Language Modeling

Unit-II

Syed Rameem Zahra
(Assistant Professor)
Department of CSE, NSUT

What is a Language Model in NLP?

- A language model learns to predict the probability of a sequence of words.
- It’s a statistical tool that analyzes the pattern of human language for the prediction of words, by estimating the relative likelihood of different phrases.
- The models are prepared for the prediction of words by learning the features and characteristics of a language.
- Language models are used in speech recognition, machine translation, part-of-speech tagging, parsing, Optical Character Recognition, handwriting recognition, information retrieval, summarization, spell correction, and many other daily tasks.

Challenges with Language Modeling

- Formal languages (like a programming language) are precisely defined, but Natural language isn't designed, it evolves according to the convenience and learning of an individual.
- There are several terms in natural language that can be used in a number of ways, which introduces ambiguity but can still be understood by humans.

Some Common Examples of Language Models

- **Speech Recognization**
  - Voice assistants such as Siri and Alexa are examples of how language models help machines in processing speech audio.
- **Machine Translation**
  - Google Translator and Microsoft Translate are examples of how NLP models can help in translating one language to another.
- **Sentiment Analysis**
  - This helps in analyzing the sentiments behind a phrase. This use case of NLP models is used in products that allow businesses to understand a customer’s intent behind opinions or attitudes expressed in the text.
- **Text Suggestions**
  - Google services such as Gmail or Google Docs use language models to help users get text suggestions while they compose an email or create long text documents, respectively.
- **Parsing Tools**
  - Parsing involves analyzing sentences or words that comply with syntax or grammar rules. Spell checking tools are perfect examples of language modelling and parsing.

Types of Language Models

- **Statistical Language Models:**
  - These models use traditional statistical techniques like **N-grams**, Hidden Markov Models (**HMM**) and certain linguistic rules to learn the probability distribution of words.
- **Neural Language Models:**
  - These are new players in the NLP town and have surpassed the statistical language models in their effectiveness. They use different kinds of Neural Networks to model language.

Goal of Probabilistic Language Modelling

- To calculate the probability of a sentence of sequence of words:
  - P(W) = P(w1,w2,..., wn)
- This is the **Joint Probability**
- It can be calculated using **Conditional probability**:
  - P(w5 | w1,w2,w3,w4)

$$P(w_5|w_1, w_2, w_3, w_4) = \frac{count(w_1, w_2, w_3, w_4, w_5)}{count(w_1, w_2, w_3, w_4)}$$

- E.g. for two words: X, Y, we have:
  - P(X|Y) = P(X,Y)/P(Y)
  - then, P(X,Y) = P(X|Y) P(Y)
- Similarly, for three words: P(X,Y,Z) = P(X) P(Y|X) P(Z|X,Y)
- This method is used when we have short/limited sentence.
- For longer sentences, we can use **Markov assumption**:
  - P(wn | w1,w2,w3,...,wn-1) ~ P(wn|wn-1) or P(wn|wn-2,wn-1)

N-Gram Model

- This is one of the simplest approaches to language modelling.
- Here, a probability distribution for a sequence of **‘n’** is created, where ‘n’ can be any number and defines the size of the gram (or sequence of words being assigned a probability).
- If n=4, a gram may look like: “can you help me”.
- Basically, ‘n’ is the amount of context that the model is trained to consider.
- There are different types of N-Gram models such as **unigrams, bigrams, trigrams**, etc.
- The intuition of the n-gram model is that **instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.**

(Uni-) 1-gram model

- The simplest case of Markov assumption is case when the size of prefix is one.
- This will provide us with grammar that only consider one word. As a result it produces a set of unrelated words.
- It actually would generate sentences with random word order.

Bigram Model

- Approximates the probability of a word given all the previous words by using only the conditional probability of the preceding word.
- we consider a 2-word (**tandem**) bigrams correlations
- In other words, instead of computing the probability
    - P(the|Walden Pond's water is so transparent that)
- we approximate it with the probability
    - P(the|that)

How to estimate these bigram or n-gram probabilities?

- An intuitive way to estimate probabilities is called **maximum likelihood estimation** or **MLE**.
- We get Maximum likelihood estimate for the parameters of an n-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1.
- ***For example***, to compute a particular bigram probability of a word $\mathbf{w_n}$ given a previous word $\mathbf{w_{n-1}}$, we'll compute the count of the bigram $\mathbf{C(w_{n-1}w_n)}$ and normalize by the sum of all the bigrams that share the same first word $\mathbf{w_{n-1}}$

Note: A **corpus** is a collection of authentic text or audio organized into datasets.

Example-1

- We have a mini-corpus of three sentences:
- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like green eggs and ham </s>
- Here are the calculations for some of the bigram probabilities from this corpus:

$$P(\texttt{I}|\texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam}|\texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am}|\texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>}|\texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam}|\texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do}|\texttt{I}) = \tfrac{1}{3} = .33$$

- In practice it's more common to use **trigram** models, which condition on the previous two words rather than the previous word, or **4-gram** or even **5-gram** models, when there is sufficient training data.

Example-2: Bi-gram probabilities

- What is the most probable next word predicted by the model for the following word sequence:

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Example-2: Bi-gram probabilities (Contd…)

1) <S> Do ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability W_{i-1}=do

Next word	Probability Next Word= $\frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$
P(</S> | do)	0/4
P(<I> | do)	**2/4**
P(<am>| do)	0/4
P(<Henry>| do)	1/4
P(<like | do)	1/4
P(<college | do)	0/4
P(do | do)	0/4

I is more probable

Example-2: Bi-gram probabilities (Contd…)

2) <S> I like Henry ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability W_{i-1}**=Henry**

Next word	Probability Next Word= $\frac{N}{D} = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$
P(</S> | Henry)	**3/5**
P(<I> | Henry)	1/5
P(<am>| Henry)	0
P(<Henry>| Henry)	0
P(<like | Henry)	1/5
P(<college | Henry)	0
P(do | Henry)	0

</S> is more probable

Example-2: Bi-gram probabilities (Contd…)

Which of the following sentence is better. i.e. Gets a higher probability with this model. Use Bi-gram

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1. <S> I like college </S>

<S> like college </S>=?

$=P(I \mid \langle S\rangle) \times P(like \mid I) \times P(college \mid like) \times P(\langle /S\rangle \mid college)$

$=3/7 \times 3/6 \times 3/5 \times 3/3 = 9/70 = 0.13$

2. <S> Do I like Henry </S>

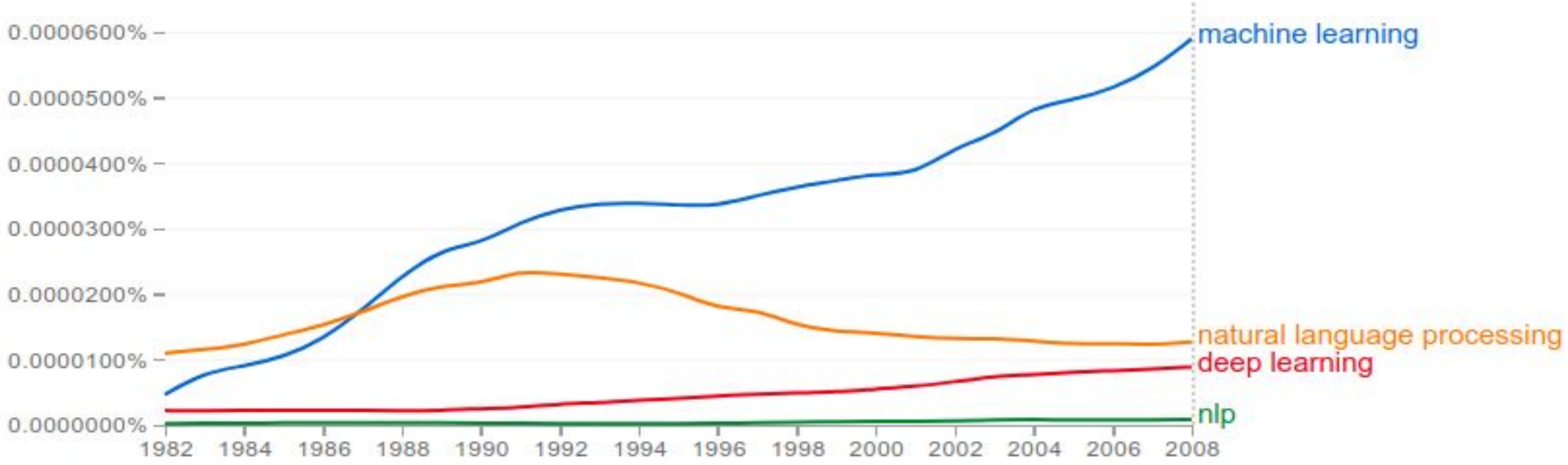
$=P(do \mid \langle S\rangle) \times P(I \mid do) \times P(like \mid I) \times P(Henry \mid like) \times P(\langle /S\rangle \mid Henry)$

$=3/7 \times 2/4 \times 3/6 \times 2/5 \times 3/5 = 9/350 = 0.0257$

ANS: First statement is more probable

Publicly available corpora

- **Gutenberg Project** providing with text format of some books.
- **Google** also released a publicly available corpus, trillion word corpus with over 13 million unique words.



N-gram Language Model

- **Advantages:**
  - Easy to understand and implement
  - Conversion from one type of gram to another is easy.
- **Disadvantages:**
  - Underflow due to multiplication of probabilities
    - Solution: Use Log (which will add probabilities)
  - Zero probability problem
    - Solution: Use Laplace smoothing

Using Log to solve underflow problem

Which of the following sentence is better. i.e. Gets a higher probability with Bi-gram model.

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

First statement is more probable

1. <S> I like college </S>

=P(I| <S>) × P(like | I) × P(college| like) × P(</S> | college)

$=3/7 \times 3/6 \times 3/5 \times 3/3 = 9/70 = \mathbf{0.13}$

$= \log(3/7) + \log(3/6) + \log(3/5) + \log(3/3) = \mathbf{-2.0513}$

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

$=3/7 \times 2/4 \times 3/6 \times 2/5 \times 3/5 = 9/350 = \mathbf{0.0257}$

$=\log(3/7) + \log(2/4) + \log(3/6) + \log(2/5) + \log(3/5) = \mathbf{-3.6607}$

Zero Probability Problem

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Second statement is more probable

1. <S> like college </S>

=P(like | <S>) × P(college | like) × P(</S> | college)

=0/7 × 3/5 ×3/3 =**0**

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

=3/7 × 2/4 × 3/6 ×2/5 ×3/5 = 9/350=**0.0257**

Smoothing

- To keep a language model from assigning zero probability to these unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen.
- This modification is called smoothing (or discounting).
- There are many ways to do smoothing, and some of them are:
  - Add-1 smoothing (Laplace Smoothing)
  - Add-k smoothing,
  - Backoff
  - Kneser-Ney smoothing.

Bigram	Frequency
CS 421	8
CS 590	5
CS 594	2
CS 521	0 😢

Bigram	Frequency
CS 421	7
CS 590	5
CS 594	2
CS 521	1 🥰

Laplace Smoothing

- Add one to all n-gram counts before they are normalized into probabilities.
- Not the highest-performing technique for language modeling, but useful method for text classification

$$P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

Laplace Smoothing

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Unique words are : <S>, </S>, I, Henry do, like, am, college

Total unique words: 8

But we exclude <S> as it never comes in bi-gram calculations

Total unique words: 7

Give the following bi-gram probabilities estimated by Laplace model.

Applying Laplace Smoothing

1. <S> like college </S>

$=P(\text{like} \mid \text{<S>}) \times P(\text{college} \mid \text{like}) \times P(\text{</S>} \mid \text{college})$

$=(0+1)/(7+7) \times (3+1)/(5+7) \times (3+1)/(3+7)$

$=1/14 \times 4/12 \times 4/10$

$=\mathbf{0.0095}$

2. <S> Do I like Henry </S>

$=P(\text{do} \mid \text{<S>}) \times P(\text{I} \mid \text{do}) \times P(\text{like} \mid \text{I}) \times P(\text{Henry} \mid \text{like}) \times P(\text{</S>} \mid \text{Henry})$

$=(3+1)/(7+7) \times (2+1)/(4+7) \times (3+1)/(6+7) \times (2+1)/(5+7) \times (3+1)/(5+7)$

$= 4/14 \times 3/11 \times 4/13 \times 3/12 \times 4/12$

$=\mathbf{0.0020}$

Add-K Smoothing

- Rather than adding one to each count, add a fractional count (0.5, 0.05, 0.01 etc.)
- The value of K can be optimized on a validation set

$$P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Add-K}}(w_i) = \frac{c_i + k}{N + kV}$$

$$P(w_n | w_{n-1}) = \frac{c(w_{n-1}w_n)}{c(w_{n-1})} \rightarrow P_{\text{Add-K}}(w_n | w_{n-1}) = \frac{c(w_{n-1}w_n) + k}{c(w_{n-1}) + kV}$$

Backoff and Interpolation

- Add-K smoothing is useful for some tasks, but still tends to be suboptimal for language modeling.
- Other techniques are:
  - **Backoff:** we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram.
    - In other words, we only “back off” to a lower-order n-gram if we have zero evidence for a higher-order n-gram.
  - **Interpolation:** we always mix the probability estimates from all the n-gram estimators, weighing and combining the trigram, bigram, and unigram counts.

Simple Linear Interpolation

- we combine different order n-grams by linearly interpolating all the models.

$$
\begin{aligned}
\hat{P}(w_n|w_{n-1}w_{n-2}) \;=\; & \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\
& +\lambda_2 P(w_n|w_{n-1}) \\
& +\lambda_3 P(w_n)
\end{aligned}
\qquad \sum_i \lambda_i = 1
$$

Language Model Evaluation:

Language model is better if it is assigning a high probability to the real, frequently observed and grammatical sentence over false, rarely observed and ungrammatical sentences.

Two different criteria for evaluation

1) Extrinsic 2) Intrinsic

Extrinsic Evaluation

It evaluate the language model when solving a specific task.

For e.g. Speech recognition accuracy, Machine translation accuracy, Spelling correction accuracy Compare 2 (or more) models, and check which works best.

Disadvantage:

- Expensive
- Time consuming

Intrinsic Evaluation:

The language model is best when it predicts an unseen test set.

Definition of Perplexity:

It is the inverse probability of the test data which is normalized by the number of words.

Lower the value of perplexity: **Better Model**

More value of perplexity: **Confused for prediction**

Perplexity

The language model is best when it predicts an unseen test set.

Definition of Perplexity:

It is the inverse probability of the test data which is normalized by the number of words.

$$PP(w) = P(w_1, w_2, w_3, \ldots . w_N)^{-\frac{1}{N}}$$

$$PP(w) = \left(\prod_i \frac{1}{P(w_i \mid w_1, w_2, \ldots\ldots, w_{i-1})} \right)^{\frac{1}{N}} \qquad PP(w) = \left(\prod_i \frac{1}{P(w_i \mid w_{i-1})} \right)^{\frac{1}{N}}$$

Lower the value of perplexity: **Better Model**

More value of perplexity: **Confused for prediction**

Example

WSJ Corpus **Training:** 38 million words **Test:** 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Perplexity for Bigram <S> I like college </S>

$=P(I \mid \langle S \rangle) \times P(like \mid I) \times P(college \mid like) \times P(\langle /S \rangle \mid college)$

$=3/7 \times 3/6 \times 3/5 \times 3/3 = 9/70 = \mathbf{0.13}$

$\mathbf{PP(w)=(1/0.13)^{1/4}=1.67}$

Perplexity for Trigram <S> I like college </S>

$P(w)=P(like \mid \langle S \rangle\, I) \times P(college \mid I\ like) \times P(\langle /S \rangle \mid like\ college)$

$P(w) = 1/3 \times 2/3 \times 3/3 = 2/9 = \mathbf{0.22}$

$\mathbf{PP(w)=(1/0.22)^{1/3}=1.66}$

Text Classification

- Text Classification (Text Categorization) is the task of assigning a label or categorization category to an entire text or document.
- Some of common text categorization tasks are:
    - **Sentiment analysis**
        - extraction of sentiment, positive or negative orientation that writer expresses toward an object.
    - **Spam detection**
        - binary classification task of assigning an email to one of the two classes spam or not-spam.
    - **Authorship identification**
        - determining a text’s author.
    - **Age/gender identification**
        - determining a text’s author characteristics like gender and age.
    - **Language Identification**
        - finding the language of a text.

Text Classification: Definition

- Text classification can be defined as follows:
- Input:
  - a document d
  - a fixed set of classes C = {c1 , c2 ,…, cn }
- Output:
  - a predicted class c belongs-to C

Classification Methods: Hand-Coded Rules

- The goal of classification is to take a single observation, extract some useful features, and thereby classify the observation into one of a set of discrete classes.
- One method for classifying text is to use hand-written rules.
- Rules based on combinations of words or other features
  - spam: black-list-address OR (“dollars” AND “have been selected”)
- Accuracy can be high if rules carefully are refined by experts.
- But building and maintaining these hand-written rules can be expensive.
  - Rules can be fragile.
  - It may require domain knowledge.

Classification Methods: Supervised Machine Learning

- Most cases of text classification in language processing are done via supervised machine learning methods.
- The goal of a supervised machine learning algorithm is to learn how to map from a new observation to a correct output.
- Input:
  - a document d
  - a fixed set of classes C = {c1 , c2 ,…, cn }
  - a training set of m hand-labeled documents (d1 ,c1),....,(dm,cm)
- Output:
  - a learned classifier model: d → c

Classification Methods: Supervised Machine Learning

- Our goal is to learn a classifier that is capable of mapping from a new document d to its correct class c belongs-to C.
- A probabilistic classifier additionally will tell us the probability of the observation being in the class.
- Generative classifiers like Naive Bayes build a model of how a class could generate some input data.
    - Given an observation, they return the class most likely to have generated the observation.
- Discriminative classifiers like logistic regression instead learn what features from the input are most useful to discriminate between the different possible classes.
- Some classifiers:
    - Naïve Bayes
    - Logistic regression
    - Support-vector machines
    - k-Nearest Neighbors, …

Text Classification: Evaluation

- In order to evaluate **how good is our classifier**, we can use different evaluation metrics.
- In evaluation, we compare the **test set results** of our classifier with **gold labels** (the human labels for the test set documents).
- As a result of this comparison, first we build a **contingency table (or confusion matrix)** before calculate our evaluation metrics:

		gold standard labels		
		gold positive	gold negative	
system output labels	system positive	**true positive**	**false positive**	**precision** $= \frac{tp}{tp+fp}$
	system negative	**false negative**	**true negative**	
		recall $= \frac{tp}{tp+fn}$		**accuracy** $= \frac{tp+tn}{tp+fp+tn+fn}$

Text Classification: Evaluation

- **Accuracy** is percentage of all the observations our system labeled correctly.
- **Precision** measures percentage of items that system detected that are in fact positive.
- **Recall** measures percentage of items actually present in the input that were correctly identified by the system.
- Precision and Recall, unlike Accuracy, emphasize true positives.
  - Looking only one of them can be misleading.
  - tp=1 fp=0 fn=99, then, Precision = 100% (while Recall=1%)
  - tp=1 fp=99 fn=0, then, Recall= 100% (while Precision=1%)
- **F-measure** is a single metric that incorporates aspects of both precision and recall.

Text Classification: Evaluation

$$F_\beta = \frac{(\beta^2+1)*P*R}{\beta^2*P+R}$$

The **β** parameter differentially weights the importance of recall and precision.

- values of $\beta>1$ favor recall, while values of $\beta<1$ favor precision.

The most frequently used metric, and is called $F_{\beta=1}$ or just F_1:

$$F_1 = \frac{2*P*R}{P+R}$$

Text Classification with more than two classes

		gold labels			
		urgent	normal	spam	
system output	urgent	8	10	1	**precision**$_u = \frac{8}{8+10+1}$
	normal	5	60	50	**precision**$_n = \frac{60}{5+60+50}$
	spam	3	30	200	**precision**$_s = \frac{200}{3+30+200}$
		recall$_u = \frac{8}{8+5+3}$	**recall**$_n = \frac{60}{10+60+30}$	**recall**$_s = \frac{200}{1+50+200}$	**Accuracy** $= \frac{\sum_i c_{ii}}{\sum_j \sum_i c_{ij}}$

Microaveraging and Macroaveraging

- In order to derive a single metric that tells us how well the system is doing, we can combine these values in two ways.
  - In **macroaveraging**, compute performance for each class, and then average over classes.
  - In **microaveraging**, collect decisions for all classes into a single contingency table, and then compute precision and recall from that table.

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\begin{array}{c}\text{microaverage}\\ \text{precision}\end{array} = \frac{268}{268+99} = .73$$

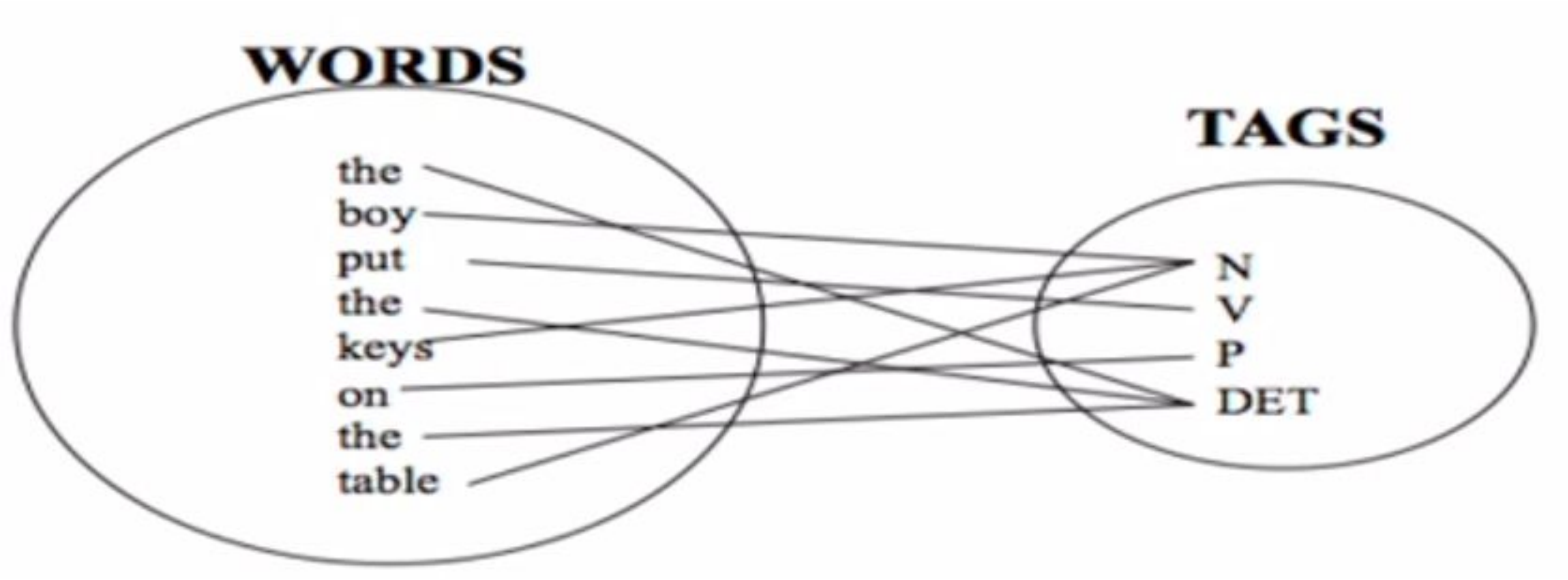
$$\begin{array}{c}\text{macroaverage}\\ \text{precision}\end{array} = \frac{.42+.52+.86}{3} = .60$$

Cross-validation

- we randomly choose a training and test set division of our data, train our classifier, and then compute the error rate on the test set.
- Then we repeat with a different randomly selected training set and test set.
- We do sampling process 10 times and average these 10 runs to get an average error rate.
- This is called 10-fold cross-validation.

Part-of-Speech Tagging

- Given a text of english, identify the parts of speech of each word



Part-of-Speech Tagging

- **Open class words** (content words):
  - Nouns, verbs, adjectives, adverbs.
  - They refer to objects, actions and features in the world.
  - They are open class because new words are added all the time.
- **Closed class words:**
  - Pronouns, determiners, prepositions, connectives,...
  - They are limited.
  - Mostly functional: to tie the concepts of a sentence together.l

Part-of-Speech Tagging

N	noun	chair, bandwidth, pacing
V	verb	study, debate, munch
ADJ	adj	purple, tall, ridiculous
ADV	adverb	unfortunately, slowly,
P	preposition	of, by, to
PRO	pronoun	I, me, mine
DET	determiner	the, a, that, those

POS Tagging: Choosing a target

- For POS tagging, we need to choose a standard set.
- E.g., We could choose a very coarse targets like N, V, Adj, Adv etc.
- A commonly used set is: “UPenn TreeBank” tagset, which contains 45 tags.

UPenn TreeBank POS tagset

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	$
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	“	Left quote	(‘ or “)
POS	Possessive ending	*'s*	”	Right quote	(’ or ”)
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PRP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

POS Tagging is hard???

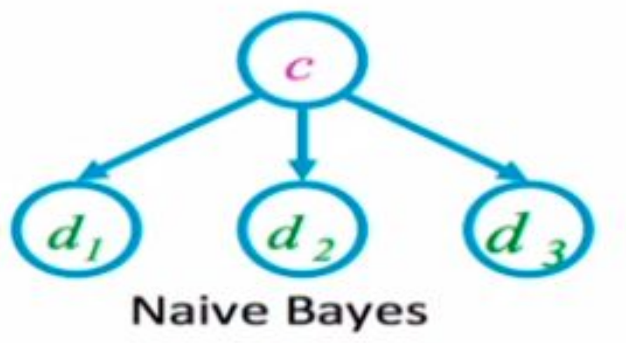
- A word in a sentence may have multiple POS tags depending on the context.
    - E.g., For the word “Back”, we have:
    - The back door: *back/JJ* -> Adjective
    - On my back: *back/NN* *-> Noun*
    - Win the voters back: *back/RB* *-> Adverb*
- It has been seen that mostly we have 2 to 3 tags for many words (Ambiguity problem)
- We can use any valid corpus to find the highest probability of a word for tagging it to a particular POS tag, which designing a model.
    - E.g.: Some words may only be nouns like arrow
    - Some words are ambiguous like flies
    - Probability may help us if one tag is more likely than another.
    - Also the local context can be used.

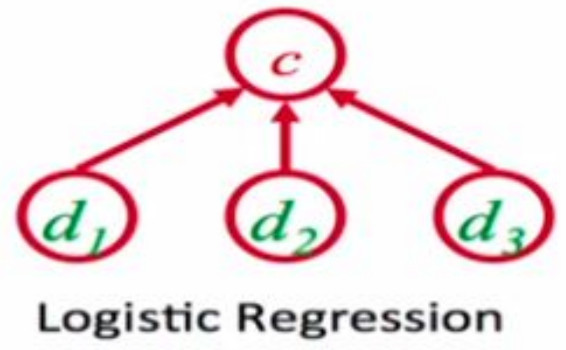
POS Tagging Approaches

- **Rule based approach:**
  - Assign each word in the input sentence a list of potential POS tags.
  - Then, scale down the list to a single tag using hand-written rules.
- **Statistical tagging:**
  - Get a training corpus of tagged text, learn the transformation rules from most frequently tags (e.g. TBL (Transformation Based Learning) Tagger).
  - Find the most likely sequence of tags for a sequence of words using probability.

Generative and Conditional Models

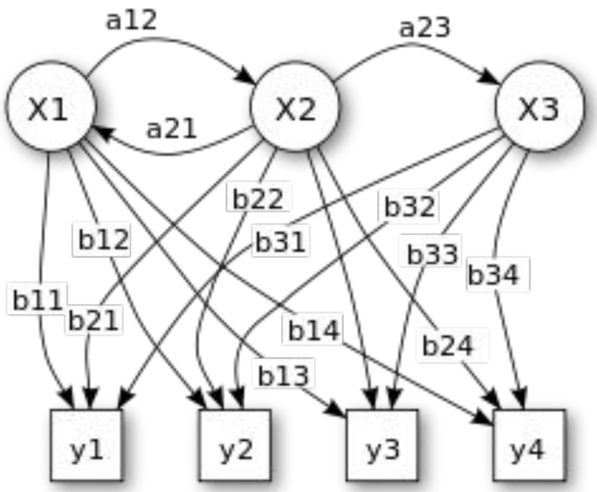
- **Generative (Joint) Model:**
  - Generate the observed data from hidden stuff, i.e., put a probability over the observation given the class: P(d,c) in terms of P(d|c)
  - E.g., Naive bayes classification, Hidden Markov Models etc.
- **Conditional (Discriminative) Models:**
  - Take the data as given and put a probability over hidden structure given the data: P(c|d)
  - E.g., Logistic regression, max. Entropy models, SVMs, Perceptron etc.





Generative Model: Hidden Markov Model

- A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process — call it **X** with unobservable ("hidden") states.
- As part of the definition, HMM requires that there be an observable process **Y** whose outcomes are "influenced" by the outcomes of **X** in a known way.
- Since **X** cannot be observed directly, the goal is to learn about **X** by observing **Y**.
- HMM has an additional requirement that the outcome of Y at time $\mathbf{t = t_o}$ must be "influenced" exclusively by the outcome of **X** at time $\mathbf{t = t_o}$ and that the outcomes of X and **Y** at $\mathbf{t < t_o}$ must not affect the outcome of **Y** at $\mathbf{t = t_o}$.
- Similar to N-Gram models
- Model the text as a sequence
- For ngrams, we modeled the probability of each word conditioned on the previous n-1 words.
- Here, we model each tag conditioned on previous tags
- Still uses **Markov assumption**: only look back a few tags



X — states
y — possible observations
a — state transition probabilities
b — output probabilities

Generative Model: Hidden Markov Model

- We want the most likely tag sequence for a sequence of words:

  $$p(\langle \mathrm{S} \rangle\ t_1\ t_2\ \ldots\ t_n\ \langle \mathrm{E} \rangle \mid \langle \mathrm{S} \rangle\ w_1\ w_2\ \ldots\ w_n\ \langle \mathrm{E} \rangle)$$

  Remember that order matters!

- For simplicity, we'll write this as $t_1^n = \langle t_1, t_2, \ldots, t_n \rangle$

- So we want

  $$\hat{t}_1^n = \mathrm{argmax}_{t_1^n}\ p(\langle \mathrm{S} \rangle\ t_1\ t_2\ \ldots\ t_n\ \langle \mathrm{E} \rangle \mid \langle \mathrm{S} \rangle\ w_1\ w_2\ \ldots\ w_n\ \langle \mathrm{E} \rangle)$$

  but it's hard to estimate

- Like for ngrams, use Bayes rule:

  $$\begin{aligned} \hat{t}_1^n &= \mathrm{argmax}_{t_1^n} \frac{p(\langle \mathrm{S} \rangle\ w_1\ w_2\ \ldots\ w_n\ \langle \mathrm{E} \rangle \mid \langle \mathrm{S} \rangle\ t_1\ t_2\ \ldots\ t_n\ \langle \mathrm{E} \rangle) \cdot p(\langle \mathrm{S} \rangle\ t_1\ t_2\ \ldots\ t_n\ \langle \mathrm{E} \rangle}{p(\langle \mathrm{S} \rangle\ w_1\ w_2\ \ldots\ w_n\ \langle \mathrm{E} \rangle)} \\ &= \mathrm{argmax}_{t_1^n}\ p(\langle \mathrm{S} \rangle\ w_1\ w_2\ \ldots\ w_n\ \langle \mathrm{E} \rangle \mid \langle \mathrm{S} \rangle\ t_1\ t_2\ \ldots\ t_n\ \langle \mathrm{E} \rangle) \cdot p(\langle \mathrm{S} \rangle\ t_1\ t_2\ \ldots\ t_n\ \langle \mathrm{E} \rangle) \end{aligned}$$

- Two major independence assumptions:
  - Like ngrams, assume probability of a sequence is dependent only on recent past:

  $$\begin{aligned} p(\langle \mathrm{S} \rangle\ t_1\ t_2\ \ldots\ t_n\ \langle \mathrm{E} \rangle) &\approx p(t_1 \mid \langle \mathrm{S} \rangle) \cdot p(t_2 \mid t_1) \cdot p(t_2 \mid t_2) \cdot \ldots \cdot p(t_n \mid t_{n-1}) \cdot p(\langle \mathrm{E} \rangle \mid t_n) \\ &= \prod_{i=1}^{n} p(t_i \mid t_{i-1}) \end{aligned}$$

Generative Model: Hidden Markov Model

- Also assume word is only dependent on its tag:

$$p(\langle \mathrm{S} \rangle\ w_1\ w_2\ \dots\ w_n\ \langle \mathrm{E} \rangle \mid \langle \mathrm{S} \rangle\ t_1\ t_2\ \dots\ t_n\ \langle \mathrm{E} \rangle) \approx p(w_1 \mid t_1) \cdot p(w_2 \mid t_2) \cdot \dots \cdot p(w_n \mid t_n)$$
$$= \prod_{i=1}^{n} p(w_i \mid t_i)$$

- Together:

$$\begin{aligned}
\hat{t}_1^n &= \mathrm{argmax}_{t_1^n}\ p(\langle \mathrm{S} \rangle\ t_1\ t_2\ \dots\ t_n\ \langle \mathrm{E} \rangle \mid \langle \mathrm{S} \rangle\ w_1\ w_2\ \dots\ w_n\ \langle \mathrm{E} \rangle) \\
&= \mathrm{argmax}_{t_1^n}\ p(\langle \mathrm{S} \rangle\ w_1\ w_2\ \dots\ w_n\ \langle \mathrm{E} \rangle \mid \langle \mathrm{S} \rangle\ t_1\ t_2\ \dots\ t_n\ \langle \mathrm{E} \rangle) \cdot p(\langle \mathrm{S} \rangle\ t_1\ t_2\ \dots\ t_n\ \langle \mathrm{E} \rangle) \\
&\approx \mathrm{argmax}_{t_1^n}\ \prod_{i=1}^{n} p(w_i \mid t_i) \cdot \prod_{i=1}^{n} p(t_i \mid t_{i-1}) \\
&= \mathrm{argmax}_{t_1^n}\ \prod_{i=1}^{n} p(w_i \mid t_i) \cdot p(t_i \mid t_{i-1})
\end{aligned}$$

Estimating Parameters: MLE in Hidden Markov Model

Two probability distributions to estimate:

- Transitions: probability of a tag, given previous tag, $p(t_i \mid t_{i-1})$
- Emissions: probability of a word, given its tag, $p(w_i \mid t_i)$

MLE

- MLE estimation is just like before (naïve Bayes, ngrams, ...): normalized counts
- Transitions: $p(t_i \mid t_{i-1}) = \frac{C(t_{i-1}\ t_i)}{\sum_x C(t_{i-1}\ x)} = \frac{C(t_{i-1}\ t_i)}{C(t_{i-1})}$
- Emissions: $p(w_i \mid t_i) = \frac{C(t_i, w_i)}{\sum_x C(t_i, x)} = \frac{C(t_i, w_i)}{C(t_i)}$

Example

Example dataset (punctuation excluded for simplicity!):

```
<S>|<S> the|D man|N walks|V the|D dog|N <E>|<E>
<S>|<S> the|D dog|N runs|V <E>|<E>
<S>|<S> the|D dog|N walks|V <E>|<E>
<S>|<S> the|D man|N walks|V <E>|<E>
<S>|<S> a|D man|N saw|V the|D dog|N <E>|<E>
<S>|<S> the|D cat|N walks|V <E>|<E>
```

Some probabilities:

- $p(t_i = \text{D} \mid t_{i-1} = \text{N}) = \frac{C(\text{N D})}{\sum_x C(\text{N } x)} = \frac{0}{8} = 0.0$
- $p(t_i = \text{V} \mid t_{i-1} = \text{N}) = \frac{C(\text{N V})}{\sum_x C(\text{N } x)} = \frac{6}{8} = 0.75$
- $p(w_i = \text{dog} \mid t_i = \text{N}) = \frac{C(\text{N,dog})}{\sum_x C(\text{N},x)} = \frac{4}{8} = 0.50$
- $p(w_i = \text{the} \mid t_i = \text{N}) = \frac{C(\text{N,the})}{\sum_x C(\text{N},x)} = \frac{0}{8} = 0.0$

Note: We can add smoothing techniques also (as discussed earlier)

Three Tasks for HMM

- **Likelihood:** Given a tagged sequence, determine its likelihood
- **Decoding:** Given an untagged sequence, determine the best tag sequence for it
- **Learning:** Given an untagged sequence, and a set of tags, learn the HMM parameters

Likelihood of a tagged sentence

We can compute the likelihood of a particluar sequence of tags for a sentence:

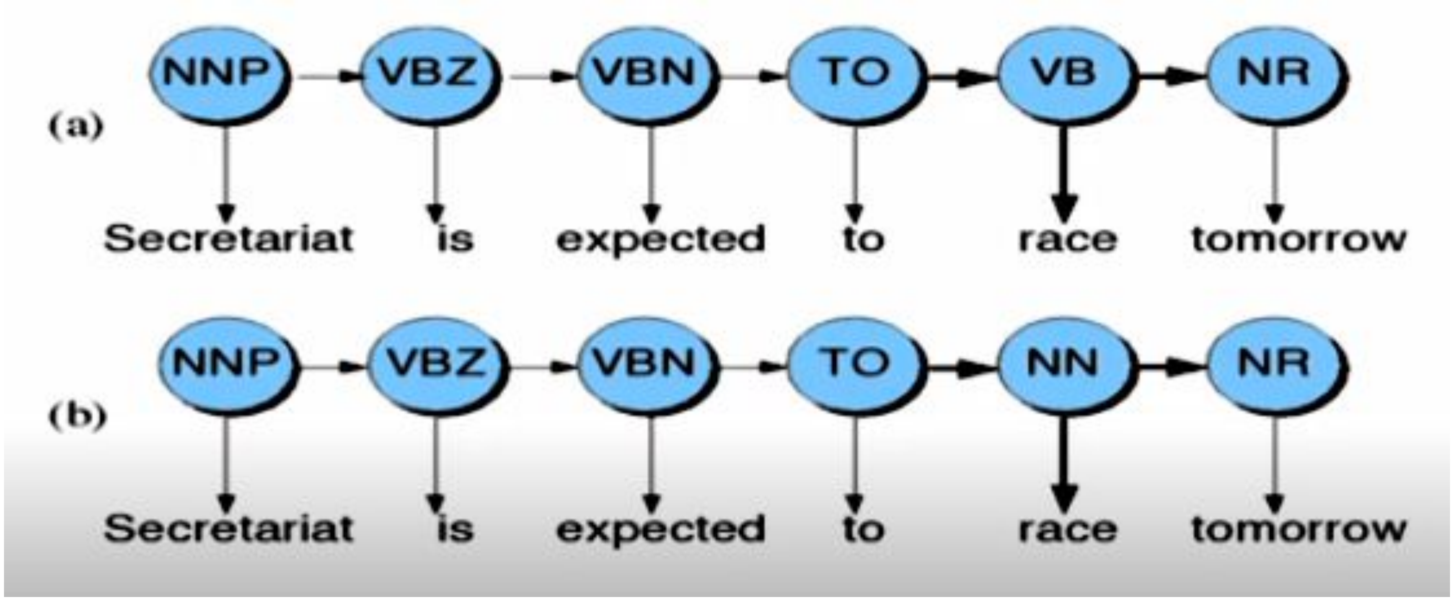
- $p(t_1 ... t_n \mid w_1 ... w_n) = \prod_{i=1}^{n} p(w_i \mid t_i) \cdot p(t_i \mid t_{i-1})$

Example: "the|D dog|N walks|V"

$$p(t_1 ... t_n \mid w_1 ... w_n) \approx \prod_{i=1}^{n} p(w_i \mid t_i) \cdot p(t_i \mid t_{i-1})$$

$$= p(D \mid \langle \mathrm{S} \rangle) \cdot p(the \mid D) \cdot p(N \mid D) \cdot p(dog \mid N) \cdot p(V \mid N) \cdot p(walks \mid V) \cdot p(\langle \mathrm{E} \rangle \mid V)$$

Example: Disambiguation of “race”



- Using HHM and available corpus, we can find most likely sequence of tags using probability values.

Issues with Markov Model Tagging

- Missing probabilities for **unknown words**.
  - ***Solution:*** Use morphological cues (Capitalization or suffixes etc.) to assign a more calculated guess.
- **Limited context** may not be sufficient for correct tagging.
  - ***Solution:*** use higher order HMM (like 2nd order or 3rd order etc.) and combine various N-gram models.

Maximum Entropy Model: Discriminative Model

- Uses a combination of heterogeneous set of features to create a probabilistic model, which is able to select a correct PoS tag for a current word, e.g.:
  - Whether the next word is **to**.
  - Whether one of the last 5 words is a **preposition**. etc.

$$p_\lambda(y|x) = \frac{1}{Z_\lambda(x)} exp\left(\sum_i \lambda_i f_i(x,y)\right)$$

where

- $Z_\lambda(x)$ is a normalizing constant given by

$$Z_\lambda(x) = \sum_y exp\left(\sum_i \lambda_i f_i(x,y)\right)$$

- λ_i is a weight given to a feature f_i
- x denotes an observed datum and y denotes a class

- **Principles of Max. Entropy Model:** Given a collection of facts (features), choose a model which is consistent (Uniform) with all the facts.

Features in Max. Entropy Model

- Features encode elements of the context x for predicting tag y
- Context x is taken around the word w, for which a tag y is to be predicted
- Features are binary values functions, e.g.,

$$f(x,y) = \left\{ \begin{array}{ll} 1 & \text{if } isCapitalized(w) \& y = NNP \\ 0 & otherwise \end{array} \right\}$$

Examples of features:

Example: Named Entities

- LOCATION (in Arcadia)
- LOCATION (in Québec)
- DRUG (taking Zantac)
- PERSON (saw Sue)

Example Features

- $f_1(x,y) = [y = LOCATION \wedge w_{-1} = \text{“in”} \wedge isCapitalized(w)]$
- $f_2(x,y) = [y = LOCATION \wedge hasAccentedLatinChar(w)]$
- $f_3(x,y) = [y = DRUG \wedge ends(w, \text{“c”})]$

PoS Tagging in Max. Entropy Model

- $W = w_1 \ldots w_n$ - words in the corpus (observed)
- $T = t_1 \ldots t_n$ - the corresponding tags (unknown)

Tag sequence candidate $\{t_1, \ldots, t_n\}$ has conditional probability:

$$P(t_1, \ldots, t_n | w_1 \ldots, w_n) = \prod_{i=1}^{n} p(t_i | x_i)$$

- The context x_i also includes previously assigned tags for a fixed history.

Entropy in Max Entropy Model

- It measures the uncertainty (surprise) of a distribution.
- For an event x with probability of occurrence p_x, Entropy = log $(1/p_x)$
- Entropy H for a random variable X with probability distribution P is given as:

$$E_p\left[log_2 \frac{1}{p_x}\right] = -\sum_x p_x log_2 p_x$$

- So, in Max. Entropy Model, we choose a model with max. Entropy, subjected to feature-based constraints.
- We will start from a uniform distribution (because it has max. Entropy) and the add constraints, which will decrease the entropy and make it closer to the given data.

Example

Consider the maximum entropy model for POS tagging, where you want to estimate $P(tag|word)$. In a hypothetical setting, assume that *tag* can take the values *D*, *N* and *V* (short forms for Determiner, Noun and Verb). The variable *word* could be any member of a set V of possible words, where V contains the words *a*, *man*, *sleeps*, as well as additional words. The distribution should give the following probabilities

- $P(D|a) = 0.9$
- $P(N|man) = 0.9$
- $P(V|sleeps) = 0.9$
- $P(D|word) = 0.6$ for any word other than *a*, *man* or *sleeps*
- $P(N|word) = 0.3$ for any word other than *a*, *man* or *sleeps*
- $P(V|word) = 0.1$ for any word other than *a*, *man* or *sleeps*

It is assumed that all other probabilities, not defined above could take any values such that $\sum_{tag} P(tag|word) = 1$ is satisfied for any word in V.

- Define the features of your maximum entropy model that can model this distribution. Mark your features as f_1, f_2 and so on. Each feature should have the same format as explained in the class. [**Hint:** 6 Features should make the analysis easier]
- For each feature f_i, assume a weight λ_i. Now, write expression for the following probabilities in terms of your model parameters
  - $P(D|cat)$
  - $P(N|laughs)$
  - $P(D|man)$
- What value do the parameters in your model take to give the distribution as described above. (i.e. $P(D|a) = 0.9$ and so on. *You may leave the final answer in terms of equations*)

Example Contd…

- **F1** = F1(x,y) = [word = ‘a’ & tag = ‘D’]
- **F2** = F2(x,y) = [word = ‘man’ & tag = ‘N’]
- **F3** = F3(x,y) = [word = ‘sleeps’ & tag = ‘V’]
- **F4** = F4(x,y) = [word ∈ V` & tag = ‘D’], Where V` =V - {a,man,sleeps}; V is Vocabulary
- **F5** = F5(x,y) = [word ∈ V` & tag = ‘N’]
- **F6** = F6(x,y) = [word ∈ V` & tag = ‘V’]
- Now, **P(D|cat) = $e^{(\sum \lambda i F i)}$/Z**
  - $\sum \lambda_i F_i = \lambda_1 * 0 + \lambda_2 * 0 + \lambda_3 * 0 + \lambda_4 * 1 + \lambda_5 * 0 + \lambda_6 * 0 = \lambda_4$
  - To calculate Z, we need to calculate P(N|cat) and P(V|cat)
  - P(N|cat) = $e^{\lambda 5}$/Z and P(V|cat) = $e^{\lambda 6}$/Z; Also, P(D|cat)+P(N|cat)+P(V|cat)=1 => Z = $e^{\lambda 4} + e^{\lambda 5} + e^{\lambda 6}$
  - Hence, **P(D|cat) = $e^{\lambda 4}/(e^{\lambda 4} + e^{\lambda 5} + e^{\lambda 6})$**
- Similarly, we can calculate **P(N|laugh)** and **P(D|man)**

Example Contd…

- **P(D|a)** = $e^{\lambda 1}/Z$, to calculate Z here, we have:
  - PV|a)=$e^0/Z=1/Z$ P(N|a)=$e^0/Z=1/Z$
  - Hence, P(D|a) = $e^{\lambda 1}/(e^{\lambda 1}+2) = 0.9$
- Similarly, we can calculate equation for other given constraints and get values of λs, which will represent the overall Max. entropy model for the given problem.