CSE6328.3
Speech & Language Processing

HTK and the Project two

Prof. Hui Jiang
Department of Computer Science and Engineering
York University

HTK: A Toolkit for HMM-based Speech Recognition

- HTK: software toolkit for HMM-based speech recognition
- Originally built in Cambridge Univ. (UK); Acquired and released by Microsoft Inc..
- HTK provides a set of tools to process speech data, transcription, grammar network, HMM training, HMM decoding, ASR evaluation, …
- Unix-style of command-line:

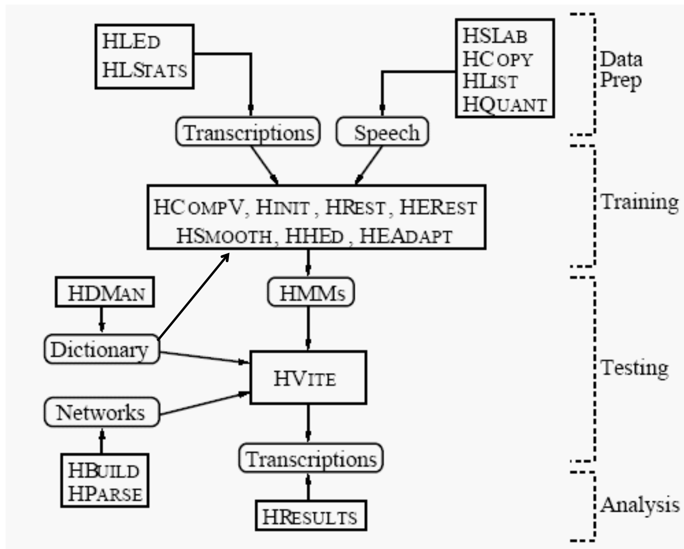
  tool-name [options] mandatory_arguments
- Easy to write shell scripts to perform large-scale experiments.
- A Linux version of HTK is available at:

  /cs/course/6328/project2/HTK34
- Also sample and tutorial directories:

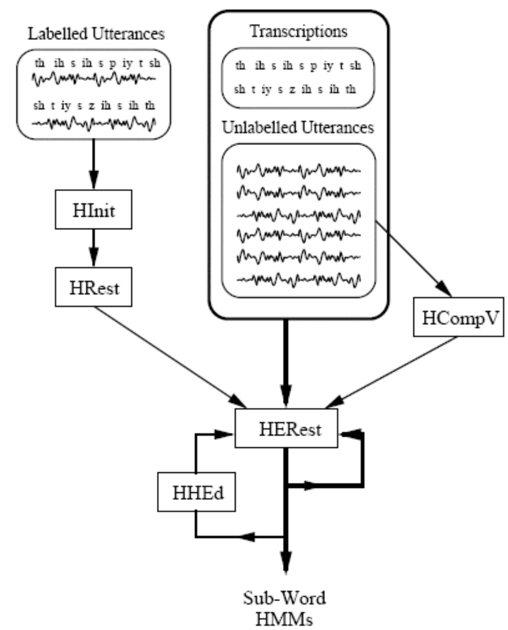
  /cs/course/6328/…/HTK-samples/HTKDemo

  /cs/course/6328/…/HTK-samples/HTKTutorial
- Use Linux machines: *indigo, navy, cherry, hickory, hemlock, willow*, etc..

HTK Tools Overview



HMM Training in HTK



HMM Training in HTK

- HMM Training in HTK:
  1. Get initial segmentation of data (uniform, hand labels, or forced alignment)
  2. Train monophone HMMs on segments (*HInit*, *HRest, HCompV*)
  3. Train monophone HMMs using embedded training (*HERest*)
  4. Create triphones from monophones by cloning (HHEd)
  5. Train triphone HMMs using embedded training (*HERest*)
  6. Create context clustering for tying parameters using decision tree (HHEd)
  7. Tie all logical states in triphones to form state-tied triphone HMMs (HHEd)
  8. Run embedded training (HERest); split mixtures (HHEd); repeat.

Important Tools for the project

- *HSLab*: view and play voice data.
- *HCopy*: feature extraction
- *HList*: list data files
- *HCompV*: flat initialization of HMMs
- *HInit*: initialize HMMs from uniform segmentation
- *HRest*: training HMMs based on the Baum-Welch algorithm.
- *HParse/HBuild*: building recognition network.
- *HVite*: Viterbi decoding with 2-gram LM.
- HDecode: Viterbi decoding (more efficient) with 2- or 3-gram LM.
- *HHEd*: change HMM model structure, decision tree based parameter tying
- *HResults*: measure recognition performance.

General Info of the project (1)

- Use HTK to build an ASR system from training data.
- Do experiments to improve your system.
- Evaluate your systems on test data and report the best.
- Requirements:
  – Use mixture Gaussian CDHMM.
  – Use mono-phone and state-tied tri-phone models
  – Can’t use any test data in HMM training.
- Progressive model training procedure:
  – Simple models → complex models
  – Single Gaussian → more mixtures
  – Mono-phone → tri-phone

General Info of the project (2)

- Project key issues: be careful on data formats
  – Use a given 3-gram LM
  – Acoustic modeling: speech unit selection → model estimation (initialization, refining, etc.)
  – Dictionary is provided
  – No need to record data (use the provided database)

General Info of the project (3)

- The expected strategy:
  1) Properly initialize HMM's from scratch.
  2) Evaluate HMM's on test set.
  3) Think about ideas to improve models.
  4) Retrain/update/enhance HMM's.
  5) Evaluate the enhanced HMM's again.
  6) Goto 3) to repeat until find your best HMM setting.
- You need to hand in the following electronic files:
  – A report (max 8 pages), *report_X.xxx*, to describe all conducted experiments; why you did them; your methods to improve the system; your best system setting and the best performance you achieved; others.
  – A training script, *train_X.script*, to get your best HMMs from scratch.
  – A test script, *test_X.script,* to evaluate your HMMs on test data.
  – A *readme_X.txt* file to explain how to run your scripts.

General Info of the project (3)

- My marking scheme:
  – The methodology you adopt in building the system.
  – Whether you follow the project specification.
  – What ideas you come up with for improving the system.
  – A full course from scratch to your best system.
  – Your best recognition performance.
  – The best systems will get bonus marks.
  – How well you do in your project presentation
- Deadlines and presentation dates

Hints for the Project

- Focus on:
  - How to initialize models ?
    - HTK provides two solutions
  - How to choose modeling unit ?
    - mono-phone
    - Consider context-dependent phone, such as left and/or right bi-phone and tri-phone.
  - How to decide the optimal Gaussian mixture number per HMM state?
  - Other ideas you come up with …
- Get your version 1.0 ASAP.

CSE6328.3
Speech & Language Processing

YORK UNIVERSITY redefine THE POSSIBLE.

No.8

Statistical Language Modeling

Prof. Hui Jiang
Department of Computer Science and Engineering
York University

YORK UNIVERSITÉ UNIVERSITY redefine THE POSSIBLE.

Automatic Speech Recognition (II): Language Modeling (LM)

Prof. Hui Jiang
Department of Computer Science and Engineering
York University, Toronto, Canada
hj@cse.yorku.ca

ASR Solution

$$\hat{W} = \arg\max_{W \in \Gamma} p(W \mid X) = \arg\max_{W \in \Gamma} P(W) \cdot p(X \mid W)$$

$$= \arg\max_{W \in \Gamma} \overline{P}_\Gamma(W) \cdot \overline{p}_\Lambda(X \mid W)$$

- $\overline{p}_\Lambda(X \mid W)$ — ***Acoustic Model (AM)***: gives the probability of generating feature ***X*** when ***W*** is uttered.
  - Need a model for every ***W*** to model all speech signals (features) from ***W*** → HMM is an ideal model for speech
  - Speech unit selection: what speech unit is modeled by each HMM? (phoneme, syllable, word, phrase, sentence, etc.)
    - Sub-word unit is more flexible (better)
- $\overline{P}_\Gamma(W)$ — ***Language Model (LM)***: gives the probability of W (word, phrase, sentence) is chosen to say.
  - ***Need a flexible model to calculate the probability for all kinds of W → Markov Chain model (n-gram)***

ASR Problems

- **Training Stage:**
  - ***Acoustic modeling***: how to select speech unit and estimate HMMs reliably and efficiently from available training data.
  - ***Language modeling***: how to estimate n-gram model from text training data; handle data sparseness problem.
- **Test Stage:**
  - ***Search***: given HMM's and n-gram model, how to efficiently search for the optimal path from a huge grammar network.
    - Search space is extremely large
    - Call for an efficient pruning strategy

N-gram Language Model (LM)

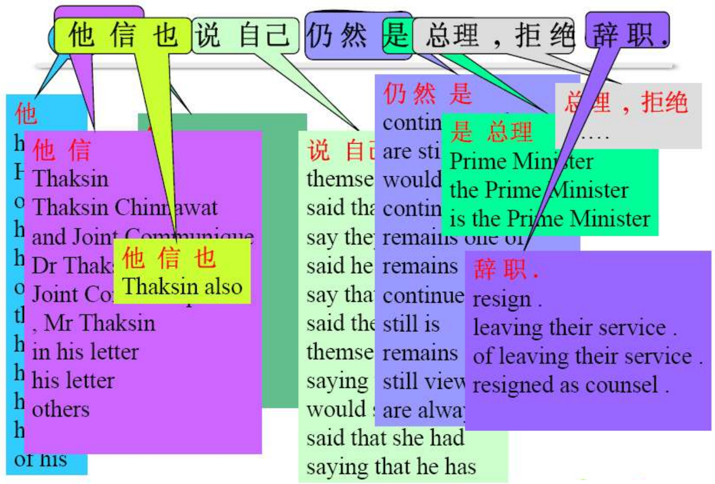
- **N-gram Language model (LM) essentially is a Markov Chain model, which is composed of a set of multinomial distributions.**
- **Given $W=w_1,w_2,\ldots,w_M$, LM probability $Pr(W)$ is expressed as**

$$\Pr(W) = \Pr(w_1, w_2, \cdots, w_M) = \prod_{i=1}^{M} p(w_i \mid h_i)$$

  - where $h_t=w_{t-n+1},\ldots,w_{t-1}$ is history of w_t.
  - In unigram, $h_t=null$ (parameters ~|V|, |V| vocabulary size)
  - In bigram, $h_t=w_{t-1}$ (parameters ~|V|*|V|)
  - In trigram, $h_t=w_{t-2}w_{t-1}$ (parameters ~|V|*|V|*|V|)
  - In 4-gram, $h_t=w_{t-3}w_{t-2}w_{t-1}$ (parameters ~|V|*|V|*|V|*|V|)
- **How to evaluate performance of LM?**

N-gram LM: Applications

- N-gram LM has many applications in speech recognition, OCR, machine translation, etc..
- N-gram LM for statistical machine translation:



Perplexity of LM

- Perplexity: the most widely used performance measure for LM.
- Given an LM { *Pr(.)* } with vocabulary size |V|, and a sufficiently long test word sequence $W=w_1,w_2,\ldots,w_M$:
  - Calculate a negative log-prob quantity per word:

$$LP = -\frac{1}{M}\log_2 \Pr(W)$$

  - Perplexity of LM is computed as

$$PP = 2^{LP}$$

- Perplexity: indicates the prediction of the LM is about as difficult as guessing a word among PP equally likely words.
- Perplexity: the smaller PP value, the better LM prediction capability.
- Training-set perplexity: how much LM fits or explain the data
- Test-set perplexity: generalization capability of the LM to predict new text data.

LM: vocabulary selection

- Large vocabulary size
  - ➔ exponential growth of various n-grams
  - ➔ exponential increasement of LM model parameters
  - ➔ much more training data and computing resources
- Need to control vocabulary size in LM.
- Given the training text data,
  - limit vocabulary of LM to the most frequent words occurring in the training corpus, e.g., the top N words.
  - All other words are mapped as unknown word, *<UNK>.*
  - This gives the lowest rate of out-of-vocabulary (OOV) words for the same vocabulary size.
- Example: English newspaper WSJ (Wall Street Journal)
  - Training corpus: 37 million words (full 3-year archive)
  - Vocabulary: 20,000 words
  - OOV rate: 4%
  - 2-gram PP: 114
  - 3-gram PP: 76

LM Training (1)

- Collect text corpus: need > tens of millions of words for 3-gram
- Corpus preprocessing: (*very time-consuming*)
  - Text clean-up: remove punctuation and other symbols
  - Normalization: 0.1% → (zero) point one percent
    6:00 → six o' clock; 1/2 → one half, …
  - Surrounding each sentence with TAGS <s> and </s>
  - Language-specific processing: e.g., for some oriental languages (Chinese, Japanese, etc.) do tokenization → find word boundaries from a stream of characters.
  - Output: clean text

<s> w1 w2 w3 w4 w5 </s>
<s> w11 w21 w32 w41 w52 w12 w22 w33 w44 w54 w16 w26 w36 w43 w56 </s>
<s> w12 w23 w31 w42 w51 w11 w23 w34 w44 w5 </s>
….

LM Training (2)

- LM parameter estimation from clean text:
  - The entire training text can be mapped into an ordered sample of n-grams without loss of information:

    $S=h_1w_1, h_2w_2, \ldots h_Tw_T$

    (assume we have T words in training corpus)
  - Group together all n-grams with the same history h :

    $S_h=hw_{x1}, hw_{x2}, \ldots , hw_{xn}$
  - S_h can be viewed as an i.i.d. sample from $Pr(w|h)$.
  - We denote $p_{hw}=p(w|h)$ for all possible w's and h's.
  - So probability of S_h follows a multinomial distribution:

$$\Pr(S_h) \propto \prod_{w \in V} [p(w \mid h)]^{N(hw)}$$

where $N(hw)$ is frequency of n-gram hw occurring in S_h.

LM Training (3): ML estimation

- Maximum Likelihood (ML) estimation of multinomial distribution is easy to derive.
- The ML estimate of n-gram LM is:

$$\arg\max_{p(w|h)} \sum_{w \in V} N(hw) \cdot \ln p(w \mid h) = \arg\max_{p_{hw}} \sum_{w \in V} N(hw) \cdot \ln p_{hw}$$

$$\text{subject to constrants } \sum_{w \in V} p_{hw} = 1 \text{ for all } h.$$

$$\Rightarrow p_{hw}^{(ML)} = \frac{N(hw)}{\sum_{w \in V} N(hw)} = \frac{N(hw)}{N(h)}$$

LM Training (3): MAP estimation

- The natural conjugate prior of multinomial distribution is the Dirichlet distribution.
- Choose Dirichlet distribution as priors:

$$p(\{p_{hw}\}) \propto \prod_{w \in V} [p_{hw}]^{K(hw)}$$

  - where { *K(hw)}* are hyper-parameters to specify the prior.
- Derive posterior p.d.f. from Bayesian learning:

$$p(\{p_{hw}\} | S_h) \propto \prod_{w \in V} [p_{hw}]^{K(hw)+N(hw)}$$

- Maximization of posteriori p.d.f. → the MAP estimate:

$$p_{hw}^{(MAP)} = \frac{N(hw)+K(hw)}{\sum_{w \in V}[N(hw)+K(hw)]}$$

- MAP estimates of n-gram LM can be used for smoothing.

Data Sparseness in LM estimation

- ML estimation never works due to data sparseness.
- Example: in 1.2 million words English text (vocabulary 1000 words)
  - 20% bigrams and 60% trigrams occur only once.
  - 85% of trigrams occur less than five times.
  - After observing the whole 1.2 Mw data, the expected chance of seeing a new bi-gram is 22%, a new tri-gram 65%.
- In ML estimation: zero-frequency → zero probability
- Data sparseness problem can not be solved by collecting more data.
  - Extremely uneven distribution of n-grams in natural language.
  - After amount of data reaches a certain point, the speed of reducing OOV rate or rate of new n-grams by adding more data becomes extremely slow.
- Call for a better estimation strategy: ***smoothing ML estimates***
  - ***Back-off scheme: discounting + redistributing***
  - ***Linear interpolation scheme***

Statistical Estimators

Example:

Corpus: five Jane Austen novels

N = 617,091 words

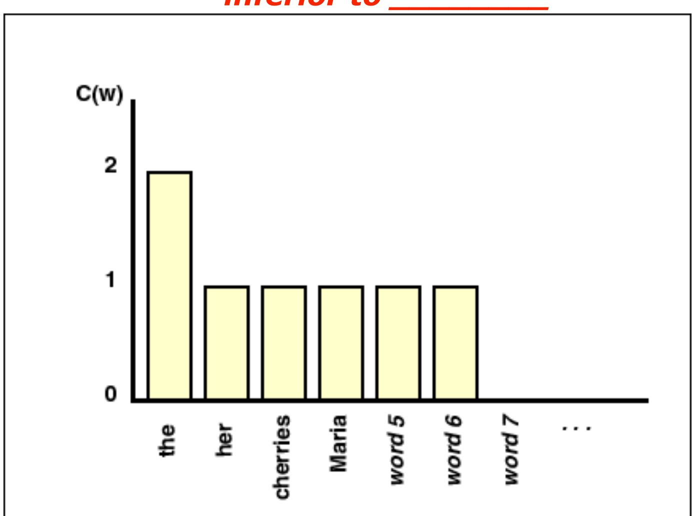
V = 14,585 unique words

Task: predict the next word of the trigram "inferior to ________"

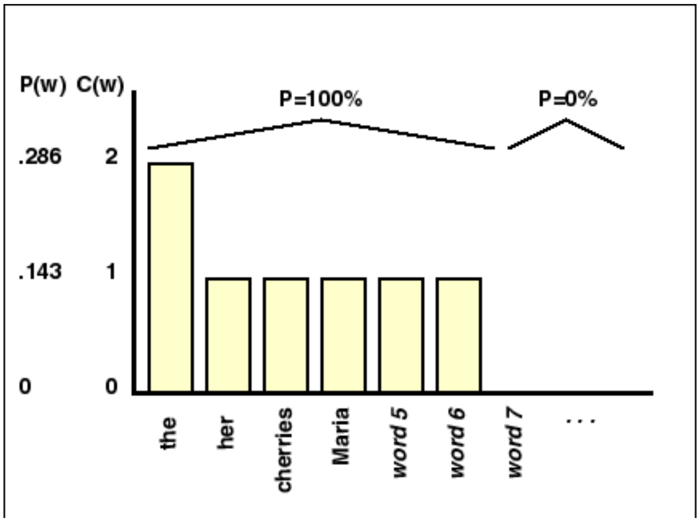
from test data: "[In person, she was] inferior to *both* [sisters.]"

Instances in the Training Corpus:

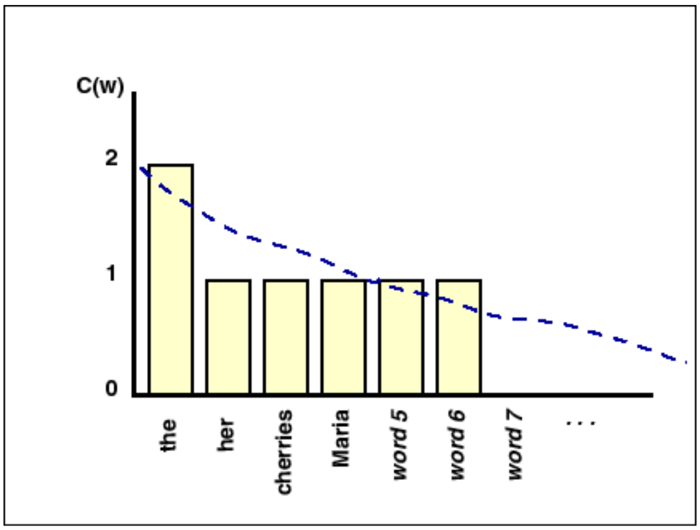
"inferior to ________"



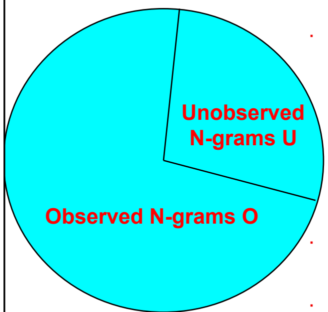
Maximum Likelihood Estimate:



Actual Probability Distribution:



Back-off Scheme(1): discounting



- How to estimate probability of an n-gram never observed ?

$$\text{For O set}: \; p(w|h) = \frac{N(hw)}{N(h)}$$

$$\text{For U set}: \; p(w|h) = 0$$

- Discounting is related to the zero-frequency estimation problem
- Discounting: discount probability mass in set O, and re-distribute to set U.
  - For set O: discounted probability

Q1: how much to discount ?

$$p^*(w|h) \quad (0 \le p^*(w|h) \le p(w|h))$$

- Total discounted mass: the zero-frequency probability

Q2: how to distribute the total discounted mass among all unobserved n-grams?

$$\lambda(h) = 1 - \sum_w p^*(w|h) \quad (\text{note} \sum_w p(w|h) = 1)$$

Back-off Scheme(1): discounting based on MAP Estimation

- Laplace's law (Floor discounting)
  - MAP estimation when setting uniform priors:

$$p_{FL}(w|h) = \frac{N(hw)+1}{N(h)+|V|}$$

  - Total discounted mass:

$$\lambda(h) = 1 - \sum_w p_{FL}(w|h) = \frac{N_0(h)}{N(h)+|V|}$$

total number of unobserved n-grams with the history h

  - Laplace's law usually over-discounts in LM estimation

- Lidstone's law

$$p_{lid}(w|h) = \frac{N(hw)+\varepsilon}{N(h)+|V|\cdot\varepsilon}$$

$$\lambda_{lid}(h) = 1 - \sum_w p_{lid}(w|h) = \frac{N_0(h)\cdot\varepsilon}{N(h)+|V|\cdot\varepsilon}$$

Back-off Scheme(1): Good-Turing Discounting (I)

- Good-Turing discounting: discount n-gram counts directly.
- *r*: frequency (occurring r times)
- *N_r*: total number of distinct n-grams occurring exactly *r* times.
- Good-Turing discounting rule:

$$r^* = (r+1)\frac{E(N_{r+1})}{E(N_r)} \quad (< r)$$

- Total probability mass reserved for unseen n-grams:

$$\lambda(h) = \frac{E(N_1)}{N(h)}$$

- How to calculate expectation *$E(N_r)$*?

r	*N_r*
0	212,522,973
1	138,741
2	25,413
3	10,531
4	5,997
5	3,565
6	2,486
7	1,754
8	1,342
…	…
1366	1
1917	1
2233	1
2507	1

Back-off Scheme(1): Good-Turing Discounting (II)

- How to get *$E(N_r)$?*
  - Directly use N_r to approximate the expectation.
    - Only adjust low frequency words (to say, r<=10)
    - No need to adjust high frequency words (r>10)
  - Fit all observed *(r,Nr)* to a function *S*, then use the smoothed value *S(r) as* the expectation*.*
    - Usually use hyperbolic function

$$E(N_r) = S(r) = a \cdot b^r \, (\text{with b} < \text{-1})$$

- Good-Turing estimate is *r*/N(h).*
- Re-normalize to a proper prob dist

r	r*	N_r
0	0.0007	212,522,973
1	0.3663	138,741
2	1.228	25,413
3	2.122	10,531
4	3.058	5,997
5	4.015	3,565
6	4.984	2,486
7	5.96	1,754
8	6.942	1,342
…	…	…
1366	1366	1
1917	1917	1
2233	2233	1
2507	2507	1

Back-off Scheme(2): Re-distributing methods

- Uniform re-distributing: the total discounted mass (a.k.a. the zero-frequency probability) *λ(h)* is uniformly distributed to all unseen ngrams.

- Katz's recursive re-distributing: the total discounted mass *λ(h)* is distributed over all unobserved events proportionally to a less specific distribution *p(w|h')*.
  - Build unigram *p(w)*, distribute *λ(h)* uniformly.
  - Build bi-gram *p(w|w')*, distribute *λ(h)* over all unseen *w* proportionally to *p(w).*
  - Build tri-gram *p(w|w'w''), distribute λ(h)* proportional to *p(w|w").*
  - So on so forth.

Katz's Back-off LM Scheme

- Recursively build from unigram → bi-gram → tri-gram →…
- Use Good-Turing method to discount low frequency events ONLY, say r≤k (k=6). No discounting for high frequency events, say, r>k.
- The total discounted probability mass is re-distributed to all unseen events proportional to their probabilities calculated from 1 level lower n-gram LM.

$$p_{Katz}(w\,|\,h) = \begin{cases} r / N(h) & \text{if } r > k, r = N(hw) \\ d_r \cdot r / N(h) & \text{if } 0 < r \le k \\ \alpha_h \cdot p(w\,|\,h') & \text{if } r = 0 \end{cases}$$

where $d_r = \dfrac{\frac{r^*}{r} - \frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}$ and $\alpha_h = \dfrac{1 - \sum_{w:r>0} p_{Katz}(w\,|\,h)}{1 - \sum_{w:r>0} p(w\,|\,h')}$

Back-off Scheme(3): Other Simple Discounting methods

- **Absolute discounting: all non-zero frequencies are discounted by a small constant, the total discounted mass is uniformly distributed over unseen events:**

$$p_{abs} = \begin{cases} (r-\delta)/N(h) & \text{if } r>0 \\ (|V|-N_0)\delta/N_0 N(h) & \text{otherwise} \end{cases}$$

- **Linear discounting: all non-zero frequencies are scaled by a constant slightly less than one, uniformly re-distribute.**

$$p_{ld} = \begin{cases} (1-\alpha)\cdot r/N(h) & \text{if } r>0 \\ C\cdot\alpha/N_0 N(h) & \text{otherwise} \end{cases}$$

Interpolation Scheme

- **Simple linear interpolation of several ML n-grams:**

$$p(w\,|\,w'w'') = \varepsilon_1\cdot p^{ML}(w) + \varepsilon_2\cdot p^{ML}(w\,|\,w'') + \varepsilon_3\cdot p^{ML}(w\,|\,w'w'')$$
$$\text{with } 0\le\varepsilon_1,\varepsilon_2,\varepsilon_3\le 1 \text{ and } \varepsilon_1+\varepsilon_2+\varepsilon_3=1.$$

- **General linear interpolation: weights are a function of history.**

$$p(w\,|\,h) = \sum_{i=1}^{K}\lambda_i(h)\cdot p_i(w\,|\,h)$$
$$\text{with } 0\le\lambda_i(h)\le 1 \text{ and } \sum_{i=1}^{K}\lambda_i(h)=1.$$

  - **Weights must be tied based on equivalence classes of *h*.**
- **How to estimate interpolation weights?**
  - **Held-out method: split training data to two parts; one for LMs and the other for weights.**
  - **Cross-Validation: split data into *N* parts; estimate LM's from any (N-1) parts, estimate weight from the other; rotate N times; average all estimates for the final results.**

Interpolation Weights Estimation

- All interpolation weights are estimated from the held-out training sample $W_h=\{w_1,w_2,\ldots,w_T\}$ by means of the EM algorithm.
- For simple interpolation:

$$\lambda_i^{(n+1)} = \frac{1}{T}\cdot\sum_{t=1}^{T}\frac{\lambda_i^{(n)}\cdot p_i(w_t)}{\sum_j \lambda_j^{(n)}\cdot p_j(w_t)}$$

- For general linear interpolation:

$$\lambda_i^{(n+1)}(h) = \frac{1}{T}\cdot\sum_{t=1}^{T}\frac{\delta(h_t=h)\cdot\lambda_i^{(n)}(h)\cdot p_i(w_t\mid h)}{\sum_j \lambda_j^{(n)}(h)\cdot p_j(w_t\mid h)}$$

where

$$\delta(x)=\begin{cases}0 & \mathrm{x}\neq 0\\ 1 & \mathrm{x}=0\end{cases}$$

Class-based N-Gram LM

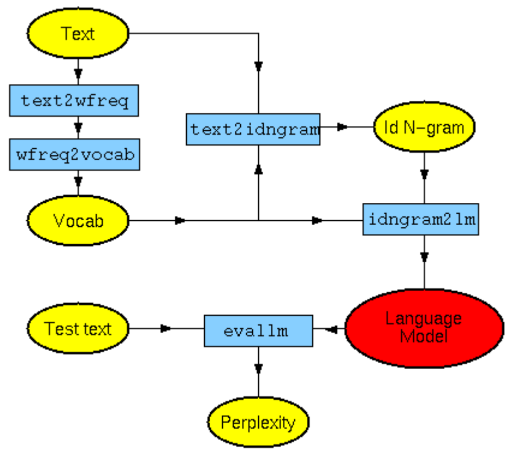
- To reduce LM parameters, group words into classes:
  - Based on morphology, e.g. part-of-speech, etc.
  - Based on semantic meaning: city name, time, number, etc.
- There exist many automatic word clustering algorithms.
- In class-based n-gram LM, each Markov state is a word class, conditional probabilities all depend on word classes rather than individual words.
- Greatly reduce parameter # → require less training data.
- Class-based trigram model:

$$p(w_i\mid w_{i-2}w_{i-1}) = \Pr(w_i\mid C(w_i))\cdot p(C(w_i)\mid C(w_{i-2})C(w_{i-1}))$$

where $C(w_i)$ means the class w_i belongs to.

N-gram LM Toolkit

- No need to write your own program to do all above-mentioned n-gram LM computation.



- A free Statistical Language Model Toolkit is available:

http://mi.eng.cam.ac.uk/~prc14/toolkit.html

Other Issues

- LM Interpolation
  - Task-independent (TI) text data:
    - Purchasing, Web Crawling, etc.
  - Task-dependent (TD) text data: collecting.
- Rapid LM Adaptation
  - Topic adaptation
- Discriminative Training of LM
  - Convex Optimization: Linear programming (LP)
  - Discriminative Adaptation:
    - Constrained optimization → LP

Text Categorization

- Each category is modeled by a mixture of multinomial model (MMM).

$$p(X_t \mid C_i) = c_t \sum_k \omega_{ik} \prod_d u_{ikd}^{m_{td}}$$

- Maximum likelihood Estimation:
  - EM Algorithm
- Discriminative Learning – Large Margin Estimation:
  - AM algorithm:
    - M-approx and E-approx

Discriminative Training

- Convex Optimization: Linear Programming

$$\begin{aligned}
&\min\ -\rho \\
&s.\,t. \\
&c_{tij} + \sum_d d^{u}_{ikd}\psi^{u}_{ikd} + d^{\omega}_{ik}\psi^{\omega}_{ik} - \sum_d d^{u}_{jk'd}\psi^{u}_{jk'd} - d^{\omega}_{jk'}\psi^{\omega}_{jk'} \succ \rho \quad (t_s) \\
&\sum_d e^{\psi^{u(0)}_{ikd}}\psi^{u}_{ikd} \le \sum_d e^{\psi^{u(0)}_{ikd}}\psi^{u(0)}_{ikd} \quad (ik_s) \\
&\psi^{u}_{ikd} \le \psi^{u(0)}_{ikd} + \xi \quad (ikd_s) \\
&\psi^{u}_{ikd} \ge \psi^{u(0)}_{ikd} - \xi \quad (ikd_s) \\
&\sum_k e^{\psi^{\omega(0)}_{ik}}\psi^{\omega}_{ik} \le \sum_k e^{\psi^{\omega(0)}_{ik}}\psi^{(0)}_{ik} \quad (i_s) \\
&\psi^{\omega}_{ik} \le \psi^{\omega(0)}_{ik} + \xi \quad (ik_s) \\
&\psi^{\omega}_{ik} \ge \psi^{\omega(0)}_{ik} - \xi \quad (ik_s)
\end{aligned}$$

Experimental Results: Text Classification Error Rate

- The USAA corpus:

Method	*Bank-HT*	*Bank-ASR*
Vector-based	06.84%	09.10%
NBCv1+DT	05.86%	09.09%
NBCv2+DT	05.54%	09.42%
soft margin LS MAM	***05.54%***	***08.82%***

- The RCV1 corpus:

		Training set	Test set
Multinomial (m=1)	MLE	n/a	7.5%
Mix of multinomial (m=6)	MLE-EM	n/a	5.5%
Mix of multinomial (m=6)	LME-AM	0.71%	3.9%