Linear Models for Classification: Discriminative Learning (Perceptron, SVMs, MaxEnt)

Nathan Schneider
(some slides borrowed from Chris Dyer)
ENLP | 8 February 2024

Outline

- Words, probabilities → Features, weights

previous lecture

- Geometric view: decision boundary

this lecture

- Perceptron
- Generative vs. Discriminative
- More discriminative models: Logistic regression/MaxEnt; SVM
- Loss functions, optimization
- Regularization; sparsity

Perceptron Learner

$\mathbf{X} \rightarrow$

$\mathbf{Y} \rightarrow$

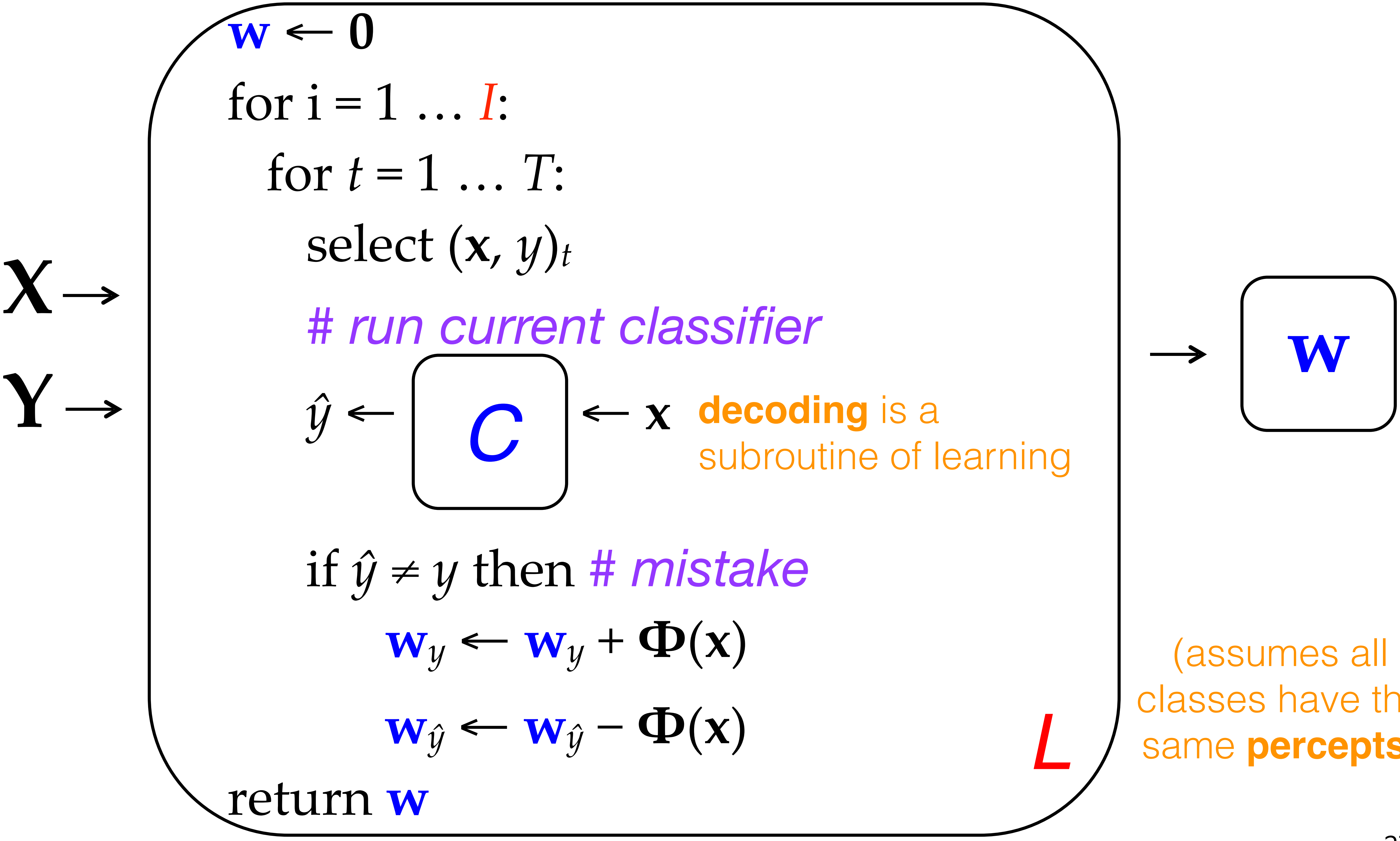
```
$\mathbf{w} \leftarrow \mathbf{0}$
for i = 1 … $I$:
    for $t$ = 1 … $T$:
        select $(\mathbf{x}, y)_t$
        # run current classifier
        $\hat{y} \leftarrow \arg\max_{y'} \mathbf{w}_{y'}^{\top} \mathbf{\Phi}(\mathbf{x})$

        if $\hat{y} \neq y$ then # mistake
            $\mathbf{w}_y \leftarrow \mathbf{w}_y + \mathbf{\Phi}(\mathbf{x})$
            $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - \mathbf{\Phi}(\mathbf{x})$
return $\mathbf{w}$
```

L

$\rightarrow \boxed{\mathbf{w}}$

(assumes all classes have the same **percepts**)

Perceptron Learner



```
$\mathbf{w} \leftarrow \mathbf{0}$
for i = 1 … $I$:
  for $t$ = 1 … $T$:
    select $(\mathbf{x}, y)_t$
    # run current classifier
    $\hat{y} \leftarrow \boxed{C} \leftarrow \mathbf{x}$
    if $\hat{y} \neq y$ then # mistake
      $\mathbf{w}_y \leftarrow \mathbf{w}_y + \mathbf{\Phi}(\mathbf{x})$
      $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - \mathbf{\Phi}(\mathbf{x})$
return $\mathbf{w}$
```

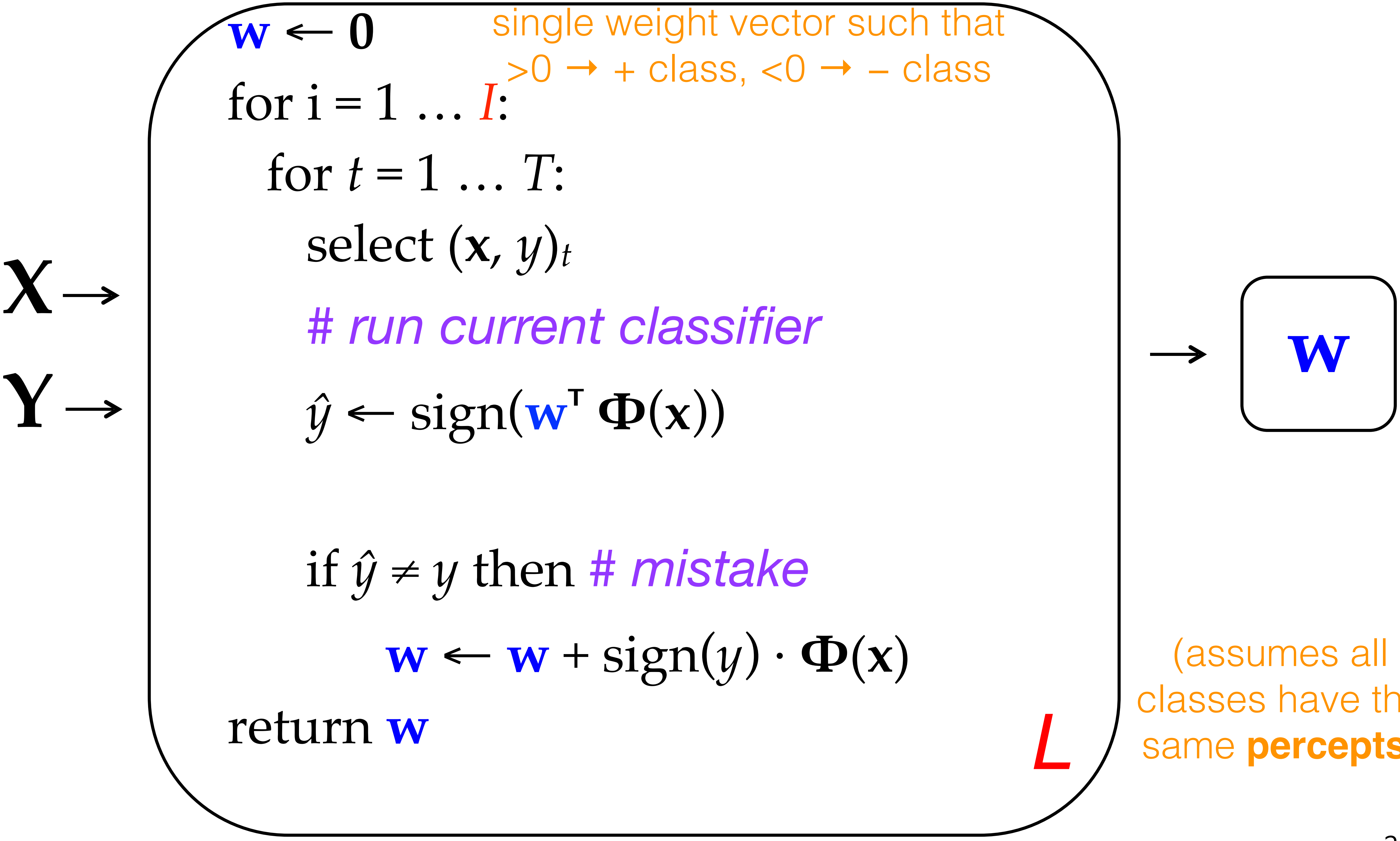
decoding is a subroutine of learning

(assumes all classes have the same **percepts**)

Perceptron Learner

for **binary classification**

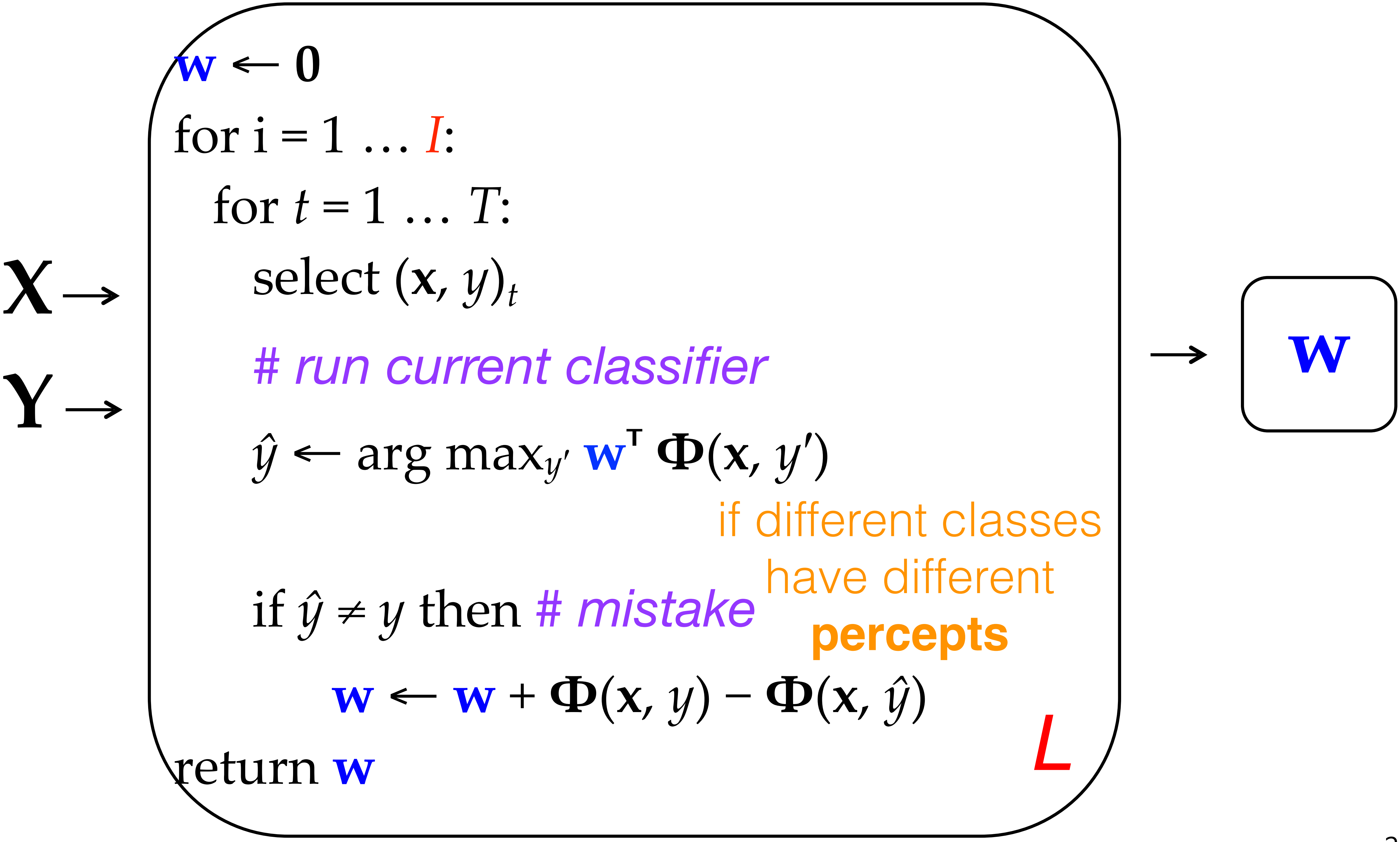
single weight vector such that
>0 → + class, <0 → – class



```
$\mathbf{w} \leftarrow \mathbf{0}$
for i = 1 … $I$:
    for $t$ = 1 … $T$:
        select $(\mathbf{x}, y)_t$
        # run current classifier
        $\hat{y} \leftarrow \text{sign}(\mathbf{w}^\top \boldsymbol{\Phi}(\mathbf{x}))$

        if $\hat{y} \neq y$ then # mistake
            $\mathbf{w} \leftarrow \mathbf{w} + \text{sign}(y) \cdot \boldsymbol{\Phi}(\mathbf{x})$
return $\mathbf{w}$
```

(assumes all classes have the same **percepts**)

Perceptron Learner



```
$\mathbf{w} \leftarrow \mathbf{0}$
for i = 1 … $I$:
    for $t$ = 1 … $T$:
        select $(\mathbf{x}, y)_t$
        # run current classifier
        $\hat{y} \leftarrow \arg\max_{y'} \mathbf{w}^\top \mathbf{\Phi}(\mathbf{x}, y')$
        if $\hat{y} \neq y$ then # mistake
            $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{\Phi}(\mathbf{x}, y) - \mathbf{\Phi}(\mathbf{x}, \hat{y})$
return $\mathbf{w}$
```

if different classes have different **percepts**

work through example on the board

x1 = “I thought it was great”	y1 = +
x2 = “not so great”	y2 = −
x3 = “good but not great”	y3 = +

Perceptron Learner

- **The perceptron doesn't estimate probabilities.** It just adjusts weights up or down until they classify the training data correctly.
  - No assumptions of feature independence necessary! ⇒ Better accuracy than NB
- The perceptron is an example of an **online** learning algorithm because it potentially updates its parameters (weights) with each training datapoint.
- Classification, a.k.a. **decoding**, is called with the latest weight vector. Mistakes lead to weight updates.
- One hyperparameter: I, the number of iterations (passes through the training data).
- Often desirable to make several passes over the training data. The number can be tuned. Under certain assumptions, it can be proven that the learner will converge.

Perceptron: Avoiding overfitting

- Like any learning algorithm, the perceptron risks overfitting the training data. Two main techniques to improve generalization:
  - **Averaging:** Keep a copy of each weight vector as it changes, then average all of them to produce the final weight vector. Daumé chapter has a trick to make this efficient with large numbers of features.
  - **Early stopping:** Tune I by checking held-out accuracy on dev data (or cross-val on train data) after each iteration. If accuracy has ceased to improve, stop training and use the model from iteration $I - 1$.

Questions for Discussion

- After running the perceptron to train a binary classifier, a binary feature observed in the training data has a final weight of 0. From this, can you conclude it was observed an equal number of times for the two classes? Why or why not?

- What would be some good features for a **named entity classifier**, where the input is a word in a sentence and the output is one of: {NonName, Person, Org, Location}?

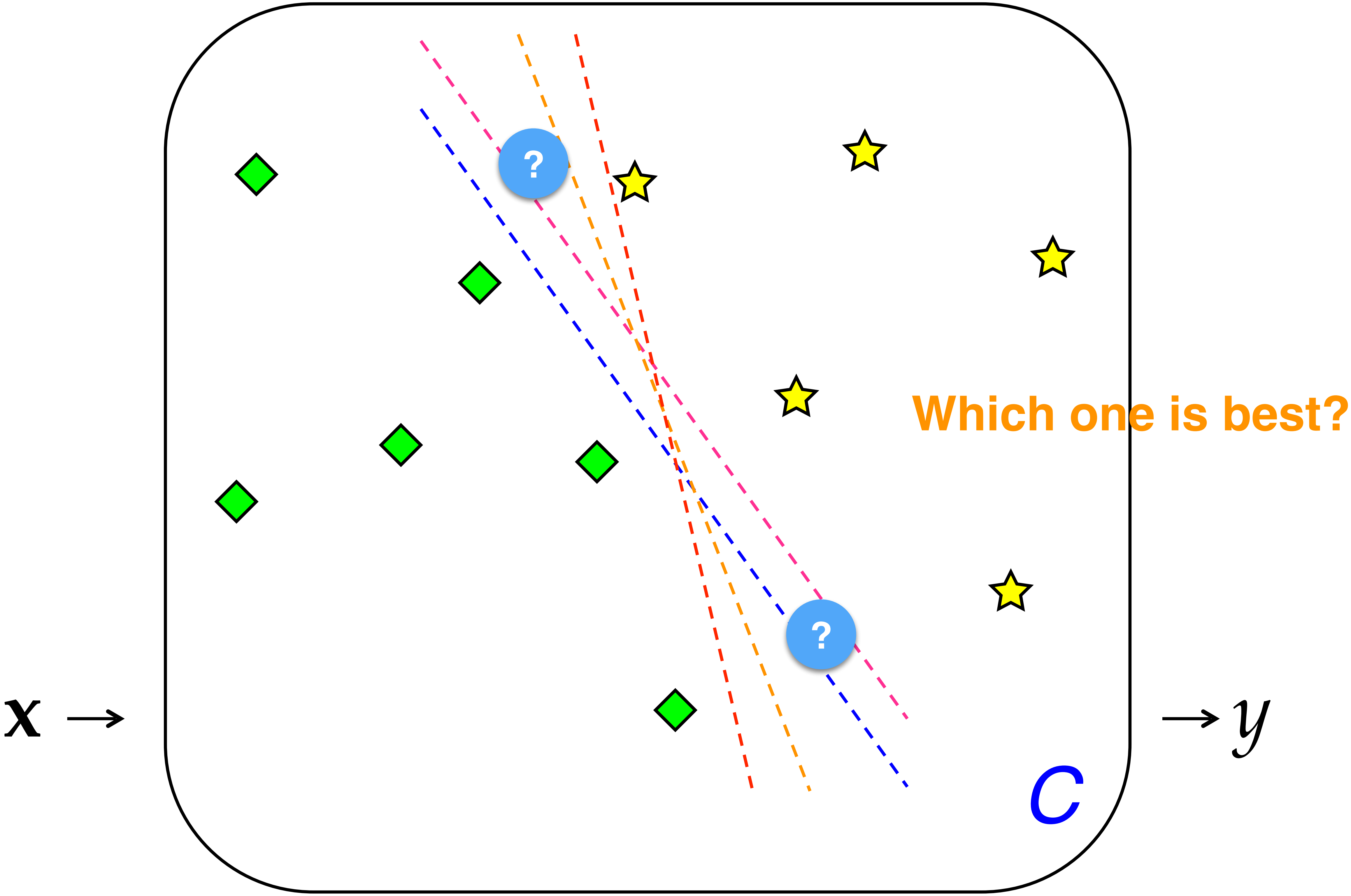
Generative vs. Discriminative

- Naïve Bayes allows us to classify via the **joint probability** of $\mathbf{x}$ and y:
  - $p(y \mid \mathbf{x}) \propto p(y) \prod_{w \in \mathbf{x}} p(w \mid y)$

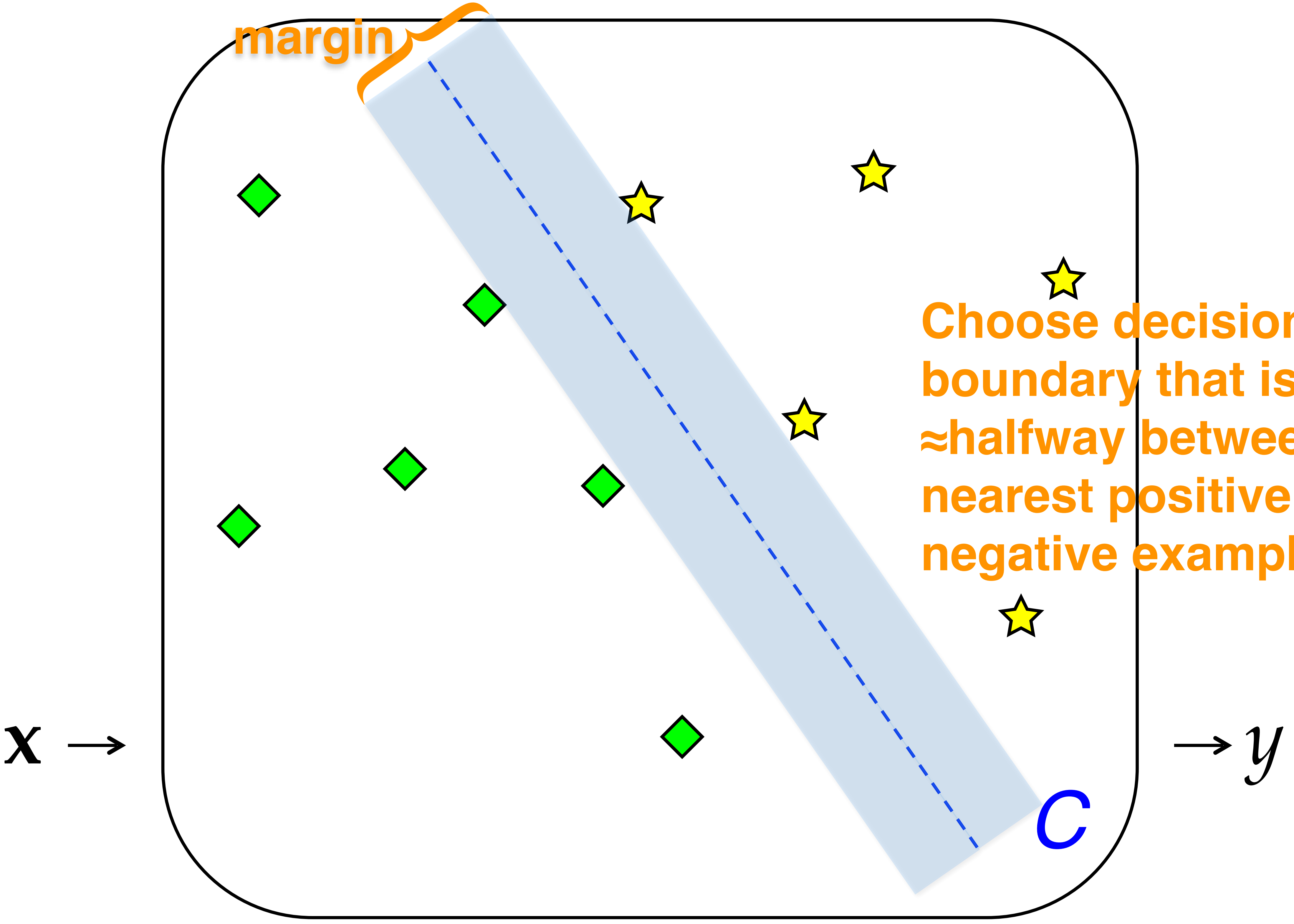
    $= p(y)\, p(\mathbf{x} \mid y)$ (per the independence assumptions of the model)

    $= p(y, \mathbf{x})$ (chain rule)
  - This means the model accounts for BOTH **x** and *y*. From the joint distribution p(**x**,*y*) it is possible to compute p(**x**) as well as p(*y*), p(**x** | *y*), and p(*y* | **x**).
- NB is called a **generative** model because it assigns probability to linguistic objects (**x**). It could be used to generate "likely" language corresponding to some *y*. (Subject to its naïve modeling assumptions!)
  - (Not to be confused with the "generative" school of linguistics.)
- Some other linear models, including the perceptron, are **discriminative**: they are trained directly to classify given **x**, and cannot be used to estimate the probability of **x** or generate **x** | *y*.

Many possible decision boundaries



Max-Margin Methods (e.g., SVM)



Choose decision boundary that is ≈halfway between nearest positive and negative examples

Max-Margin Methods

- **Support Vector Machine (SVM)**: most popular max-margin variant

- Closely related to the perceptron; can be optimized (learned) with a slight tweak to the perceptron algorithm.

- Like perceptron, discriminative, non-probabilistic.

Maximum Entropy (MaxEnt) a.k.a. (Multinomial) Logistic Regression

- What if we want a discriminative classifier with **probabilities**?
  - E.g., need confidence of prediction, or want the full distribution over possible classes
- Wrap the linear score computation ($\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y')$) in the **softmax** function:

  score can be negative; exp(score) is always positive

  - $$\log \mathrm{p}(y \mid \mathbf{x}) = \log \frac{\exp(\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y))}{\Sigma_{y'} \exp(\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y'))} = \mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y) - \log \Sigma_{y'} \exp(\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y'))$$

    Denominator = normalization (makes probabilities sum to 1).

    Sum over all classes ⇒ same for all numerators ⇒ can be ignored at classification time.

  - **Binary case:**

    $$\log \mathrm{p}(y{=}1 \mid \mathbf{x}) = \log \frac{\exp(\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y{=}1))}{\exp(\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y{=}1)) + \exp(\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y{=}0))}$$

    $$= \log \frac{\exp(\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y{=}1))}{\exp(\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y{=}1)) + 1}$$ (fixing $\mathbf{w}^\intercal\, \mathbf{\Phi}(\mathbf{x}, y{=}0) = 0$)

- MaxEnt classifiers are a special case of **MaxEnt** a.k.a. **log-linear models**.
  - Why the term "Maximum Entropy"? See Smith *Linguistic Structure Prediction*, appendix C.

Objectives

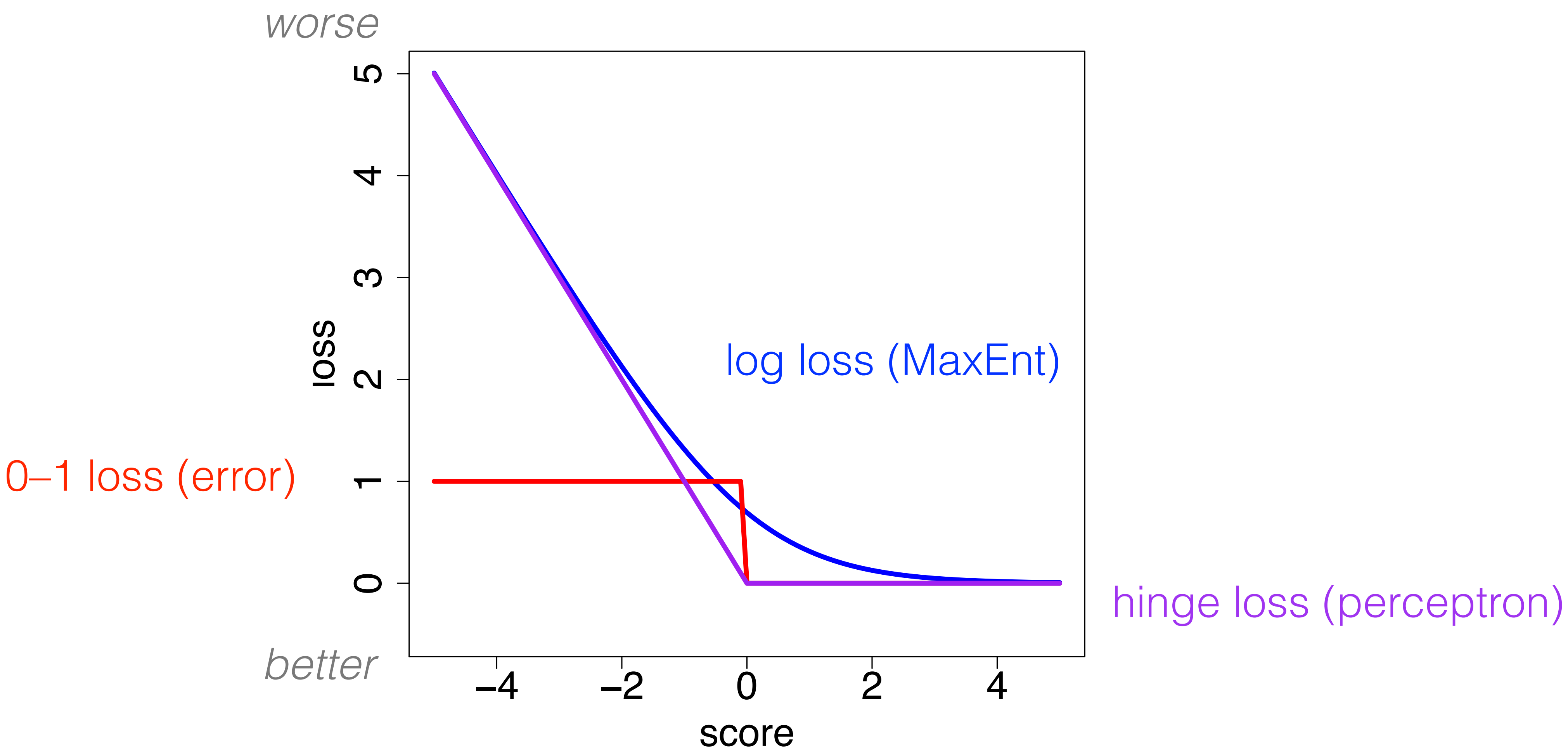
- For all linear models, the **classification rule** or **decoding objective** is: $y \leftarrow \arg\max_{y'} \mathbf{w}^\top \mathbf{\Phi}(\mathbf{x}, y')$
  - Objective function = function for which we want to find the optimum (in this case, the max)
- There is also a **learning objective** for which we want to find the optimal **parameters**. Mathematically, NB, MaxEnt, SVM, and perceptron all optimize different learning objectives.
  - When the learning objective is formulated as a **minimization** problem, it's called a **loss** function.
  - A loss function scores the "badness" of the training data under any possible set of parameters. Learning = choosing the parameters that minimize the badness.

Objectives

- Naïve Bayes learning objective: **joint data likelihood**
  - $\mathbf{p}^* \leftarrow \arg\max_{\mathbf{p}} L_{\text{joint}}(\mathbf{D}; \mathbf{p})$
    $= \arg\max_{\mathbf{p}} \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathbf{D}} \log \mathbf{p}(\mathbf{x}, y) = \arg\max_{\mathbf{p}} \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathbf{D}} \log (\mathbf{p}(y)\mathbf{p}(\mathbf{x} \mid y))$
  - It can be shown that relative frequency estimation (i.e., count and divide, no smoothing) is indeed the maximum likelihood estimate
- MaxEnt learning objective: **conditional log likelihood**
  - $\mathbf{p}^* \leftarrow \arg\max_{\mathbf{p}} L_{\text{cond}}(\mathbf{D}; \mathbf{p})$
    $= \arg\max_{\mathbf{p}} \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathbf{D}} \log \mathbf{p}(y \mid \mathbf{x})$
    $\mathbf{w} \leftarrow \arg\max_{\mathbf{w}} \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathbf{D}} \mathbf{w}^{\top} \boldsymbol{\Phi}(\mathbf{x}, y) - \log \Sigma_{y'} \exp(\mathbf{w}^{\top} \boldsymbol{\Phi}(\mathbf{x}, y'))$ [2 slides ago]
  - This has no closed-form solution. Hence, we need an optimization algorithm to try different weight vectors and choose the best one.
  - With thousands or millions of parameters—not uncommon in NLP—it may also overfit.

Objectives

Visualizing different loss functions for binary classification



$\mathbf{w}^{\top}\,\boldsymbol{\Phi}(\mathbf{x}, y')$

figure from Noah Smith

Objectives

- Why not just penalize error directly if that's how we're going to evaluate our classifier (accuracy)?
    - Error is difficult to optimize! Log loss and hinge loss are easier. Why?
        - Because they're differentiable.
        - Can use stochastic (sub)gradient descent (SGD) and other gradient-based optimization algorithms (L-BFGS, AdaGrad, …). There are freely available software packages that implement these algorithms.
        - With supervised learning, these loss functions are **convex**: local optimum = global optimum (so in principle the initialization of weights doesn't matter).
        - The perceptron algorithm can be understood as a special case of subgradient descent on the hinge loss!
- N.B. I haven't explained the math for the hinge loss (perceptron) or the SVM. Or the derivation of gradients. See further reading links if you're interested.

A likelihood surface

Visualizes the likelihood objective (vertical axis) as a function of 2 parameters.
Likelihood = maximization problem. Flip upside down for the loss.

Gradient-based optimizers choose a point on the surface, look at its curvature,
and then successively move to better points.

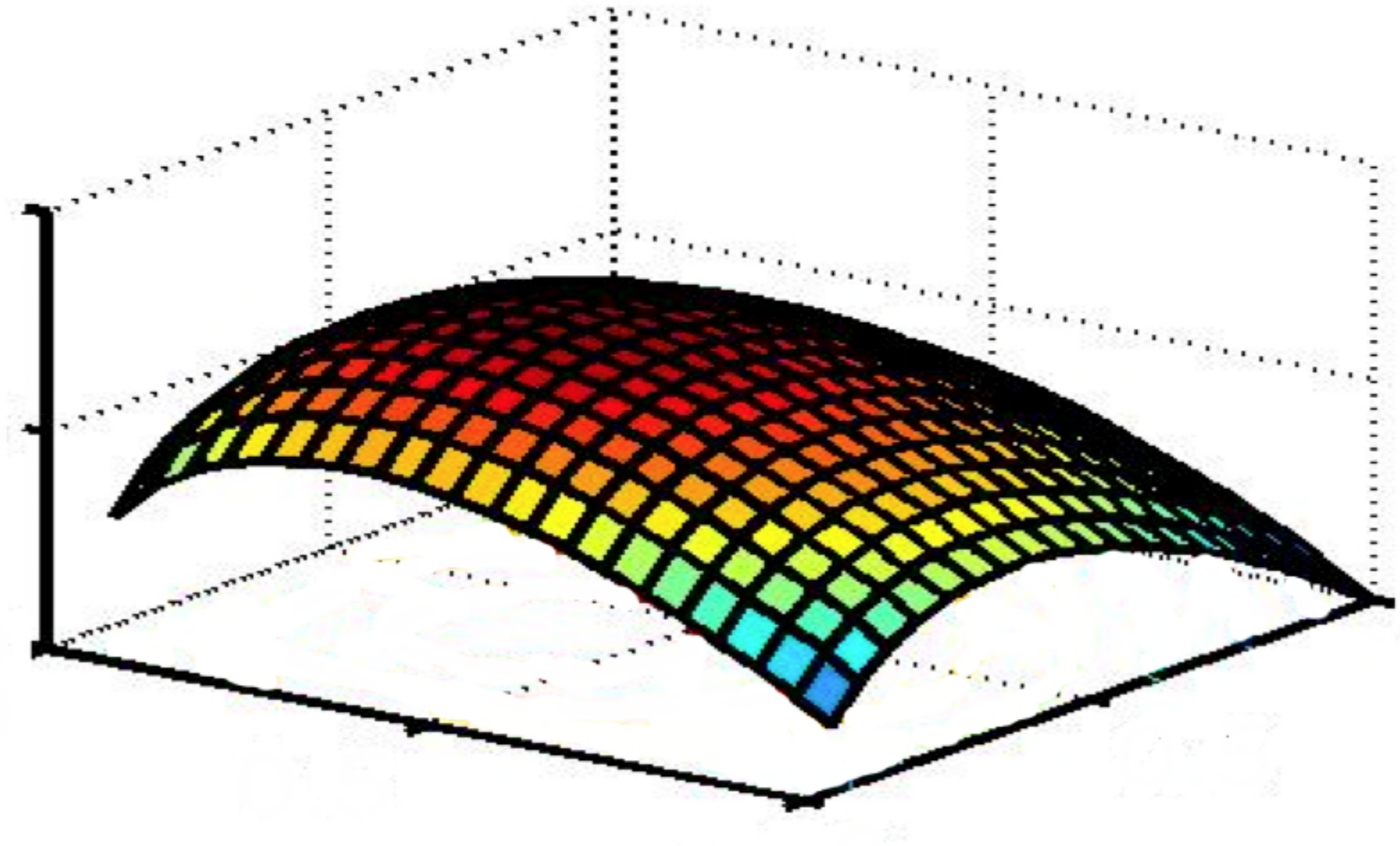


figure from Chris Manning

Regularization

- Better MaxEnt learning objective: **regularized conditional log likelihood**
  - $\mathbf{w}^* \leftarrow \arg\max_{\mathbf{w}} -\lambda \mathrm{R}(\mathbf{w}) + \boldsymbol{\Sigma}_{(\mathbf{x},\, y) \in \mathrm{D}}\ \mathbf{w}^{\mathsf{T}}\, \boldsymbol{\Phi}(\mathbf{x}, y) - \log \Sigma_{y'} \exp(\mathbf{w}^{\mathsf{T}}\, \boldsymbol{\Phi}(\mathbf{x}, y'))$

- **To avoid overfitting, the regularization term ("regularizer") $-\lambda \mathrm{R}(\mathbf{w})$ penalizes complex models (i.e., parameter vectors with many large weights).**
  - Close relationship to Bayesian prior (a priori notion of what a "good" model looks like if there is not much training data). Note that the regularizer is a function of the weights only (not the data)!

- In NLP, most popular values of $\mathrm{R}(\mathbf{w})$ are the ℓ_1 norm ("Lasso") and the ℓ_2 norm ("ridge"):
  - $\ell_2 = \|\mathbf{w}\|_2 = (\Sigma_i\, w_i^2)^{-1/2}$ encourages most weights to be **small in magnitude**
  - $\ell_1 = \|\mathbf{w}\|_1 = \Sigma_i\, |w_i|$ encourages most weights to be **0**
  - λ determines the tradeoff between regularization and data-fitting. Can be tuned on dev data.

- SVM objective also incorporates a regularization term. Perceptron does not (hence, averaging and early stopping).

Sparsity

- ℓ_1 regularization is a way to promote **model sparsity**: many weights are pushed to 0.
  - A vector is sparse if (# **nonzero** parameters) « (total # parameters).
  - Intuition: if we define very general feature templates—e.g. one feature per word in the vocabulary—we expect that *most features should not matter* for a particular classification task.
- In NLP, we typically have sparsity in our **feature vectors** as well.
  - E.g., in WSD, all words in the training data but *not* in context of a particular token being classified are effectively 0-valued features.
  - Exception: **dense** word representations popular in recent neural network models (we'll get to this later in the course).
- Sometimes the word "sparsity" or "sparseness" just means "not very much data."

Summary: Linear Models

Classifier: $y \leftarrow \arg\max_{y'} \mathbf{w}^\intercal \mathbf{\Phi}(\mathbf{x}, y')$

	kind of model	*loss function*	*learning algorithm*	*avoiding overfitting*
Naïve Bayes	Probabilistic, generative	Likelihood	Closed-form estimation	Smoothing
Logistic regression (MaxEnt)	Probabilistic, discriminative	Conditional likelihood	Optimization	Regularization penalty
Perceptron	Non-probabilistic, discriminative	Hinge	Optimization	Averaging; Early stopping
SVM (linear kernel)	Non-probabilistic, discriminative	Max-margin	Optimization	Regularization penalty

Take-home points

- Feature-based linear classifiers are helpful for NLP tasks where interpretability is important.
    - You define the features, an algorithm chooses the weights. Some classifiers then exponentiate and normalize to give probabilities.
    - More features ⇒ more flexibility, also more risk of overfitting. Because we work with large vocabularies, not uncommon to have millions of features.
- Learning objective/loss functions formalize training as choosing parameters to optimize a function.
    - Some model **both** the language and the class (generative); some directly model the class *conditioned on* the language (discriminative).
    - In general: **Generative** ⇒ training is cheaper, but lower accuracy.

      Discriminative ⇒ higher accuracy with sufficient training data and computation (optimization).
- Some models, like naïve Bayes, have a closed-form solution for parameters. Learning is cheap!
- Other models require fancier optimization algorithms that may iterate multiple times over the data, adjusting parameters until convergence (or some other stopping criterion).
    - The advantage: fewer modeling assumptions. Weights can be interdependent.

Which linear classifier to use?

- Fast and simple: **naïve Bayes**
- Very accurate, still simple: **perceptron**
- Very accurate, probabilistic, more complicated to implement: **MaxEnt**
- Potentially best accuracy, more complicated to implement: **SVM**
- All of these: watch out for overfitting!
- Check the web for free and fast implementations,
  e.g. scikit-learn, SVM^{light}
- Later in the course, we'll also see **nonlinear** models (neural networks)

Evaluation

- How to interpret classifier performance?
  - Tuning hyperparameters; overfitting vs. underfitting
  - Feature ablations: remove (or add) features and compare accuracy
  - Majority baseline: always predict the most frequent class in the **training** data

Further Reading: Basics & Examples

- **Manning:** features in linear classifiers
  http://www.stanford.edu/class/cs224n/handouts/MaxentTutorial-16x9-FeatureClassifiers.pdf

- **Goldwater:** naïve Bayes & MaxEnt examples
  http://www.inf.ed.ac.uk/teaching/courses/fnlp/lectures/07_slides.pdf

- **O'Connor:** MaxEnt—incl. step-by-step examples, comparison to naïve Bayes
  http://people.cs.umass.edu/~brenocon/inlp2015/04-logreg.pdf

- **Daumé:** "The Perceptron" (*A Course in Machine Learning*, ch. 3)
  http://www.ciml.info/dl/v0_8/ciml-v0_8-ch03.pdf

- **Neubig:** "The Perceptron Algorithm"
  http://www.phontron.com/slides/nlp-programming-en-05-perceptron.pdf

Further Reading: Advanced

- **Neubig:** “Advanced Discriminative Learning”—MaxEnt w/ derivatives, SGD, SVMs, regularization
  http://www.phontron.com/slides/nlp-programming-en-06-discriminative.pdf

- **Manning:** generative vs. discriminative, MaxEnt likelihood function and derivatives
  http://www.stanford.edu/class/cs224n/handouts/MaxentTutorial-16x9-MEMMs-Smoothing.pdf, slides 3–20

- **Daumé:** linear models
  http://www.ciml.info/dl/v0_8/ciml-v0_8-ch06.pdf

- **Smith:** A variety of loss functions for text classification
  http://courses.cs.washington.edu/courses/cse517/16wi/slides/tc-intro-slides.pdf & http://courses.cs.washington.edu/courses/cse517/16wi/slides/tc-advanced-slides.pdf