

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

Nathan Schneider  
(some slides borrowed from Chris Dyer)  
ENLP | 23 February 2021

# Outline

- Words, probabilities → Features, weights

*previous lecture*

- Geometric view: decision boundary

- Perceptron

*this lecture*

- Generative vs. Discriminative

- More discriminative models: Logistic regression/MaxEnt; SVM

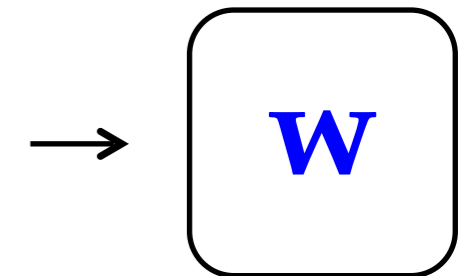
- Loss functions, optimization

- Regularization; sparsity

# Perceptron Learner

$\mathbf{X} \rightarrow$   
 $\mathbf{Y} \rightarrow$

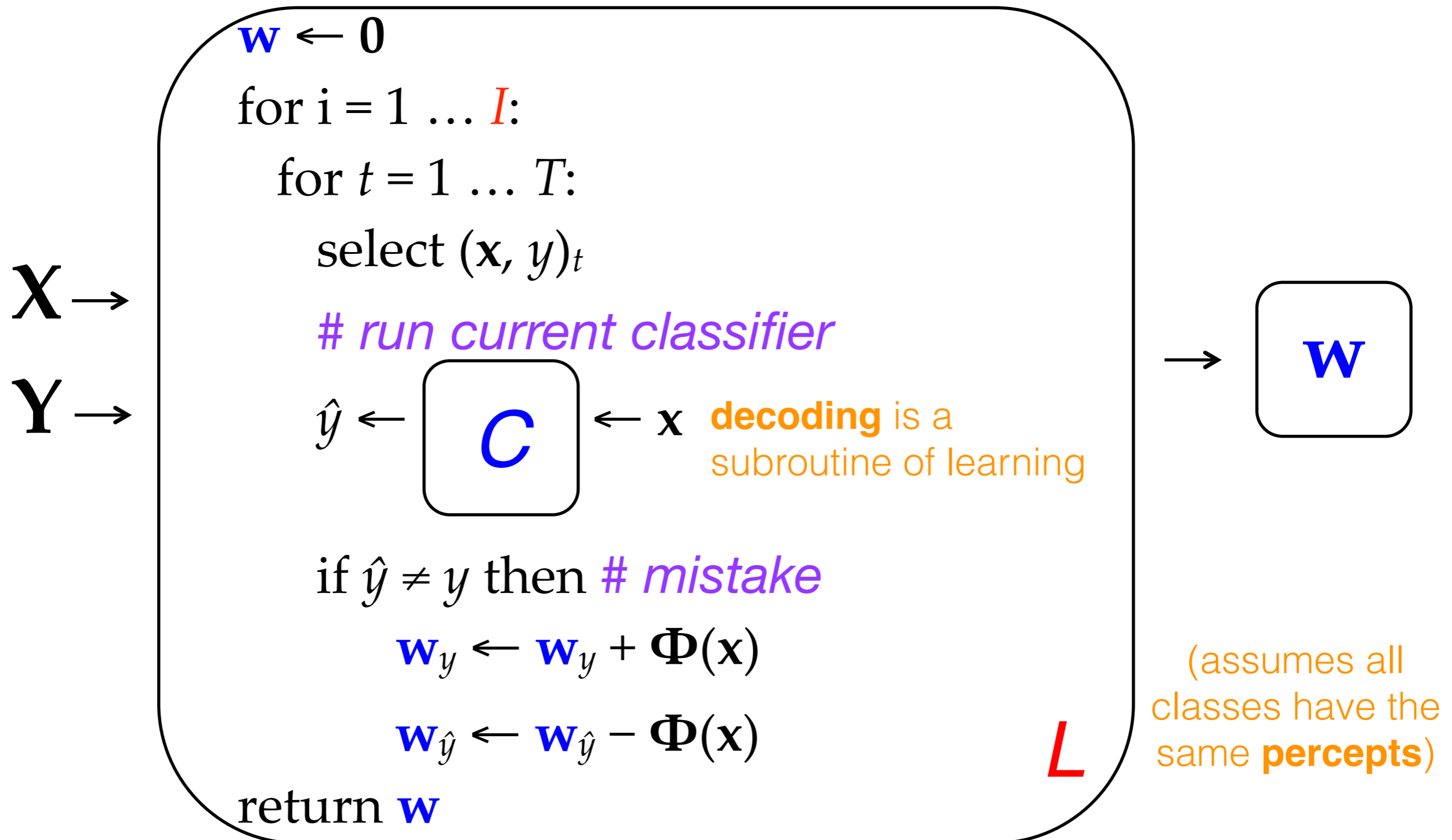
```
 $\mathbf{w} \leftarrow \mathbf{0}$   
for  $i = 1 \dots I$ :  
  for  $t = 1 \dots T$ :  
    select  $(\mathbf{x}, y)_t$   
    # run current classifier  
     $\hat{y} \leftarrow \arg \max_{y'} \mathbf{w}_{y'}^\top \Phi(\mathbf{x})$   
  
    if  $\hat{y} \neq y$  then # mistake  
       $\mathbf{w}_y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



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

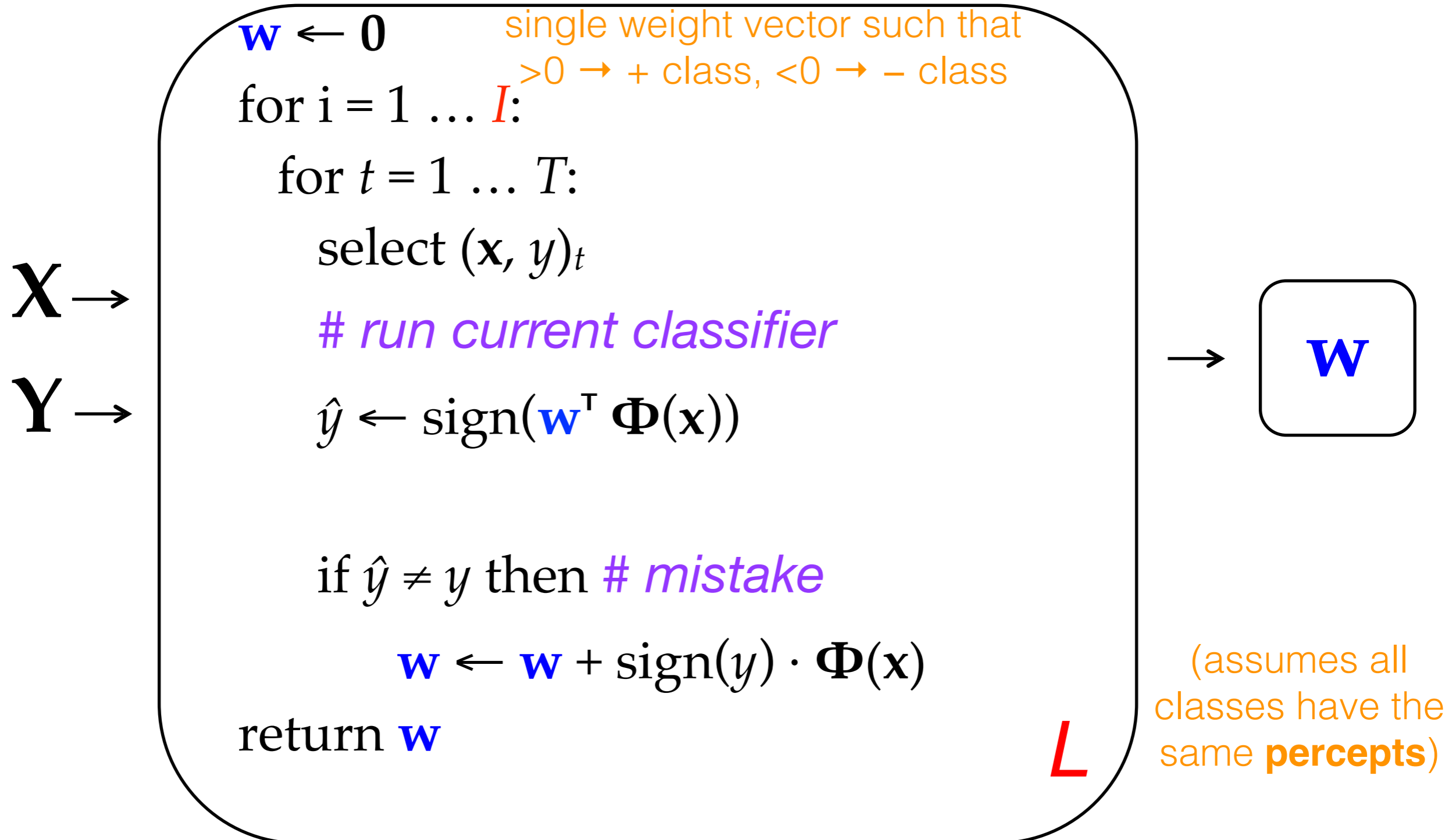
**L**

# Perceptron Learner

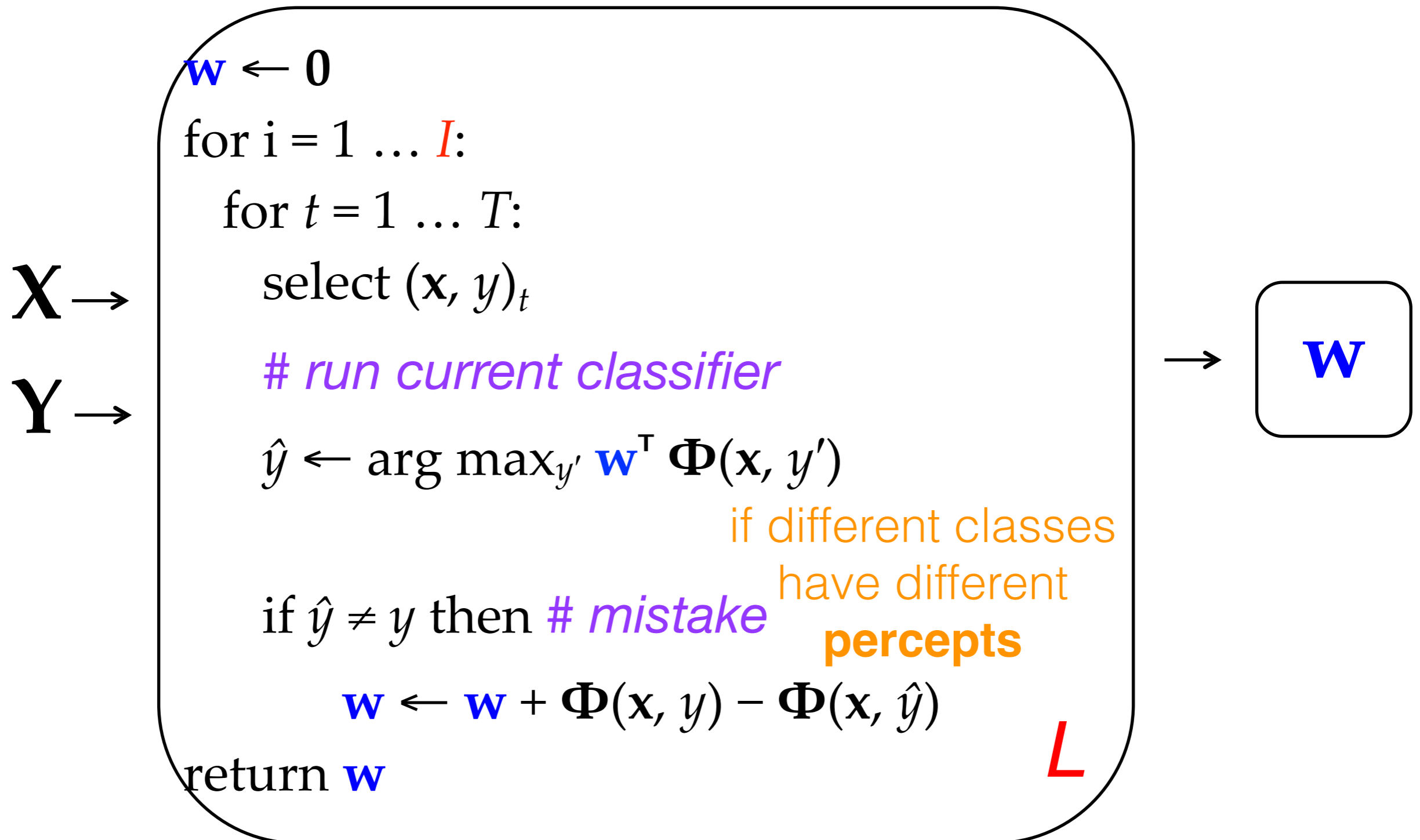


# Perceptron Learner

for **binary classification**



# Perceptron Learner



# work through example on the board

$x_1 = \text{"I thought it was great"}$

$y_1 = +$

$x_2 = \text{"not so great"}$

$y_2 = -$

$x_3 = \text{"good but not great"}$

$y_3 = +$

# 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!  $\Rightarrow$  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.



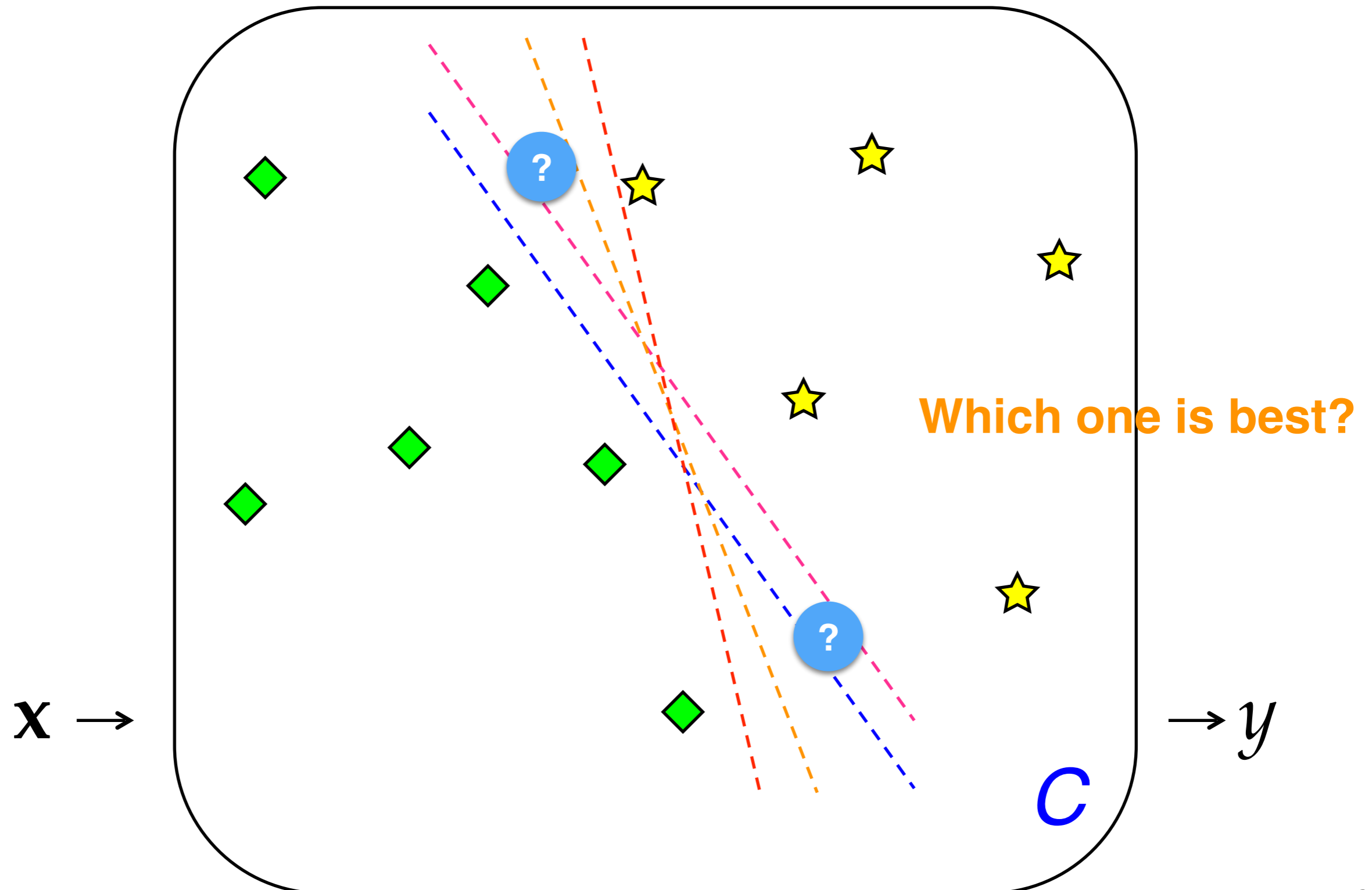
# Perceptron: Avoiding overfitting

- 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  $I - 1$ .

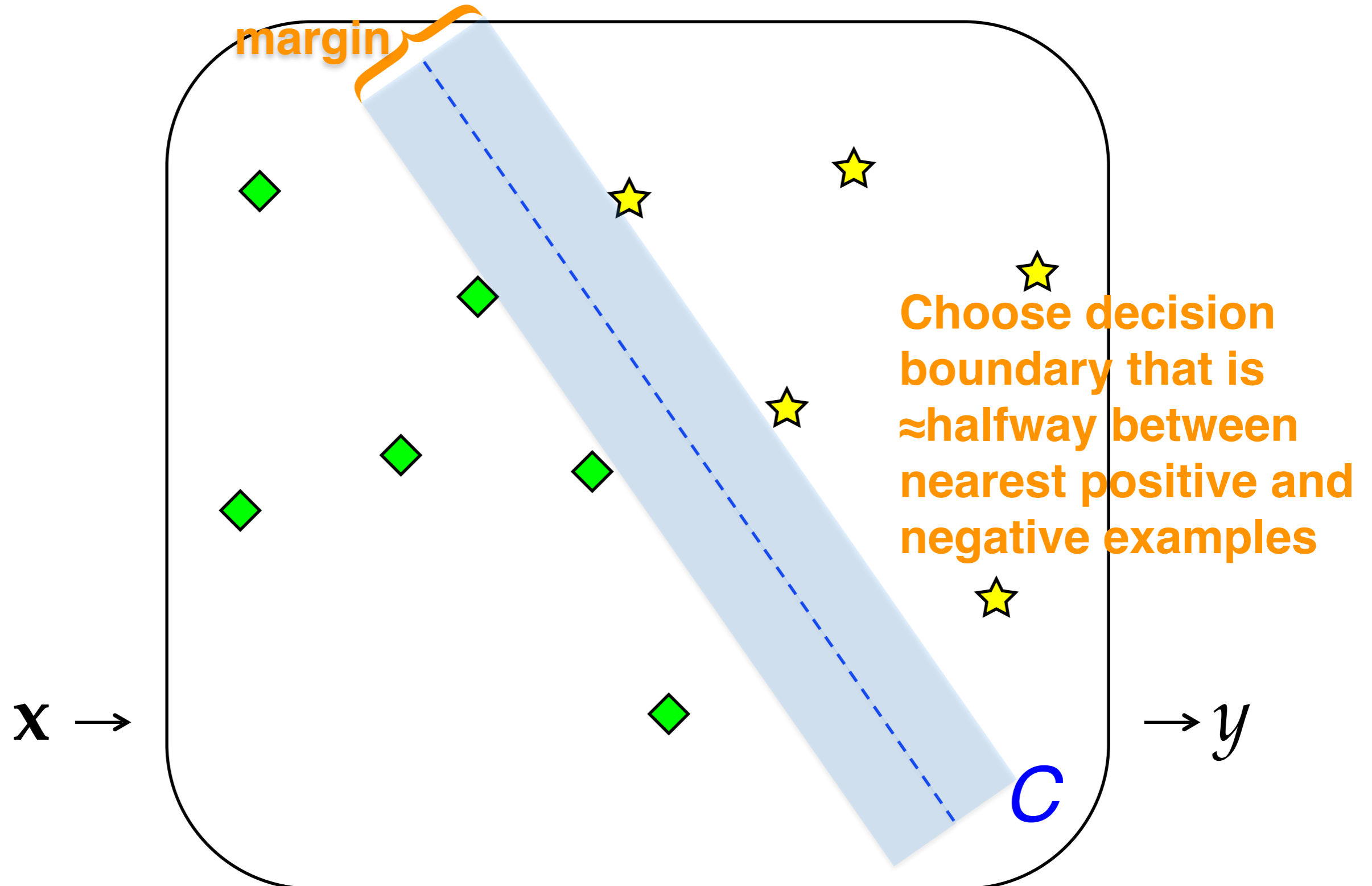
# Generative vs. Discriminative

- Naïve Bayes allows us to classify via the **joint probability** of  $\mathbf{x}$  and  $y$ :
  - $p(y | \mathbf{x}) \propto p(y) \prod_{w \in \mathbf{x}} p(w | y)$   
=  $p(y) p(\mathbf{x} | y)$  (per the independence assumptions of the model)  
=  $p(y, \mathbf{x})$  (chain rule)
  - This means the model accounts for BOTH  $\mathbf{x}$  and  $y$ . From the joint distribution  $p(\mathbf{x}, y)$  it is possible to compute  $p(\mathbf{x})$  as well as  $p(y)$ ,  $p(\mathbf{x} | y)$ , and  $p(y | \mathbf{x})$ .
- NB is called a **generative** model because it assigns probability to linguistic objects ( $\mathbf{x}$ ). 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  $\mathbf{x}$ , and cannot be used to estimate the probability of  $\mathbf{x}$  or generate  $\mathbf{x} | y$ .

# Many possible decision boundaries



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



# 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 ( $\mathbf{w}^T \Phi(\mathbf{x}, y')$ ) in the **softmax** function:
  - $\log p(y | \mathbf{x}) = \log \frac{\exp(\mathbf{w}^T \Phi(\mathbf{x}, y))}{\sum_{y'} \exp(\mathbf{w}^T \Phi(\mathbf{x}, y'))} = \mathbf{w}^T \Phi(\mathbf{x}, y) - \log \sum_{y'} \exp(\mathbf{w}^T \Phi(\mathbf{x}, y'))$ 

score can be negative; exp(score) is always positive
  - **Binary case:** Denominator = normalization (makes probabilities sum to 1).  
Sum over all classes  $\Rightarrow$  same for all numerators  $\Rightarrow$  can be ignored at classification time.  
$$\log p(y=1 | \mathbf{x}) = \log \frac{\exp(\mathbf{w}^T \Phi(\mathbf{x}, y=1))}{\exp(\mathbf{w}^T \Phi(\mathbf{x}, y=1)) + \exp(\mathbf{w}^T \Phi(\mathbf{x}, y=0))}$$
$$= \log \frac{\exp(\mathbf{w}^T \Phi(\mathbf{x}, y=1))}{\exp(\mathbf{w}^T \Phi(\mathbf{x}, y=1)) + 1} \quad (\text{fixing } \mathbf{w}^T \Phi(\mathbf{x}, y=0) = 0)$$
- 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:  $y \leftarrow \arg \max_{y'} \mathbf{w}^T \Phi(\mathbf{x}, y')$ 
  - 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**
  - $\mathbf{p}^* \leftarrow \arg \max_{\mathbf{p}} L_{\text{joint}}(\mathbf{D}; \mathbf{p})$   
 $= \arg \max_{\mathbf{p}} \sum_{(\mathbf{x}, y) \in \mathbf{D}} \log \mathbf{p}(\mathbf{x}, y) = \arg \max_{\mathbf{p}} \sum_{(\mathbf{x}, y) \in \mathbf{D}} \log (\mathbf{p}(y)\mathbf{p}(\mathbf{x} | y))$
  - 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**
  - $\mathbf{p}^* \leftarrow \arg \max_{\mathbf{p}} L_{\text{cond}}(\mathbf{D}; \mathbf{p})$   
 $= \arg \max_{\mathbf{p}} \sum_{(\mathbf{x}, y) \in \mathbf{D}} \log \mathbf{p}(y | \mathbf{x})$
  - $\mathbf{w} \leftarrow \arg \max_{\mathbf{w}} \sum_{(\mathbf{x}, y) \in \mathbf{D}} \mathbf{w}^T \Phi(\mathbf{x}, y) - \log \sum_{y'} \exp(\mathbf{w}^T \Phi(\mathbf{x}, y'))$  [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

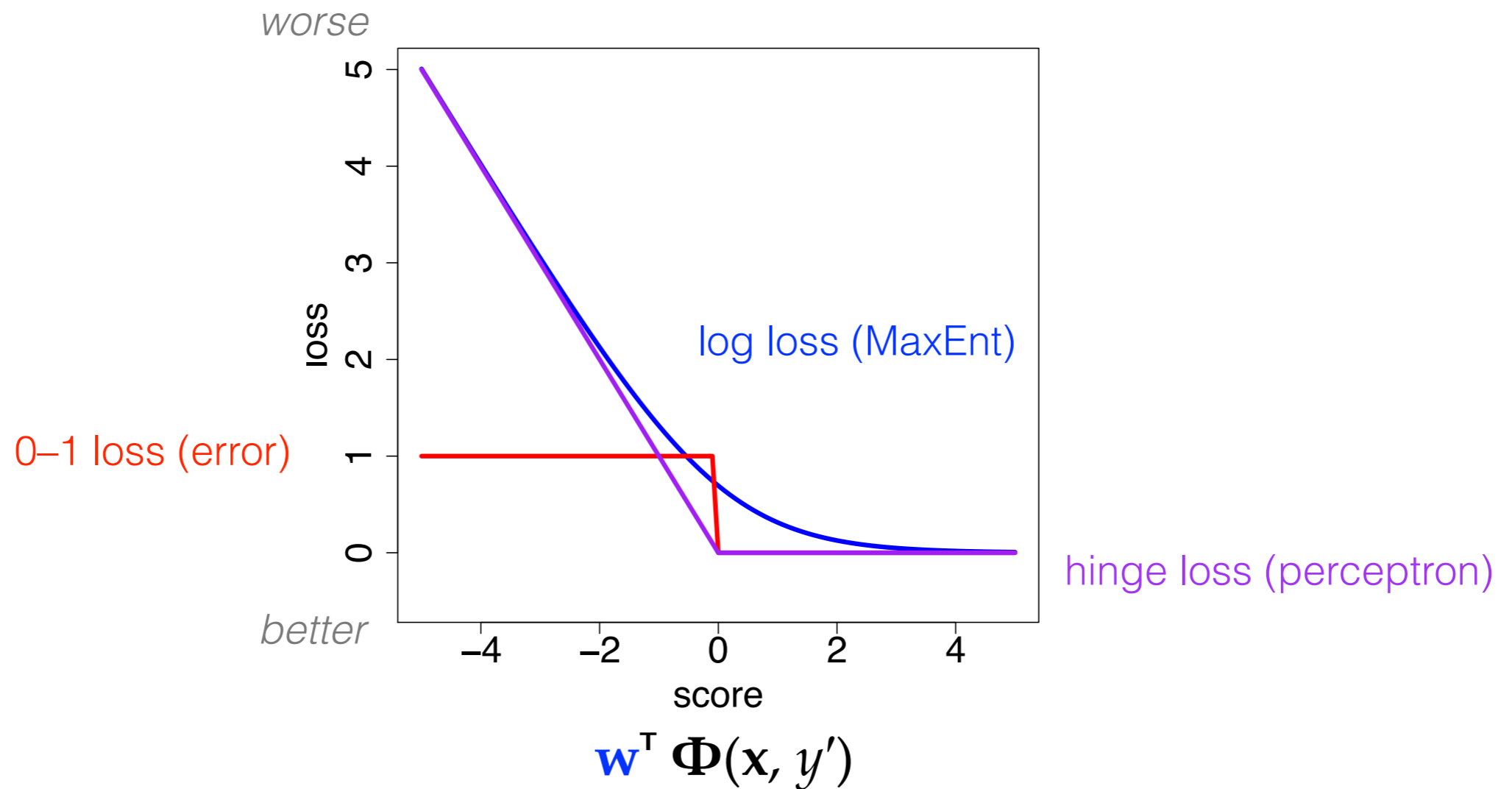


figure from Noah Smith

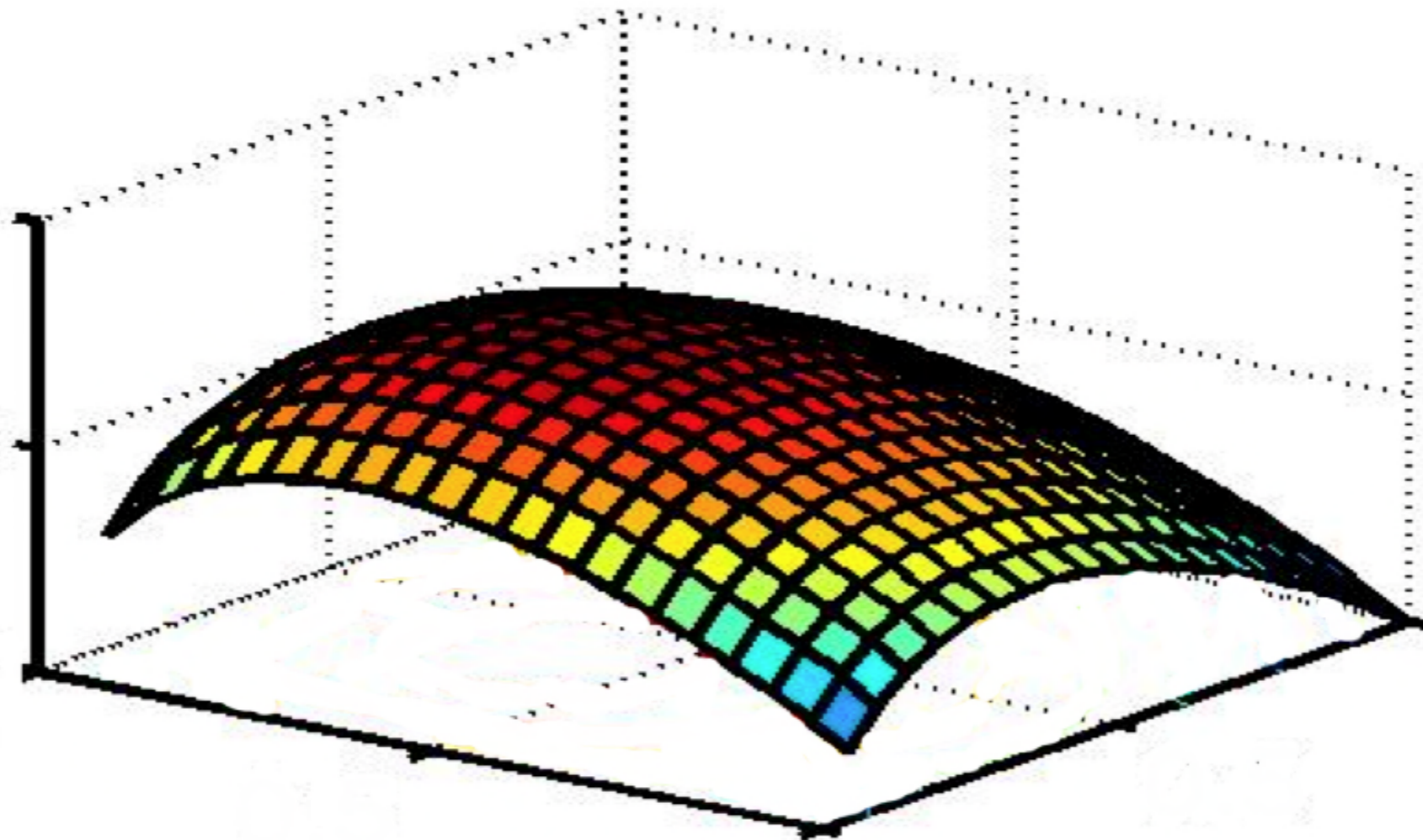
# Objectives

- 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**
  - $\mathbf{w}^* \leftarrow \arg \max_{\mathbf{w}} -\lambda R(\mathbf{w}) + \sum_{(\mathbf{x}, y) \in D} \mathbf{w}^T \Phi(\mathbf{x}, y) - \log \sum_{y'} \exp(\mathbf{w}^T \Phi(\mathbf{x}, y'))$
- **To avoid overfitting, the regularization term (“regularizer”)  $-\lambda R(\mathbf{w})$  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  $R(\mathbf{w})$  are the  $\ell_1$  norm (“Lasso”) and the  $\ell_2$  norm (“ridge”):
  - $\ell_2 = \|\mathbf{w}\|_2 = (\sum_i w_i^2)^{-1/2}$  encourages most weights to be **small in magnitude**
  - $\ell_1 = \|\mathbf{w}\|_1 = \sum_i |w_i|$  encourages most weights to be **0**
  - $\lambda$  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

- $\ell_1$  regularization is a way to promote **model sparsity**: many weights are pushed to 0.
  - A vector is sparse if (# **nonzero** parameters)  $\ll$  (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

$$\text{Classifier: } y \leftarrow \arg \max_{y'} \mathbf{w}^T \Phi(\mathbf{x}, y')$$

	<i>kind of model</i>	<i>loss function</i>	<i>learning algorithm</i>	<i>avoiding overfitting</i>
<b>Naïve Bayes</b>	Probabilistic, generative	Likelihood	Closed-form estimation	Smoothing
<b>Logistic regression (MaxEnt)</b>	Probabilistic, discriminative	Conditional likelihood	Optimization	Regularization penalty
<b>Perceptron</b>	Non-probabilistic, discriminative	Hinge	Optimization	Averaging; Early stopping
<b>SVM (linear kernel)</b>	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  $\Rightarrow$  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**  $\Rightarrow$  training is cheaper, but lower accuracy.  
**Discriminative**  $\Rightarrow$  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<sup>light</sup>
- Later in the course, we'll also see **nonlinear** models (neural networks)



# 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](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](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](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>

# Evaluating Multiclass Classifiers and Retrieval Algorithms

# Accuracy

- Assume we are disambiguating word senses such that every token has 1 gold sense label.
- The classifier predicts 1 label for each token in the test set.
- Thus, every test set token has a predicted label (*pred*) and a gold label (*gold*).
- The **accuracy** of our classifier is just the % of tokens for which the predicted label matched the gold label:  $\#_{pred=gold} / \#_{tokens}$

# Precision and Recall

- To measure the classifier with respect to a certain label  $y$ , and there are  $>2$ , we distinguish precision and recall:
  - **precision** = proportion of times the label was predicted and that prediction matched the gold:  $\#_{pred=gold=y} / \#_{pred=y}$
  - **recall** = proportion of times the label was in the gold standard and was recovered correctly by the classifier:  
 $\#_{pred=gold=y} / \#_{gold=y}$
- The harmonic mean of precision and recall, called **F<sub>1</sub>-score**, balances between the two.  
 $F_1 = 2 * precision * recall / (precision + recall)$

# Evaluating Retrieval Systems

- Precision/Recall/F-score are also useful for evaluating retrieval systems.
- E.g., consider a system which takes a word as input and is supposed to retrieve all rhymes.
- Now, for a single input (the query), there are often many correct outputs.
- **Precision** tells us whether most of the given outputs were valid rhymes; **recall** tells us whether most of the valid rhymes in the gold standard were recovered.

# Rhymes for “hinge”

**Gold**

**System**

klinge  
minge  
vinje

binge  
cringe  
fringe  
hinge  
impinge  
infringe  
syringe  
tinge  
twinge  
unhinge

ainge

# Rhymes for “hinge”

**Gold**

**System**

False Positive  
(Type I error)

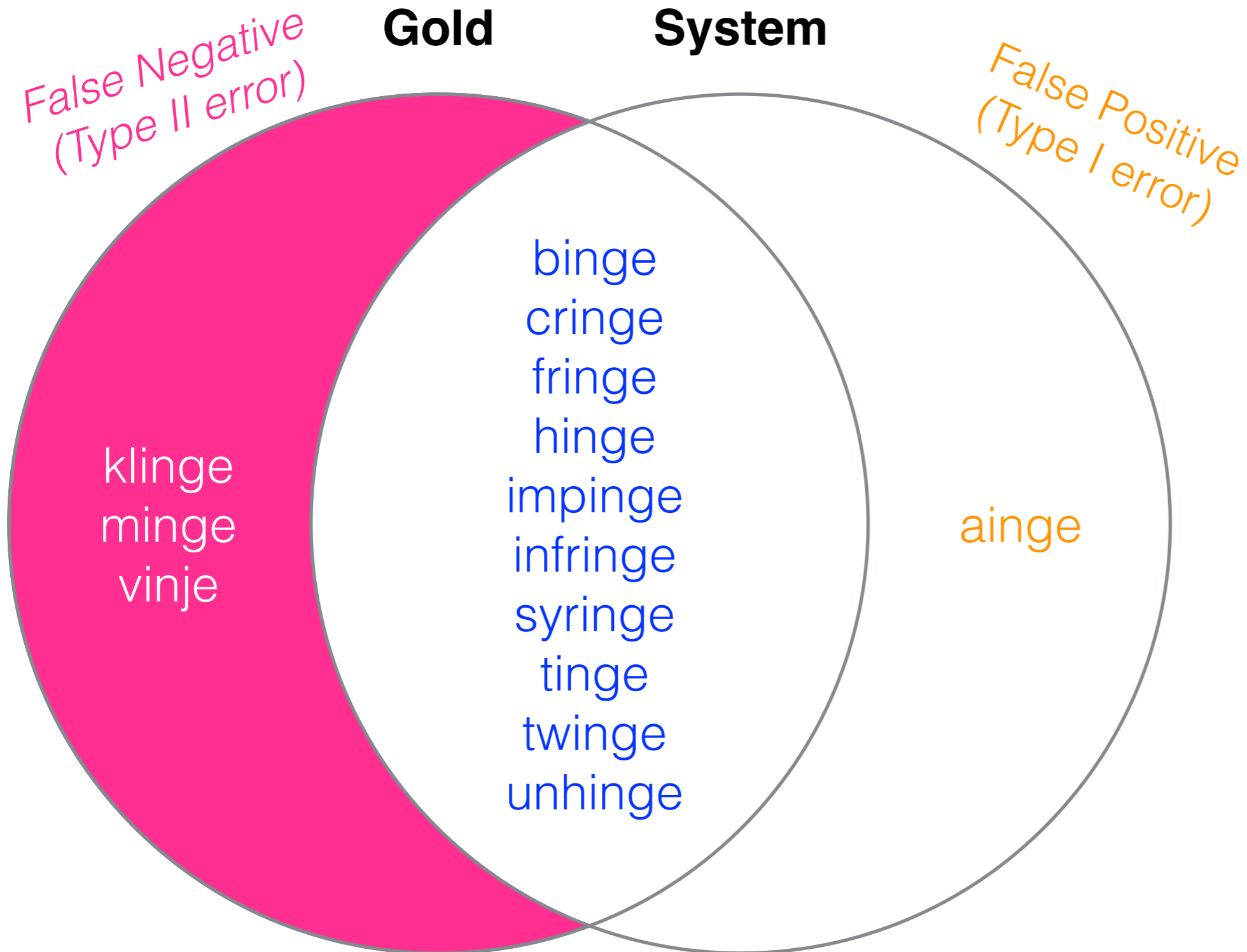
klinge  
minge  
vinje

binge  
cringe  
fringe  
hinge  
impinge  
infringe  
syringe  
tinge  
twinge  
unhinge

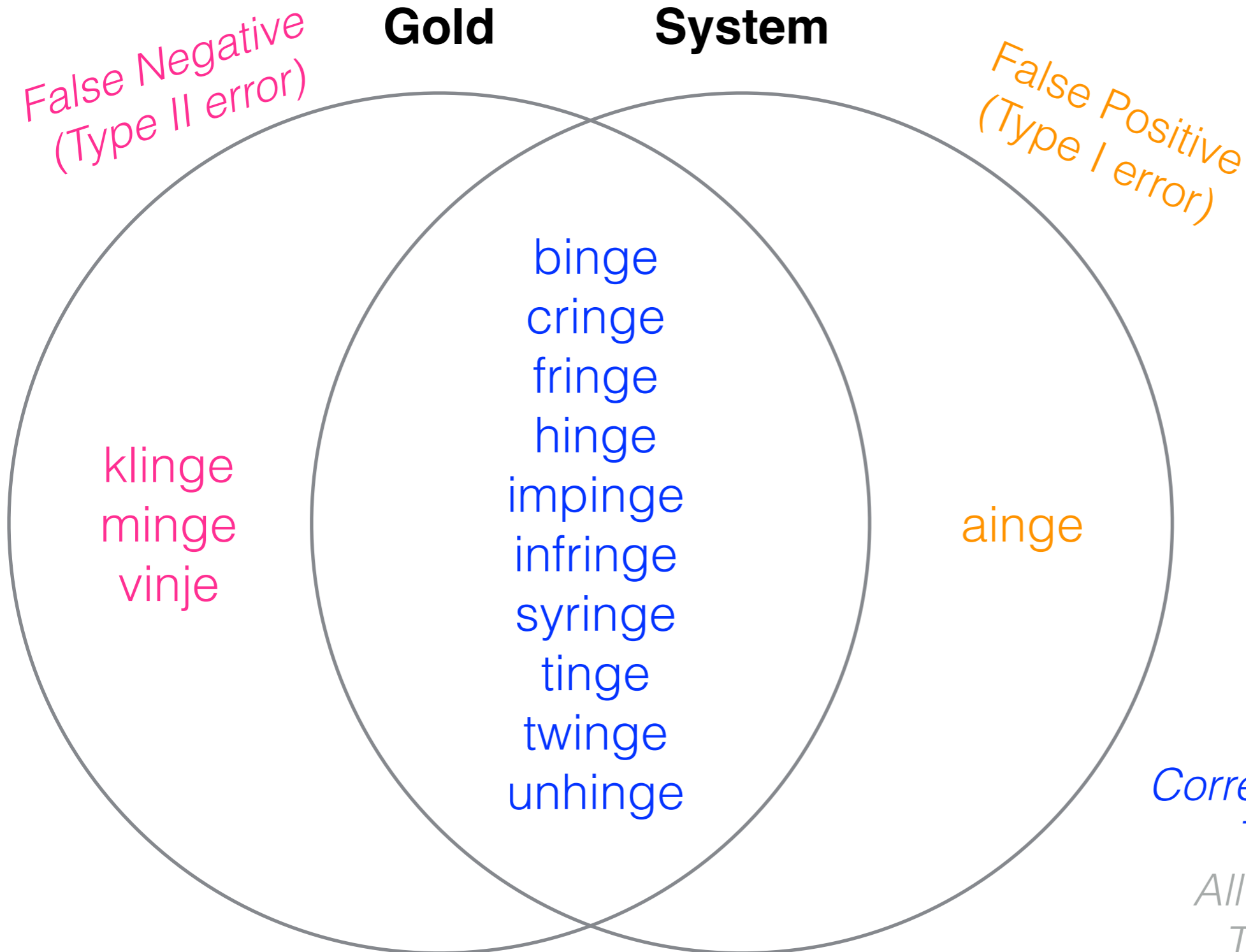
ainge



# Rhymes for “hinge”



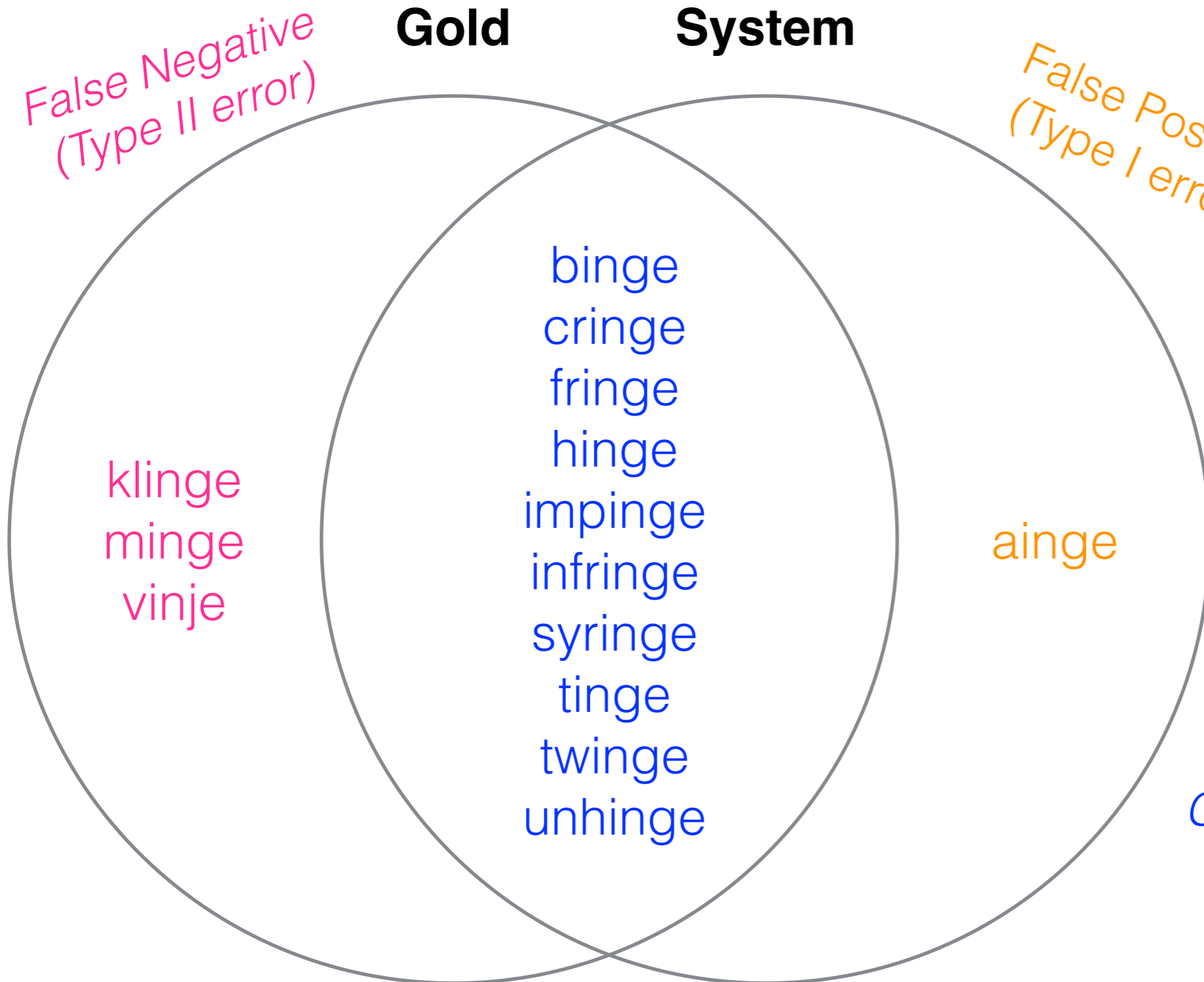
# Rhymes for “hinge”



	Sys=Y	Sys=N
Gold=Y	10	3
Gold=N	1	(large)

*Correctly predicted = True Positive*  
*All other words = True Negative*

# Precision & Recall



False Negative  
(Type II error)

False Positive  
(Type I error)

	Sys=Y	Sys=N
Gold=Y	10	3
Gold=N	1	(large)

**Precision = TP/(TP+FP)**  
= 10/11 = 91%

**Recall = TP/(TP+FN)**  
= 10/13 = 77%

**F<sub>1</sub> = 2·P·R/(P+R) = 83%**

*Correctly predicted = True Positive*  
*All other words = True Negative*