CSEP 517
Natural Language Processing
Autumn 2013

Text Classification

Luke Zettlemoyer - University of Washington

[Many slides from Dan Klein and Michael Collins]

Overview: Classification

- Classification Problems
  - Spam vs. Non-spam, Text Genre, Word Sense, etc.
- Supervised Learning
  - Naïve Bayes
  - Log-linear models (Maximum Entropy Models)
  - Weighted linear models and the Perceptron
- Unsupervised Learning
  - The EM Algorithm for Naïve Bayes
  - Simple Semi-supervised approach

Where are we?

- So far: language models give P(s)
  - Help model fluency for various noisy-channel processes (MT, ASR, etc.)
  - N-gram models don't represent any deep variables involved in language structure or meaning
  - Usually we want to know something about the input other than how likely it is (syntax, semantics, topic, etc)
- Next: Naïve Bayes models
  - We introduce a single new probabilistic variable
  - Still a very simplistic model family
  - Lets us model properties of text, but only very non-local ones…
  - In particular, we can only model properties which are largely invariant to word order (like topic)

Text Categorization

- Want to classify documents into broad semantic topics

Obama is hoping to rally support for his $825 billion stimulus package on the eve of a crucial House vote. Republicans have expressed reservations about the proposal, calling for more tax cuts and less spending. GOP representatives seemed doubtful that any deals would be made.

California will open the 2009 season at home against Maryland Sept. 5 and will play a total of six games in Memorial Stadium in the final football schedule announced by the Pacific-10 Conference Friday. The original schedule called for 12 games over 12 weekends.

- Which one is the politics document? (And how much deep processing did that decision take?)
- First approach: bag-of-words and Naïve-Bayes models
- More approaches later…
- Usually begin with a labeled corpus containing examples of each class

Example: Spam Filter

- Input: email
- Output: spam/ham
- Setup:
  - Get a large collection of example emails, each labeled “spam” or “ham”
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future emails
- Features: The attributes used to make the ham / spam decision
  - Words: FREE!
  - Text Patterns: $dd, CAPS
  - Non-text: SenderInContacts
  - …

✗

> Dear Sir.
>
> First, I must solicit your confidence in this transaction, this is by virture of its nature as being utterly confidencial and top secret. …

✗

> TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.
>
> 99 MILLION EMAIL ADDRESSES
> FOR ONLY $99

✓

> Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

Word Sense Disambiguation

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

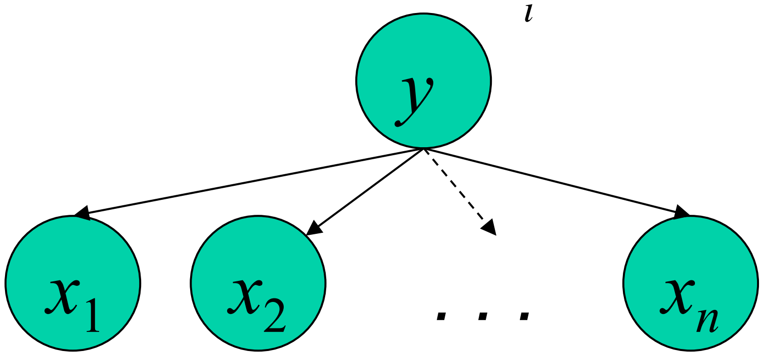
    The manufacturing plant which had previously sustained the town's economy shut down after an extended labor strike.

  - It's just text categorization! (at the word level)
  - Each word sense represents a topic

Naïve-Bayes Models

- Generative model: pick a topic, then generate a document using a language model for that topic
- Naïve-Bayes assumption: all words are independent given the topic.

$$p(y, x_1, x_2 \ldots x_n) = q(y) \prod_i q(x_i \mid y)$$



- Compare to a unigram language model:

$$p(x_1, x_2, \ldots x_n) = \prod_i q(x_i)$$

Using NB for Classification

- We have a joint model of topics and documents

$$p(y, x_1, x_2 \ldots x_n) = q(y) \prod_i \boxed{q(x_i \mid y)}$$

We have to smooth these!

- To assign a label y* to a new document $<x_{1,} x_1 \ldots x_n>$:

$$y^* = \arg\max_y p(y, x_1, x_2 \ldots x_n) = \arg\max_y q(y) \prod_i q(x_i \mid y)$$

- How do we do learning?
- Smoothing? What about totally unknown words?
- Can work shockingly well for textcat (especially in the wild)
- How can unigram models be so terrible for language modeling, but class-conditional unigram models work for textcat?
- Numerical / speed issues?

Language Identification

- How can we tell what language a document is in?

The 38th Parliament will meet on Monday, October 4, 2004, at 11:00 a.m. The first item of business will be the election of the Speaker of the House of Commons. Her Excellency the Governor General will open the First Session of the 38th Parliament on October 5, 2004, with a Speech from the Throne.

La 38e législature se réunira à 11 heures le lundi 4 octobre 2004, et la première affaire à l'ordre du jour sera l’élection du président de la Chambre des communes. Son Excellence la Gouverneure générale ouvrira la première session de la 38e législature avec un discours du Trône le mardi 5 octobre 2004.

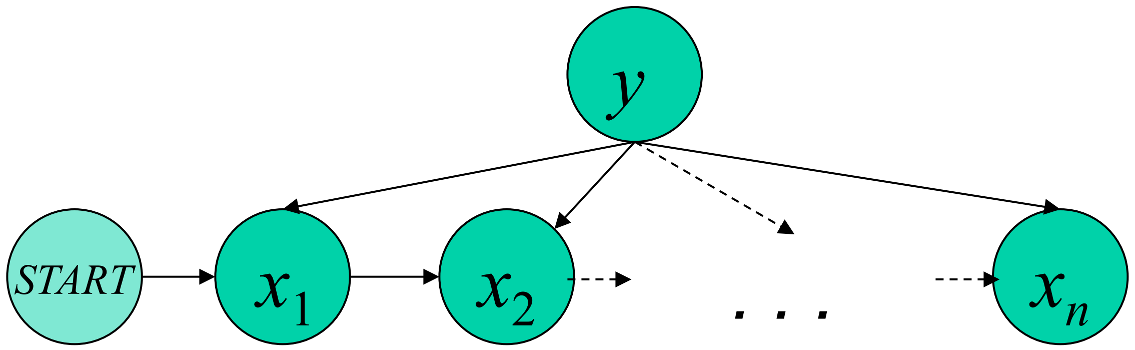
- How to tell the French from the English?
  - Treat it as word-level textcat?
    - Overkill, and requires a lot of training data
    - You don’t actually need to know about words!
  - Option: build a character-level language model

Σύμφωνο σταθερότητας και ανάπτυξης
Patto di stabilità e di crescita

Class-Conditional LMs

- Can add a topic variable to richer language models

$$p(y, x_1, x_2 \ldots x_n) = q(y) \prod_i q(x_i \mid y, x_{i-1})$$



- Could be characters instead of words, used for language ID
- Could sum out the topic variable and use as a language model
- How might a class-conditional n-gram language model behave differently from a standard n-gram model?
- Many other options are also possible!

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 a naive-bayes classifier?
- 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

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

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

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 be captured poorly (but robustly) with modified Naïve Bayes approach
- 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, directly estimate:

  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
  - Log-linear models extremely popular in the NLP community!

Learning Probabilistic Classifiers

- Two broad approaches to predicting classes y*

- Joint: work with a *joint* probabilistic model of the data, weights are (often) local conditional probabilities
  - E.g., represent p(y,x) as Naïve Bayes model, compute $y^*=\text{argmax}_y\ p(y,x)$
  - Advantages: learning weights is easy, smoothing is well-understood, backed by understanding of modeling

- Conditional: work with *conditional* probability p(y|x)
  - We can then direct compute $y^* = \text{argmax}_y\ p(y|x)$
  - Advantages: Don't have to model p(x)! Can develop *feature rich* models for p(y|x).

Feature Representations

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

$\Rightarrow$

$$\left\{\begin{array}{l}
\text{context:jail} = 1 \\
\text{context:county} = 1 \\
\text{context:feeding} = 1 \\
\text{context:game} = 0 \\
\ldots \\
\text{local-context:jail} = 1 \\
\text{local-context:meals} = 1 \\
\ldots \\
\text{subcat:NP} = 1 \\
\text{subcat:PP} = 0 \\
\ldots \\
\text{object-head:meals} = 1 \\
\text{object-head:ball} = 0
\end{array}\right\}$$

- Features are indicator functions which count the occurrences of certain patterns in the input
- We will have different feature values for every pair of input x and class y

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

Linear Models: Scoring

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

$$\phi(x, SPORTS) = [1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$\phi(x, POLITICS) = [0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$$

$$w = [\ 1\ \ 1\ \ -1\ -2\ \ 1\ \ -1\ \ 1\ \ -2\ \ -2\ \ -1\ \ -1\ \ 1]$$

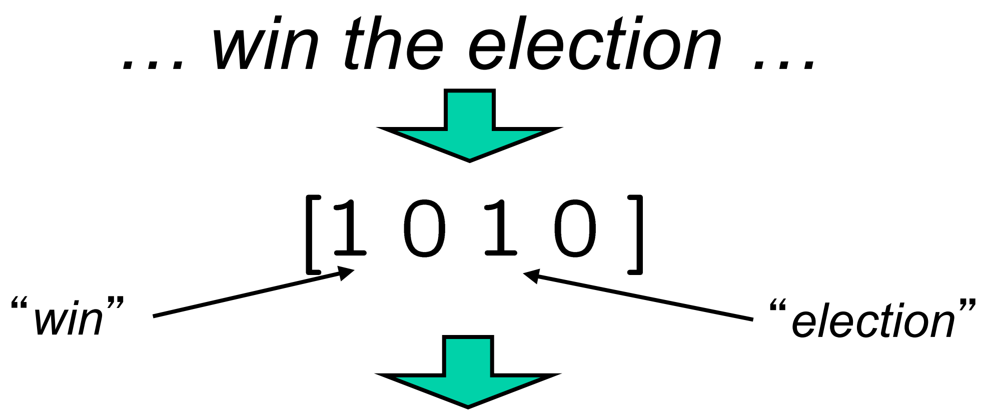
- We compare ys on the basis of their linear scores:

$$score(x, y; w) = w \cdot \phi(x, y)$$

$$score(x, POLITICS; w) = 1 \times 1 + 1 \times 1 = 2$$

Block Feature Vectors

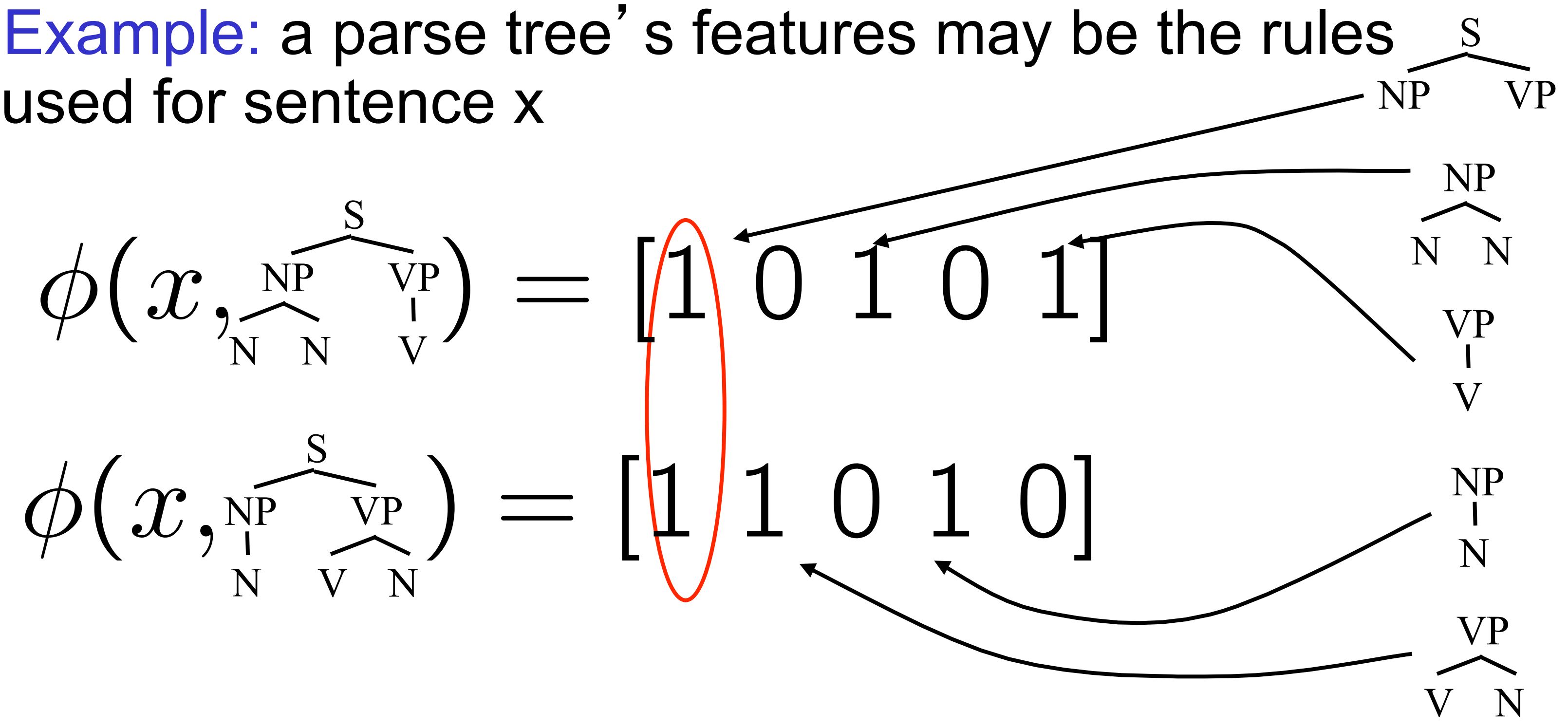
- Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates



$$\phi(x, SPORTS) = [1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$
$$\phi(x, POLITICS) = [0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$$
$$\phi(x, OTHER) = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0]$$

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 rules used for sentence x



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

Log-linear Models (Maximum Entropy)

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

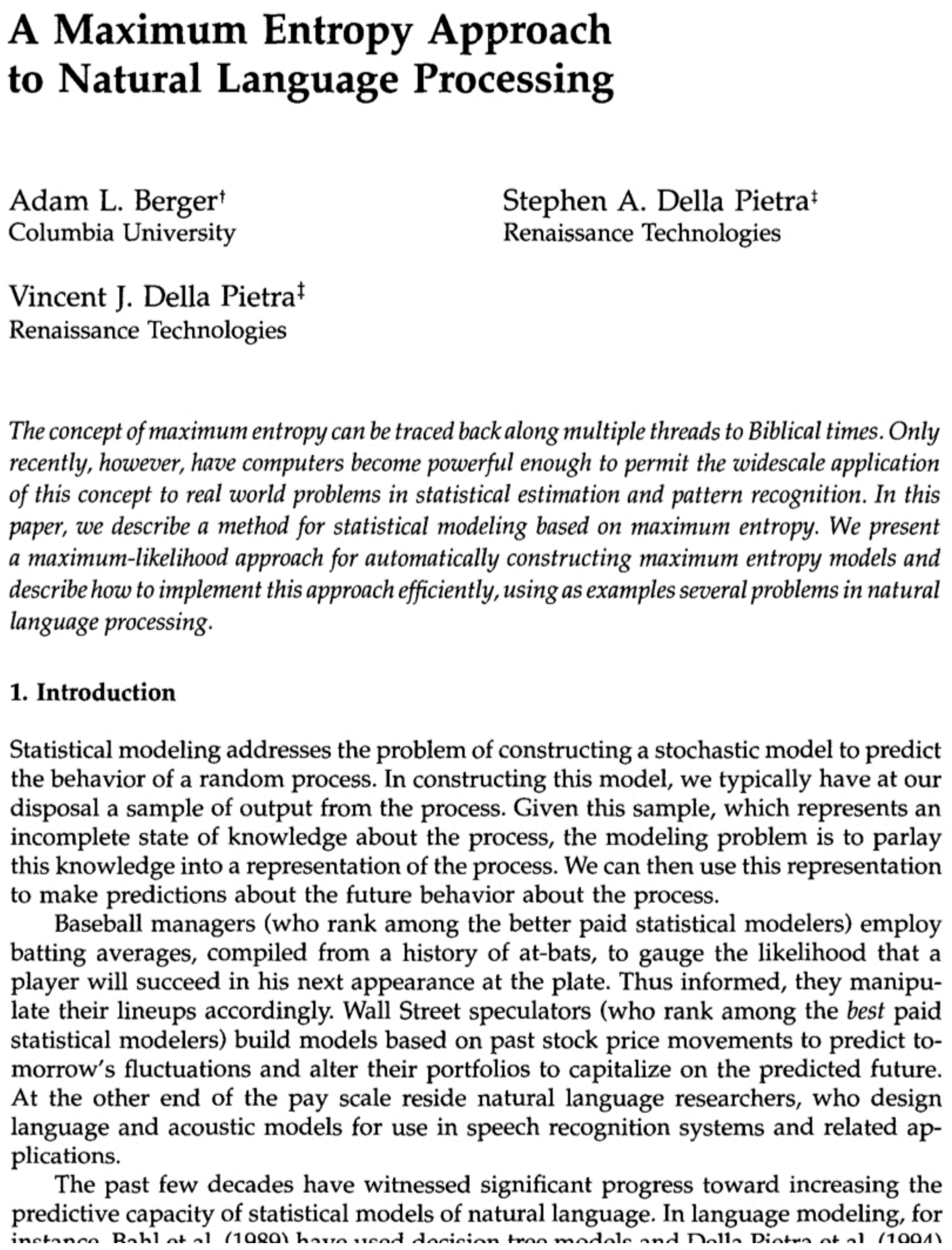
$$p(y|x;w) = \frac{\exp\left(w \cdot \phi(x,y)\right)}{\sum_{y'} \exp\left(w \cdot \phi(x,y')\right)} \quad \begin{array}{l} \longleftarrow \text{Make positive} \\[1em] \longleftarrow \text{Normalize} \end{array}$$

  - Learning: maximize the (log) conditional likelihood of training data $\{(x_i, y_i)\}_{i=1}^{n}$

$$L(w) = \sum_{i=1}^{n} \log p(y_i|x_i;w) \qquad\qquad w^* = \arg\max_{w} L(w)$$

  - Prediction: output argmax$_y$ p(y|x;w)

Adam L. Bergert;
Vincent J. Della Pietra;
Stephen A. Della Pietra.
Computational Linguistics,
22(1), 1996



A Maximum Entropy Approach to Natural Language Processing

Adam L. Berger[†]
Columbia University

Stephen A. Della Pietra[‡]
Renaissance Technologies

Vincent J. Della Pietra[‡]
Renaissance Technologies

The concept of maximum entropy can be traced back along multiple threads to Biblical times. Only recently, however, have computers become powerful enough to permit the widescale application of this concept to real world problems in statistical estimation and pattern recognition. In this paper, we describe a method for statistical modeling based on maximum entropy. We present a maximum-likelihood approach for automatically constructing maximum entropy models and describe how to implement this approach efficiently, using as examples several problems in natural language processing.

1. Introduction

Statistical modeling addresses the problem of constructing a stochastic model to predict the behavior of a random process. In constructing this model, we typically have at our disposal a sample of output from the process. Given this sample, which represents an incomplete state of knowledge about the process, the modeling problem is to parlay this knowledge into a representation of the process. We can then use this representation to make predictions about the future behavior about the process.

Baseball managers (who rank among the better paid statistical modelers) employ batting averages, compiled from a history of at-bats, to gauge the likelihood that a player will succeed in his next appearance at the plate. Thus informed, they manipulate their lineups accordingly. Wall Street speculators (who rank among the *best* paid statistical modelers) build models based on past stock price movements to predict tomorrow's fluctuations and alter their portfolios to capitalize on the predicted future. At the other end of the pay scale reside natural language researchers, who design language and acoustic models for use in speech recognition systems and related applications.

The past few decades have witnessed significant progress toward increasing the predictive capacity of statistical models of natural language. In language modeling, for instance, Bahl et al. (1989) have used decision tree models and Della Pietra et al. (1994)

Derivative of Log-linear Models

- Unfortunately, argmax_w L(w) doesn't have a close formed solution
- We will have to differentiate and use gradient ascent

$$L(w) = \sum_{i=1}^{n} \log p(y_i | x_i; w)$$

$$L(w) = \sum_{i=1}^{n} \left(w \cdot \phi(x_i, y_i) - \log \sum_{y} \exp(w \cdot \phi(x_i, y)) \right)$$

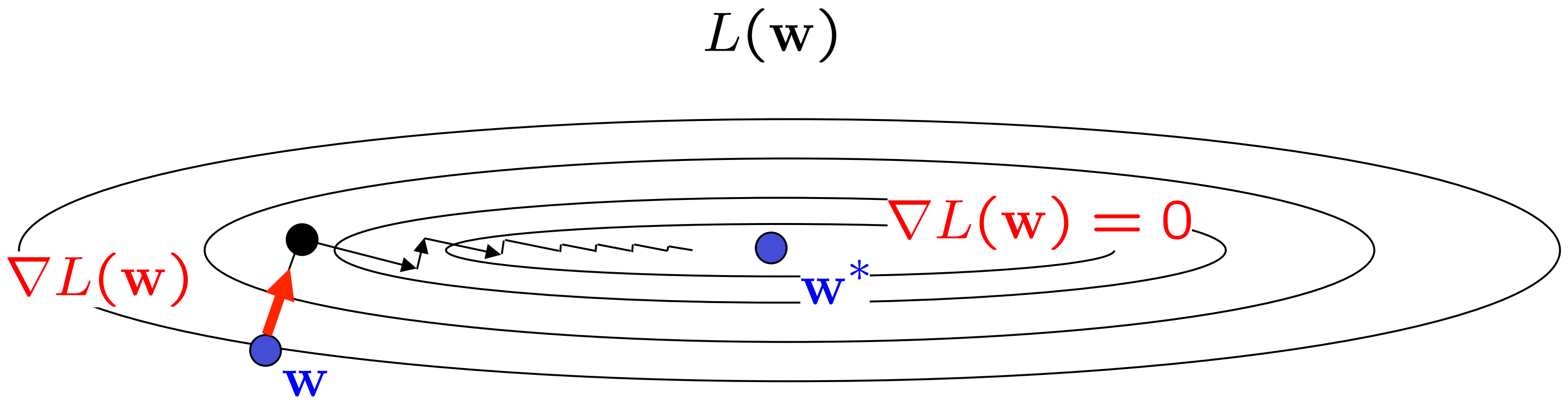
$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^{n} \left(\phi_j(x_i, y_i) - \sum_{y} p(y | x_i; w) \phi_j(x_i, y) \right)$$

Total count of feature j in correct candidates

Expected count of feature j in predicted candidates

Unconstrained Optimization

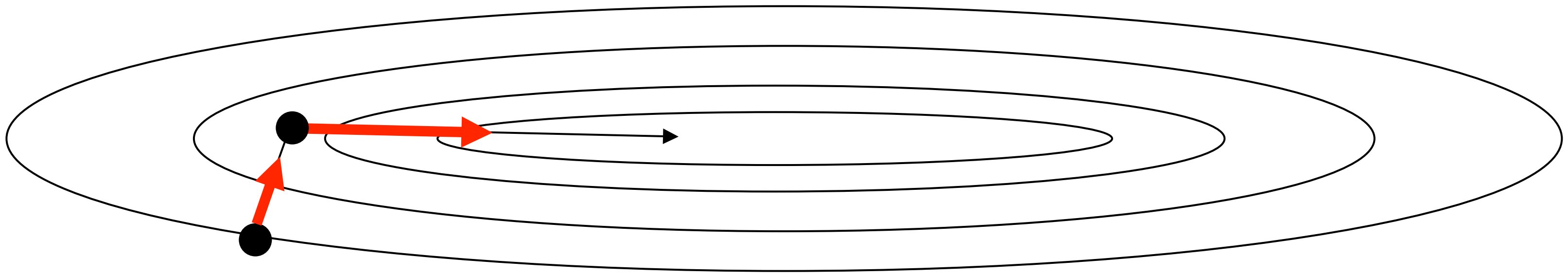
- The maxent objective is an unconstrained optimization problem



- 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

What About Overfitting?

- For Language Models and Naïve Bayes, we were worried about zero counts in MLE estimates
  - Can that happen here?

- Regularization (smoothing) for Log-linear models
  - Instead, we worry about large feature weights
  - Add a regularization term to the likelihood to push weights towards zero

$$L(w) = \sum_{i=1}^{n} \log p(y_i | x_i; w) - \frac{\lambda}{2} ||w||^2$$

Derivative for Regularized Maximum Entropy

- Unfortunately, argmax_w L(w) still doesn't have a close formed solution
- We will have to differentiate and use gradient ascent

$$L(w) = \sum_{i=1}^{n} \left(w \cdot \phi(x_i, y_i) - \log \sum_{y} \exp(w \cdot \phi(x_i, y)) \right) - \frac{\lambda}{2} ||w||^2$$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^{n} \left(\phi_j(x_i, y_i) - \sum_{y} p(y|x_i; w) \phi_j(x_i, y) \right) - \lambda w_j$$

Total count of feature j in correct candidates

Expected count of feature j in predicted candidates

Big weights are bad

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
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	*Gr*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Current signature	Xx	0.80	0.46
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**

Word Sense Disambiguation Results

[Suarez and Palomar, 2002]

- With clever features, small variations on simple log-linear (Maximum Entropy – ME) models did very well in an word sense competition:

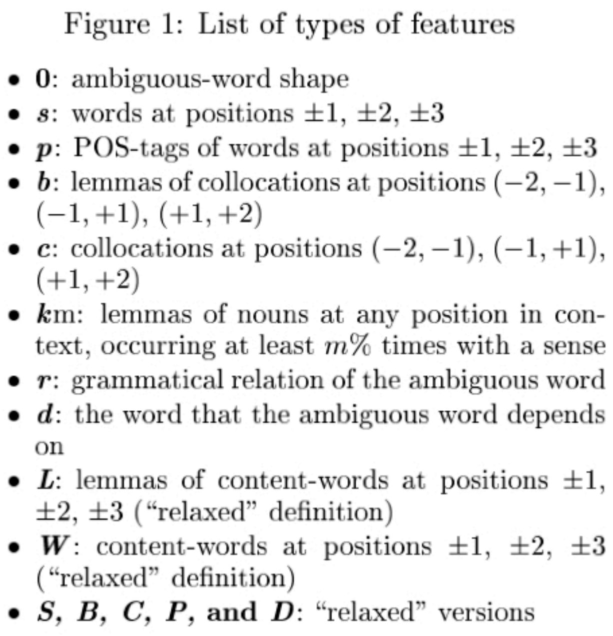


Figure 1: List of types of features

- **0**: ambiguous-word shape
- ***s***: words at positions ±1, ±2, ±3
- ***p***: POS-tags of words at positions ±1, ±2, ±3
- ***b***: lemmas of collocations at positions (−2, −1), (−1, +1), (+1, +2)
- ***c***: collocations at positions (−2, −1), (−1, +1), (+1, +2)
- ***k***m: lemmas of nouns at any position in context, occurring at least $m\%$ times with a sense
- ***r***: grammatical relation of the ambiguous word
- ***d***: the word that the ambiguous word depends on
- ***L***: lemmas of content-words at positions ±1, ±2, ±3 ("relaxed" definition)
- ***W***: content-words at positions ±1, ±2, ±3 ("relaxed" definition)
- ***S, B, C, P,* and *D***: "relaxed" versions

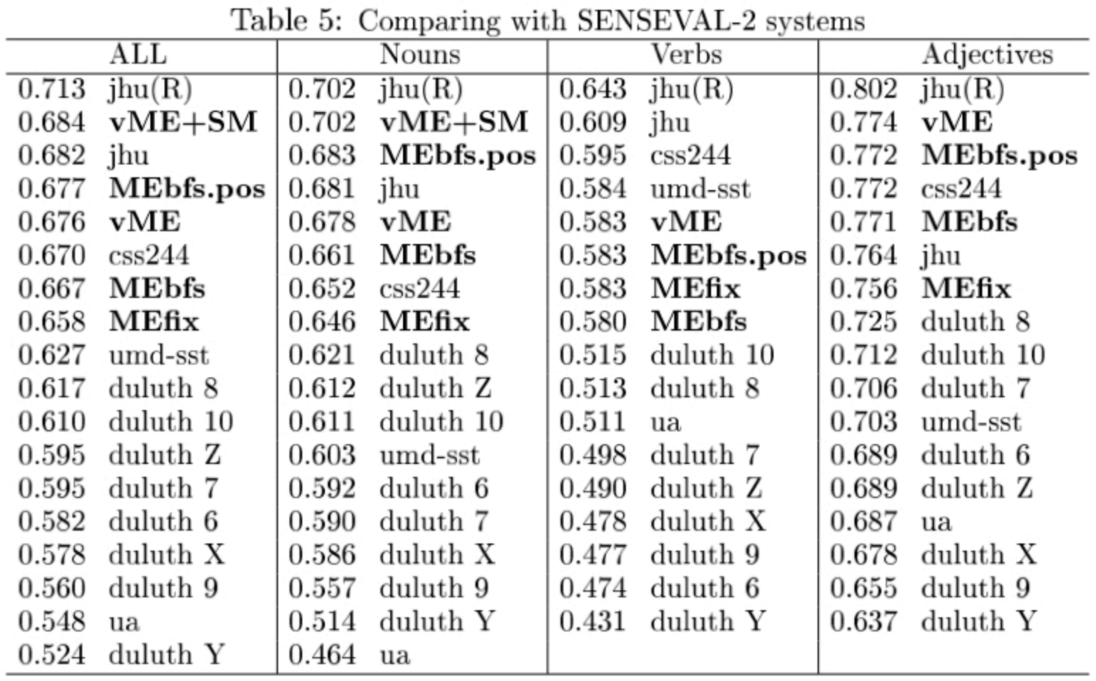


Table 5: Comparing with SENSEVAL-2 systems

ALL		Nouns		Verbs		Adjectives	
0.713	jhu(R)	0.702	jhu(R)	0.643	jhu(R)	0.802	jhu(R)
0.684	**vME+SM**	0.702	**vME+SM**	0.609	jhu	0.774	**vME**
0.682	jhu	0.683	**MEbfs.pos**	0.595	css244	0.772	**MEbfs.pos**
0.677	**MEbfs.pos**	0.681	jhu	0.584	umd-sst	0.772	css244
0.676	**vME**	0.678	**vME**	0.583	**vME**	0.771	**MEbfs**
0.670	css244	0.661	**MEbfs**	0.583	**MEbfs.pos**	0.764	jhu
0.667	**MEbfs**	0.652	css244	0.583	**MEfix**	0.756	**MEfix**
0.658	**MEfix**	0.646	**MEfix**	0.580	**MEbfs**	0.725	duluth 8
0.627	umd-sst	0.621	duluth 8	0.515	duluth 10	0.712	duluth 10
0.617	duluth 8	0.612	duluth Z	0.513	duluth 8	0.706	duluth 7
0.610	duluth 10	0.611	duluth 10	0.511	ua	0.703	umd-sst
0.595	duluth Z	0.603	umd-sst	0.498	duluth 7	0.689	duluth 6
0.595	duluth 7	0.592	duluth 6	0.490	duluth Z	0.689	duluth Z
0.582	duluth 6	0.590	duluth 7	0.478	duluth X	0.687	ua
0.578	duluth X	0.586	duluth X	0.477	duluth 9	0.678	duluth X
0.560	duluth 9	0.557	duluth 9	0.474	duluth 6	0.655	duluth 9
0.548	ua	0.514	duluth Y	0.431	duluth Y	0.637	duluth Y
0.524	duluth Y	0.464	ua				

- The winning system is a famous semi-supervised learning approach by Yarowsky
- The other systems include many different approaches: Naïve Bayes, SVMS, etc

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

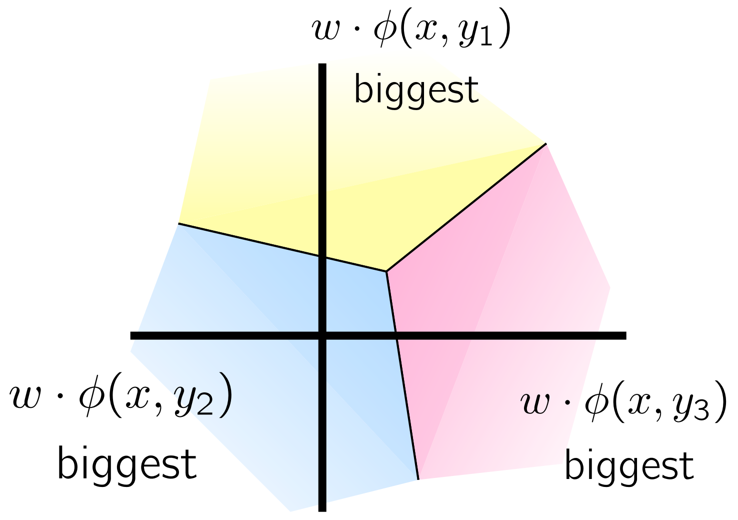
Learning Classifiers

- Two probabilistic approaches to predicting classes y*
  - Joint: work with a *joint* probabilistic model of the data, weights are (often) local conditional probabilities
    - E.g., represent p(y,x) as Naïve Bayes model, compute $y^*=\text{argmax}_y\ p(y,x)$
  - Conditional: work with *conditional* probability p(y|x)
    - We can then direct compute $y^* = \text{argmax}_y\ p(y|x)$ Can develop *feature rich* models for p(y|x).

- But, why estimate a distribution at all?
  - Linear predictor: $y^* = \text{argmax}_y\ w \bullet \phi(x,y)$
  - Perceptron algorithm
    - Online
    - Error driven
    - Simple, additive updates

Multiclass Perceptron Decision Rule

- Compare all possible outputs
  - Highest score wins
  - Boundaries are more complex
  - Harder to visualize



$$y^* = \arg\max_y w \cdot \phi(x, y)$$

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 (x_i,y_i) one by one
    - Make a prediction

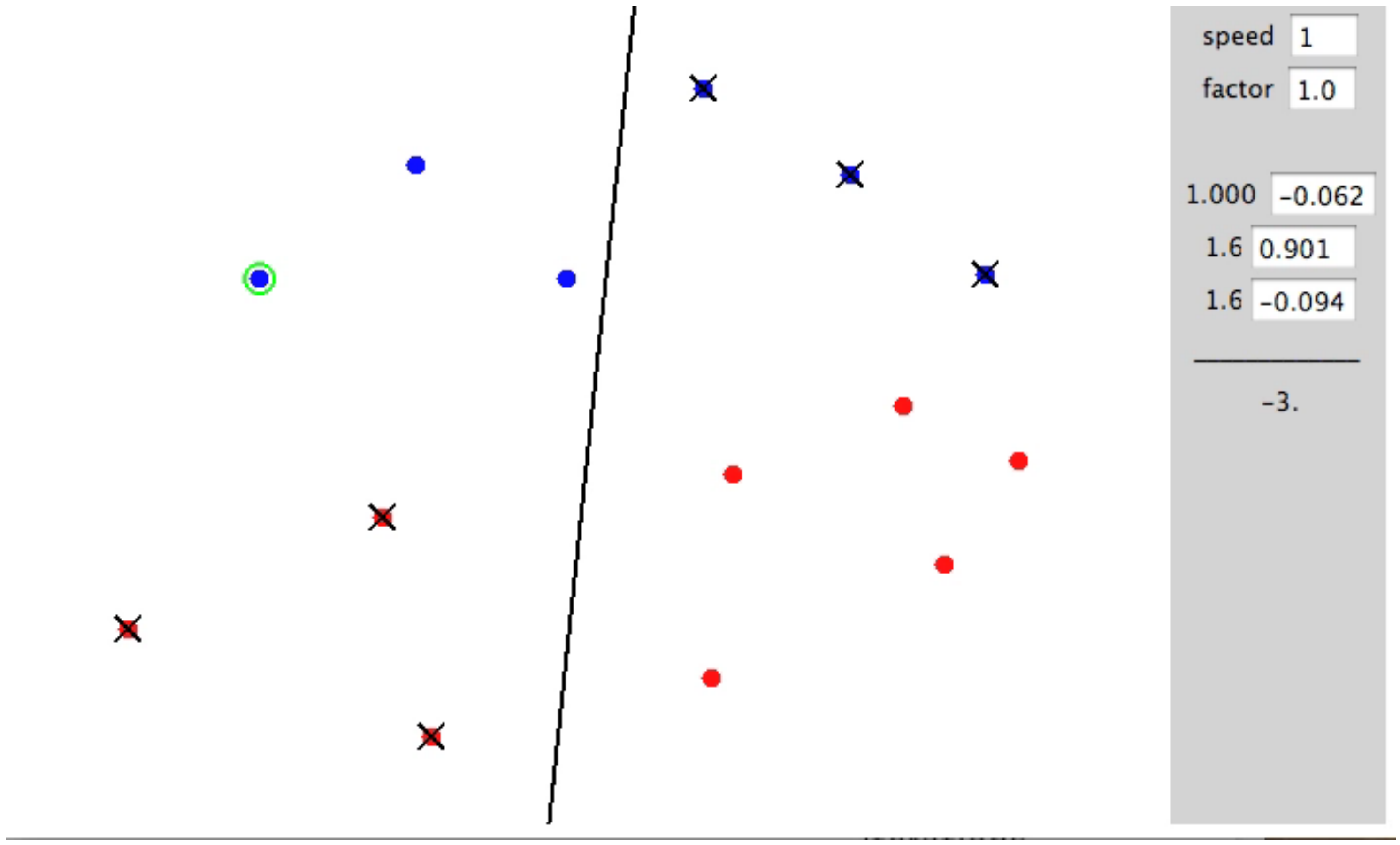
$$y^* = \arg\max_y w \cdot \phi(x_i, y)$$

    - If correct ($y^*==y_i$): no change, goto next example!
    - If wrong: adjust weights

$$w = w + \phi(x_i, y_i) - \phi(x_i, y^*)$$

Example: Perceptron

- The separable case



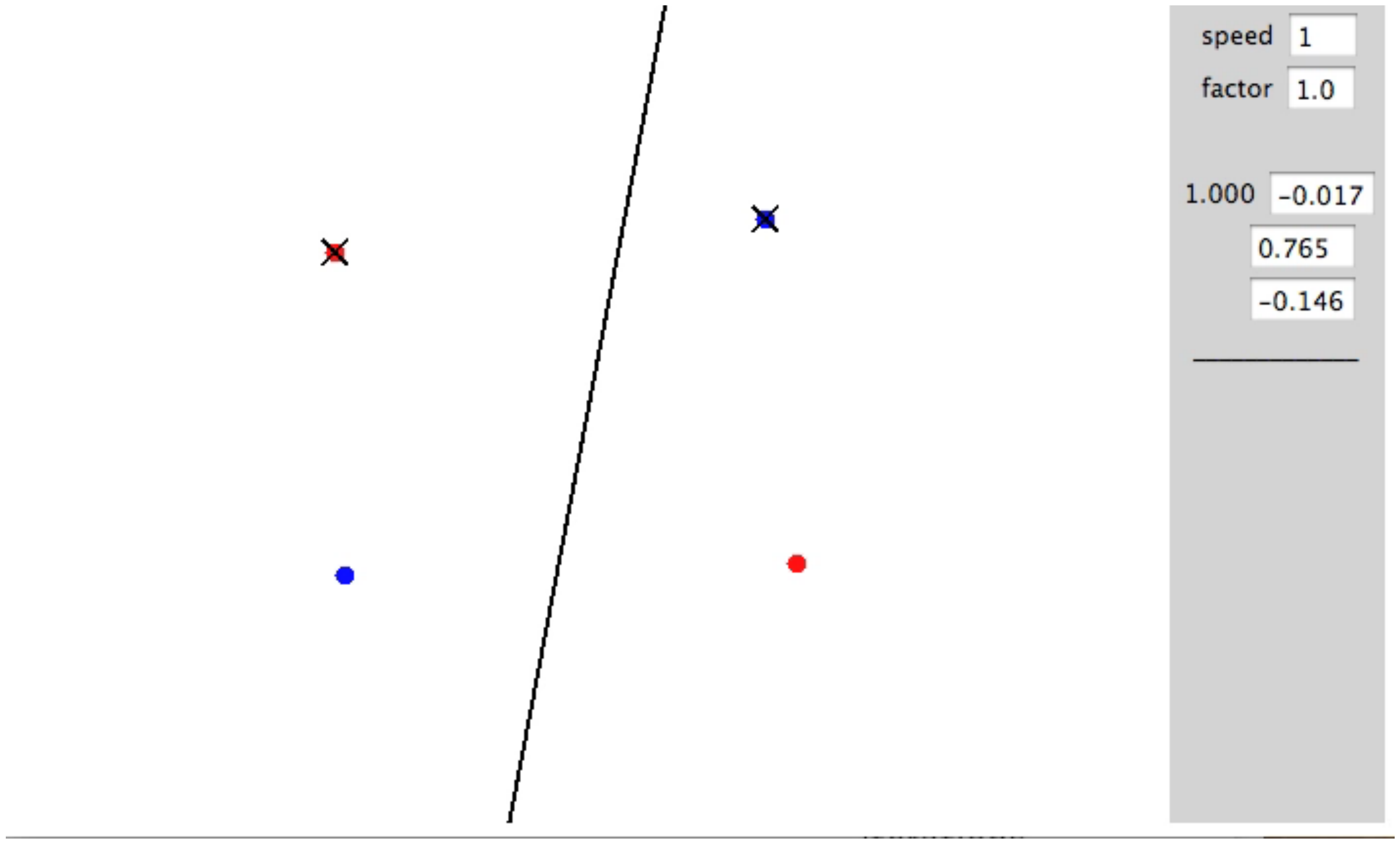
speed 1
factor 1.0

1.000 -0.062
1.6 0.901
1.6 -0.094

-3.

Example: Perceptron

- The inseparable case



speed 1

factor 1.0

1.000 -0.017

0.765

-0.146

Properties of Perceptrons

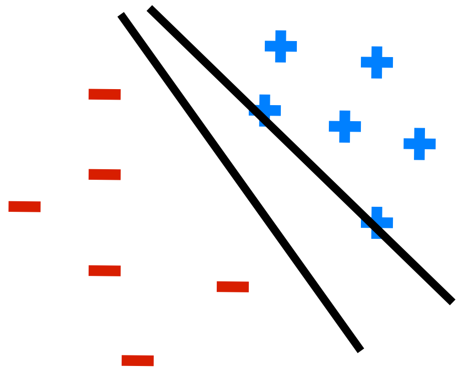
- Separability: some parameters get the training set perfectly correct

- Convergence: if the training is separable, perceptron will eventually converge

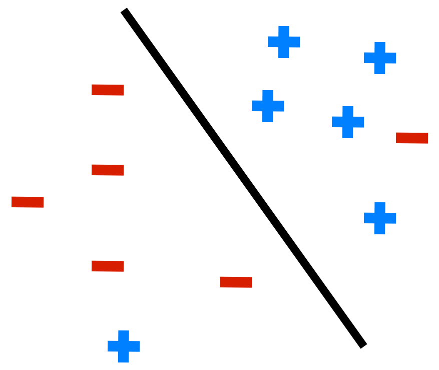
- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable

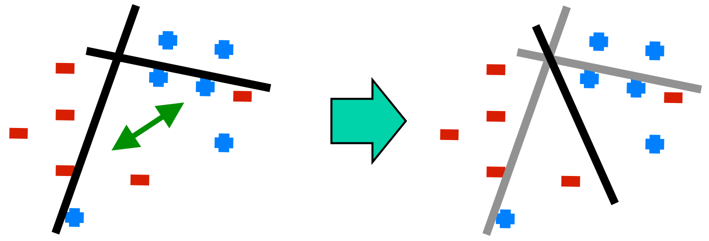


Non-Separable

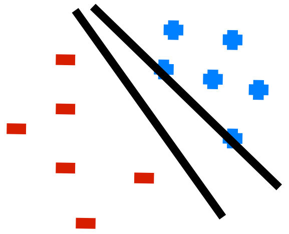


Problems with the Perceptron

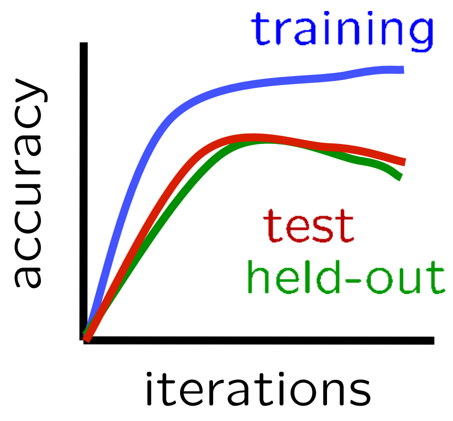
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



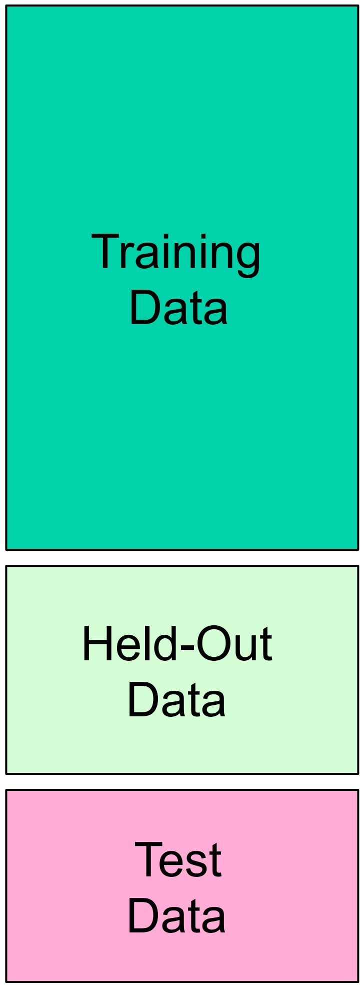
- Mediocre generalization: finds a "barely" separating solution



- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



Summary: Three Views of Classification



- Naïve Bayes:
  - Parameters from data statistics
  - Parameters: probabilistic interpretation
  - Training: one pass through the data
- Log-linear models:
  - Parameters from gradient ascent
  - Parameters: linear, probabilistic model, and discriminative
  - Training: gradient ascent (usually batch), regularize to stop overfitting
- The Perceptron:
  - Parameters from reactions to mistakes
  - Parameters: discriminative interpretation
  - Training: go through the data until held-out accuracy maxes out