

Witten-Bell Smoothing for Bigram

$T \equiv T(w_0)$ is the no. of *distinct word types* following w_0
$T(w_0)$ is the no. of *seen bigram types* following w_0
$Z(w_0)$ is the no. of *unseen bigram types* following w_0

$$T(w_0) + Z(w_0) = V$$

$$P(w|w_0) = \frac{C(w_0 w)}{C(w_0) + T(w_0)}, \text{ if } C(w_0 w) > 0$$

$$P(w|w_0) = \frac{T(w_0)}{Z(w_0)(C(w_0) + T(w_0))}, \text{ if } C(w_0 w) = 0$$

Interpolation

Combining lower and higher order n-grams which have different strengths and weaknesses
- Higher-order utilize more context, but have sparser counts
- Lower-order consider limited context, but more robust counts

ie. trigrams, where $\sum_i \lambda_i = 1$

$$\hat{P}(w_0|w_{-2}w_{-1}) = \lambda_1 P(w_0|w_{-2}w_{-1}) + \lambda_2 P(w_0|w_{-1}) + \lambda_3 P(w_0)$$

Backoff

If no examples of a particular trigram $w_{-2}w_{-1}w_0$ to compute $P(w_0|w_{-2}w_{-1})$, estimate by using bigram probability $P(w_0|w_{-1})$

Backoff for Bigram

$$\hat{P}(w_0|w_{-1}) = \begin{cases} \tilde{P}(w_0|w_{-1}) & \text{if } C(w_{-1}w_0) > 0 \\ \alpha(w_{-1}) \cdot \tilde{P}(w_0) & \text{if } C(w_{-1}w_0) = 0 \end{cases}$$

$\tilde{P}(w_0|w_{-1})$, $\tilde{P}(w_0)$: discounted probability

Kneser-Ney Smoothing for Bigram

Most effective smoothing method, absolute discounting for seen bigrams

$$\frac{C(w_{-1}w_0) - D}{C(w_{-1})}, 0 \leq D \leq 1$$

Entropy

Entropy rate (per-word entropy):

$$\frac{1}{n} H(W_1^n) = -\frac{1}{n} \sum_{W_1^n \in L} p(W_1^n) \log_2 p(W_1^n)$$

Entropy of a Language:

$$H(L) = \lim_{n \to \infty} -\frac{1}{n} \log_2 p(w_1, ..., w_n)$$

(Stochastic) POS Tagging

Hidden Markov Model assumption

$$\hat{T} = \arg\max_{t_1,...,t_T} \left\{ (\prod_{i=1}^{T} P(t_i|t_{i-1}) \cdot P(w_i|t_i)) \cdot P(< /s > |t_T) \right\}$$

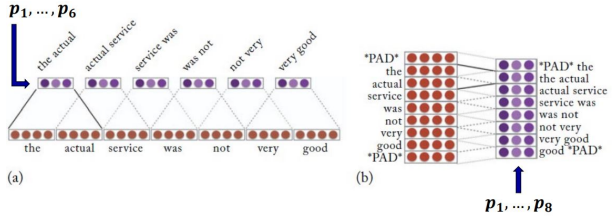
If w_i is an unknown word:

$P(w_i|t_i) = P(\text{unknownword}|t_i) \cdot P(\text{capital}|t_i) \cdot P(\text{ending/hyph}|t_i)$

Convolutional Neural Network

Restricted to mostly local patterns (n-grams)
- Apply a nonlinear function over each k-word sliding window in the sentence
- l filters $U = (u_1, ..., u_l)$ are applied to produce an l-dimensional vector for each sliding window of k words
- A pooling (ie. max or avg) operation combines the vectors into a l-dimensional vector, which is fed further into the NN

The filter function learns to identify informative k-grams.
Pooling focuses on the most important aspects of a sentence, regardless of location.
- p_i is a collection of l values representing the i-th window
- m: number of p_i vectors ($i = 1, ...m$)
- (a) narrow convolution (no padding)
  $m = n - (k-1) = n - k + 1$
- (b) wide convolution (pad $k-1$ words to each side)
  $m = n + (k-1) = n + k - 1$



Word Embeddings
Cosine Similarity

$$\text{sim}_{\cos}(u,v) = \frac{\boldsymbol{u}\cdot\boldsymbol{v}}{||\boldsymbol{u}||_2||\boldsymbol{v}||_2} = \frac{\sum_i \boldsymbol{u}_{[i]}\cdot\sum_i \boldsymbol{v}_{[i]}}{\sqrt{\sum_i(\boldsymbol{u}_{[i]})^2}\sqrt{\sum_i(\boldsymbol{v}_{[i]})^2}}$$

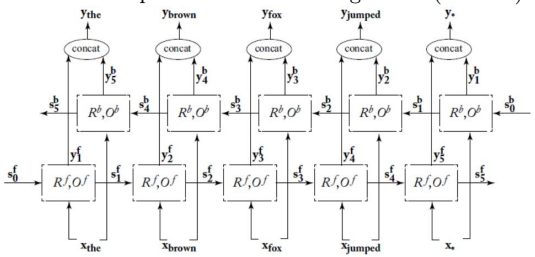
Recurrent Neural Network

Sensitive to the order of words in a sequence. Consider infinite window and zoom in on informative sequential patterns in the window

$$RNN^*(x_{1:n} : s_0) = y_{1:n}$$
$$s_i = R(s_{i-1}, x_i)$$
$$y_i = O(s_i)$$

- the functions R and O are the same across the sequence positions
- the last y_n encodes the entire input sequence and can be used for further prediction

The sequential nature of RNNs hinders parallel computation. Deals with long-range dependencies, and the large no. of training steps needed.
RNN as Acceptor: Make a prediction based on the final state y_n, ie. predict whether a sentence conveys a positive or negative sentiment
RNN as Encoder: y_n treated as an encoding of $x_1, ..., x_n$, ie. a document summariser first encodes an input document, then uses it with other features to select the sentences to include in the summary
RNN as Transducer: Produces an output $\hat{t}_i$ after each input x_i is read, ie. a sequence tagger that predicts the tag of each word
Bidirectional RNN
Use both the previous and following words (context)



Sequence2Sequence Model

Attention weights $\alpha^j_{[i]}$ can be added, to vary the importance of words in a sentence and reveal which parts i of the source sentence the decoder finds relevant at output step j.

Transformer Networks

Self-attention: compute the similarity between a vector in a sequence and every vector in the sequence. Relevance of a vector to another vector ("attention")
Positional Encoding: Order of words is important in language processing. Positional embedding is a vector of sine and cosine functions for each position. Positional embedding added to word embedding for each word
Bidirectionl **E**ncoder **R**epresentations from **T**ransformers: Pre-trained contextualized representations from unlabeled text, which consider both left and right context. Fine-tune using labeled training data from downstream NLP tasks.

Context-Free Grammar

$G = (N, \sum, P, S)$
- A set of non-terminal symbols N
- A set of terminal symbols $\sum$
- A set of productions P, each of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (\sum \cup N)$
- A designated start symbol $S \in N$

Strong equivalence: Two grammars are strongly equivalent if they generate the same set of strings **and** they assign the same phrase structure to each string.
Weak equivalence: Generate the same set of strings but do not assign the same phrase structure to each string

Chomsky Normal Form (CNF)

Each production of the grammar is either of the form $A \to BC$ or $A \to a$, ie. either 2 non-terminal symbols or 1 terminal symbol on the RHS

1. Construct a new set of productions P', by including all non-unit productions. For non-terminals symbols A and B, if $A \to A1 \to A2 \to ... \to B$, for each non-unit production $B \to \alpha$, add a new production $A \to \alpha$ to P'
2. For each production in P' of the form $A \to X_1X_2...X_m$ where $m \geq 2$. If X_i is a terminal symbol a, introduce a new non-terminal symbol C and a production $C \to a$. Then replace X_i by c.
3. For each production in P' of the form $A \to B1\ B2\ B3$, replace this production by

$$A \to B1\ D1 \text{ and } D1 \to B2\ B3$$

Typed dependency parse: Links in parse tree are labeled from a fixed list of grammatical relations
Untyped dependency parse: Links are unlabeled
Phrase Structure Parse to Untyped Dependency Parse:

- Annotate the lexical head of each node in the phrase structure parse
- In the dependency parse, make the head of each non-head-child depend on the head of the head-child

CKY Algorithm
Bottom-up DP algo which requires the CFG to be in CNF
Returns all possible parses, and chooses the most probable parse (statistical parsing)
Probabilistic CFG
In addition to properties of standard CFG, include
- A function D that assigns a probability to each rule in P

Parse Tree Scoring Function

$$s_{\text{best}}(i,j) = \begin{cases} \max_l[s(i,j,l)] & \text{if } j-i=1 \\ \max_l[s(i,j,l)] + \\ \max_k[s_{\text{best}}(i,k) + s_{\text{best}}(k,j)] & \text{if } j-i>1 \end{cases}$$

Evaluating Parsers
A constituent is correct if there is a constituent in the treebank's parse that *spans the same words* with the same non-terminal symbol

$$\text{labeled recall } R = \frac{\#\text{correct constituents in parser's parse of } S}{\#\text{constituents in treebank's parse of } S}$$

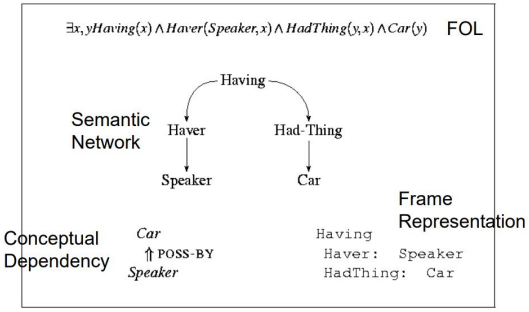
$$\text{labeled precision } P = \frac{\#\text{correct constituents in parser's parse of } S}{\#\text{constituents in parser's parse of } S}$$

$$\text{labeled } F1 = \frac{2RP}{R+P}$$

Meaning Representation

- Verifiability: able to match the representation to the sentence
- Unambiguous Representations: meaning representation free from ambiguity
- Canonical Form: different sentences with the same meaning are assigned the same representation
- Inference and Variables: valid inferences not explicitly stated can be drawn
- Expressiveness: adequately express and represent any meaning

Sentence: I have a car.



Reference Resolution

Two referring expressions *corefer* when they are used to refer to the same entity. The first of the two expressions is the *antecedent* and the second is the *anaphor*. A reference to an entity that has been previously introduced into the discourse is known as *anaphora*.
Coreference resolution: finding expressions that refer to the same entity
Coreference chain: set of coreferring expressions

Neural Coreference Resolution

Consider all possible spans. Learn to rank antecedent spans. Aggressive pruning of search space.
Task: Assign to each span i an antecedent $y_i \in y(i)$
$y_i = \epsilon$ (dummy antecedent) represents two possible scenarios:

- Span i is not an entity mention; or
- Span i is an entity mention but it is not coreferent with any previous span

Pairwise score $s(i,j)$ for a coreference link between span i and span j, where j is an antecedent of anaphor i

$$s(i,j) = \begin{cases} 0 & j = \epsilon \\ s_m(i) + s_m(j) + s_a(i,j) & j \neq \epsilon \end{cases}$$

Pruning:
To maintain efficiency

- Consider spans of up to L words
- Keep up to λT spans with the highest mention scores $s_m(i)$
- Consider up to K antecedents for each span
- Still maintains a high recall rate