[Day-1] Introduction to Statistical Natural Language Processing

(Part V: Techniques for Improving Performance)

Keh-Yih Su and Jing-Shin Chang
kysu@bdc.com.tw, shin@nlp.csie.ncnu.edu.tw

(2002/8/17)

Behavior Design Corporation (BDC)
No. 5, 2F, Industrial East Road IV, Science-Based Industrial Park
Hsinchu, Taiwan 30077, R.O.C.

Department of Computer Science and Information Engineering
National Chi-Nan University
Puli, Nantou, Taiwan 545, R.O.C.

Day-1: Introduction to Statistical Natural Language Processing (mainly on Supervised Learning)

- Part I: Introduction (1)
  - Problems and Characteristics of Natural Language Processing
- Part II: Introduction (2)
  - What, When and Why Statistical Approach
- Part III: Basic Concepts and Background
  - Feature Space, Probability, Estimator, Stochastic Process, Data Set Classification, and Performance Measure
- Part IV: Typical Applications
  - Word Segmentation, Tagging, Selecting Parse Tree, Aligning Bilingual Corpus
- **Part V: Techniques for Improving Performance**
  - **Smoothing, Class-Based Model, Adaptive Learning, Tips for Checking**
- Part VI: Advanced Topics: SVM, ME
  - Support Vector Machine, Maximum Entropy Models
- Appendix: Related Techniques
  - Parameter Estimation, Fractional Factorial Experiment Design, Decision Tree

Part V: Techniques for Improving Performance

- Language Modeling
  - Feature Selection Methods
  - Class-based Modeling
- Parameter Smoothing
  - Deleted Interpolation
  - Good-Turing
  - Back-off
- Adaptive Learning
  - Discrimination Enhancement
    - Parameter Tying
  - Robustness Enhancement
  - An NLP Example
- Tips for Checking
  - Performance trends should be observed

Feature Selection

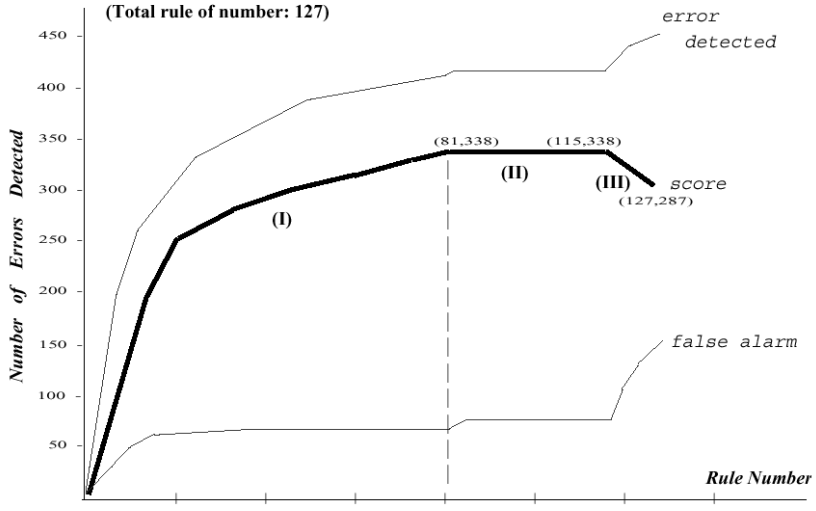
- Goal: select the best subset of *d* features which optimizes a criterion function from a large set of features.
  - Improve Performance:
    - Select the most discriminative features for processing
    - Eliminate irrelevant or noisy features to reduce their effects on performance
  - Reduce system resource required:
    - Reduce redundant information without degrading system performance
    - Reduce the dimension of the feature space and the size of the associated parameters with minimum performance degradation

Sequential Forward Selection [Devijver 82]

- Procedures:
  - Initially the feature set contains no feature.
  - Add one feature to the current feature set to form an enlarged feature set.
    - The one being selected is the one that maximizes some criterion function (e.g., accuracy rate) when used jointly with the current feature set.
  - Repeat until the feature set contains *d* features.
- Other variations: Sequential Backward Selection, Plus-l-take-away-r, etc. [Devijver and Kittler 82]

Example of Rule Selection with SFS [Liu 93]:

Score = (number of error detected)- (number of false alarm)
Rule number: based on the score of a rule (i.e., rule-ordering)
(Total rule of number: 127)

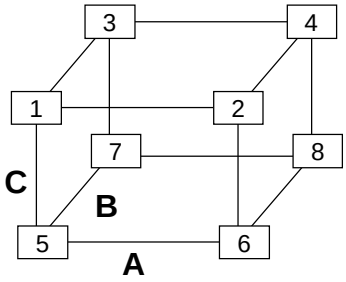


Fractional Factorial Experiment Design [Montgomery 2001]

- Test features one at a time is not efficient
  - Three factors need 4x4 = 16 runs
  - Interaction effect is still unknown
- Factorial Experimental Design
  - For k factors, the complete set includes 2 to the k runs. If k=3, it needs 8 runs.
- Fractional Factorial Experimental Design
  - Decide the total number of desired runs (i.e., select desired fraction)
  - Look up the table to find out the suitable generator and its associated combinations
  - Analyze the result to see if further runs are required
  - Add additional runs to enhance the resolution of interested cases
  - Rule of Thumb: don't spend over 25% of total resources in first stage

Fractional Factorial Experiment Design (Cont.)

- Effect of each factor and interaction
  - A = (2 + 4 + 6 + 8 – 1 – 3 – 5 – 7) / 4
  - B = (3 + 4 + 7 + 8 – 1 – 2 – 5 – 6) / 4
  - C = (1 + 2 + 3 + 4 – 5 – 6 – 7 – 8) / 4
  - AB = (1 + 4 + 5 + 8 – 2 – 3 – 6 - 7) / 4
  - AC = (2 + 4 + 5 + 7 – 1 – 3 – 6 - 8) / 4
  - BC = (3 + 4 + 5 + 6 – 1 – 2 – 7 - 8) / 4
  - ABC = (1 + 4 + 6 + 7 – 2 – 3 – 5 - 8) / 4



- However, still too many runs required when k is large
  - Total 2 to the k runs. If k = 8, it needs 256 runs
  - Fractional Factorial design is recommended

Class-Based Modeling

- Goal:
  - To reduce the number of parameters such that the parameters can be estimated more reliably.
  - To improve statistical language modeling:
    - to provide a partial solution in dealing with the estimation of parameters for unseen events.
- Example: Word Bi-gram model with vocabulary size of "100,000"

$$P(w_1, w_2, \cdots, w_n) \approx P(w_n \mid w_{n-1}) \times P(w_{n-1} \mid w_{n-2}) \times \cdots \times P(w_2 \mid w_1) \times P(w_1)$$

- Number of possible parameters: $\left(10^5\right)^2 = 10^{10}$

Class-Based Modeling (Cont.)

- Class-based model: 100 classes, two classes per word

$$\begin{aligned} & P(w_1, w_2, \cdots, w_n) \\ &= \sum_{t_1, t_2, \cdots, t_n} P(w_1, w_2, \cdots, w_n, t_1, t_2, \cdots, t_n) \\ &= \sum_{t_1, t_2, \cdots, t_n} P(w_1, w_2, \cdots, w_n \mid t_1, t_2, \cdots, t_n) \times P(t_1, t_2, \cdots, t_n) \\ &\approx \sum_{t_1, t_2, \cdots, t_n} \prod_{i=1}^{n} P(w_i \mid t_i) \times P(t_i \mid t_{i-1}) \end{aligned}$$

- Number of parameters: (100,000 x 2) + (100 x 100) = 210,000

Parameter Smoothing

- WHY: Not Enough Data to Train Parameters
  - IBM-Model-1 use 81 million parameters from 40,000 sentence pairs, about 800,000 words in each language.
  - In general, if the size of the training set is over 10 x N_p then we can achieve good generalization in the test set, where N_p is the number of parameters.
  - When data is not enough, smoothing technique must be used to achieve robustness
- Parameter Smoothing Techniques:
  - Adding a flattening constant [Fienberg 72, Su 89]
  - Clipping with a floor value
  - Deleted Interpolation [Bahl 83]
    - Use the information from correlated and less restricted parameters, thus better estimated.
  - Good-Turing estimate [Good 53]
  - Back-Off procedures [Katz 87]
  - Maximum Entropy (to be described in Part VI)

Smoothing Techniques (1)

- Adding a flattening constant [Fienberg & Holland 72, Su 89]

$$P(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n) + \lambda}{\sum (C(w_1 \cdots w_n) + \lambda)} \qquad (\lambda = 0.5 \text{ or } 1)$$

w	C(w)	C(w)+0.5	P
Book	5	5.5	5.5/(8+1.5)
Box	3	3.5	3.5/(8+1.5)
Bank	0	0.5	0.5/(8+1.5)

Pmin=0.002

- Clipping with a floor value

$$P(w_1 \cdots w_n) = \max\left(\frac{C(w_1 \cdots w_n)}{\sum_{w_1^n} C(w_1 \cdots w_n)}, p_{\min} \right)$$

	f(w)	P
Book	0.3	0.3
Box	0.05	0.05
Bank	0.0001	0.002

Smoothing Techniques (2)

- Deleted Interpolation [Bahl 83]
  - Switching Model

$$P(H_i \mid x) = \sum \alpha P(H_i) P(x \mid H_i) = \sum \lambda_i P(x \mid H_i)$$

  - Linear interpolation

$$P(w_i \mid w_{i-1}, w_{i-2}) = \lambda_1 P_1(w_i) + \lambda_2 P_2(w_i \mid w_{i-1}) + \lambda_3 P_3(w_i \mid w_{i-1}, w_{i=2}), \sum \lambda_i = 1$$

  - Log-linear interpolation

$$\log P(w_i \mid w_{i-1}, w_{i-2}) = \lambda_1 \log P_1(w_i) + \lambda_2 \log P_2(w_i \mid w_{i-1}) + \lambda_3 \log P_3(w_i \mid w_{i-1}, w_{i=2}), \quad \sum \lambda_i = 1$$

Smoothing Techniques (3)

- Good-Turing Method [Good 53]:

$$C^*(x) = r^* \approx (r+1)\frac{N_{r+1}}{N_r} \qquad P^*_{GT}(X = x_i) \approx \frac{C^*(X = x_i)}{\sum_j C^*(X = x_j)}$$

  - Where *C(x)=r* is the number of occurrence of event *X=x*; *C*(x)=r** is the estimated frequency count that *x* would occur, and N_r is the number of events that occurs *r* times.
  - For better performance, *r* can be first smoothed
  - N_0 is estimated by linear extrapolation of Log-Probability plot (Zipf's law), or just use the number of unseen events in the cases of close-set for simplicity
  - Example:

$$P(\text{Mary} \mid \text{John, likes}) = \frac{C^*(\text{John, likes, Mary})}{\sum_X C^*(\text{John, likes, X})}$$

Smoothing Techniques (4)

- Better approach to compute conditional probability with Good-Tuning smoothing: Computing from *unconditional* probabilities
  - Instead of directly smoothing those conditional probabilities, perform Good-Tuning on unconditional probability factor [Jelinek 97], then evaluate the associated conditional probabilities from those unconditional probabilities

$$\hat{P}(w_i \mid w_{i-1}, w_{i-2}) = \frac{P^*_{GT}(w_i, w_{i-1}, w_{i-2})}{\sum_w P^*_{GT}(w, w_{i-1}, w_{i-2})}$$

Smoothing Techniques (5)

- Back-off Method [Katz 87]:
  - Recursively reduce contextual window if the frequency is zero for larger window size

$$P_{BF}\left(c_i \mid c_{i-m}^{i-1}\right) = \begin{cases} P_{GT}\left(c_i \mid c_{i-m}^{i-1}\right) & C\left(c_{i-m}^{i-1}\right) > 0, C\left(c_{i-m}^{i-1}, c_i\right) > 0 \\ \alpha\left(c_{i-m}^{i-1}\right) \cdot P_{BF}\left(c_i \mid c_{i-(m-1)}^{i-1}\right) & C\left(c_{i-m}^{i-1}\right) > 0, C\left(c_{i-m}^{i-1}, c_i\right) = 0 \\ P_{BF}\left(c_i \mid c_{i-(m-1)}^{i-1}\right) & C\left(c_{i-m}^{i-1}\right) = \sum_x C\left(c_{i-m}^{i-1}, x\right) = 0 \end{cases}$$

Smoothing Techniques (6)

- Back-off Example [Su et al., 1996]:
  - Unseen events:

$$C(n,a,pron)=0$$
$$C(n,a,quan)=0$$
$$C(n,a,conj)=0$$

  - Bigram Probabilities:

$$P(pron|a)=0.1$$
$$P(quan|a)=0.2$$
$$P(conj|a)=0.3$$

  - Discounted probability mass from seen events:

$$1-\sum_{pos_i\neq\{pron,quan,conj\}} P(pos_i|a,n)=0.03$$

Smoothing Techniques (7)

- Back-off Example (Cont.):
  - Back-off estimates (proportional to bi-gram probabilities)

$$P_{BF}(pron|a,n)=0.03\times\frac{P_{GT}(pron|a)}{P_{GT}(pron|a)+P_{GT}(quan|a)+P_{GT}(conj|a)}=0.005,$$
$$P_{BF}(quan|a,n)=0.03\times\frac{P_{GT}(quan|a)}{P_{GT}(pron|a)+P_{GT}(quan|a)+P_{GT}(conj|a)}=0.010,$$
$$P_{BF}(conj|a,n)=0.03\times\frac{P_{GT}(conj|a)}{P_{GT}(pron|a)+P_{GT}(quan|a)+P_{GT}(conj|a)}=0.015.$$

- Good-Tuning estimates (equally likely) P_{GT} = 0.03/3 = 0.01

Smoothing Effect

- Training Set Performance
  - After smoothing, as the parameter set will deviate from that obtained from MLE, the likelihood value in the training set will decrease
  - Also, although not guaranteed, the error rate usually increase as it has deviated from the most fitted model
- Test Set Performance
  - After smoothing, the likelihood value in the testing set usually increases as those unseen events have been covered
  - Also, the error rate generally decreases as those unseen events have been taken care

Discrimination and Robustness (1)

- Traditional statistical approaches search the parameters that can fit the given training data set, under the given model, as closely as possible
  - If MLE is adopted, we try to fit the probabilistic distribution as closely as possible
  - The fitness is measured by its associated likelihood value
  - Therefore, the recognition problem is indirectly pursued by the parameters estimation approach
- However, what we really care is the error rate
  - For example: P(f | Mi) and P(f | Mj). Assume true values are 0.81 and 0.79, you still make error if 0.795 and 0.805 are estimated; however, you get correct answer if 0.87 and 0.72 are used instead
  - Therefore, it is actually the correct ranking order of the desired candidate, not the parameter estimation error, that we really care.
  - Maximizing the likelihood for the training set thus doe not imply minimizing the error rate of the training set

Discrimination and Robustness (2)

- Motivation for the above traditional approaches:
  - Bayesian classifier is the minimum error rate classifier, if the true density function is available
  - If we approach the true density function by refining the model (choosing better form and adopting better estimation method), we can obtain the best performance

Discrimination and Robustness (3)

- Drawbacks of the above approach:
  - The form of the true density function is not really known; and even the true density function is given, it might not be manageable (e.g., don't have enough data to support the associated complexity)
  - A feature that fits every class equally well adds nothing to our performance. It is the competition among different candidates, not how well each candidate performs, that really matters. However, each model is independently trained only on its own data set
  - Each data is weighted equally during training. However, the deviation of the estimated density function off the decision boundary does not affect the performance (so why we should bother about that?); and those data that are far away from the decision boundary might even bring in adverse effect

Discrimination and Robustness (4)

- Why not directly minimize error rate?
  - Although Bayesian classifier gives the best decision boundary, it is not the only way to find that
  - Any classifier that can find the best decision boundary also deliver the best performance
  - Only the training data that are near the decision boundary (i.e., the classes separation plane) will really affect performance; those data that are far away from the decision boundary might even bring in adverse effect. They should be weighted differently.
  - Discrimination capability can be enhanced if we train the model by jointly considering data from various competing classes
  - Directly pursue the minimum error rate (in the training set only) is called Discriminative Training [Juang 92]

Discrimination and Robustness (5)

- Discrimination issue mainly addresses the criterion mismatch problem by directly pursuing the minimum error rate criterion
- Moreover, what we really care is the error rate in the testing set, not in the training set
  - Can the ranking characteristics learned from the training set be preserved in the testing set?
  - Minimizing the error rate in the training set does not imply we can also minimize the error rate in the testing set
- Robustness issue mainly addresses the problem of statistical parameters variation between the training set and the testing set

Discrimination Enhancement [Su and Lee 1994]

- Find a discrimination function $g_j\left(C_j;O'_j,\lambda'_j\right)$ which can well preserve the correct ranking orders.
  - What we want is:

$$P(\arg\max_j P_{True}\left(C_j \mid O;\lambda_j\right) = \arg\max_j g_j(C_j;O'_j,\lambda'_j)) \rightarrow 1.0$$

  - Find a measuring function $g_j\left(C_j;O'_j,\lambda'_j\right)$ for the transformed observation vector $\boldsymbol{O_j'}$ and adjusted parameter set λ_j' that can maximize the probability of getting the correct ranks

Discrimination Enhancement (Cont.)

- We don't know how to directly pursue $g_j\left(C_j;O'_j,\lambda'_j\right)$;

  therefore, we still start from Bayses' Classifier
- Three ways to search for a good discrimination function, starting from a preliminary parameter $\hat{\lambda}$:
  - (1) change $\hat{\lambda} \rightarrow \lambda'$
  - (2) transform $O \rightarrow O'$
  - (3) adopt a good measuring function (e.g., $P(\cdot) \rightarrow g_j(\cdot)$).

Adaptive Learning (1)

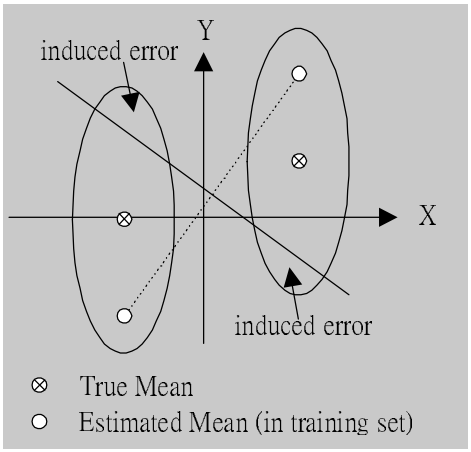
- Why adaptive learning
  - Discrimination Issues
    - The assumed model might not be the real one (e.g., modeling bi-modes distribution by only one Gaussian). The distortion of decision boundary (resulted from adopting inappropriate model) can be compensated by twisting its associated parameter
    - Since the training data is limited, the estimation error is significant. The decision boundary affected by the estimation error can be compensated by twisting its associated parameter
    - Only data points near the decision boundaries would affect the performance. The adverse effect from those outliers can be lessened by weighting various data differently during adaptive learning process
    - In short, the criterion of maximizing likelihood is not equivalent to maximizing the recognition rate in the training corpus. The criterion mis-match can be compensated by adjusting parameters under the same form

Adaptive Learning (2)

- Why adaptive learning (Cont.)
  - Robustness Issues
    - Statistic variation between training corpora and unseen text frequently is not considered in parameter estimation
    - Hence, minimizing the error rate in the training corpora is not equivalent to maximizing the recognition rate in unseen text
    - The parameters should be adjusted to pursue the decision boundary that can maximize separation between different classes
- Adaptive learning is required to adjust the estimated parameters according to misjudged instances or unreliably recognized instances

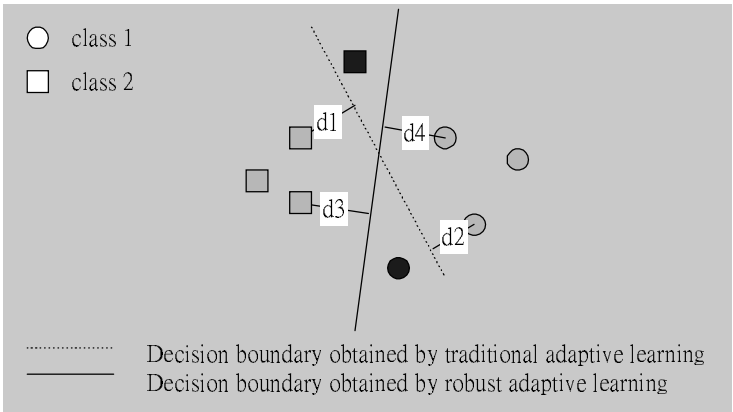
Select Robust Feature Set *via.* Subspace Projection

- Projecting observations into subspace to reduce error rate



Achieve Maximum Separation

- Maximum Separation Classification [Su and Lee 1994]
  - Green: Training Set; Red: Testing Set
  - Traditional -- Training Set is separated BUT testing set is NOT
  - Robustness -- Testing set is better improved by Maximum Separation



Adaptive Learning (Supervised) [Amari 1967]

- Directly minimizing error rate *via* gradient-descending search
- Basic Concepts:
  - Loss Function: the loss associated with an instance of recognition error
    - e.g., one recognition error ⇔ loss: 1
    - (or approximate it within the range [0...1], depending on how bad it was mis-recognized)

$$l(d) = \tan^{-1}\left(\frac{d}{d_0}\right), \qquad \begin{cases} d: \text{mis-classification distance} \\ d_0: \text{window size} \end{cases}$$

  - Risk Function: expected value of the loss function

$$R = E\left[l(d)\right] \approx \frac{1}{N}\sum_{i=1}^{N} l(d_i)$$

Adaptive Learning (Cont.)

- Basic Concepts (Cont.):
  - Minimizing risk function *via* gradient-descending
    - Approximate error function by an analytical loss function (such as arctan or sigmoid)
    - Find adjusting direction that might reduce risk: adjust the parameter vector in the *reverse* direction of the gradient of the Risk function ($-\nabla R$), so that the adjustment leads to least risk
    - Adjust parameter set iteratively: by adjusting a small step size of the parameter vector iteratively; the risk will then be reduced in *average*

Robustness Enhancement [Su and Lee 1994]

- Enlarge the inter-cluster distance and reduce intra-cluster variance to achieve maximum separation in the feature space
- Discard unreliable features (projection into subspace)
- Enlarge the margin between the correct analysis and the competing candidates in its confusing set
- Force to train the same model for the non-discriminative part
  - Eliminate the possible variation introduced from each individual model
  - For example, Chinese characters of knife, blade, and strength
  - Another example, vowel part in English E-Set
- Adopt robust estimator (e.g., median is more robust than mean)

Learning Procedure (1)

- **Initialization:** Initialize the parameters with maximum likelihood estimation (+ smoothing).
- **Calculate the miss-recognition distance:** Let the highest two scores and the correct score be

$$^{1}SC_z, ^{2}SC_z, ^{c}SC_z, \quad z \in \{syl, lex, syn, sem\}$$

then the distance *d* for incorrect recognition is defined as follows:

$$d = \begin{cases} ^{1}SC_z - ^{c}SC_z; & \text{if error} \\ ^{1}SC_z - ^{2}SC_z; & \text{if correct \&} \ \dfrac{^{1}SC_z - ^{2}SC_z}{\max\left\{\left|^{1}SC_z\right|, \left|^{2}SC_z\right|\right\}} < \beta\,\% \end{cases}$$

Learning Procedure (2)

- Adjust the parameters:
  - Decide the amount of adjustment
    - A loss *l(d),* which is a function of the distance *d*, is defined for miss-recognition.
    - The amount of adjustment of parameters $\Delta\Lambda^{(t)}$ in the *t*-th iteration is determined such that the risk function *R=E*[*l*(*d*)] (the expected loss) decrease:

$$R = E\left[l(d)\right] \approx \frac{1}{N}\sum_{i=1}^{N} l(d_i), \quad l(d) = \tan^{-1}\left(\frac{d}{d_0}\right), \quad l'(d) = \frac{d_0}{d_0^2 + d^2}$$

$$\Lambda^{(t+1)} = \Lambda^{(t)} + \Delta\Lambda^{(t)}$$

$$\Delta\Lambda^{(t)} = -\varepsilon(t) U \nabla R$$

    - $\varepsilon(t)$ is the learning rate function which is a monotonically-decreasing function of the iteration number *t*.
    - d_0 is a small positive constant.
    - *U* is a positive definite matrix controlling convergent speed of parameters.

Learning Procedure (3)

- Adjust the parameters (Cont.):
  - The parameters are adjusted such that the score of the correct candidate is increased while the score of the top rank candidate is decreased.
  - **Robustness enhancement:** the learning process continuously proceeds until $^{c}SC \geq {}^{2}SC+\delta$; that is, the margin between the correct analysis ($^{c}SC = {}^{1}SC$) and the second highest candidate exceeds a preset threshold δ.
  - The learning procedure only converges in mean, which means the *average risk* would decrease as the learning procedure proceeds. Performance oscillation with iteration index is frequently observed

Example of Learning

- Sentence: "Press the left button“
- Correct tag sequence: "v art *adj* n“
  - Score: ^{c}SC =S(v|Press)+S(art|the)+S(**adj**|left)*+S(n|button)
    +S(v|@)+S(art|v)+S(**adj**|art)*+S(n|**adj**)*
- Tag sequence with the highest score: "v art *v* n“
  - Score: ^{1}SC =S(v|Press)+S(art|the)+S(**v** |left)*+S(n|button)
    +S(v|@)+S(art|v)+S(**v** |art)*+S(n|**v**)*

Example of Learning (cont.)

- Parameters before learning

		Press	the	left	button	subtotal	total
candidate 1	@	v	art	v	n		
LS		0	0	-0.3	0	-0.3	-2.38
CS		-0.7	-0.52	-0.7	-0.16	-0.28	
candidate 2	@	v	art	n	n		
LS		0	0	-0.7	0	-0.7	-2.92
CS		-0.7	-0.52	-0.3	-0.7	-2.22	
candidate 3	@	v	art	adj	n		
LS		0	0	-0.52	0	-0.52	-2.42
CS		-0.7	-0.52	-0.52	-0.16	-1.9	

- Parameters after learning

		Press	the	left	button	subtotal	total
candidate 1	@	v	art	v	n		
LS		0	0	-0.35	0	-0.35	-2.51
CS		-0.7	-0.52	-0.74	-0.2	-2.16	
candidate 2	@	v	art	n	n		
LS		0	0	-0.7	0	-0.7	-2.92
CS		-0.7	-0.52	-0.3	-0.7	-2.22	
candidate 3	@	v	art	adj	n		
LS		0	0	-0.48	0	-0.48	-2.29
CS		-0.7	-0.52	-0.48	-0.11	-1.81	

  - LS: Lexical Score CS: Context Score @: beginning of sentence marker

Discrimination Enhancement: Parameter Tying

- Why Parameter Tying ?
  - Unseen (or rarely occurred) events could be smoothed before adaptive learning
  - BUT, they cannot be *adjusted* during adaptive learning (since they are not in annotated training corpus or seed corpus).
- Solution for reliable estimation: tied to other *highly correlated parameters*, so that adaptive learning process has more chance to adjust them
  - Tagging example [Lin et al. 95]: see next page
  - Another example: Parse Tree selection [Chiang, Lin, and Su, 96]

Parameter Tying (1)

- Example: POS Tagging [Lin 95]
  - 3-gram contextual probability (MLE): $P(c_i|c_{i-1},c_{i-2})=\frac{C(c_{i-2},c_{i-1},c_i)}{\sum_i C(c_{i-2},c_{i-1},c_i)}\equiv\frac{N_i}{D_i}$
    - D_i (denominator) < Q_d (threshold) $\Rightarrow$ insufficient sample size $\Rightarrow$ unreliably estimated
    - N_i (numerator) < Q_n $\Rightarrow$ not well trainable (i.e., has no good chance to be adjusted by adaptive learning)
  - If a probability is both *unreliably estimated* ($D_i<Q_d$) and *not well trainable* ($N_i< Q_n$) $\Rightarrow$ tied to reliable 2-gram probability:

Parameter Tying (2)

- POS Tagging (Cont.)
  - 3-grams with the same 2-gram context (c_i,c_{i-1}) are assigned with the same 2-gram contextual probability $p(c_i|c_{i-1})$

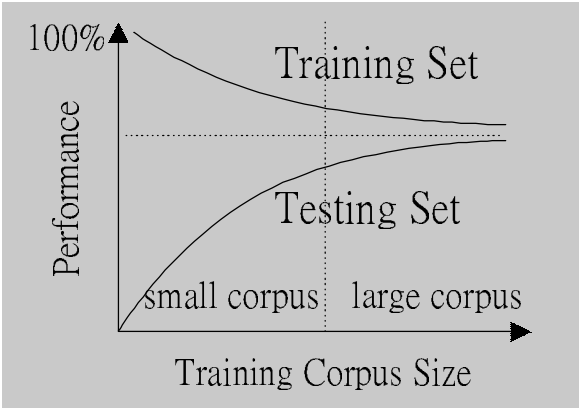
$$P(c_i \mid c_{i-1}, c_{i-2}) = \frac{C(c_{i-2}, c_{i-1}, c_i)}{\sum_{c_i} C(c_{i-2}, c_{i-1}, c_i)} \equiv \frac{N_i}{D_i}$$

Parameters tied to P(CD | IN)		
Parameter	Ni	Di
P(CD | IN, PN)	4	372
P(CD | IN, PPS)	0	24
P(CD | IN, PPSS)	0	21
P(CD | IN, WPS)	3	27

  - Qd and Qn are quite robust (in the ranges Qd=415~1245, Qn=1~10)
  - Tied 3-gram has slightly larger number of parameters than 2-gram, and much smaller than pure 3-gram; and has the best test set performance
  - A good compromise between estimation errors and modeling errors

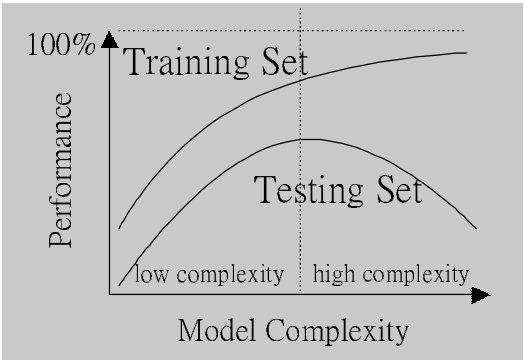
Performance Trends versus Training Corpus Size

- Problems of Corpora with Small size
  - Estimation Error: Training Set Performance /= Testing Set Performance

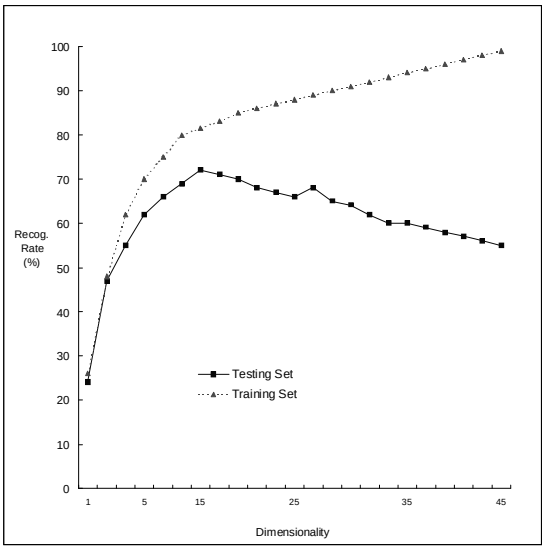


Performance Trends versus Module Complexity:

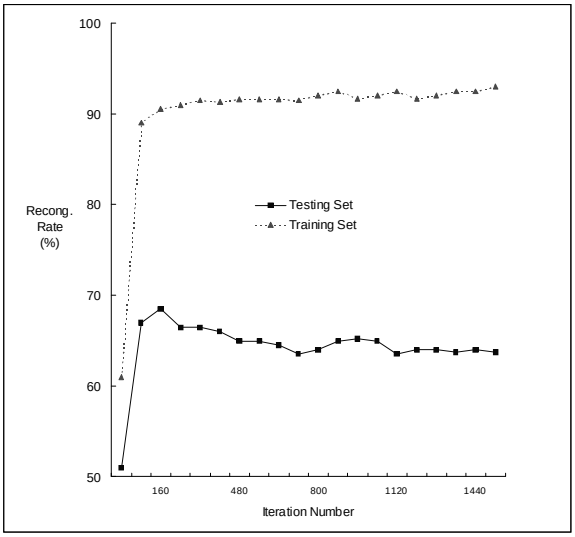
- Problems of Overfitting (Models with High Complexity)
  - Although increasing the Model Complexity can reduce the modeling error in the training set (thus reducing the error rate in the training set), it does not increase testing set performance without limit



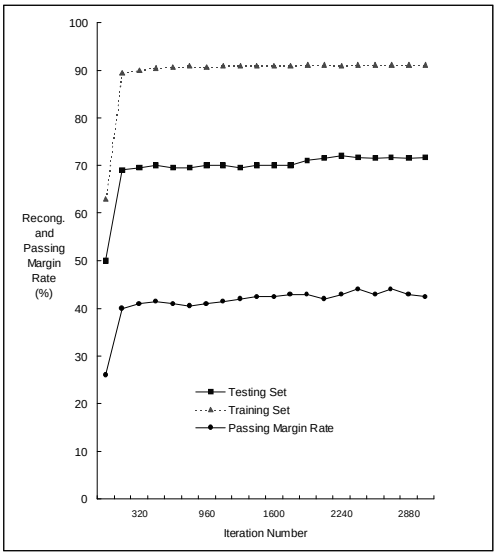
Overfitting:
E-Set Recognition Rate vs. Dimensionality



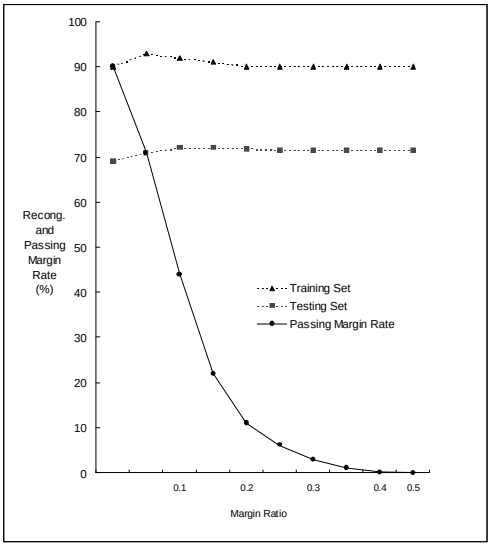
Overtuning:
E-Set Recognition Rate vs. Iteration Number



Robust E-Set Recognizer:
Recognition Rate vs. Iteration Number



Robust E-Set Recognizer (Cont.):
Recognition Rate vs. Margin Ratio



Suggested Checking Steps (1)

- How do you know that your program has been correctly coded?
  - Can you sense something wrong if it really has problems?
  - Do you know what the correct answers should look like?
  - Do you know the reasonable ranges of those estimated parameters?
  - What the behavior (or trend) should be if everything works as you expected?
  - Do you know how to distinguish coding error from modeling defect?
  - If all the answers are “No”, then it might be too early for you to write the code

Suggested Checking Steps (2)

- Suggested Checking Procedure
  - Don't rush into the testing set; check the training set first. You are not so lucky in most of the times
  - First test simple well-known model, with your program, in the training set (e.g., try context-free model with your context-sensitive program, or try artificial context-free data with your context-sensitive program)
  - Reduce the training set size (or model complexity) to see if the desired trend can be observed
  - Evaluate a few simple cases by hand, and check if the values generated by your program match them

Suggested Checking Steps (3)

- Get unsatisfactory training set performance?
  - You don't need to try the testing set; it usually should be even worse (if your evaluation program is correct)
  - If you get good training set performance, it might be a false phenomenon (it might be due to over-fitting or over-tuning)
  - However, if you still get bad training set performance, after the above checking procedure, don't doubt about it, just believe it
  - Check the including rate (with more candidates) first. Sometimes, the desired result just cannot be generated (e.g., correct tag is not listed in the dictionary)
  - Go back to check your model: Important features not adopted? Wrongly derived? Implicitly use incorrect assumptions (might be introduced during your simplifying the original model)?

Suggested Checking Steps (4)

- Good training set performance, but unsatisfactory testing set performance?
  - Simple one first: evaluate a few simple cases by hand, and check if the values generated by your program match them (too many places can go wrong, e.g., adopt different parameter set in testing set, etc.)
  - Check the including rate (with more candidates). Sometimes, the desired result just cannot be generated (e.g., have many unknown words, and always assign “noun” to them)
  - Check if it is over-fitting (too many unseen events)? Over-tuning? Testing set size too small (large performance measure variance)?

Suggested Checking Steps (5)

- Good training set performance, but unsatisfactory testing set performance (Cont.)?
  - Check if the characteristics of testing set deviates too much from the training set (e.g., the parameters are trained from the technical domain and then tested in a general domain, and the model is not robust enough to cover different styles)
  - Check if the characteristics of the adopted features are robust enough to be preserved in the testing set. Are features too primitive (e.g., using crossing-count to recognize Chinese hand written characters)?