TTIC 31190: Natural Language Processing

Kevin Gimpel
Spring 2018

Lecture 6: Learning for Classification; Language Modeling

- assignment 1 due today
- questions?
- if you want to use late day(s), state that on your report

- assignment 2 will be posted tomorrow

- start thinking about your project, who you might want to work with, etc.

- short quiz at start of class Wed., April 18th
- covering material up to and including Mon., April 9th
- don't stress about it
- grading will be check-minus/check/check-plus

Roadmap

- words, morphology, lexical semantics
- text classification
- simple neural methods for NLP
- language modeling and word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks

Text Classification

- datasets
- classification
  - modeling
  - inference
  - learning

Classifiers

- one simple type:
  – for any input ***x***, assign a score to each label *y*

$$\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

  – classify by choosing highest-scoring label:

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\text{argmax}} \; \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

Linear Models

- parameters are arranged in a vector $\mathbf{w}$
- score function is linear in $\mathbf{w}$:

$$\text{score}(\boldsymbol{x}, y, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y) = \sum_i w_i f_i(\boldsymbol{x}, y)$$

- $\mathbf{f}$: vector of feature functions

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \operatorname*{argmax}_{y} \ \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

learning: choose $\boldsymbol{w}$

- **Learning**: How do we choose the weights ***w***?

Regularized Empirical Risk Minimization

surrogate loss

regularization strength

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

regularization term

Regularized Empirical Risk Minimization

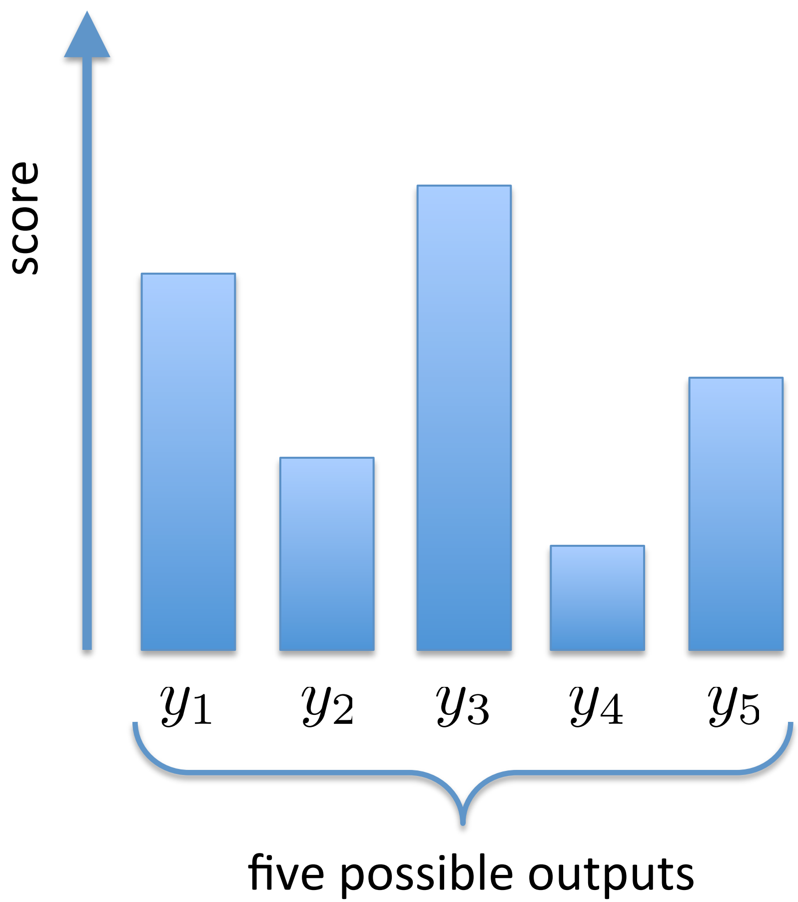
$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\text{argmin}} \sum_{i=1}^{|\mathcal{T}|} \text{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

encourages model to fit the training data well

encourages model to be "simpler" in the hope that this will help it to generalize to new data

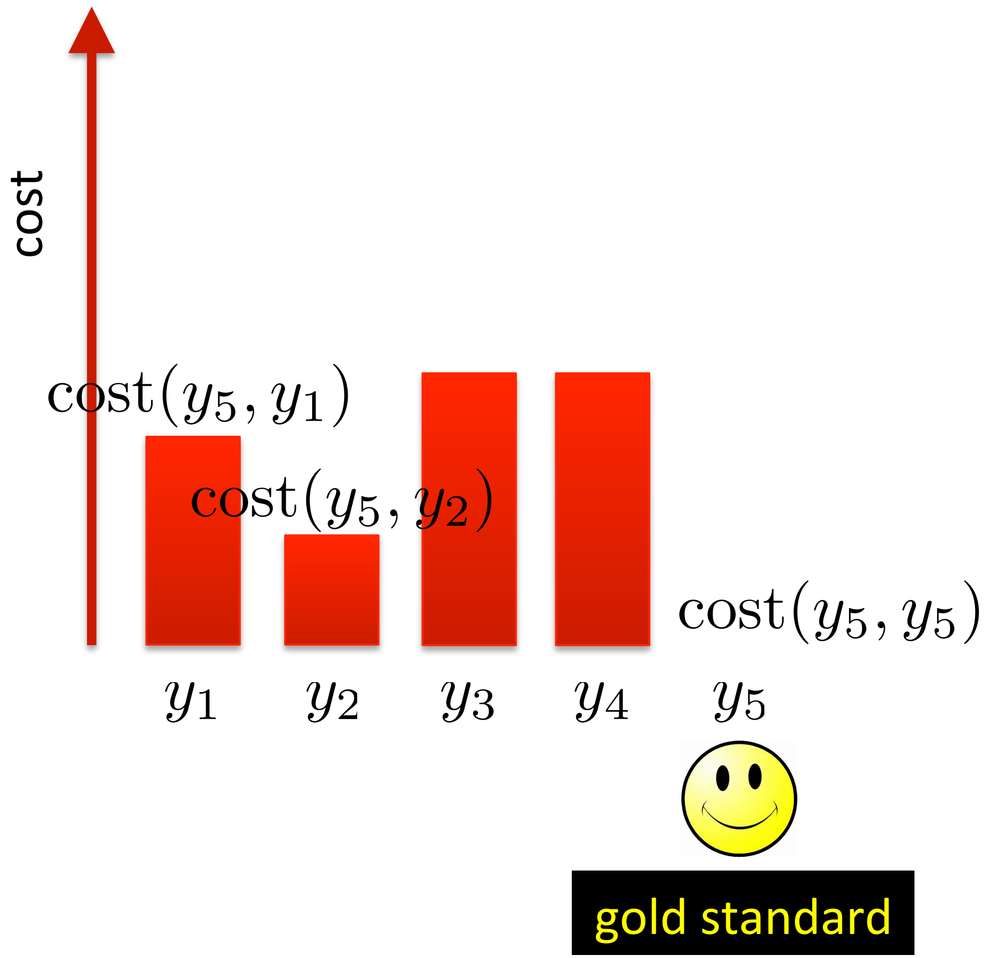
Visualization

for a single input ***x***



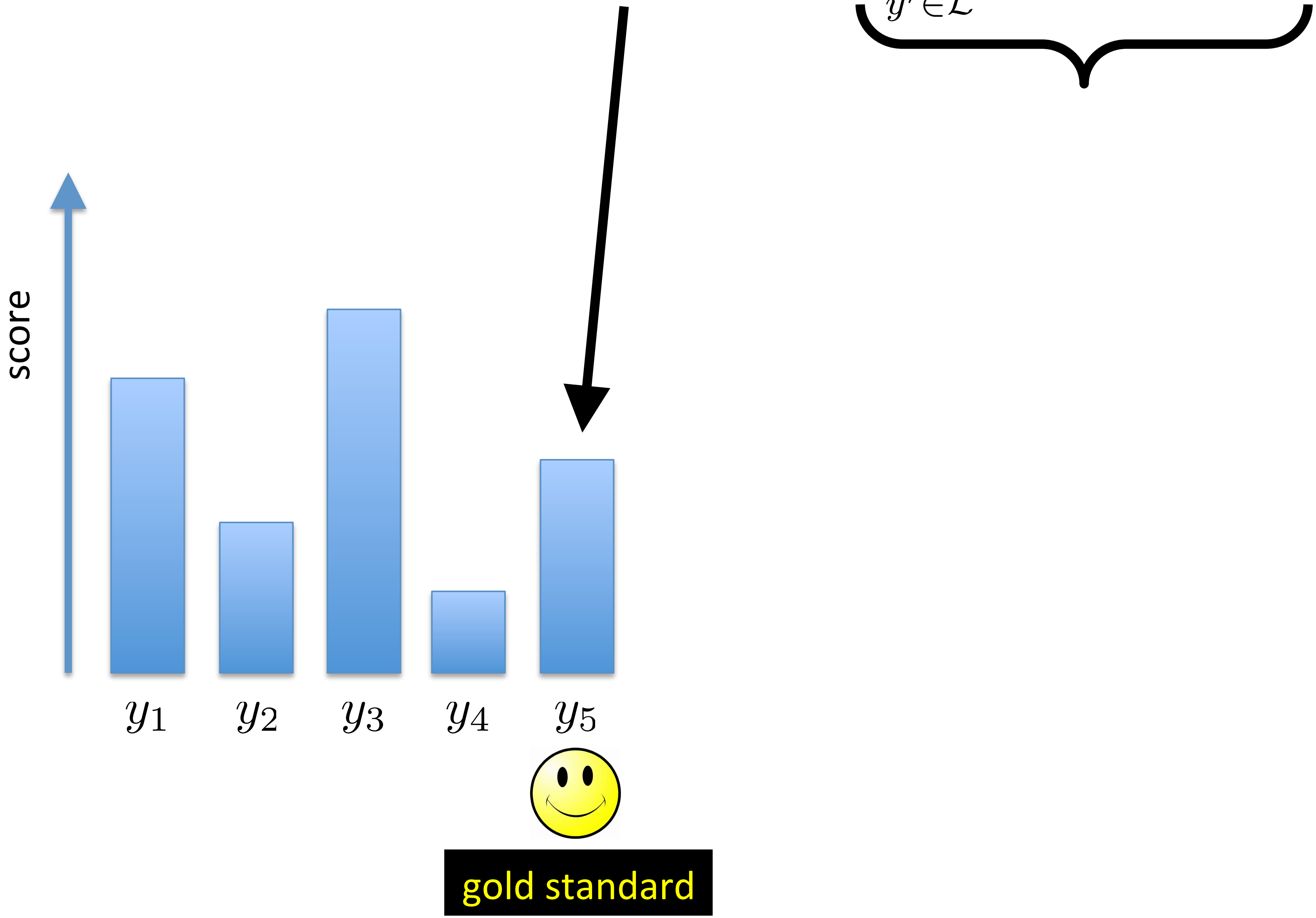
Visualization

for a single input ***x***



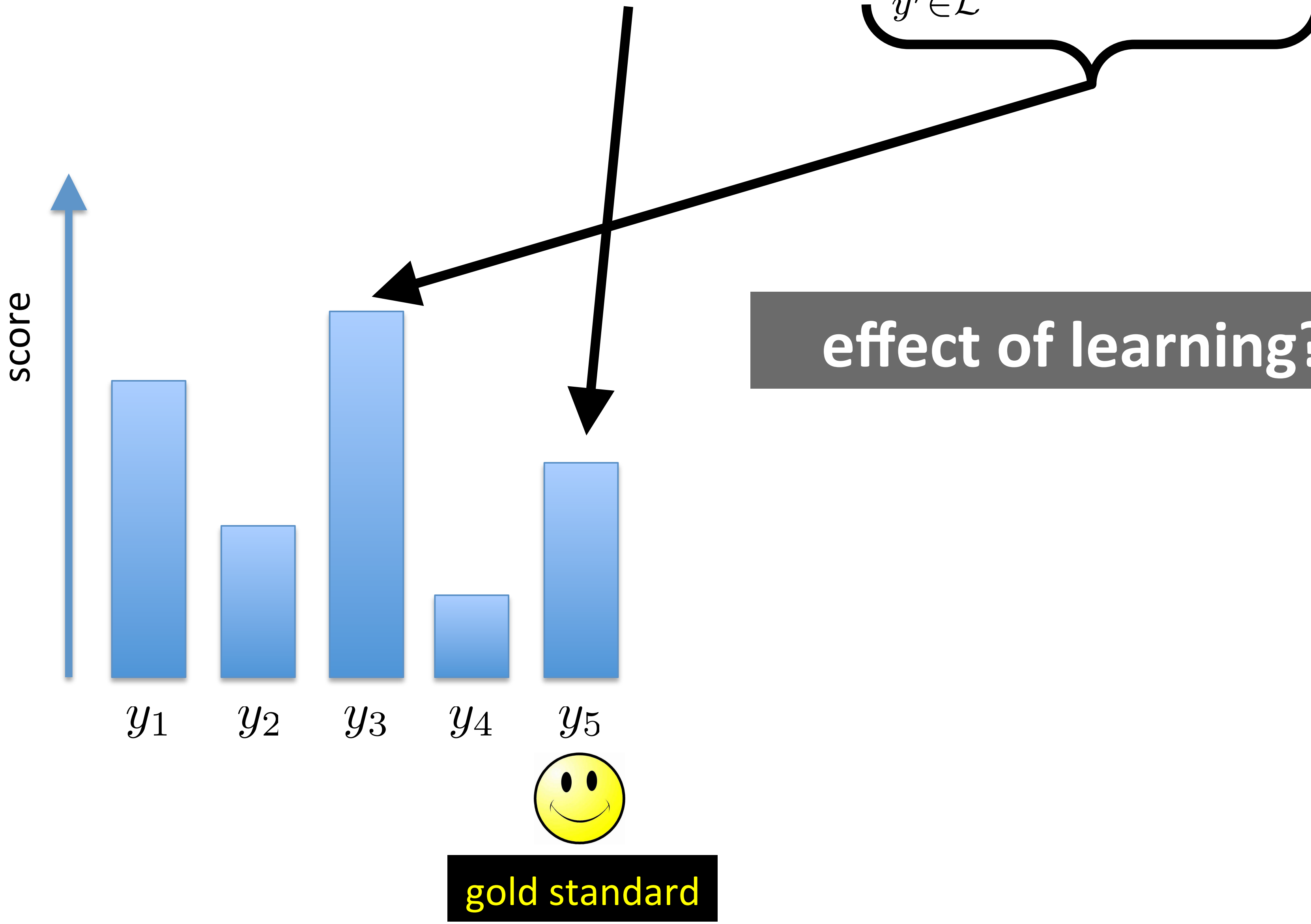
perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$



perceptron loss:

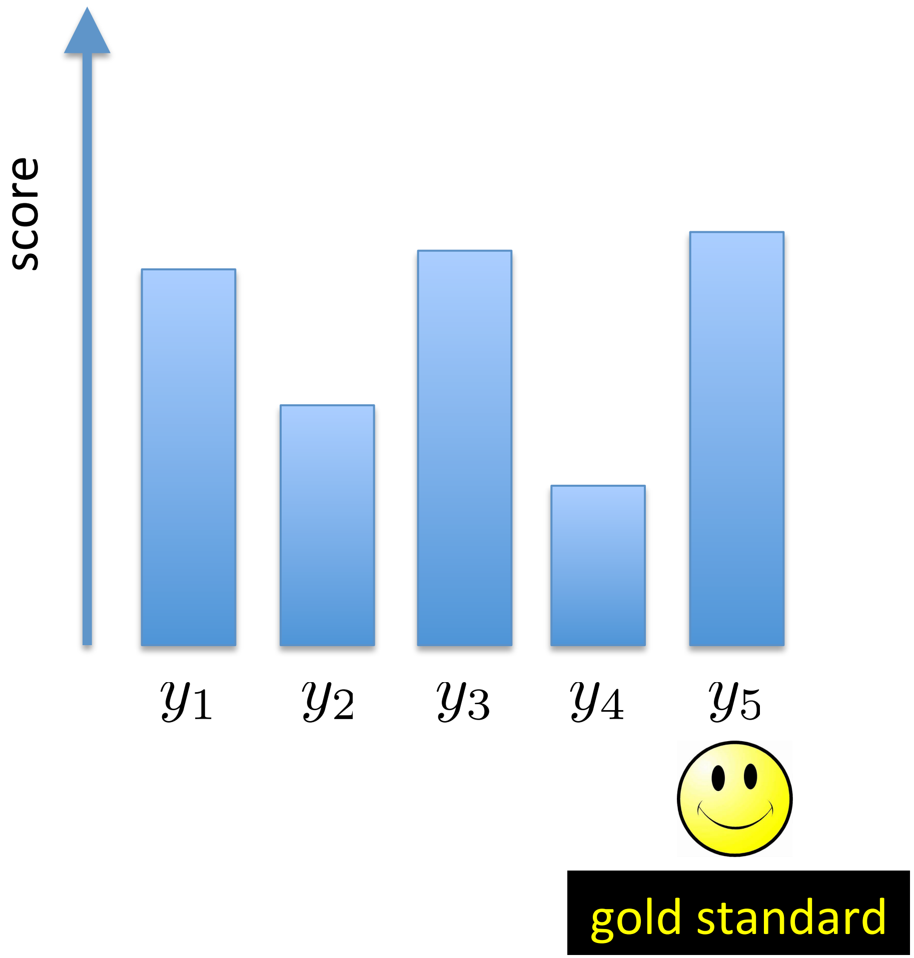
$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$



effect of learning?

perceptron loss:

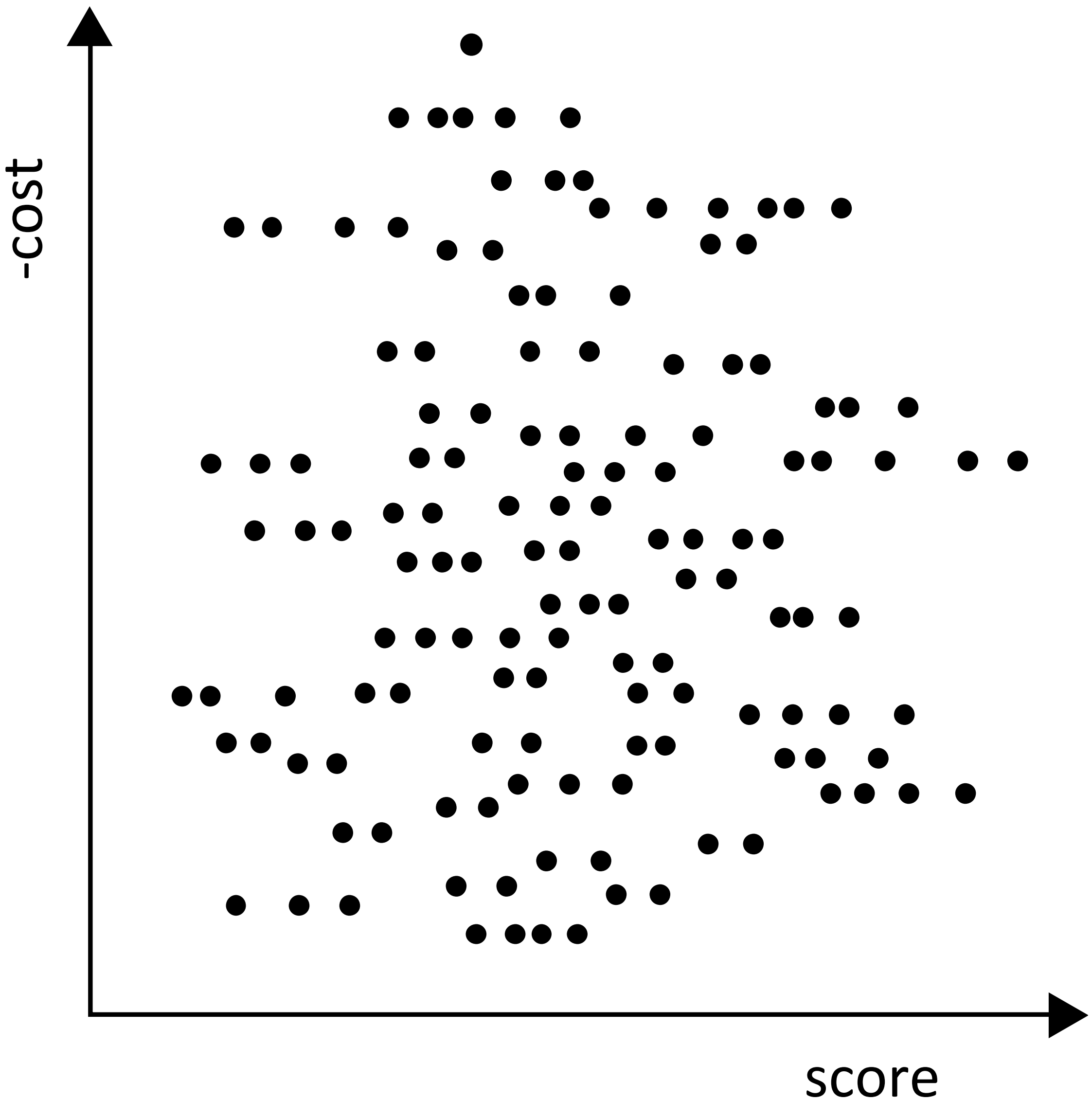
$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$



effect of learning:
gold standard will have highest score

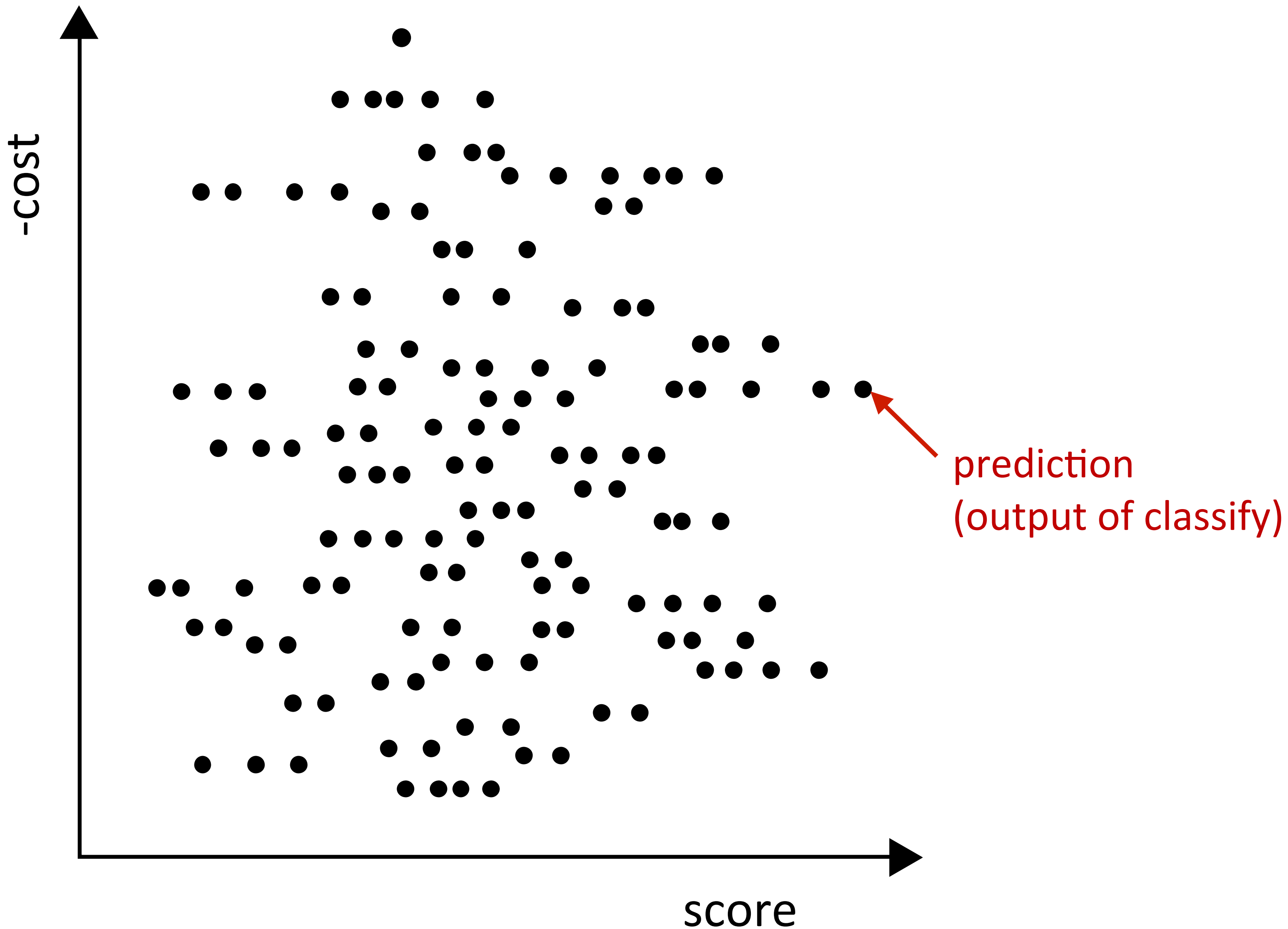
Visualization for a single ***x***:

each point is a possible *y*



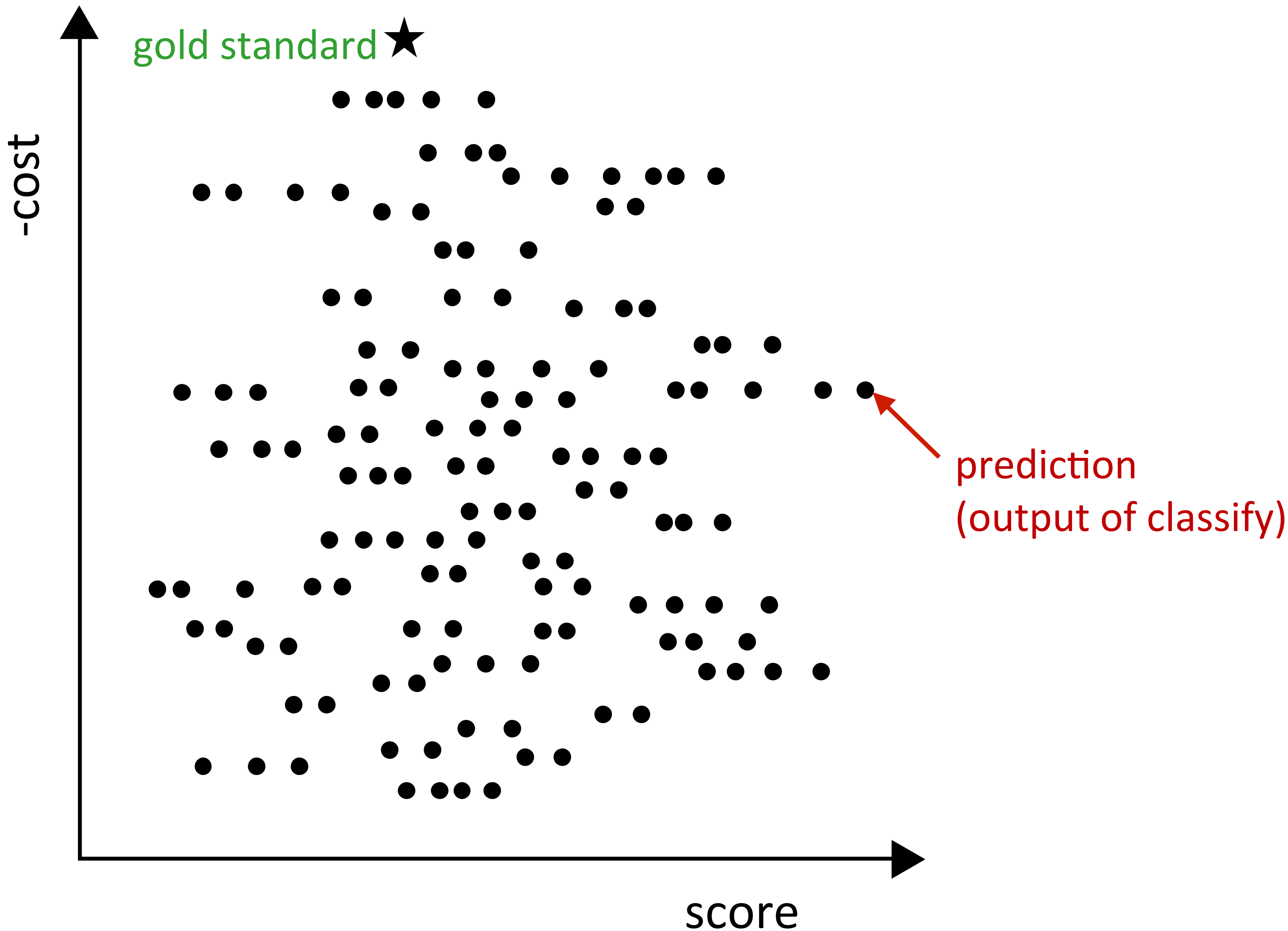
Visualization for a single ***x***:

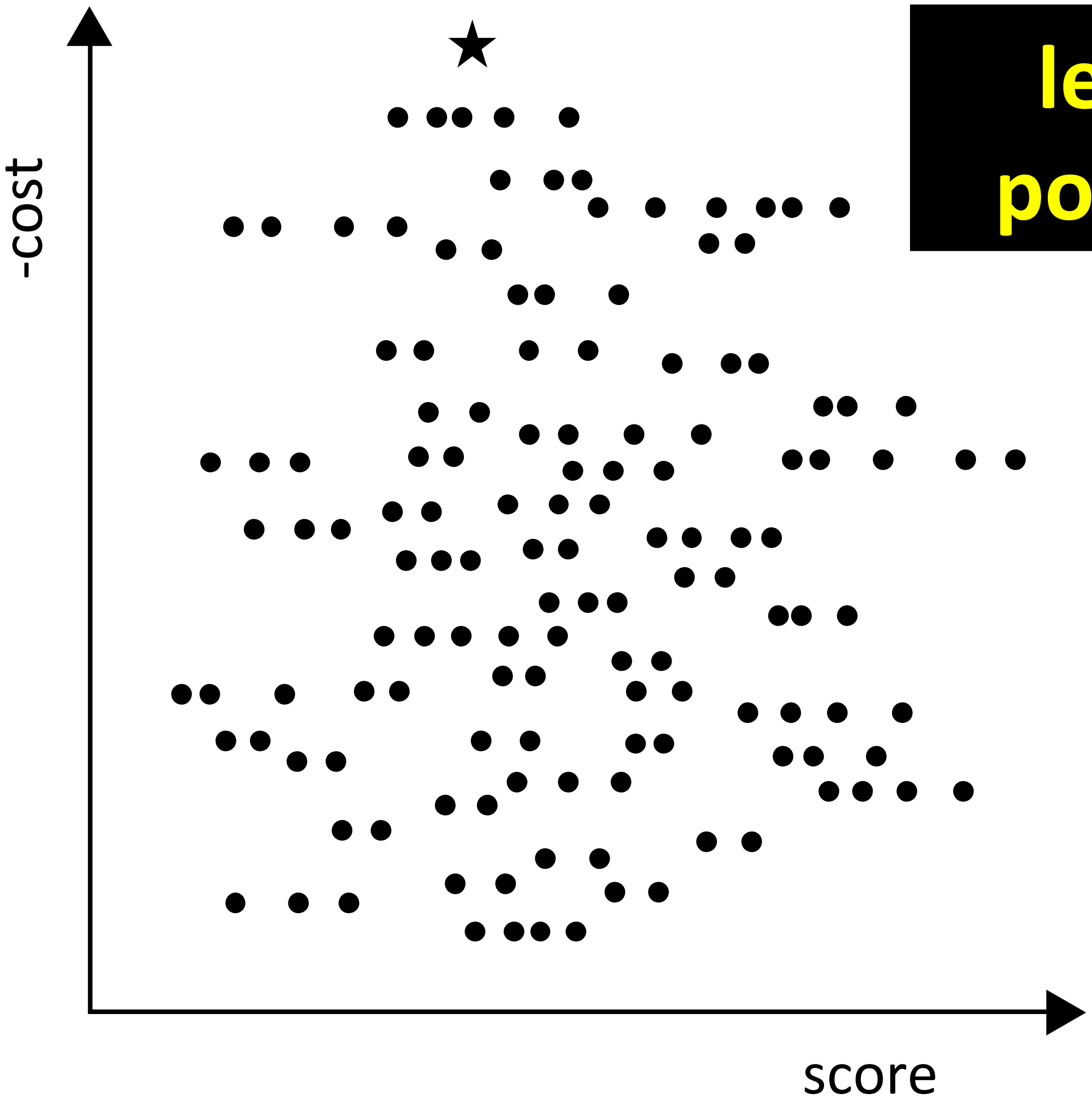
each point is a possible *y*



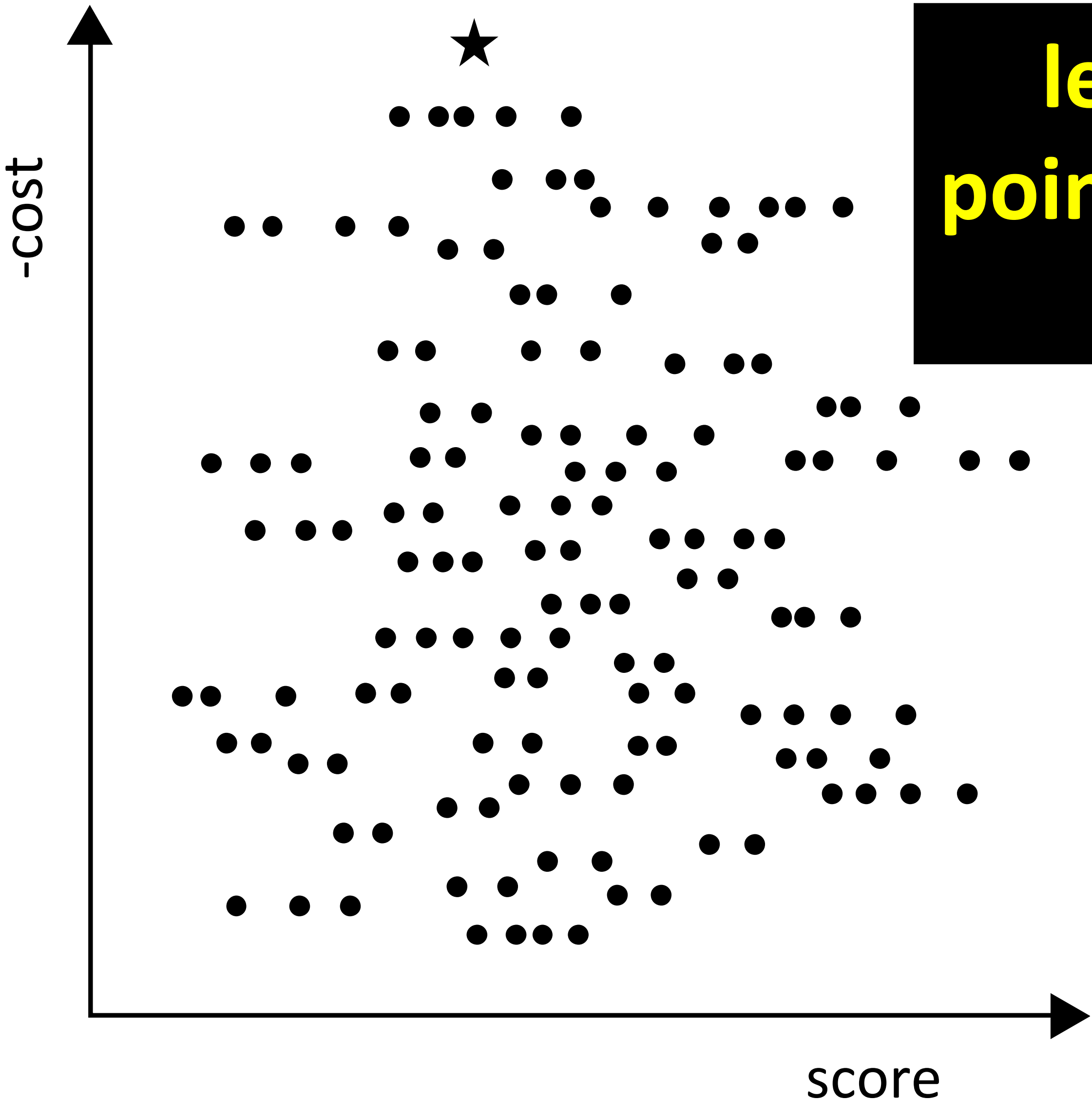
Visualization for a single ***x***:

each point is a possible *y*

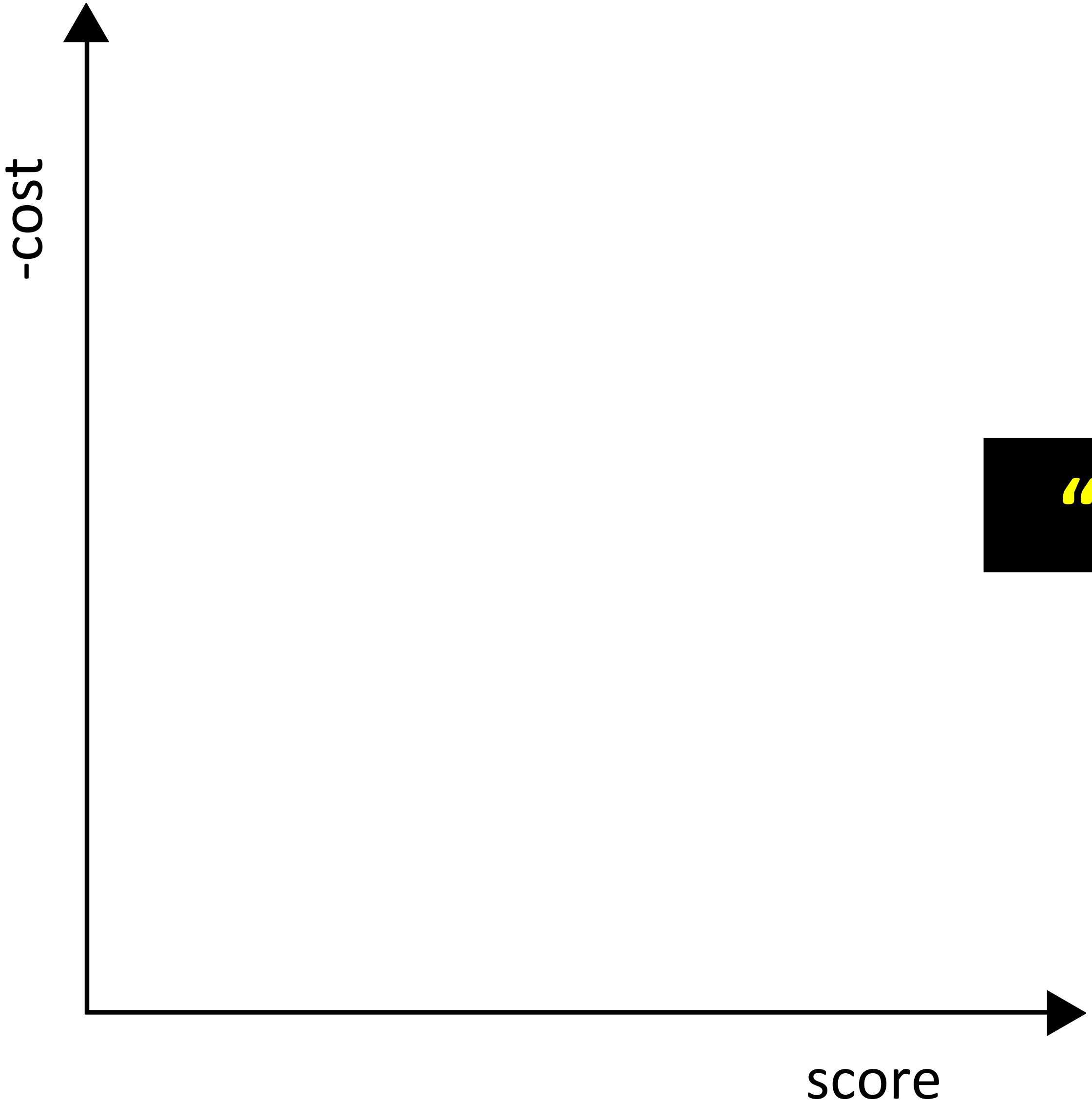




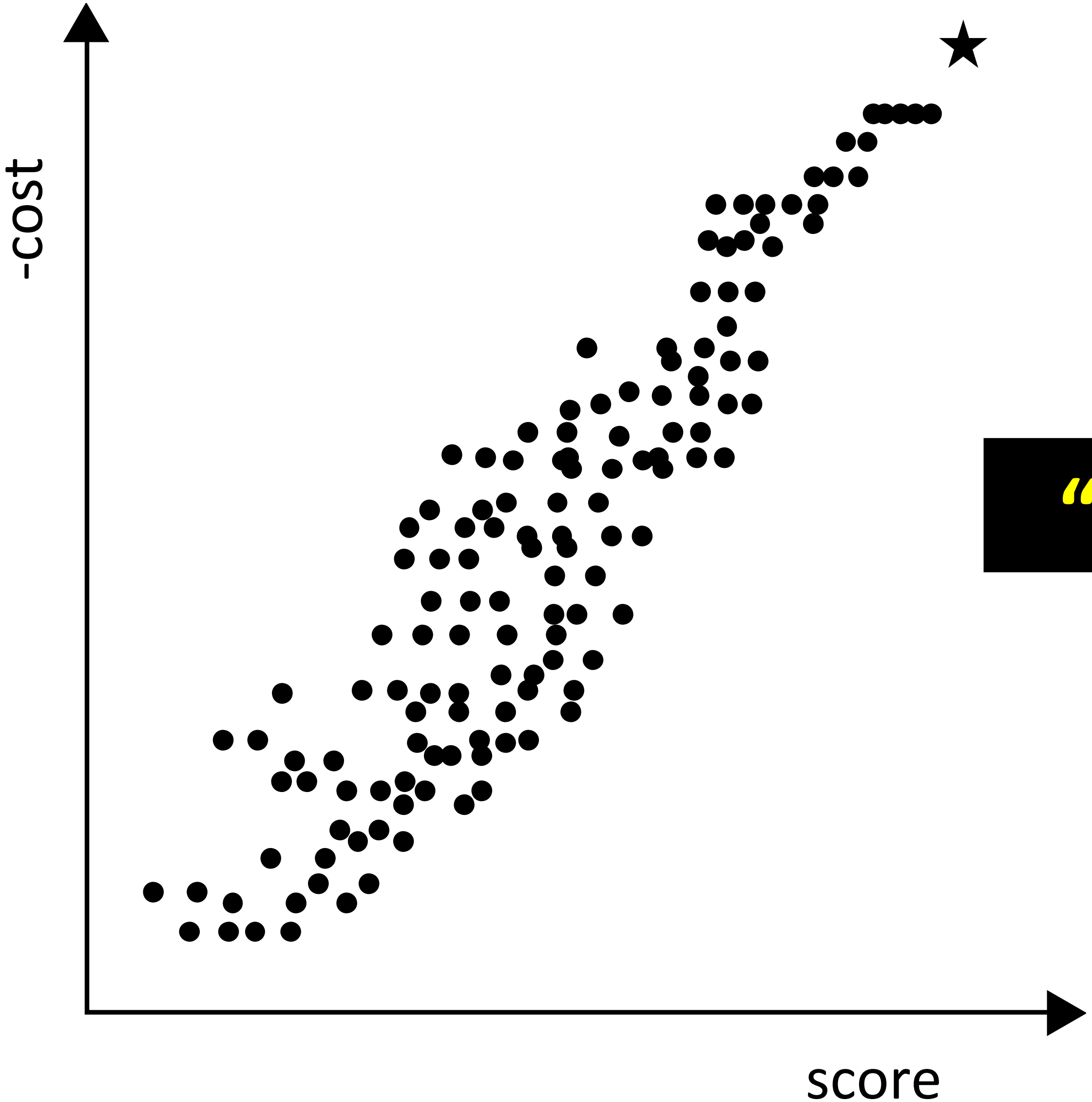
learning moves points in this plot



learning moves points left or right in this plot

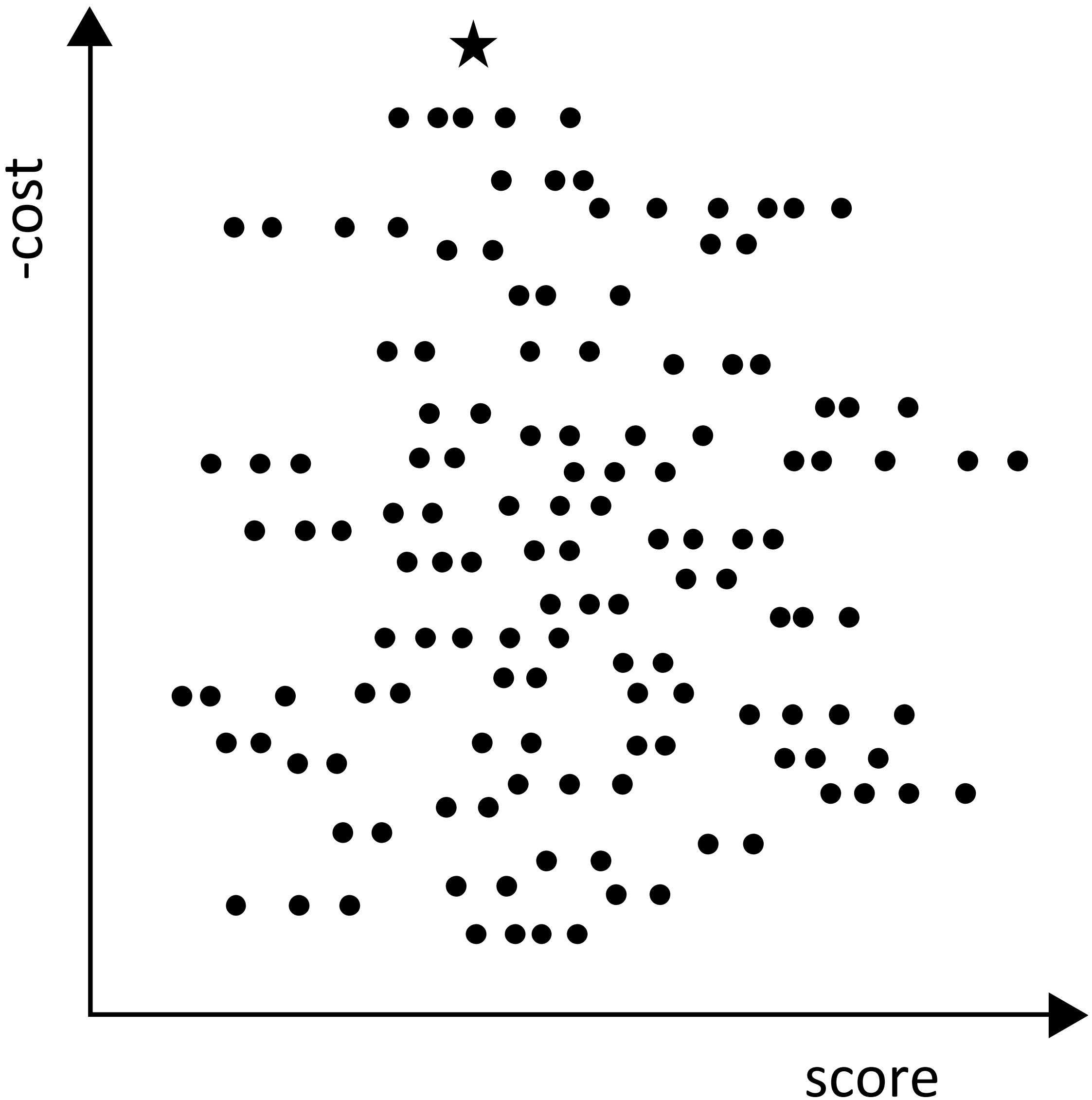


“ideal” model?

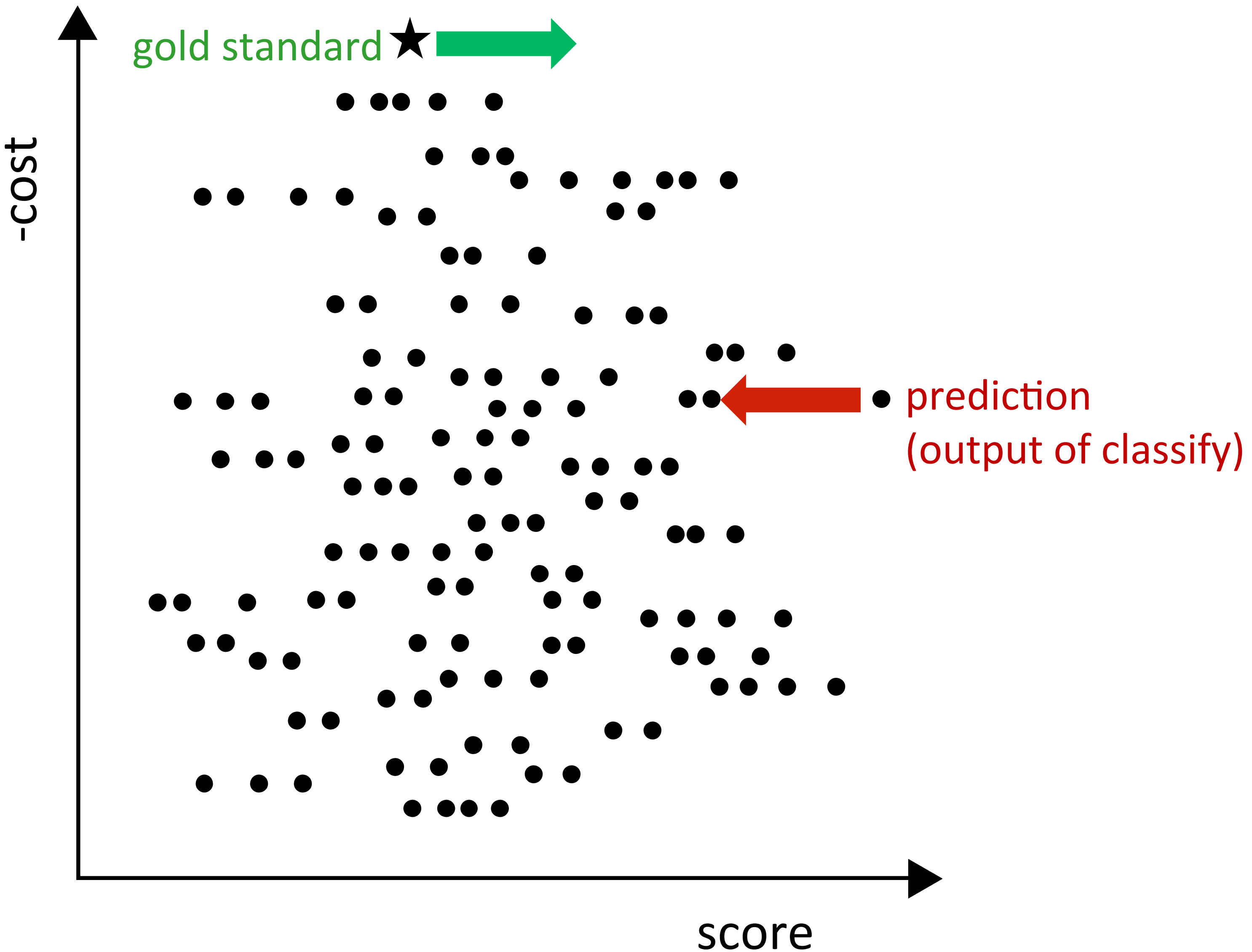


“ideal” model?

Perceptron Loss?



Perceptron Loss



Losses for Linear Models

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

$$\downarrow$$

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y')$$

Losses for Linear Models

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

$$\downarrow$$

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y')$$

$$= -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

Loss Subgradients for Linear Models

- some of our loss functions are not differentiable:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

- but they are subdifferentiable:

$$\frac{\partial \text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w})}{\partial w_j} =$$

Loss Subgradients for Linear Models

- some of our loss functions are not differentiable:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

- but they are subdifferentiable:

$$\frac{\partial \text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w})}{\partial w_j} = -f_j(\boldsymbol{x}, y) +$$

Loss Subgradients for Linear Models

- some of our loss functions are not differentiable:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

- but they are subdifferentiable:

$$\frac{\partial \text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w})}{\partial w_j} = -f_j(\boldsymbol{x}, y) + \frac{\partial}{\partial w_j} \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

Loss Subgradients for Linear Models

- some of our loss functions are not differentiable:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

- but they are subdifferentiable:

$$\frac{\partial}{\partial w_j} \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y') =$$

find subgradient of the function
that achieves the max

Loss Subgradients for Linear Models

- some of our loss functions are not differentiable:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

- but they are subdifferentiable:

$$\frac{\partial}{\partial w_j} \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y') = \frac{\partial}{\partial w_j} \sum_i w_i f_i(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w}))$$

find subgradient of the function that achieves the max

Loss Subgradients for Linear Models

- some of our loss functions are not differentiable:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

- but they are subdifferentiable:

$$\frac{\partial}{\partial w_j} \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y') = \frac{\partial}{\partial w_j} \sum_i w_i f_i(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w})) = f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w}))$$

find subgradient of the function
that achieves the max

Loss Subgradients for Linear Models

- perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) \; + \; \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

- subderivative for a single parameter:

$$\frac{\partial \text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w})}{\partial w_j} = -f_j(\boldsymbol{x}, y) + f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w}))$$

Loss Subgradients for Linear Models

- perceptron loss and subgradient:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) \; + \; \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

$$\frac{\partial \text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w})}{\partial w_j} = -f_j(\boldsymbol{x}, y) + f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w}))$$

- parameter update:

$$w_j \leftarrow w_j + f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w}))$$

Loss Subgradients for Linear Models

- perceptron loss and subgradient:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

$$\frac{\partial \text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w})}{\partial w_j} = -f_j(\boldsymbol{x}, y) + f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w}))$$

- parameter update:

$$w_j \leftarrow w_j + \eta \left(f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w})) \right)$$

step size / learning rate for stochastic subgradient descent

Perceptron Loss with Regularization

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}_{\text{perc}}(\boldsymbol{x}^{(i)}, y^{(i)}, \mathbf{w}) + \lambda R_{\text{L2}}(\mathbf{w})$$

$$R_{\text{L2}}(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_i w_i^2$$

- update rule from before:

$$w_j \leftarrow w_j + \eta\left(f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w}))\right)$$

- with L2 regularization:

Perceptron Loss with Regularization

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{|\mathcal{T}|} \text{loss}_{\text{perc}}(\boldsymbol{x}^{(i)}, y^{(i)}, \mathbf{w}) + \lambda R_{\text{L2}}(\mathbf{w})$$

$$R_{\text{L2}}(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_i w_i^2$$

- update rule from before:

$$w_j \leftarrow w_j + \eta \left(f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w})) \right)$$

- with L2 regularization:

$$w_j \leftarrow w_j + \eta \left(f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w})) - 2\lambda w_j \right)$$

Perceptron Loss with Regularization

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \text{loss}_{\text{perc}}(\boldsymbol{x}^{(i)}, y^{(i)}, \mathbf{w}) + \lambda R_{\text{L2}}(\mathbf{w})$$

$$R_{\text{L2}}(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_i w_i^2$$

- update rule from before:

$$w_j \leftarrow w_j + \eta \left(f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w})) \right)$$

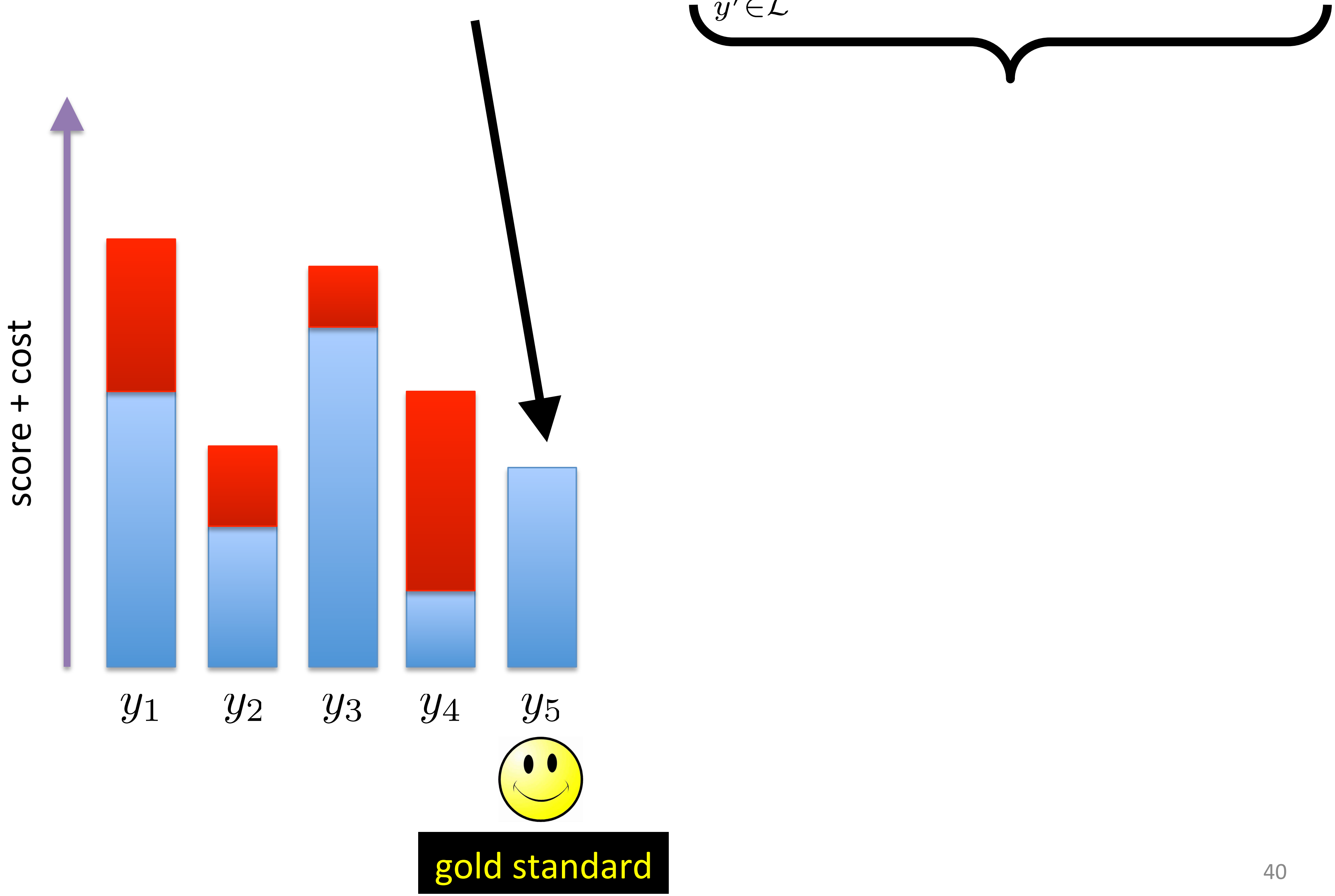
- with L2 regularization:

$$w_j \leftarrow w_j + \eta \left(f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w})) - 2\lambda w_j \right)$$

pushes weights closer to zero ("weight decay")

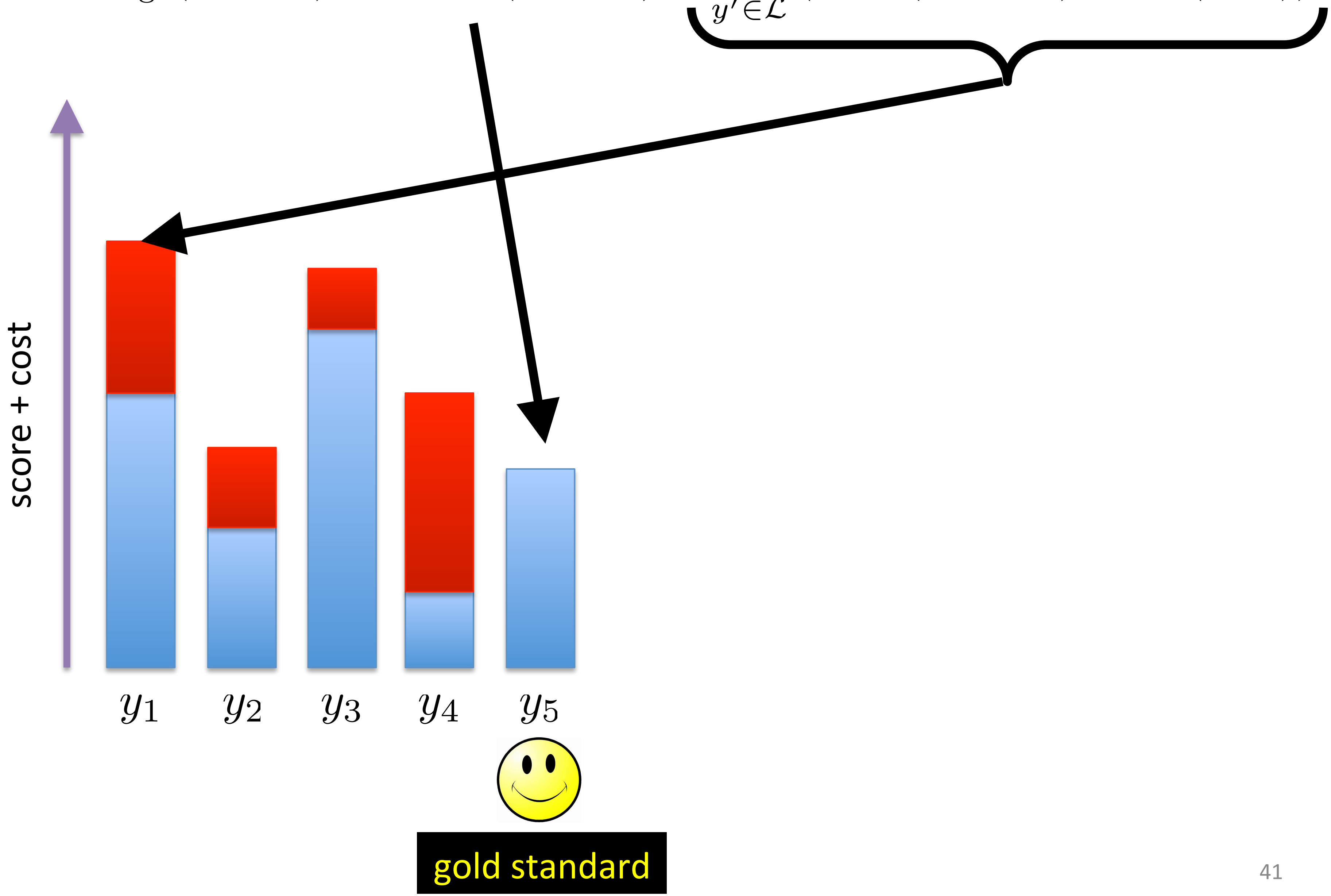
hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \left(\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y')\right)$$



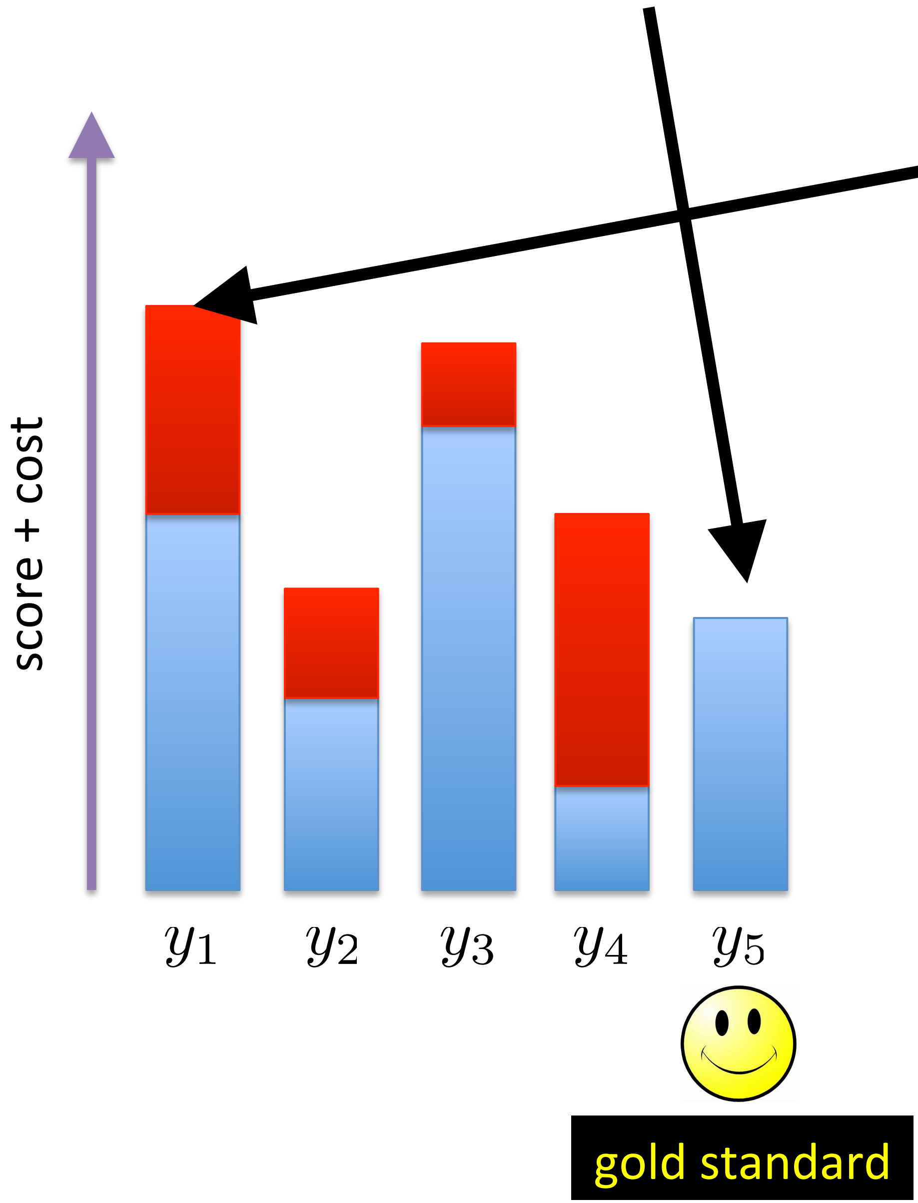
hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y'))$$



hinge loss:

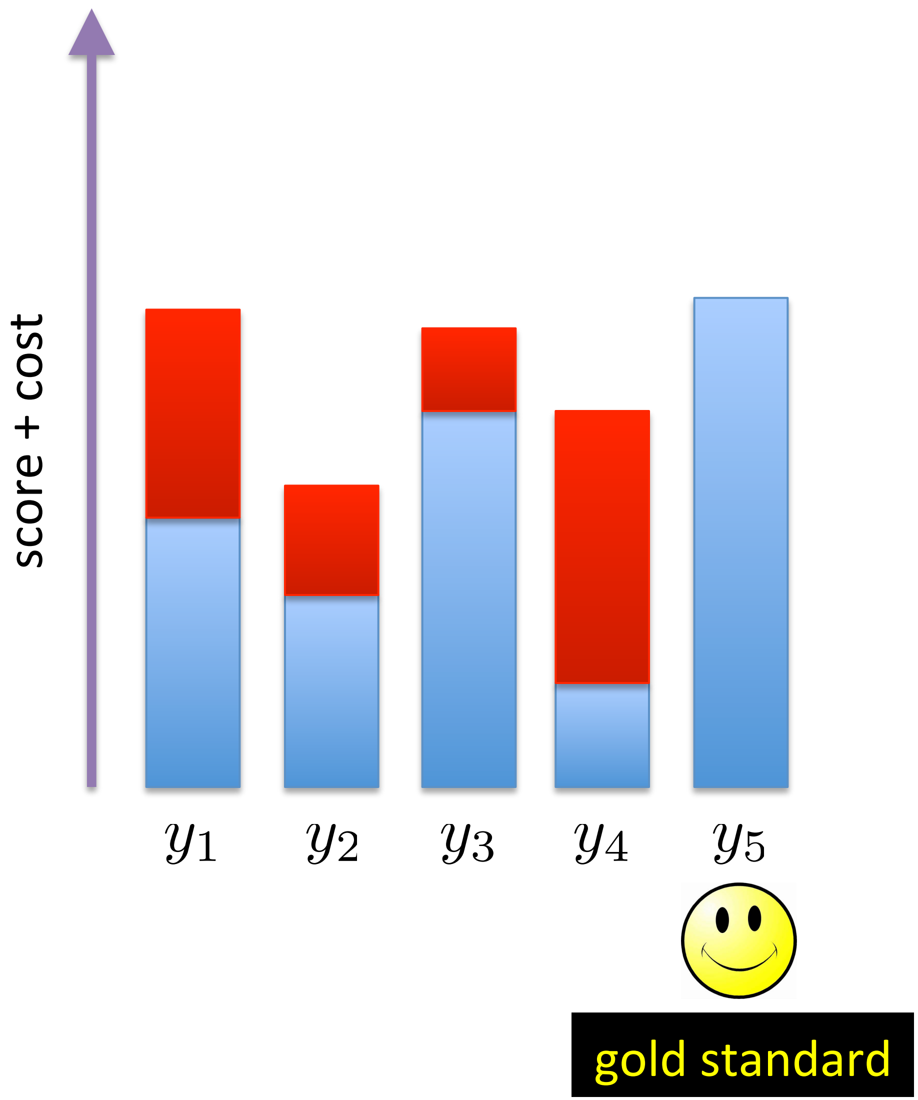
$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y'))$$



effect of learning?

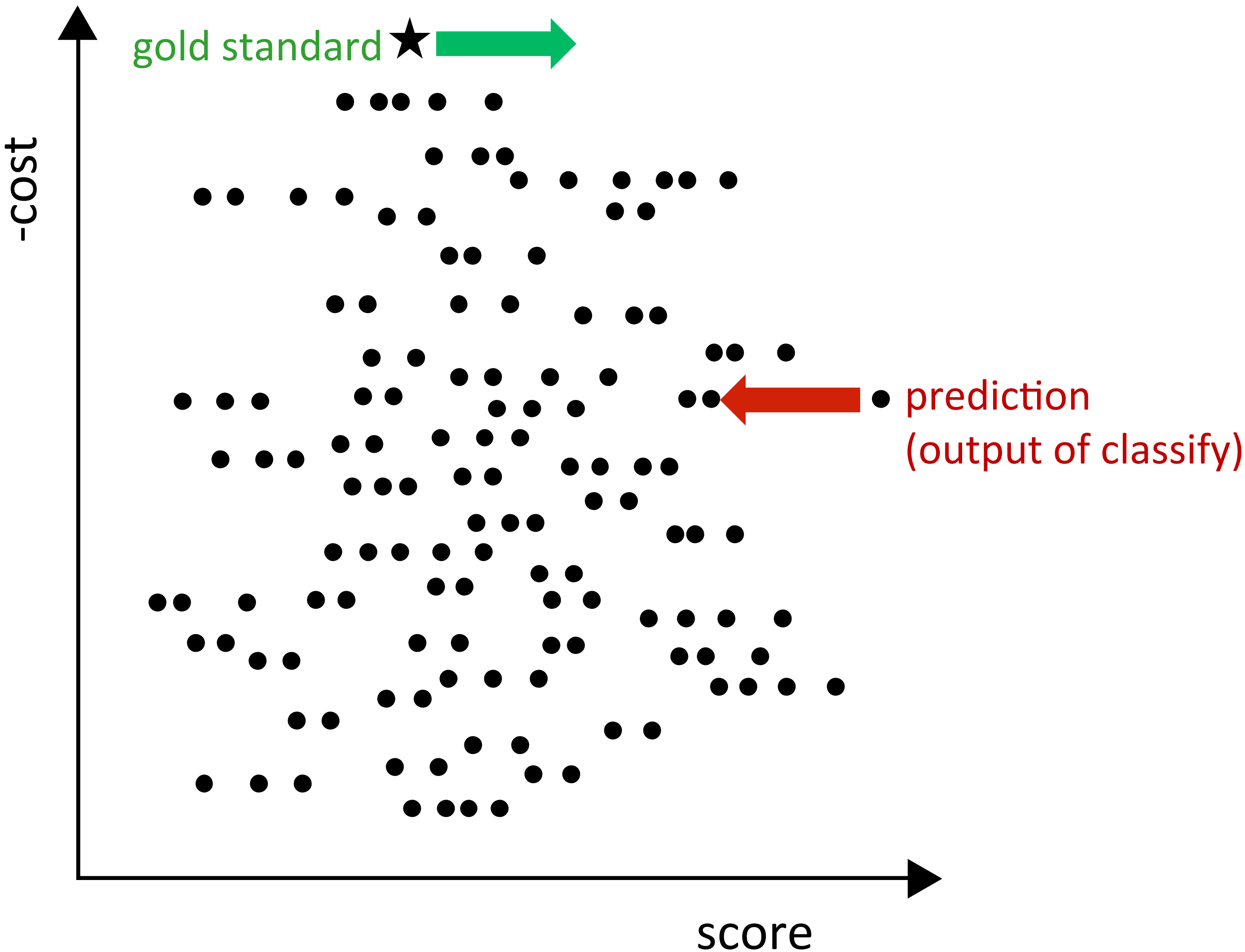
hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \left(\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y')\right)$$

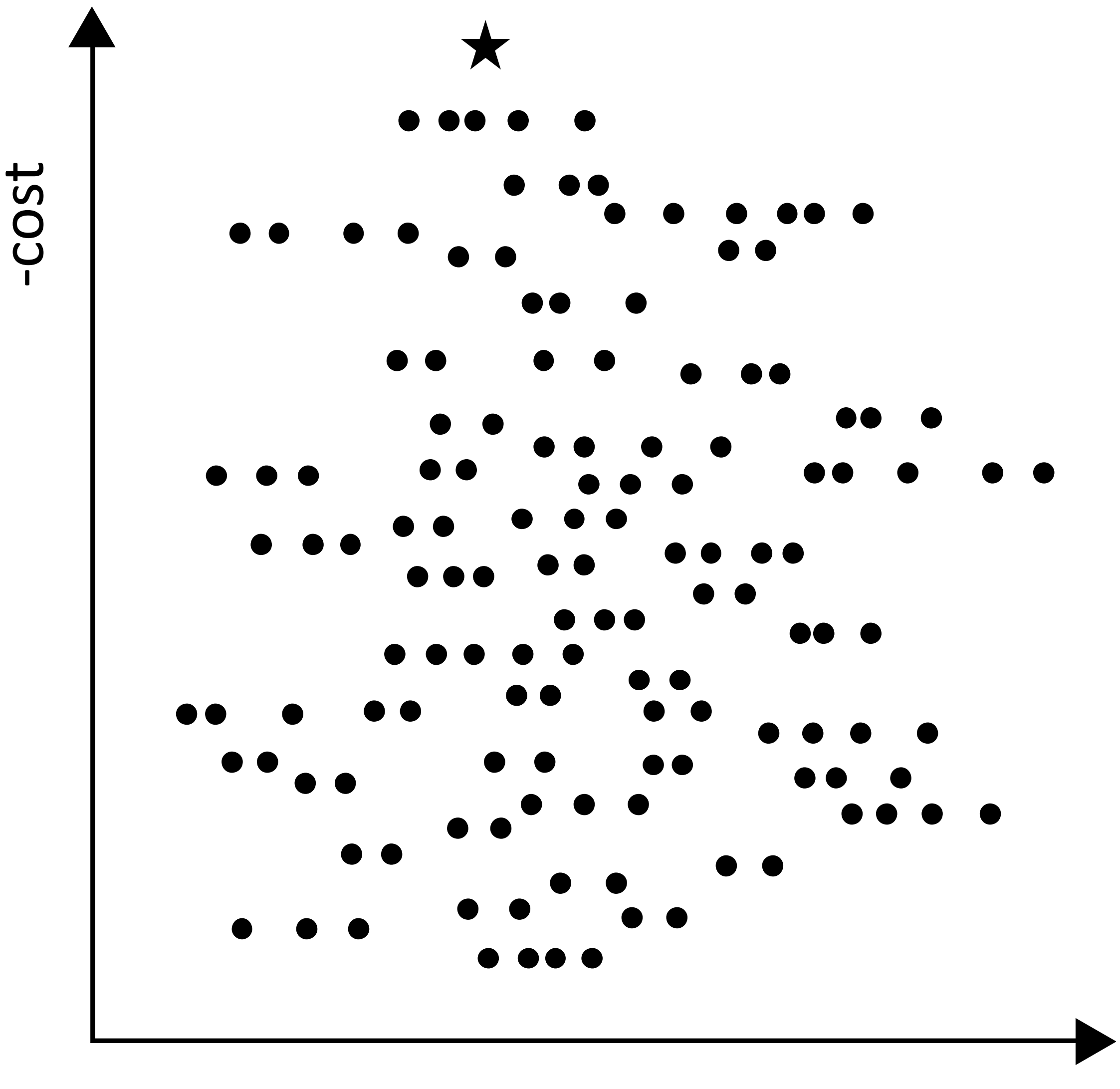


effect of learning:
score of gold standard will be higher than score+cost of all others

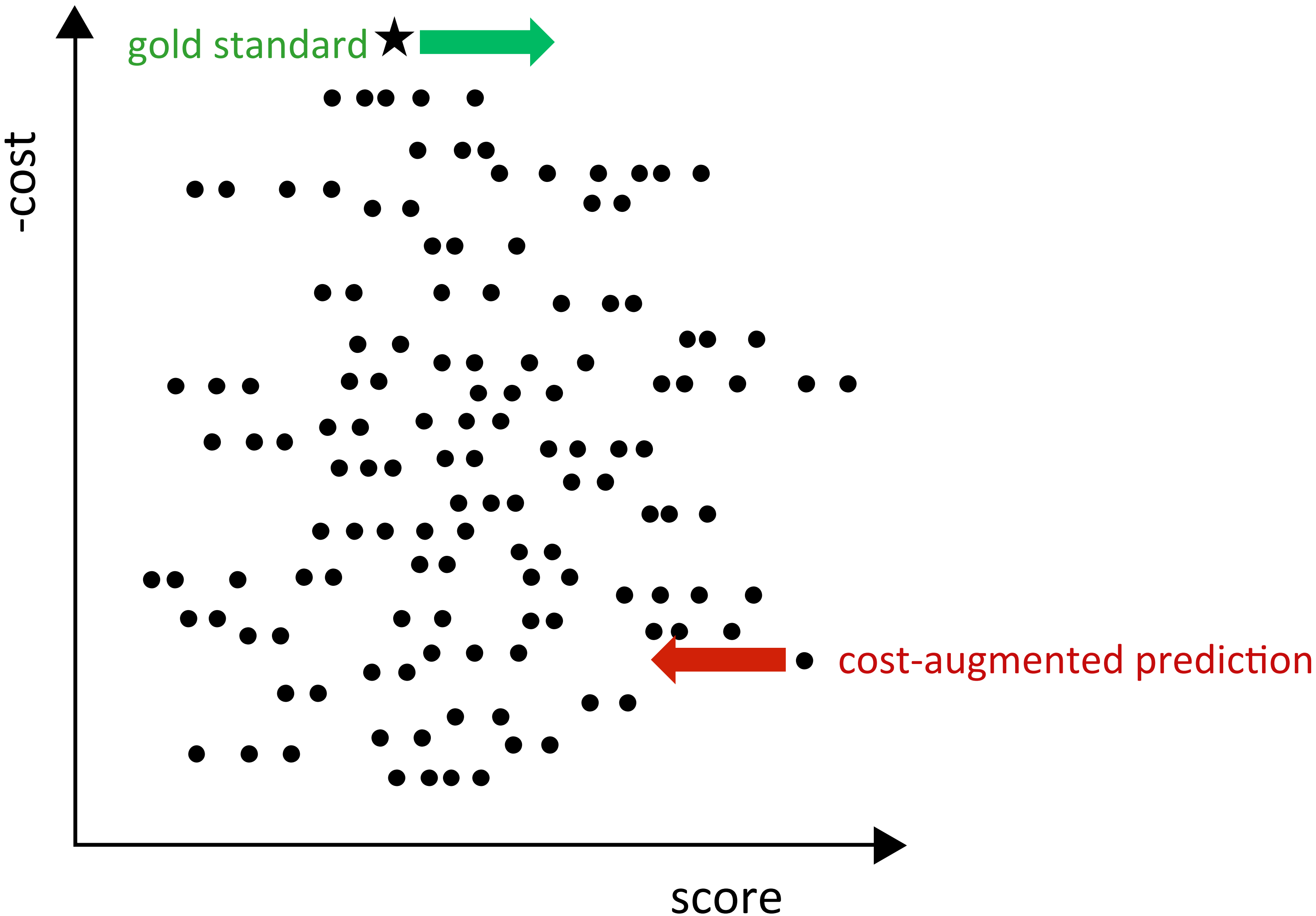
Perceptron Loss



Hinge Loss?



Hinge Loss



Perception → Hinge

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) \; + \; \max_{y' \in \mathcal{L}} \sum_i w_i f_i(\boldsymbol{x}, y')$$

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \left(\sum_i w_i f_i(\boldsymbol{x}, y') + \text{cost}(y, y') \right)$$

Loss Subgradients for Linear Models

- hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \left(\sum_i w_i f_i(\boldsymbol{x}, y') + \text{cost}(y, y') \right)$$

- subderivative for a single parameter:

Loss Subgradients for Linear Models

- hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \left(\sum_i w_i f_i(\boldsymbol{x}, y') + \text{cost}(y, y') \right)$$

- subderivative for a single parameter:

$$\frac{\partial \text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \mathbf{w})}{\partial w_j} = -f_j(\boldsymbol{x}, y) + f_j(\boldsymbol{x}, \text{costClassify}(\boldsymbol{x}, y, \mathbf{w}))$$

Loss Subgradients for Linear Models

- hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \mathbf{w}) = -\sum_i w_i f_i(\boldsymbol{x}, y) + \max_{y' \in \mathcal{L}} \left(\sum_i w_i f_i(\boldsymbol{x}, y') + \text{cost}(y, y') \right)$$

- subderivative for a single parameter:

$$\frac{\partial \text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \mathbf{w})}{\partial w_j} = -f_j(\boldsymbol{x}, y) + f_j(\boldsymbol{x}, \text{costClassify}(\boldsymbol{x}, y, \mathbf{w}))$$

$$\text{costClassify}(\boldsymbol{x}, y, \mathbf{w}) = \operatorname*{argmax}_{y' \in \mathcal{L}} \left(\sum_i w_i f_i(\boldsymbol{x}, y') + \text{cost}(y, y') \right)$$

"cost-augmented inference" or "cost-augmented decoding"

Feature count cut-off of zero?

- perceptron loss update rule:

$$w_j \leftarrow w_j + \eta \left(f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w})) \right)$$

- what do you expect to happen to weights of features with count 0 in the training data? (if they are initialized to 0)

Feature count cut-off of zero?

- perceptron loss update rule:

$$w_j \leftarrow w_j + \eta\left(f_j(\boldsymbol{x}, y) - f_j(\boldsymbol{x}, \text{classify}(\boldsymbol{x}, \mathbf{w}))\right)$$

- what do you expect to happen to weights of features with count 0 in the training data? (if they are initialized to 0)
  - they will stay at zero or become negative

Roadmap

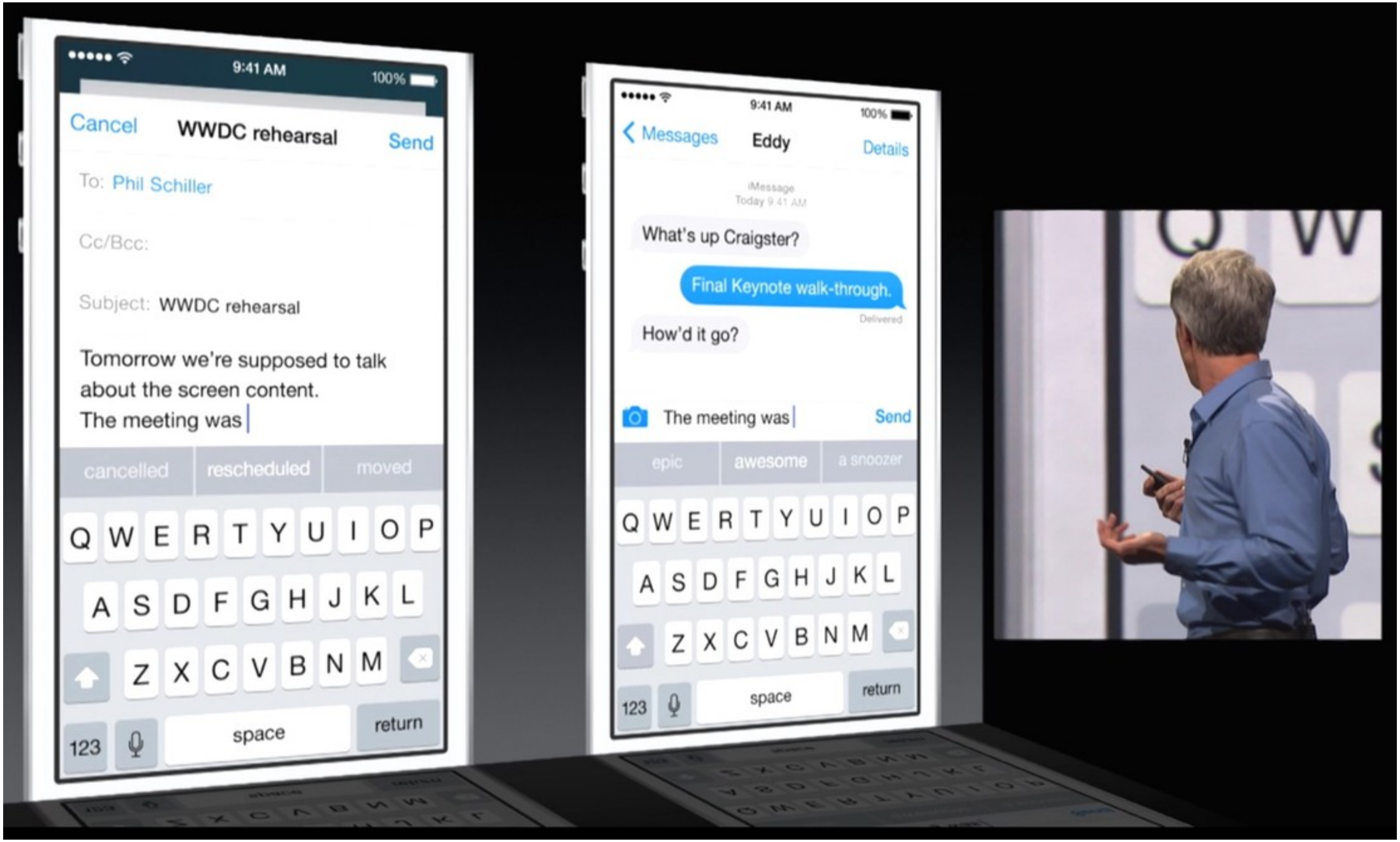
- words, morphology, lexical semantics
- text classification
- simple neural methods for NLP
- language modeling and word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks

Probabilistic Language Models

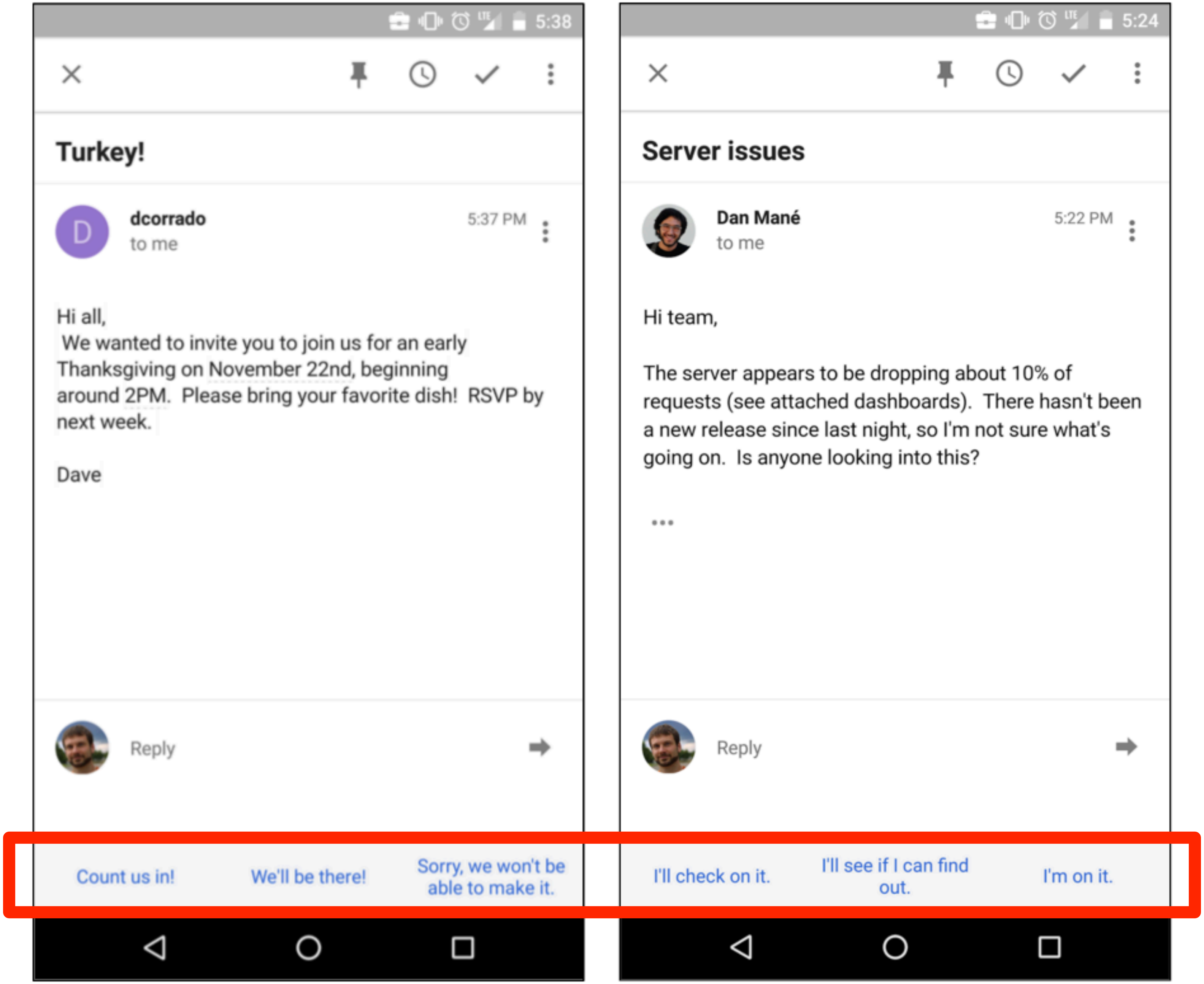
- language modeling: assign probabilities to sentences
- Why?
  - machine translation:
    - P(**high** winds tonite) > P(**large** winds tonite)
  - spelling correction:
    - The office is about fifteen **minuets** from my house
      - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
  - speech recognition:
    - P(I saw a van) >> P(eyes awe of an)
  - summarization, question answering, etc.!

J&M/SLP3

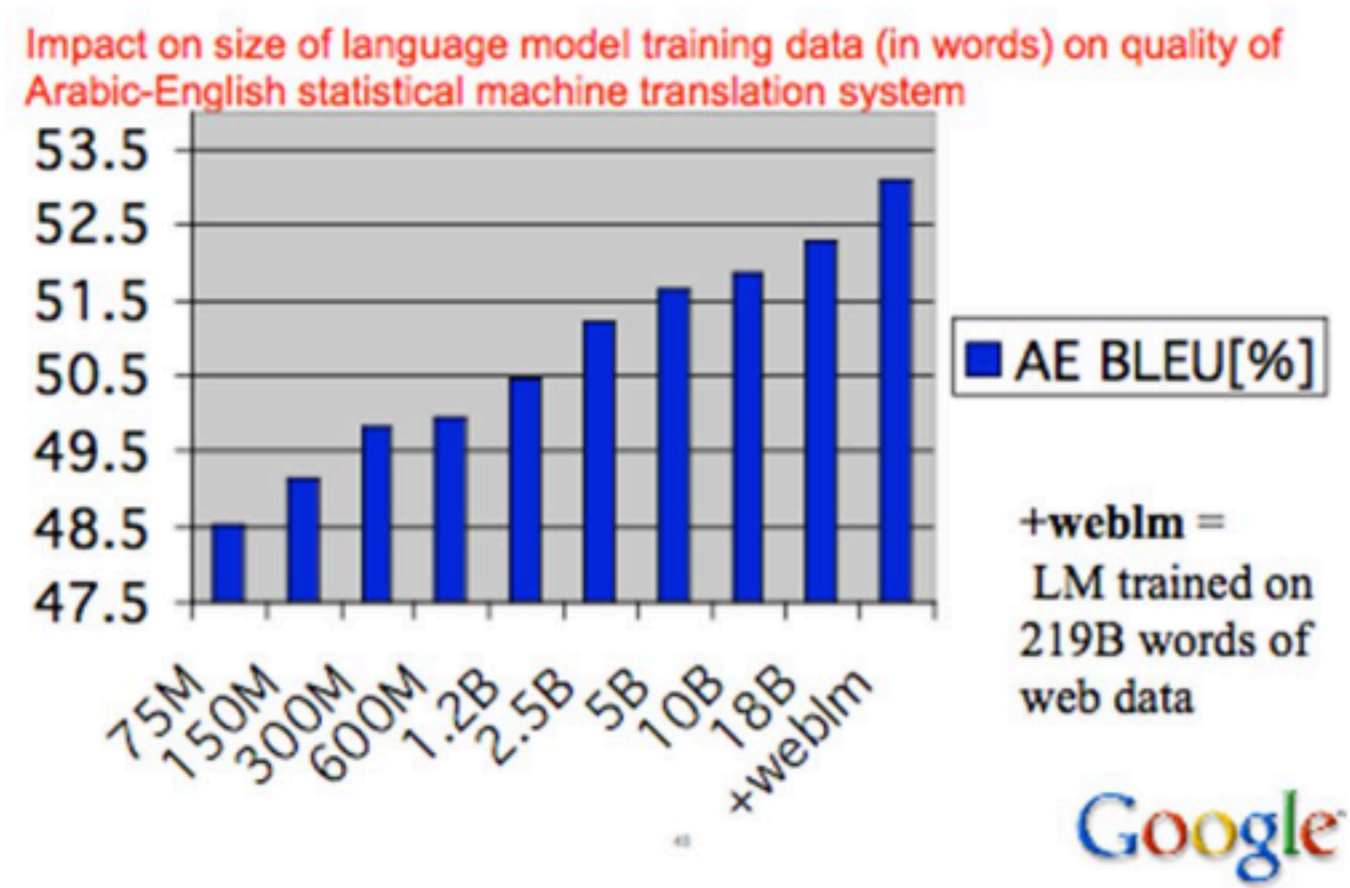
Automatic Completion



Automatic Completion


5:38
Turkey!
dcorrado
to me
5:37 PM
Hi all,
We wanted to invite you to join us for an early Thanksgiving on November 22nd, beginning around 2PM. Please bring your favorite dish! RSVP by next week.
Dave
Reply
Count us in!
We'll be there!
Sorry, we won't be able to make it.
5:24
Server issues
Dan Mané
to me
5:22 PM
Hi team,
The server appears to be dropping about 10% of requests (see attached dashboards). There hasn't been a new release since last night, so I'm not sure what's going on. Is anyone looking into this?
Reply
I'll check on it.
I'll see if I can find out.
I'm on it.

Language Modeling for Machine Translation



Probabilistic Language Modeling

- goal: compute the probability of a sequence of words:

$$P(\boldsymbol{w}) = P(w_1, w_2, ..., w_n)$$

- related task: probability of next word:

$$P(w_4 \mid w_1, w_2, w_3)$$

- a model that computes either of these:

$$P(\boldsymbol{w}) \quad \text{or} \quad P(w_k \mid w_1, w_2, ..., w_{k-1})$$

is called a **language model (LM)**

J&M/SLP3

How to compute $P(\boldsymbol{w})$

- How to compute this joint probability:
  - P(its, water, is, so, transparent, that)
- Intuition: let's rely on the Chain Rule of Probability

Reminder: Chain Rule

- factor joint probability into product of conditional probabilities:

$$P(w_1, w_2, ..., w_n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \dots P(w_n \mid w_1, w_2, ..., w_{n-1})$$

- we have not yet made any independence assumptions

Chain Rule for computing joint probability of words in sentence

$$P(w_1, w_2, ..., w_n) = \prod_i P(w_i \mid w_1, w_2, ..., w_{i-1})$$

P("*its water is so transparent*") =

P(*its*) × P(*water* | *its*) × P(*is* | *its water*)

× P(*so* | *its water is*) × P(*transparent* | *its water is so*)

How to estimate these probabilities

- could we just count and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

- no! too many possible sentences!
- we'll never see enough data for estimating these

Markov Assumption

Andrei Markov

- simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- or maybe:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption

- i.e., we approximate each component in the product:

$$P(w_i \mid w_1, ..., w_{i-2}, w_{i-1}) \approx P(w_i \mid w_{i-k}, ..., w_{i-2}, w_{i-1})$$

Simplest case: Unigram model

$$P(w_1, w_2, ..., w_n) = \prod_i P(w_i)$$

automatically generated sentences from a unigram model:

```
fifth an of futures the an incorporated a a the
inflation most dollars quarter in is mass

thrift did eighty said hard ’m july bullish

that or limited the
```

Bigram model

condition on the previous word:

$$P(w_1, w_2, ..., w_n) = \prod_i P(w_i \mid w_{i-1})$$

automatically generated sentences from a bigram model:

```
texaco rose one in this issue is pursuing growth in a boiler
house said mr. gurria mexico 's motion control proposal
without permission from five hundred fifty five yen

outside new car parking lot of the agreement reached

this would be a record november
```

J&M/SLP3

n-gram models

- we can extend to trigrams, 4-grams, 5-grams
- in general this is an insufficient model of language
  - because language has **long-distance dependencies**:

"The computer which I had just put into the machine room on the fifth floor crashed."

- but we can often get away with n-gram models