midterm

01 Intro

○ NLP's goal ○ Common Applications ○ Interpretation of garden-path sentences

Regex	Match (single characters)	Example Patterns Matched
`/[^A-Z]/`	not an upper case letter	"Oyfn pripetchik"
`/[^Ss]/`	neither 'S' nor 's'	"I have no exquisite reason for't"
`/[^.]/`	not a period	"our resident Djinn"
`/[e^]/`	either 'e' or '^'	"look up ^ now"
`/a^b/`	the pattern 'a^b'	"look up a^ b now"

/a*/ means "any string of zero or more as"
a+ means "one or more occurrences of the immediately preceding character or regular expression
ˆ matches the start of a line. The pattern /ˆThe/ matches the word The only at the start of a line. $ end of line
\b matches a word boundary, and \B matches a non-boundary
The pattern /cat|dog/ matches either the string cat or the string dog.() means and `a\.{24}z/` will match a followed by 24 dots followed by z (but not a followed by 23 or 25 dots followed by a z)

Regex	Expansion	Match	First Matches
`\d`	`[0-9]`	any digit	Party␣of␣5
`\D`	`[^0-9]`	any non-digit	Blue␣moon
`\w`	`[a-zA-Z0-9_]`	any alphanumeric/underscore	Daiyu
`\W`	`[^\w]`	a non-alphanumeric	!!!!
`\s`	`[␣\r\t\n\f]`	whitespace (space, tab)	in Concord
`\S`	`[^\s]`	Non-whitespace	in␣Concord

{n} exactly n occurrences of the previous char or expression
{n,m} from n to m occurrences of the previous char or expression
{n,} at least n occurrences of the previous char or expression
{,m} up to m occurrences of the previous char or expression \n a newline \t a tab
In that case we use a non-capturing group, which is specified by putting the special commands ?: after the open parenthesis, in the form (?: pattern).
The operator `(?! pattern)` only returns true if a pattern does not match, but again is zero-width and doesn't advance the cursor.
Herdan's Law (Herdan, 1960) or Heaps' Law (Heaps, 1978) $|V| = kN^{\beta}$
Two events, A and B, are independent iff: P(A, B) = P(A)P(B)
word tokens - an individual word instance. (a list) word types - distinct words. (a set)
V - "vocabulary" |V| - vocabulary size (number of types) N - number of tokens
Corpus - a natural language dataset
○ Tokenization ■ Parts of speech ■ word tokenizer ■ word piece tokenization (conceptually)

BPE

– more data-driven; no predefined words or rules 如何合并的
– allow for subwords (e.g. "unlikeliest" -> "un", "like", "liest") – better for unseen words or capturing semantics of parts of words.

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

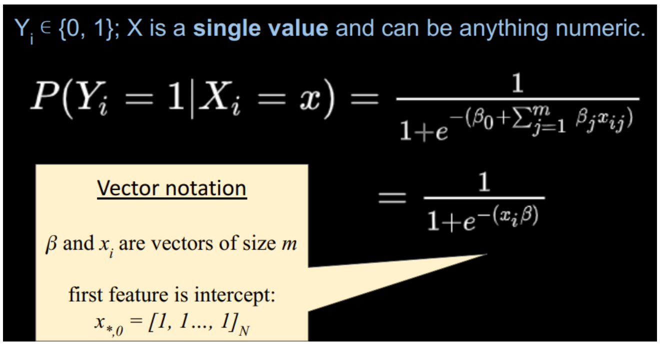
2 3 Maximum Entropy Classifier Supervised Machine Learning

Build a “model” that can estimate P(Y=1|X=?) Y=1 if target is a verb
使用逻辑回归的函数作为decision boundary $L(\beta|X,Y)=\Pi_{i=1}^{n}P(Y_i=1|x_i)^{y_i}(1-P(Y_i=1|x_i))^{1-y_i}$
best fit: whatever maximizes the likelihood function $J(\beta)=-\frac{1}{N}\sum_{i=1}^{N}(y_i\log p(x_i)+(1-y_i)\log(1-p)(x_i))$

Logistic Regression on a single feature (x)

$Y_i \in 0,1$; X is a single value and can be anything numeric.

$$
\begin{aligned}
P(Y_i=1 \mid X_i=x) &= \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}} \\
&= \frac{1}{1+e^{-\left(\beta_0+\sum_{j=1}^{m}\beta_j x_{ij}\right)}}
\end{aligned}
$$



We’re still learning a linear separating hyperplane, but fitting it to a logit outcome.应用逻辑函数将线性预测结果转换成对数几率，得到分类概率，不改变决策边界
$\boldsymbol{\beta}$ ≈ weight ≈ coefficient ≈ parameters ≈ θ Logistic Regression ≈ Maximum Entropy Classifier loss function ≈ cost function

```
z = torch.matmul(x, beta)#将输入向量x与参数向量beta进行矩阵乘法运算，得到一个线性预测结果z。这里假设x是一个输入特征向量，beta是一个参数向量，二者的维度兼容。
yhat = nn.functional.relu(z)#对线性预测结果z进行ReLU（修正线性单元）激活函数的操作，得到预测输出yhat。ReLU函数将所有负值变为零，并保持非负值不变。
loss = nn.MSELoss(yhat, torch.Tensor(y))#计算预测输出yhat与目标值y之间的均方误差（Mean Squared Error）。这里假设y是目标值，通过将其转换为`torch.Tensor`类型，与预测输出yhat进行比较计算损失值。

sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.mean(-torch.sum(y*torch.log(y_pred))
#training loop:
for i in range(epochs):
        model.train()
        sgd.zero_grad()
        #forward pass:
        ypred = model(X)
        loss = loss_func(ypred, y)
        #backward: /(applies gradient descent)
        loss.backward()
        sgd.step()
        if i % 20 == 0:
                print(" epoch: %d, loss: %.5f" %(i, loss.item()))
```

`torch.nn.BECLoss()` binary cross entropy = `loss(x, y) = -[y * log(x) + (1 - y) * log(1 - x)]`

Machine Learning: How to setup data

○ Likelihood function to Loss function row of features; e.g.
➔ number of capital letters
➔ whether “I” was mentioned or not
➔ k features indicating whether k words were mentioned or not
Feature extraction: one-hot, multi-hot representations
Multi-hot Encoding
● Each word gets an index in the vector ● 1 if present; 0 if not
Multiple One-hot encodings for one observation
(1) word before; (2) word after (3) percent capitals

L1 Regularization - “The Lasso” Zeros out features by adding values that keep from perfectly fitting the data.
set betas that maximize penalized L $L\left(\beta_0, \beta_1, \ldots, \beta_k \mid X, Y\right)=\prod_{i=1}^n p\left(x_i\right)^{y_i}\left(1-p\left(x_i\right)\right)^{1-y_i}-\frac{1}{C} \sum_{j=1}^m\left|\beta_j\right|$
C is the hyperparameter, set betas that maximize penalized L
L2 Regularization - “Ridge” Shrinks features by adding values that keep from perfectly fitting the data.
$L\left(\beta_0, \beta_1, \ldots, \beta_k \mid X, Y\right)=\prod_{i=1}^n p(x_i)^{y_i}\left(1-p\left(x_i\right)\right)^{1-y_i}-\frac{1}{C} \sum_{j=1}^m \beta_j^2$ Sometimes writing as $||\beta_j||_2^2$
After doing them, we can avoid overfitting.
Overfitting is when a model is more accurate than the held-out data. --True
discriminative learning and generating learning

2. Which of the following are true about discriminative learning as compared to generative: (8 points)
   - A generative model can be used to generate features supposing it is of some class c.
   - A discriminative model is often used to directly create new data.
   - A model which assigns classes c to a document d by seeking to estimate P(c|d) is an example of a generative modeling approach.
   - Logistic regression is an example of a discriminative modeling approach.
   - A model which assigns classes c to a document d by seeking to compute a likelihood term P(d|c) and a prior P(c) is an example of a discriminative modeling approach.

answer: A D

3. Recall the L2 regularized likelihood function below.

$$L(\beta_0, \beta_1, ..., \beta_k|X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1-p(x_i))^{1-y_i} - \frac{1}{C}\sum_{j=1}^{m}\beta_j^2$$

When using L2 regularization for logistic regression with penalty parameter c, which of the following statements are true?
   - Each beta is guaranteed to shrink
   - As c increases, betas on average increase
   - As c increases, betas on average decrease
   - Logistic regression will find the value of c which maximizes the likelihood function over your data

Answer: BD

4. Suppose you have the following vocabulary of words in alphabetical order for a one-hot encoding scheme.

```
[
  "I",        # 0
  "NLP",      # 1
  "dislike",  # 2
  "doing",    # 3
  "like",     # 4
  "skipping"  # 5
]
```

```
[
  "I",        # 0
  "NLP",      # 1
  "dislike",  # 2
  "doing",    # 3
  "like",     # 4
  "skipping"  # 5
]
```

You are going to produce a feature vector that concatenates the one-hot encoding of the word before and after a target word.

[1-hot of word before target] + [1-hot of word after target]

Given the sentence "I like doing NLP" select the indices below that should contain a 1 (be "hot") in the specified input feature vector if the target word is "doing".

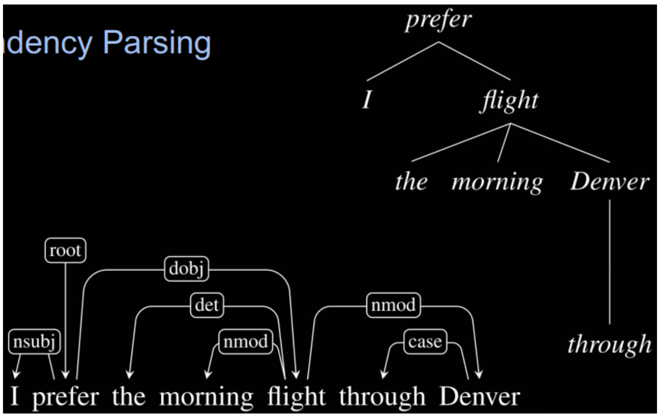
0 # I || 1 # NLP || 2 # dislike || 3 # doing || 4 # like || 5 # skipping || 6 # I || 7 # NLP || 8 # dislike || 9 # doing || 10 # like || 11# skipping

4 Dependency Parsing

○ Relations (core universal dependency relations) ○ head and dependent
○ Transition-based dependency parsing ○ Projectivity ○ Idea of semantic roles and verbal predicates

- 关系树和依存关系 和算法

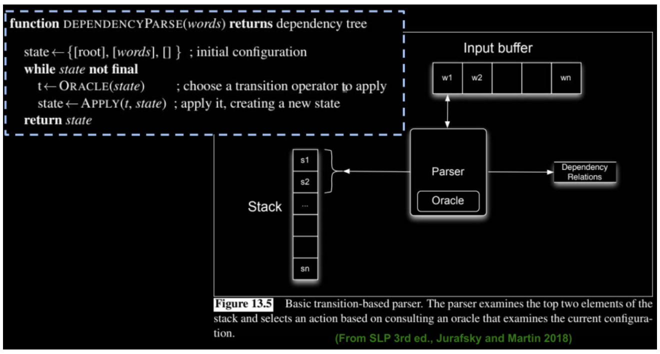
Dependency Parsing



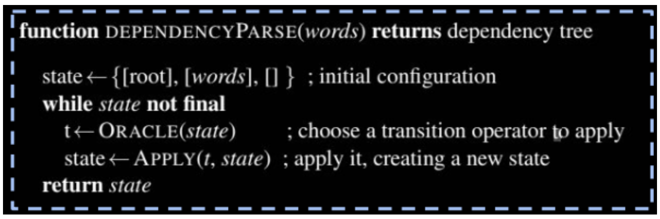
Transition-based (Shift-Reduce algorithm)
Restrictions:

1. Single designated ROOT with no incoming arcs
2. Every vertex only has one head (parent, governor); i.e. only one incoming arc
3. unique path from ROOT to every vertex

Transition-based Dependency Parsing



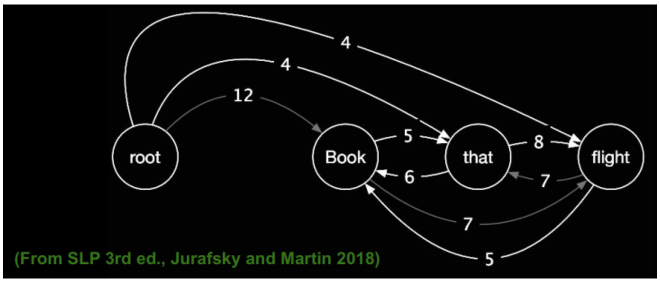
● S: stack, initialized with “ROOT” ● B: input buffer, initialized with tokens (w1, w2, ….) of sentence
● A: set of dependency arcs, initialized empty ● T: Actions, given wi (next token in stack)



From Syntax to Semantics
● We've already seen words have many meanings. ○ Context is key
● Verbs can been seen as functions (predicates) that take arguments. ○ Syntactic arguments fulfill semantic roles
● Words have implicit syntactic relationships with each other in given sentences.
○ Dependency Parsing: each word has one head
○ Easily constructed through 3 actions of shift-reduce parsing.
Takeaway: There is an interplay between word meaning and sentence structure!

Graph-based Approaches

Search through all possible trees and pick best.



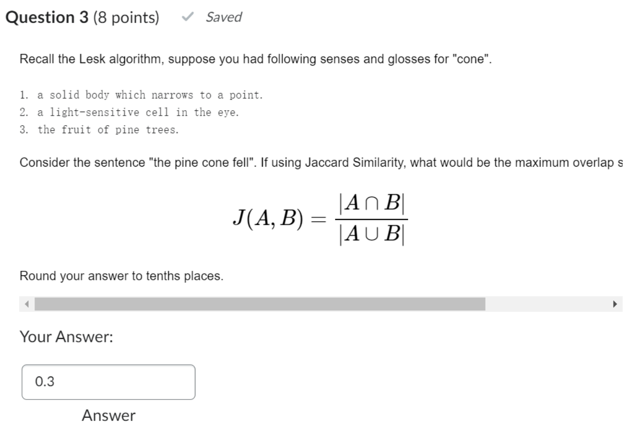
For each word, pick the most likely head. Then check if still a fully-connected tree, and adjust.
Complex and slow but leads to state of the art. Now done with neural models.

Semantic Roles

Roles are restricted to nouns, but signalled through the verb and other parts of speech.

5 Lexical and Vector Semantics

○ terminology (lemmas, homonymy, etc…) ○ different types of word sense disambiguation
○ Lesk algorithm ○ distributional hypothesis ○ concept of vector semantics
○ word2vec - skip-gram model ○ topic modeling – LDA



Question 3 (8 points) ✓ *Saved*

Recall the Lesk algorithm, suppose you had following senses and glosses for "cone".

```
1. a solid body which narrows to a point.
2. a light-sensitive cell in the eye.
3. the fruit of pine trees.
```

Consider the sentence "the pine cone fell". If using Jaccard Similarity, what would be the maximum overlap s

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Round your answer to tenths places.

Your Answer:

0.3

Answer

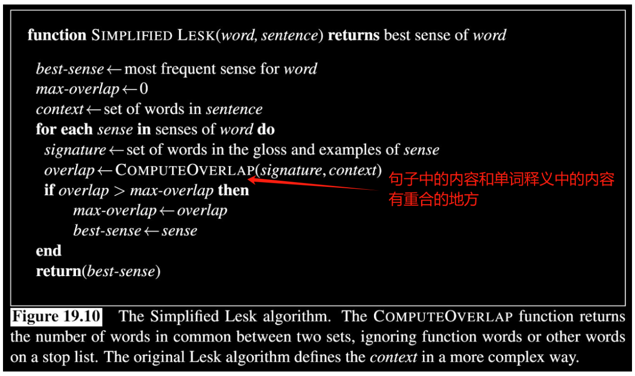
2/7 gloss the dictionary definition of a word.
part of speech: a category to which a word is assigned in accordance with its syntactic functions.
distributional hypothesis: a word's meaning is defined by all the
different contexts it appears in (i.e. how it is "distributed" in natural language).
word sense: a discrete representation of one aspect of the meaning of a word.
lemma: the canonical form, dictionary form, or citation form of a set of word forms.

● Lexical Ambiguity (why word sense disambiguation)

Word Sense Disambiguation (WSD) `f (sent_tokens, (target_index, lemma, POS)) -> word_sense`
Distributional hypothesis -- A word’s meaning is defined by all the different contexts it appears in (i.e. how it is “distributed” in natural language).
Firth, 1957: “You shall know a word by the company it keeps”

Approaches to WSD

1. Bag of words for context --E.g. multi-hot for any word in a defined “context”.
2. Surrounding window with positions --E.g. one-hot per position relative to word).
3. Lesk algorithm --E.g. compare context to sense definitions.



function SIMPLIFIED LESK(*word, sentence*) **returns** best sense of *word*

best-sense ← most frequent sense for *word*
max-overlap ← 0
context ← set of words in *sentence*
for each *sense* **in** senses of *word* **do**
  signature ← set of words in the gloss and examples of *sense*
  overlap ← COMPUTEOVERLAP(*signature, context*) 句子中的内容和单词释义中的内容有重合的地方
  if *overlap* > *max-overlap* **then**
    max-overlap ← *overlap*
    best-sense ← *sense*
end
return(*best-sense*)

Figure 19.10 The Simplified Lesk algorithm. The COMPUTEOVERLAP function returns the number of words in common between two sets, ignoring function words or other words on a stop list. The original Lesk algorithm defines the *context* in a more complex way.

有问题在于

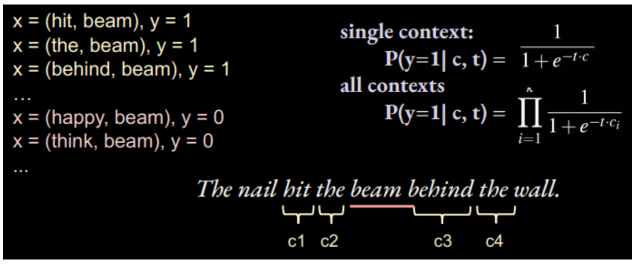
- 单词释义更长的单词有可能和他的使用场景有更多overlap
- 有可能对应了意思，但是并没有overlap（相同的）单词

4. Selectors -- other target words that appear with same context
   E.g. counts for any selector.找到同义字 Sets of selectors tend to vary extensively by word sense
5. Contextual Embeddings --E.g. real valued vectors that “encode” the context (TBD).

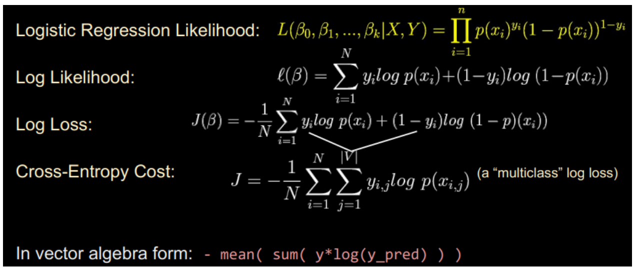
● Word Vectors

word2vec Principal: Predict missing word. Similar to classification where y = context and x = word.
2 Versions of Context:

1. Continuous bag of words (CBOW): Predict word from context
2. Skip-Grams (SG): predict context words from target
   1.Treat the target word and a neighboring context word as positive examples.
   2.Randomly sample other words in the lexicon to get negative samples
   3.Use logistic regression to train a classifier to distinguish those two cases --assume dim * |vocab| weights for each of c and t, initialized to random values (e.g. dim = 50 or dim = 300)
   4.Use the weights as the embeddings

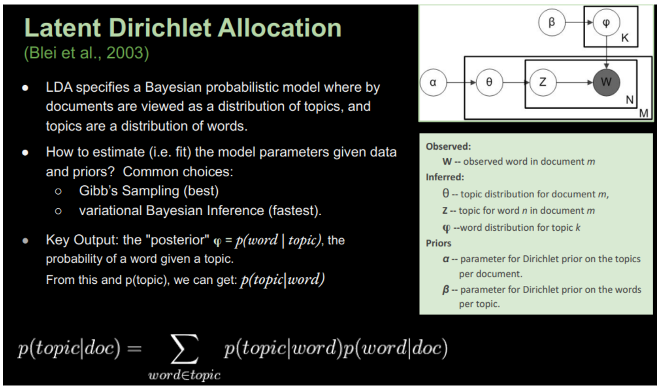


单一上下文 简化计算 固定窗口
Intuition: t·c is a measure of similarity: But, it is not a probability! To make it one, apply logistic activation: $\sigma(z) = 1/(1+e^{-z})$
c context 里面出现的词 有可能是single的有可能是all，t target(如何相乘)
Maximizes similarity of (c, t) in positive data (y = 1) Minimizes similarity of (c, t) in negative data (y = 0)



● Topic Modeling

Latent Dirichlet Allocation (LDA)



Common applications:
● Open vocabulary content analysis: Describing the latent semantic categories of words or phrases present across a set of documents
● Embeddings for predictive task: for all topics, use p(topic|document) as score. Feed to predictive model (e.g. classifier).

- PCA-Based Embeddings -- try to represent with only p’ dimensions --also known as "Latent Semantic Analysis
- SVD-Based Embeddings -- Dimensionality Reduction - PCA

6 Introduction to Language Modeling

○ 2 task versions and their equivalence
○ applications ○ chain rule, markov assumption
○ unigram, bigram LMs -- assigning a probability to sequences of words.
Version 1: Compute P(w1, w2, w3, w4, w5) = P(W)
:probability of a sequence of words --even the Web isn’t large enough to enable good estimates of most phrases
Version 2: Compute P(w5| w1, w2, w3, w4)= P(wn| w1, w2, …, wn-1)
:probability of a next word given history
P(B|A) = P(B, A) / P(A) A,B) = P(A)P(B|A) ⇔ P(A)P(B|A) = P(B,A) P(A, B, C) = P(A)P(B|A)P(C| A, B)
The Chain Rule : P(X1, X2,…, Xn) = P(X1)P(X2|X1)P(X3|X1, X2)...P(Xn|X1, ..., Xn-1)
$P(X_1, X_2, \ldots, X_n) = \Pi_{i=1}^{n} P(X_i|X_1, X_2, \ldots, X_{i-1})$
Markov Assumption : P(Xn| X1…, Xn-1) ≈ P(Xn| Xn-k, …, Xn-1) where k < n
Unigram Model: k = 0; Bigram Moel: k=1;
Perplexity Inverse of probability (i.e. minimize perplexity = maximize likelihood) and (weighted) average branching metric for scoring how well learned model works on test. (an intrinsic evaluation)
MLE An intuitive way to **estimate probabilities** is called maximum likelihood estimation or MLE.
Practical Consideration:
● Use log probability for assessing perplexity to keep numbers reasonable and save computation. (uses addition rather than multiplication)
● Use Out-of-vocabulary (OOV)
Choose minimum frequency or total vocabulary size and mark as `<OOV>`
● Sentence start and end: `<s> this is a sentence </s>`
Advantage: models word probability at beginning or end.