NLP

Introduction to NLP

511.

Classification

Classification

- Assigning documents or sentences to predefined categories
  - topics, languages, users …
- *Input*:
  - a document (or sentence) d
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
- *Output*: a predicted class $c \in C$

Variants of the Problem Formulation

- Binary categorization: only two categories
  - Retrieval: {relevant-doc, irrelevant-doc}
  - Spam filtering: {spam, non-spam}
  - Opinion: {positive, negative}
- K-category categorization: more than two categories
  - Topic categorization: {sports, science, travel, business,…}
  - Word sense disambiguation:{bar1, bar2, bar3, …}
- Hierarchical vs. flat
- Overlapping (soft) vs non-overlapping (hard)

Hierarchical Classification

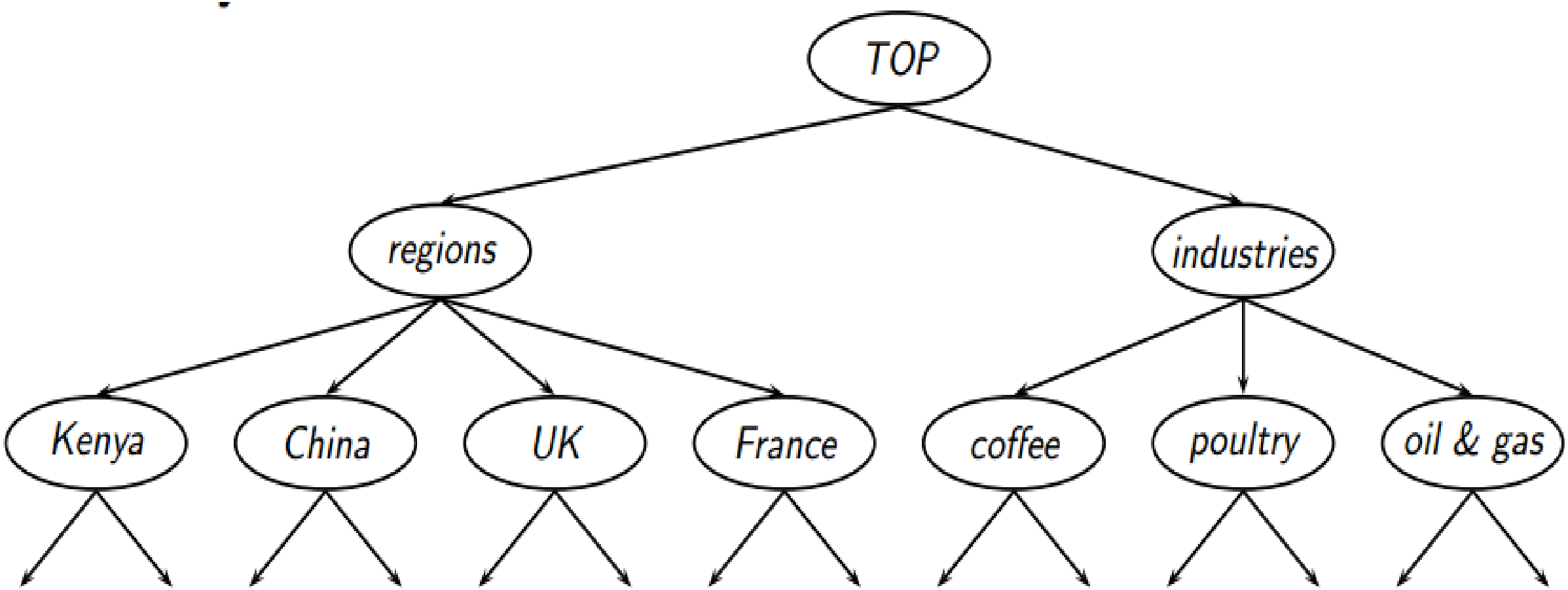


Image from Schütze & Krisnawati

Components of a Classification System

1. A **feature representation** of the input. For each input observation $x^{(i)}$, this will be a vector of features $[x_1, x_2, \ldots, x_n]$. We will generally refer to feature i for input $x^{(j)}$ as $x_i^{(j)}$, sometimes simplified as x_i, but we will also see the notation f_i, $f_i(x)$, or, for multiclass classification, $f_i(c,x)$.
2. A classification function that computes $\hat{y}$, the estimated class, via $p(y|x)$. In the next section we will introduce the **sigmoid** and **softmax** tools for classification.
3. An objective function for learning, usually involving minimizing error on training examples. We will introduce the **cross-entropy loss function**
4. An algorithm for optimizing the objective function. We introduce the **stochastic gradient descent** algorithm.

Mathematical Formulation

- Input: a numeric representation $\boldsymbol{x}$ of text d
- Output: scores $\Psi(\boldsymbol{x}, y; \boldsymbol{\theta}) \in \mathbb{R}, \forall y \in \mathcal{Y}$

Classification

$$\hat{y} = \arg\max_{y' \in \mathcal{Y}} \Psi(\boldsymbol{x}, y'; \boldsymbol{\theta})$$

Mathematical Formulation

- Training set $\mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
- Development set $\mathcal{D} = \{(x^{(j)}, y^{(j)})\}_{j=1}^{M}$
- Test set $\mathcal{U} = \{(x^{(l)}, y^{(l)})\}_{l=1}^{L}$

Mathematical Formulation

Score function

$$\Psi(x, y; \theta) = \theta^\top f(x, y)$$

- How to design a feature function $f(x, y)$?
  - Bag-of-words representation
- How to learn θ?
  - Perceptron algorithm

Bag of Words Representation

- Doc 1= “information retrieval”
- Doc 2 = “computer information retrieval information”
- Doc 3 = “computer computer retrieval”

- Vocabulary: information, retrieval, computer
- Doc 1 = <1, 1, 0>
- Doc 2 = <2, 1, 1>
- Doc 3 = <0, 1, 2>

information, retrieval, computer

$$D = \begin{bmatrix} 1 & 1 & 0 \\ 2 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Question: Doc 4 = “retrieval information retrieval” ?

Mathematical Formulation

Decision function

$$\Psi(x, y) = w_y^\top f(x, \theta)$$

- x: data point
- y: label
- w_y: classification weights with respect to label y
- $f(x, \theta)$: feature function
- θ: parameter of feature function

Features for Classification

- Vector-based
  - Words: “cat”, “dog”, “great”, “horrible”, etc.
  - Meta features: document length, author name, etc.
  - Each document (or sentence) is represented as a vector in an *n*-dimensional space
    - Similar documents appear nearby in the vector space
  - (more later)

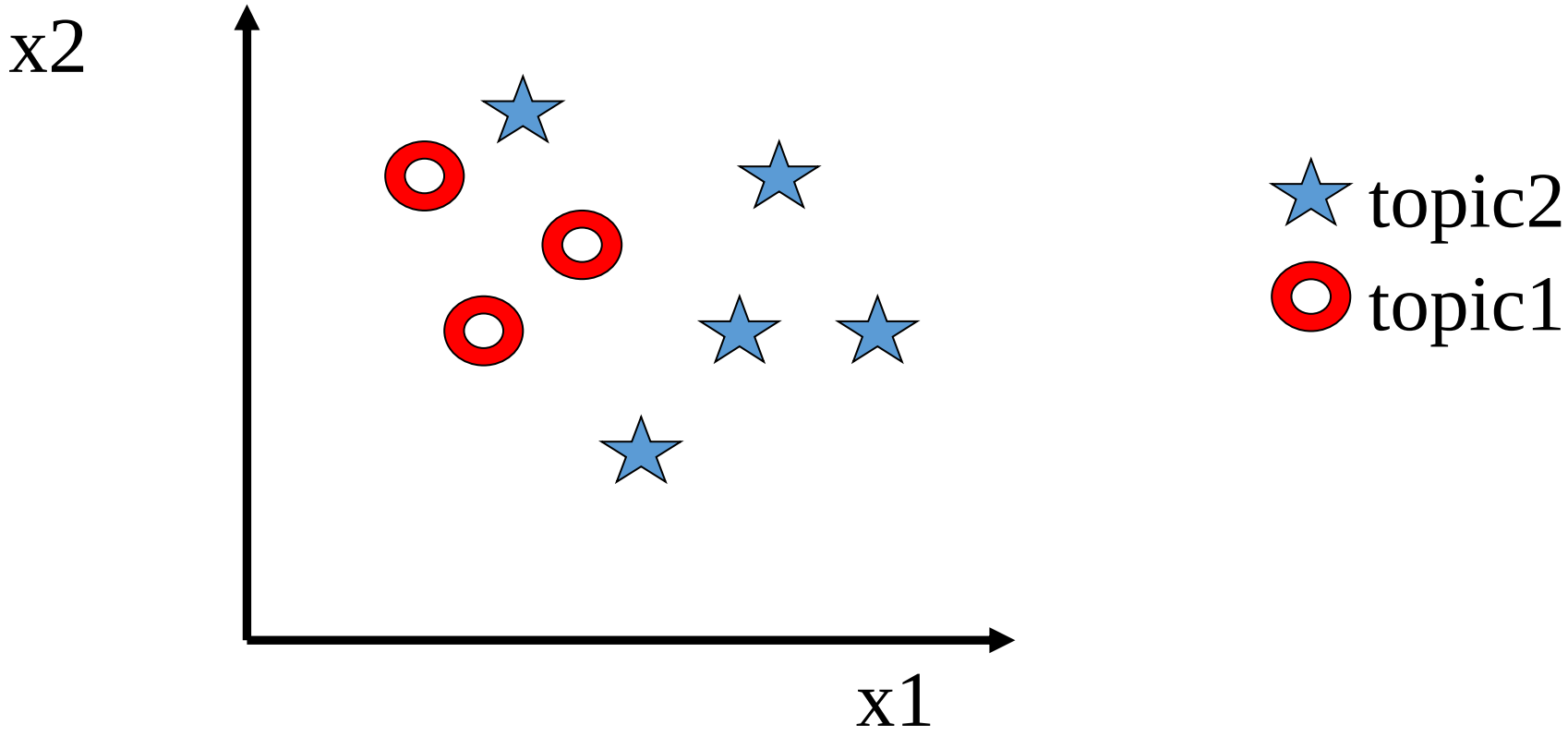
Classification in NLP

- Part of speech tagging
- Sentiment analysis
- Word sense disambiguation
- Parsing
- Optical character recognition
- Spelling correction
- Named entity classification

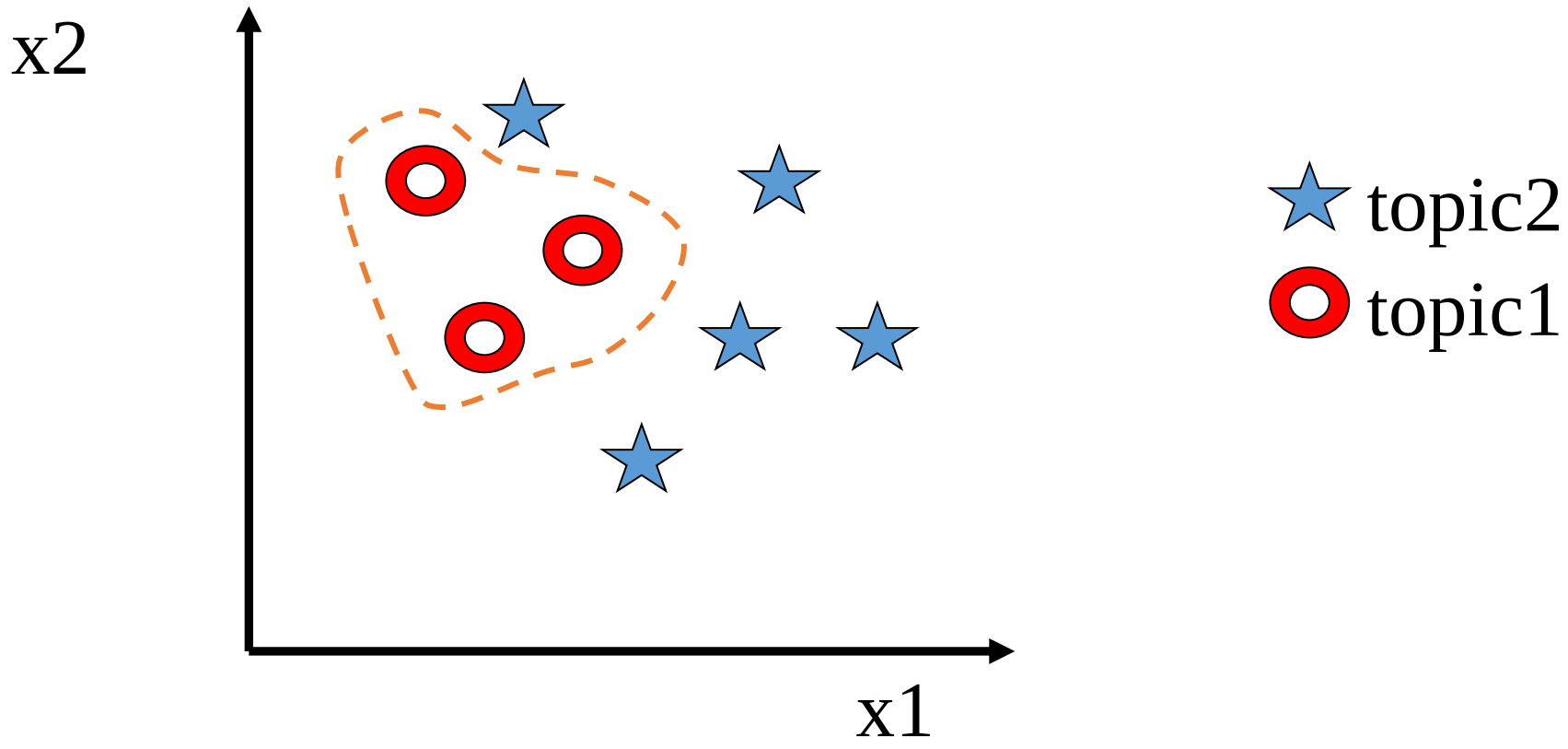
Introduction to NLP

Vector Space Classification

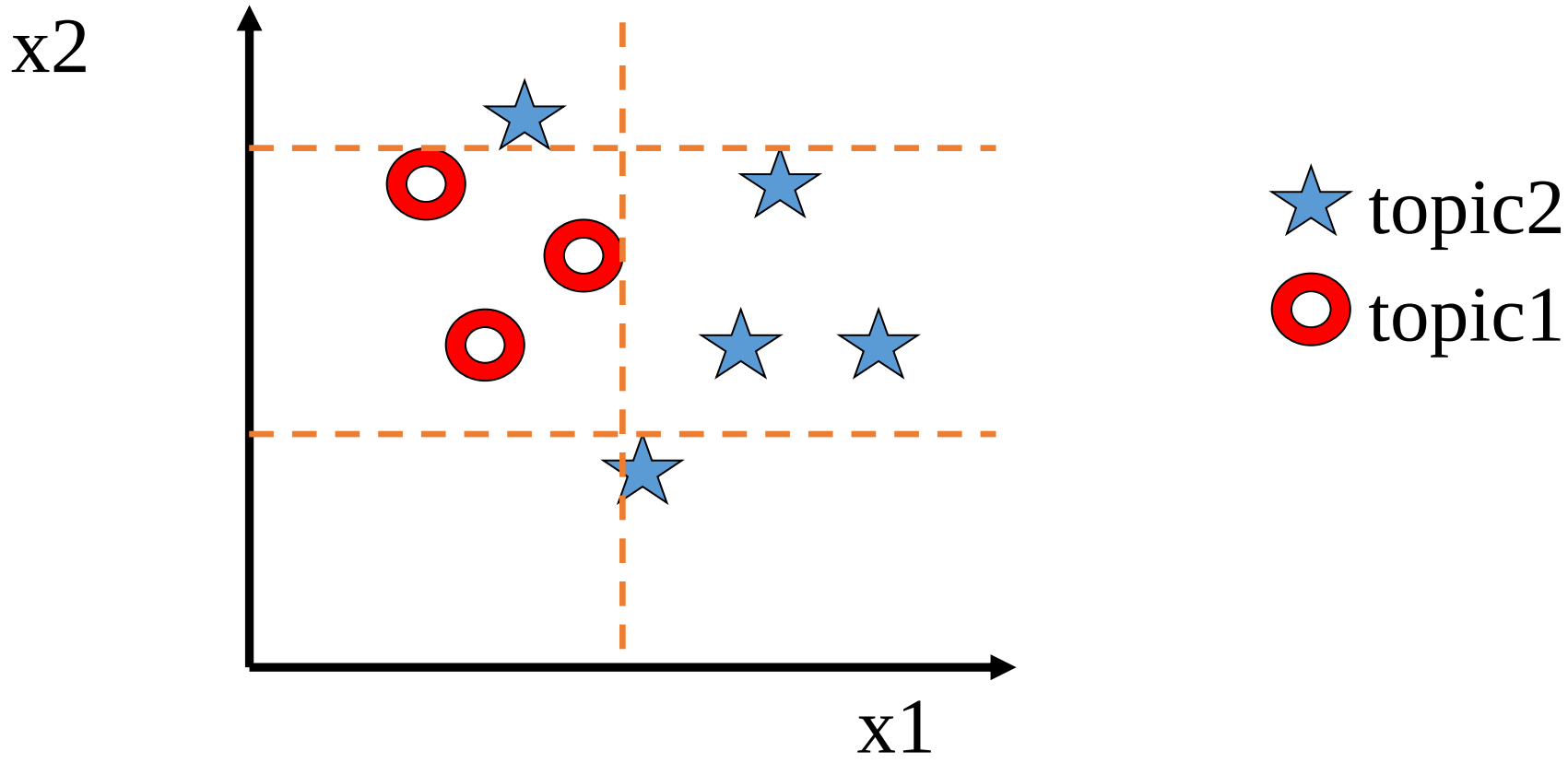
Vector Space Classification



Decision surfaces



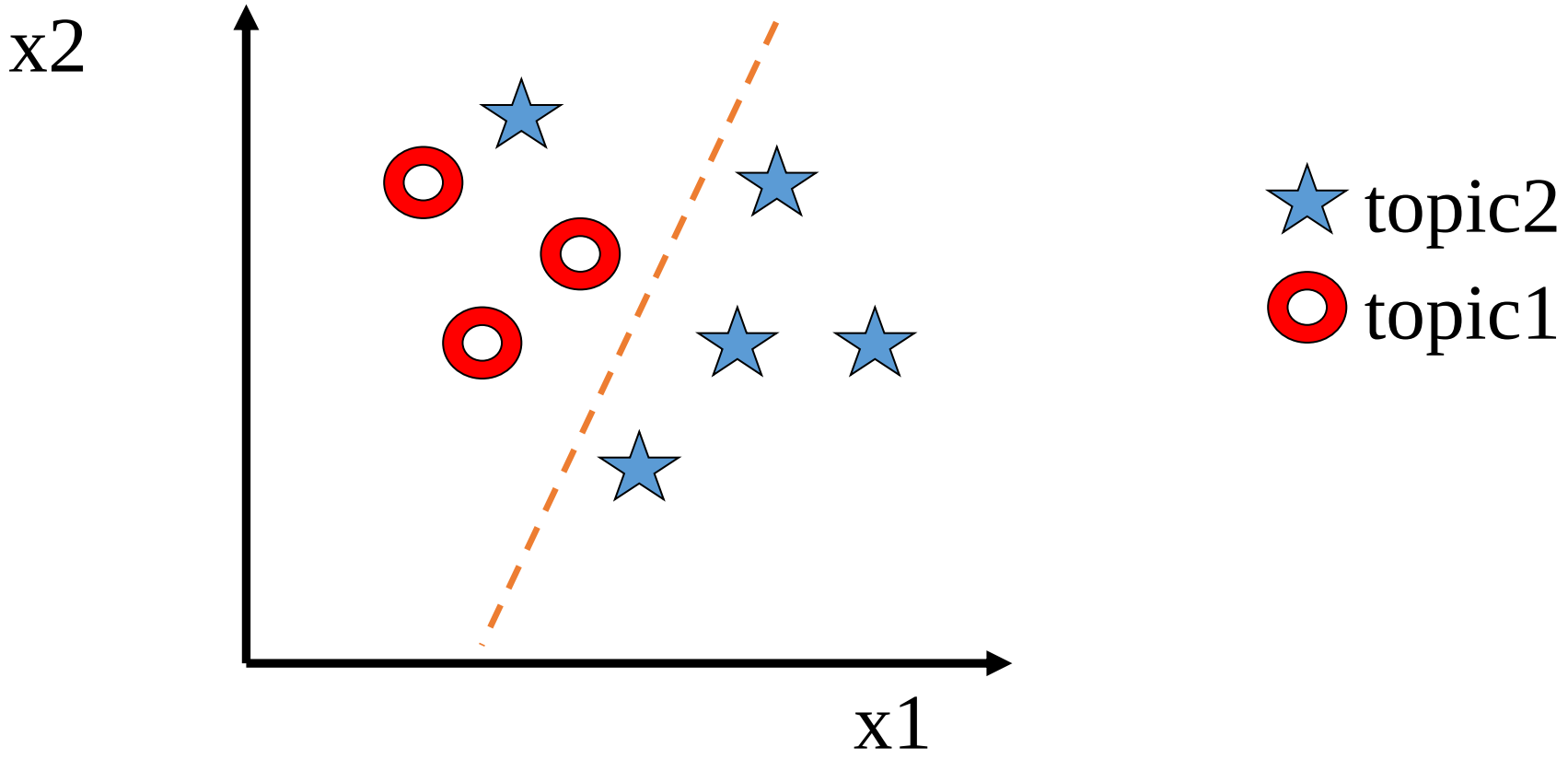
Decision trees



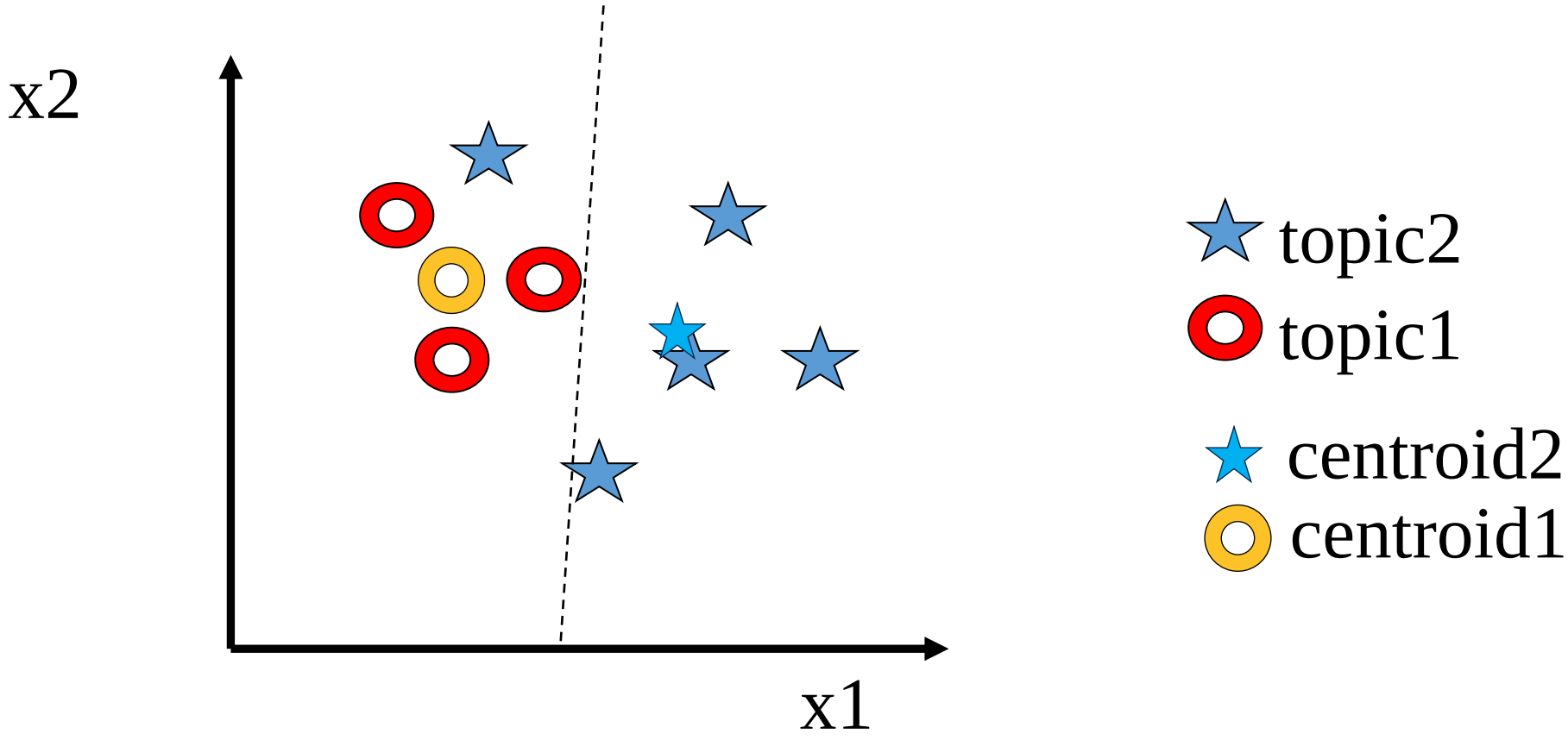
Classification Using Centroids

- Centroid
  - the point most representative of a class
- Compute centroid by finding vector average of known class members
- Decision boundary is a line that is equidistant from two centroids.
- New document on one side of the goes in one class; new document on the other side goes in the other.

Linear boundary



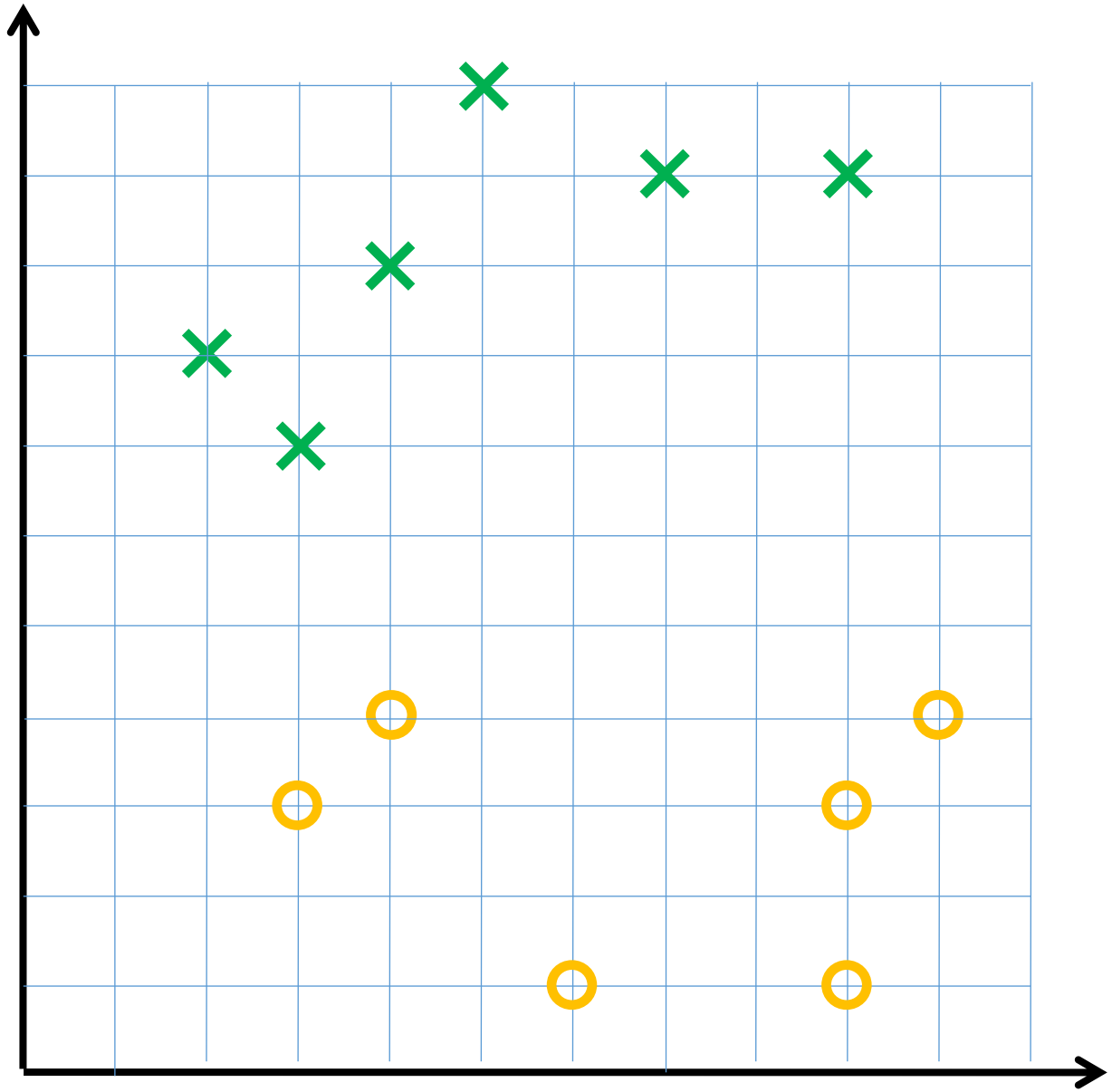
Classification Using Centroids



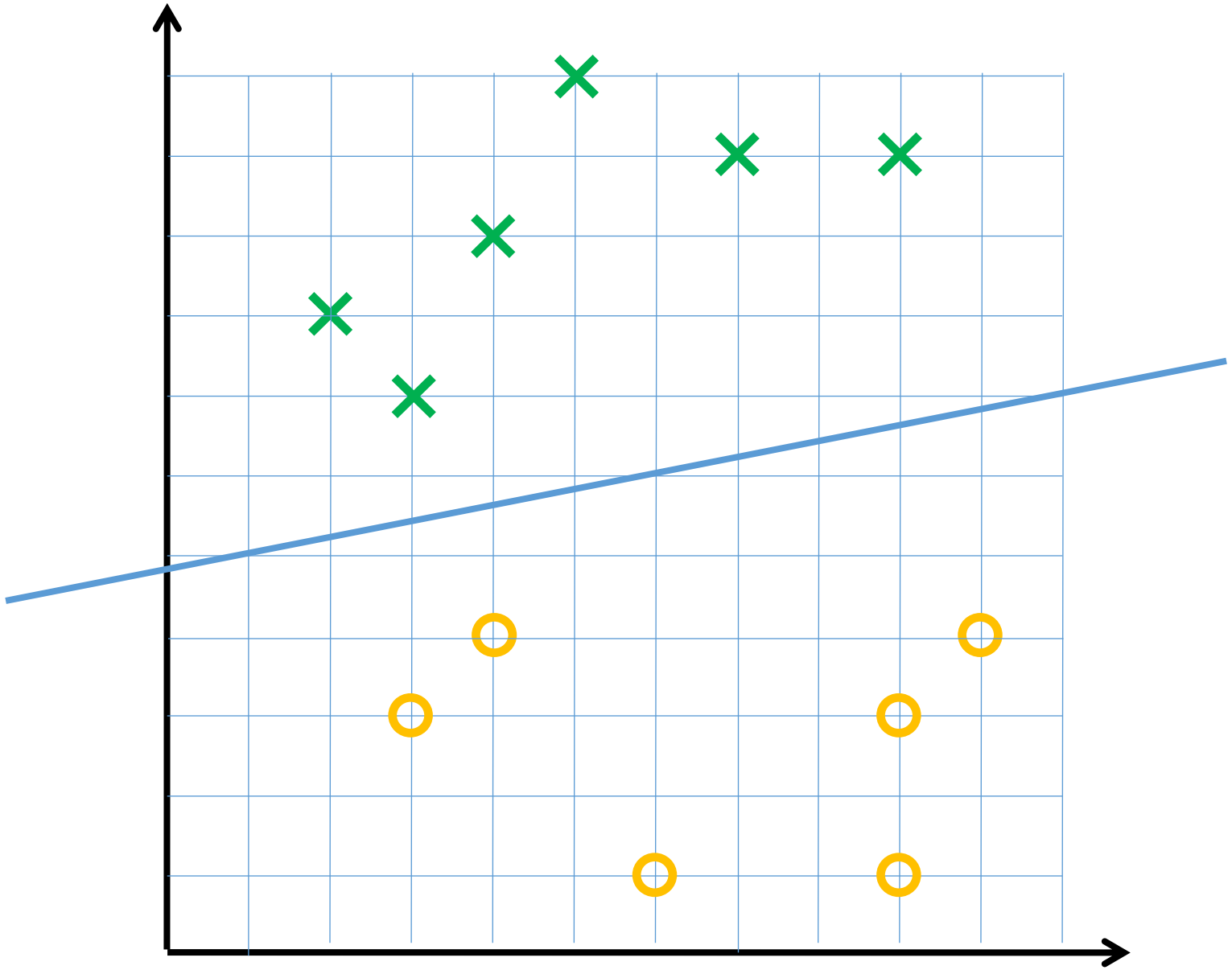
Introduction to NLP

Linear Classifiers

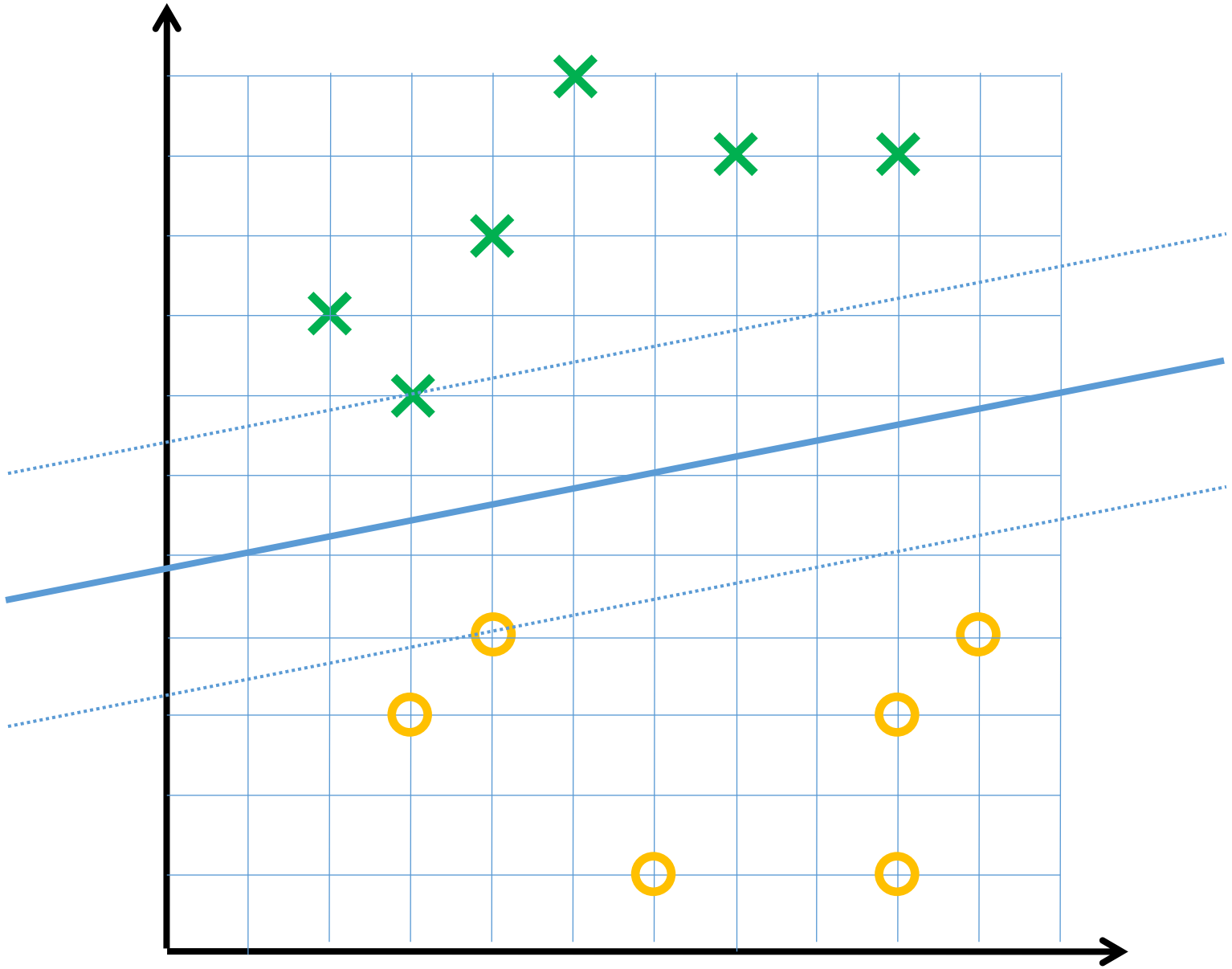
Decision Boundary



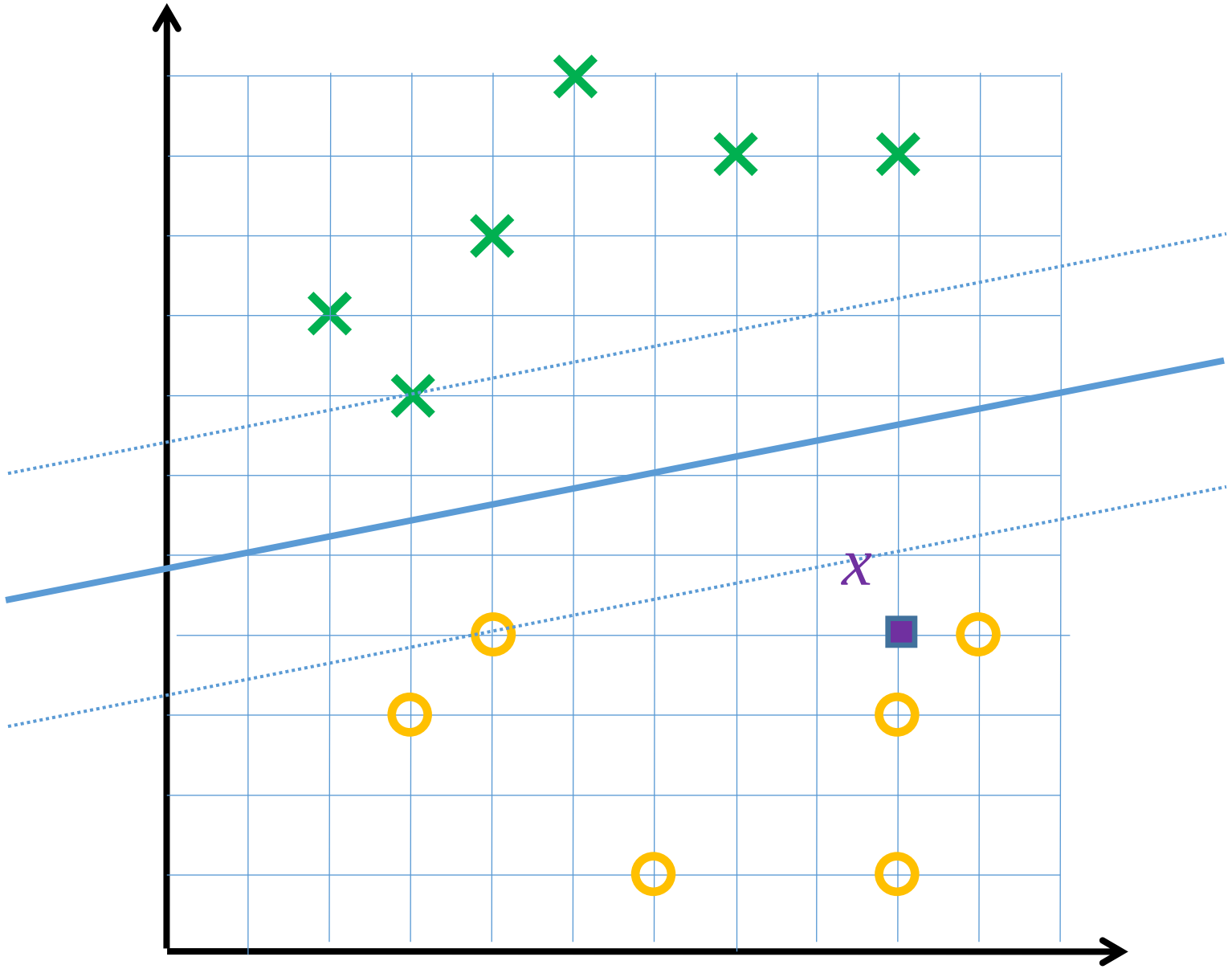
Decision Boundary



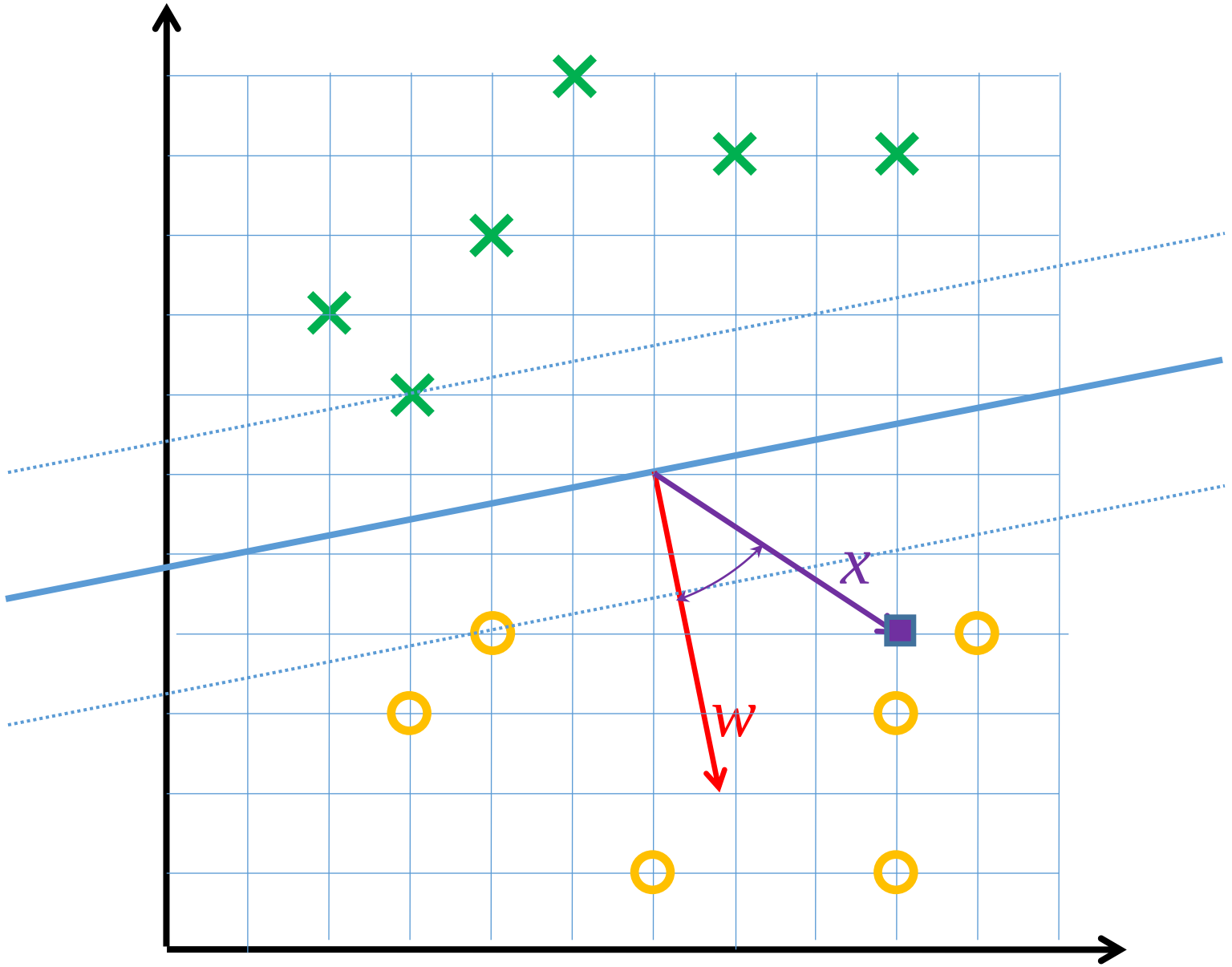
Decision Boundary



Decision Boundary



Decision Boundary



Linear Separators

- Two-dimensional line:

  $w_1x_1+w_2x_2=b$ is the linear separator

  $w_1x_1+w_2x_2>b$ for the positive class

- In *n*-dimensional spaces:

$$\vec{w}^T\vec{x} = \sum_{i=1}^{n} w_i x_i = w_1x_1 + w_2x_2 + \cdots + wnx_{\mathrm{n}}$$

- One can also add $w_0=1$, $x_0=b$ (constant)
- *w* is the weight vector
- *x* is the feature vector

Example

$$\vec{w}^T\vec{x}=b$$

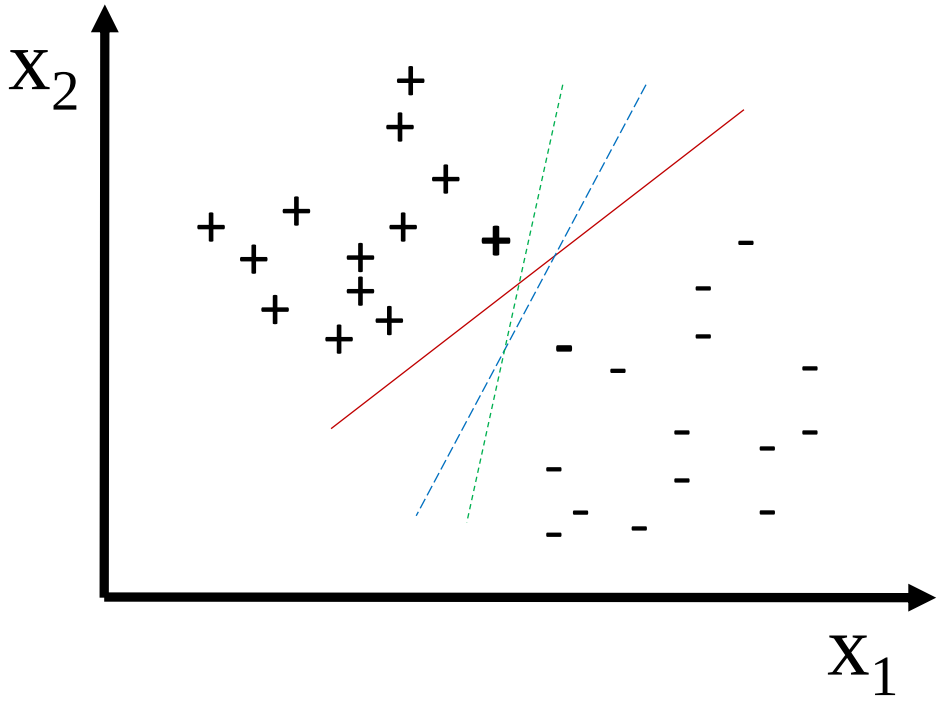
- Bias b=0 (in this example)
- Sentence is “A D E H”
- Its score will be

  0.6*1+0.4*1+0.4*1+(-0.5)*1 = 0.9>0

w_i	x_i	w_i	x_i
0.6	A	-0.7	G
0.5	B	-0.5	H
0.5	C	-0.3	I
0.4	D	-0.2	J
0.4	E	-0.2	K
0.3	F	-0.2	L

How to Find the Linear Boundary?

- Find the linear boundary = find $\vec{W}$
- Many methods
  - E.g., Perceptron
- Problem:
  - There are infinite number of linear boundaries if the two classes are linearly separable!
  - Maximum margin: Support Vector Machines (SVM)



General Training Idea

- Go through the training data
- Predict the class y (1 or -1)

$$\hat{y} = \arg\max_{y'} \theta^\top f(x, y')$$

- If the prediction is wrong, update ϑ

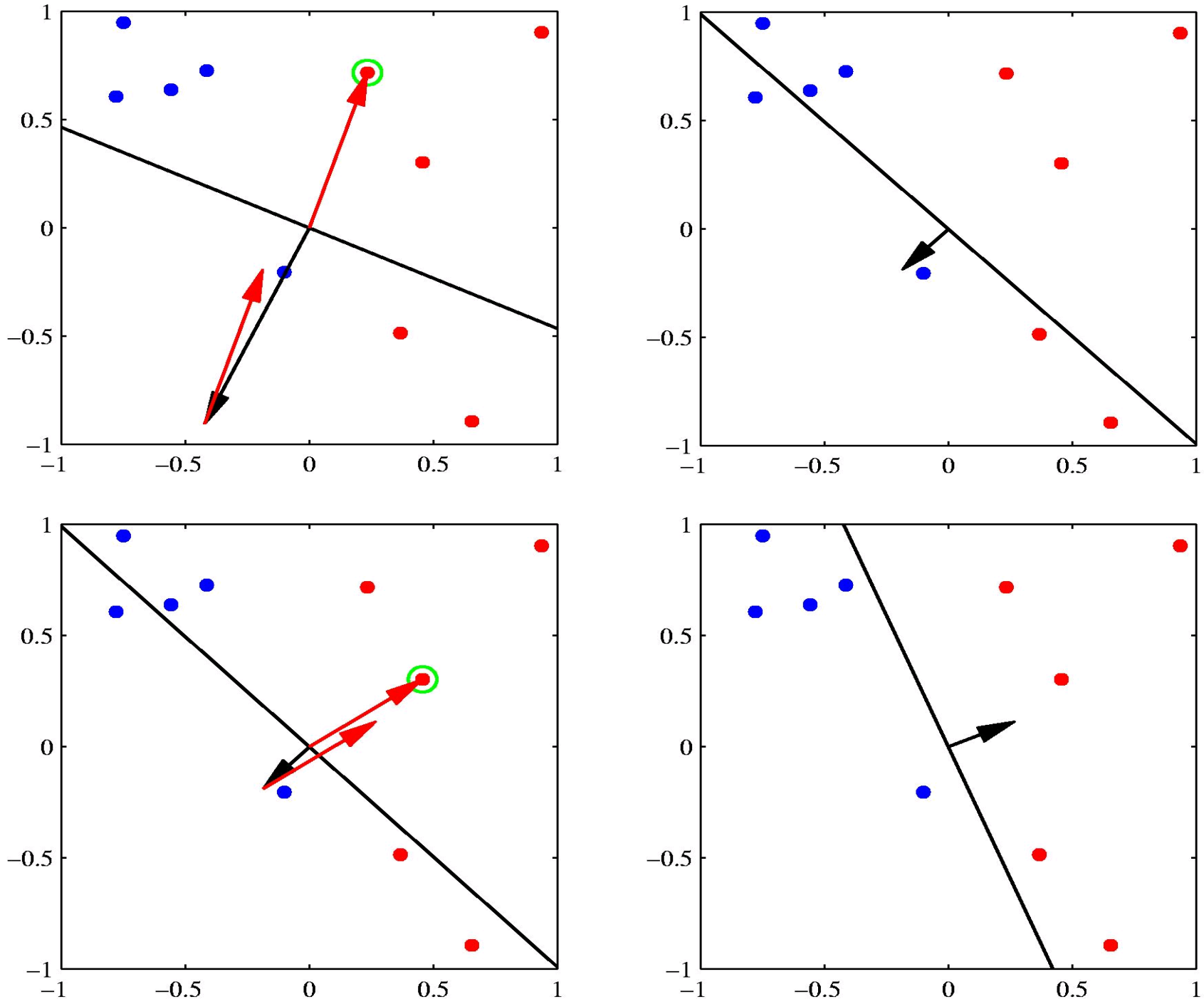
If $\hat{y} \neq y$:

$$\theta^{(\text{new})} \leftarrow \theta^{(\text{old})} + f(x, y) \quad \rightsquigarrow \quad \text{ground truth}$$
$$- f(x, \hat{y}) \quad \rightsquigarrow \quad \text{predicted label}$$

Perceptron Algorithm (preview)

Algorithm 3 Perceptron learning algorithm

procedure PERCEPTRON($\boldsymbol{x}^{(1:N)}, y^{(1:N)}$)
  $t \leftarrow 0$
  $\boldsymbol{\theta}^{(0)} \leftarrow \mathbf{0}$
  repeat
    $t \leftarrow t + 1$
    Select an instance i
    $\hat{y} \leftarrow \text{argmax}_y \, \boldsymbol{\theta}^{(t-1)} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y)$
    if $\hat{y} \neq y^{(i)}$ **then**
      $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$
    else
      $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$
  until tired
  return $\boldsymbol{\theta}^{(t)}$



[Example: Chris Bishop]

Interpolation using Features

- Linear interpolation for language modeling
  - Estimating the trigram probability P(c|ab) using P_{MLE}(c|a), P_{MLE}(c), etc.
  - Weights λ_1, λ_2, etc.
- We may want to consider other features
  - E.g., POS tags of previous words, heads, word endings, etc.
- General idea
  - Compute the conditional probability P(y|x)
  - P(y|x) = sum of weights*features
- Label Y for a given history x in X

Logistic Regression (preview)

- Model: $P_w(y|x) = \frac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$
- Inference: $\text{argmax}_y P_w(y|x)$
- Learning: gradient ascent on the discriminative log-likelihood

$$f(x,y^*) - \mathbb{E}_y[f(x,y)] = f(x,y^*) - \sum_y [P_w(y|x) f(x,y)]$$

"towards gold feature value, away from expectation of feature value"

[discriminative method]

[Greg Durrett]

Naïve Bayes (preview)

- Multinomial Naïve Bayes is a linear model

x = [1, x_1, x_2 …]

w = [log P(y), log P(w_1|y), log P(w_2|y) …]

Parametric classifiers

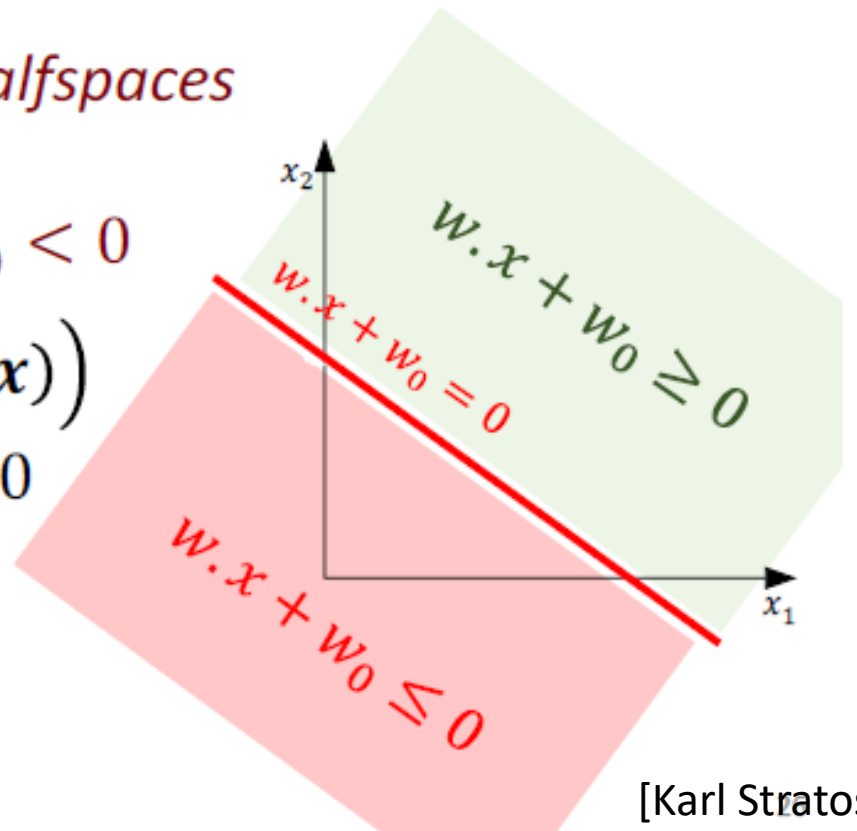
- What is the equivalent of linear regression?
  - something easy to train
  - something easy to use at test time
- $f(\boldsymbol{x}) = f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}.\boldsymbol{x} + w_0$
- $\mathcal{H} = \{f_{\boldsymbol{w}} = \boldsymbol{x} \to \boldsymbol{w}.\boldsymbol{x} + w_0 : \boldsymbol{w} \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$
- but $f(\boldsymbol{x}) \notin \{-1,1\}$! how do we get labels?
  - reasonable choice

    $\hat{y}(\boldsymbol{x}) = 1$ if $f_{\hat{\boldsymbol{w}}}(\boldsymbol{x}) \geq 0$ and $\hat{y}(\boldsymbol{x}) = -1$ otherwise
  - linear classifier: $\hat{y}(\boldsymbol{x}) = \text{sign}(\hat{\boldsymbol{w}}.\boldsymbol{x} + \hat{w}_0)$

[Karl Stratos]

Parametric classifiers

- $\mathcal{H} = \{f_{\boldsymbol{w}} = \boldsymbol{x} \to \boldsymbol{w}.\boldsymbol{x} + w_0 : \boldsymbol{w} \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$
- $\hat{y}(\boldsymbol{x}) = \text{sign}(\hat{\boldsymbol{w}}.\boldsymbol{x} + \hat{w}_0)$
- $\hat{\boldsymbol{w}}.\boldsymbol{x} + \hat{w}_0 = 0$ (linear) decision boundary or separating *hyperplane*
  - that separates $\mathbb{R}^d$ into two *halfspaces (regions)*
    $\hat{\boldsymbol{w}}.\boldsymbol{x} + \hat{w}_0 > 0$ and $\hat{\boldsymbol{w}}.\boldsymbol{x} + \hat{w}_0 < 0$
- more generally, $\hat{y}(\boldsymbol{x}) = \text{sign}\left(\hat{f}(\boldsymbol{x})\right)$
  → decision boundary is $\hat{f}(\boldsymbol{x}) = 0$



[Karl Stratos]

Classification vs Regression

- Label-values do not have meaning
  - $\mathcal{Y} = \{\text{spam}, \text{nospam}\}$ or $\mathcal{Y} = \{0,1\}$ or $\mathcal{Y} = \{-1,1\}$
- Ordering of labels does not matter (for most parts)
  - $f(x) =$ "0" when $y =$ "1" is as bad as $f(x) =$ "9" when $y =$ "1"
- Often $f(x)$ does not return labels y
  - e.g. in binary classification with $\mathcal{Y} = \{-1,1\}$ we often estimate $f: \mathcal{X} \to \mathbb{R}$ and then post process to get $\hat{y}\big(f(x)\big) = \mathbf{1}[f(x) \geq 0]$
  - mainly for computational reasons
    - remember, we need to solve $\min_{f \in \mathcal{H}} \sum_i \ell\big(f\big(x^{(i)}\big), y^{(i)}\big)$
    - discrete values → combinatorial problems → hard to solve
  - more generally $\mathcal{H} \subset \{f: \mathcal{X} \to \mathbb{R}\}$ and loss $\ell: \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$
    - compare to regression, where typically $\mathcal{H} \subset \{f: \mathcal{X} \to \mathcal{Y}\}$ and loss $\ell: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$

[Karl Stratos]

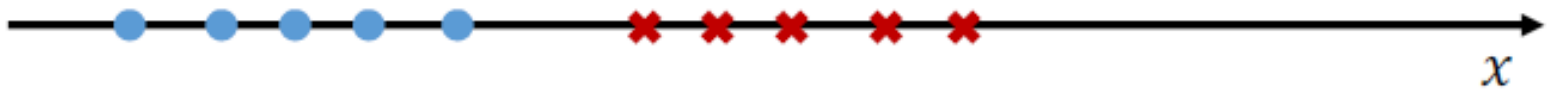
Classification as regression

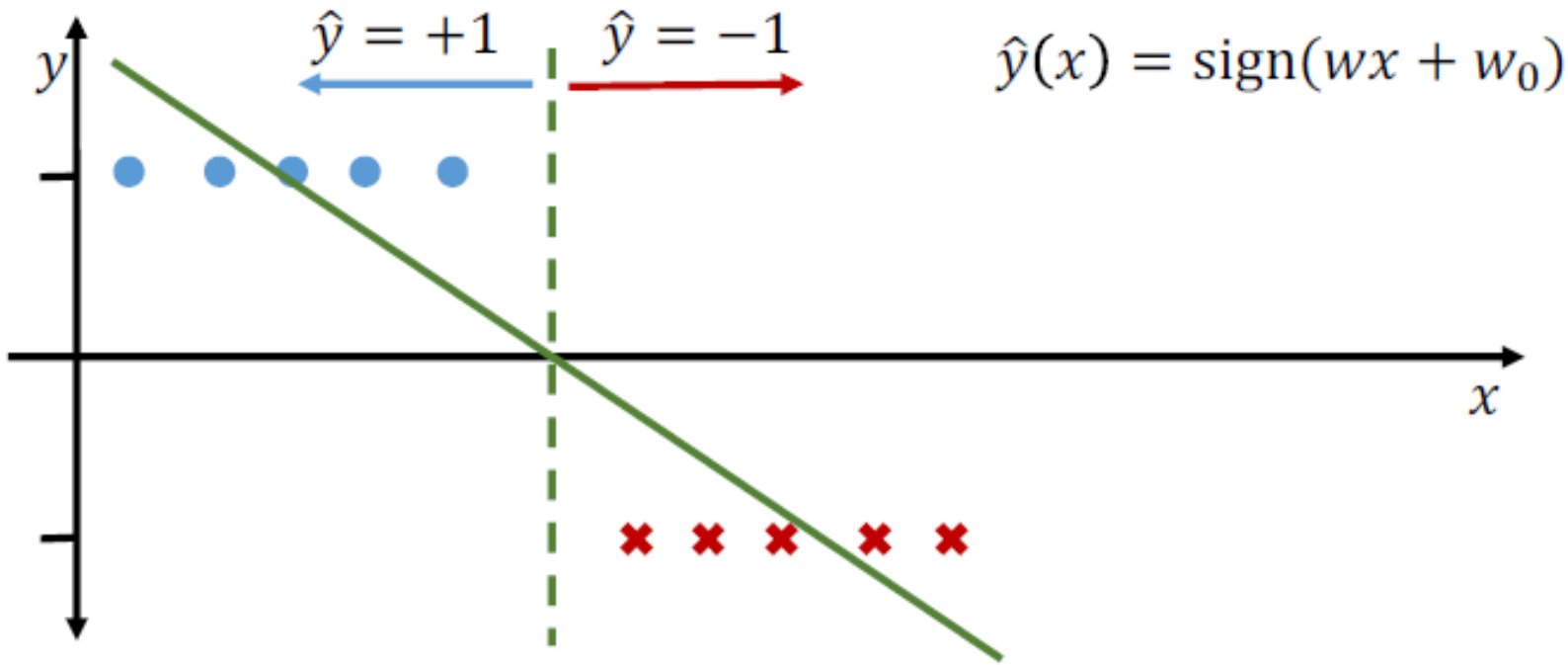
- Binary classification $\mathcal{Y} = \{-1,1\}$ and $\mathcal{X} \in \mathbb{R}^d$
- Treat it as regression with squared loss, say linear regression
  - Training data $S = \{(\boldsymbol{x}^{(i)}, y^{(i)}) : i = 1,2, \dots, N\}$
  - ERM

$$\widehat{\boldsymbol{w}}, \widehat{w}_0 = \underset{w, w_0}{\operatorname{argmin}} \sum_i \left(\boldsymbol{w}.\boldsymbol{x}^{(i)} + w_0 - y^{(i)}\right)^2$$

[Karl Stratos]

Classification as regression

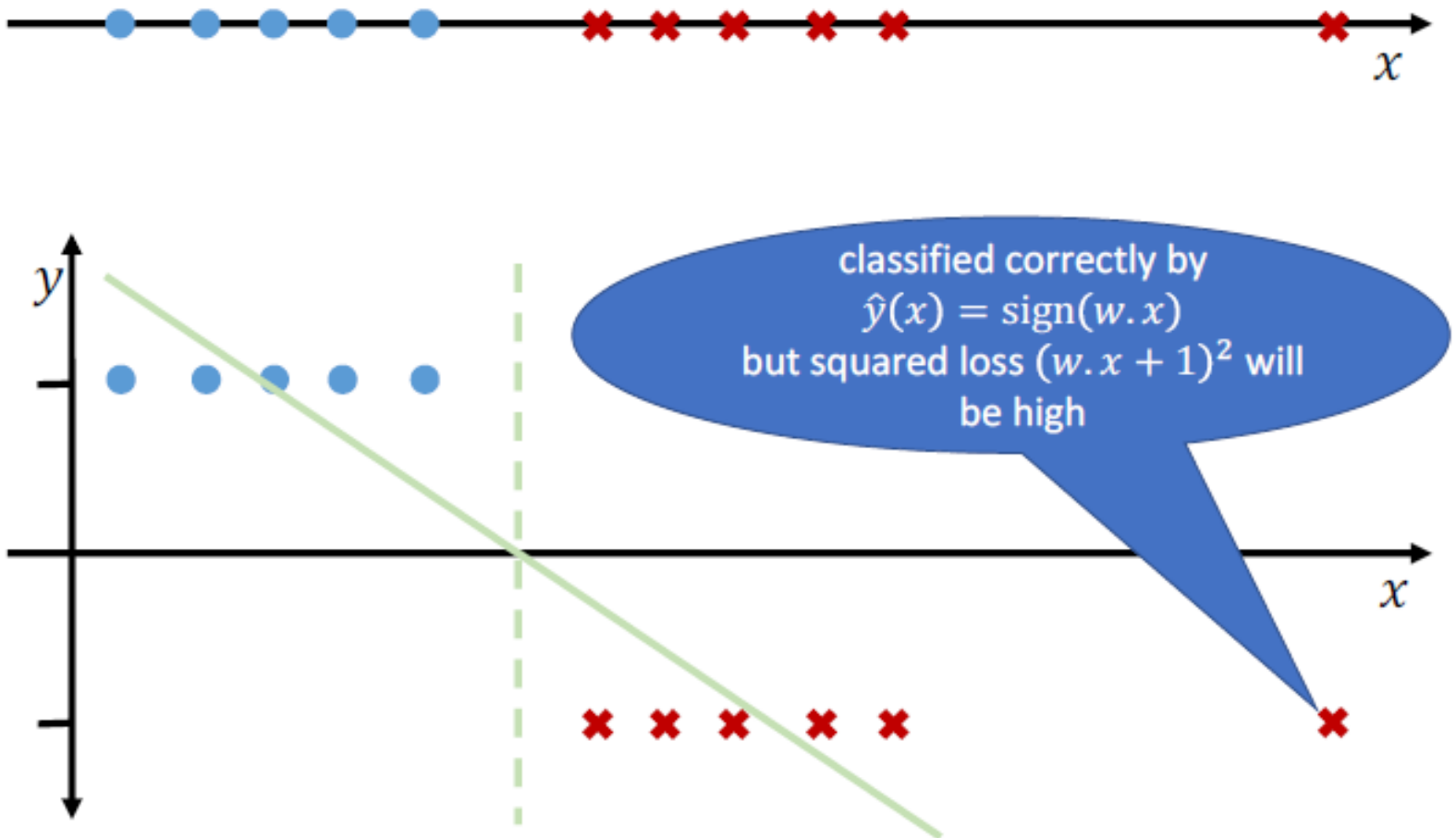




Example credit: Greg Shaknarovich

[Karl Stratos]

Classification as regression



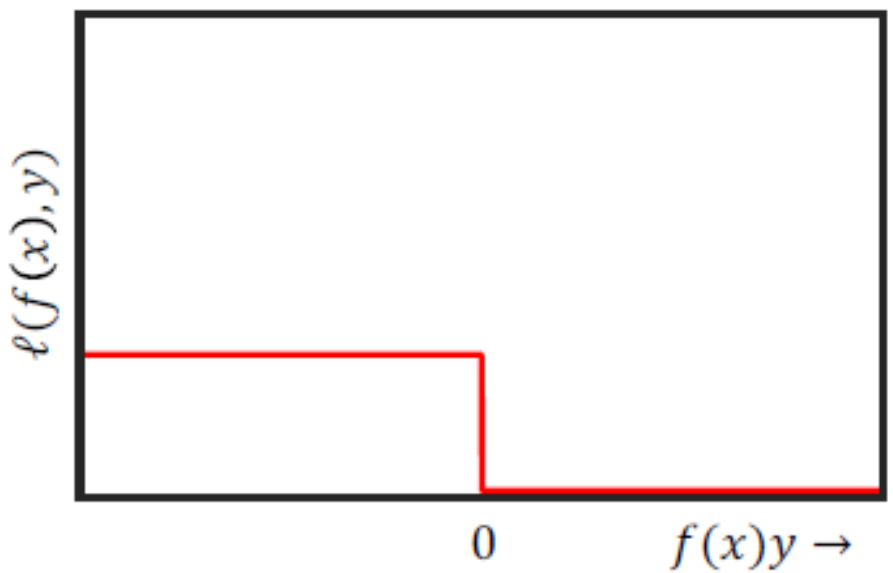
Example credit: Greg Shaknarovich

[Karl Stratos]

Surrogate Losses

- The correct loss to use is 0-1 loss ***after*** thresholding

$$\ell^{01}(f(x), y) = \mathbf{1}\left[\text{sign}\left(f(x)\right) \neq y\right]$$
$$= \mathbf{1}[\text{sign}(f(x)y) < 0]$$



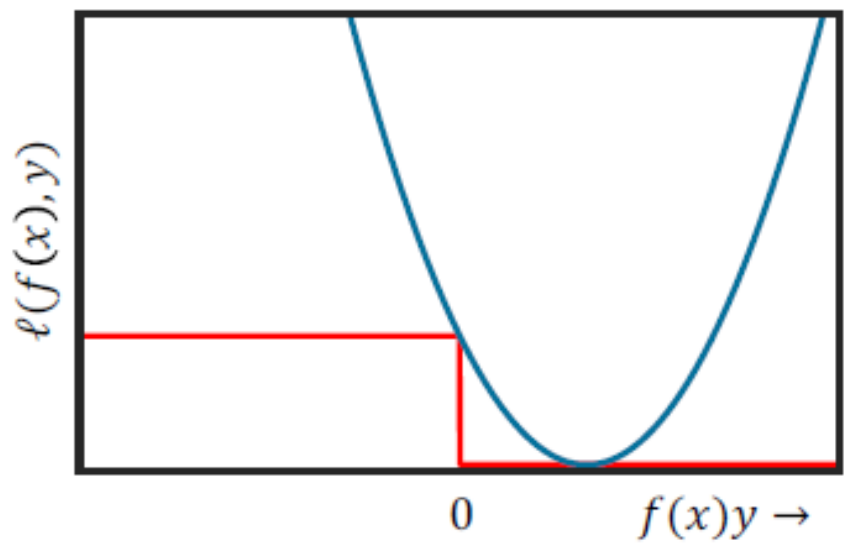
[Karl Stratos]

Surrogate Losses

- The correct loss to use is 0-1 loss ***after*** thresholding

$$\ell^{01}(f(x), y) = \mathbf{1}\left[\text{sign}\left(f(x)\right) \neq y\right]$$
$$= \mathbf{1}[\text{sign}(f(x)y) < 0]$$

- Linear regression uses $\ell^{LS}(f(x), y) = (f(x) - y)^2$



- Why not do ERM over $\ell^{01}(f(x), y)$ directly?
  - non-continuous, non-convex

[Karl Stratos]

NLP

Introduction to NLP

513.

Naïve Bayes

Generative vs. Discriminative Models

Generative

- Learn a model of the joint probability p(d, c)
- Use Bayes' Rule to calculate p(c|d)
- Build a model of each class; given example, return the model most likely to have generated that example
- Examples:
  - Naïve Bayes
  - HMM
  - Probabilistic CFG

Discriminative

- Model posterior probability p(c|d) directly
- Class is a function of document vector
- Find the exact function that minimizes classification errors on the training data
- Examples:
  - Logistic regression
  - Neural Networks (NNs)
  - Support Vector Machines (SVMs)
  - Decision Trees

Assumptions of Discriminative Classifiers

- Data examples (documents) are represented as vectors of features (words, phrases, ngrams, etc)
- Looking for a function that maps each vector into a class
- This function can be found by minimizing the errors on the training data (plus other various criteria)
- Different classifiers vary on what the function looks like, and how they find the function

Discriminative vs. Generative

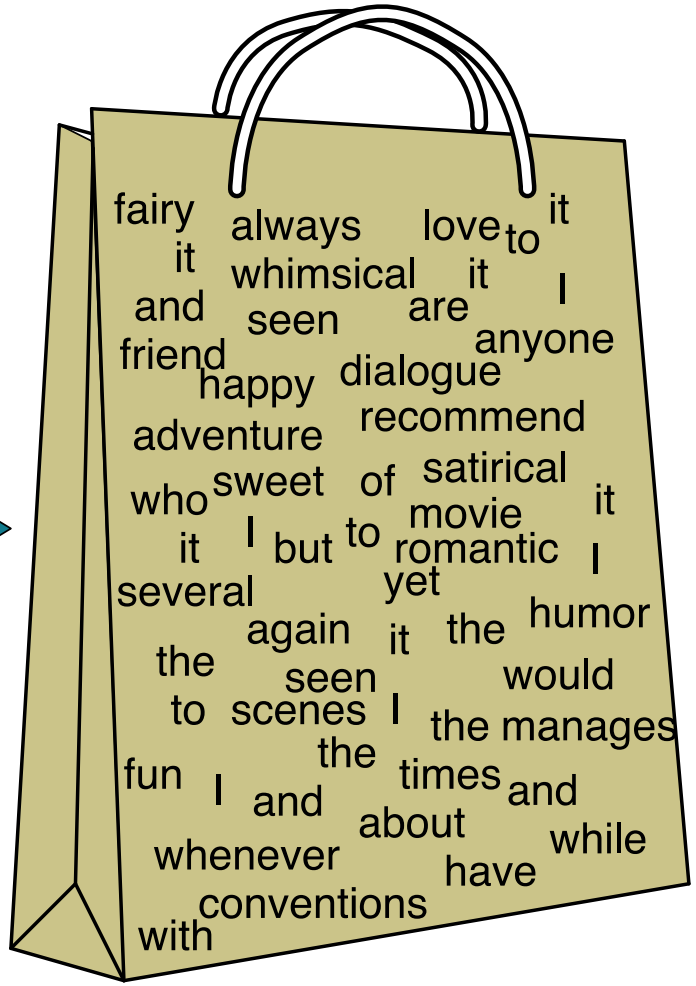
- Discriminative classifiers are generally more effective, since they directly optimize the classification accuracy. But
  - They are all sensitive to the choice of features, and so far these features are extracted heuristically
  - Also, overfitting can happen if data is sparse
- Generative classifiers are the “opposite”
  - They directly model text, an unnecessarily harder problem than classification
  - They can easily exploit unlabeled data

Naïve Bayes Intuition

- Simple (“naïve”) classification method based on Bayes rule
- Relies on very simple representation of document
  - Bag of words

Bag of Words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Remember Bayes' Rule?

Bayes' Rule: $$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

- *d* is the document (represented as a list of features of a document, x_1, …, x_n)
- *c* is a class (e.g., "not spam")

Naïve Bayes Classifier

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(c \mid d)$$

MAP is "maximum a posteriori" = most likely class

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(d \mid c)P(c)}{P(d)}$$

Bayes Rule

$$= \underset{c \in C}{\operatorname{argmax}} P(d \mid c)P(c)$$

Dropping the denominator

Naïve Bayes Classifier

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(d \mid c) P(c)$$

$$= \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, \ldots, x_n \mid c) P(c)$$

Document d represented as features x1..xn

But where will we get these probabilitites?

Naïve Bayes Classifier

- Naïve Bayesian classifier

$$P(d \in C | F_1, F_2, .. F_k) = \frac{P(F_1, F_2, .. F_k | d \in C) P(d \in C)}{P(F_1, F_2, .. F_k)}$$

- Assuming statistical independence

$$P(d \in C | F_1, F_2, .. F_k) = \frac{\prod_{j=1}^{k} P(F_j | d \in C) P(d \in C)}{\prod_{j=1}^{k} P(F_j)}$$

- Features = words (or phrases) typically

Training Naïve Bayes

function TRAIN NAIVE BAYES(D, C) **returns** log $P(c)$ and log $P(w|c)$

for each class $c \in C$ # Calculate $P(c)$ terms
  N_{doc} = number of documents in D
  N_c = number of documents from D in class c
  $logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$
  $V \leftarrow$ vocabulary of D
  $bigdoc[c] \leftarrow$ **append**(d) **for** d $\in$ D **with** class c
  for each word w in V # Calculate $P(w|c)$ terms
    $count(w,c) \leftarrow$ # of occurrences of w in $bigdoc[c]$
    $loglikelihood[w,c] \leftarrow \log \frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$
return $logprior$, $loglikelihood$, V

function TEST NAIVE BAYES($testdoc$, $logprior$, $loglikelihood$, C, V) **returns** best c

for each class $c \in C$
  $sum[c] \leftarrow logprior[c]$
  for each position i in $testdoc$
    $word \leftarrow testdoc[i]$
    if $word \in V$
      $sum[c] \leftarrow sum[c] + loglikelihood[word,c]$
return $\text{argmax}_c\ sum[c]$

Figure 4.2 The naive Bayes algorithm, using add-1 smoothing. To use add-α smoothing instead, change the $+1$ to $+\alpha$ for loglikelihood counts in training.

Learning the Parameters

- First attempt: maximum likelihood estimates
  - use the frequencies in the data

$$\hat{P}(c_j) = \frac{doccount(C = c_j)}{N_{doc}}$$

$$\hat{P}(w_i \mid c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$$

Parameter Estimation

$$\hat{P}(w_i|c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$$

fraction of times word w_i appears among all words in documents of topic c_j

- Create mega-document for topic *j* by concatenating all docs in this topic
  - Use frequency of *w* in mega-document

Problem with Maximum Likelihood

- What if we have seen no training documents with the word ***fantastic*** and classified in the topic **positive** (***thumbs-up)***?

$$\hat{P}(\text{"fantastic"}|\text{positive}) = \frac{count(\text{"fantastic"}, \text{positive})}{\sum_{w \in V} count(w, \text{positive})} = 0$$

- Zero probabilities cannot be conditioned away, no matter the other evidence!

$$c_{MAP} = \text{argmax}_c \hat{P}(c) \prod \hat{P}(x_i | c)$$

Laplace Smoothing

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c)}{\sum_{w \in V} (count(w, c))}$$

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} (count(w, c) + 1)}$$

$$= \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c) \right) + |V|}$$

Multinomial Naïve Bayes Independence Assumptions

$$P(x_1, x_2, \ldots, x_n \mid c)$$

- Bag of Words assumption
  - Assume position doesn't matter
- Conditional Independence
  - Assume the feature probabilities $P(x_i \mid c)$ are independent given the class c.

$$P(x_1, \ldots, x_n \mid c) = P(x_1 \mid c) \bullet P(x_2 \mid c) \bullet P(x_3 \mid c) \bullet \ldots \bullet P(x_n \mid c)$$

[Jurafsky and Martin]

Multinomial Naïve Bayes

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, \ldots, x_n \mid c) P(c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c_j) \prod_{x \in X} P(x \mid c)$$

This is why it's naïve!

[Jurafsky and Martin]

Multinomial NB - Learning

- From training corpus, extract *Vocabulary*

- Calculate $P(c_j)$ terms
  - For each c_j in C do

    $docs_j \leftarrow$ all docs with class $=c_j$

$$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$

- Calculate $P(w_k \mid c_j)$ terms
  - $Text_j \leftarrow$ single doc containing all $docs_j$
  - For each word w_k in *Vocabulary*

    $n_k \leftarrow$ # of occurrences of w_k in $Text_j$

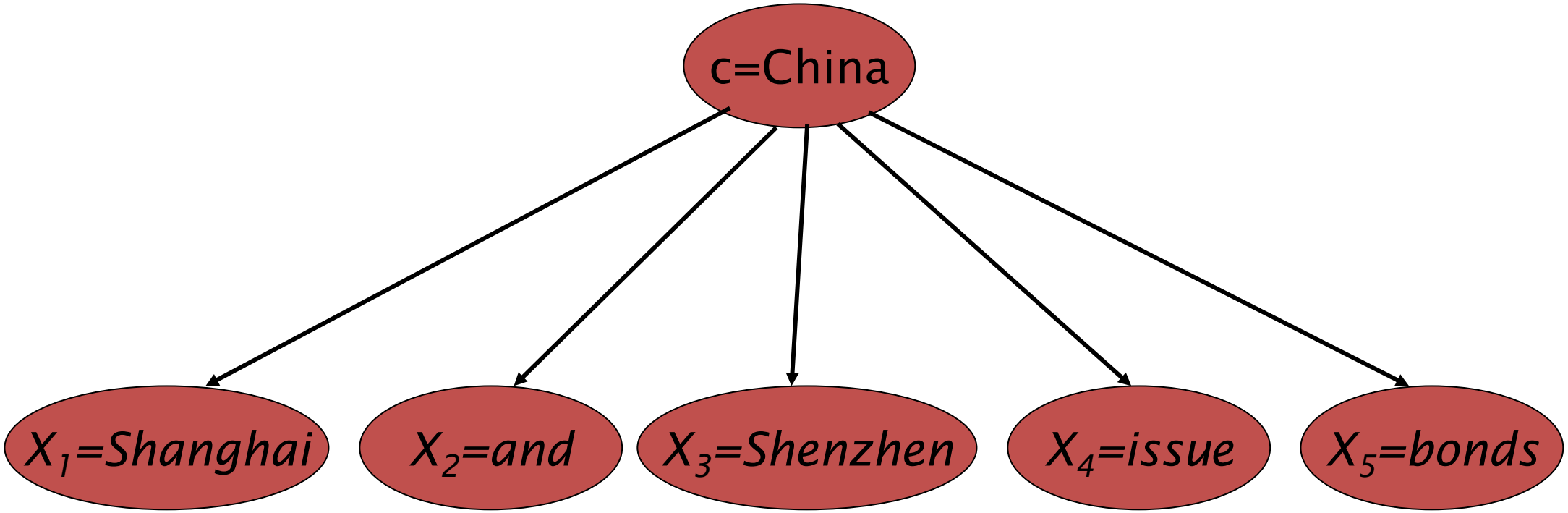
$$P(w_k \mid c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha|Vocabulary|}$$

Multinomial NB Classifier for Text

positions ← all word positions in test document

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_{i \in positions} P(x_i \mid c_j)$$

Generative Model for Multinomial NB



Example

- Features =
  [I hate love this book]
- Training
  - I hate this book = [1 1 0 1 1]
  - Love this book = [0 0 1 1 1]
- What is P(Y|X)?
- Prior p(Y)
- Testing
  - hate book = [0 1 0 0 1]
- Different conditions
  - a = 0 (no smoothing)
  - a = 1 (smoothing)

$$P(Y) = [1/2 \quad 1/2]$$

a = 0 (no smoothing)

$$M = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$P(X|Y) = \begin{bmatrix} 1/4 & 1/4 & 0 & 1/4 & 1/4 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$P(Y|X) \propto [1/2 \times 1/4 \times 1/4 \quad 1/2 \times 0 \times 1/3]$$
$$= [1 \quad 0]$$

a = 1 (smoothing)

$$M = \begin{bmatrix} 2 & 2 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 & 2 \end{bmatrix}$$

$$P(X|Y) = \begin{bmatrix} 2/9 & 2/9 & 1/9 & 2/9 & 2/9 \\ 1/8 & 1/8 & 2/8 & 2/8 & 2/8 \end{bmatrix}$$

$$P(Y|X) \propto [1/2 \times 2/9 \times 2/9 \quad 1/2 \times 1/8 \times 2/8]$$
$$= [0.613 \quad 0.387]$$

Another Example

this movie was great! would watch again	+
I liked it well enough for an action flick	+
I expected a great film and left happy	+
brilliant directing and stunning visuals	+
that film was awful, I'll never watch again	—
I didn't really like that movie	—
dry and a bit distasteful, it misses the mark	—
great potential but ended up being a flop	—

$$P(+) = \frac{1}{2}$$

$$P(-) = \frac{1}{2}$$

$$P(\text{great}|+) = \frac{1}{2}$$

$$P(\text{great}|-) = \frac{1}{4}$$

it was great ⟶ $P(y|x) \propto \begin{bmatrix} P(+)P(\text{great}|+) \\ P(-)P(\text{great}|-) \end{bmatrix} = \begin{bmatrix} 1/4 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix}$

[Greg Durrett]

Summary (1)

- Data point $x = (x_1, ..., x_n)$, label $y \in \{0, 1\}$
- Formulate a probabilistic model that places a distribution $P(x, y)$
- Compute $P(y|x)$, predict $\text{argmax}_y P(y|x)$ to classify

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)}$$

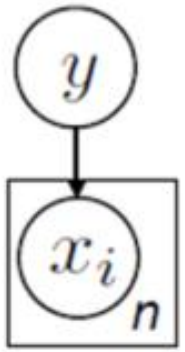
Bayes' Rule

$P(x)$ ← constant: irrelevant for finding the max

$$\propto P(y)P(x|y)$$

"Naive" assumption:

$$= P(y)\prod_{i=1}^{n} P(x_i|y)$$



linear model!

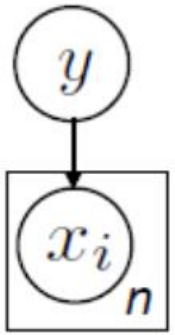
$$\text{argmax}_y P(y|x) = \text{argmax}_y \log P(y|x) = \text{argmax}_y \left[\log P(y) + \sum_{i=1}^{n} \log P(x_i|y)\right]$$

[Greg Durrett]

Summary (2)

- Model

$$P(x, y) = P(y) \prod_{i=1}^{n} P(x_i|y)$$



- Inference

$$\text{argmax}_y \log P(y|x) = \text{argmax}_y \left[\log P(y) + \sum_{i=1}^{n} \log P(x_i|y) \right]$$

- Alternatively: $\log P(y = +|x) - \log P(y = -|x) > 0$

$$\Leftrightarrow \log \frac{P(y = +|x)}{P(y = -|x)} + \sum_{i=1}^{n} \log \frac{P(x_i|y = +)}{P(x_i|y = -)} > 0$$

- Learning: maximize $P(x, y)$ by reading counts off the data

[Greg Durrett]

Not So Naïve after all!

- Very fast, low storage requirements
- Robust to irrelevant features
- Irrelevant features cancel each other without affecting results
- Very good in domains with many equally important features
  - Decision trees suffer from *fragmentation* in such cases – especially if little data
- Optimal if the independence assumptions hold
  - If assumed independence is correct, then it is the Bayes Optimal Classifier for problem
- A good, dependable baseline for text classification
  - But other classifiers give better accuracy

[Jurafsky and Martin]

Naïve Bayes is a Generative Model

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

Scoping

```
didnt like this movie , but I
```

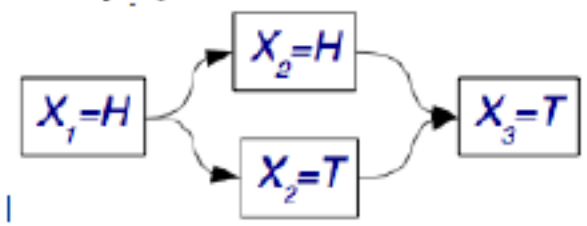
becomes

```
didnt NOT_like NOT_this NOT_movie , but I
```

Missing features

One of the key strengths of Bayesian approaches is that they can naturally handle missing data

- What happens if we don't have value of some feature $x_k^{(i)}$
  - e.g., applicants credit history unknown
  - e.g., some medical tests not performed
- How to compute $\Pr(x_1, x_2, \ldots x_{j-1}, ?, x_{j+1} \ldots, x_d | y)$?
  - e.g., three coin tosses $\mathrm{E} = \{H, ?, T\}$
  - $\Rightarrow \Pr(E) = \Pr(\{H, H, T\}) + \Pr(\{H, T, T\})$



- More generally

$$\Pr(x_1, x_2, \ldots x_{j-1}, ?, x_{j+1} \ldots, x_d | y) = \sum_{z_j} \Pr(x_1, x_2, \ldots x_{j-1}, z_j, x_{j+1} \ldots, x_d | y)$$

[Karl Stratos]

Missing features in naive Bayes

$$\Pr(x_1, x_2, \ldots x_{j-1}, ?, x_{j+1} \ldots, x_d | y)$$

$$= \sum_{z_j} \Pr(x_1, x_2, \ldots x_{j-1}, z_j, x_{j+1} \ldots, x_d | y)$$

$$= \sum_{z_j} \left[\Pr(z_j | y) \prod_{k \neq j} \Pr(x_k | y) \right]$$

$$= \prod_{k \neq j} \Pr(x_k | y) \sum_{z_j} \Pr(z_j | y)$$

$$= \prod_{k \neq j} \Pr(x_k | y)$$

- Simply ignore the missing values and compute likelihood based only observed features
- no need to fill-in or explicitly model missing values

[Karl Stratos]

NLP

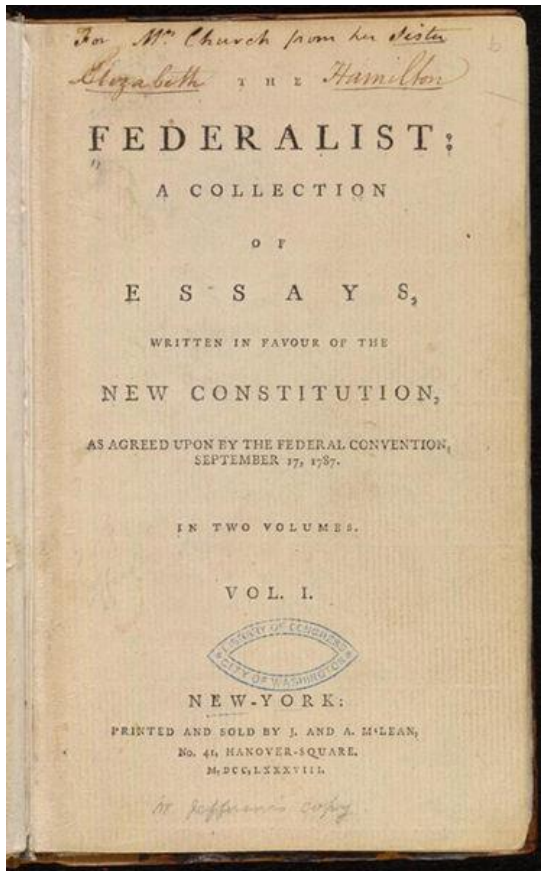
Introduction to NLP

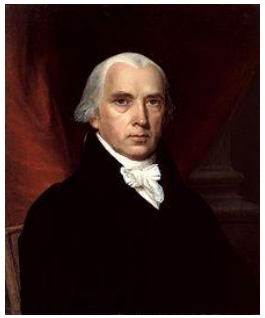
541.

Text Classification

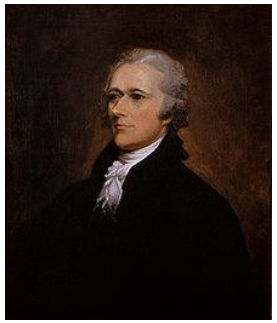
Who wrote which Federalist papers?

- 1787-8: anonymous essays try to convince New York to ratify U.S Constitution: Jay, Madison, Hamilton.
- Authorship of 12 of the letters in dispute
- 1963: solved by Mosteller and Wallace using Bayesian methods

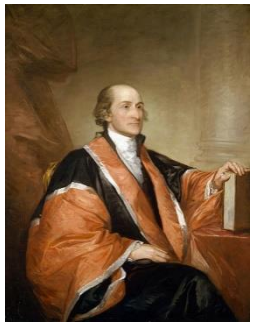




James Madison



Alexander Hamilton



John Jay

Positive or negative movie review?

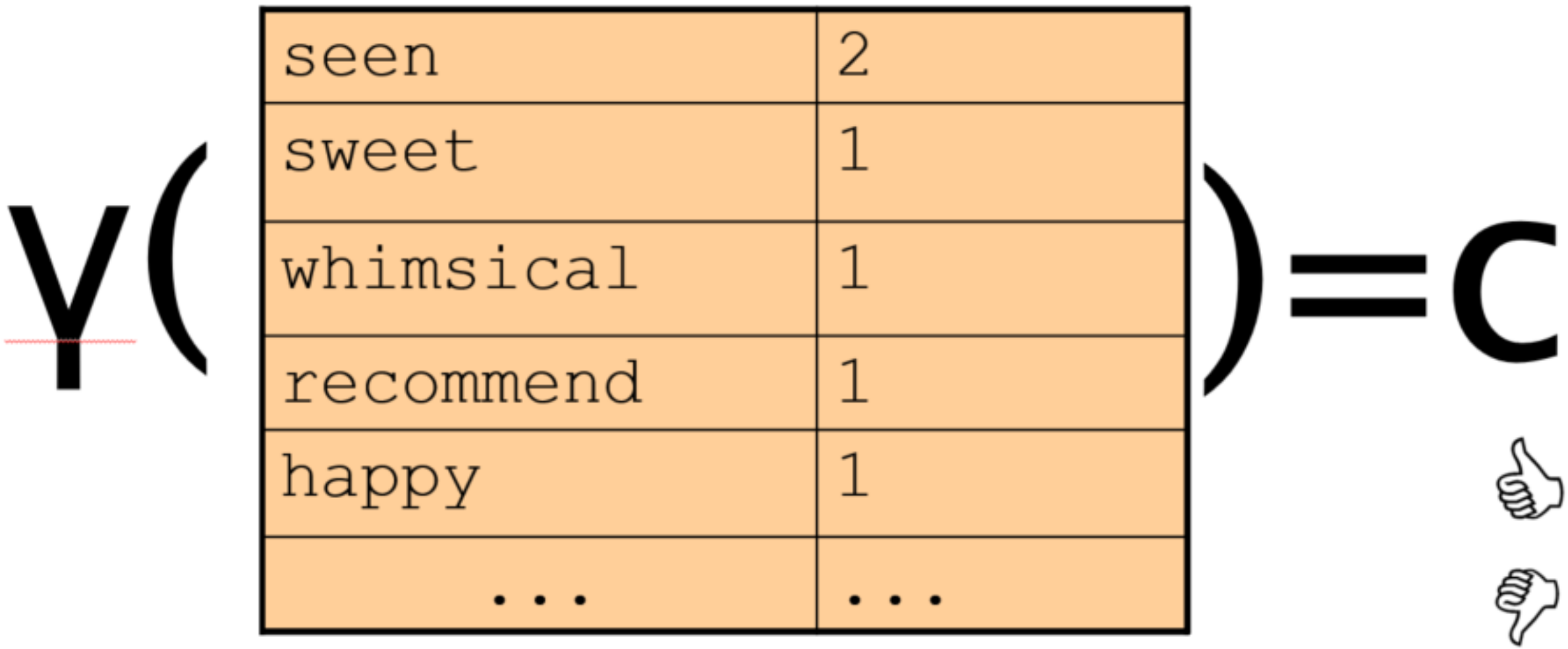
- unbelievably disappointing

- Full of zany characters and richly applied satire, and some great plot twists

- this is the greatest screwball comedy ever filmed

- It was pathetic. The worst part about it was the boxing scenes.

The bag of words representation



What is the subject of this article?

MEDLINE Article

Syntactic frame and verb bias in aphasia: Plausibility judgments of undergoer-subject sentences

?

MeSH Subject Category Hierarchy

- Antagonists and Inhibitors
- Blood Supply
- Chemistry
- Drug Therapy
- Embryology
- Epidemiology
- …

Introduction to NLP

Evaluation of Classification

Accuracy

- Percentage of items correctly classified
- Not a great metric for imbalanced data sets. Why?

Contingency Table

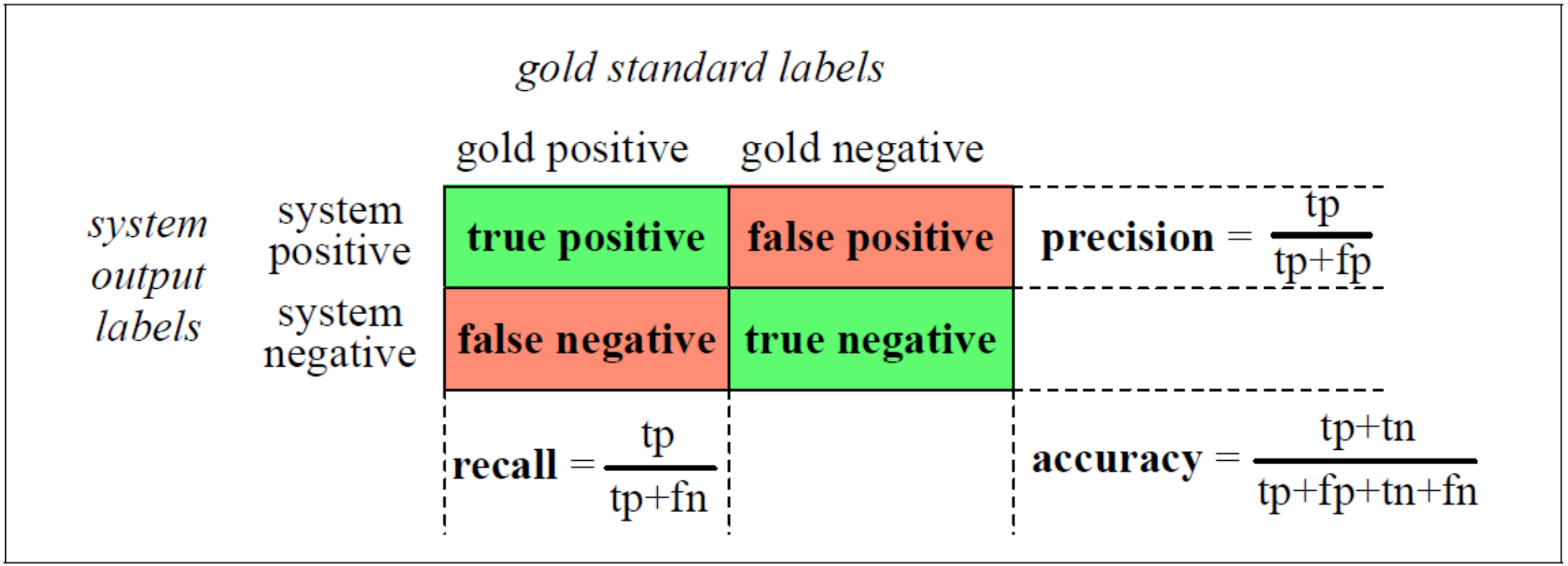


Figure 4.4 Contingency table

A combined measure: F

- A combined measure that assesses the P/R tradeoff is F measure (weighted harmonic mean):

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

- The harmonic mean is a very conservative average
- People usually use balanced F1 measure
  - i.e., with $\beta = 1$ (that is, $\alpha = ½$): $F1 = 2PR/(P+R)$

More than two classes

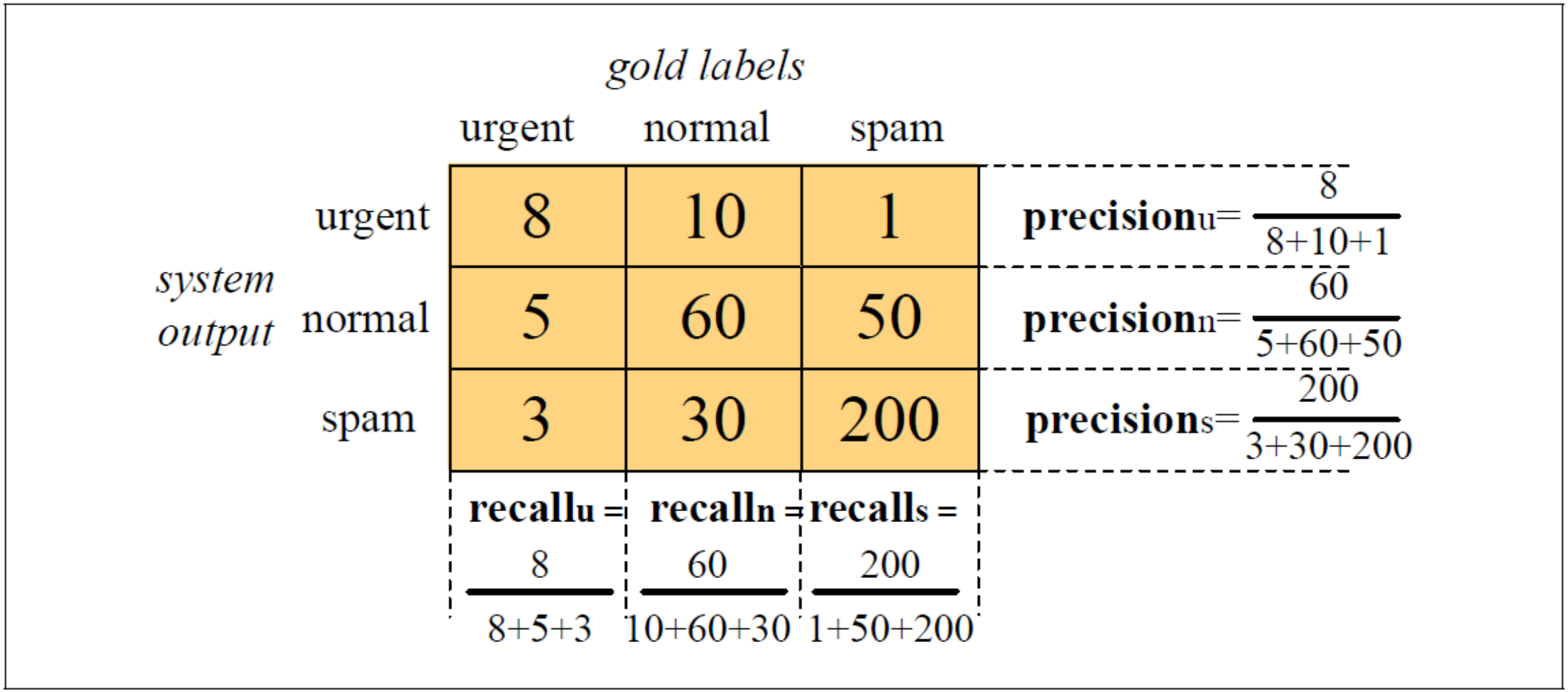


Figure 4.5 Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2), how many documents from c_1 were (in)correctly assigned to c_2

Micro-averaging and Macro-averaging

If we have more than one class, how do we combine multiple performance measures into one quantity?

- **Macroaveraging**: Compute performance for each class, then average.
- **Microaveraging**: Collect decisions for all classes, compute contingency table, evaluate.

Micro-averaging and Macro-averaging

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{precision} = \frac{8}{8+11} = .42 \qquad \text{precision} = \frac{60}{60+55} = .52 \qquad \text{precision} = \frac{200}{200+33} = .86$$

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

Figure 4.6 Separate contingency tables for the 3 classes from the previous figure, showing the pooled contingency table and the microaveraged and macroaveraged precision.

Classic Reuters-21578 Data Set

- Most (over)used data set, 21,578 docs (each 90 types, 200 tokens)
- 9603 training, 3299 test articles (ModApte/Lewis split)
- 118 categories
  - An article can be in more than one category
  - Learn 118 binary category distinctions
- Average document (with at least one category) has 1.24 classes
- Only about 10 out of 118 categories are large

Common categories
(#train, #test)

- Earn (2877, 1087)
- Acquisitions (1650, 179)
- Money-fx (538, 179)
- Grain (433, 149)
- Crude (389, 189)
- Trade (369,119)
- Interest (347, 131)
- Ship (197, 89)
- Wheat (212, 71)
- Corn (182, 56)

Reuters-21578 document

<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="12981" NEWID="798">

<DATE> 2-MAR-1987 16:51:43.42</DATE>

<TOPICS><D>livestock</D><D>hog</D></TOPICS>

<TITLE>AMERICAN PORK CONGRESS KICKS OFF TOMORROW</TITLE>

<DATELINE> CHICAGO, March 2 - </DATELINE><BODY>The American Pork Congress kicks off tomorrow, March 3, in Indianapolis with 160 of the nations pork producers from 44 member states determining industry positions on a number of issues, according to the National Pork Producers Council, NPPC.

Delegates to the three day Congress will be considering 26 resolutions concerning various issues, including the future direction of farm policy and the tax law as it applies to the agriculture sector. The delegates will also debate whether to endorse concepts of a national PRV (pseudorabies virus) control and eradication program, the NPPC said.

A large trade show, in conjunction with the congress, will feature the latest in technology in all areas of the industry, the NPPC added. Reuter

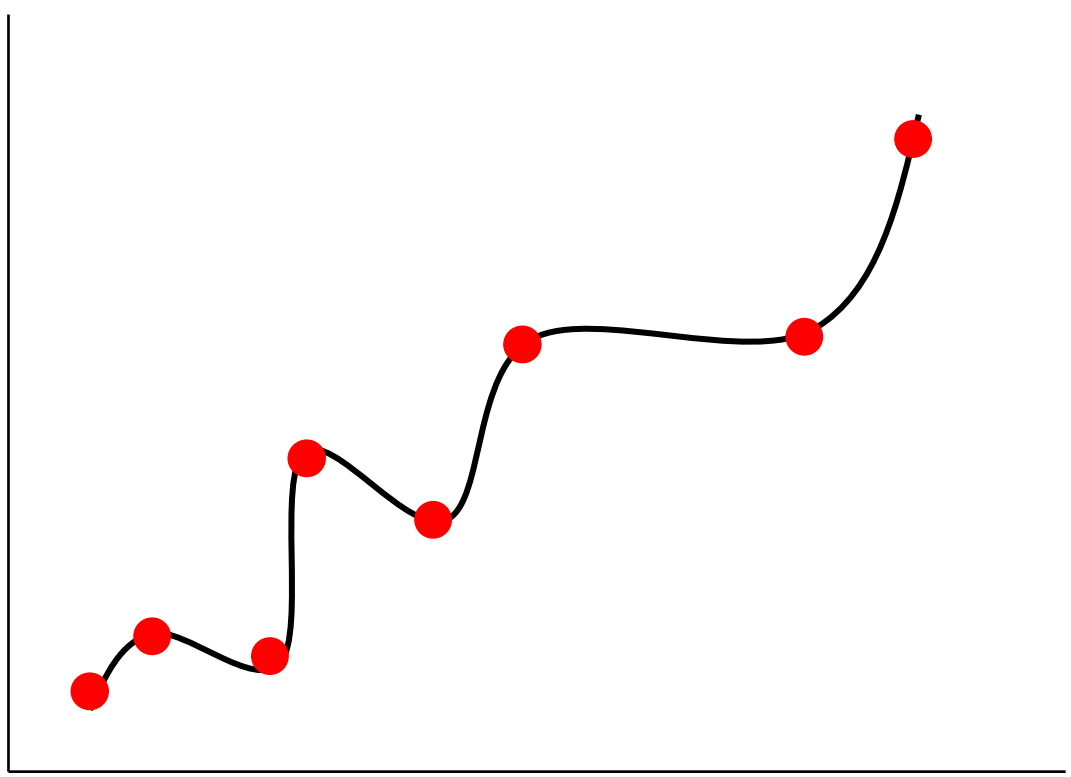
&#3;</BODY></TEXT></REUTERS>

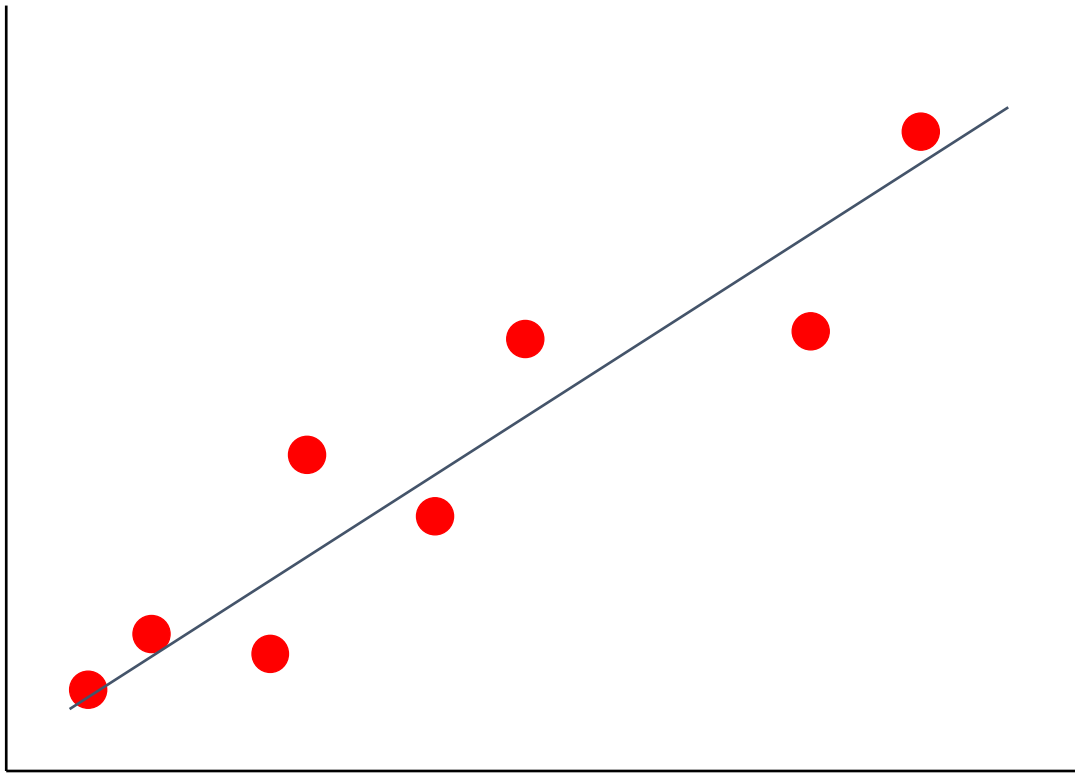
Confusion Matrix

- For each pair of classes <c_1,c_2> how many documents from c_1 were incorrectly assigned to c_2?
  - $c_{3,2}$: 90 wheat documents incorrectly assigned to poultry

Docs in test set	Assigned UK	Assigned poultry	Assigned wheat	Assigned coffee	Assigned interest	Assigned trade
True UK	95	1	13	0	1	0
True poultry	0	1	0	0	0	0
True wheat	10	90	0	1	0	0
True coffee	0	0	0	34	3	7
True interest	-	1	2	13	26	5
True trade	0	0	2	14	5	10

Pitfall: Overfitting

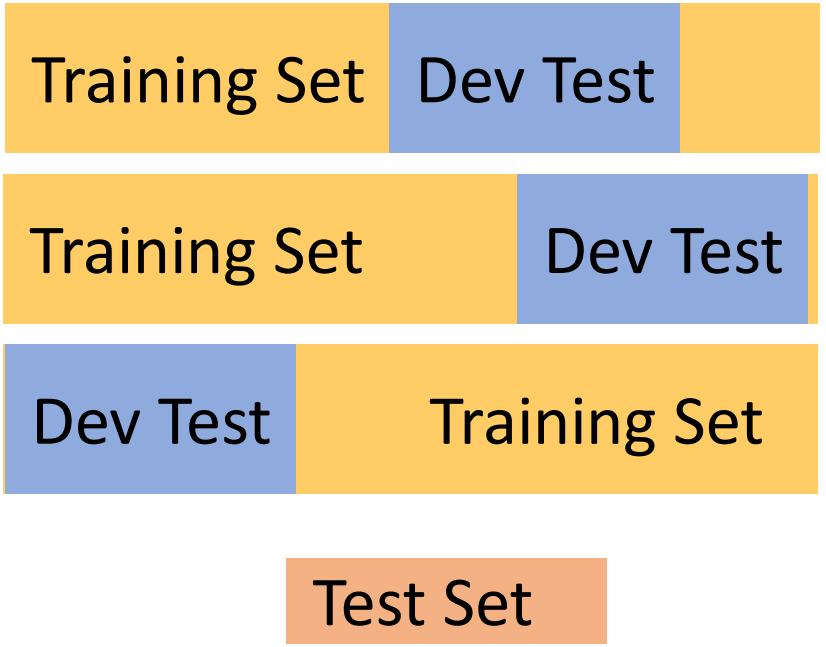




What happens when your model learns your training data a little too well?

Development Test Sets and Cross-validation

- Metric: P/R/F1 or Accuracy
- Unseen test set
  - avoid overfitting ('tuning to the test set')
  - more conservative estimate of performance
- Cross-validation over multiple splits
  - Handle sampling errors from different datasets
  - Pool results over each split
  - Compute pooled dev set performance



Cross-validation

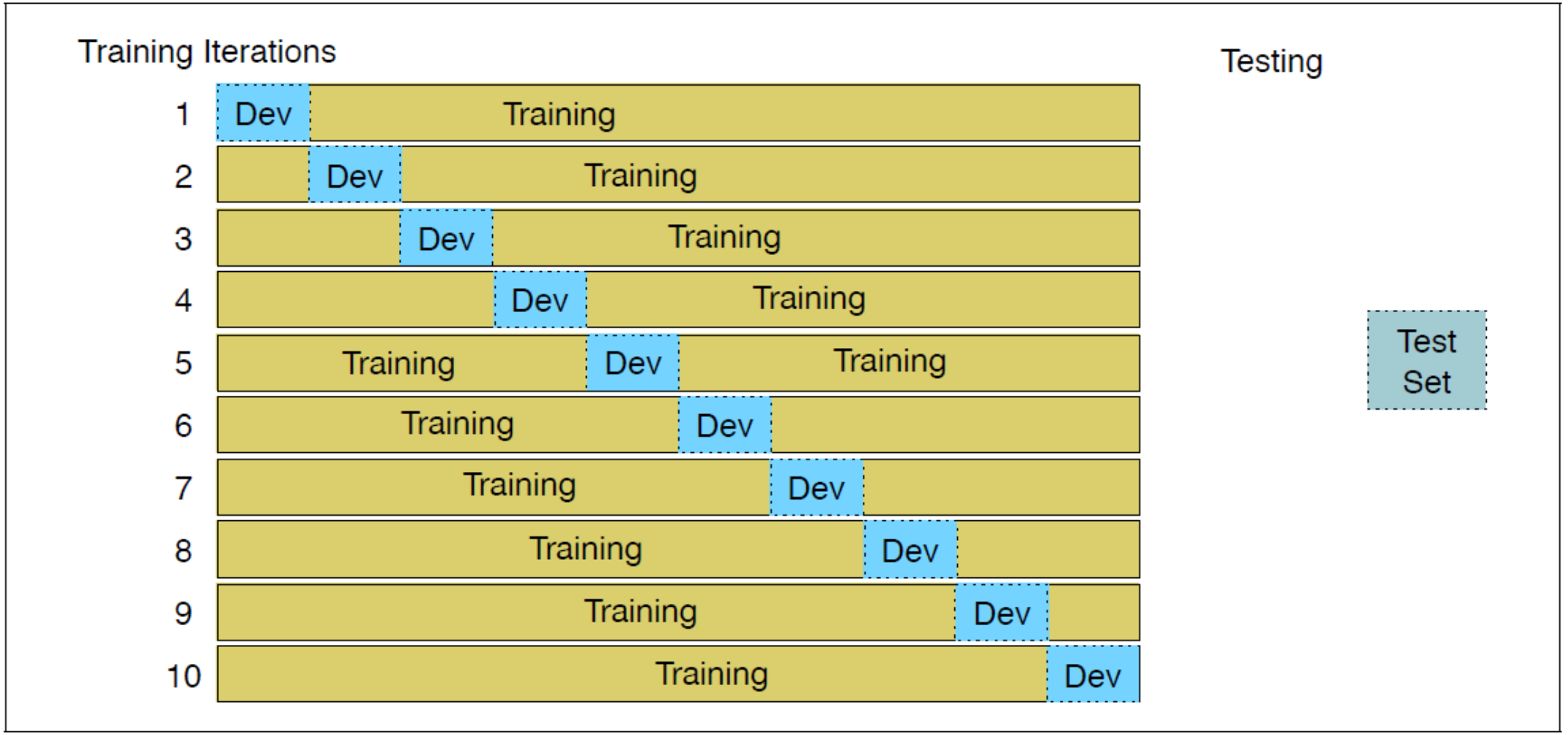


Figure 4.7 10-fold cross-validation

NLP

Introduction to NLP

516.

Logistic Regression

Combining Features

- Linear interpolation for language modeling
  - Estimating the trigram probability P(c|ab)?
  - features: $P_{MLE}(c|a)$, $P_{MLE}(c)$, etc.
  - Weights: λ_1, λ_2, etc.
- We may want to consider other features
  - E.g., POS tags of previous words, heads, word endings, etc.
- General idea
  - Compute the conditional probability P(y|x)
  - P(y|x) = sum of weights*features
- Label Y for a given history x in X

Logistic Regression

- Similar to Naïve Bayes (but discriminative!)
  - Log-linear model
  - Features don’t have to be independent
- Examples of features
  - Anything of use
  - Linguistic and non-linguistic
  - Count of “good”
  - Count of “not good”
  - Sentence length
- Convex optimization problem
  - It can be solved using gradient descent

Feature Templates

- For each word $w
  - $w_count
  - $w_ends_in_er
  - $w_is_negated
  - $w_is_all_caps

Logistic Regression

- An example of a discriminative classifier
- Input:
  - Training example pairs of $(\vec{x}, y)$ where $\vec{x}$ is the feature vector and *y* is the label
- Goal:
  - Build a model that predicts the probability of the label
- Output:
  - Set of weights $\vec{w}$ that maximizes likelihood of correct labels on training examples

Loglinear Models

- Naive Bayes, HMM, and Logistic Regression can all be expressed in a similar way:

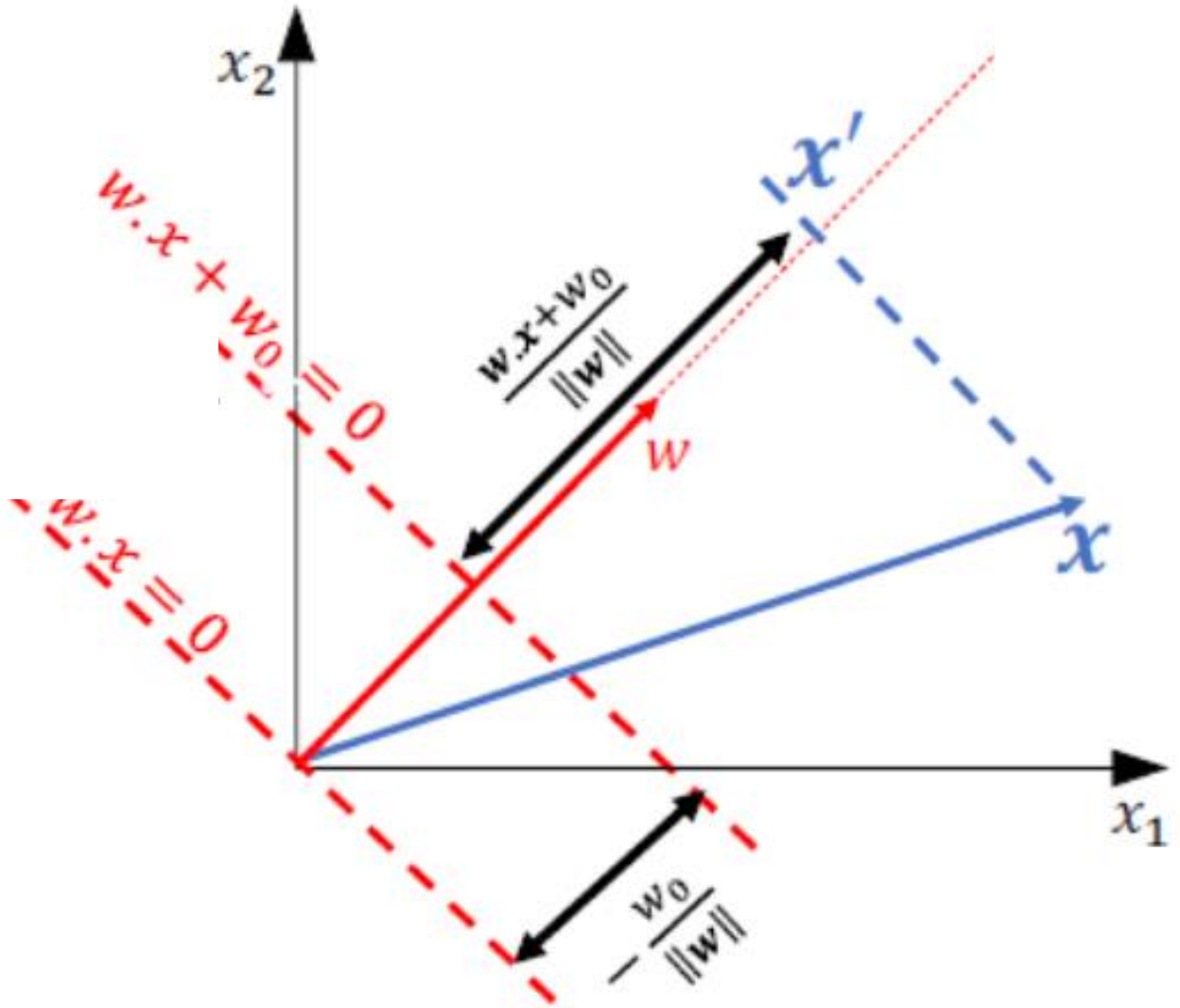
$$G(y, \theta) = \theta^T f(x, y)$$

$$p(y|x, \theta) \propto \exp G(y, \theta) \Leftrightarrow \log p(y|x) = C + G(y, \theta)$$

- Decision rule:

$$\underset{y^*}{\operatorname{argmax}}\, G(y^*, \theta)$$

Mapping x to a 1-D coordinate



[image from Greg Shakhnarovich]

Mapping x to a 1-D coordinate

$$\widehat{\boldsymbol{w}}, \widehat{w}_0 = \underset{\boldsymbol{w}, w_0}{\operatorname{argmin}} \sum_i \log\left(1 + \exp\left(-\left(\boldsymbol{w}.\boldsymbol{x}^{(i)} + w_0\right) y^{(i)}\right)\right)$$

- Learns a linear decision boundary
  - $\{\boldsymbol{x}: \boldsymbol{w}.\boldsymbol{x} + w_0 = 0\}$ is a hyperplane in $\mathbb{R}^d$ - decision boundary
  - $\{\boldsymbol{x}: \boldsymbol{w}.\boldsymbol{x} + w_0 = 0\}$ divides $\mathbb{R}^d$ into two halfspace (regions)
  - $\{\boldsymbol{x}: \boldsymbol{w}.\boldsymbol{x} + w_0 \geq 0\}$ will get label +1 and $\{\boldsymbol{x}: \boldsymbol{w}.\boldsymbol{x} + w_0 < 0\}$ will get label -1

- Maps $\boldsymbol{x}$ to a 1D coordinate

$$x' = \frac{\boldsymbol{w}.\boldsymbol{x} + w_0}{\|\boldsymbol{w}\|}$$

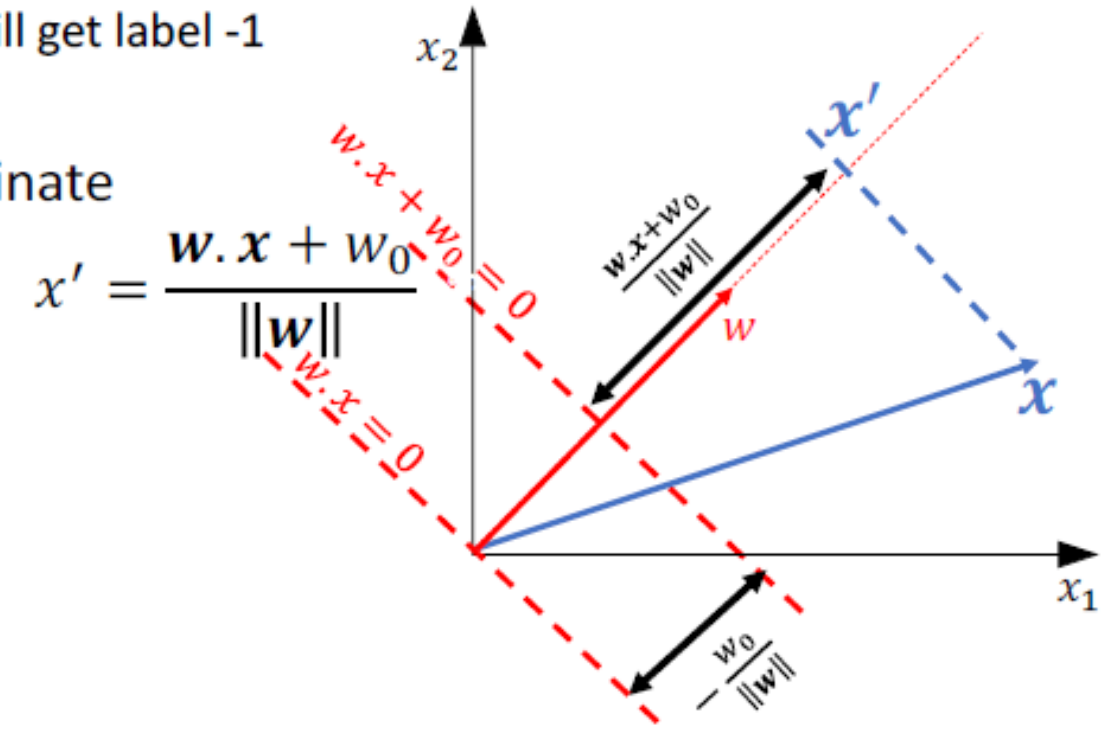


Figure credit: Greg Shaknarovich

Logistic Regression

- Model: $P_w(y|x) = \frac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$
- Inference: $\text{argmax}_y P_w(y|x)$
- Learning: gradient ascent on the discriminative log-likelihood

$$f(x,y^*) - \mathbb{E}_y[f(x,y)] = f(x,y^*) - \sum_y [P_w(y|x) f(x,y)]$$

"towards gold feature value, away from expectation of feature value"

[discriminative method]

[Greg Durrett]

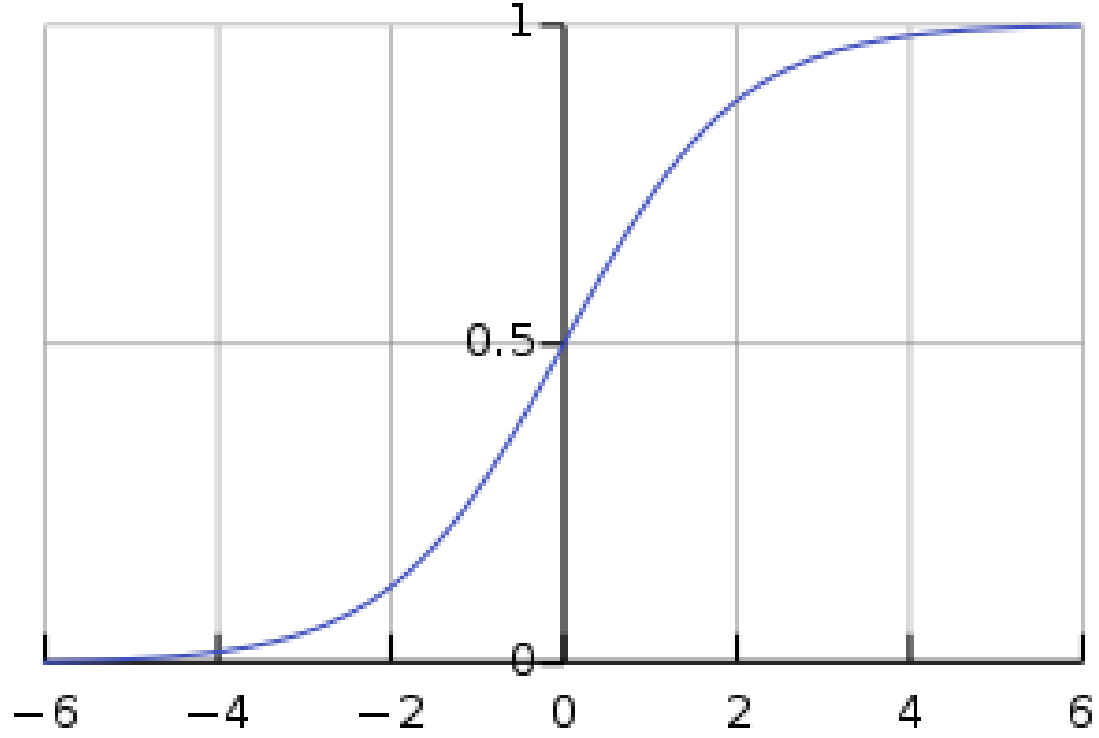
Including a Nonlinearity

- Compute the feature vector *x*
- Multiply with weight vector *w*

$$z = \sum w_i x_i$$

- Compute the logistic function (sigmoid)

$$f(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid Function

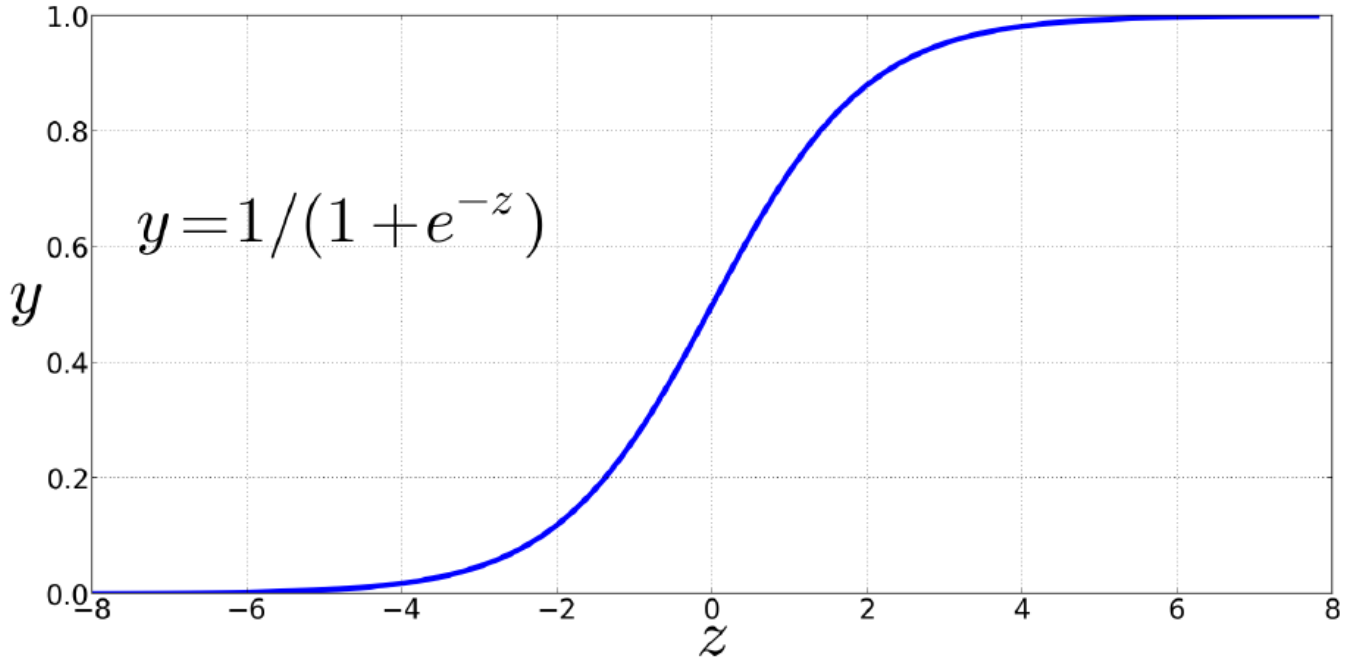


Figure 5.1 The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0, 1]$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

Non-linear Activation Functions

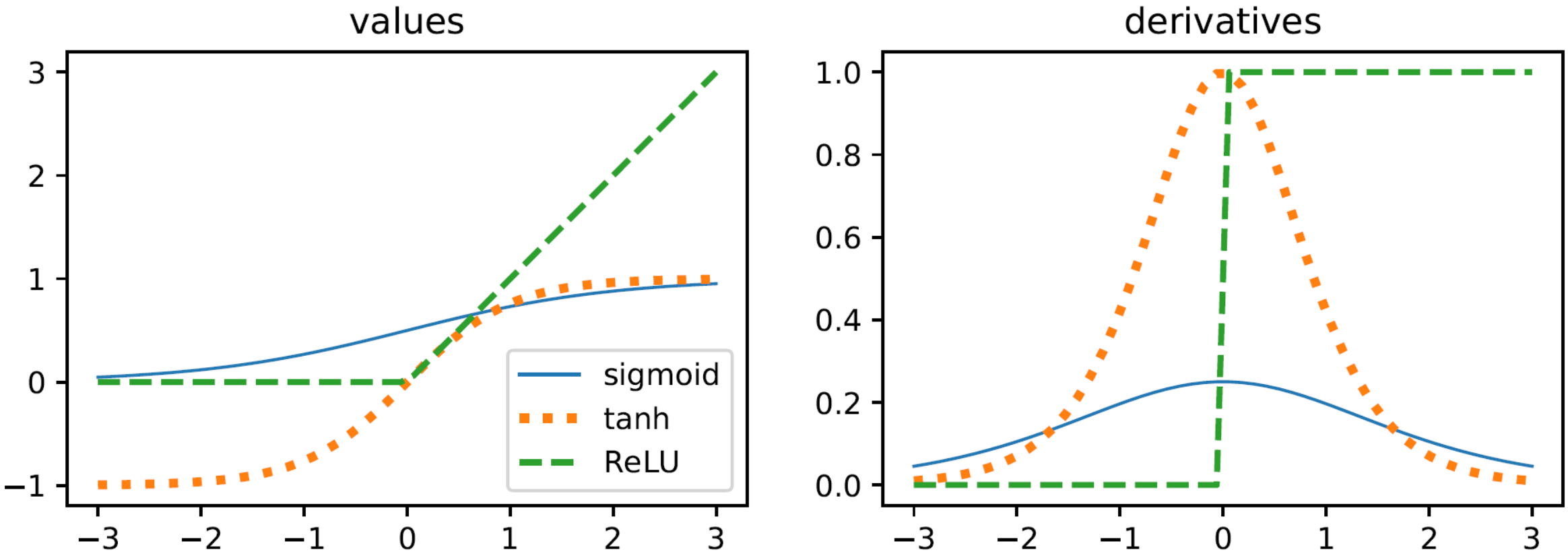


Figure 3.2: The sigmoid, tanh, and ReLU activation functions

Training and Testing LR

Logistic regression has two phases:

training: we train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss.

test: Given a test example x we compute $p(y|x)$ and return the higher probability label $y = 1$ or $y = 0$.

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

Classification using LR

$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\begin{aligned} P(y=1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1+e^{-(w \cdot x+b)}} \end{aligned}$$

$$\begin{aligned} P(y=0) &= 1-\sigma(w \cdot x + b) \\ &= 1-\frac{1}{1+e^{-(w \cdot x+b)}} \\ &= \frac{e^{-(w \cdot x+b)}}{1+e^{-(w \cdot x+b)}} \end{aligned} \quad (5.5)$$

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Examples

- Example 1

  $x = (2,1,1,1)$

  $w = (1,-1,-2,3)$

  $z = 2-1-2+3=2$

  $f(z) = 1/(1+e^{-2})$

- Example 2

  $x = (2,1,0,1)$

  $w = (0,0,-3,0)$

  $z = 0$

  $f(z) = 1/(1+e^{0}) = 1/2$

Example

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Example

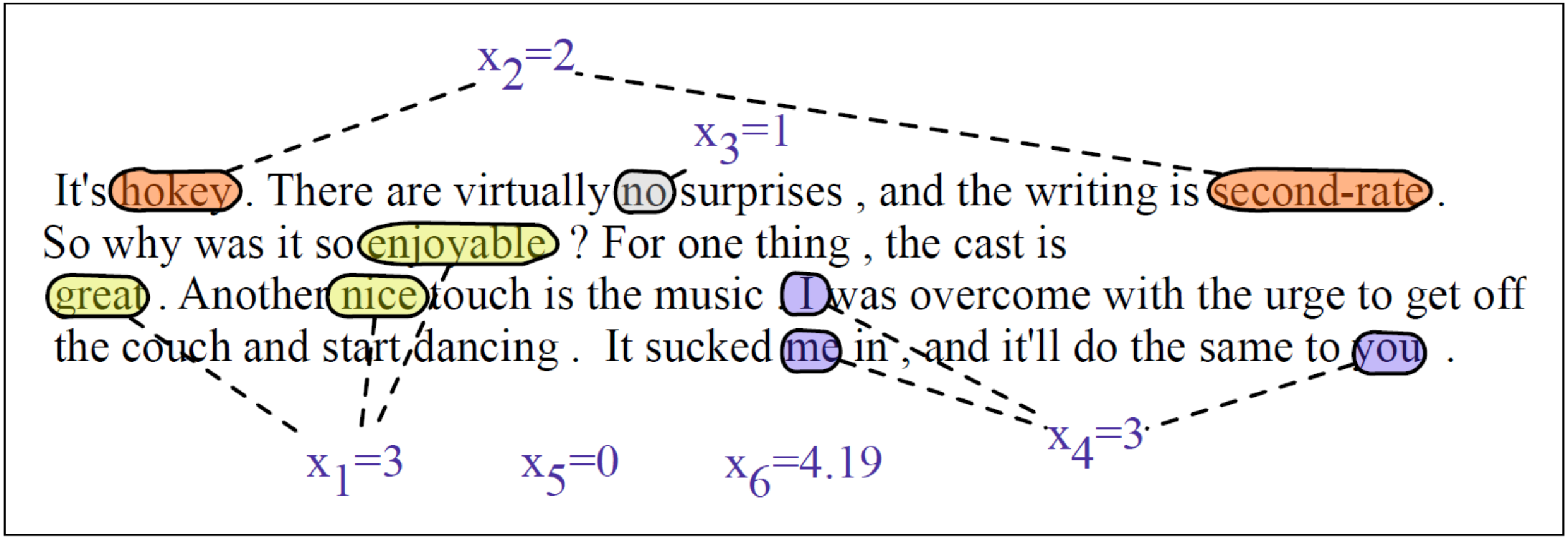


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Per-class Probabilities

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w\cdot x+b) \\
&= \sigma([2.5,-5.0,-1.2,0.5,2.0,0.7]\cdot[3,2,1,3,0,4.19]+0.1) \\
&= \sigma(.833) \\
&= 0.70 \qquad (5.6) \\
p(-|x) = P(Y=0|x) &= 1-\sigma(w\cdot x+b) \\
&= 0.30
\end{aligned}
$$

Period Disambiguation

$$x_1 = \begin{cases} 1 & \text{if } \text{“}Case(w_i) = \text{Lower”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if } \text{“}w_i \in \text{AcronymDict”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if } \text{“}w_i = \text{St. \& } Case(w_{i-1}) = \text{Cap”} \\ 0 & \text{otherwise} \end{cases}$$

Cross-Entropy Loss (5.9)

$$L(\hat{y}, y) = \text{How much } \hat{y} \text{ differs from the true } y$$

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

$$\begin{aligned} \log p(y|x) &= \log \left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y \log \hat{y} + (1-y) \log(1-\hat{y}) \end{aligned} \tag{5.9}$$

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \qquad \hat{y} = \sigma(w \cdot x + b)$$

$$L_{CE}(w, b) = -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \tag{5.11}$$

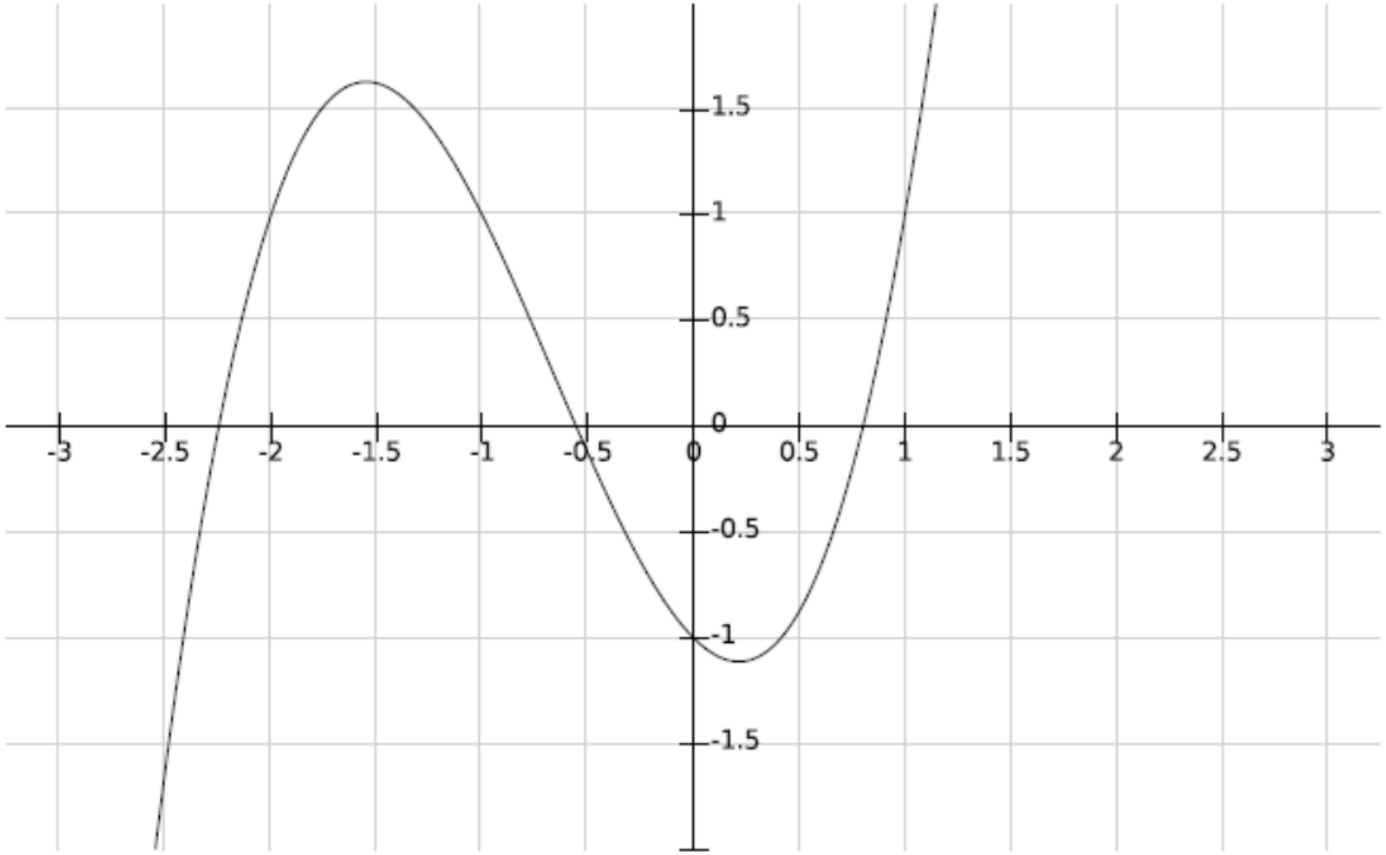
Two examples (.69 vs .31)

$$
\begin{aligned}
L_{CE}(w,b) &= -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\
&= -[\log \sigma(w \cdot x + b)] \\
&= -\log(.69) \\
&= .37
\end{aligned}
$$

$$
\begin{aligned}
L_{CE}(w,b) &= -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\
&= -[\log (1 - \sigma(w \cdot x + b))] \\
&= -\log (.31) \\
&= 1.17
\end{aligned}
$$

Function derivatives (refresher)

Minimize $f(x) = x^3 + 2x^2 - x - 1$ over x



(Courtesy to FooPlot)

$$ax^2 + bx + c = 0$$

a = 3

b = 4

c = -1

$$x = -\frac{2}{3} \pm \frac{\sqrt{7}}{3}$$

$$x = 0.21525$$

$$x = -1.54858$$

Gradient Descent

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

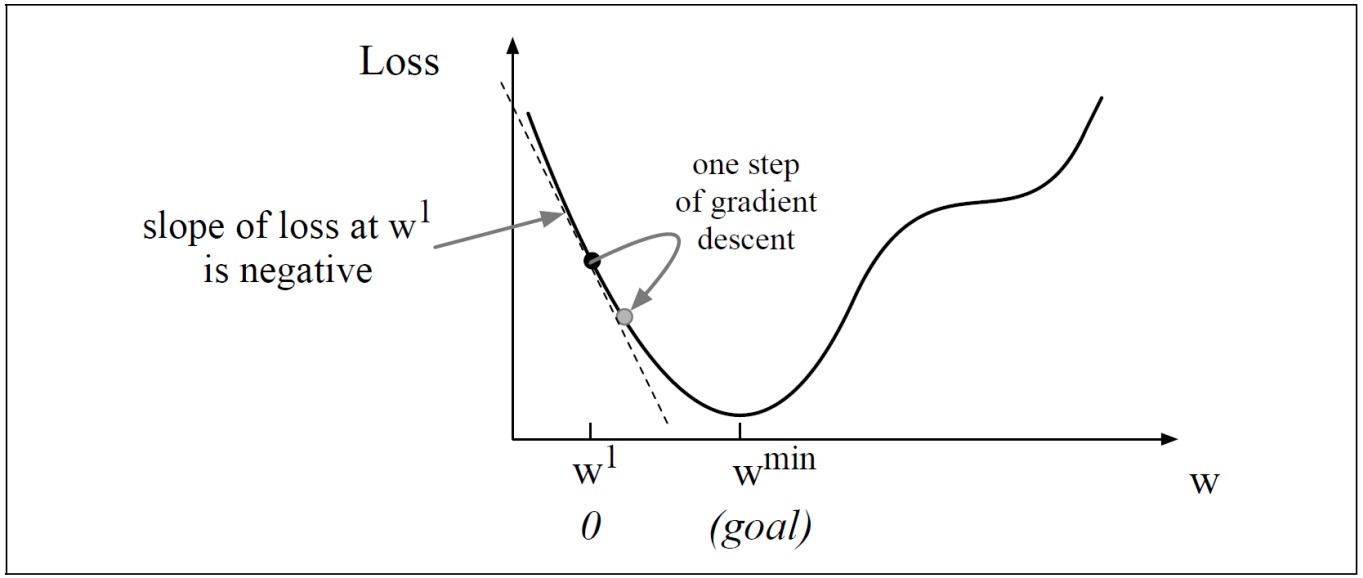


Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the initial value of w (which is 0), w^2 at the second step, and so on.

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$$

Bivariate Gradient

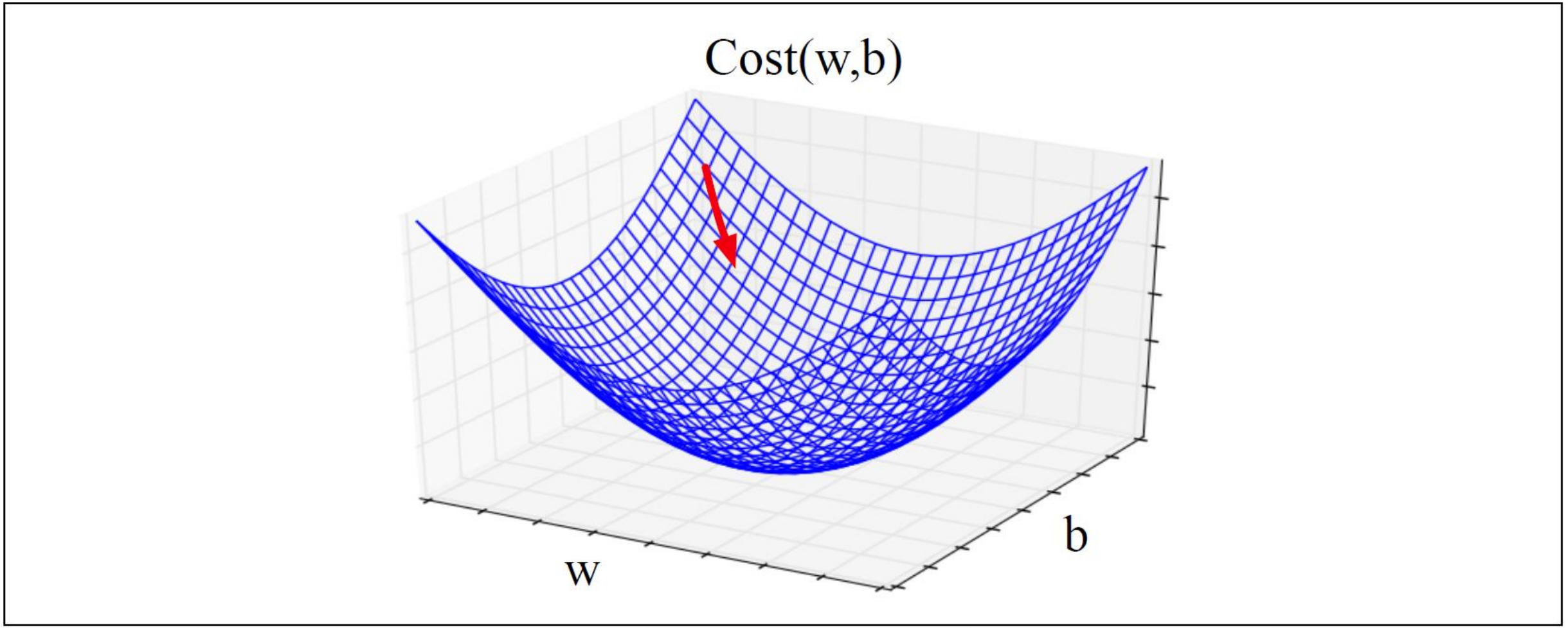


Figure 5.4 Visualization of the gradient vector in two dimensions *w* and *b*.

Using the Gradient

In an actual logistic regression, the parameter vector w is much longer than 1 or 2, since the input feature vector x can be quite long, and we need a weight w_i for each x_i. For each dimension/variable w_i in w (plus the bias b), the gradient will have a component that tells us the slope with respect to that variable. Essentially we're asking: "How much would a small change in that variable w_i influence the total loss function L?"

In each dimension w_i, we express the slope as a partial derivative $\frac{\partial}{\partial w_i}$ of the loss function. The gradient is then defined as a vector of these partials. We'll represent $\hat{y}$ as $f(x;\theta)$ to make the dependence on θ more obvious:

Updating using the Gradient

$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

The final equation for updating θ based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$$

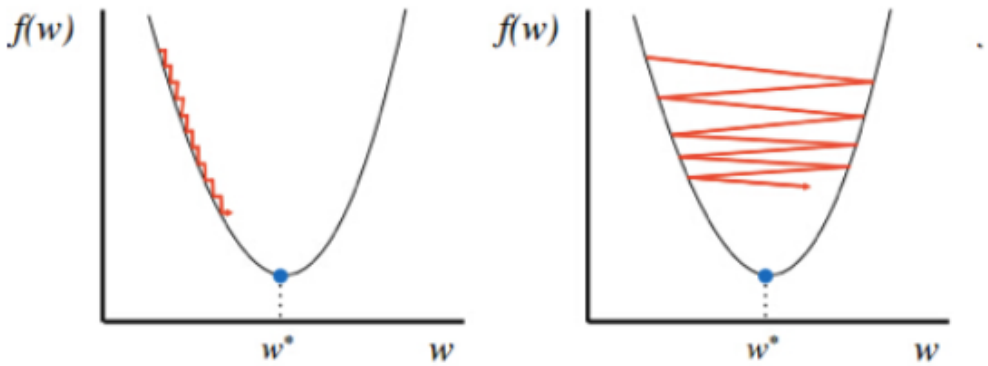
Gradient for Logistic Regression

$$L_{CE}(w,b) = -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))]$$

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

Learning Rate

- Updates: $\theta^{t+1} = \theta^t - \eta \frac{d}{d\theta} f(x; \theta)$
- Magnitude of movement along gradient
- Higher/faster learning rate = larger updates to parameters



Too small: converge very slowly

Too big: overshoot and even diverge

[Danqi Chen]

Stochastic Gradient Descent

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

```
    # where: L is the loss function
    #        f is a function parameterized by θ
    #        x is the set of training inputs x^(1), x^(2), ..., x^(n)
    #        y is the set of training outputs (labels) y^(1), y^(2), ..., y^(n)

θ ← 0
repeat til done    # see caption
  For each training tuple (x^(i), y^(i)) (in random order)
    1. Optional (for reporting):          # How are we doing on this tuple?
       Compute ŷ^(i) = f(x^(i); θ)        # What is our estimated output ŷ?
       Compute the loss L(ŷ^(i), y^(i))   # How far off is ŷ^(i)) from the true output y^(i)?
    2. g ← ∇_θ L(f(x^(i); θ), y^(i))      # How should we move θ to maximize loss?
    3. θ ← θ − η g                        # Go the other way instead
return θ
```

Figure 5.5 The stochastic gradient descent algorithm. Step 1 (computing the loss) is used to report how well we are doing on the current tuple. The algorithm can terminate when it converges (or when the gradient $< \varepsilon$), or when progress halts (for example when the loss starts going up on a held-out set).

Let's walk though a single step of the gradient descent algorithm. We'll use a simplified version of the example in Fig. 5.2 as it sees a single observation x, whose correct value is $y = 1$ (this is a positive review), and with only two features:

$$\begin{aligned} x_1 &= 3 \quad \text{(count of positive lexicon words)} \\ x_2 &= 2 \quad \text{(count of negative lexicon words)} \end{aligned}$$

Let's assume the initial weights and bias in θ^0 are all set to 0, and the initial learning rate η is 0.1:

$$\begin{aligned} w_1 = w_2 = b &= 0 \\ \eta &= 0.1 \end{aligned}$$

The single update step requires that we compute the gradient, multiplied by the learning rate

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$$

Example (Cont'd)

In our mini example there are three parameters, so the gradient vector has 3 dimensions, for w_1, w_2, and b. We can compute the first gradient as follows:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(w,b)}{\partial w_1} \\ \frac{\partial L_{CE}(w,b)}{\partial w_2} \\ \frac{\partial L_{CE}(w,b)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example (Cont'd)

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta^2 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

So after one step of gradient descent, the weights have shifted to be: $w_1 = .15$, $w_2 = .1$, and $b = .05$.

Regularization

To avoid overfitting, a new **regularization** term $R(\theta)$ is added to the objective function in Eq. 5.12, resulting in the following objective for a batch of m examples (slightly rewritten from Eq. 5.12 to be maximizing log probability rather than minimizing loss, and removing the $\frac{1}{m}$ term which doesn't affect the argmax):

$$\hat{\theta} = \operatorname*{argmax}_{\theta} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) - \alpha R(\theta) \tag{5.23}$$

L2 Regularization

compute this regularization term $R(\theta)$. **L2 regularization** is a quadratic function of the weight values, named because it uses the (square of the) L2 norm of the weight values. The L2 norm, $||\theta||_2$, is the same as the **Euclidean distance** of the vector θ from the origin. If θ consists of n weights, then:

$$R(\theta) = ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2 \tag{5.24}$$

The L2 regularized objective function becomes:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} \theta_j^2$$

L1 Regularization

L1 regularization is a linear function of the weight values, named after the L1 norm $||W||_1$, the sum of the absolute values of the weights, or **Manhattan distance** (the Manhattan distance is the distance you'd have to walk between two points in a city with a street grid like New York):

$$R(\theta) = ||\theta||_1 = \sum_{i=1}^{n} |\theta_i| \tag{5.26}$$

The L1 regularized objective function becomes:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |\theta_j|$$

Multinomial LR

- More than two classes

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq k$$

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{k} e^{z_i}}, \ldots, \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}} \right]$$

Using Softmax

The denominator $\sum_{i=1}^{k} e^{z_i}$ is used to normalize all the values into probabilities. Thus for example given a vector:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

the result softmax(z) is

$$[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

Sigmoid vs. Softmax

Again like the sigmoid, the input to the softmax will be the dot product between a weight vector w and an input vector x (plus a bias). But now we'll need separate weight vectors (and bias) for each of the K classes.

$$p(y=c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}} \tag{5.33}$$

Derivative of the Sigmoid

$$g_{\text{logistic}}(z) = \frac{1}{1+e^{-z}}$$

$$\begin{aligned}
g'_{\text{logistic}}(z) &= \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} (\text{by chain rule}) \\
&= \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\
&= \frac{1+e^{-z}}{(1+e^{-z})^2} - \left(\frac{1}{1+e^{-z}}\right)^2 \\
&= \frac{1}{(1+e^{-z})} - \left(\frac{1}{1+e^{-z}}\right)^2 \\
&= g_{\text{logistic}}(z) - g_{\text{logistic}}(z)^2 \\
&= g_{\text{logistic}}(z)(1 - g_{\text{logistic}}(z))
\end{aligned}$$

(double negation)

https://dustinstansbury.github.io/theclevermachine/derivation-common-neural-network-activation-functions

Derivative of tanh

$$
\begin{aligned}
g_{\tanh}(z) &= \frac{\sinh(z)}{\cosh(z)} \\
&= \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}
\end{aligned}
$$

$$
\begin{aligned}
g'_{\tanh}(z) &= \frac{\partial}{\partial z}\frac{\sinh(z)}{\cosh(z)} \\
&= \frac{\frac{\partial}{\partial z}\sinh(z)\times\cosh(z)-\frac{\partial}{\partial z}\cosh(z)\times\sinh(z)}{\cosh^{2}(z)} \\
&= \frac{\cosh^{2}(z)-\sinh^{2}(z)}{\cosh^{2}(z)} \\
&= 1-\frac{\sinh^{2}(z)}{\cosh^{2}(z)} \\
&= 1-\tanh^{2}(z)
\end{aligned}
$$

https://dustinstansbury.github.io/theclevermachine/derivation-common-neural-network-activation-functions

Logistic Regression - Summary

- Model

$$P(y = +|x) = \frac{\exp(\sum_{i=1}^{n} w_i x_i)}{1 + \exp(\sum_{i=1}^{n} w_i x_i)}$$

- Inference

$\text{argmax}_y P(y|x)$ fundamentally same as Naive Bayes

$P(y = 1|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$

- Learning: gradient ascent on the (regularized) discriminative log-likelihood

[Greg Durrett]

Summary

- Logistic regression is a supervised machine learning classifier that extracts real-valued features from the input, multiplies each by a weight, sums them, and passes the sum through a **sigmoid** function to generate a probability. A threshold is used to make a decision.
- Logistic regression can be used with two classes (e.g., positive and negative sentiment) or with multiple classes (**multinomial logistic regression**, for example for n-ary text classification, part-of-speech labeling, etc.).
- Multinomial logistic regression uses the **softmax** function to compute probabilities.
- The weights (vector w and bias b) are learned from a labeled training set via a loss function, such as the **cross-entropy loss**, that must be minimized.
- Minimizing this loss function is a **convex optimization** problem, and iterative algorithms like **gradient descent** are used to find the optimal weights.
- **Regularization** is used to avoid overfitting.
- Logistic regression is also one of the most useful analytic tools, because of its ability to transparently study the importance of individual features.

Summary of Learning Methods

It is natural to ask which learning algorithm is best, but the answer depends on what characteristics are important to the problem you are trying to solve.

Naïve Bayes *Pros*: easy to implement; estimation is fast, requiring only a single pass over the data; assigns probabilities to predicted labels; controls overfitting with smoothing parameter. *Cons*: often has poor accuracy, especially with correlated features.

Perceptron *Pros*: easy to implement; online; error-driven learning means that accuracy is typically high, especially after averaging. *Cons*: not probabilistic; hard to know when to stop learning; lack of margin can lead to overfitting.

Support vector machine *Pros*: optimizes an error-based metric, usually resulting in high accuracy; overfitting is controlled by a regularization parameter. *Cons*: not probabilistic.

Logistic regression *Pros*: error-driven and probabilistic; overfitting is controlled by a regularization parameter. *Cons*: batch learning requires black-box optimization; logistic loss can "overtrain" on correctly labeled examples.

Non-linear Learning Methods

- **Kernel methods** are generalizations of the **nearest-neighbor** classification rule, which classifies each instance by the label of the most similar example in the training set. The application of the **kernel support vector machine** to information extraction is described in chapter 17.
- **Decision trees** classify instances by checking a set of conditions. Scaling decision trees to bag-of-words inputs is difficult, but decision trees have been successful in problems such as coreference resolution (chapter 15), where more compact feature sets can be constructed (Soon et al., 2001).
- **Boosting** and related **ensemble methods** work by combining the predictions of several "weak" classifiers, each of which may consider only a small subset of features. Boosting has been successfully applied to text classification (Schapire and Singer, 2000) and syntactic analysis (Abney et al., 1999), and remains one of the most successful methods on machine learning competition sites such as Kaggle (Chen and Guestrin, 2016).

NLP

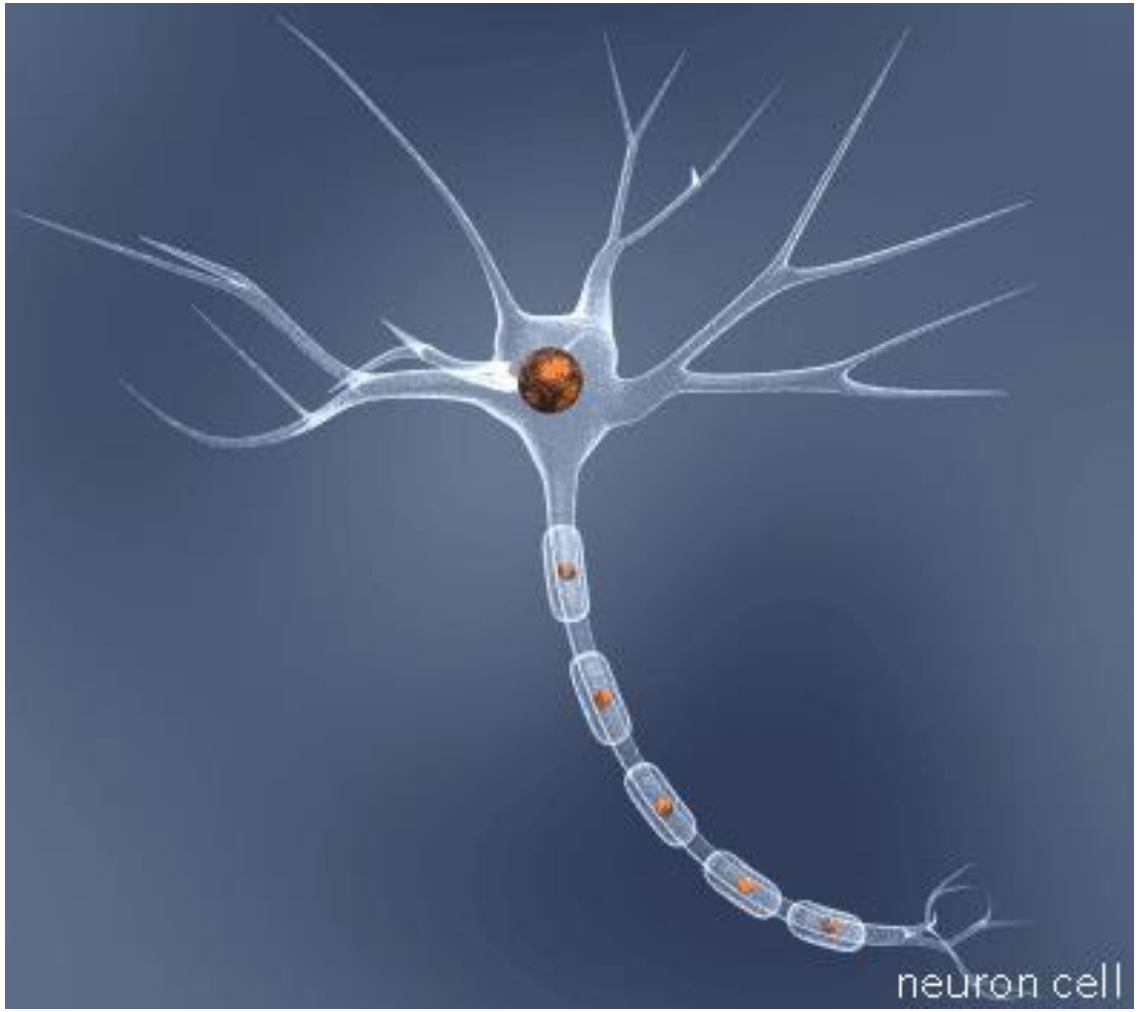
Introduction to NLP

515.

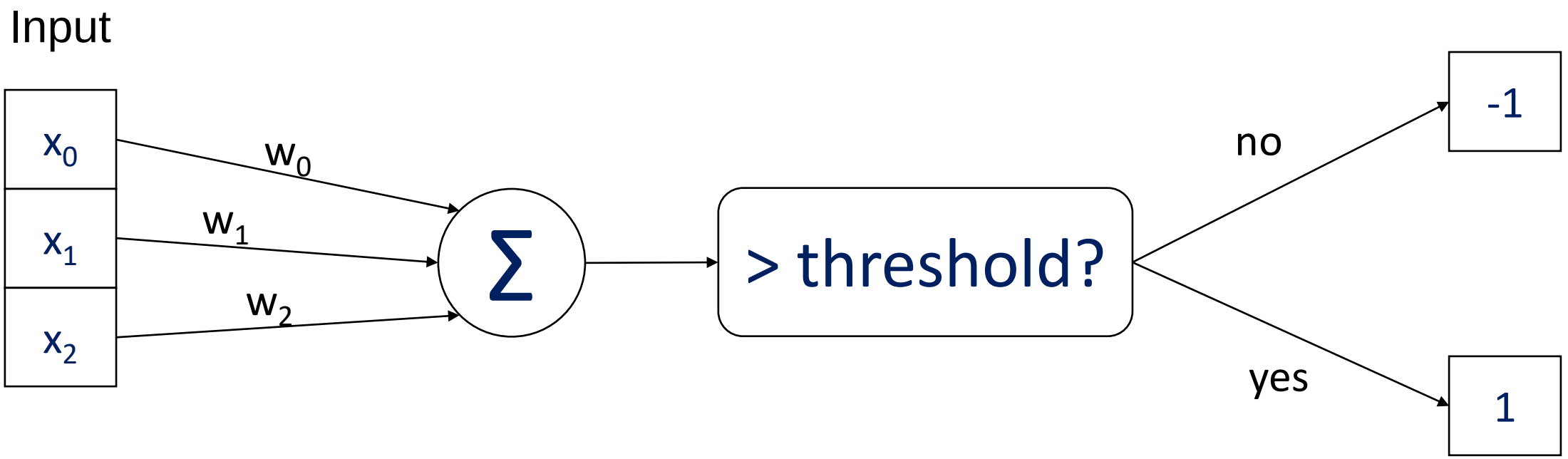
Perceptrons

The Perceptron

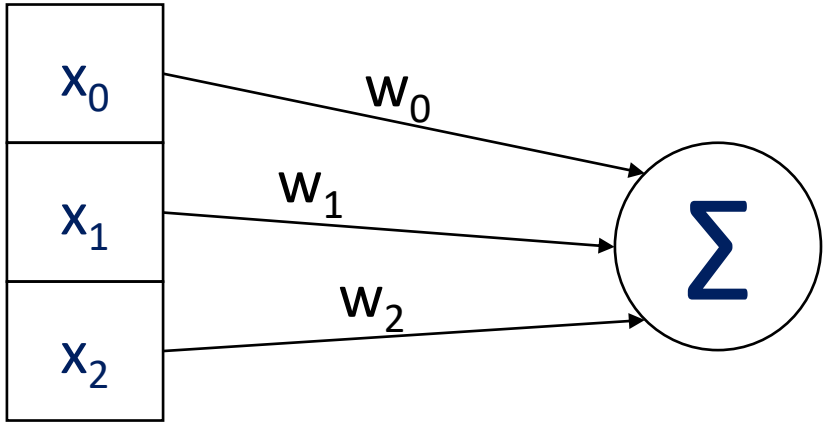
- A simple but very important (discriminative) classifier
- Model of a neuron
  - Input excitations
  - If excitation > inhibition, send an electrical signal out the axon
- Earliest neural network
  - invented in 1957 by Frank Rosenblatt at Cornell



Perceptron Idea



So we can rewrite



- as a dot product of two vectors

$$\vec{w} \cdot \vec{x}$$

NLP