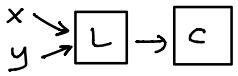
Lecture # 8: Text Classification

The goal ⇒ take single observation, extract some useful features and thereby classify the observation into one set of discrete classes.

$X = (x_1, x_2, \cdots, x_n)$
$y = (y_1, y_2, \cdots, y_n)$

$X \rightarrow \boxed{C} \rightarrow y$

A learner L infers from the classifier



⇒ N-gram models can be used for classification, but for many tasks sequential relationships b/w words are largely irrelevant.

Naive Bayes → family of probabilistic algorithms

↳ For a document d, out of all classes $c \in C$ the classifier returns the class $\hat{c}$ which has max. posterior prob. given the document.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

Breaking down $P(x|y)$ ⇒ $P(x|y) = \frac{P(y|x)P(x)}{P(y)}$

So,

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

Since the P(d) or prob. of document is always same, we can drop it.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

Modelling P(d|c)

Define a set of features that help classify the doc.

$d = (f_1, f_2, \cdots, f_n)$

$P(d|c) = P(f_1, f_2, \cdots, f_n|c)$

Since data is sparse, make the naive assumption: features are conditionally independent given the class c.

$$P(f_1, f_2, \cdots, f_n|c) = P(f_1|c)P(f_2|c), \cdots, P(f_n|c)$$

⇒ Prob. of a word occuring depends only on a class.

So, we can say:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i=1}^{n} P(f_i|c)$$

$P(c)$ → prob. of class before observing any data

$P(f_i|c)$ → prob. of f_i in class c

Estimating Class Priors

$$\hat{P}(c) = \frac{N_c}{N}$$

⇒ P(c) is estimated with MLE

N_c → total number of training documents in class c
N → total training documents

In other words, $\hat{P}(c)$ is simply proportion of training documents belonging to class c.

Consider the example:

	the	your	model	cash	Money	class	account	orderz	spam?
doc 1	12	3	1	0	0	2	0	0	-
doc 2	10	4	0	4	0	0	2	0	+
doc 3	25	4	0	0	0	1	1	0	-
doc 4	14	2	0	1	3	0	1	1	+
doc 5	17	5	0	2	0	0	1	1	+

$\hat{P}(\text{spam}+) = 3/5$
$\hat{P}(\text{spam}-) = 2/5$

⇒ $P(f_i|c)$ can be estimated with simple smoothing.

$$\hat{P}(f_i|c) = \frac{\text{count}(f_i, c) + \alpha}{\sum_{f \in F}(\text{count}(f, c) + \alpha)}$$

count (f_i, c) → # of times f_i occurs in class c.
F → set of possible features
α → smoothing parameter, optimized on held out data.

$$\hat{P}(\text{your}|+) = \frac{(4+2+5+\alpha)}{\left(\begin{matrix}\text{all words in}\\ \text{+ class}\end{matrix}\right) + \alpha|F|} = \frac{11+\alpha}{(68+\alpha|F|)}$$

$$\hat{P}(\text{your}|-) = \frac{7+\alpha}{49+\alpha|F|} \qquad \hat{P}(\text{orderz}|+) = \frac{2+\alpha}{(68+\alpha|F|)}$$

Let a test document be: get your cash and your orderz → d

$$P(+|d) \propto P(+)\prod_{i=1}^{n} P(f_i|+)$$

Let $\hat{P}(f_i|+) = \frac{\alpha}{68+\alpha|F|}$ for all words (features) not shown

$$= P(+) \cdot \frac{\alpha}{68+\alpha|F|} \cdot \frac{11+\alpha}{(68+\alpha|F|)} \cdot \frac{7+\alpha}{(68+\alpha|F|)} \cdot \frac{2+\alpha}{(68+\alpha|F|)} \cdot \frac{11+\alpha}{(68+\alpha|F|)} \cdot \frac{2+\alpha}{(68+\alpha|F|)}$$

→ Repeat the same for P(−|d)
→ Choose the class (+ or −) with high score

Since multiplying large numbers of small prob is problematic.
Actual implementation of Naive Bayes uses costs.
→ costs are −ve log prob.
→ we can sum them and avoid underflow
→ look for lowest cost overall

So, the equation becomes:

$$\hat{c} = \underset{c \in C}{\operatorname{argmin}} \left(-\log(P(c)) + \sum_{i=1}^{n} -\log(P(f_i|c)) \right)$$

⇒ Choosing features can be tricky.

Advantages of Naive Bayes

- Very easy to implement, fast to train & test.
- Doesn't require much data.

Problems

- Naive Bayes is naive.
- Features are not usually independent

→ Adding multiple feature types leads to stronger correlations.

Maximum Entropy Model (MaxEnt)

→ Commonly known as Multinomial Logistic Regression.
↓
more than 2 classes.

Also called log-linear model, one-layer neural network, single neuron classifier.

Can be defined as: $\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$

Unlike NB, we do not apply Bayes Rule, we simply model P(c|d) directly.

Example: Given a webpage document, what topic does it belong to?

$\vec{x}$ → are the words in document
c → latent class

In MaxEnt, $\vec{x}$ is the observed data. features are functions on both observations $\vec{x}$ and class c.

c	class
1	Travel
2	Sport
3	Finance

MaxEnt is used when dependent variable in question is nominal, and for which there are more than two categories.

If there are three classes, features come in groups of three.
For example,

f_1 : contains(ski) & $c = 1$
f_2 : contains(ski) & $c = 2$
f_3 : contains(ski) & $c = 3$

- training docs from c1 with ski in it will have f_1 active.

Each feature f_i has a real-valued weight w_i (learned in training).

Classification with MaxEnt

Choose the class that has highest probability.

$$P(c|\vec{x}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\vec{x}, c)\right)$$

where $Z = \sum_{c'} \exp(\sum_i w_i f_i(\vec{x}, c))$ → normalization constant
w.f → is just a dot product
⇒ $P(c|x)$ is just a monotonic fn of this dot product

So, we will end up choosing the class for which $\vec{w} \cdot \vec{f}$ is highest.

Consider the following features and weights.

f_1 :	contains('ski') & $c = 1$	$w_1 = 1.2$
f_2 :	contains('ski') & $c = 2$	$w_2 = 2.3$
f_3 :	contains('ski') & $c = 3$	$w_3 = -0.5$
f_4 :	link_to('expedia.com') & $c = 1$	$w_4 = 4.6$
f_5 :	link_to('expedia.com') & $c = 2$	$w_5 = -0.2$
f_6 :	link_to('expedia.com') & $c = 3$	$w_6 = 0.5$
f_7 :	num_links & $c = 1$	$w_7 = 0.0$
f_8 :	num_links & $c = 2$	$w_8 = 0.2$
f_9 :	num_links & $c = 3$	$w_9 = -0.1$

Suppose test document contains ski and 6 outgoing links.

Travel: $\sum_i w_i f_i(\vec{x}, c) = (1.2) + (0.0)(6) = 1.2$
Sports: $\sum_i w_i f_i(\vec{x}, c=2) = (2.3) + (0.2)(6) = 3.5$
Finance: $\sum_i w_i f_i(\vec{x}, c=3) = -0.5 + (-0.1)(6) = -1.1$

We can conclude that Sports will be most probable.

Training the model

Given annotated data, choose weights that make the labels most probable under the model.
Given $x^1, \ldots, x^n$ choose $c^1, \ldots, c^n \Rightarrow \hat{w} = \underset{w}{\text{argmax}} \sum_j \log P(c^j|x^j)$ → CMLE

Lecture # 13: Context Free Parsing

CFGs → Vocabulary of terminal symbols Σ
↳ set of non-terminal symbols N
↳ start symbol $S \in N$
↳ Production rules $X \to \alpha$
$X \in N, \alpha \in (N \cup \Sigma)^*$

Input: sentence $w = (w_1, \ldots, w_n)$ and CFG G
Output (recog): true iff $w \in$ Language(G)
Output (parsing): one or more derivations for w, under G

```
Agenda = {state 0}
while (Agenda not empty)
    s = pop a state from agenda
    if s is success-state return s
    elif s is not failure state:
        generate new states from s
        push new states onto agenda
return nil
```

CFGs in CNF (Chomsky Normal Form)

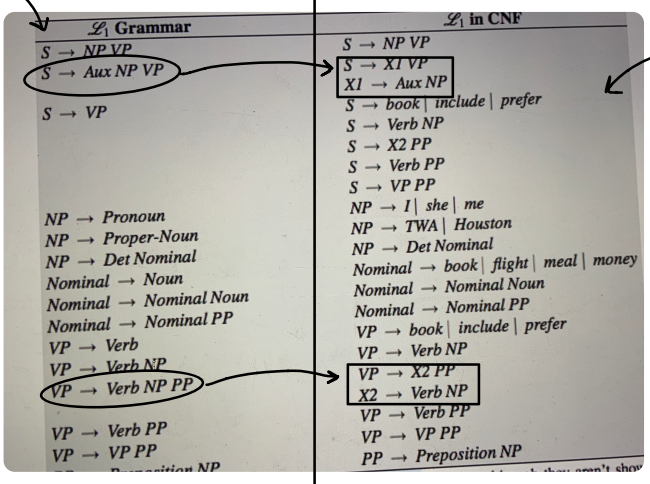
- Vocabulary of terminal symbols Σ
- Set of non-terminal symbols N
- Special start symbol $S \in N$
- Production rules of $X \to \alpha$
  $X \in N$
  $\alpha \in N, N \cup \Sigma$

Convert CFGs to CNF

For each rule,
$X \to ABC$
Rewrite
$X \to AX2$
$X2 \to BC$

- Introduce a new non-terminal
  $X2$

CFG



$\mathcal{L}_1$ Grammar	$\mathcal{L}_1$ in CNF
$S \to NP\ VP$	$S \to NP\ VP$
$S \to Aux\ NP\ VP$	$S \to X1\ VP$
	$X1 \to Aux\ NP$
$S \to VP$	$S \to book \mid include \mid prefer$
	$S \to Verb\ NP$
	$S \to X2\ PP$
	$S \to Verb\ PP$
	$S \to VP\ PP$
$NP \to Pronoun$	$NP \to I \mid she \mid me$
$NP \to Proper\text{-}Noun$	$NP \to TWA \mid Houston$
$NP \to Det\ Nominal$	$NP \to Det\ Nominal$
$Nominal \to Noun$	$Nominal \to book \mid flight \mid meal \mid money$
$Nominal \to Nominal\ Noun$	$Nominal \to Nominal\ Noun$
$Nominal \to Nominal\ PP$	$Nominal \to Nominal\ PP$
$VP \to Verb$	$VP \to book \mid include \mid prefer$
$VP \to Verb\ NP$	$VP \to Verb\ NP$
$VP \to Verb\ NP\ PP$	$VP \to X2\ PP$
	$X2 \to Verb\ NP$
$VP \to Verb\ PP$	$VP \to Verb\ PP$
$VP \to VP\ PP$	$VP \to VP\ PP$
	$PP \to Preposition\ NP$

CNF

$S \to Aux\ NP\ VP$
↓
$S \to X1\ VP$
$X1 \to Aux\ NP$

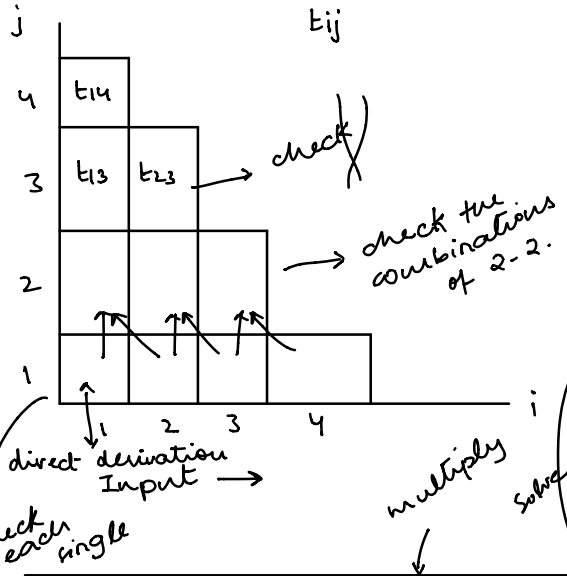
$VP \to Verb\ NP\ PP$
$VP \to X2\ PP$
$X2 \to Verb\ NP$

CYK - Algorithm

→ CFG parsing algorithm
→ Work only with CFG in CNF.

```
for i = 1 ... n
    C[i-1, i] = {V | V → wi}
for l = 2 ... n
    for i = 0 ... n-l
        k = i+l
            for j = i+1 ... k-1
                C[i,k] = C[i,k] ∪ {V | V → YZ, Y ∈ C[i,j], Z ∈ C[j,k]}
return true if S ∈ C[0,n]
```

CYK Algorithm Dry Run



Book that meal flight → n = 4

example

→ Bottom_up approach

S NP = 0 VP NP = 0 Verb NP = S, VP, X2 Nom NP = 0 Noun NP = 0	[1,2][2,4] Det Nominal = NP NP Noun = 0 NP Nominal = 0

0,4 = [0,2][2,4] ∪ [0,3][3,4] ∪ [0,1][1,4] → 0

S NP = 0 Nom NP = 0 VP NP = 0 Verb NP = S, VP, X2 N NP = 0	S Noun = 0 S Nom = 0 VP Noun = 0 VP Nom = 0

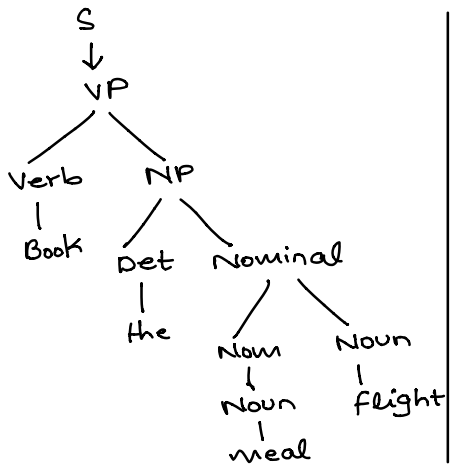
solve

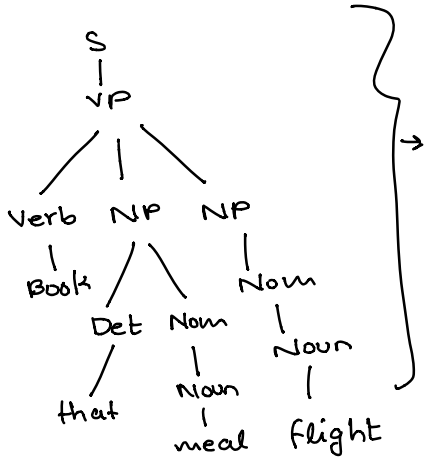
0,1 S, Nominal, VP, Verb, Noun	0,2 = [0,1][1,2] S Det = 0 Nom Det = 0 VP Det = 0 V Det = 0 N Det = 0	0,3 = [0,1][1,3] ∪ [0,2][2,3] ∪ 0 = 0 S, NP, X2	0,4 (S) NP, X2 → since S is here, it is correct.
Book	1,2 Det	1,3 = [1,2][2,3] Det N = 0 Det Nom = NP	1,4 = [1,2][2,4] ∪ [1,3][3,4] NP
	that	2,3 Noun, Nominal	2,4 = [2,3][3,4] NN = 0 N Noun = 0 Nom N = Nominal Nom Nom = 0
		meal	3,4 Noun, Nominal

flight

Probabilistic CYK

→ used if ambiguous parse





two different parse trees.

→ So, prob. CYK solves this by giving parse which has maximum prob.

Example.

S → NP VP .80
NP → Det N .30
VP → V NP .20
V → includes .05
Det → the .50
Det → a .40
N → meal .01
N → flight .02

Sentence:
the flight includes a meal

0,1	0,2	0,3	0,4	0,5
Det .50	[0,1][1,2] NP =.3x.5x.02 = 0.03	0	0	S 0.000 000 288
the	1,2 N .02	1,3 [1,2][2,3] 0	1,4 0	1,5 0
	flight	2,3 V .05	2,4 [2,3][3,4] 0	2,5 VP .000012
		includes	3,4 Det .40	3,5 [3,4][4,5] NP= .3x.4 x.01 = .0012
			a	4,5 N .01
				meal

$[0,3] = \underbrace{[0,1][1,3]}_{0} \cup \underbrace{[0,2][2,3]}_{0}$

$= 0$

$[1,4] = \underbrace{[1,2][2,4]}_{0} \cup \underbrace{[1,3][3,4]}_{0}$

$[2,5] = \underbrace{[2,3][3,5]}_{VP=.2x.05 \quad x.0012 \quad VP = .000012} \cup \underbrace{[2,4][4,5]}_{0}$

$[1,5] = [1,2][2,5] \cup [1,3][3,5] \cup [1,4][4,5]$

$= 0 \cup 0 \cup 0$

$= 0$

$[0,5] = \underbrace{[0,1][1,5]}_{S} \cup \underbrace{[0,2][2,5]}_{S} \cup \underbrace{[0,3][3,5]}_{0} \cup \underbrace{[0,4][4,5]}_{0}$

$= .8x.03x .000012 = 0.000000288$

$[0,4] = \underbrace{[0,1][1,4]}_{0} \cup \underbrace{[0,2][2,4]}_{0} \cup \underbrace{[0,3][3,5]}_{0}$

$= 0$

Earley Parsing

- Allows arbitrary CFGs
- Top down control
- Fills a table in a single sweep
- Table is length N+1 ; N → # of words
- Table entries represent
  - ✓ Completed constituents and their locations
  - ✓ In-progress constituents
  - ✓ Predicted constituents

✓✓ Table entries are called states and are represented with dotted rules.

S → .VP
NP → Det . Nominal
VP → V NP.

} VP is predicted
NP is in-progress
VP is completed

S → .VP [0,0]
NP → Det. Nominal [1,2]
VP → V NP. [0,3]

} VP is predicted at the start of sentence
NP is in progress; det goes from 1 to 2.
A VP has been found at 0 & ending at 3.

Steps

1) Predict all the states you can upfront
2) Read a word
   - Extend states based on matches
   - Generate new predictions
   - Go to step 2
3) When out of words, look at chart & see if a winner.

3 parts → Predictor, Scanner, Completer

→ S → .NP VP

Progress
S → NP.VP

S → NP VP.

→ tells which NT leads to terminal

→ Last cell shows the parsing to be complete.
→ Each cell should have NT → T transition.

Ex. Book that meal flight

→ Since n=4, so table would be 5x5

→ Construct the table 5x5, the last 0,4 should show completion.

4,0	4,1	4,2	4,3	4,4
			3,3 (Pred) PP → . Prep NP	3,4
		2,2 (Pred) Nominal → . Noun Nom → . Nom Noun Nom → . Nom PP	2,3 (Scan) Noun → meal Nom → Noun . Nom → Nom . Noun Nom → Nom . PP	2,4 (Scan) Noun → flight Nom → Nom Noun .
	1,1 Pred NP → . Pronoun × NP → . Det Nom NP → . Proper Noun × PP → . Preposition ×	1,2 Scanner Det → that (Completer) NP → Det . Nominal	1,3 (Completer) NP → Det Nom .	1,4
0,0 (Pred) Temp → . S S → . NP VP S → . Aux NP VP S → . VP NP → . Pronoun NP → . Proper Noun	0,1 (Scanning) Verb → book (Completer) VP → Verb . VP → Verb . NP VP → Verb . NP PP VP → Verb . PP S → VP .	0,2	0,3 (Comp) VP → Verb NP . VP → Verb NP . PP VP → Verb . PP VP → VP . PP S → VP .	0,4 VP → Verb NP . S → VP .

× disregard because book not aux. So, do not derive.

So, disregard all NP → cases.

write this.

NP → . Det Nom
VP → . Verb
VP → . Verb NP
VP → . Verb NP PP
VP → . Verb PP
VP → . VP PP

Outcome:

Verb → Book [0,1]
Det → that [1,2]
Noun → meal [2,3]
Noun → flight [2,4]

Noun → Nominal Noun [2,4]
NP → Det Nominal [1,3]
VP → Verb NP [0,4]
S → VP [0,4]

→ Should have complete sentence

Note

- When we are predicting, we go up in the table.
- When we have scanner or completor, we move right.

Runtime → $O(n^3)$
Memory → $O(n^2)$

Lecture # 15: Lexical Semantics

Lexical semantics → linguistic study of meaning
Decomposition → What the components of meanings in a word are?
Ontological → How the meaning of word relates to meanings of other words.
Distributional → What contexts the word is found in, relative to other words.

Semantic Relations
- Synonymy → Equivalence
- Antonymy → Opposition
- Hyponymy → subset, is-a relation
- Hypernymy → superset
- Meronymy → Part-of relation
- Holonymy → has-a relation

✓ WordNet is a useful resource → requires many years of manual effort
- There are intrinsic limit to this
  ↳ The ontology is only as good as the ontologist.

Word Similarity: Distributional Models
The intuition → meaning of a word is related to distribution of words around it.

- Represent word w as a feature vector.
- Suppose we had one binary feature fi representing each of N words in lexicon vi. The feature means w occurs in neighborhood of word vi and hence takes value 1 if w and vi are in same context window and 0 otherwise.

→ We could represent meaning of word w as the feature vector

$$\vec{w} = (f_1, f_2, f_3, \ldots, f_N)$$

Example:
The corpus is
- S1 = The sky is blue & beautiful
- S2 = Love this blue & beautiful sky
- S3 = Today is sunny
- S4 = The sky is very blue & sky is very beautiful today.

If w = sky, v1 = blue, v2 = beautiful, v3 = love & v4 = today
co-occurence vector for w from corpus above would be:

$$\vec{w} = (1, 1, 0, \ldots)$$

Co-occurence Vector → Remove stopwords

S1 sky, blue, beautiful
S2 love, blue, beautiful, sky
S3 today, sunny
S4 sky, very, blue, beautiful, today

	sky	blue	beautiful	love	today	sunny	very
sky	1	1	1	1	1	0	1
blue	1	0	1	1	1	0	1
beautiful	1	1	0	1	1	0	1
love	1	1	1	0	0	0	0
today	1	1	1	0	0	1	1
sunny	0	0	0	0	1	0	0
very	1	1	1	0	1	0	0

4 kinds of vector models

- Sparse vector representations
  - → Mutual information weighted word co-occurence matrices
- Dense vector representations
  - → SVD (and Latent Semantic Analysis)
  - → NN-inspired models (Skipgrams, CBOW)
  - → Brown clusters

Distributional Similarity Measure

- Specifying a distributional similarity measure requires that we specify three parameters:
  - → how co-occurence terms are defined (neighbor?)
  - → how these terms are weighted
  - → what vector distance metric we use (cosine?)

✓ If vocab size = N, each w (word) had N features specifying whether vocabulary element vj occured in the neighborhood.

✓ Neighborhoods range from small window of words to very large windows of ±500 words.

Even with removal of stopwords, when used on large corpa, these co-occurence vectors tend to be large. So choose words that occur in some sort of grammatical relation or syntactic dependency to target word.

→ tea, water, coffee → drink etc

→ Since each word can be in a variety of different dependency relations with other words, augment the feature space. Each feature is now a pair of word and a relation. So we have vector of N×R features where R = # of possible relations.

Co-occurence values are typically thought of as weights or measures of association b/w each target word w and given feature f.

Association measure can be:

- Binary → for each feature, 1 ∈ relevant word occured in context otherwise 0.
- Relative → with which the particular context feature had co-occured with target word.

⇒ Freq or prob are better measures of association.

Measures of Association with context

For target word w, each element of its co-occurence vector is a feature f consisting of a relation r and a related word w'; we can say f=(r,w')

The prob. of feature f given a target word w is P(f|w) for which MLE is

$$P(f|w) = \frac{count(f,w)}{count(w)}$$

Similarly for joint prob. P(f,w)

$$P(f,w) = \frac{count(f,w)}{\sum_{w'} count(f,w')}$$

⇒ words like it, they, are ⇒ not good discriminative, no info.

Mutual information b/w two random variables X & Y:

$$I(X,Y) = \sum_x \sum_y P(x,y) \log_2 \left(\frac{P(x,y)}{P(x)P(y)} \right)$$

Pointwise MI ⇒ measure of how often two events x and y occur compared with what we would expect if they were independent.

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

We can use this to compute co-occurence matrix target w (word), feature f

$$assoc_{PMI}(w,f) = \log_2 \frac{P(w,f)}{P(w)P(f)}$$

Computing PPMI on a term-context matrix

count(w, context)

	computer	data	pinch	result	sugar	
apricot	0	0	1	0	1	
pineapple	0	0	1	0	1	
digital	2	1	0	1	0	
information	1	6	0	4	0	= 11

word ↓ context →

f_{ij}

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N} \qquad p(c_j) = \frac{\sum_{i=1}^{W} f_{ij}}{N}$$

sum

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_i * p_j} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } >0 \\ 0 & \text{otherwise} \end{cases}$$

PPMI → positive pointwise mutual information
↳ replace all -ve values with zero.

$$N = \sum_{i}^{W}\sum_{j}^{C} f_{ij} = 19$$

p(w=information, c=data) = 6/19 = 0.32
p(w=information) = 11/19 = 0.58
p(w=data) = 7/19 = 0.37

	p(w,context)					p(w)
	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

Now we can compute pmi.

$$pmi(info, data) = \log\left(\frac{0.32}{0.58 \times 0.37}\right) = 0.58$$

→ Similarly we can compute for all.
For PPMI table, 0 for all -ve values.

PPMI(w,context)

	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

→ To define similarity between two target words v & w, we need a measure of taking two such vectors and giving a measure of vector similarity.
Two simplest are:

$$\text{Manhattan (L1)} = \sum^{N} |x-y|$$

$$\text{Euclidean (L2)} = \sqrt{\sum^{N} (|x-y|)^2}$$

⇒ But both are sensitive to extreme values.

$$\text{dot-product}(v,w) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1w_1 + v_2w_2 + \cdots + v_Nw_N$$

↳ High when two vectors in similar directions.
longer if vector is longer ⇒ higher values in each dimension

$$|\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

→ more frequent words will have higher dot products.
So, divide by lengths.

$$\cos(v,w) = \frac{v\cdot w}{|v||w|} = \frac{\sum^{N} v_i w_i}{\sqrt{\sum^{N} v_i^2}\sqrt{\sum^{N} w_i^2}}$$

0 → orthogonal
−1 → vectors in opposite direction
+1 → vectors in same direction

PPMI are positive, so range → 0 − 1

So,

$$\text{cosine(apricot, information)} = \frac{1\times1 + 0\times6 + 1\times0}{\sqrt{1^2+0^2+0^2}\times\sqrt{1^2+6^2+1^2}} = 0.16$$

and so on.

Lecture # 16: SVD

Alternative to PPMI for measuring association

tf-idf (term frequency - inverse document freq)

freq. of a word

df_i → document frequency of word i

$$idf_i = \log\left(\frac{N}{df_i}\right)$$

w_{ij} = word i in document j

$$w_{ij} = tf_{ij}\, idf_j$$

→ PPMI vectors are sparse & long

dense vectors → short & easy to learn
↳ generalize better than storing counts

Three ways to get dense vectors:

- SVD → Singular Value Decomposition
- Neural Language Model → skip grams, CBOW
- Brown Clustering

Dense Vectors via SVD

- Approx N-dimensional dataset using fewer dimensions.
- By first rotating the axes into new space
  → Highest order dimension captures the most variance

Any rectangular wxc matrix X equals the product of 3 matrices.

W: rows corresponding to original but m columns represent dimensions in a new latent space such that
- M column vectors are orthogonal to each other
- Columns are ordered by amount of variance in dataset each new dimension accounts for.

S: diagonal mxm matrix of singular values expressing importance of each dimension

C: columns corresponding to original but m rows corresponding to singular values

$$\underset{w \times c}{\text{words } X} = \underset{w \times m}{W} \; \underset{m \times m}{S} \; \underset{m \times c}{C}$$

(context)

→ Instead of keeping all m dimensions, keep top-k singular values.

Each row of W:
→ A k-dimensional vector for word w

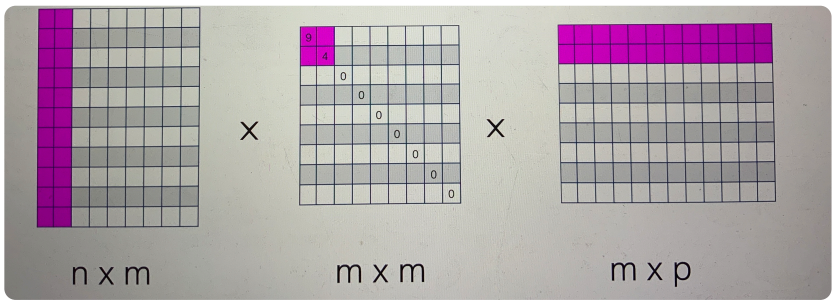
→ words X contexts = W S C → LSA
wxc wxk kxk kxc

LSA → k=300 commonly
The cells are weighted by product of two weights.
Local weight = log freq
Global weight = Either idf or entropy measure

Any nxp matrix X can be decomposed into product of three matrices (where m = # of linearly independent rows)

X = nxm, mxm, mxp
nxp

we can approx the full matrix by considering the leftmost k terms in the diagonal matrix



LSA → applying SVD to term-document co-occurence matrix
- Terms are typically weighed by tf-idf
- Dimensionality reduction → for terms, from a D-dimensionality sparse vector to a k-dimensionality dense one.
K << D

Applying SVD to PPMI

X = W C
|V|x|V| |V|xk kxk kx|V|

→ Each row in W is k-dimensional rep. of each word w.
→ k → ranges from 50 to 1000.

Example

A = mxn, m = terms, n = documents / concepts

d1: Romeo & Juliet
d2: Juliet: o happy dagger!
d3: Romeo died by dagger
d4: live free or die, that's New Hampshire's motto
d5: Did you know, New Hampshire is in New England.

Query: dies, dagger

Construct matrix A.

A(words, document)

	d1	d2	d3	d4	d5
Romeo	1	0	1	0	0
Juliet	1	1	0	0	0
happy	0	1	0	0	0
dagger	0	0	1	0	0
live	0	0	0	1	0
die	0	0	1	1	0
free	0	0	0	1	0
New Hampshire	0	0	0	1	1

fill based on frequencies

$A = USV^T$

Eigen vector of B

$B = AA^T$ (term-term matrix)

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 2 \end{bmatrix}$$

(8x5) A^T (5x8) (8x8)

$B =$

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1	2	1	0	1	0	1	0	0
2	1	2	1	0	0	0	0	0
3	0	1	1	0	0	0	0	0
4	1	0	0	1	0	1	0	0
5	0	0	0	0	1	1	1	1
6	1	0	0	1	1	2	1	1
7	0	0	0	0	1	1	1	1
8	0	0	0	0	1	1	1	2

the eigen vector of B

$U =$

0.3902	-0.3497	0.0567	-0.1645	-0.4192	-0.2853	-0.5745	0.3285
-0.3902	0.3497	-0.0567	-0.1645	-0.1093	0.5812	-0.5745	0.1237
0.3902	-0.3497	0.0567	0.3971	0.5503	0.4518	-0.2380	0.0316
0.0018	0.3820	-0.5950	0.3971	0.2404	-0.4147	-0.2380	0.2365
-0.1995	-0.4766	-0.5867	-0.3971	0.1311	0.1665	0.2380	0.3601
-0.3920	-0.0323	0.5383	-0.0000	0.3715	-0.2482	0.0000	0.5966
0.5915	0.5089	0.0484	-0.3971	0.1311	0.1665	0.2380	0.3601
-0.0000	-0.0000	0.0000	0.5615	-0.5286	0.2959	0.3366	0.4522

$C = A^T A$ → (doc-doc matrix)

$$C = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 & 0 \\ 1 & 1 & 3 & 1 & 0 \\ 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$V^T =$

-0.4298	0.3039	0.3039	-0.3039	0.7337
-0.5905	0.4926	0.2152	0.1173	-0.5905
-0.1957	-0.6832	0.6271	-0.2518	-0.1957
-0.6219	-0.4397	-0.4397	0.4397	0.1821
0.2040	0.0701	0.5241	0.7982	0.2040

↓ eigen vectors of C

The S → square root of eigen values of either AA^T(B) or A^TA(C).

So, the S would be

0.7654	0	0	0	0
0	0.8952	0	0	0
0	0	1.5121	0	0
0	0	0	1.8478	0
0	0	0	0	2.2164

We can set $k = 2$ (or any to get top *)

$K = 2$

$A_2 = U_2 S_2 V_2^T$ } Truncated SVD

So,

$$S_2 = \begin{bmatrix} 0.7654 & 0 \\ 0 & 0.8952 \end{bmatrix}$$

Similarly we can take 2 rows from U & 2 cols from V.

Term = $U \times S$ → doc-doc matrix's eigen vectors

$$\text{Romeo} = \begin{bmatrix} 0.7654 \times & 0.3902 \\ 0.8952 \times & -0.3497 \end{bmatrix} = \begin{bmatrix} 0.2986 \\ -0.3130 \end{bmatrix}$$

$$\text{Juliet} = \begin{bmatrix} 0.7654 \times & -0.3902 \\ 0.8952 \times & 0.3497 \end{bmatrix} = \begin{bmatrix} -0.2986 \\ 0.3130 \end{bmatrix}$$

Doc = SV^T

$$\text{Query} = \frac{\overset{\text{die}}{\begin{bmatrix} -1.197 \\ -0.494 \end{bmatrix}} + \overset{\text{dagger}}{\begin{bmatrix} -1.001 \\ 0.742 \end{bmatrix}}}{2} = \begin{bmatrix} -1.099 \\ 0.124 \end{bmatrix}$$

⇒ compute similarity using cosine similarity

In $K=2$, $A_2 = U_2 S_2 V_2^T$

0.3902	-0.3497	0.0567	-0.1645	-0.4192	-0.2853	-0.5745	0.3285
-0.3902	0.3497	-0.0567	-0.1645	-0.1093	0.5812	-0.5745	0.1237
0.3902	-0.3497	0.0567	0.3971	0.5503	0.4518	-0.2380	0.0316
0.0018	0.3820	-0.5950	0.3971	0.2404	-0.4147	-0.2380	0.2365
-0.1995	-0.4766	-0.5867	-0.3971	0.1311	0.1665	0.2380	0.3601
-0.3920	-0.0323	0.5383	-0.0000	0.3715	-0.2482	0.0000	0.5966
0.5915	0.5089	0.0484	-0.3971	0.1311	0.1665	0.2380	0.3601
-0.0000	-0.0000	0.0000	0.5615	-0.5286	0.2959	0.3366	0.4522

U_2

-0.4298	0.3039	0.3039	-0.3039	0.7337
-0.5905	0.4926	0.2152	0.1173	-0.5905
-0.1957	-0.6832	0.6271	-0.2518	-0.1957
-0.6219	-0.4397	-0.4397	0.4397	0.1821
0.2040	0.0701	0.5241	0.7982	0.2040

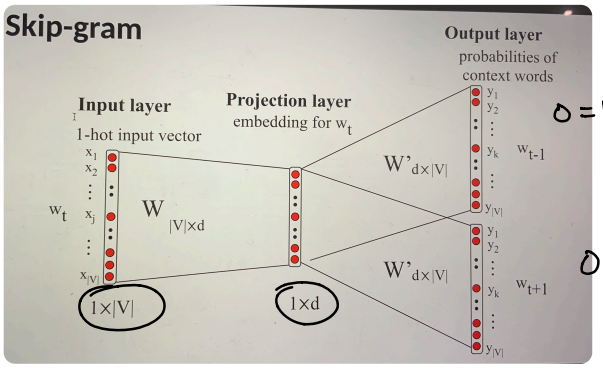
V_2^T

Neural Language Models

→ Skip-gram & CBoW

↓

faster, trainable, dense



$O_K = v_K' \cdot v_j$

Use softmax to turn into probs.

$$p(w_k|w_j) = \frac{\exp(v_K' \cdot v_j)}{\sum_{w \in |V|} \exp(v_w' \cdot v_j)}$$

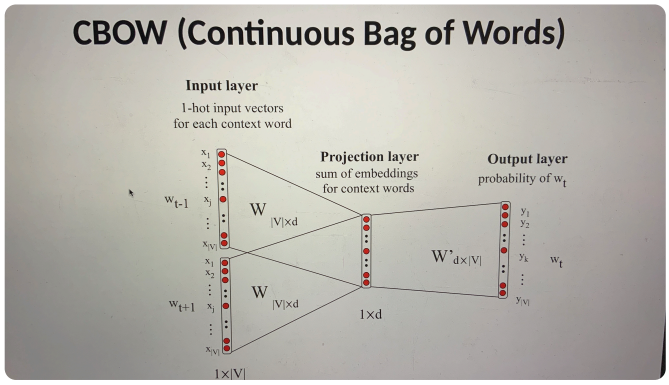
Since we have two embeddings v_j and v_j' for each word w_j

✓ Perform linear combination.

→ $WW^T = M^{PMI} - \log k$

→ So, skip gram is implicitly factoring a shifted version of the PMI matrix into the two embedding matrices.

CBOW



Brown Clustering

An agglomerative clustering algo that clusters based on which word preceeds or follow them

Evaluating Similarity

Extrinsic (task-based, end to end)
→ QA, Spell check, Essay grading

Intrinsic
→ correlation b/w algo & human word sim.

Lecture #9 & 10: Classification

Perceptron Learning

If n-th member of training set $x(n)$ is correctly classified by weight vector $w(n)$ computed at n-th iteration, no correction is made to weight vector.

$w(n+1) = w(n)$ if $w^T x(n) > 0$ & $x(n) \in C_1$
$w^T x(n) \leq 0$ & $x(n) \in C_2$

Otherwise weight vector is updated

→ $w(n+1) = w(n) - \eta(n)x(n)$ if $w^T x(n) > 0$ & $x(n) \in C_1$
$w(n+1) = w(n) + \eta(n)x(n)$ if $w^T x(n) \leq 0$ & $x(n) \in C_2$

or

$w(n+1) = w(n) + \eta(n)e(n)x(n)$

where

$e(n) = d(n) - y(n)$

{ If $d=0$ → subtract $(wn - x(n))$
If $d=1$ → add $(wn + x(n))$ }

Lecture # 11 : Syntax

Morphology deals with internal structure of the words.

Syntax deals with combination of words.
↳ Phrases, sentences

often irregular

Often regular
made up of general rules that apply across the board.

Semantics → about meaning
Syntax → about structure alone
A sentence can be syntactically well but semantically ill-formed.

Constituent: A group of words that go together
↳ larger than a word are called phrases.

→ Every Prep. P → contains a NP

CFG
↙
context-free because there is no context in LHS of rules – there is just one symbol.

CFGs → declarative programming
→ specify what is to be computed in terms of rules and let generalized computation mechanisms solve for particular cases.

Grammatical → a sentence in the language
→ A string is grammatical iff there exists a derivation for it.

A good grammar:
- Doesn't over-generate (high precision)
- Doesn't under-generate (high recall)

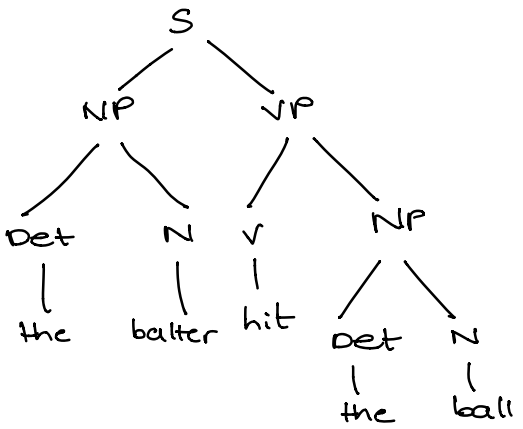
Parse tree → One step in discovering relations (grammatical) which can help discover semantic roles

Myths

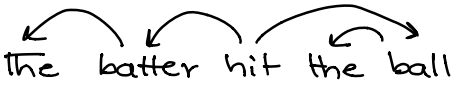
(1) the subj is first noun-phrase in a sentence
(2) the subj is the actor in a sentence
(3) the subj is what the sentence is about

} often true but not always

Constituency tree



Dependency tree



Advantages and Disadvantages

Advantages of Constituency/Phrase Structure Grammar

- There are widely agreed-upon tests for constituency; there is little agreement above what constitutes a dependency relation
- Constituency maps more cleanly on to formal semantic representations than dependency
- This makes constituency useful in natural language understanding

Advantages of Dependency Grammar

- It is easier to identify grammatical relations (like subject and object) in a dependency parse
- Dependency parses of sentences having the same meaning are more similar across languages that constituency parses
- Dependency parses are also useful for NLU (ask Google)
- Dependency trees are typically simpler

→ Advantages & disadvantages of constituent & dependency grammar.

CFGs → not prob.
↳ not very convenient.

Lecture # 12: CFG

A CFG consists of a set of rules or productions, each of which expresses the ways that symbols of language can be grouped & ordered together.
→ rules are hierarchically embedded, so we can combine the previous rules with others.

Lecture # 14: Lexical Semantics

Lexical semantics → linguistic study of meaning
↗ least useful approach

- Decompositional ⇒ what the components of meaning in a word are
- Ontological ⇒ How meaning of word relates to meaning of other words
- Distributional ⇒ what context the word is found in, relative to other words

Synonymy – equivalence (small, little)
Antonymy – opposition (small, large)
Meronymy – part-of (liver, body)

Hyponymy – subset; is-a relation (dog, mammal)
Hypernymy – superset (mammal, dog)
Holonymy – has-a relation (body, liver)

Propositional Meaning → substituable without changing meaning

→ Define synonymy as a relation b/w senses other than between words.
→ group of antonyms
→ Reversies : some of change or movement in opposite directions
Antonyms → words with opposite meaning defining a binary opposition or are two ends of some scale.

Ontology

→ describes common words, concepts and relationships b/w concepts used to describe and rep an area of knowledge.

WordNet → lexical resource that organizes words according to their semantic relations.

Synset → A set of words that are roughly synonymous for a particular sense.

In a wordnet, each sense of word is associated with a synset.

Each synset is associated with another through relations → antonymy, hyponymy etc.

consists of three databases → one for nouns
→ one for verbs
→ one for adverbs/adjectives

⇒ closed class words are not included.

Verb Relations in WordNet

Relation	Definition	Example
Hypernym	From events to superordinate events	$fly^9 \rightarrow travel^5$
Troponym	From a verb (event) to a specific manner elaboration of that verb	$walk^1 \rightarrow stroll^1$
Entails	From verbs (events) to the verbs (events) they entail	$snore^1 \rightarrow sleep^1$
Antonym	Opposites	$increase^1 \Longleftrightarrow decrease^1$

Figure 19.3 Verb relations in WordNet.

Noun Relations in WordNet

Relation	Also called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	*breakfast*^{1} → *meal*^{1}
Hyponym	Subordinate	From concepts to subtypes	*meal*^{1} → *lunch*^{1}
Member Meronym	Has-Member	From groups to their members	*faculty*^{2} → *professor*^{1}
Has-Instance		From concepts to instances of the concept	*composer*^{1} → *Bach*^{1}
Instance		From instances to their concepts	*Austen*^{1} → *author*^{1}
Member Holonym	Member-Of	From members to their groups	*copilot*^{1} → *crew*^{1}
Part Meronym	Has-Part	From wholes to parts	*table*^{2} → *leg*^{3}
Part Holonym	Part-Of	From parts to wholes	*course*^{7} → *meal*^{1}
Antonym		Opposites	*leader*^{1} → *follower*^{1}

Figure 19.2 Noun relations in WordNet.

Information Content

$$IC(c) = -\log \frac{\#\text{ of words that are equivalent to or are hyponyms of } c}{\#\text{ of words in corpus}}$$

Lecture # 17: Word2Vec

→ predict rather than count

Train a binary classifier on the task:

Is w likely to show up near "apricot"? (any word)

We take the learned classifier weights as word embeddings

A word s near apricot

- Acts as a gold "correct answer" to the question.

W2V → provides alot of options.

Skipgrams with Negative Sampling

- Treat target & neighboring word as +ve examples
- Randomly sample other words to get -ve examples
- Use log regression to train a classifier to distinguish two cases
- Use weights as embeddings

Use sigmoid instead of dot prod. to get prob.

$$\sigma(x) = \frac{1}{1+e^{x}}$$

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

$$P(-|t,c) = 1 - P(+|t,c)$$

target, context

We have to maximize

$$\sum_{(t,c)\in +} \log P(+|t,c) + \sum_{(t,c)\in -} \log P(-|t,c)$$