Scaling up with Online Learning

Colin Cherry
(with a few slides borrowed from Hal Daumé III)

Who is this guy?

- Research Officer at National Research Council
- Expertise:
  - Learning to classify, annotate or transform text
- Applications:
  - Machine translation
  - Medical text processing

Goal

- Highlight and address problems with scaling machine learning classifiers
- Convey the strengths and weaknesses of various online algorithms
- Convince you that online algorithms are:
  - (a) Efficient, easy
  - (b) Not sketchy

Outline

- What are common problems of scale?
- Generalizing linear classification
- Training linear classifiers online:
  - Perceptron
  - Passive Aggressive / MIRA
  - Structured SVM
- Algorithm Comparison

What do we mean by scale?

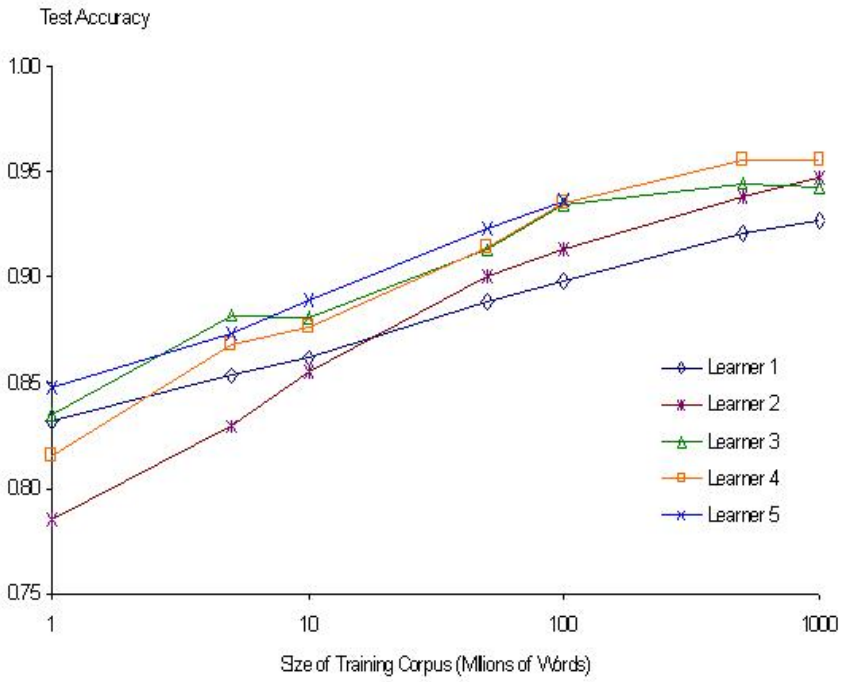
1. Too much training data
2. Too many features
3. Too many classes

Too much training data

- For many tasks, if you can get one example, you can get millions:
  - Example: predict whether a Tweet will get a reply
- Memory (RAM) is almost always the bottleneck
  - If your method relies on N x N matrices, you will be dead in the water
  - Similarly it may be impossible to load every training example into memory at once

PS: There is no such thing as "Too much data"

- "*It never pays to think until you've run out of data*" – Eric Brill, head of eBay Research



Too Many Features

- To use lots of features, vectors use sparse representations:

- Dense: 0,0,0,1,0,0,0,0,5,0,0,0,0,-3,0,0,0,0,0,0,2
- Sparse: 3:1, 8:5, 13:-3, 20:2

- Many large scale learning packages (such as liblinear) use this notation.
- I'll often use strings instead of integers for feature names, for the sake of clarity

Too many features
Bag of Unigrams
10,711

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
it was the epoch of belief,
it was the epoch of incredulity,
it was the season of Light,
it was the season of Darkness,
it was the spring of hope,
it was the winter of despair,
we had everything before us,
we had nothing before us,
we were all going direct to Heaven,
we were all going direct the other way--

the:11
was:10
of:10
it:10
we:4
before:2
times,:2
us,:2
were:2
age:2
all:2
had:2
direct:2
going:2
epoch:2
season:2
…

Too many features
Bag of Unigrams & Bigrams
70,115

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
it was the epoch of belief,
it was the epoch of incredulity,
it was the season of Light,
it was the season of Darkness,
it was the spring of hope,
it was the winter of despair,
we had everything before us,
we had nothing before us,
we were all going direct to Heaven,
we were all going direct the other way--

the:11
was_the:10
of:10
it_was:10
was:10
it:10
we:4
age_of:2
us,:2
all_going:2
were:2
age:2
had:2
going:2
we_had:2
season:2

Too many features
Bag of Unigrams & Bigrams & Trigrams
179,328

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
it was the epoch of belief,
it was the epoch of incredulity,
it was the season of Light,
it was the season of Darkness,
it was the spring of hope,
it was the winter of despair,
we had everything before us,
we had nothing before us,
we were all going direct to Heaven,
we were all going direct the other way--

the:11
was_the:10
of:10
it_was:10
was:10
it:10
it_was_the:10
we:4
was_the_age:2
going:2
we_were:2
we_were_all:2
before:2
times,:2
the_season_of:2
the_epoch:2

Too many features cont'd

- Similar to too much data: you're going to run out of memory
- Most elegant way to handle this is with kernels - allows us to represent training set as N x N matrix
  - Which limits our amount of training data
- Solutions I will present only need one example in memory at a time
  - Never have too much data
  - Never have too many features

Too many classes

- What if we aren't doing binary classification?
  - Say we're annotating doctor's notes for insurance coding – with thousands of codes
- Things are actually still pretty okay
  - "One versus all approach"
  - Train one binary classifier for each class, then run all the classifiers on new data, chose whichever class scores highest
- Not super-fast, but scalable

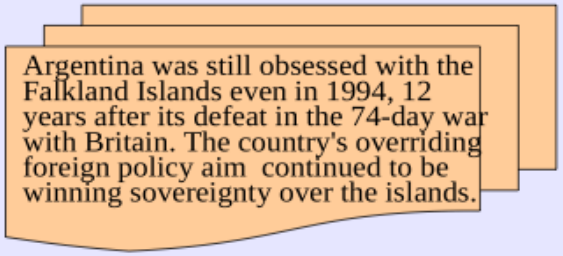
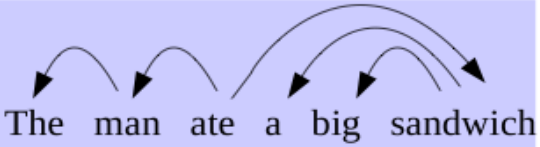
Too many classes cont'd

- What if we have much more than a thousand classes?

- Structured prediction:
  - Number of classes is an exponential function of the size of the input
  - One versus all no longer makes sense

Structured Prediction Examples

- Any task with a structured output

Task	Input	Output
Machine Translation	Ces deux principes se tiennent à la croisée de la philosophie, de la politique, de l'économie, de la sociologie et du droit.	Both principles lie at the crossroads of philosophy, politics, economics, sociology, and law.
Document Summarization		
Syntactic Analysis	The man ate a big sandwich.	
...many more...		

Too many classes cont'd

- We would like to be able to use our classification machinery to train parsers / summarizers / translators

- Fortunately, establishing a framework that does so is straight-forward

Outline

- ✔ What are common problems of scale?
- Generalizing linear classification
- Training linear classifiers online:
  - Perceptron
  - Passive Aggressive / MIRA
  - Structured SVM
- Algorithm Comparison

Generalizing Linear Classification

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \mathrm{argmax}_{y' \in Y(x)} w \cdot \phi(x, y')$$

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$\boxed{y} = \text{argmax}_{y' \in Y(x)} w \cdot \phi(x, y')$$

Class (+1 / -1)

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in Y(\boxed{x})} w \cdot \phi(\boxed{x}, y')$$

Input vector

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in \boxed{Y(x)}} w \cdot \phi(x, y')$$

{+1, -1}

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \boxed{\text{argmax}_{y' \in Y(x)}} w \cdot \phi(x, y')$$

Compare the two classes

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in Y(x)} w \cdot \boxed{\phi(x, y')}$$

Features derived from x and y':

$$\phi(x, y') = \frac{1}{2} \cdot \begin{cases} +x \text{ if } y' = +1 \\ -x \text{ if } y' = -1 \end{cases}$$

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in Y(x)} \boxed{w} \cdot \phi(x, y')$$

Model Weights

(what we want to learn)

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in Y(x)} w \cdot \phi(x, y')$$

Equivalent to:

$$y = \text{sign}(w \cdot x)$$

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \mathrm{argmax}_{y' \in Y(x)} w \cdot \phi(x, y')$$

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$\boxed{y} = \text{argmax}_{y' \in Y(x)} w \cdot \phi(x, y')$$

Translation

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in Y(\boxed{x})} w \cdot \phi(\boxed{x}, y')$$

Source sentence

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in \boxed{Y(x)}} w \cdot \phi(x, y')$$

All possible translations of x

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \boxed{\text{argmax}_{y' \in Y(x)}} w \cdot \phi(x, y')$$

Decoder
Generates y from x, or "decodes" x into y

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in Y(x)} w \cdot \boxed{\phi(x, y')}$$

Decoding Features

Translation Model
Language Model
Distortion Model
Word Penalty

Notation / Assumptions

- Will assume a linear model *w*, with features ϕ that decompose into a tractable search:

$$y = \text{argmax}_{y' \in Y(x)} \boxed{w} \cdot \phi(x, y')$$

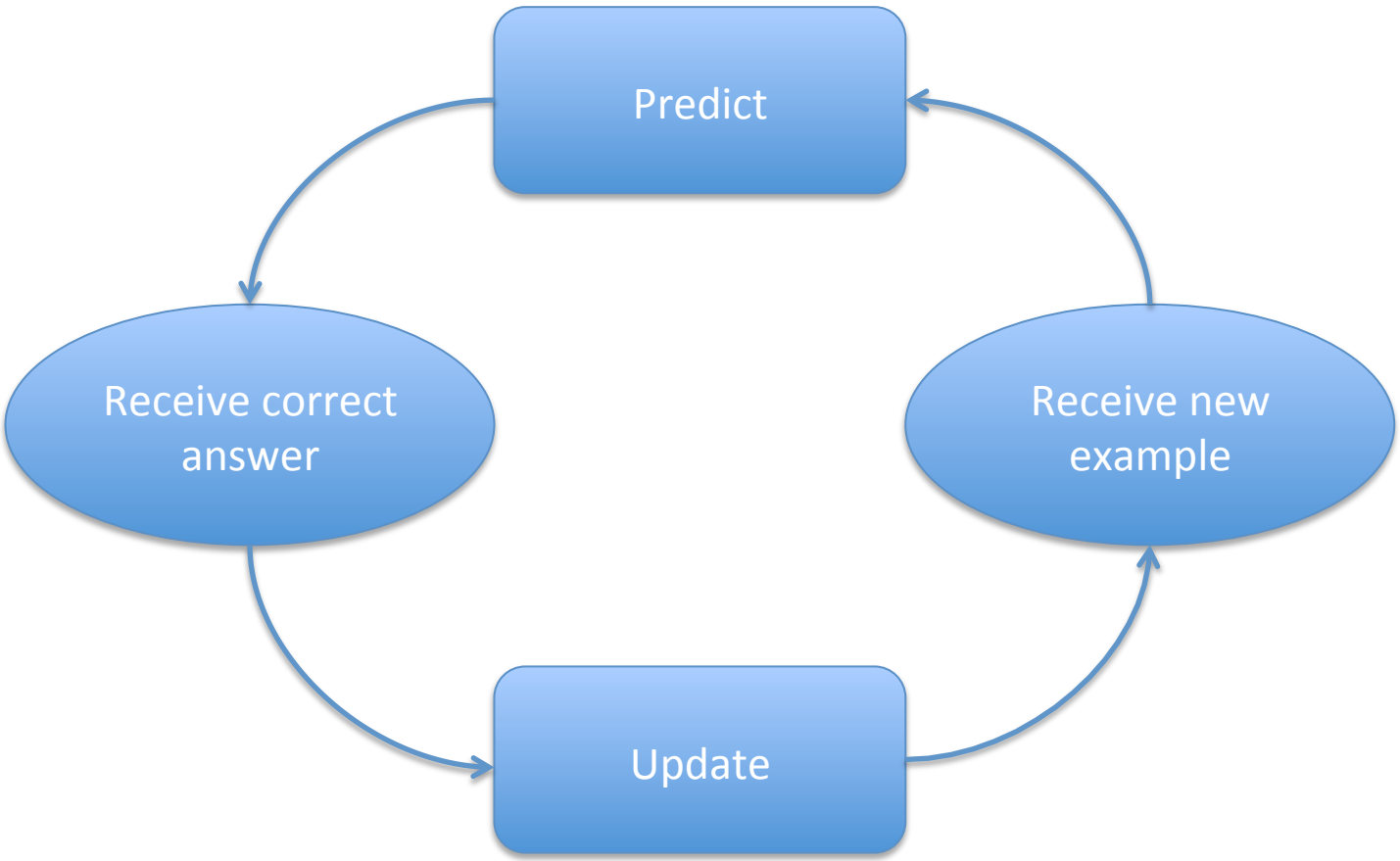
Model Weights

(what we want to learn)

Outline

- ✔ What are common problems of scale?
- ✔ Generalizing linear classification
- Training linear classifiers online:
  - Perceptron
  - Passive Aggressive / MIRA
  - Structured SVM
- Algorithm Comparison

Online learning



Perceptron Goal / Intuition

- Setting:
  - System receives a series of examples, makes predictions
  - After each example, it receives the correct label
  - Adjust weights to get that example (and hopefully others) right next time

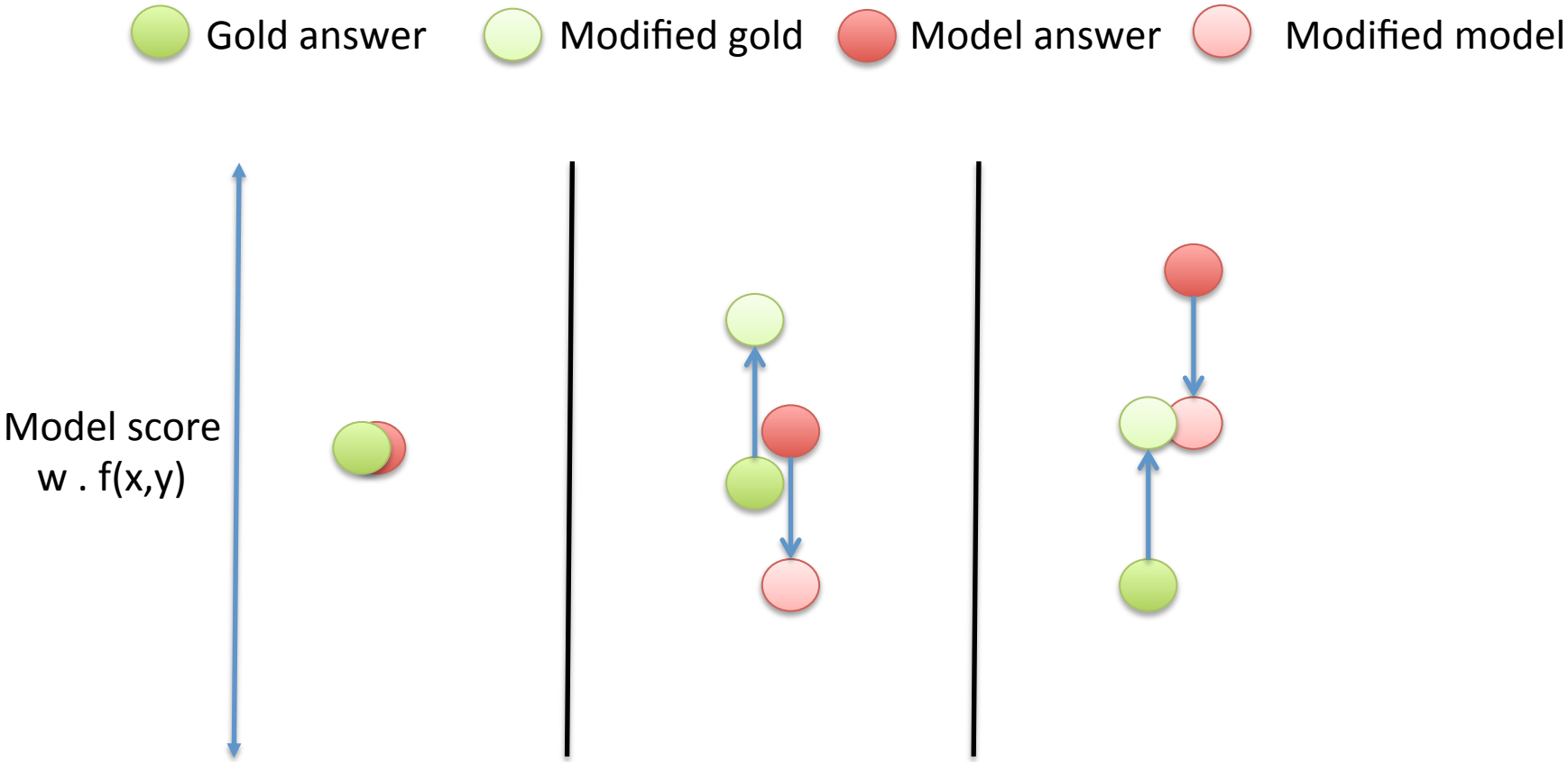
- Goal:
  - Minimize the cumulative number of mistakes

The algorithm

- For each training example (x,y):
  - Decode: $y' = \mathrm{argmax}_{\bar{y} \in Y(x)} w \cdot \phi(x, \bar{y})$
  - If y' != y
    - Update: $w = w + \phi(x, y) - \phi(x, y')$

- Do this for many passes through the data, until accuracy on a held-out set degrades

The Perceptron, Visually



Averaging

- Perceptron makes a big adjustment with each training example
  - Model will overfit to the last few examples
- Solution:
  - Return the average of all weight vectors seen during training
  - Greatly improves performance on unseen data
- Implementation:
  - When feature sets are large, tracking this average is expensive
  - There are tricks to update the average in a lazy fashion

From Online to Batch

- Online learning sees one training example at a time, batch gets a training set

- What I have a described is an online algorithm (the perceptron) adapted to a batch setting

- Passing through the data multiple times and averaging the parameter vector make up a general online-to-batch conversion strategy

Perceptron Advantages

- Very easy to code – if you can decode, you can train
- Update is lightning fast – decoder is only bottleneck
- Has some nice feature selection properties:
  - Only cares about examples it gets wrong
  - Only cares about outputs that rise to the top of the argmax
  - Can leverage this with a lazy weight vector that populates itself on demand
- Because it only needs to work with one example at a time, it is extremely memory efficient

Perceptron Disadvantages

- Feels a little sketchy
  - The averaging step is awkward (output weights don't necessarily match any weights used during training)
  - Can prove that it will do something reasonable, but where is my learning objective? SVM maximizes the margin for a training set – what is this doing?
- Lots of possible modifications / improvements
  - Visit examples in random order
  - Use a learning rate
  - Many others…

PA / MIRA

- Passive Aggressive

  or

- Margin Infused Relaxed Algorithm

PA: A lot like the perceptron, but…

- Incorporates **loss** (and through it, margin)
  - Updates on more examples (uses more data)
  - Provides a clear way to make some answers less wrong than others

- Modifies the step size for each example
  - Faster convergence

PA Goal / Intuition

- Setting:
  - System sees a series of examples, makes predictions
  - After each example, it receives the correct label and a **loss** based on its prediction
  - Adjust weights to minimize loss on that example (and hopefully others) next time

- Goal:
  - Minimize the cumulative loss

What's a loss?

- A function of w, x and y that indicates how badly w handles x.
- A popular loss is the hinge loss:

$$\max_{y'} \left[\delta(y, y') + w \cdot \phi(x, y')\right] - w \cdot \phi(x, y)$$

$$\text{where } \delta(y, y') = \begin{cases} 1 & \text{if } y \neq y' \\ 0 & \text{otherwise} \end{cases}$$

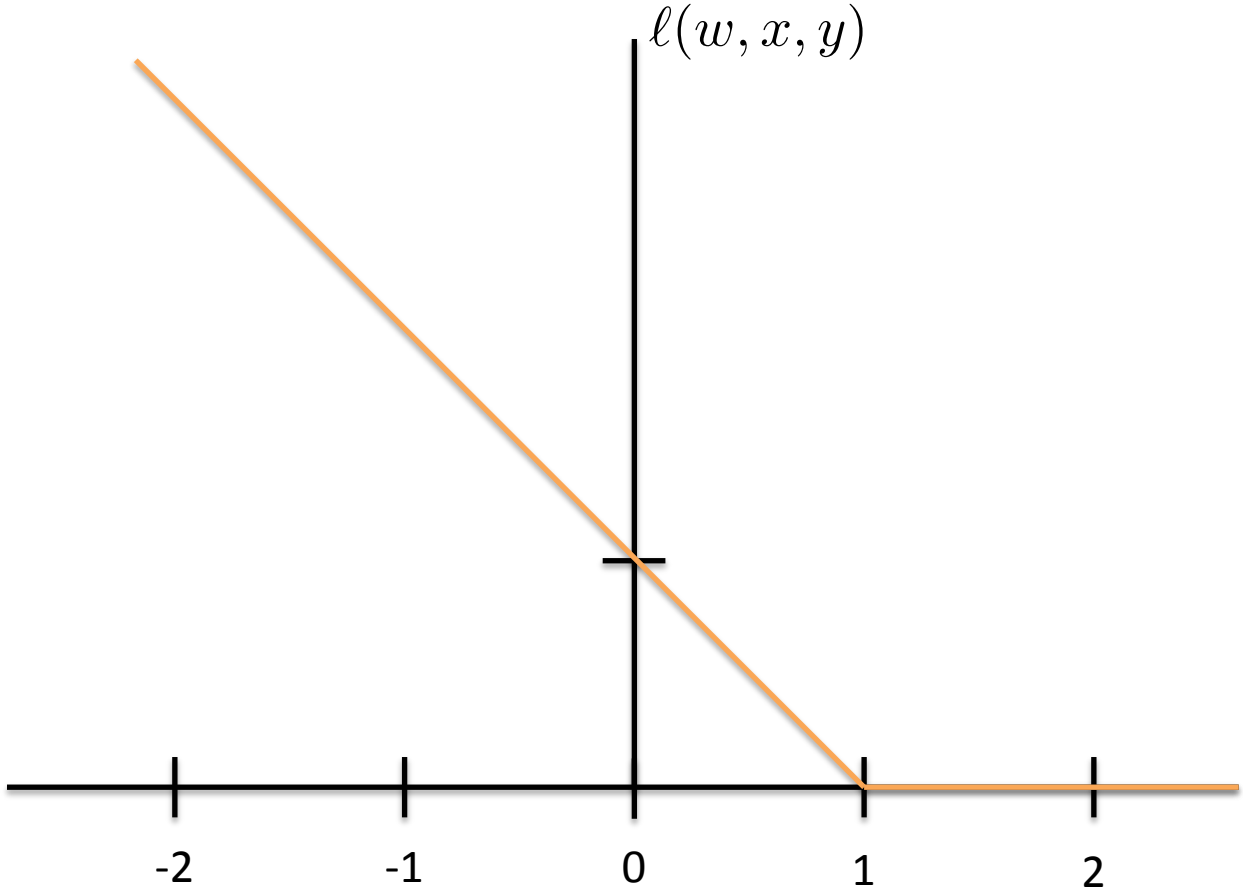
What's a loss?

- A function of w, x and y that indicates how badly w handles x.
- A popular loss is the hinge loss:

$$\max_{y'} \boxed{[\delta(y, y') + w \cdot \phi(x, y')]} - \boxed{w \cdot \phi(x, y)}$$

$$\text{where } \delta(y, y') = \begin{cases} 1 & \text{if } y \neq y' \\ 0 & \text{otherwise} \end{cases}$$

Hinge Loss Visually



$$w \cdot \phi(x, y) - \max_{y' \neq y} \left[w \cdot \phi(x, y') \right]$$

Effect of Hinge Loss

- You're only happy (0 loss) when the model ranks the gold-standard answer above the others by a margin of 1 – kind of like an SVM!

Redefining $\delta(y, y')$

- Handles cases where some answers are more wrong than others:

Pro	Md	Vb	Dt	Nn	0
Pro	Md	Md	Dt	Vb	1
Pro	Md	Md	Dt	Nn	1
Pro	Md	Nn	Dt	Md	1
Pro	Md	Nn	Dt	Nn	1
Pro	Md	Vb	Dt	Md	1
Pro	Md	Vb	Dt	Vb	1
I	can	can	a	can	

Redefining $\delta(y, y')$

- Handles cases where some answers are more wrong than others:

Pro	Md	Vb	Dt	Nn	0
Pro	Md	~~Nn~~	Dt	~~Vb~~	0.4
Pro	Md	~~Nn~~	Dt	Nn	0.2
Pro	Md	~~Nn~~	Dt	~~Nn~~	0.4
Pro	Md	~~Nn~~	Dt	Nn	0.2
Pro	Md	Vb	Dt	~~Nn~~	0.2
Pro	Md	Vb	Dt	~~Vb~~	0.2
I	can	can	a	can	

The algorithm (max loss variant)

- For each training example (x,y):
  - Decode: $y' = \text{argmax}_{\bar{y} \in Y(x)} \left[\delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \right]$
  - Update: $w = \text{argmin}_{\bar{w}} \frac{1}{2} \|w - \bar{w}\|^2$

$$\text{s.t. } \bar{w} \cdot \phi(x, y) - \bar{w} \cdot \phi(x, y') \geq \delta(y, y')$$

- Do this for many passes through the data, until accuracy on a held-out set degrades

The algorithm (max loss variant)

- For each training example (x,y):
  - Decode: $y' = \text{argmax}_{\bar{y} \in Y(x)} \left[\delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \right]$
  - Update: $w = \text{argmin}_{\bar{w}} \frac{1}{2} \|w - \bar{w}\|^2$

$$\text{s.t. } \bar{w} \cdot \phi(x, y) - \bar{w} \cdot \phi(x, y') \geq \delta(y, y')$$

- Do this for many passes through the da
until accuracy on a held-out set degrade

Find the max loss response y'

Move the weights as little as possible

Eliminate the loss from this example

Really… a constrained argmin!?!?

- Fortunately, the update has a closed form!

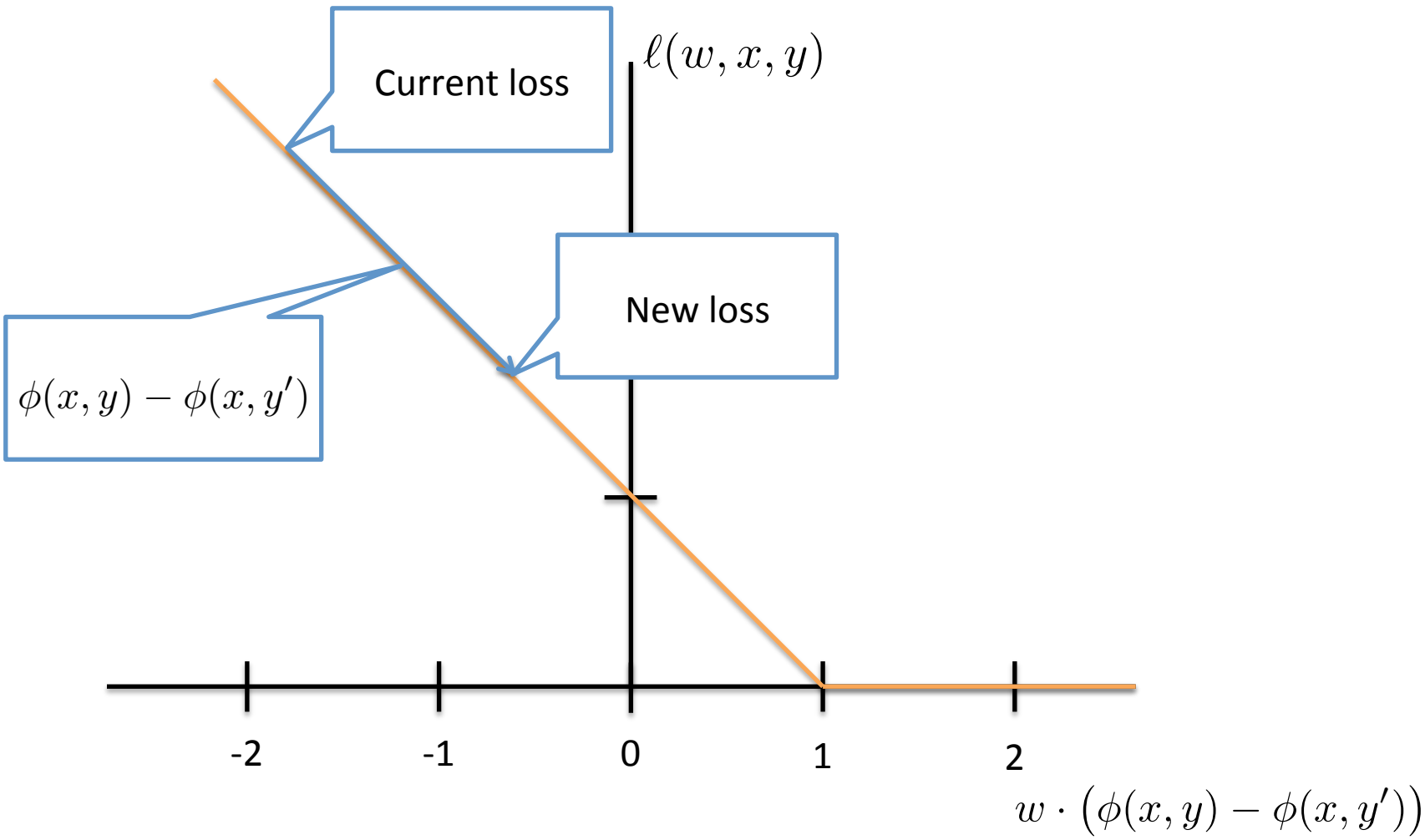
$$w = w + \tau \underbrace{\big(\phi(x,y) - \phi(x,y')\big)}_{\text{Perceptron update}}$$

Adaptive learning rate

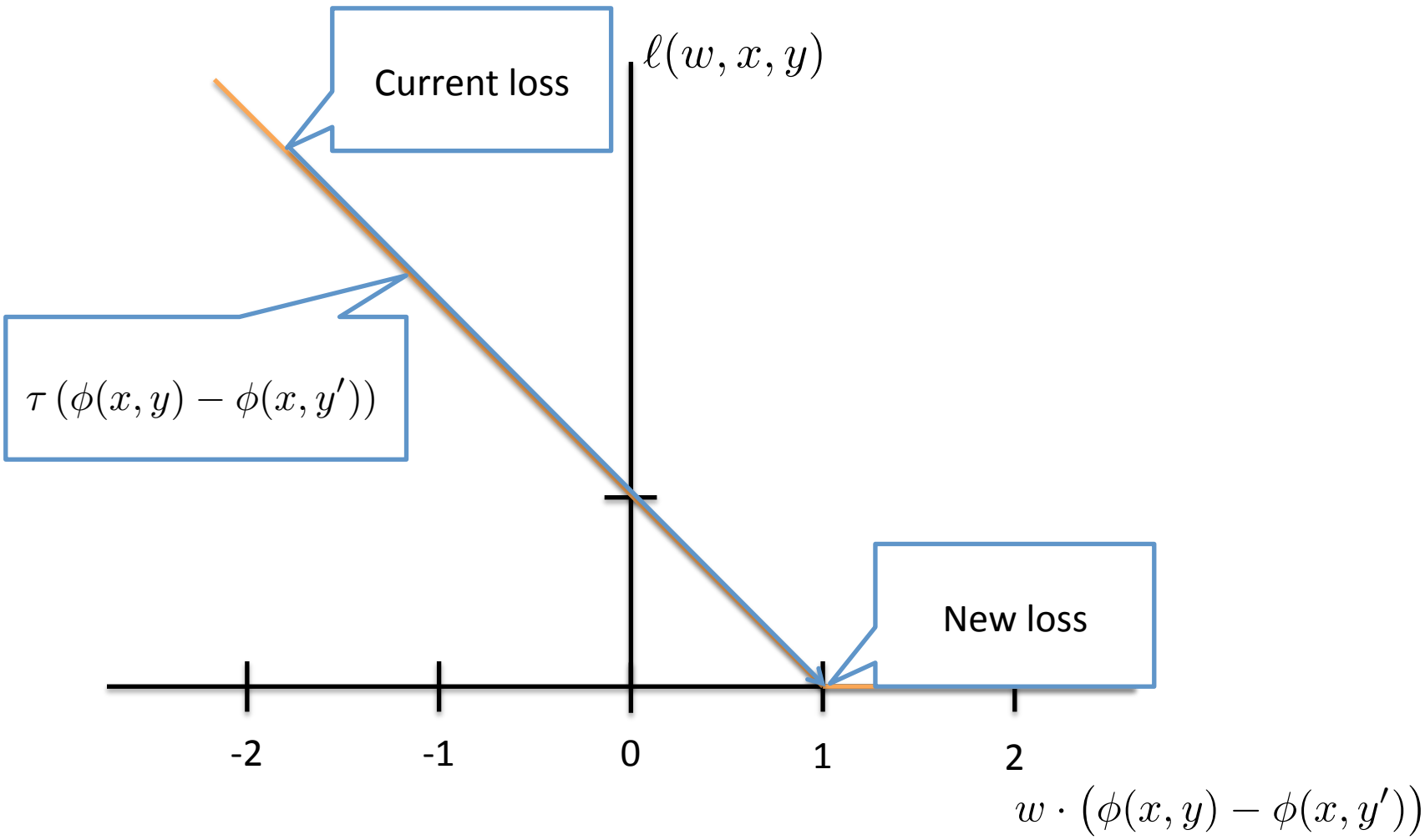
Loss from this example

$$\tau = \frac{\ell(w,x,y)}{\underbrace{\|\phi(x,y) - \phi(x,y')\|}_{\text{Size of the update vector}}}$$

Perceptron and hinge loss



PA and hinge loss

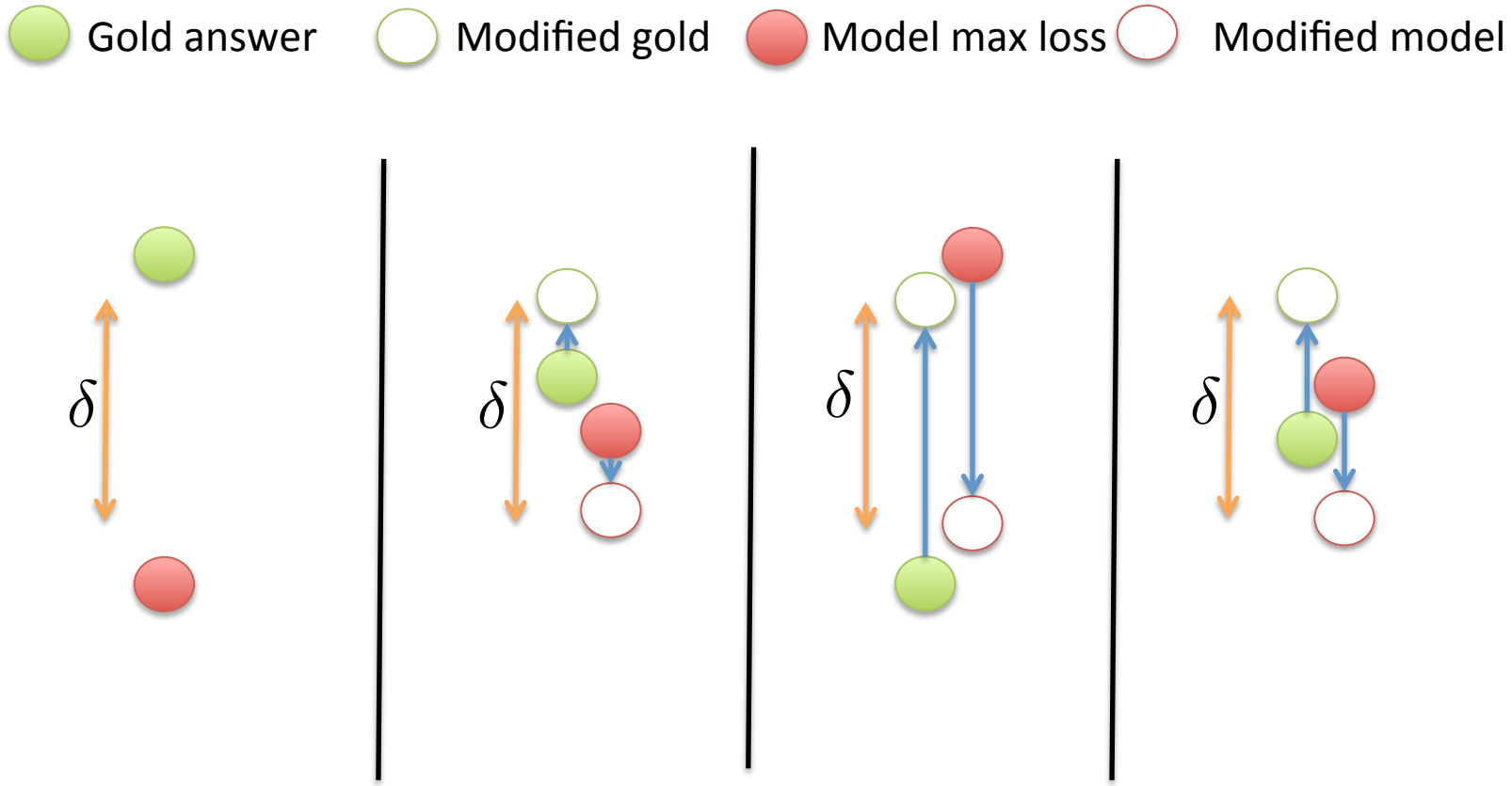


The algorithm (max loss variant)

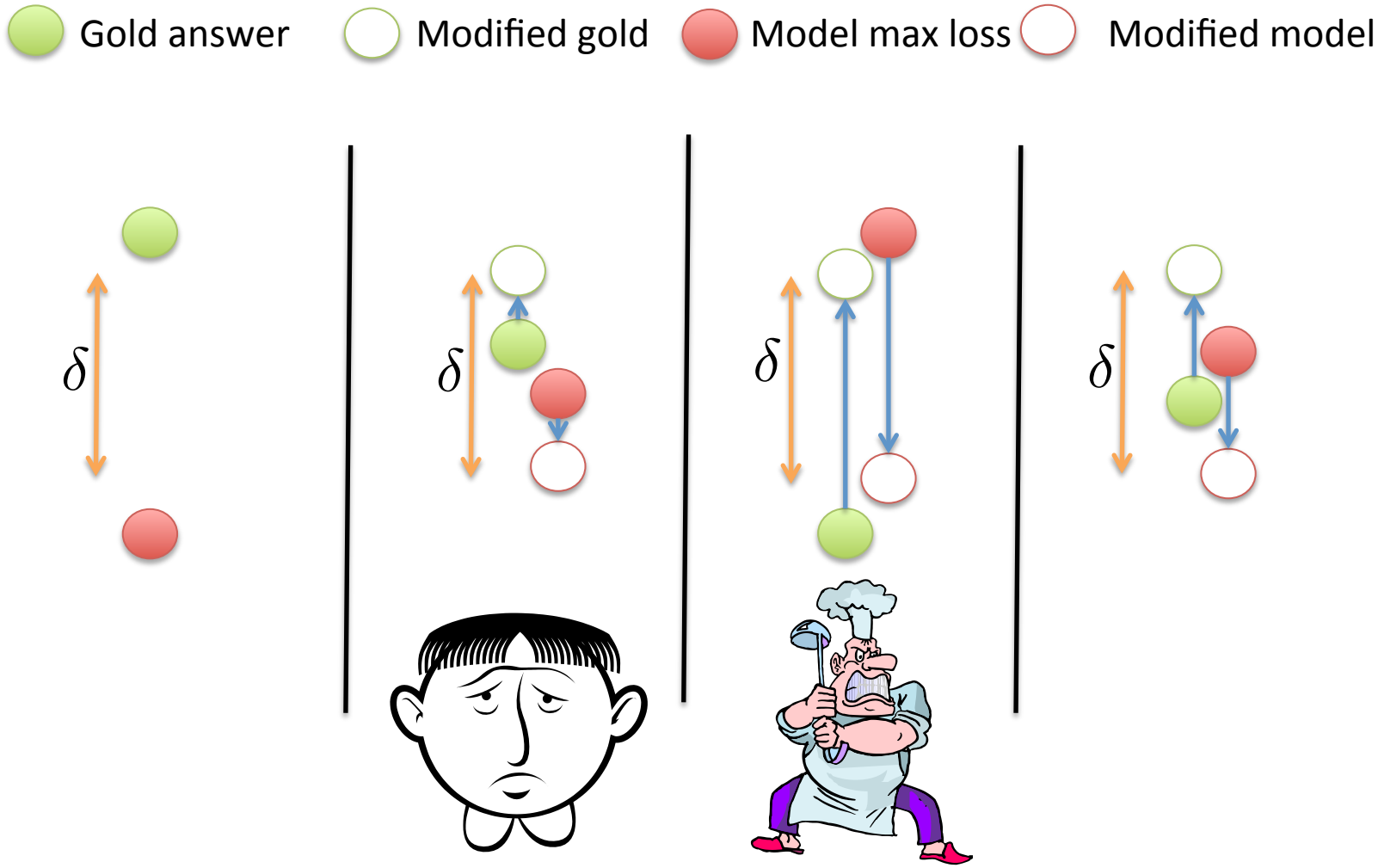
- For each training example (x,y):
  - Decode: $y' = \mathrm{argmax}_{\bar{y} \in Y(x)} \left[\delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \right]$
  - Update: $w = w + \tau \big(\phi(x, y) - \phi(x, y') \big)$

- Do this for many passes through the data, until accuracy on a held-out set degrades

PA, Visually


Gold answer
Modified gold
Model max loss
Modified model
δ
δ
δ
δ

PA, Visually


Gold answer
Modified gold
Model max loss
Modified model
δ
δ
δ
δ

Max model variant

No delta in search, good for difficult delta, like F-measure

- For each training example (x,y):
  - Decode: $y' = \text{argmax}_{\bar{y} \in Y(x)} w \cdot \phi(x, \bar{y})$
  - Update: $w = w + \tau\big(\phi(x, y) - \phi(x, y')\big)$

There's still a delta hidden in here for the update

- Do this for many passes through the data, until accuracy on a held-out set degrades

K-Best Model Variant (MIRA)

- For each training example (x,y):
  - Decode: $Y' = \mathrm{KBest}_{\bar{y} \in Y(x)} \left[w \cdot \phi(x, \bar{y}) \right]$

Still no delta, but uses k-best decoding

  - Update: $w = \mathrm{argmin}_{\bar{w}} \frac{1}{2} \|w - \bar{w}\|^2$

$$\text{s.t. } \forall y' \in Y' : \bar{w} \cdot \phi(x, y) - \bar{w} \cdot \phi(x, y') \geq \delta(y, y')$$

No closed form. Need to solve a one-example SVM

- Do this for many passes through the data, until accuracy on a held-out set degrades

Still an online algorithm

- So you still have to average your parameters to complete the online-to-batch conversion

PA Advantages

- All of the perceptron advantages, plus:
- It converges faster with smarter updates
- Tends to produce smaller weight vectors
  - (The passive part is important – it's good to only make small updates when that's all you need)
- It doesn't skip so many examples
  - Loss means it keeps learning even when it's right

PA Disadvantages

- Still averaging that weight vector
- Still not sure exactly what objective its optimizing in a batch setting

- Otherwise, it's pretty darn good – my tool of choice in an "application pull" scenario

Pegasos

Primal Estimated Subgradient Solver for SVM

- Pegasos introduced by Shalev-Shwartz et al, 2007
- SVM for structured output introduced by
  - Taskar et al, 2003
  - Altun et al, 2003
- Success stories:
  - Clinical document coding (Kiritchenko 2011)

Hey, that's me!

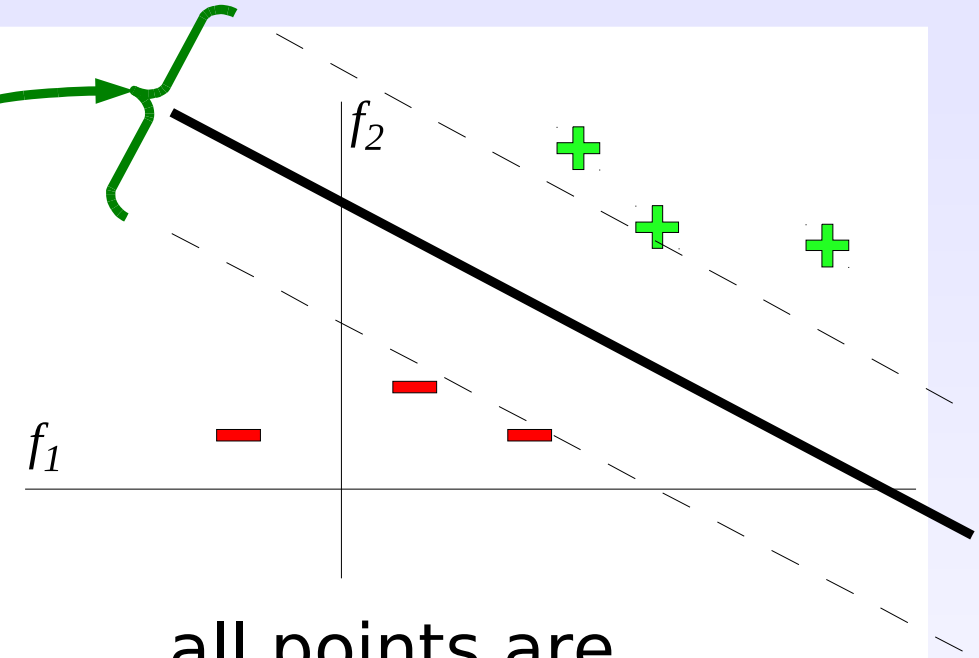
The idea

- Generalize the SVM training objective to structured outputs, then minimize it
  - For efficiency, do so using the Pegasos method

- Theoretically, nothing like the perceptron
- Algorithmically, almost identical

Support vector machines

- Explicitly optimize the ***margin***
- Enforce that all training points are correctly classified



$$\max_{\mathbf{w}} \text{ margin } \quad s.t. \quad \text{all points are correctly classified}$$

$$\max_{\mathbf{w}} \text{ margin } \quad s.t. \quad y_n \mathbf{w} \cdot \phi(x_n) \geq 1 \;, \; \forall n$$

$$\min_{\mathbf{w}} \; \|w\|^2 \quad s.t. \quad y_n \mathbf{w} \cdot \phi(x_n) \geq 1 \;, \; \forall n$$

Structured SVM

- Generalized binary SVM to Structured Setting

Training set

Separate all structures by a fixed margin

$$\min_w \|w\|^2 \text{ s.t.}$$
$$\forall (x,y) \in T :$$
$$\forall y' \in Y(x) :$$
$$w \cdot \phi(x,y) - w \cdot \phi(x,y') \geq \delta(y,y')$$

So, we just have to find the smallest weight vector that gets every training example exactly right, with margin! Great!

Structured SVM with Slacks

Beta trades off generalization (small norm) with training accuracy (low slack)

Slacks allow you to violate the margin, but must be kept small

$$\min_{w,\xi} \frac{\beta}{2}\|w\|^2 + \frac{1}{m}\sum_x \xi_x \text{ s.t.}$$
$$\forall (x,y) \in T :$$
$$\forall y' \in Y(x) :$$
$$w \cdot \phi(x,y) - w \cdot \phi(x,y') + \xi_x \geq \delta(y,y')$$

Size of training set

Important intuition: at the minimum, all >= signs will be = signs (no need to make slacks larger than necessary)

$$\xi_x = \max_{y'} \left[\delta(y,y') + w \cdot \phi(x,y')\right] - w \cdot \phi(x,y)$$

Folding in the constraints

$$\min_w \frac{\beta}{2}\|w\|^2 + \frac{1}{m}\sum_{(x,y)\in T} \ell(w,x,y)$$

where:

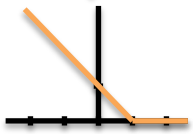
$$\ell(w,x,y) = \max_{y'} \delta(y,y') + w \cdot \phi(x,y') - w \cdot \phi(x,y)$$

Folding in the constraints

$$\min_w \frac{\beta}{2}\|w\|^2 + \frac{1}{m}\sum_{(x,y)\in T} \ell(w,x,y)$$

where:

$$\ell(w,x,y) = \underbrace{\max_{y'} \delta(y,y') + w\cdot\phi(x,y') - w\cdot\phi(x,y)}$$



So, Structured SVM training minimizes a regularized hinge loss over the training set

How do we minimize it?

- Objective is convex
- Apply stochastic sub-gradient descent
- General idea:
  - Visit an example at random (**stochastic**)
  - Find the **gradient** of the (approximate) objective for that example, take a step in that direction
  - Repeat
- Since the objective contains a max, we will need to take a **sub-gradient** (gradient at the max)

Taking the sub-gradient at x

$$\text{Let } y' \quad = \quad \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) - w \cdot \phi(x, y)$$

Taking the sub-gradient at x

$$\begin{aligned}\text{Let } y' &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) - w \cdot \phi(x, y) \\ &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y})\end{aligned}$$

Taking the sub-gradient at x

$$\begin{aligned} \text{Let } y' &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) - w \cdot \phi(x, y) \\ &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \end{aligned}$$

$$f(w) = \frac{\beta}{2} \|w\|^2 + \delta(y, y') + w \cdot \phi(x, y') - w \cdot \phi(x, y)$$

Taking the sub-gradient at x

$$\begin{aligned} \text{Let } y' &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) - w \cdot \phi(x, y) \\ &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \end{aligned}$$

$$f(w) = \frac{\beta}{2} \|w\|^2 + \delta(y, y') + w \cdot \phi(x, y') - w \cdot \phi(x, y)$$

$$f'(w) = \beta w + \phi(x, y') - \phi(x, y)$$

Taking the sub-gradient at x

$$\begin{aligned} \text{Let } y' &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) - w \cdot \phi(x, y) \\ &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \end{aligned}$$

$$f(w) = \frac{\beta}{2} \|w\|^2 + \delta(y, y') + w \cdot \phi(x, y') - w \cdot \phi(x, y)$$

$$f'(w) = \beta w + \phi(x, y') - \phi(x, y)$$

To update w, subtract gradient f' with learning rate η

$$w = w - \eta f'(w)$$

Taking the sub-gradient at x

$$\begin{aligned} \text{Let } y' &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) - w \cdot \phi(x, y) \\ &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \end{aligned}$$

$$f(w) = \frac{\beta}{2} \|w\|^2 + \delta(y, y') + w \cdot \phi(x, y') - w \cdot \phi(x, y)$$

$$f'(w) = \beta w + \phi(x, y') - \phi(x, y)$$

To update w, subtract gradient f' with learning rate η

$$\begin{aligned} w &= w - \eta f'(w) \\ &= w - \eta\big(\beta w + \phi(x, y') - \phi(x, y)\big) \end{aligned}$$

Taking the sub-gradient at x

$$\begin{aligned} \text{Let } y' &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) - w \cdot \phi(x, y) \\ &= \text{argmax}_{\bar{y}} \delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \end{aligned}$$

$$f(w) = \frac{\beta}{2}\|w\|^2 + \delta(y, y') + w \cdot \phi(x, y') - w \cdot \phi(x, y)$$

$$f'(w) = \beta w + \phi(x, y') - \phi(x, y)$$

To update w, subtract gradient f' with learning rate η

$$\begin{aligned} w &= w - \eta f'(w) \\ &= w - \eta\big(\beta w + \phi(x, y') - \phi(x, y)\big) \\ &= w + \eta\big(\underbrace{\phi(x, y) - \phi(x, y')} - \beta w\big) \end{aligned}$$

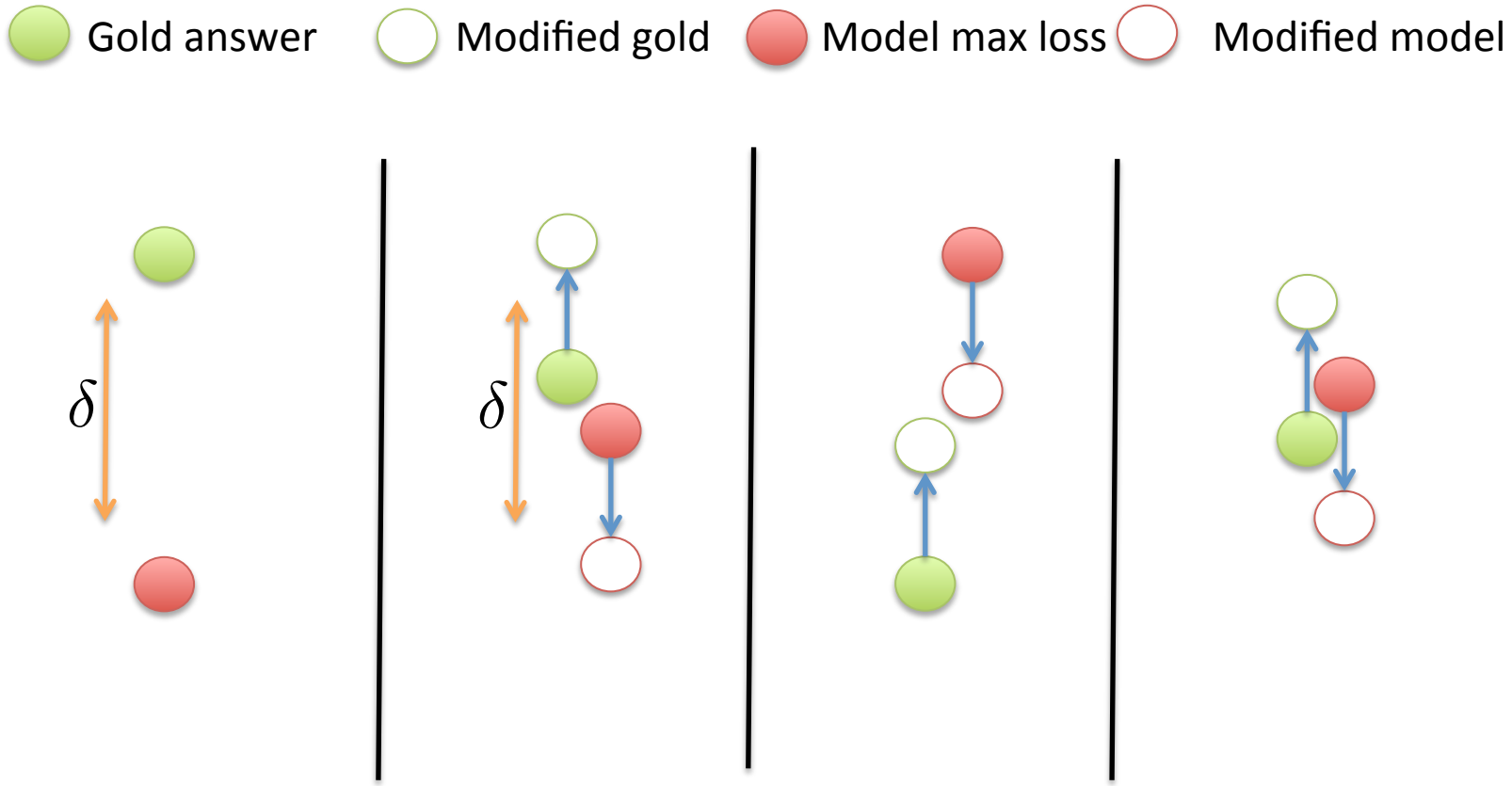
The algorithm

- On step t, sample a training example (x,y):
  - Decode: $y' = \mathrm{argmax}_{\bar{y} \in Y(x)} \left[\delta(y, \bar{y}) + w \cdot \phi(x, \bar{y}) \right]$
  - Update: $\eta = \frac{1}{\beta t}$

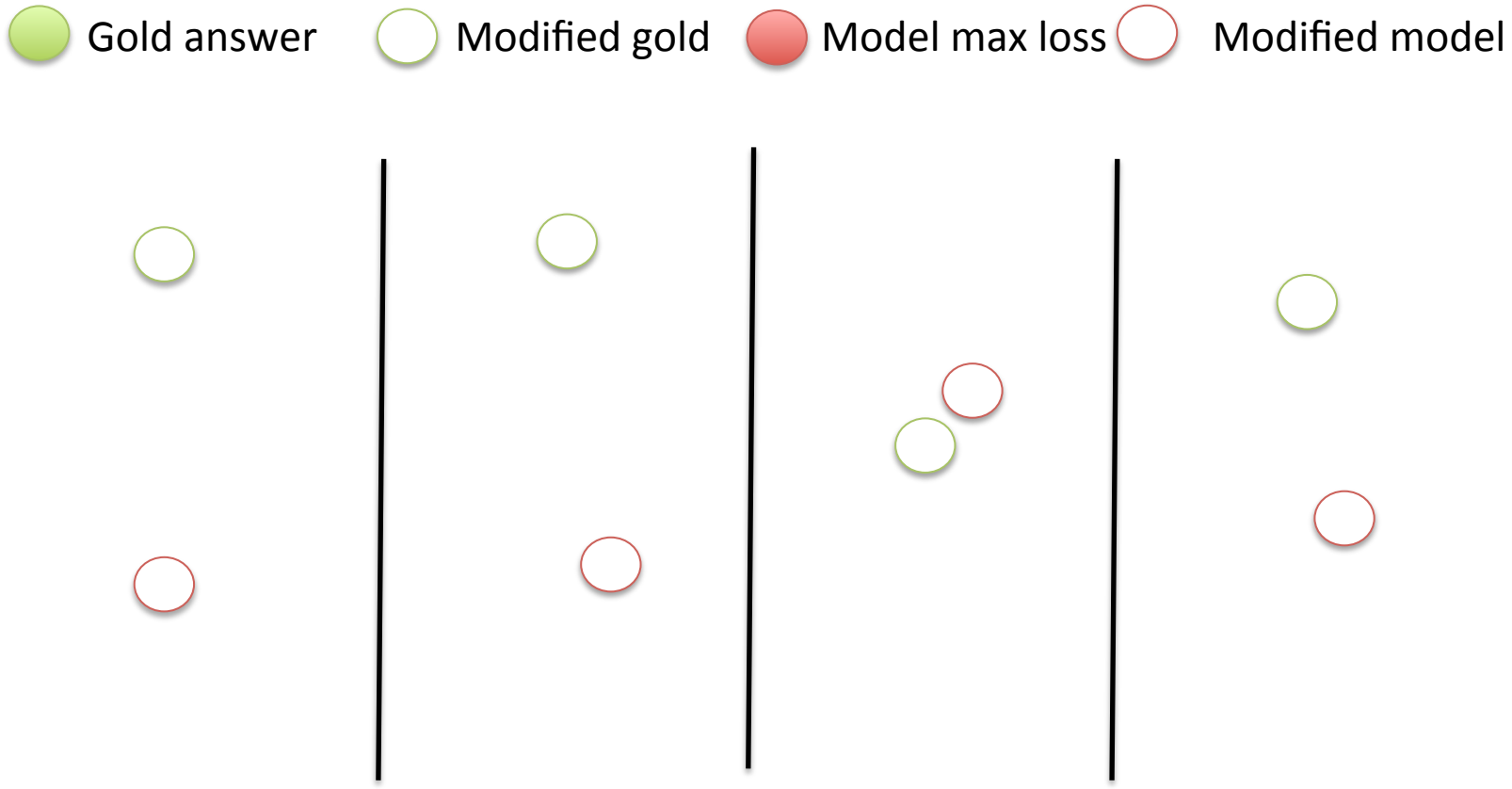
$$w = w + \eta \big(\phi(x, y) - \phi(x, y') - \beta w \big)$$

- Repeat until the SVM objective converges
- Smarter men than I have proofs that show that this learning rate is a good one

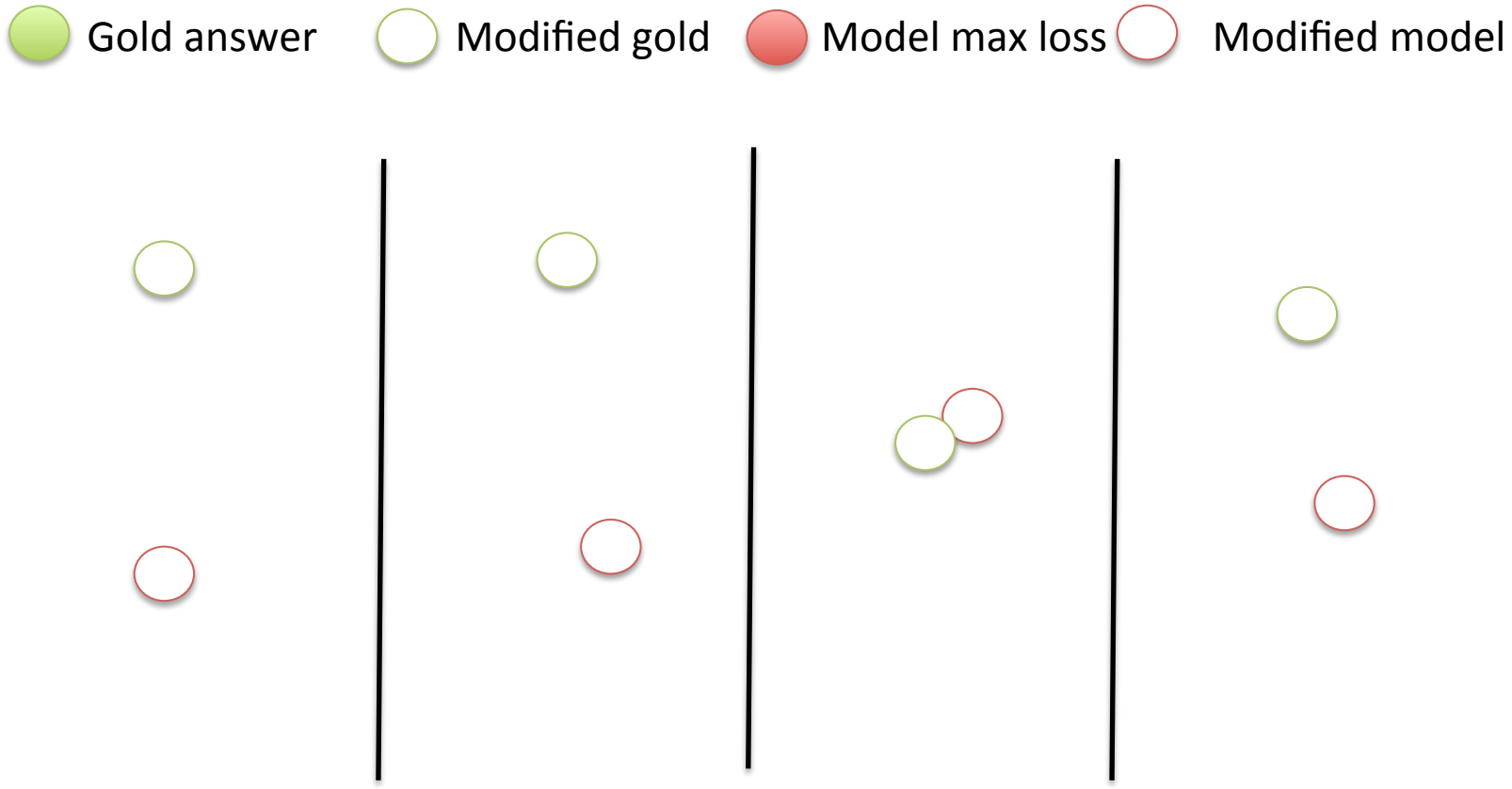
Pegasos, Visually



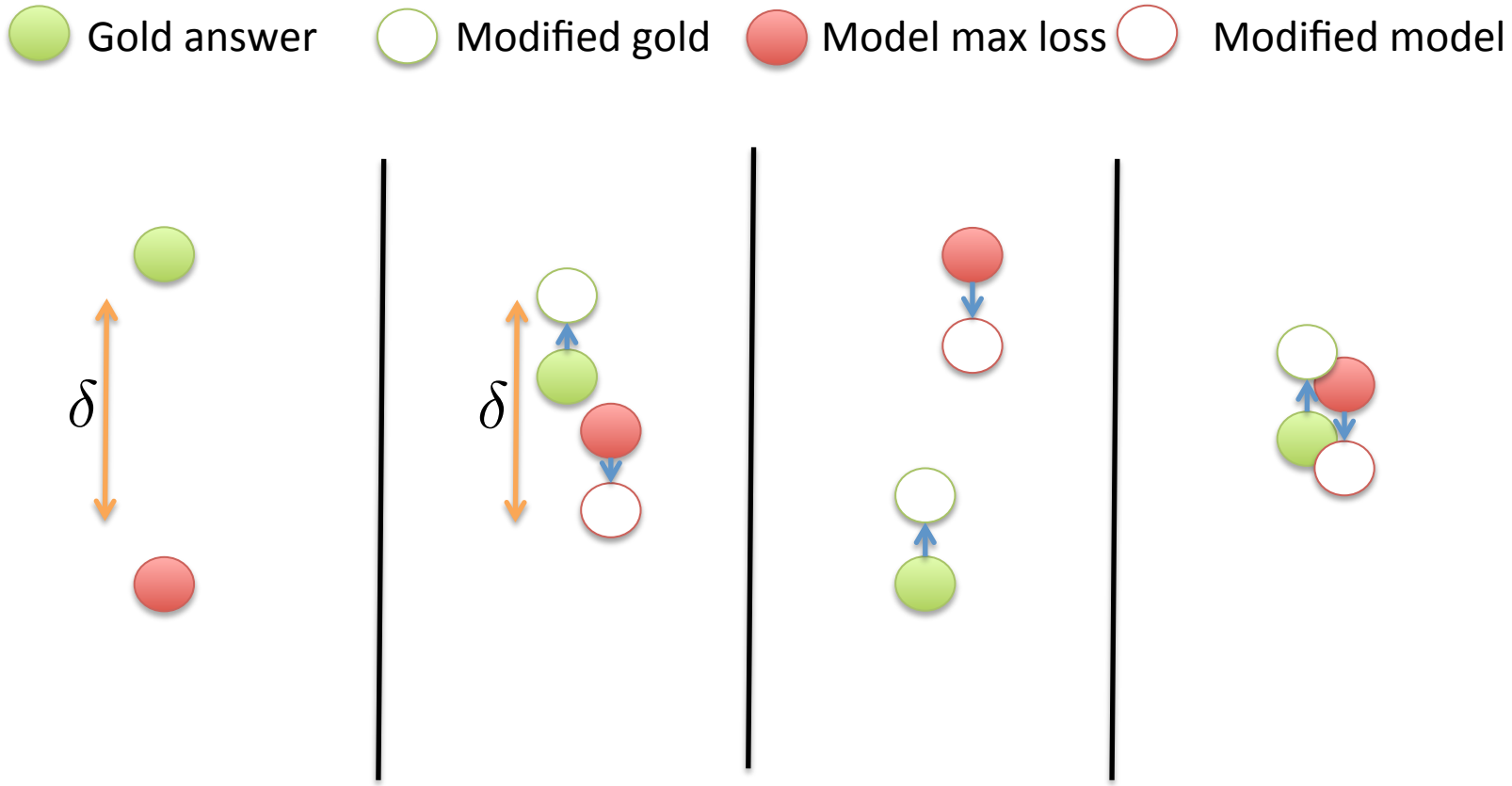
Pegasos, Visually



Pegasos, Visually



Pegasos, as learning rate decays



Implementation concerns

$$w = w + \eta\big(\phi(x, y) - \phi(x, y') - \beta w\big)$$

Subtracting the βw term can be expensive for larger feature sets

$$w = (1 - \eta\beta)w + \eta\big(\phi(x, y) - \phi(x, y')\big)$$

Scaling by $(1 - \eta\beta)$ can be done efficiently with a clever representation

Pegasos Advantages

- Has a real batch SVM training objective!
- No need to average the weight vector
  - The weight vector used in training is the weight vector used at test time
- Has an explicit regularization term – you can directly control generalization through beta
- People take SVMs seriously

Pegasos Disadvantages

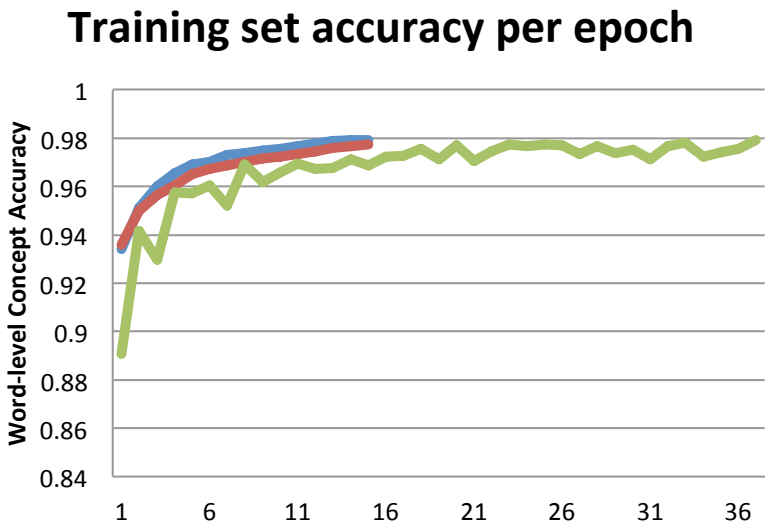
- Can take more iterations to converge than PA or perceptron
- Has a hyper-parameter to tune (beta)
- Need to do max-loss search
  - Perceptron only does max-model
  - PA is kind of ambivalent
- Otherwise, very good!
  - My tool of choice for “technology push” projects

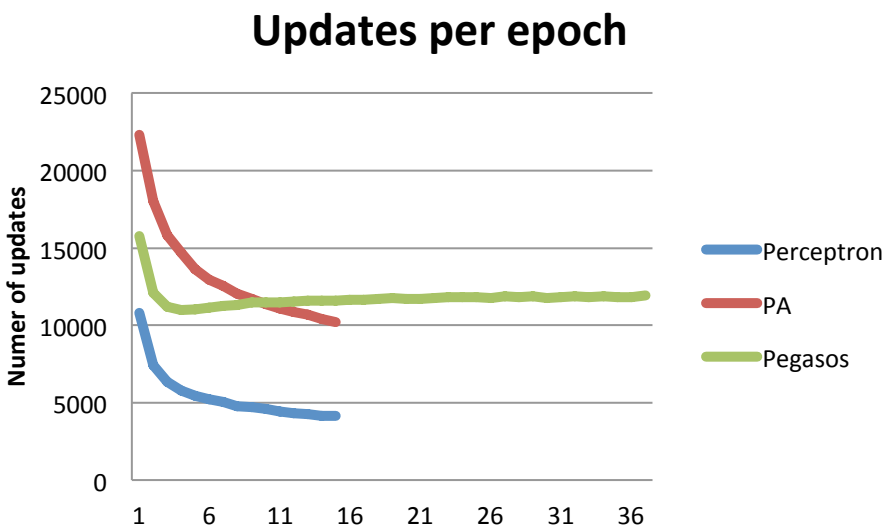
Comparison: I2B2 Concept Detection

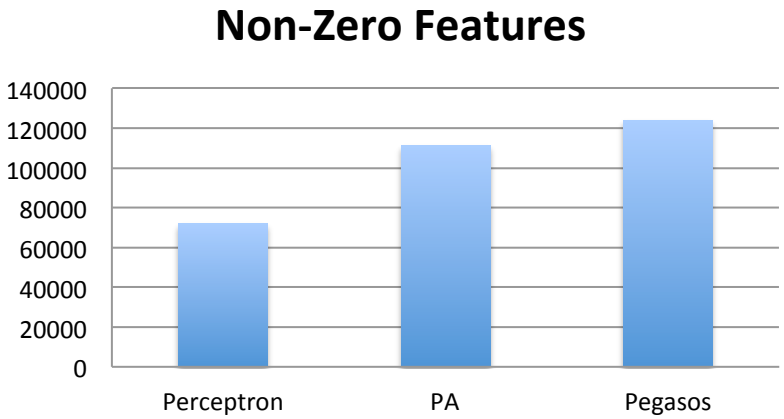
- Semi-Markov Model labels spans of text with clinical concepts

- Features: Bag of words in the concept
  - Toy feature set
- Used simple 0-1 delta, no fancy costs
  - 2-best decode, if 1-best is right, 2-best is max-loss

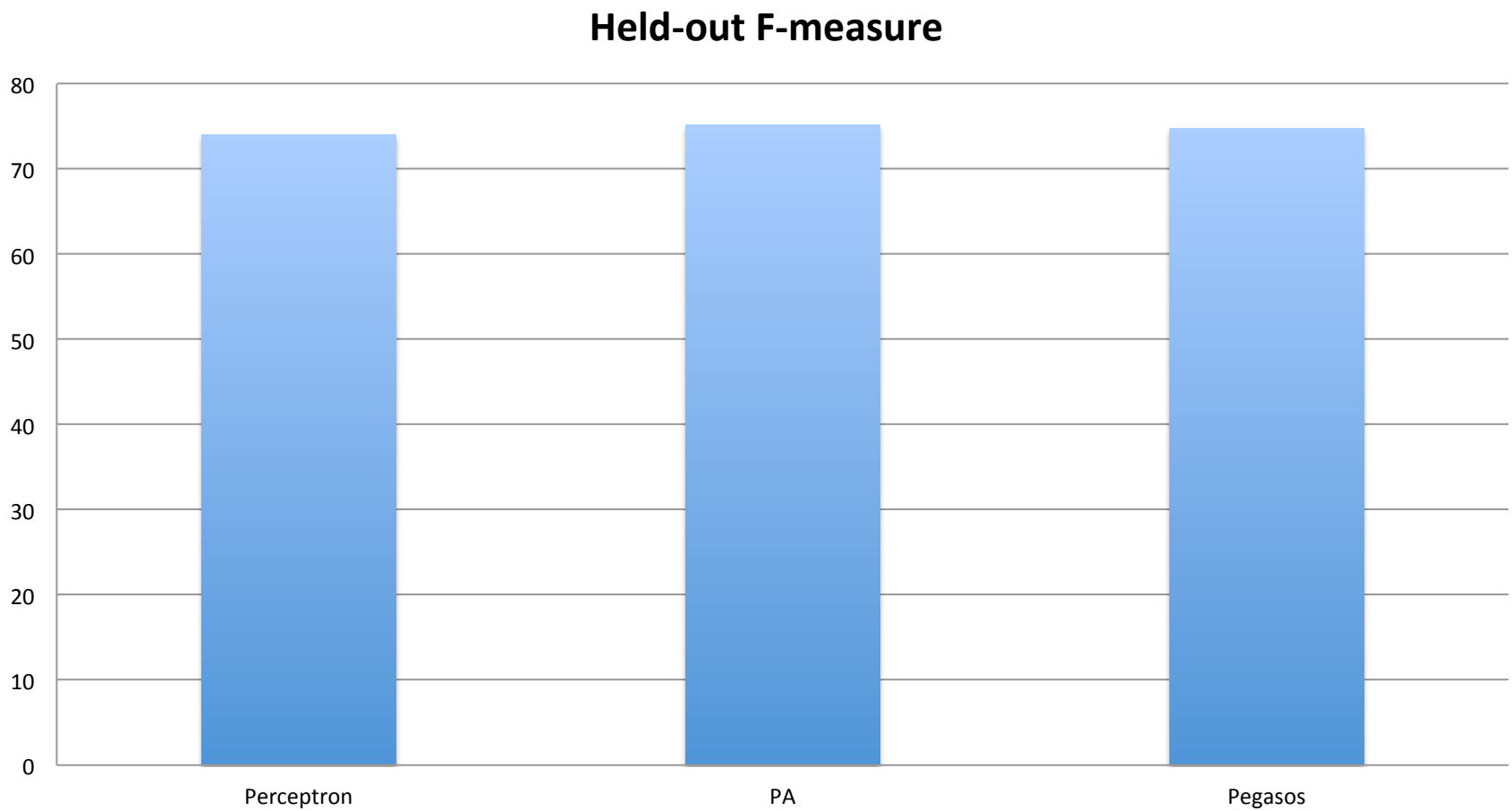
Training set performance



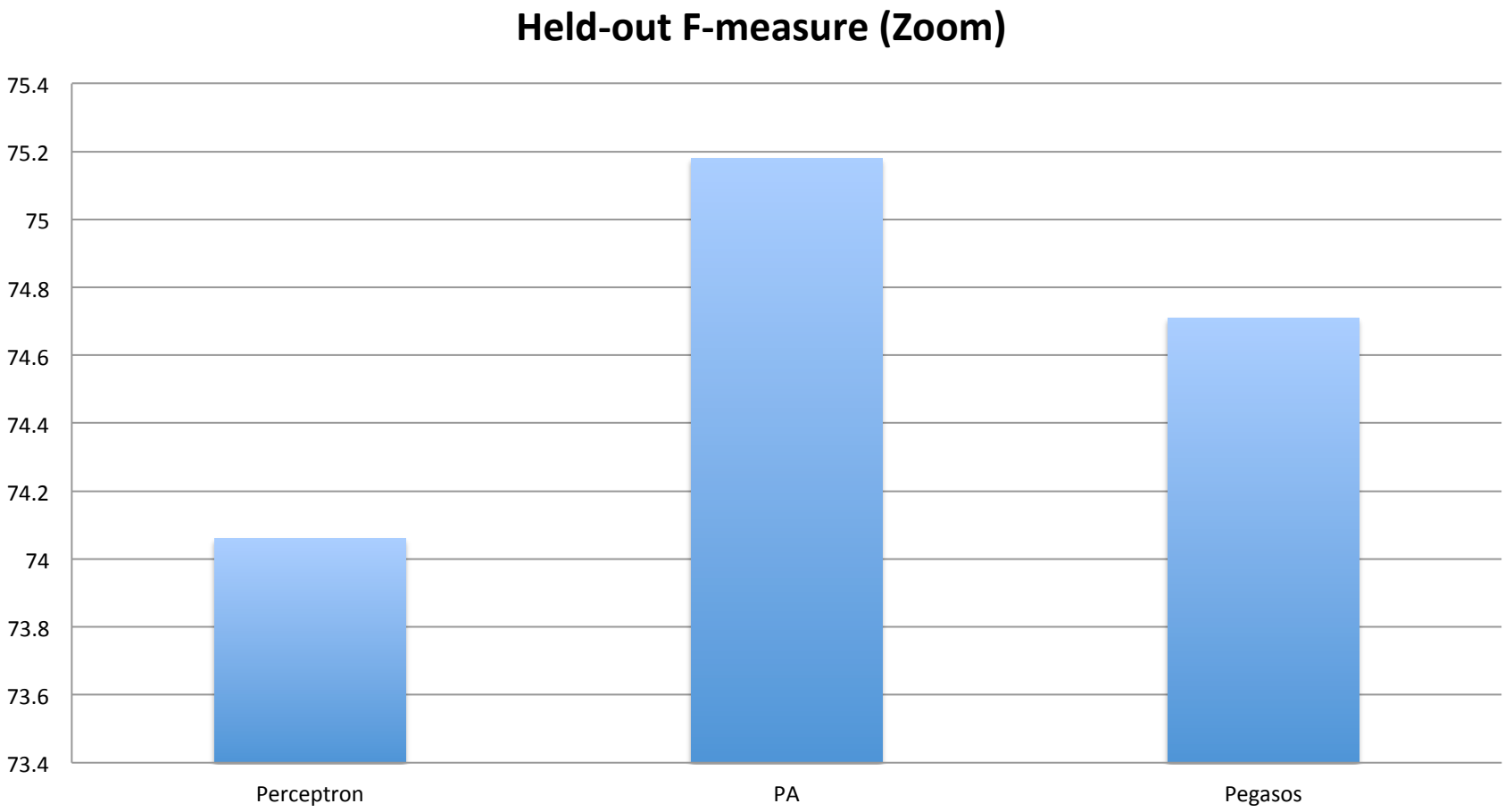




Test Set Performance



Test Set Performance



What happened in actuality

- Developed entire system with perceptron
- Implemented PA in the last week
  - Saw a 0.5-point jump in cross-validated F-measure
  - Won the competition (by more than 0.5)
- Built Pegasos later for some ML work
- Plugged it into I2B2 for this talk
  - (took 10 minutes plug in, an hour to tune beta)

Conclusion

- Presented three methods for online structure prediction
  - All can be motivated as approximations to stochastic gradient SVM training
- Online means one example at a time, which means easy on memory, feature size, coding
  - (Hard on parallelization)
- Allows a user-defined cost δ