Log Linear Models for Text Classification

Mausam

(Slides by Michael Collins, Emily Fox, Alexander Ihler, Dan Jurafsky, Dan Klein, Chris Manning, Ray Mooney, Mark Schmidt, Dan Weld, Alex Yates, Luke Zettlemoyer)

Introduction

- So far we've looked at "generative models"
  - Naive Bayes
- But there is now much use of conditional or discriminative probabilistic models in NLP, Speech, IR (and ML generally)
- Because:
  - They give high accuracy performance
  - They make it easy to incorporate lots of linguistically important features
  - They allow automatic building of language independent, retargetable NLP modules

Joint vs. Conditional Models

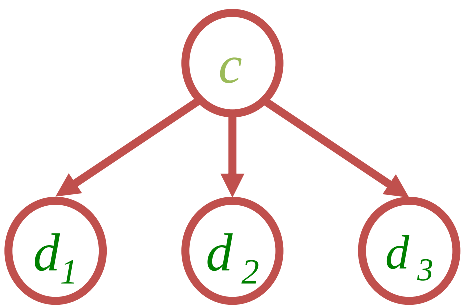
- We have some data {(*d*, *c*)} of paired observations *d* and hidden classes *c*.
- Joint (generative) models place probabilities over both observed data and the hidden stuff (generate the observed data from hidden stuff):
  - All the classic Stat-NLP models:
    - *n*-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars, IBM machine translation alignment models

Joint vs. Conditional Models

- Discriminative (conditional) models take the data as given, and put a probability over hidden structure given the data:
    - Logistic regression, conditional loglinear or maximum entropy models, conditional random fields
    - Also, SVMs, (averaged) perceptron, etc. are discriminative classifiers (but not directly probabilistic)

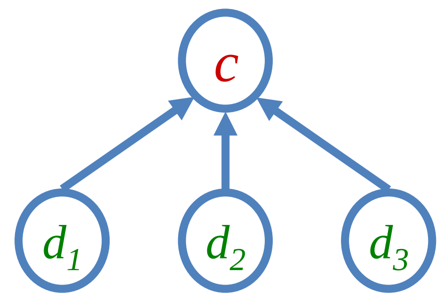
Bayes Net/Graphical Models

- Bayes net diagrams draw circles for random variables, and lines for direct dependencies
- Some variables are observed; some are hidden
- Each node is a little classifier (conditional probability table) based on incoming arcs



Naive Bayes

Generative



Logistic Regression

Discriminative

Conditional vs. Joint Likelihood

- A *joint* model gives probabilities $P(d,c)$ and tries to maximize this joint likelihood.
  - It turns out to be trivial to choose weights: just relative frequencies.
- A *conditional* model gives probabilities $P(c|d)$. It takes the data as given and models only the conditional probability of the class.
  - We seek to maximize conditional likelihood.
  - Harder to do (as we'll see…)
  - More closely related to classification error.

Text Categorization with Word Features

Data BUSINESS: Stocks hit a yearly low …
Label: BUSINESS Features {…, stocks, hit, a, yearly, low, …}

(Zhang and Oles 2001)

- Features are presence of each word in a document and the document class (they do feature selection to use reliable indicator words)
- Tests on classic Reuters data set (and others)
  - Naïve Bayes: 77.0% F_1
  - Logistic regression: 86.4%
  - Support vector machine: 86.5%

Case Study: 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 class
  - 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

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]}

- 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, 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?
  - 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, 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

- Many feature-based classification techniques out there
  - Log-linear models extremely popular in the NLP community!

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.

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 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*

Features

- In NLP uses, usually a feature specifies
  1. an indicator function – a yes/no boolean matching function – of properties of the input and
  2. a particular class

$$\phi_i(x,y) \equiv [\Phi(x) \wedge y = y_j]$$ [Value is 0 or 1]

- Each feature picks out a data subset and suggests a label for it

Example of Features

- context:jail & served:functional
- context:jail & served:dish
- …

- subcat:NP & served:functional
- subcat:NP & served:dish
- …

Feature-Based Linear Classifiers

- Linear classifiers at classification time:
  - Linear function from feature sets $\{\phi_i\}$ to classes $\{y\}$.
  - Assign a weight w_i to each feature ϕ_i.
  - We consider each class for an observed datum x
  - For a pair (x,y), features vote with their weights:
    - vote(y) = $\Sigma w_i \phi_i(x,y)$
    - Choose the class y which maximizes $\Sigma w_i \phi_i(x,y)$
  - We need probabilistic semantics to this method.
    - Log linear classifiers

Exponential Models
(log-linear, maxent, Logistic, Gibbs)

- Model: use the scores as probabilities:

$$p(y|x;w) = \frac{\exp\left(w \cdot \phi(x,y)\right)}{\sum_{y'} \exp\left(w \cdot \phi(x,y')\right)}$$

Make positive

Normalize

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

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

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

Feature-Based Linear Classifiers

- Exponential (log-linear, maxent, logistic, Gibbs) models:
  - Given this model form, we will choose parameters $\{w_i\}$ that *maximize the conditional likelihood* of the data according to this model.
  - We construct not only classifications, but probability distributions over classifications.
    - There are other (good!) ways of discriminating classes – SVMs, boosting, even perceptrons – but these methods are not as trivial to interpret as distributions over classes.

Derivative of Log-linear Model

$$p(y|x;w) = \frac{\exp\left(w \cdot \phi(x,y)\right)}{\sum_{y'} \exp\left(w \cdot \phi(x,y')\right)}$$

- 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

Proof
(Conditional Likelihood Derivative)

- Recall

$$\boxed{p(y|x;w) = \frac{\exp\left(w \cdot \phi(x,y)\right)}{\sum_{y'} \exp\left(w \cdot \phi(x,y')\right)}}$$

$$P(Y \mid X, w) = \prod_{(x,y)\in D} p(y \mid x, w)$$

- We can separate this into two components:

$$\log P(Y \mid X, w) = \sum_{(x,y)\in D} \log \exp \sum_i w_i \phi_i(x,y) \quad - \sum_{(x,y)\in D} \log \sum_{y'} \exp \sum_i w_i \phi_i(x,y')$$

- The derivative is the difference between the derivatives of each component

$$\log P(Y \mid X, w) = N(w) \ - \ D(w)$$

Proof: Numerator

$$\frac{\partial N(w)}{\partial w_i} = \frac{\partial \sum_{(x,y)\in D} \log \exp \sum_i w_i \phi_i(x,y)}{\partial w_i} = \frac{\partial \sum_{(x,y)\in D} \sum_i w_i \phi_i(x,y)}{\partial w_i}$$

$$= \sum_{(x,y)\in D} \frac{\partial \sum_i w_i \phi_i(x,y)}{\partial w_i}$$

$$= \sum_{(x,y)\in D} \phi_i(x,y)$$

Derivative of the numerator is: the empirical count(ϕ_i, y)

Proof: Denominator

$$\frac{\partial D(w)}{\partial w_i} = \frac{\partial \sum_{(x,y)\in D} \log \sum_{y'} \exp \sum_i w_i \phi_i(x,y')}{\partial w_i}$$

$$= \sum_{(x,y)\in D} \frac{1}{\sum_{y''} \exp \sum_i w_i \phi_i(x,y'')} \frac{\partial \sum_{y'} \exp \sum_i w_i \phi_i(x,y')}{\partial w_i}$$

$$= \sum_{(x,y)\in D} \frac{1}{\sum_{y''} \exp \sum_i w_i \phi_i(x,y'')} \sum_{y'} \frac{\exp \sum_i w_i \phi_i(x,y')}{1} \frac{\partial \sum_i w_i \phi_i(x,y')}{\partial w_i}$$

$$= \sum_{(x,y)\in D} \sum_{y'} \frac{\exp \sum_i w_i \phi_i(x,y')}{\sum_{y''} \exp \sum_i w_i \phi_i(x,y'')} \frac{\partial \sum_i w_i \phi_i(x,y')}{\partial w_i}$$

$$= \sum_{(x,y)\in D} \sum_{y'} P(y' \mid x, w)\, \phi_i(x,y')$$

= predicted count(ϕ_i, w)

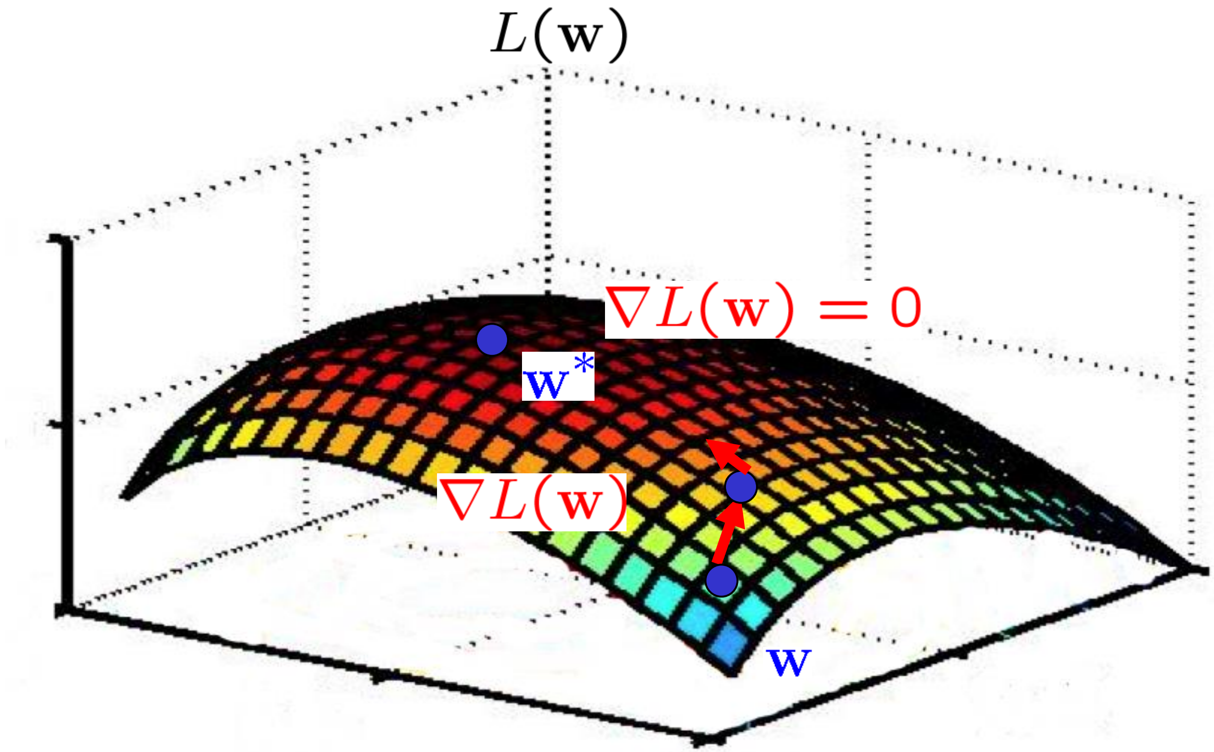
Proof (concluded)

$$\frac{\partial \log P(Y \mid X, w)}{\partial w_i} = \text{actual count}(\phi_i, Y) - \text{predicted count}(\phi_i, w)$$

- 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 feature counts are from actual data).
- These models are also called maximum entropy models because we find the model having maximum entropy and satisfying the constraints:

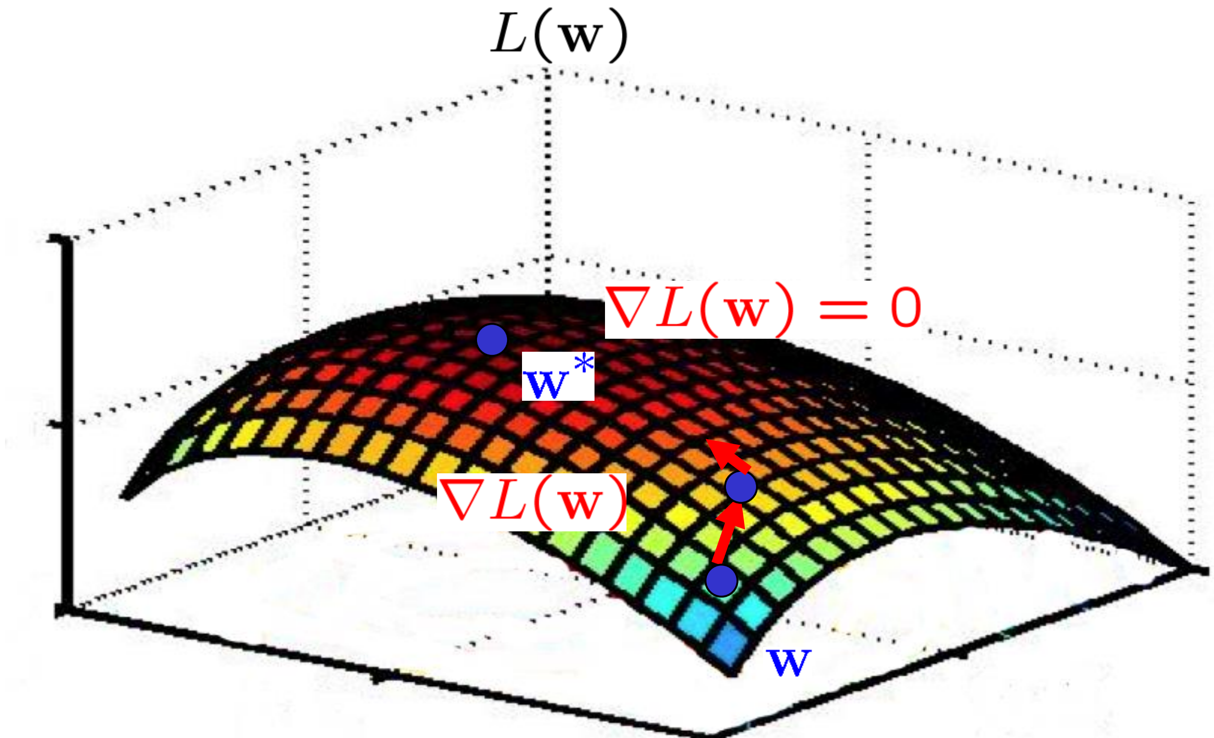
$$E_p(\phi_i) = E_{\tilde{p}}(\phi_i), \forall i$$

Unconstrained Optimization



- 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



- 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 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, $\text{argmax}_w L(w)$ 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

L1 and L2 Regularization

L2 Regularization for Log-linear models

- Instead, we worry about large feature weights
- Add a regularization term to the likelihood to push weights towards zero

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

Regularization Constant

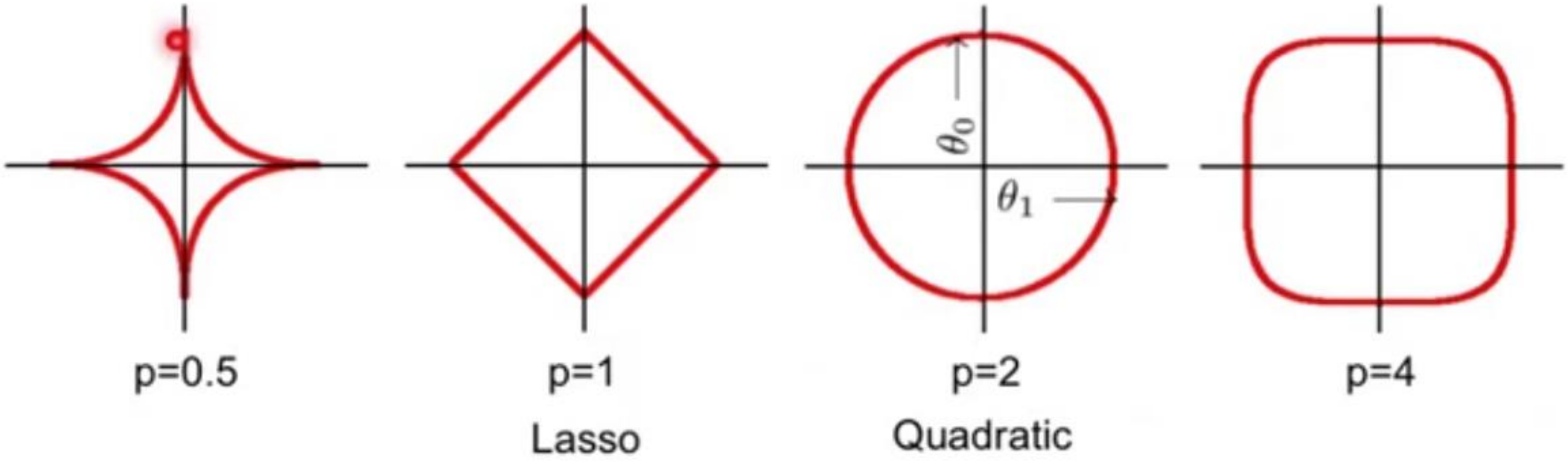
L1 Regularization for Log-linear models

- Instead, we worry about number of active features
- Add a regularization term to the likelihood to push weights to zero

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

L_p Norms for Regularization

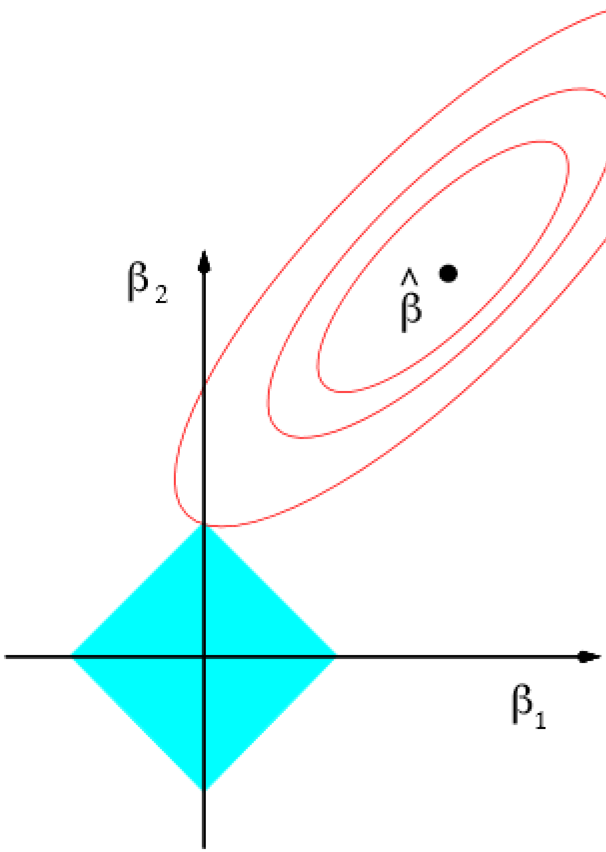
Isosurfaces: $||\theta||_p$ = constant

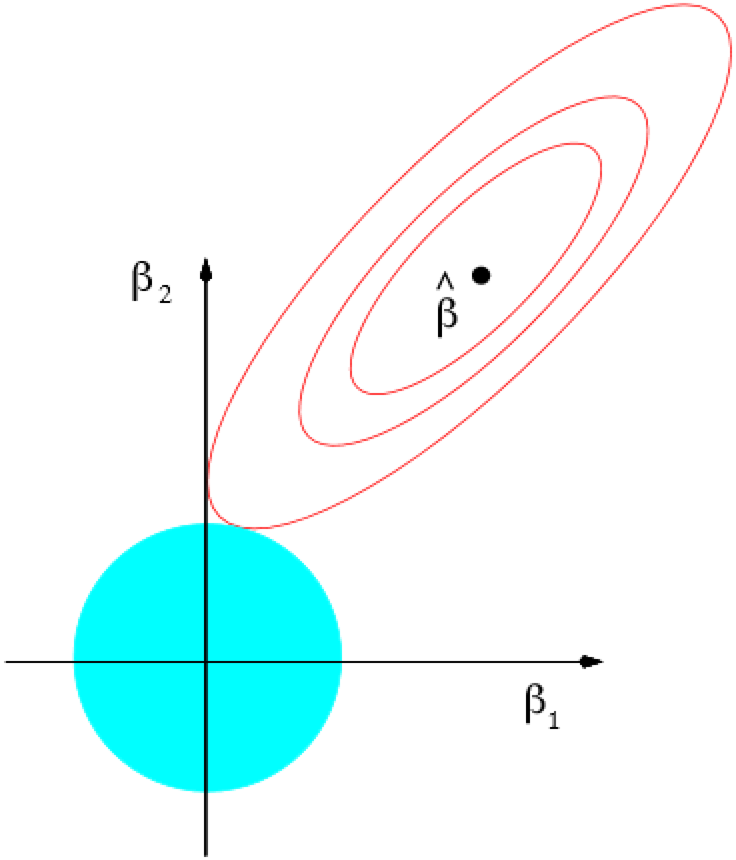


L_0 = limit as $p \to 0$: "number of nonzero weights", a natural notion of complexity

L1 vs L2

- Optimizing L1 harder
  - Discontinuous objective function
  - Subgradient descent versus gradient descent





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?
  - May not (does not) generalize to test set
  - Easy to overfit
- Use devset

Gradient Descent & Large Training Data

repeat

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \frac{\partial L}{\partial w}$$

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \left[\frac{1}{N} \sum_{i=1}^{N} \left(\phi_j(x_i, y_i) - \sum_y p(y \mid x_i, w^{(t)}) \phi_j(x_i, y) \right) - \lambda w_j^{(t)} \right]$$

until convergence

Prohibitive for large datasets

Stochastic Gradient Descent

repeat

$$\cancel{w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta\left[\frac{1}{N}\sum_{i=1}^{N}\left(\phi_j(x_i, y_i) - \sum_y p(y \mid x_i, w^{(t)})\phi_j(x_i, y)\right) - \lambda w_j^{(t)}\right]}$$

until convergence

Use gradient at current point as approx. for avg gradient!

repeat

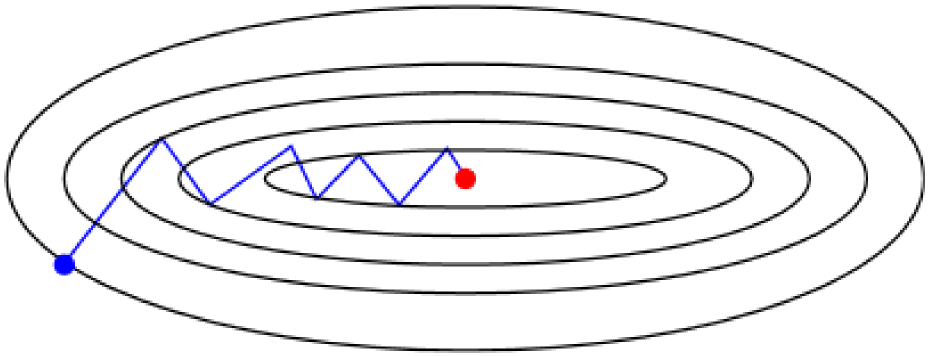
$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta^{(t)}\left[\phi_j(x_i, y_i) - \sum_y p(y \mid x_i, w^{(t)})\phi_j(x_i, y) - \lambda w_j^{(t)}\right]$$

until convergence

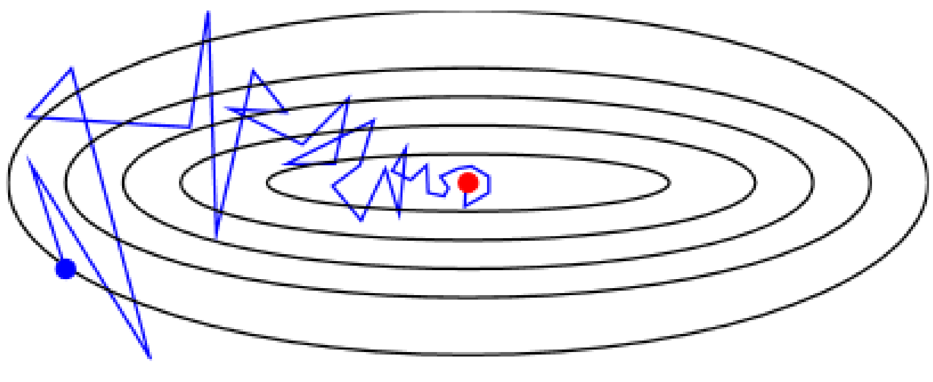
Reduce learning rate slowly (e.g., as η/t)

SGD vs. GD

- Deterministic gradient method [Cauchy, 1847]:

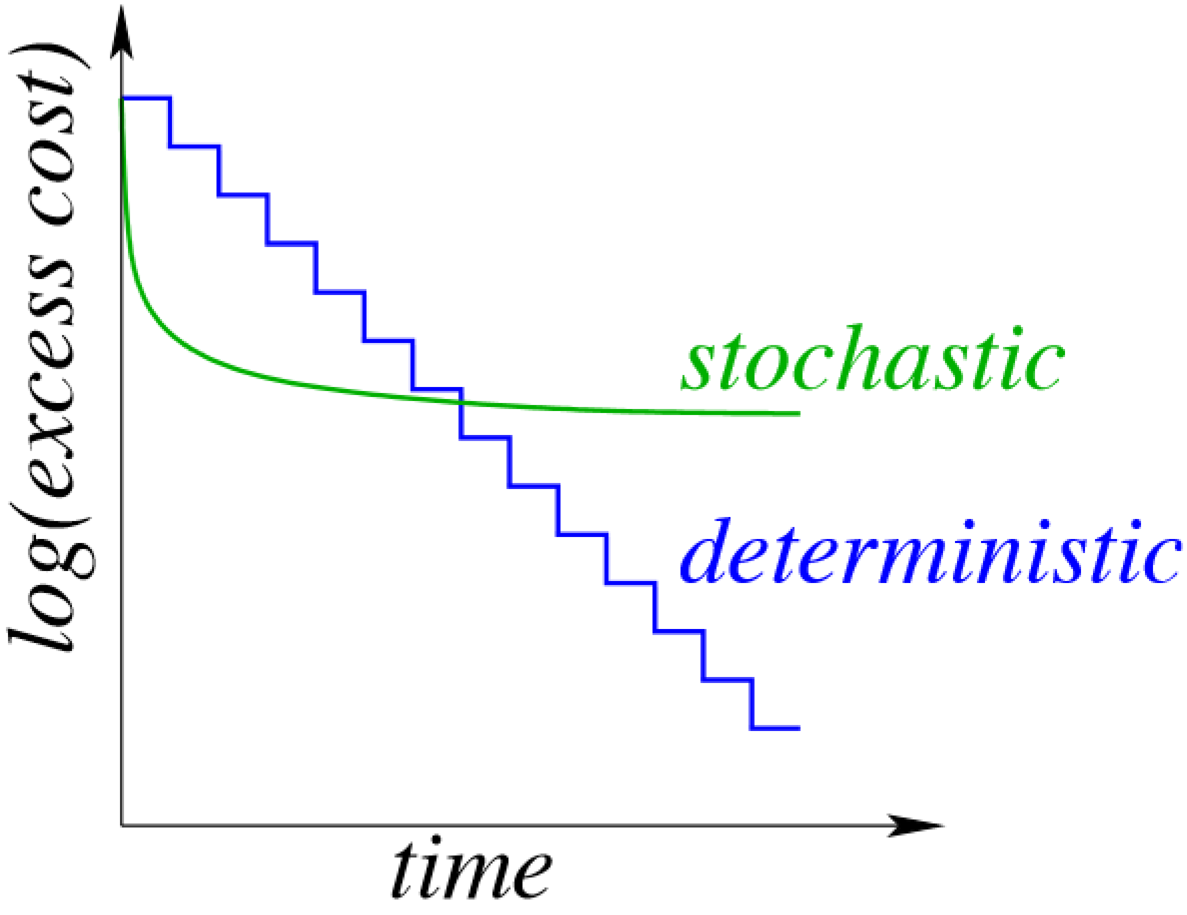


- Stochastic gradient method [Robbins & Monro, 1951]:



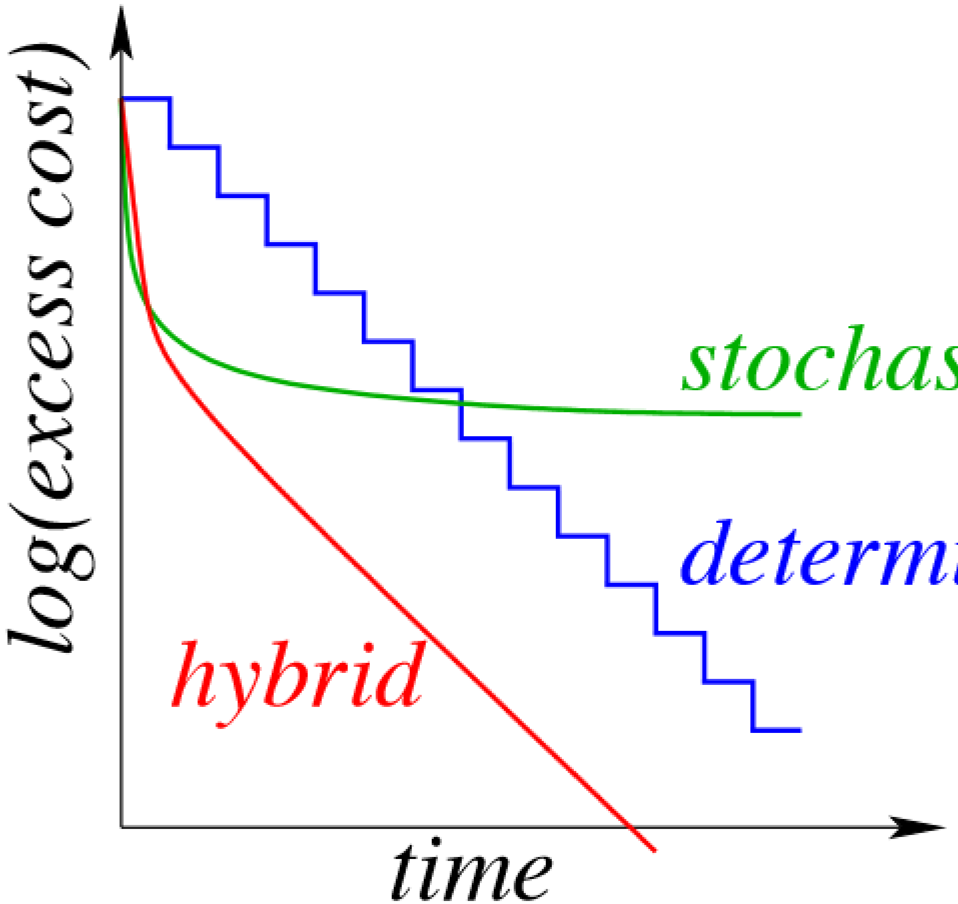
Convergence rates

- GD: O(1/t^2), SGD: O(1/sqrt(t))



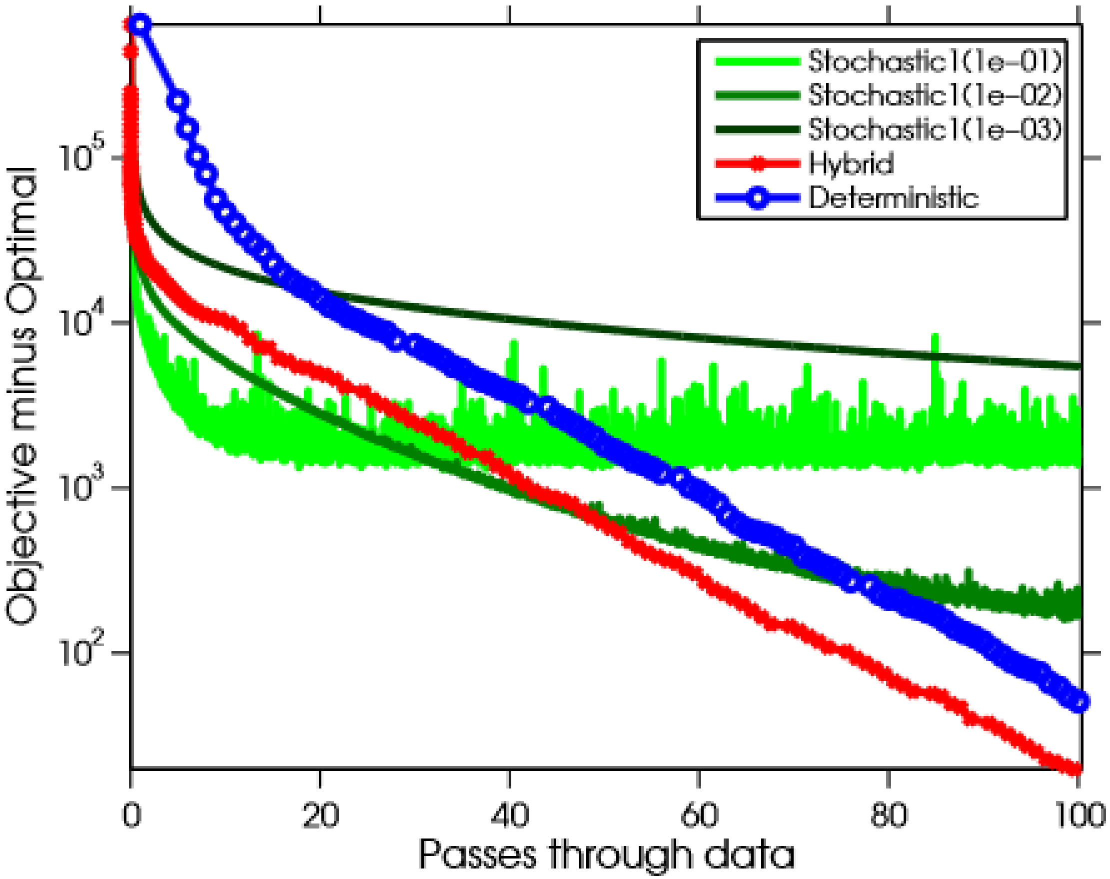
Stochastic will be superior for low-accuracy/time situations.

Hybrid Approaches



Hybrid #1: Batch

- Batch Gradient

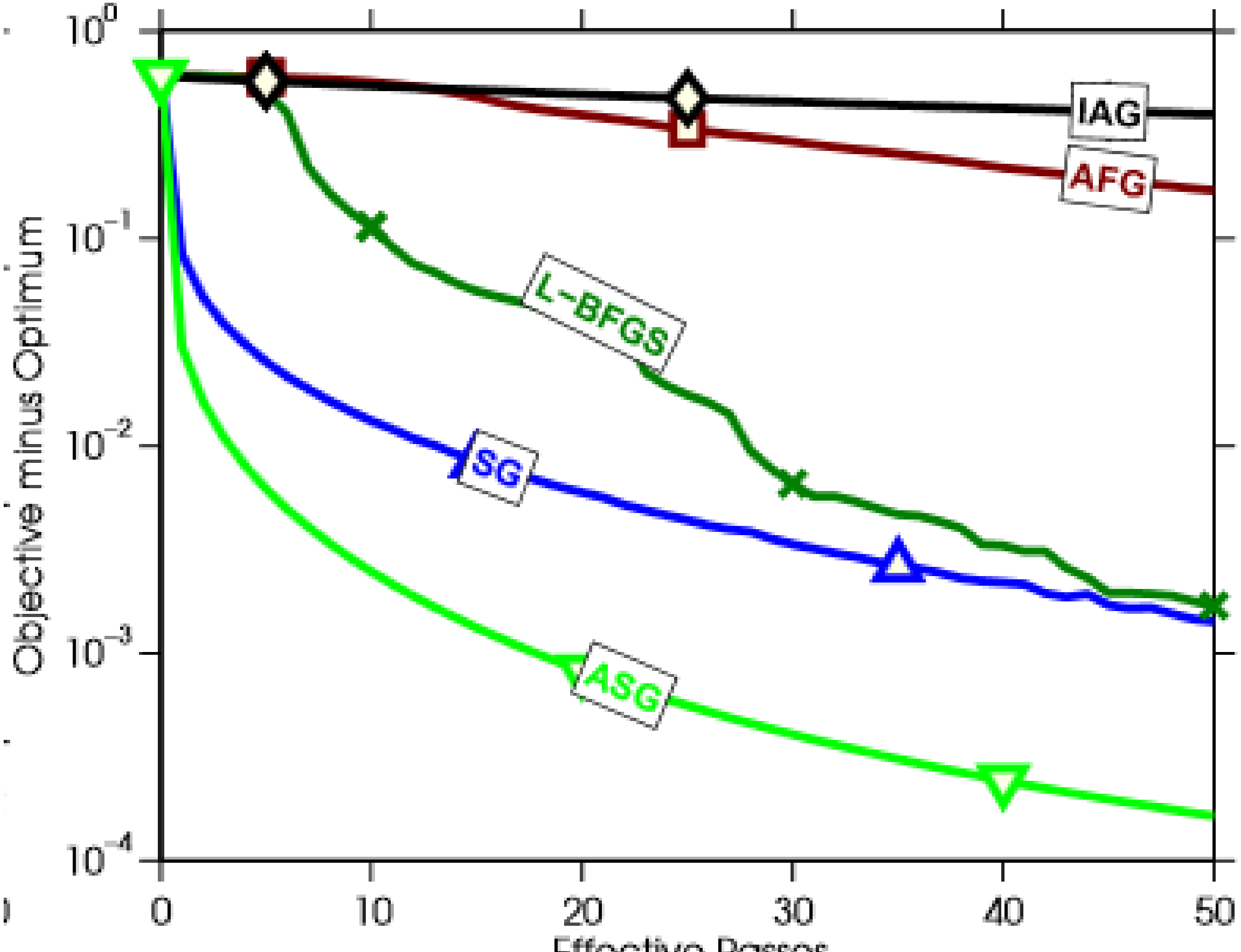


Hybrid #2: Stochastic Avg Gradient

- [Schmidt 2013]
- Use average gradient over all data points
  - Choose a datapt randomly (xi, yi)
  - Compute gradient at (xi,yi)
  - Recompute a new average gradient
    - Replace the prev gradient for (xi, yi) by the new one
  - Do the weight updates

- Assumes gradients of non-selected examples don’t change
- Better theoretical and practical convergence

Stochastic Avg Gradient



Word Sense Disambiguation Results

[Suarez and Palomar, 2002]

- With clever features, small variations on simple log-linear 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