6.891: Lecture 15 (October 29th, 2003)

Global Linear Models: Part II

Overview

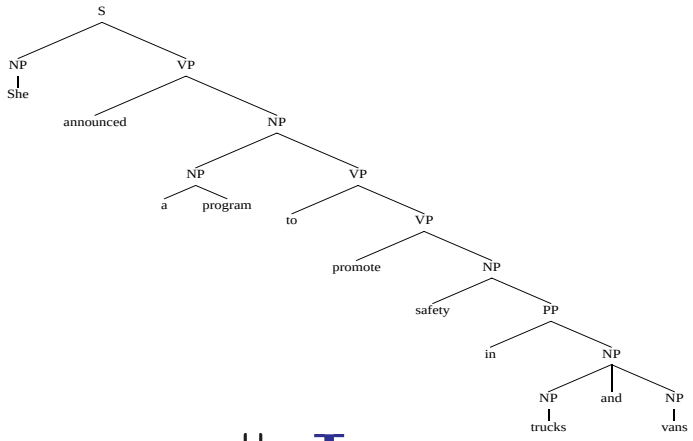
- Recap: global linear models
- A new algorithm: boosting
- Applications:
  - Parser reranking (from last lecture)
  - Natural language generation
- Efficient implementations of boosting

Three Components of Global Linear Models

- $\mathbf{\Phi}$ is a function that maps a structure (x, y) to a **feature vector** $\mathbf{\Phi}(x, y) \in \mathbb{R}^d$
- $\mathbf{GEN}$ is a function that maps an input x to a set of **candidates** $\mathbf{GEN}(x)$
- $\mathbf{W}$ is a parameter vector (also a member of $\mathbb{R}^d$)
- Training data is used to set the value of $\mathbf{W}$

Component 1: Φ

- Φ maps a candidate to a **feature vector** $\in \mathbb{R}^d$
- Φ defines the **representation** of a candidate



$\Downarrow \Phi$

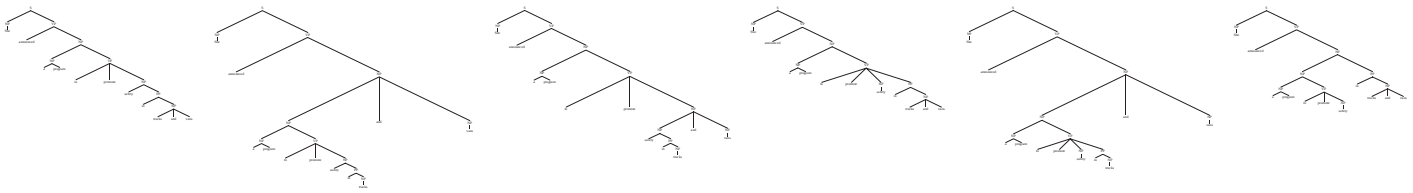
$\langle 1, 0, 2, 0, 0, 15, 5 \rangle$

Component 2: GEN

- **GEN** enumerates a set of **candidates** for a sentence

She announced a program to promote safety in trucks and vans

⇓ **GEN**



Component 2: GEN

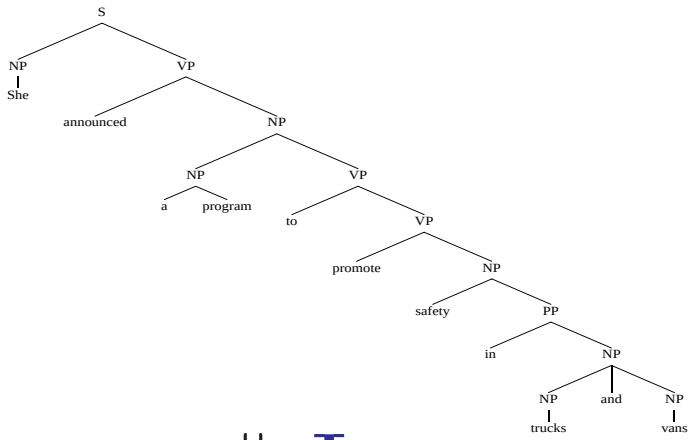
- **GEN** enumerates a set of **candidates** for an input x

- Some examples of how **GEN**(x) can be defined:
  - Parsing: **GEN**(x) is the set of parses for x under a grammar
  - Any task: **GEN**(x) is the top N most probable parses under a history-based model
  - Tagging: **GEN**(x) is the set of all possible tag sequences with the same length as x
  - Translation: **GEN**(x) is the set of all possible English translations for the French sentence x

Component 3: $\mathbf{W}$

- $\mathbf{W}$ is a **parameter vector** $\in \mathbb{R}^d$
- $\mathbf{\Phi}$ and $\mathbf{W}$ together map a candidate to a real-valued score

---



$$\Downarrow \mathbf{\Phi}$$

$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle$$

$$\Downarrow \mathbf{\Phi} \cdot \mathbf{W}$$

$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle \cdot \langle 1.9, -0.3, 0.2, 1.3, 0, 1.0, -2.3 \rangle = 5.8$$

Putting it all Together

- $\mathcal{X}$ is set of sentences, $\mathcal{Y}$ is set of possible outputs (e.g. trees)
- Need to learn a function $F : \mathcal{X} \rightarrow \mathcal{Y}$
- $\mathbf{GEN}$, $\mathbf{\Phi}$, $\mathbf{W}$ define

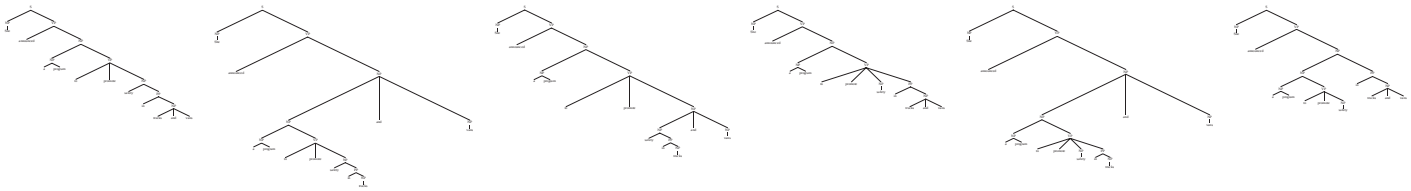
$$F(x) = \arg\max_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$$

Choose the highest scoring candidate as the most plausible structure

- Given examples (x_i, y_i), how to set $\mathbf{W}$?

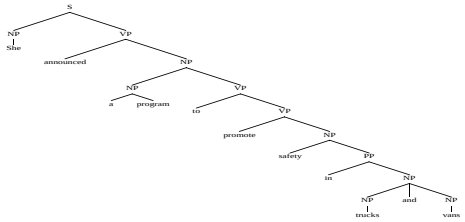
She announced a program to promote safety in trucks and vans

$\Downarrow$ **GEN**



$\Downarrow \Phi$	$\Downarrow \Phi$	$\Downarrow \Phi$	$\Downarrow \Phi$	$\Downarrow \Phi$	$\Downarrow \Phi$
$\langle 1, 1, 3, 5 \rangle$	$\langle 2, 0, 0, 5 \rangle$	$\langle 1, 0, 1, 5 \rangle$	$\langle 0, 0, 3, 0 \rangle$	$\langle 0, 1, 0, 5 \rangle$	$\langle 0, 0, 1, 5 \rangle$
$\Downarrow \Phi \cdot W$	$\Downarrow \Phi \cdot W$	$\Downarrow \Phi \cdot W$	$\Downarrow \Phi \cdot W$	$\Downarrow \Phi \cdot W$	$\Downarrow \Phi \cdot W$
13.6	12.2	12.1	3.3	9.4	11.1

$\Downarrow \arg\max$



Overview

- Recap: global linear models
- A new algorithm: boosting
- Applications:
  - Parser reranking (from last lecture)
  - Natural language generation
- An efficient boosting algorithm

Boosting

- Originated from a theoretical question in PAC learning (equivalence of weak and strong learning)
- Original algorithms gave strong theoretical results, but weren't particularly practical
- **AdaBoost** [Freund and Schapire 1995] was a practical boosting algorithm
- Numerous results have shown empirical success of AdaBoost
- AdaBoost was originally proposed for classification problems: many variants have followed, we'll focus on how to generalize the methods to Global Linear Models

A Central Question: Parameter Estimation

- **GEN**, **Φ**, **W** define

$$F(x) = \arg\max_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$$

- Given examples (x_i, y_i), how to set **W**?

Loss Functions

- A loss function measures how well the parameters fit the training data

- Up to now, the main loss function we've considered has been likelihood

  $$L(\Theta) = \sum_i \log P(y_i, x_i \mid \Theta) \quad \text{or} \quad L(\Theta) = \sum_i \log P(y_i \mid x_i, \Theta)$$

- Maximum likelihood estimation: choose

  $$\Theta_{ML} = \text{argmax}_\Theta L(\Theta)$$

  $\Rightarrow L(\Theta)$ is a measure of how well the parameters Θ "fit" the training data $(x_1, y_1) \ldots (x_n, y_n)$

The Training Data

- On each example there are several "bad parses":
  $z \in \mathbf{GEN}(x_i)$, such that $z \neq y_i$

- Some definitions:

  – There are n_i bad parses on the i'th training example
  (i.e., $n_i = |\mathbf{GEN}(x_i)| - 1$)

  – $z_{i,j}$ is the j'th bad parse for the i'th sentence

- We can think of the training data (x_i, y_i), and $\mathbf{GEN}$, providing a set of good/bad parse pairs

$$(x_i, y_i, z_{i,j}) \quad \text{for } i = 1 \ldots n, j = 1 \ldots n_i$$

Margins

- We can think of the training data (x_i, y_i), and $\mathbf{GEN}$, providing a set of good/bad parse pairs

$$(x_i, y_i, z_{i,j}) \quad \text{for } i = 1 \ldots n, j = 1 \ldots n_i$$

- The **Margin** on example $z_{i,j}$ under parameters $\mathbf{W}$ is

$$\mathbf{m_{i,j}}(\mathbf{W}) = \mathbf{\Phi}(x_i, y_i) \cdot \mathbf{W} - \mathbf{\Phi}(x_i, z_{i,j}) \cdot \mathbf{W}$$

- Some intuition:
  - The margin is the difference in score between a good parse and a bad parse
  - If the margin is positive (i.e., $\mathbf{m_{i,j}}(\mathbf{W}) > 0$) then the parameters $\mathbf{W}$ have correctly discriminated (x_i, y_i) from $(x_i, z_{i,j})$
  - If the margin is negative (i.e., $\mathbf{m_{i,j}}(\mathbf{W}) \leq 0$) then an error has been made

New Loss Functions

- Number of **ranking errors**

$$\text{Error}(\mathbf{W}) \quad = \quad \sum_{i,j} [[\mathbf{m_{i,j}}(\mathbf{W}) \leq 0]]$$

where $[[\pi]]$ is 1 if π is true, 0 otherwise

- **Exponential loss**

$$\text{ExpLoss}(\mathbf{W}) = \sum_{i,j} e^{-\mathbf{m_{i,j}}(\mathbf{W})}$$

where

$$\mathbf{m_{i,j}}(\mathbf{W}) = \mathbf{\Phi}(x_i, y_i) \cdot \mathbf{W} - \mathbf{\Phi}(x_i, z_{i,j}) \cdot \mathbf{W}$$

A New Parameter Estimation Method

- **Exponential loss**: $\text{ExpLoss}(\mathbf{W}) = \sum_{i,j} e^{-\mathbf{m_{i,j}}(\mathbf{W})}$
- We could pick parameters $\mathbf{W}^*$ as

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \text{ExpLoss}(\mathbf{W})$$

- Some properties:
  - Easy to do because ExpLoss is convex, differentiable
  - Intuition: minimizing ExpLoss encourages parameters to give positive margins on all examples
  - More justifiction: ExpLoss is an upper bound on the number of ranking errors

    $$\text{ExpLoss}(\mathbf{W}) \geq \text{Error}(\mathbf{W})$$

    (follows because $e^{-x} \geq [[x \leq 0]]$ for all x
  - Minimizing Error is at least NP-hard if $\text{Error}(\mathbf{W}) = 0$ is not possible

A New Parameter Estimation Method

- **Exponential loss**: $\text{ExpLoss}(\mathbf{W}) = \sum_{i,j} e^{-\mathbf{m_{i,j}}(\mathbf{W})}$
- We could pick parameters $\mathbf{W}^*$ as

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \text{ExpLoss}(\mathbf{W})$$

- A drawback:
  - Easy to "overfit" with this method
    especially if there are a large number of features
- An alternative:
  - Try to make good progress in minimizing ExpLoss,
    but keep most parameters $\mathbf{W}_k = 0$
  - This is a feature selection method: only a small number of features are "selected"
  - In a couple of lectures we'll talk much more about overfitting, and generalization

A Feature Selection Method

- **Exponential loss**: $\text{ExpLoss}(\mathbf{W}) = \sum_{i,j} e^{-\mathbf{m_{i,j}}(\mathbf{W})}$
- A definition:

$$\text{Upd}(\mathbf{W}, k, \delta) = \{\mathbf{W}_1, \mathbf{W}_2, \ldots \mathbf{W}_k + \delta, \ldots \mathbf{W}_d\}$$

  i.e., all parameter values stay the same, except for k'th parameter value

- An algorithm:

Input: training examples (x_i, y_i), function $\mathbf{GEN}$, representation $\mathbf{\Phi}$

Initialization: for all k, set $\mathbf{W}_k = 0$

Algorithm: for $t = 1 \ldots T$,

- Choose $(k^*, \delta^*) = \arg\min_{k,\delta} \text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta))$
- Set $\mathbf{W} = \text{Upd}(\mathbf{W}, k^*, \delta^*)$

Output: parameters $\mathbf{W}$

Input: training examples (x_i, y_i), function $\mathbf{GEN}$, representation $\mathbf{\Phi}$

Initialization: for all k, set $\mathbf{W}_k = 0$

Algorithm: for $t = 1 \ldots T$,

- Choose $(k^*, \delta^*) = \arg\min_{k,\delta} \text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta))$
- Set $\mathbf{W} = \text{Upd}(\mathbf{W}, k^*, \delta^*)$

Output: parameters $\mathbf{W}$

The basic idea:

- At beginning, all parameter values are 0
- At each step, choose a single parameter $\mathbf{W}_{k^*}$, and a weight δ^*, update the parameter by this weight
- Choose k^*, δ^* to make most impact in minimizing ExpLoss
- This is a greedy feature selection method:
  "greedy coordinate-wise gradient descent"

Overview

- Recap: global linear models
- A new algorithm: boosting
- Applications:
  - Parser reranking (from last lecture)
  - Natural language generation
- Efficient implementations of boosting

Reranking Approaches to Parsing

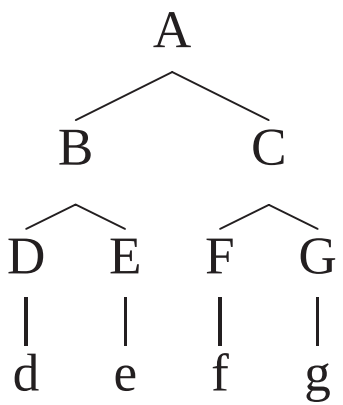
- Use a **baseline** parser to produce top N parses for each sentence in training and test data
  $\mathbf{GEN}(x)$ is the top N parses for x under the baseline model

- One method: use Model 2 to generate a number of parses
  (in our experiments, around 25 parses on average for 40,000 training sentences, giving $\approx$ 1 million training parses)

- **Supervision:** for each x_i take y_i to be the parse that is "closest" to the treebank parse in $\mathbf{GEN}(x_i)$

Practical Issues in Creating Φ

- Step 2: hash the strings to integers



$\Rightarrow$

HASRULE:A->B;C	54
HASRULE:B->D;E	118
HASRULE:C->F;G	14
HASRULE:D->d	10078
HASRULE:E->e	9000
HASRULE:F->f	1078
HASRULE:G->g	101

- In our experiments, tree is then represented as log probability under the baseline model, plus a sparse feature array:

$\mathbf{\Phi}_1(x, y) =$ log probability of (x, y) under the baseline model

$\mathbf{\Phi}_i(x, y) = 1$ for $i = \{54, 118, 14, 10078, 9000, 1078, 101\}$

$\mathbf{\Phi}_i(x, y) = 0$ for all other i

The Score for Each Parse

- In our experiments, tree is then represented as log probability under the baseline model, plus a sparse feature array:

$\mathbf{\Phi}_1(x, y) =$ log probability of (x, y) under the baseline model

$\mathbf{\Phi}_i(x, y) = 1$ for $i = \{54, 118, 14, 10078, 9000, 1078, 101\}$

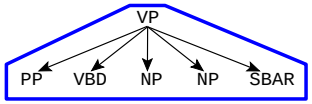
$\mathbf{\Phi}_i(x, y) = 0$ for all other i

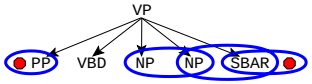
- Score for the tree (x, y):

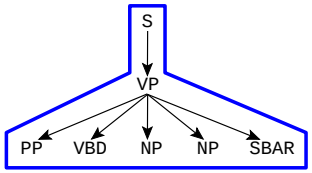
$$\begin{aligned} & \mathbf{\Phi}(x, y) \cdot \mathbf{W} \\ = & \sum_i \mathbf{\Phi}_i(x, y) \mathbf{W}_i \\ = & \mathbf{W}_1 \mathbf{\Phi}_1(x, y) + \mathbf{W}_{54} + \mathbf{W}_{118} + \mathbf{W}_{14} + \mathbf{W}_{10078} + \mathbf{W}_{9000} + \mathbf{W}_{1078} + \mathbf{W}_{101} \end{aligned}$$

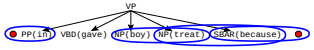
Last lecture we saw a number of features used in reranking experiments:

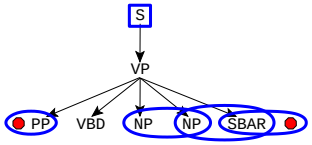
Result was $\approx 500,000$ unique features, ≈ 500 per parse

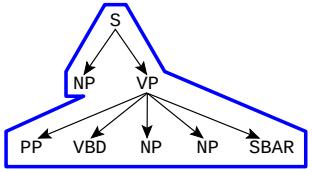


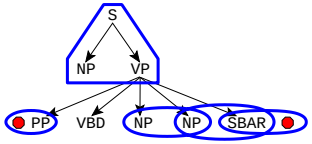


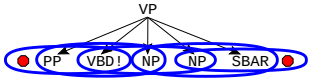


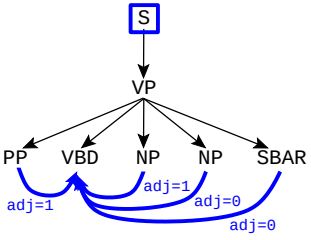


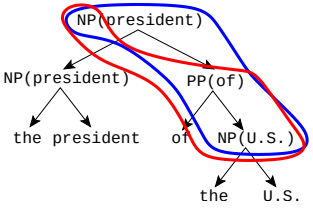












Experiments (from [Collins and Koo, in submission])

- At first step, set $\mathbf{W}_k = 0$ for all paremeters, then

$$\mathbf{W}_1 = \arg\min_{\delta} \text{ExpLoss}(\text{Upd}(\mathbf{W}, 1, \delta))$$

  i.e., optimize ExpLoss with respect to $\mathbf{W}_1$, the weight for the log probability under the baseline model

- Next steps: run feature selection for $100,000$ rounds

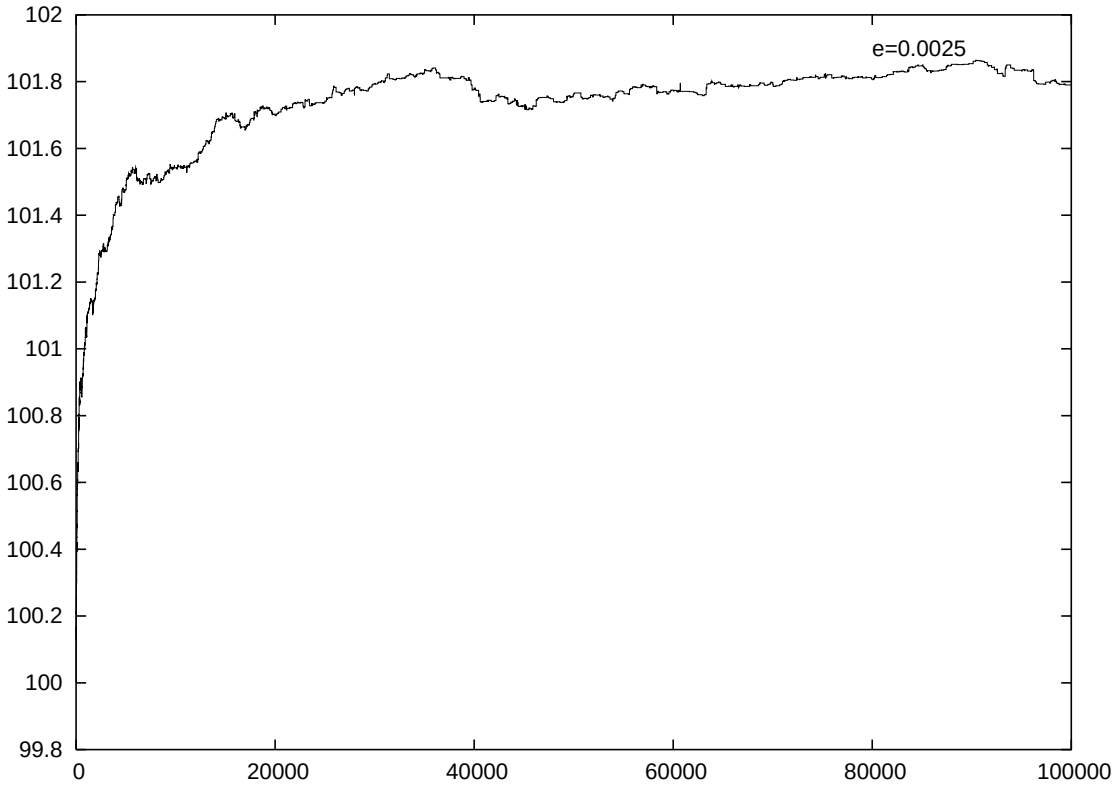
- Peak performance on development data is at around 90,000 rounds; at this point $\approx 10,000$ parameters have non-zero values

Results

MODEL	$\leq$ 100 Words (2416 sentences)				
	LR	LP	CBs	0 CBs	2 CBs
CH97	86.7%	86.6%	1.20	59.5%	83.2%
RA98	86.3%	87.5%	1.21	60.2%	—
CO99	88.1%	88.3%	1.06	64.0%	85.1%
CH00	89.6%	89.5%	0.88	67.6%	87.7%
ExpLoss	89.6%	89.9%	0.86	68.7%	88.3%

Results on Section 23 of the WSJ Treebank. **LR/LP** = labelled recall/precision. **CBs** is the average number of crossing brackets per sentence. **0 CBs,** 2 **CBs** are the percentage of sentences with 0 or $\leq$ 2 crossing brackets respectively. All the results in this table are for models trained and tested on the same data, using the same evaluation metric. CH97 = Charniak 1997, RA98 = Ratnaparkhi 1998, CH00 = Charniak 2000, CO99 = Collins 1999 Model 2 (the baseline model).

Around a 13% relative reduction in error



Graph showing the learning curve on development data for the optimal value for ϵ, i.e., $\epsilon = 0.0025$. The y-axis is the level of accuracy (100 is the baseline score), and the x-axis is the number of rounds of boosting.

Natural Language Generation
[Walker, Rambow and Rogati]

Input = text plan

implicit-confirm(orig-city:NEWARK) implicit-confirm(dest-city:DALLAS) implicit-confirm(month:9) implicit-confirm(day-number:1) request(depart-time)

Output = text

What time would you like to travel on September the 1st to Dallas from Newark?
Leaving on September 1st. What time would you like to travel from Newark to Dallas?
Leaving in September. Leaving on the 1st. What time would you, traveling from Newark to Dallas, like to leave?

An Example Dialogue

System1: Welcome. What airport would you like to fly out of?

User2: I need to go to Dallas.

System3: Flying to Dallas. What departure airport was that?

User4: from Newark on September the 1st.

System5: What time would you like to travel on September the 1st to Dallas from Newark?

The Text Planning Stage

Input = dialogue state

System1: Welcome. What airport would you like to fly out of?

User2: I need to go to Dallas.

System3: Flying to Dallas. What departure airport was that?

User4: from Newark on September the 1st.

System5: ???

Output = text plan

```
implicit-confirm(orig-city:NEWARK)
implicit-confirm(dest-city:DALLAS)
implicit-confirm(month:9)
implicit-confirm(day-number:1)
request(depart-time)
```

The Sentence Planning Stage

Input = text plan

```
implicit-confirm(orig-city:NEWARK)
implicit-confirm(dest-city:DALLAS)
implicit-confirm(month:9)
implicit-confirm(day-number:1)
request(depart-time)
```

Output = text

What time would you like to travel on September the 1st to Dallas from Newark?

A Boosting Method for Sentence Planning

Defining GEN:

- Starting point is a sentence plan with no "aggregation":

  Leaving on the 1st.
  Leaving in September.
  Going to Dallas.
  Leaving from Newark.
  What time would you like to leave?

- Several "merging" operations are possible:
  Merge, Merge-General, Soft-Merge, Soft-Merge-General, Conjunction, Relative-Clause, Adjective, Period

- Different sequences of merges lead to different sentence plans

- GEN is defined by randomly generating ≈ 20 sentence plans using the merging operations

A Boosting Method for Sentence Planning

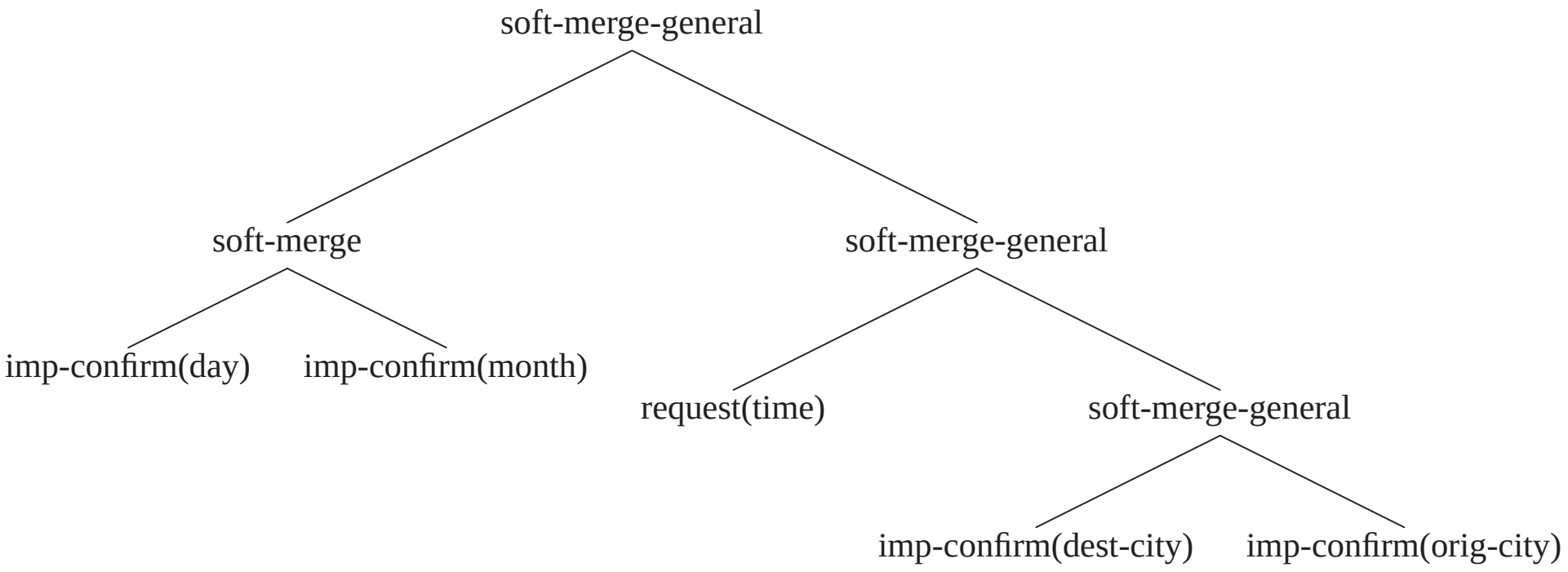
Gathering training data:

- Took 100 text plans, generated up to 20 sentence plans for each
- For each generated sentence, gave a rating on scale of $1 - 5$
- 97% of examples had at least one sentence rated 4 or higher

A Boosting Method for Sentence Planning

Defining Φ:

- Each member of $\mathbf{GEN}(x)$ can be represented as a tree
- Various features of these trees are chosen



Results

Method	Score
Upper bound	4.82
Boosting	4.56
Random	2.76

Overview

- Recap: global linear models
- A new algorithm: boosting
- Applications:
  - Parser reranking (from last lecture)
  - Natural language generation
- Efficient implementations of boosting

A Feature Selection Method

- **Exponential loss**: $\text{ExpLoss}(\mathbf{W}) = \sum_{i,j} e^{-\mathbf{m_{i,j}}(\mathbf{W})}$
- A definition:

$$\text{Upd}(\mathbf{W}, k, \delta) = \{\mathbf{W}_1, \mathbf{W}_2, \ldots \mathbf{W}_k + \delta, \ldots \mathbf{W}_d\}$$

  i.e., all parameter values stay the same, except for k'th parameter value

- An algorithm:

Input: training examples (x_i, y_i), function **GEN**, representation $\mathbf{\Phi}$

Initialization: for all k, set $\mathbf{W}_k = 0$

Algorithm: for $t = 1 \ldots T$,

- Choose $(k^*, \delta^*) = \arg\min_{k,\delta} \text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta))$
- Set $\mathbf{W} = \text{Upd}(\mathbf{W}, k^*, \delta^*)$

Output: parameters $\mathbf{W}$

A Feature Selection Method

- How do we compute

$$(k^*, \delta^*) = \arg\min_{k,\delta} \text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta)) \text{ ?}$$

- We'll assume that the features are *binary*, or *indicator functions*. i.e., each component $\mathbf{\Phi}_k(x, y)$ is 1 or 0

- For any $(x_i, y_i, z_{i,j})$ triple, there are three possibilities:
  - $\mathbf{\Phi}_k(x_i, y_i) = 1$, and $\mathbf{\Phi}_k(x_i, z_{i,j}) = 0$, so feature occurs on the **good** parse but not on the **bad** parse
  - $\mathbf{\Phi}_k(x_i, y_i) = 0$, and $\mathbf{\Phi}_k(x_i, z_{i,j}) = 1$, so feature occurs on the **bad** parse but not on the **good** parse
  - $\mathbf{\Phi}_k(x_i, y_i) - \mathbf{\Phi}_k(x_i, z_{i,j}) = 0$, so feature occurs on **both** parses or on **neither** parse

Some Definitions

- For each feature k, partition the training set into three sets:

$$\begin{aligned} A_k^+ &= \{(i,j) : \mathbf{\Phi}_k(x_i, y_i) - \mathbf{\Phi}(x_i, z_{i,j})_k = 1\} \\ A_k^- &= \{(i,j) : \mathbf{\Phi}_k(x_i, y_i) - \mathbf{\Phi}(x_i, z_{i,j})_k = -1\} \\ A_k^0 &= \{(i,j) : \mathbf{\Phi}_k(x_i, y_i) - \mathbf{\Phi}(x_i, z_{i,j})_k = 0\} \end{aligned}$$

- Define three corresponding values:

$$\begin{aligned} S_k^+ &= \sum_{(i,j) \in A_k^+} e^{-\mathbf{m_{i,j}}(\mathbf{W})} \\ S_k^- &= \sum_{(i,j) \in A_k^-} e^{-\mathbf{m_{i,j}}(\mathbf{W})} \\ S_k^0 &= \sum_{(i,j) \in A_k^0} e^{-\mathbf{m_{i,j}}(\mathbf{W})} \end{aligned}$$

Some Results

Everything depends on the S_k^+, S_k^- values:

- For any feature k,

$$\arg\min_{\delta} \text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta)) = \frac{1}{2}\log\frac{S_k^+}{S_k^-}$$

- And if $\delta^* = \frac{1}{2}\log\frac{S_k^+}{S_k^-}$, then

$$\text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta^*)) = \text{ExpLoss}(\mathbf{W}) - \left(\sqrt{S_k^+} - \sqrt{S_k^-}\right)^2$$

- How do we compute

$$(k^*, \delta^*) = \arg\min_{k,\delta} \text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta)) \ ?$$

Answer: choose

$$k^* = \text{argmax}_k \left| \sqrt{S_k^+} - \sqrt{S_k^-} \right|$$

and

$$\delta^* = \frac{1}{2} \log \frac{S_{k^*}^+}{S_{k^*}^-}$$

Smoothing

- There's a problem with the update:

$$\delta^* = \frac{1}{2} \log \frac{S^+_{k^*}}{S^-_{k^*}}$$

  sometimes $S^+_{k^*}$ or $S^-_{k^*}$ can be zero

- A solution (suggested by [Schapire and Singer, 1998]):

$$\delta^* = \frac{1}{2} \log \frac{S^+_{k^*} + \epsilon Z}{S^-_{k^*} + \epsilon Z}$$

  where ϵ is a smoothing parameter, and $Z = \sum_{i,j} e^{-\mathbf{m_{i,j}}(\mathbf{W})}$

- Works well in practice, though formal motivation is perhaps lacking

Input: training examples (x_i, y_i), function **GEN**, representation $\mathbf{\Phi}$, smoothing parameter ϵ

Initialization: for all k, set $\mathbf{W}_k = 0$

Algorithm: for $t = 1 \ldots T$,

- Calculate Z. For all k, calculate S_k^+, S_k^-
- Choose

$$k^* = \text{argmax} \left| \sqrt{S_k^+} - \sqrt{S_k^-} \right|$$

and

$$\delta^* = \frac{1}{2} \log \frac{S_{k^*}^+ + \epsilon Z}{S_{k^*}^- + \epsilon Z}$$

- Set $\mathbf{W} = \text{Upd}(\mathbf{W}, k^*, \delta^*)$

Output: parameters $\mathbf{W}$

A More Efficient Algorithm

- A naive method: calculating S_k^-, S_k^+, Z values at each iteration requires a pass over the entire training set

- An observation: in **sparse** feature spaces, the feature chosen k^* may only occur on a few examples $\Rightarrow$ only a few margins in training data are altered at each iteration

- An improved method: only visits a subset of the training data at each iteration. Can lead to dramatic speed-ups in sparse feature spaces.

Proof of the Updates

- First, for fixed k, let's calculate

$$\arg\min_{\delta} \text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta)) = \arg\min_{\delta} \sum_{i,j} e^{-\mathbf{m_{i,j}}(\text{Upd}(\mathbf{W},k,\delta))}$$

- Three cases:

$$\begin{aligned}
\mathbf{m_{i,j}}(\text{Upd}(\mathbf{W}, k, \delta)) &= \mathbf{m_{i,j}}(\mathbf{W}) + \delta && \text{for } (i,j) \in A_k^+ \\
\mathbf{m_{i,j}}(\text{Upd}(\mathbf{W}, k, \delta)) &= \mathbf{m_{i,j}}(\mathbf{W}) - \delta && \text{for } (i,j) \in A_k^- \\
\mathbf{m_{i,j}}(\text{Upd}(\mathbf{W}, k, \delta)) &= \mathbf{m_{i,j}}(\mathbf{W}) && \text{for } (i,j) \in A_k^0
\end{aligned}$$

- This implies:

$$\begin{aligned}
\text{ExpLoss}(\text{Upd}(\mathbf{W}, k, \delta)) &= \sum_{(i,j)\in A_k^+} e^{-\mathbf{m_{i,j}}(\mathbf{W})-\delta} + \sum_{(i,j)\in A_k^-} e^{-\mathbf{m_{i,j}}(\mathbf{W})+\delta} + \sum_{(i,j)\in A_k^0} e^{-\mathbf{m_{i,j}}(\mathbf{W})} \\
&= S_k^+ e^{-\delta} + S_k^- e^{\delta} + S_k^0
\end{aligned}$$

Say $F(\delta) = S_k^+ e^{-\delta} + S_k^- e^{\delta} + S_k^0$

If we differentiate $F(\delta)$ with respect to δ, and set the derivative to 0, we get

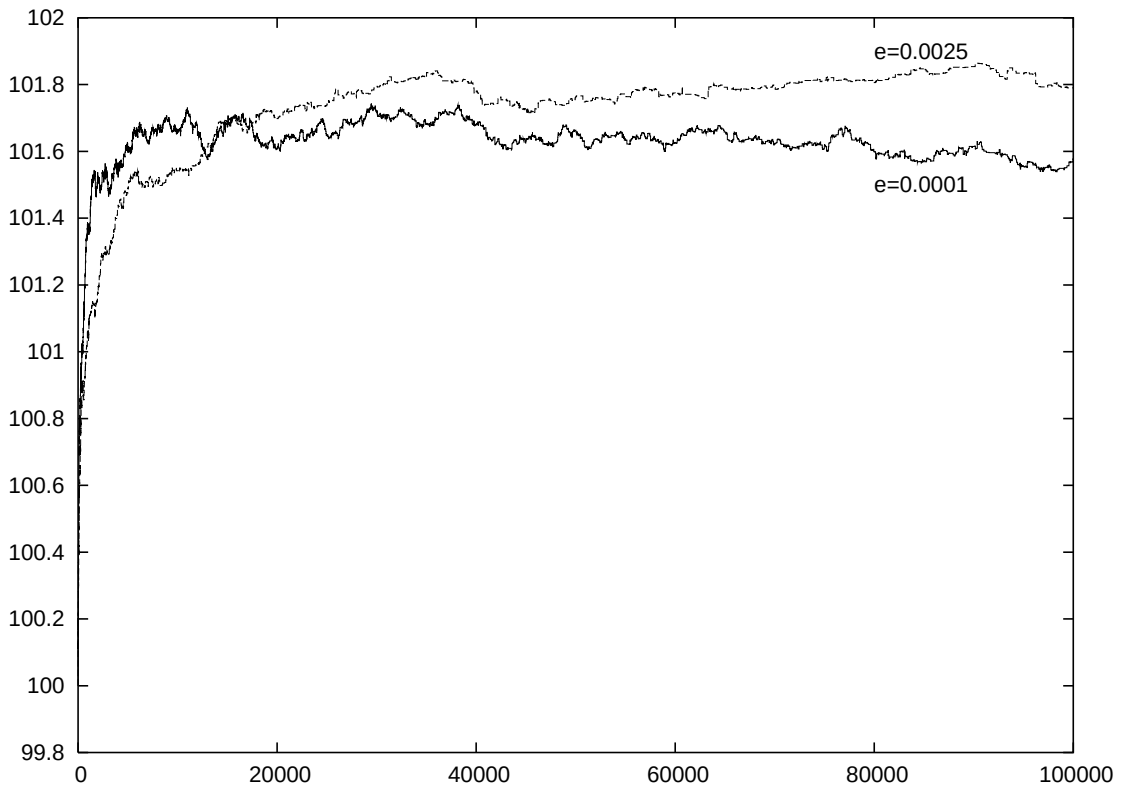
$$\delta^* = \frac{1}{2} \log \frac{S_k^+}{S_k^-}$$

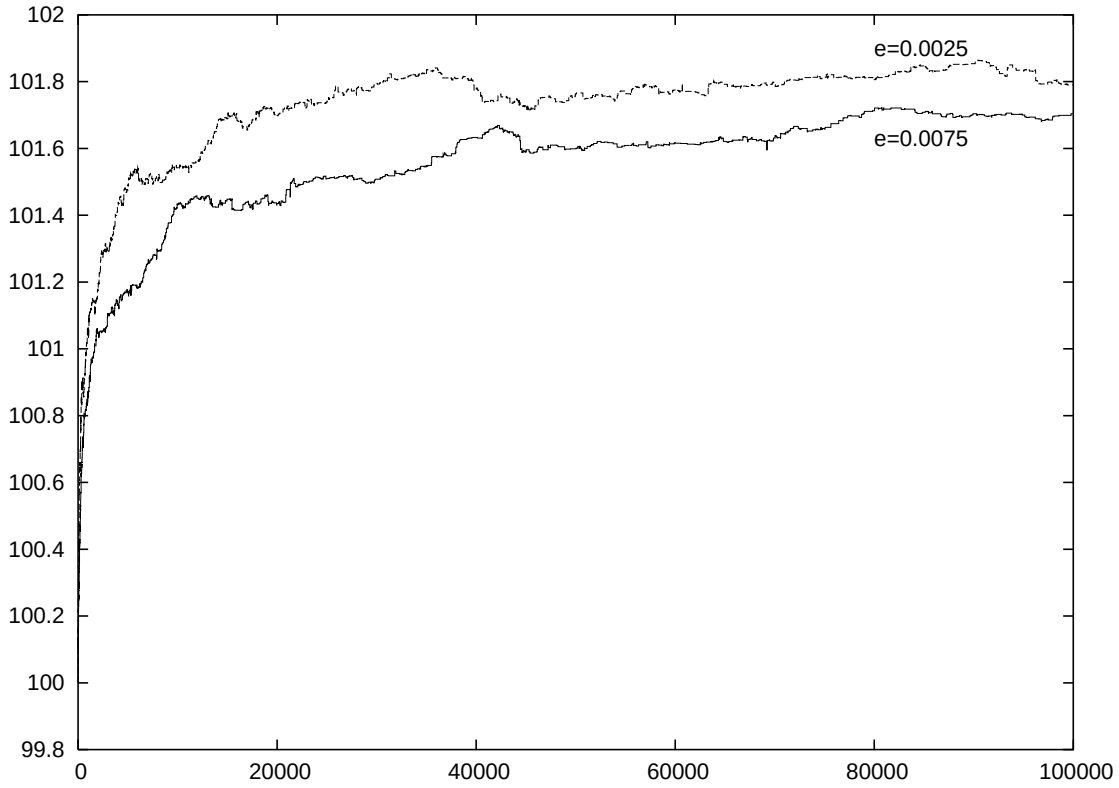
If we substitute δ^* back into $F(\delta)$, we get

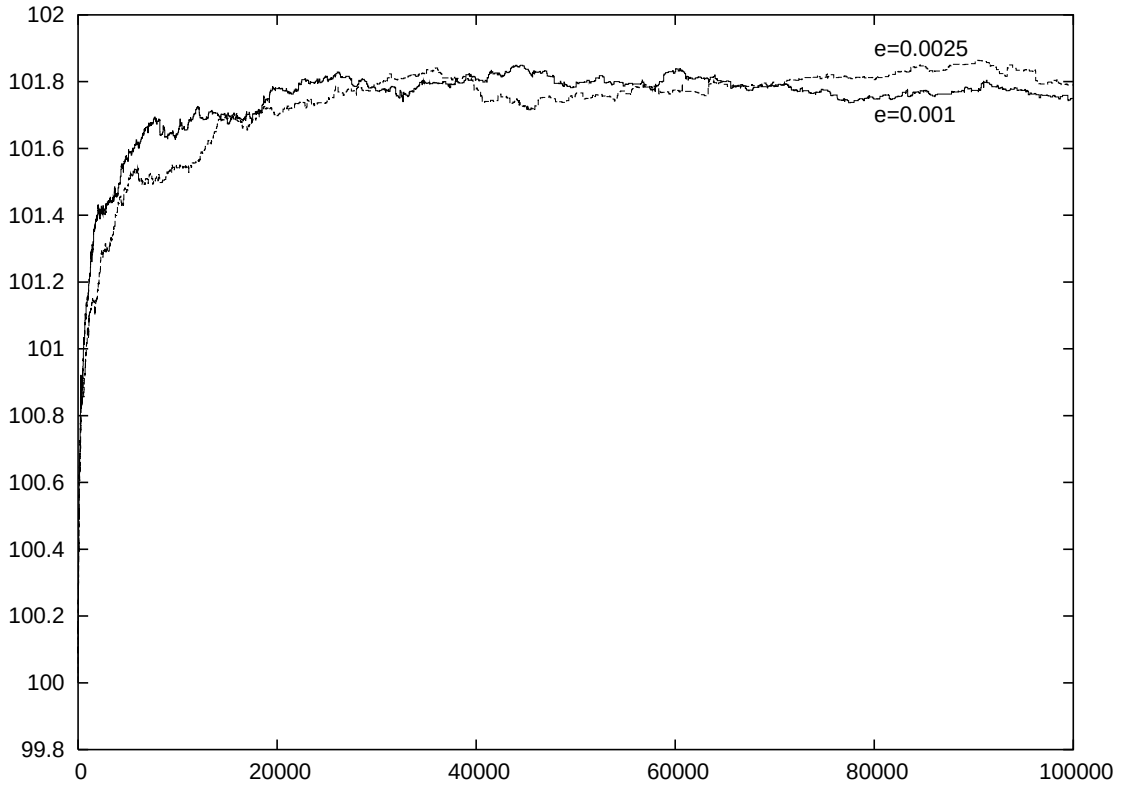
$$\begin{aligned} F(\delta^*) &= 2\sqrt{S_k^+ S_k^-} + S_k^0 \\ &= 2\sqrt{S_k^+ S_k^-} + Z - S_k^+ - S_k^- \\ &= Z - \left(\sqrt{S_k^+} - \sqrt{S_k^-}\right)^2 \end{aligned}$$

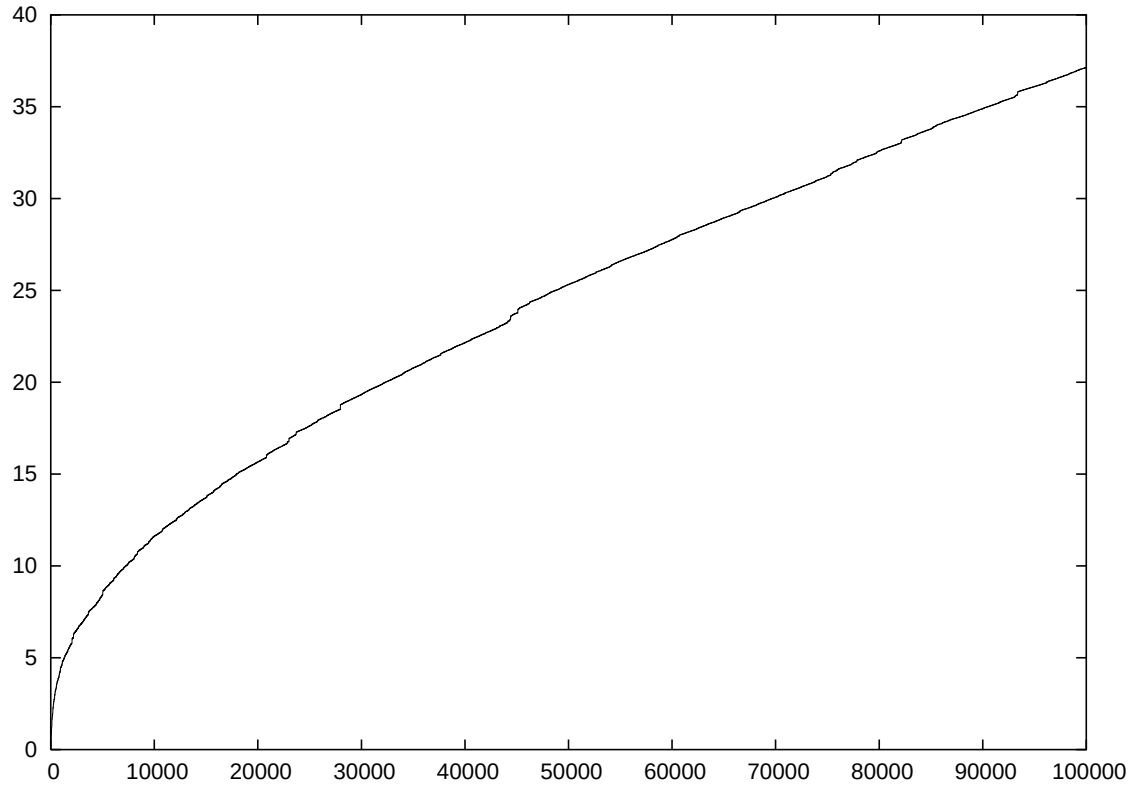
where $Z = \sum_{i,j} e^{-\mathbf{m_{i,j}(W)}} = \text{ExpLoss}(\mathbf{W})$

Effect of the ϵ (smoothing) parameter



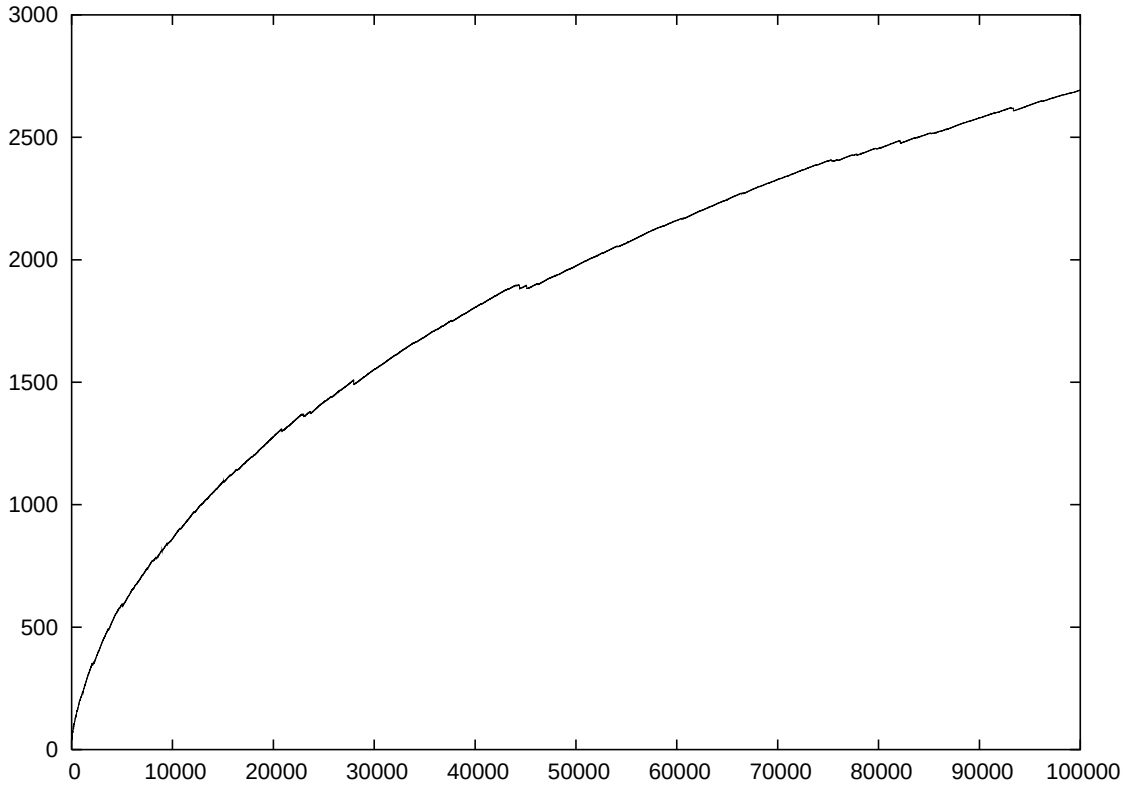

102
101.8
101.6
101.4
101.2
101
100.8
100.6
100.4
100.2
100
99.8
0
20000
40000
60000
80000
100000
e=0.0025
e=0.0075


102
101.8
101.6
101.4
101.2
101
100.8
100.6
100.4
100.2
100
99.8
0
20000
40000
60000
80000
100000
e=0.0025
e=0.001



A graph of $Work(n)$ (y-axis) vs. n (x-axis).

Here, $Work(n)$ is the computation required for n rounds of feature selection, where a single unit of computation corresponds to a pass over the entire training set.



A graph of $Savings(n)$ (y-axis) vs. n (x-axis).
$Savings(n)$ tracks the relative efficiency of the two algorithms as a function of the number of features, n. For example, if $Savings(100) = 1,200$ this signifies that for the first 100 rounds of feature selection the improved algorithm is $1,200$ times as efficient as the naive algorithm.