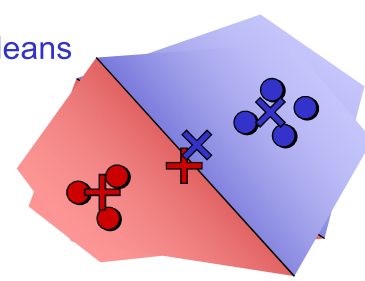
Statistical NLP
Spring 2009

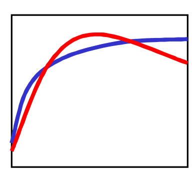
Lecture 5: WSD / Maxent

Dan Klein – UC Berkeley

Learning Models with EM

- Hard EM: alternate between
  - E-step: Find best "completions" Y for fixed θ
  - M-step: Find best parameters θ for fixed Y
- Example: K-Means





- Problem 3: Data likelihood (usually) isn't the objective you really care about
- Problem 4: You can't find global maxima anyway

Hard EM for Naïve-Bayes

- First we calculate hard completions:

$$y^* = \arg\max_y \mathsf{P}(y) \prod_i \mathsf{P}(x_i|y)$$

- Then we re-estimate parameters P(y), P(x|y) from the relevant counts:

$$c(w, y) = \sum_{x \in D: y^* = y} [c(w \in x)]$$

- Can do this when some or none of the docs are labeled

Soft EM for Naïve-Bayes

- First we calculate posteriors (soft completions):

$$\mathsf{P}(y|x) = \frac{\mathsf{P}(y) \prod_i \mathsf{P}(x_i|y)}{\sum_{y'} \mathsf{P}(y') \prod_i \mathsf{P}(x_i|y')}$$

- Then we re-estimate parameters P(y), P(x|y) from the relevant expected counts:

$$c(w, y) = \sum_{x \in D} \sum_y P(y|x) \, [c(w \in x)]$$

- Can do this when some or none of the docs are labeled

EM in General

- We'll use EM over and over again to fill in missing data
  - Convenience Scenario: we want P(x), including y just makes the model simpler (e.g. mixing weights)
  - Induction Scenario: we actually want to know y (e.g. clustering)
  - NLP differs from much of machine learning in that we often want to interpret or use the induced variables (which is tricky at best)
- General approach: alternately update y and θ
  - E-step: compute posteriors $P(y|x,\theta)$
    - This means scoring all completions with the current parameters
    - Usually, we do this implicitly with dynamic programming
  - M-step: fit θ to these completions
    - This is usually the easy part – treat the completions as (fractional) complete data
  - In general, we start with some noisy labelings and the noise adjusts into patterns based on the data and the model
  - We'll see lots of examples in this course
- EM is only locally optimal (why?)

Heuristic Clustering?

- Many methods of clustering have been developed
  - Most start with a pairwise distance function
  - Most can be interpreted probabilistically (with some effort)
  - Axes: flat / hierarchical, agglomerative / divisive, incremental / iterative, probabilistic / graph theoretic / linear algebraic
- Examples:
  - Single-link agglomerative clustering
  - Complete-link agglomerative clustering
  - Ward's method
  - Hybrid divisive / agglomerative schemes

Document Clustering

- Typically want to cluster documents by topic
- Bag-of-words models usually do detect topic
  - It's detecting deeper structure, syntax, etc. where it gets really tricky!
- All kinds of games to focus the clustering
  - Stopword lists
  - Term weighting schemes (from IR, more later)
  - Dimensionality reduction (more later)

Word Senses

- Words have multiple distinct meanings, or senses:
  - Plant: living plant, manufacturing plant, …
  - Title: name of a work, ownership document, form of address, material at the start of a film, …
- Many levels of sense distinctions
  - Homonymy: totally unrelated meanings (river bank, money bank)
  - Polysemy: related meanings (star in sky, star on tv)
  - Systematic polysemy: productive meaning extensions (metonymy such as organizations to their buildings) or metaphor
  - Sense distinctions can be extremely subtle (or not)
- Granularity of senses needed depends a lot on the task
- Why is it important to model word senses?
  - Translation, parsing, information retrieval?

Word Sense Disambiguation

- Example: living plant vs. manufacturing plant
- How do we tell these senses apart?
  - "context"

    The manufacturing plant which had previously sustained the town's economy shut down after an extended labor strike.

  - Maybe it's just text categorization
  - Each word sense represents a topic
  - Run the naive-bayes classifier from last class?
- Bag-of-words classification works ok for noun senses
  - 90% on classic, shockingly easy examples (line, interest, star)
  - 80% on senseval-1 nouns
  - 70% on senseval-1 verbs

Verb WSD

- Why are verbs harder?
  - Verbal senses less topical
  - More sensitive to structure, argument choice
- Verb Example: "Serve"
  - [function] The tree stump serves as a table
  - [enable] The scandal served to increase his popularity
  - [dish] We serve meals for the homeless
  - [enlist] She served her country
  - [jail] He served six years for embezzlement
  - [tennis] It was Agassi's turn to serve
  - [legal] He was served by the sheriff

Various Approaches to WSD

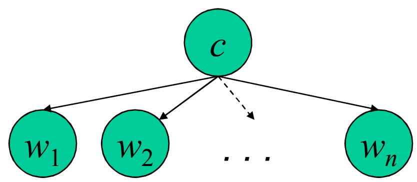
- Unsupervised learning
  - Bootstrapping (Yarowsky 95)
  - Clustering
- Indirect supervision
  - From thesauri
  - From WordNet
  - From parallel corpora
- Supervised learning
  - Most systems do some kind of supervised learning
  - Many competing classification technologies perform about the same (it's all about the knowledge sources you tap)
  - Problem: training data available for only a few words

Resources

- WordNet
  - Hand-build (but large) hierarchy of word senses
  - Basically a hierarchical thesaurus
- SensEval -> SemEval
  - A WSD competition, of which there have been 3+3 iterations
  - Training / test sets for a wide range of words, difficulties, and parts-of-speech
  - Bake-off where lots of labs tried lots of competing approaches
- SemCor
  - A big chunk of the Brown corpus annotated with WordNet senses
- OtherResources
  - The Open Mind Word Expert
  - Parallel texts
  - Flat thesauri

Knowledge Sources

- So what do we need to model to handle “serve”?
  - There are distant topical cues
    - …. point … court ………………… serve ……… game …



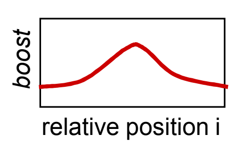
$$P(c, w_1, w_2, \ldots w_n) = P(c)\prod_i P(w_i \mid c)$$

Weighted Windows with NB

- Distance conditioning
  - Some words are important only when they are nearby
  - …. as …. point … court ………………… serve ……… game …
  - …. ……………………………………………serve as……………..

$$P(c, w_{-k}, \ldots, w_{-1}, w_0, w_{+1}, \ldots w_{+k'}) = P(c)\prod_{i=-k}^{k'} P(w_i \mid c, bin(i))$$

- Distance weighting
  - Nearby words should get a larger vote
  - … court …… serve as……… game …… point



$$P(c, w_{-k}, \ldots, w_{-1}, w_0, w_{+1}, \ldots w_{+k'}) = P(c)\prod_{i=-k}^{k'} P(w_i \mid c)^{boost(i)}$$

Better Features

- There are smarter features:
  - Argument selectional preference:
    - serve NP[meals] vs. serve NP[papers] vs. serve NP[country]
  - Subcategorization:
    - [function] serve PP[as]
    - [enable] serve VP[to]
    - [tennis] serve <intransitive>
    - [food] serve NP {PP[to]}
  - Can capture poorly (but robustly) with local windows
  - … but we can also use a parser and get these features explicitly
- Other constraints (Yarowsky 95)
  - One-sense-per-discourse (only true for broad topical distinctions)
  - One-sense-per-collocation (pretty reliable when it kicks in: manufacturing plant, flowering plant)

Complex Features with NB?

- Example: Washington County jail served 11,166 meals last month - a figure that translates to feeding some 120 people three times daily for 31 days.
- So we have a decision to make based on a set of cues:
  - context:jail, context:county, context:feeding, …
  - local-context:jail, local-context:meals
  - subcat:NP, direct-object-head:meals
- Not clear how build a generative derivation for these:
  - Choose topic, then decide on having a transitive usage, then pick “meals” to be the object’s head, then generate other words?
  - How about the words that appear in multiple features?
  - Hard to make this work (though maybe possible)
  - No real reason to try

A Discriminative Approach

- View WSD as a discrimination task (regression, really)

  P(sense | context:jail, context:county,
  context:feeding, …
  local-context:jail, local-context:meals
  subcat:NP, direct-object-head:meals, ….)

- Have to estimate multinomial (over senses) where there are a huge number of things to condition on
  - History is too complex to think about this as a smoothing / back-off problem
- Many feature-based classification techniques out there
- We tend to need ones that output distributions over classes (why?)

Feature Representations

d

Washington County jail served 11,166 meals last month - a figure that translates to feeding some 120 people three times daily for 31 days.

$\{f_i(d)\}$

context:jail = 1
context:county = 1
context:feeding = 1
context:game = 0
…
local-context:jail = 1
local-context:meals = 1
…
subcat:NP = 1
subcat:PP = 0
…
object-head:meals = 1
object-head:ball = 0

- Features are indicator functions f_i which count the occurrences of certain patterns in the input
- We map each input to a vector of feature predicate counts

Example: Text Classification

- We want to classify documents into categories

DOCUMENT	CATEGORY
… win the election …	*POLITICS*
… win the game …	*SPORTS*
… see a movie …	*OTHER*

- Classically, do this on the basis of words in the document, but other information sources are potentially relevant:
  - Document length
  - Average word length
  - Document's source
  - Document layout

Some Definitions

Sometimes, we want Y to depend on x

INPUTS	$\mathbf{x}^i$	*… win the election …*
OUTPUT SPACE	$\mathcal{Y}$	*SPORTS*, *POLITICS*, *OTHER*
OUTPUTS	$\mathbf{y}$	*SPORTS*
TRUE OUTPUTS	$\mathbf{y}^i$	*POLITICS*
FEATURE VECTORS	$\mathbf{f}_i(\mathbf{y})$	[0 0 0 0 1 0 1 0 0 0 0 0]

Either x is implicit, or y contains x

SPORTS ∧ *"win"* *POLITICS* ∧ *"election"* *POLITICS* ∧ *"win"*

Block Feature Vectors

- Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates

$\mathbf{x}$ … *win the election* …

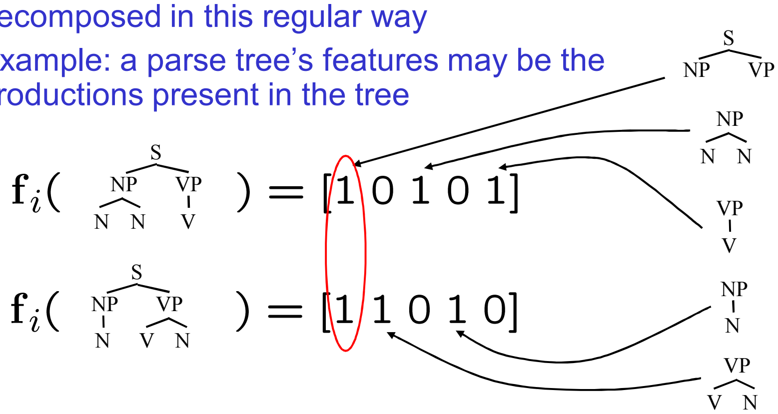
"$\mathbf{f}_i(\mathbf{x})$" [1 0 1 0]

"*win*" "*election*"

$$\mathbf{f}_i(\textit{SPORTS}) = [1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$
$$\mathbf{f}_i(\textit{POLITICS}) = [0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$$
$$\mathbf{f}_i(\textit{OTHER}) = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0]$$

Non-Block Feature Vectors

- Sometimes the features of candidates cannot be decomposed in this regular way
- Example: a parse tree's features may be the productions present in the tree



- Different candidates will thus often share features
- We'll return to the non-block case later

Linear Models: Scoring

- In a linear model, each feature gets a weight w

$$
\begin{aligned}
\mathbf{f}_i(\mathit{POLITICS}) &= [\ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0] \\
\mathbf{f}_i(\mathit{SPORTS}) &= [\ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0] \\
\mathbf{w} &= [\ 1\ \ 1\ \ {-1}\ \ {-2}\ \ 1\ \ {-1}\ \ 1\ \ {-2}\ \ {-2}\ \ {-1}\ \ {-1}\ \ 1]
\end{aligned}
$$

- We compare hypotheses on the basis of their linear scores:

$$score(\mathbf{x}^i, \mathbf{y}, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$$

$$
\begin{aligned}
\mathbf{f}_i(\mathit{POLITICS}) &= [\ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0] \\
\mathbf{w} &= [\ 1\ \ 1\ \ {-1}\ \ {-2}\ \ 1\ \ {-1}\ \ 1\ \ {-2}\ \ {-2}\ \ {-1}\ \ {-1}\ \ 1]
\end{aligned}
$$

$$score(\mathbf{x}^i, \mathit{POLITICS}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2$$

Linear Models: Prediction Rule

- The linear prediction rule:

$$prediction(\mathbf{x}^i, \mathbf{w}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$$

$$
\begin{aligned}
score(\mathbf{x}^i, \mathit{SPORTS}, \mathbf{w}) &= 1 \times 1 + (-1) \times 1 = 0 \\
score(\mathbf{x}^i, \mathit{POLITICS}, \mathbf{w}) &= 1 \times 1 + 1 \times 1 = 2 \\
score(\mathbf{x}^i, \mathit{OTHER}, \mathbf{w}) &= (-2) \times 1 + (-1) \times 1 = -3
\end{aligned}
$$

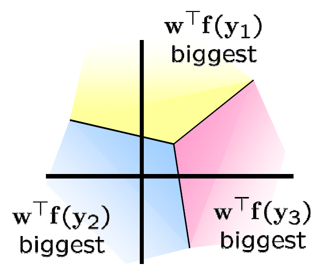
$$\Downarrow$$

$$prediction(\mathbf{x}^i, \mathbf{w}) = \mathit{POLITICS}$$

- We've said nothing about where weights come from!

Multiclass Decision Rule

- If more than two classes:
  - Highest score wins
  - Boundaries are more complex
  - Harder to visualize



$$prediction(\mathbf{x}^i, \mathbf{w}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$$

- There are other ways: e.g. reconcile pairwise decisions

Learning Classifier Weights

- Two broad approaches to learning weights
- Generative: work with a probabilistic model of the data, weights are (log) local conditional probabilities
  - Advantages: learning weights is easy, smoothing is well-understood, backed by understanding of modeling
- Discriminative: set weights based on some error-related criterion
  - Advantages: error-driven, often weights which are good for classification aren't the ones which best describe the data
- We'll mainly talk about the latter

How to pick weights?

- Goal: choose “best” vector w given training data
  - For now, we mean “best for classification”
- The ideal: the weights which have greatest test set accuracy / F1 / whatever
  - But, don’t have the test set
  - Must compute weights from training set
- Maybe we want weights which give best training set accuracy?
  - Hard discontinuous optimization problem
  - May not (does not) generalize to test set
  - Easy to overfit

Though, min-error training for MT does exactly this.

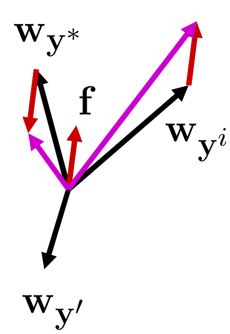
Linear Models: Perceptron

- The perceptron algorithm
  - Iteratively processes the training set, reacting to training errors
  - Can be thought of as trying to drive down training error
- The (online) perceptron algorithm:
  - Start with zero weights
  - Visit training instances one by one
    - Try to classify

$$\mathbf{y}^* = \arg\max_{\mathbf{y}\in\mathcal{Y}} \mathbf{w}^\top \mathbf{f}_i(\mathbf{y})$$

    - If correct, no change!
    - If wrong: adjust weights

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}_i(\mathbf{y}^i)$$
$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{f}_i(\mathbf{y}^*)$$



Linear Models: Maximum Entropy

- Maximum entropy (logistic regression)
  - Use the scores as probabilities:

$$\mathrm{P}(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{\exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}))}{\sum_{\mathbf{y}'} \exp(\mathbf{w}^\top \mathbf{f}(\mathbf{y}'))}$$

Make positive

Normalize

  - Maximize the (log) conditional likelihood of training data

$$L(\mathbf{w}) = \log \prod_i \mathrm{P}(\mathbf{y}^i|\mathbf{x}^i,\mathbf{w}) = \sum_i \log \left(\frac{\exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i))}{\sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}))} \right)$$

$$= \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

Derivative for Maximum Entropy

$$L(\mathbf{w}) = \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}_n} = \sum_i \left(\mathbf{f}_i(\mathbf{y}^i)_n - \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}_i) \mathbf{f}_i(\mathbf{y})_n \right)$$

Total count of feature n in correct candidates

Expected count of feature n in predicted candidates

Expected Counts

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}_n} = \sum_i \left(\mathbf{f}_i(\mathbf{y}^i)_n - \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}_i)\mathbf{f}_i(\mathbf{y})_n \right)$$

		y^i	$P(y \mid x^i, w)$
x^i's	meal, jail, …	food	.4
	jail, term, …	prison	.8
The weight for the "context-word:jail and cat:prison" feature:		actual = 1	empirical = 1.2

- The optimum parameters are the ones for which each feature's predicted expectation equals its empirical expectation. The optimum distribution is:
  - Always unique (but parameters may not be unique)
  - Always exists (if features counts are from actual data).

Maximum Entropy II

- Motivation for maximum entropy:
  - Connection to maximum entropy principle (sort of)
  - Might want to do a good job of being uncertain on noisy cases…
  - … in practice, though, posteriors are pretty peaked
- Regularization (smoothing)

$$\max_{\mathbf{w}} \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right) - k||\mathbf{w}||^2$$

Example: NER Smoothing

Because of smoothing, the more common prefixes have larger weights even though entire-word features are more specific.

Local Context

	Prev	Cur	Next
State	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

Feature Weights

Feature Type	Feature	PERS	LOC
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*<G*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Previous state	Other	-0.70	-0.92
Current signature	Xx	0.80	0.46
Prev state, cur sig	O-Xx	0.68	0.37
Prev-cur-next sig	x-Xx-Xx	-0.69	0.37
P. state - p-cur sig	O-x-Xx	-0.20	0.82
…			
Total:		**-0.58**	**2.68**

Derivative for Maximum Entropy

$$L(\mathbf{w}) = -k||\mathbf{w}||^2 + \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}_n} = -2k\mathbf{w}_n + \sum_i \left(\mathbf{f}_i(\mathbf{y}^i)_n - \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}_i)\mathbf{f}_i(\mathbf{y})_n \right)$$

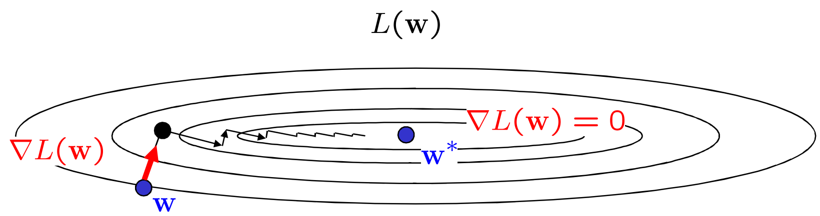
Big weights are bad

Expected count of feature n in predicted candidates

Total count of feature n in correct candidates

Unconstrained Optimization

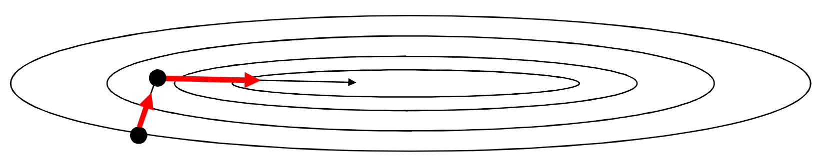
- The maxent objective is an unconstrained optimization problem



- Basic idea: move uphill from current guess
- Gradient ascent / descent follows the gradient incrementally
- At local optimum, derivative vector is zero
- Will converge if step sizes are small enough, but not efficient
- All we need is to be able to evaluate the function and its derivative

Unconstrained Optimization

- Once we have a function f, we can find a local optimum by iteratively following the gradient



- For convex functions, a local optimum will be global
- Basic gradient ascent isn't very efficient, but there are simple enhancements which take into account previous gradients: conjugate gradient, L-BFGs
- There are special-purpose optimization techniques for maxent, like iterative scaling, but they aren't better