# CS388: Natural Language Processing

## Lecture 2: Binary Classification

Greg Durrett
TEXAS
The University of Texas at Austin


credit: Machine Learning Memes on Facebook

# Administrivia

- P1 autograders released soon (P1 due January 26)
- Recordings on Canvas

# This Lecture

- Linear binary classification fundamentals

- Feature extraction

- Logistic regression

- Perceptron/SVM

- Optimization

- Sentiment analysis

# Linear Binary Classification

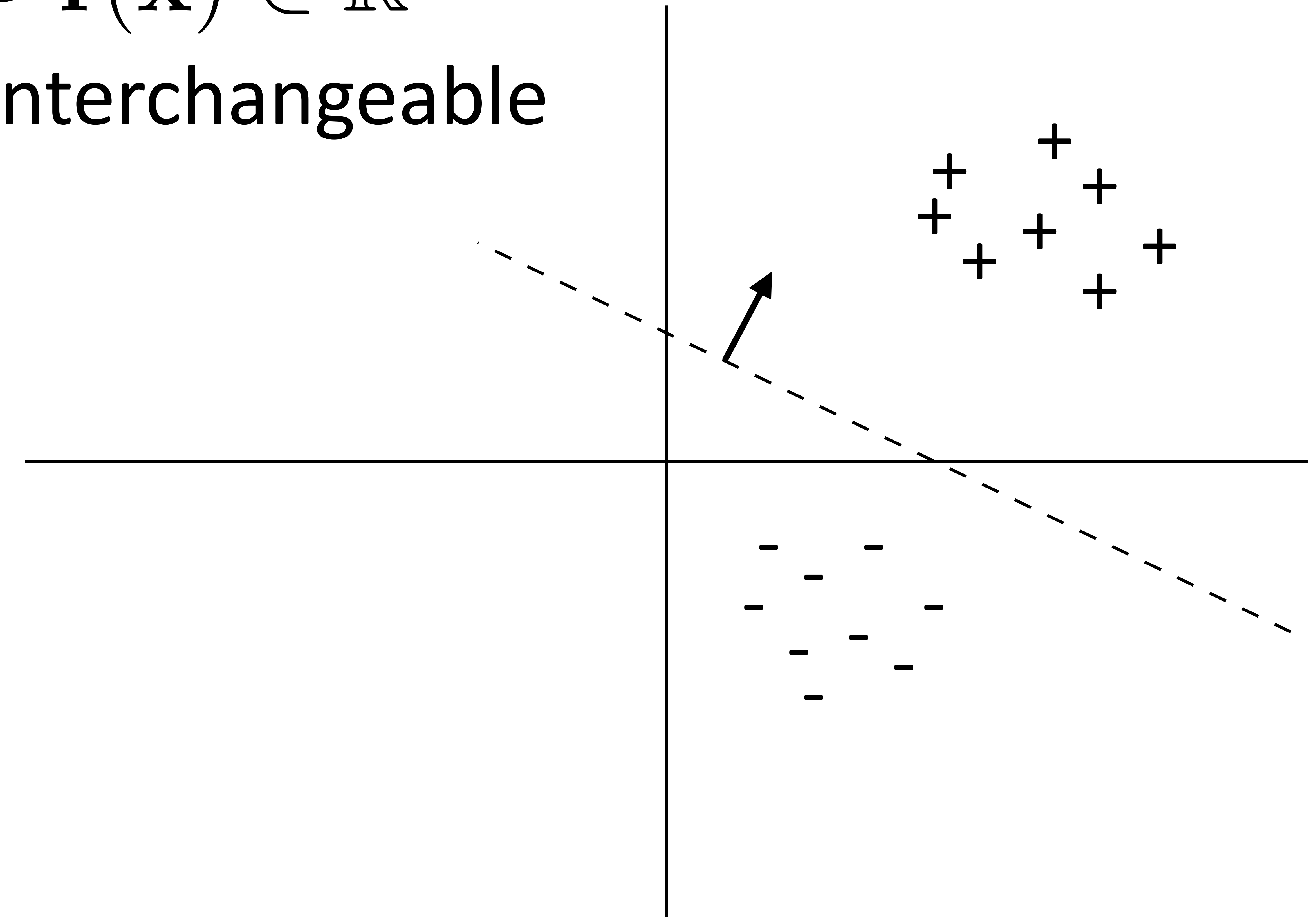
# Classification

- Datapoint $\mathbf{x}$ with label $y \in \{0, 1\}$

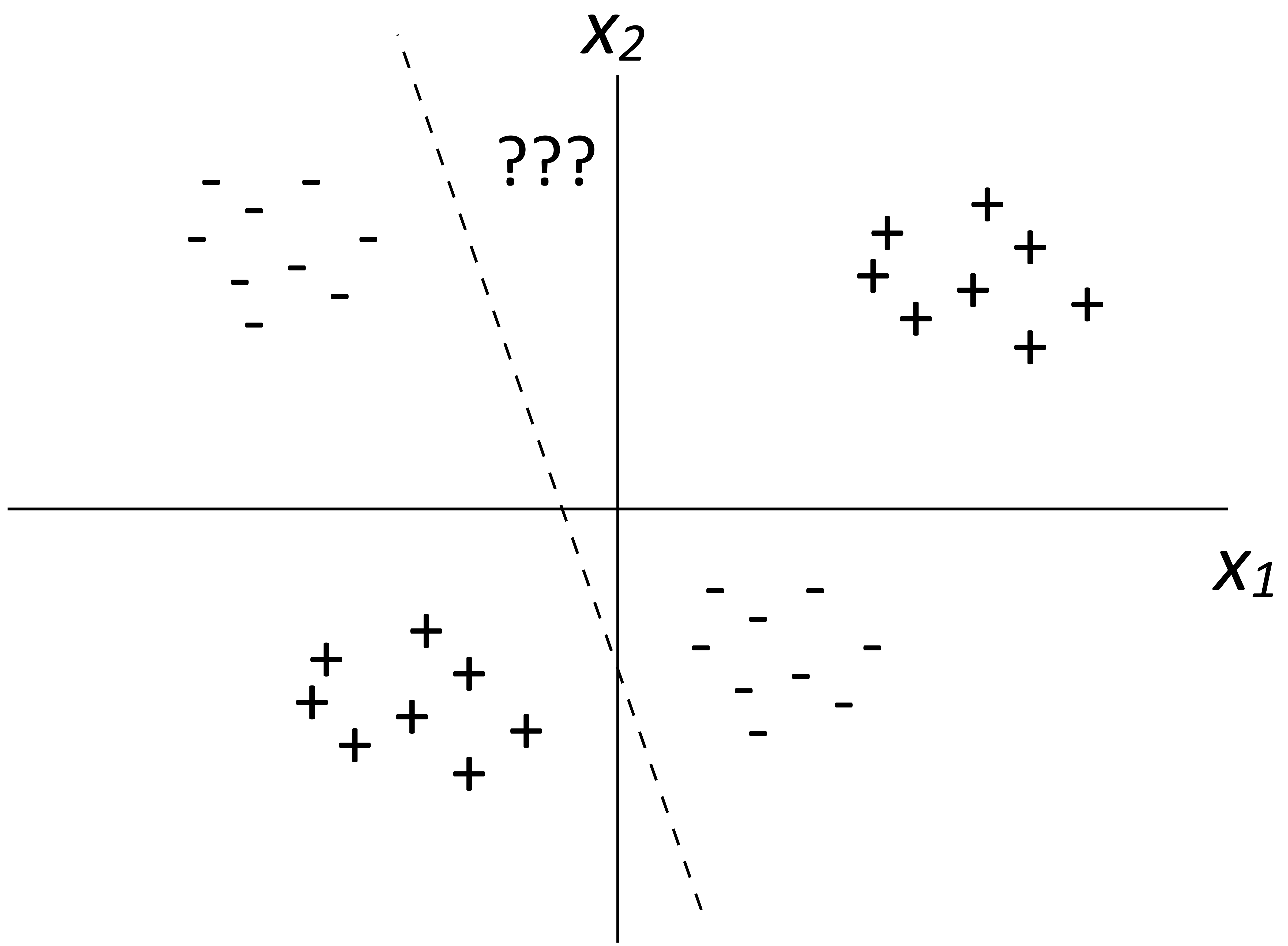
- Embed datapoint in a feature space $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^n$
  but in this lecture $\mathbf{f}(\mathbf{x})$ and $\mathbf{x}$ are interchangeable

- Linear decision rule: $\mathbf{w}^\top \mathbf{f}(\mathbf{x}) > 0$

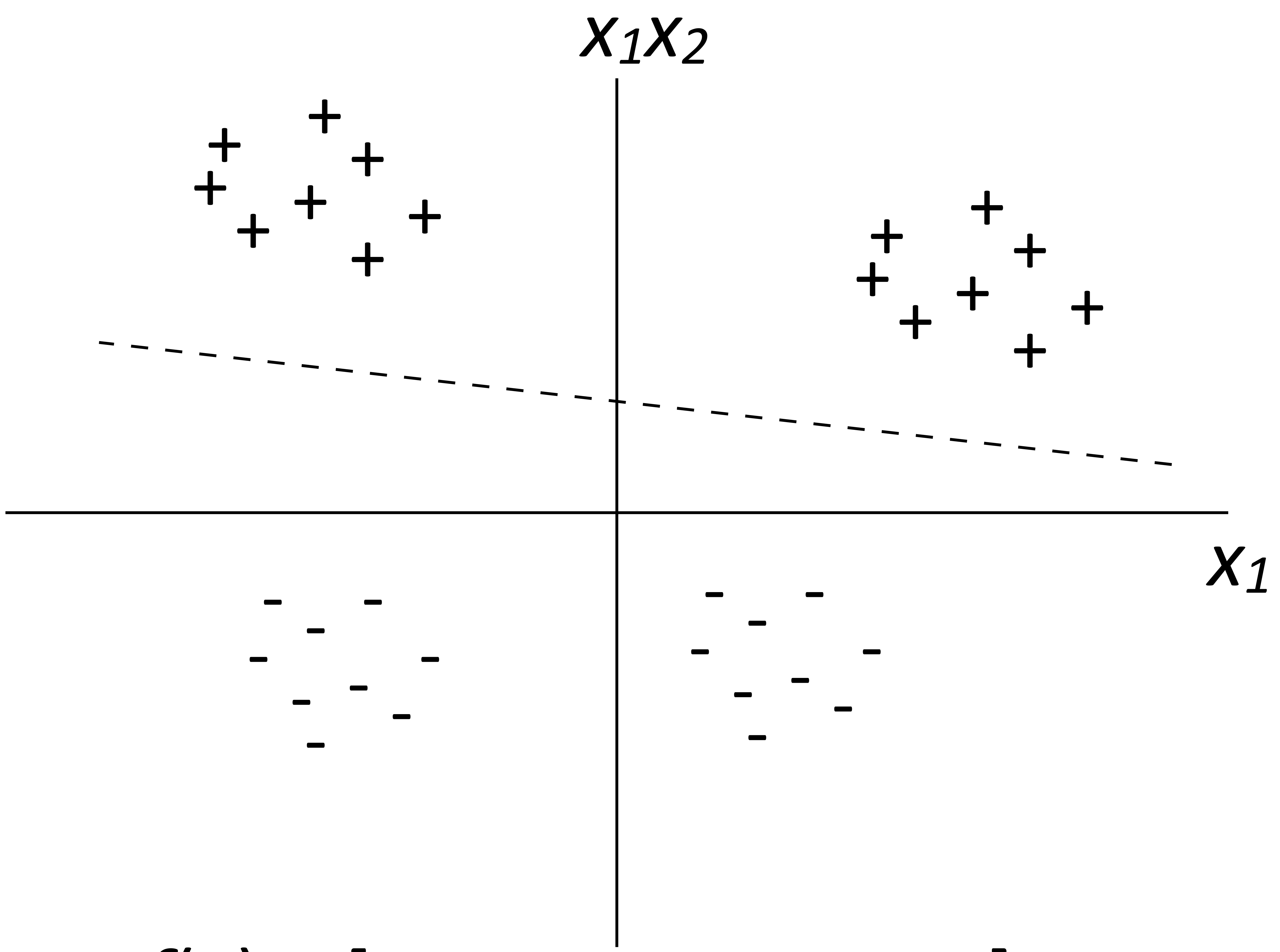
  (No bias term *b* — we have lots of features and it isn't needed)



# Linear functions are powerful!




$\boldsymbol{f}(\boldsymbol{x}) = [x_1, x_2]$




$\boldsymbol{f}(\boldsymbol{x}) = [x_1, x_2, x_1^2, x_2^2, x_1x_2]$

- “Kernel trick” does this for “free,” but is too expensive to use; with *n* examples training is $O(n^2)$ instead of $O(n \cdot (\mathrm{num\ feats}))$

# Classification: Sentiment Analysis

*this movie was great! would watch again* Positive

*that film was awful, I'll never watch again* Negative

- Surface cues can basically tell you what's going on here: presence or absence of certain words (*great*, *awful*)
- Steps to classification:
  - Turn examples like this into feature vectors
  - Pick a model / learning algorithm
  - Train weights on data to get our classifier

# Feature Extraction

# Feature Representation

*this movie was great! would watch again* Positive

- Convert this example to a vector using *bag-of-words features*

| [contains *the*] | [contains *a*] | [contains *was*] | [contains *movie*] | [contains *film*] | … |
|---|---|---|---|---|---|
| position 0 | position 1 | position 2 | position 3 | position 4 | |
| ***f***(***x***) = [0 | 0 | 1 | 1 | 0 | … |

- Very large vector space (size of vocabulary), sparse features (how many per example?)

# Feature Representation

- What are some preprocessing operations we might want to do before we map to words?

# Feature Extraction Details

- Tokenization:

  *“I thought it wasn’t that great!” critics complained.*

  *“ I thought it was n’t that great ! ” critics complained .*

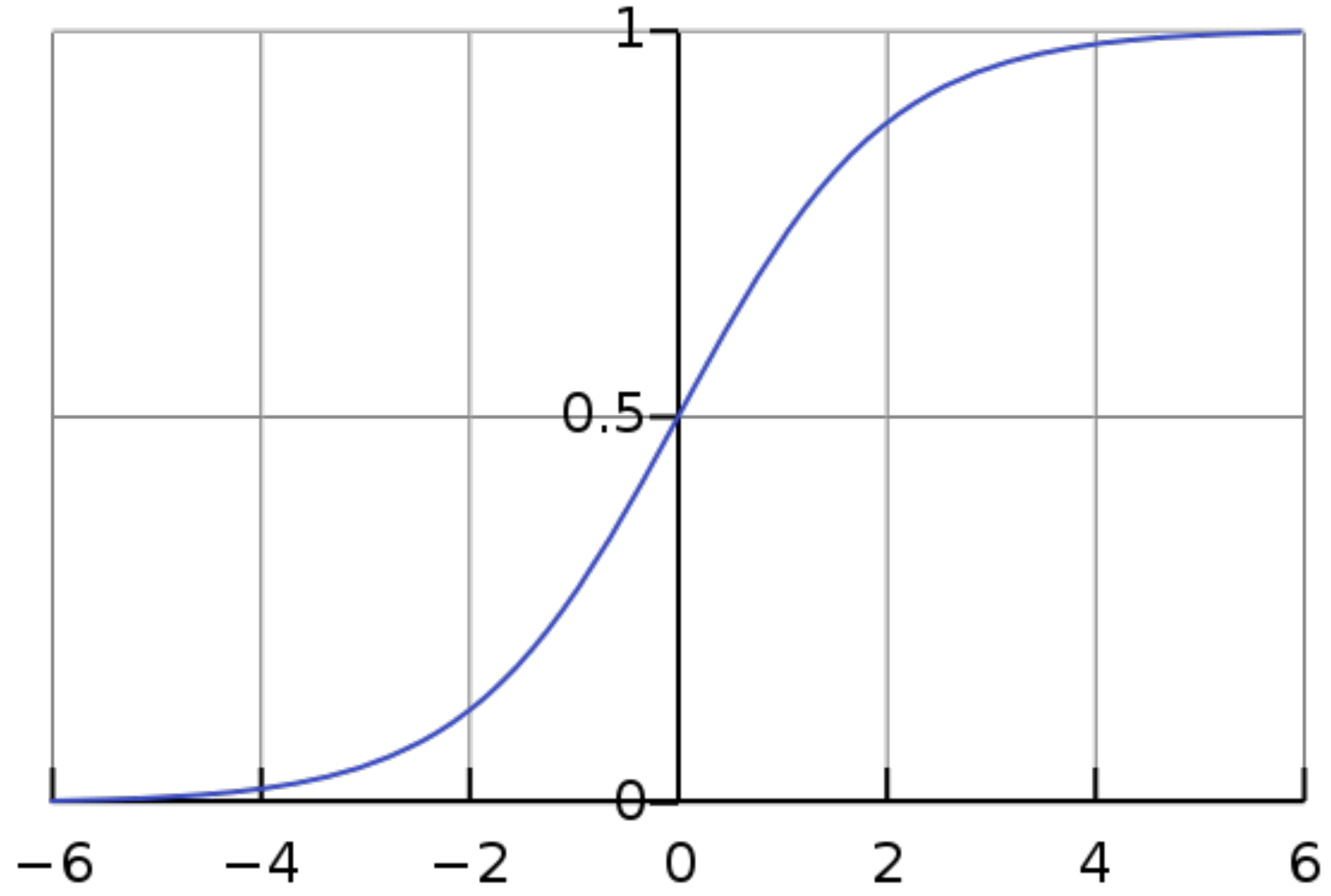
  - Split out punctuation, contractions; handle hyphenated compounds
- Lowercasing (maybe)
- Filtering stopwords (maybe)
- Buildings the feature vector requires *indexing* the features (mapping them to axes). Store an invertible map from string -> index
  - [contains “the”] is a single feature — put this whole bracketed thing into the indexer to give it a position in the feature space

# Logistic Regression

# Logistic Regression

$$P(y = +|x) = \text{logistic}(w^\top x)$$

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




- To learn weights: maximize discriminative log likelihood of data (log P(y|x))

$$\mathcal{L}(\{x_j, y_j\}_{j=1,\ldots,n}) = \sum_j \log P(y_j|x_j)$$ corpus-level LL

$$\mathcal{L}(x_j, y_j = +) = \log P(y_j = +|x_j)$$ one (positive) example LL

$$= \sum_{i=1}^n w_i x_{ji} - \log\left(1 + \exp\left(\sum_{i=1}^n w_i x_{ji}\right)\right)$$

sum over features

# Logistic Regression

$$\mathcal{L}(x_j, y_j = +) = \log P(y_j = +|x_j) = \sum_{i=1}^{n} w_i x_{ji} - \log\left(1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)\right)$$

$$\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w_i} = x_{ji} - \frac{\partial}{\partial w_i} \log\left(1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)\right)$$

$$= x_{ji} - \frac{1}{1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)} \frac{\partial}{\partial w_i}\left(1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)\right)$$

deriv of log

$$= x_{ji} - \frac{1}{1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)} x_{ji} \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)$$

deriv of exp

$$= x_{ji} - x_{ji} \frac{\exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)}{1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)} = x_{ji}(1 - P(y_j = +|x_j))$$

# Logistic Regression

- Update for ***w*** on positive example $= \mathbf{x}(1 - P(y = + \mid \mathbf{x}))$ (gradient with step size = 1)

  If P(+ | ***x***) is close to 1, make very little update

  Otherwise make ***w*** look more like ***x***, which will increase P(+ | ***x***)

- Update for ***w*** on negative example $= \mathbf{x}(-P(y = + \mid \mathbf{x}))$

  If P(+ | ***x***) is close to 0, make very little update

  Otherwise make *w* look less like ***x***, which will decrease P(+ | ***x***)

- Let *y* = 1 for positive instances, *y* = 0 for negative instances.
- Can combine these updates as $\mathbf{x}(y - P(y = 1 \mid \mathbf{x}))$

# Example

(1) *this movie was great! would watch again* +

(2) *I expected a great movie and left happy* +

(3) *great potential but ended up being a flop* —

$f(x_1)$ = [1 1]

$f(x_2)$ = [1 1]

$f(x_3)$ = [1 0]

[contains *great*] [contains *movie*]

position 0 position 1

**w** = [0, 0] ⟶ $P(y = 1 \mid \mathbf{x}_1)$ = exp(0)/(1 + exp(0)) = 0.5 ⟶ $g$ = [0.5, 0.5]

**w** = [0.5, 0.5] ⟶ $P(y = 1 \mid \mathbf{x}_2)$ = logistic(1) ≈ 0.75 ⟶ $g$ = [0.25, 0.25]

**w** = [0.75, 0.75] ⟶ $P(y = 1 \mid \mathbf{x}_3)$ = logistic(0.75) ≈ 0.67 ⟶ $g$ = [-0.67, 0]

**w** = [0.08, 0.75] …

$$P(y = +|x) = \text{logistic}(w^\top x)$$

pos upd: $\mathbf{x}(1 - P(y = + \mid \mathbf{x}))$

neg upd: $\mathbf{x}(-P(y = + \mid \mathbf{x}))$

# Regularization

- Regularizing an objective can mean many things, including an L2-norm penalty to the weights:

$$\sum_{j=1}^{m} \mathcal{L}(x_j, y_j) - \lambda \|w\|_2^2$$

- Keeping weights small can prevent overfitting
- For most of the NLP models we build, explicit regularization isn't necessary
  - We always stop early before full convergence
  - Large numbers of sparse features are hard to overfit in a really bad way
  - For neural networks: dropout and gradient clipping

# 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

$$\text{argmax}_y P(y|x)$$

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

- Learning: gradient ascent on the (regularized) discriminative log-likelihood. Same interpretation as gradient descent on log-loss (in a few slides)

# Perceptron/SVM

# Perceptron

- Simple error-driven learning approach similar to logistic regression

- Decision rule: $\mathbf{w}^\top \mathbf{f}(\mathbf{x}) > 0$
  - If incorrect: if positive, $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x})$

    if negative, $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{f}(\mathbf{x})$

Logistic Regression

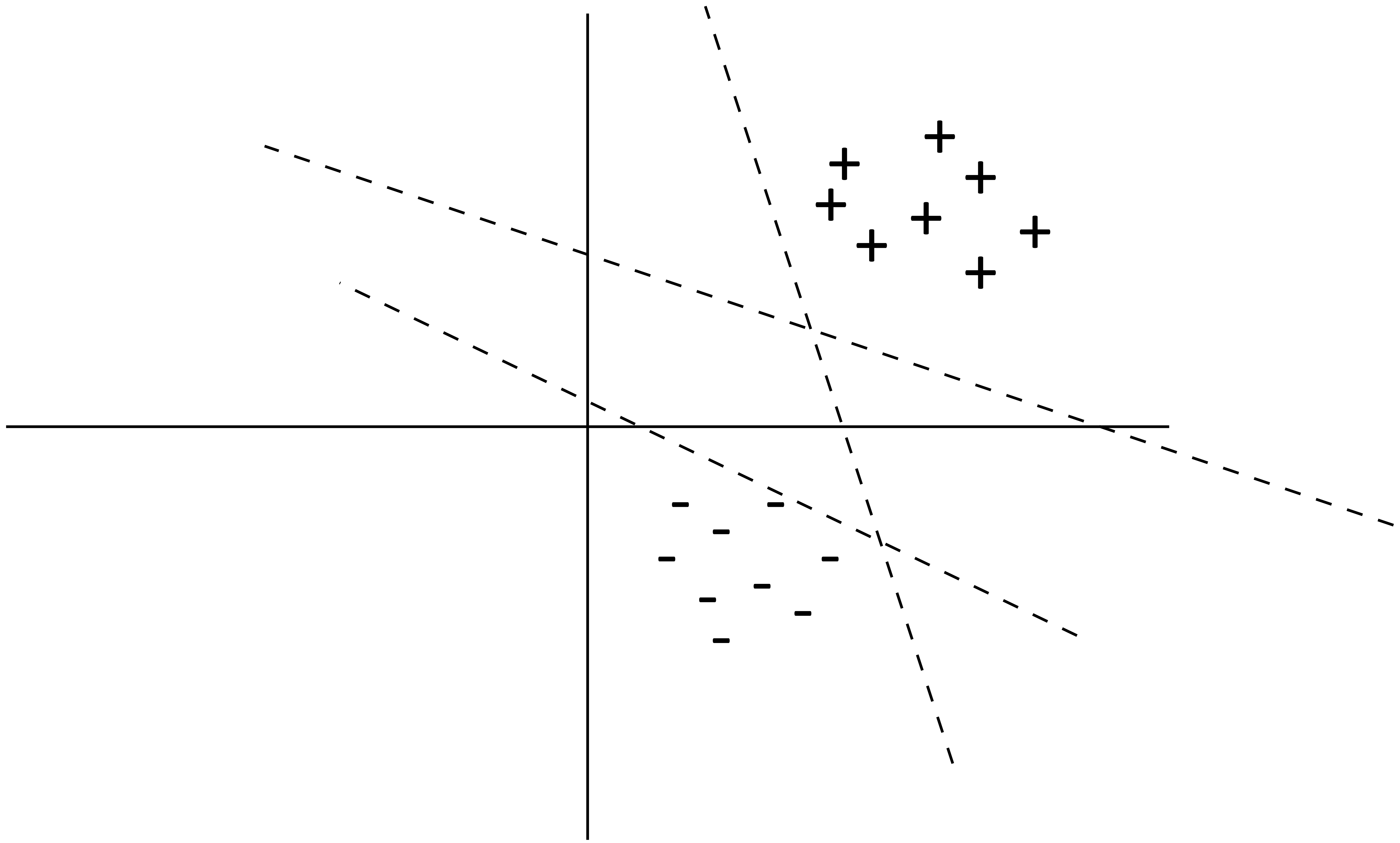
$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x})(1 - P(y = + \mid \mathbf{x}))$$

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{f}(\mathbf{x})P(y = + \mid \mathbf{x})$$

- Guaranteed to eventually separate the data if the data are separable

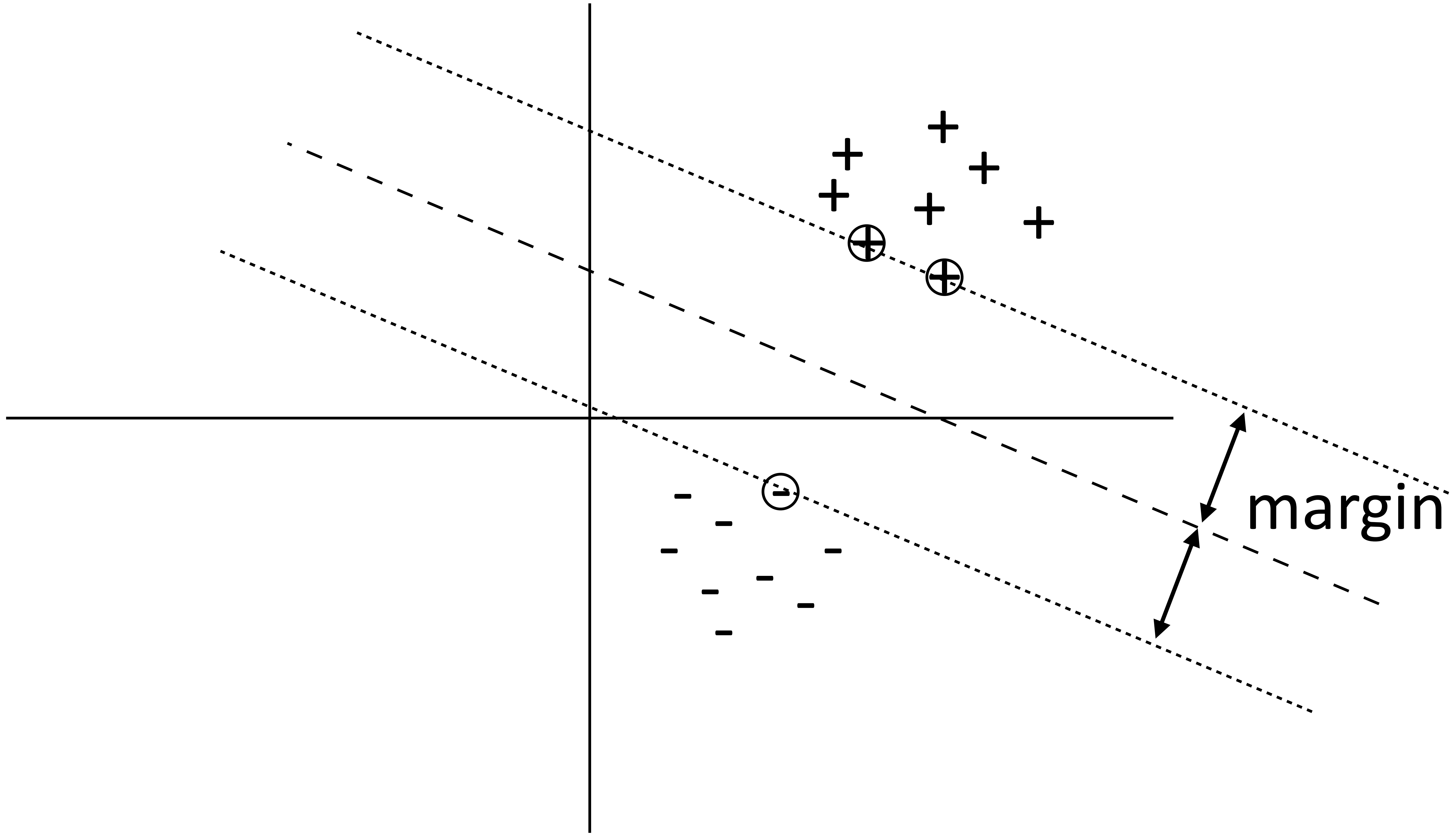
# Support Vector Machines

- Many separating hyperplanes — is there a best one?



# Support Vector Machines

- Many separating hyperplanes — is there a best one?




- Max-margin hyperplane found by SVMs

# Perceptron and Logistic Losses

- Throughout this course: view classification as *minimizing loss*

- Let's focus on loss of a positive example

  - Perceptron: loss = $\begin{cases} 0 & \text{if } \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}) > 0 \\ -\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}) & \text{if } \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}) < 0 \end{cases}$

    Take the gradient: no update if $\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}) > 0$, else update with $+\boldsymbol{f}(\boldsymbol{x})$)

  - Logistic regression: loss = — log $P(+|\boldsymbol{x})$

    (maximizing log likelihood = minimizing negative log likelihood)

# Gradient Updates on Positive Examples

**Logistic regression**

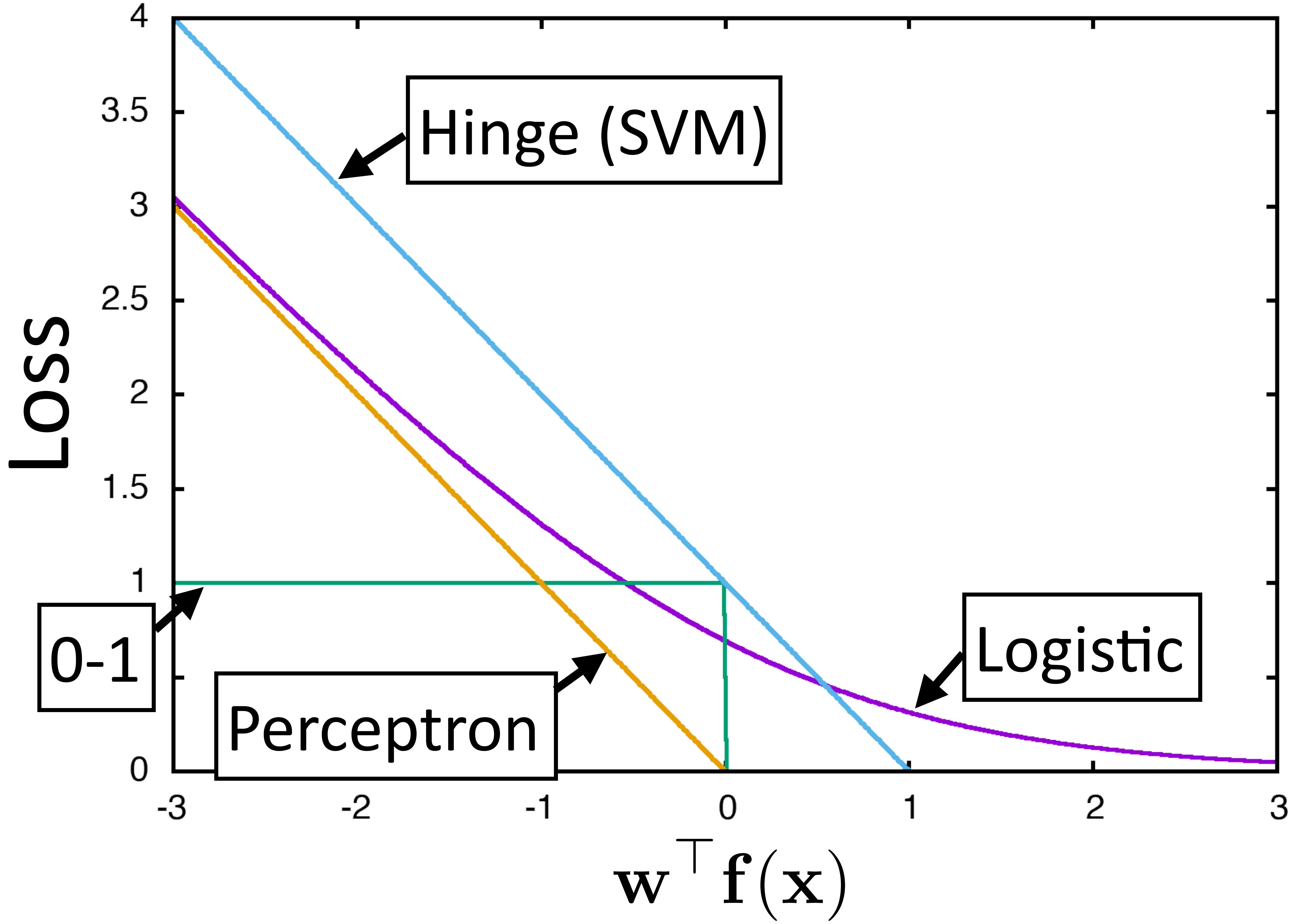
$$\mathbf{f}(\mathbf{x})(1 - \text{logistic}(\mathbf{w}^\top \mathbf{f}(\mathbf{x}))$$

**Perceptron**

$$\mathbf{f}(\mathbf{x}) \text{ if } \mathbf{w}^\top \mathbf{f}(\mathbf{x}) < 0, \text{ else } 0$$

**SVM (ignoring regularizer)**

$$\mathbf{f}(\mathbf{x}) \text{ if } \mathbf{w}^\top \mathbf{f}(\mathbf{x}) < 1, \text{ else } 0$$

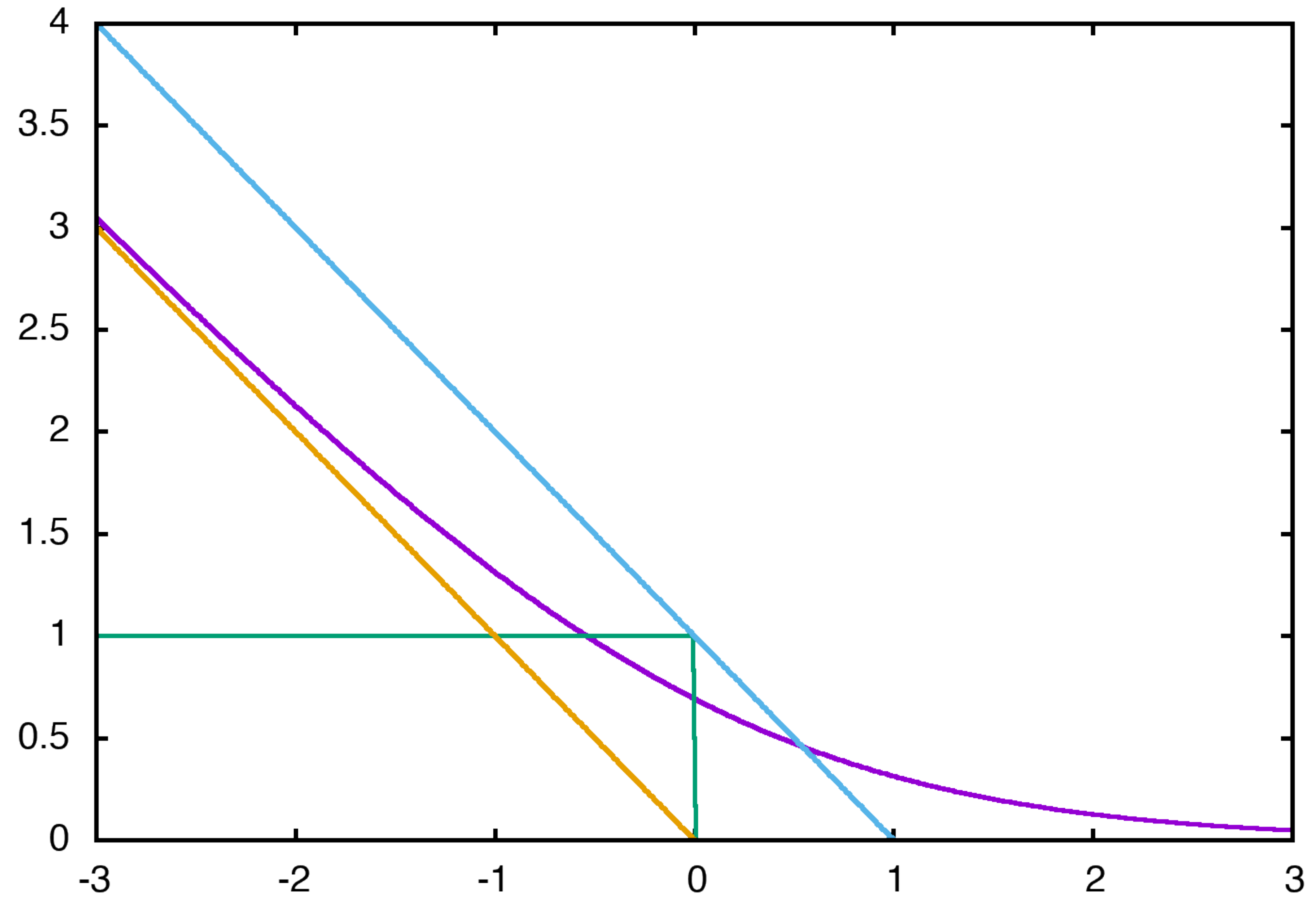



*sign of gradients flipped to give intuitive update

# Optimization

# Statistical Modeling

- Four elements of a structured machine learning method:
  - Model: probabilistic, max-margin, deep neural network
  - Objective




  - Inference: just maxes and simple expectations so far, but there can be other questions too (e.g. posterior over a variable)
  - Optimization: **gradient descent**

# Optimization

- Stochastic gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g} \qquad \mathbf{g} = \frac{\partial}{\partial \mathbf{w}} \mathcal{L}$$

  - Very simple to code up
  - "First-order" technique: only relies on having gradient
  - Can avg gradient over a few examples and apply update once (minibatch)
  - Setting step size is hard (decrease when held-out performance worsens?)
- Newton's method

$$\mathbf{w} \leftarrow \mathbf{w} - \left( \frac{\partial^2}{\partial \mathbf{w}^2} \mathcal{L} \right)^{-1} \mathbf{g}$$

  - Second-order technique
  - Optimizes quadratic instantly

Inverse Hessian: *n* x *n* mat, expensive!

- Quasi-Newton methods: L-BFGS, etc. approximate inverse Hessian

# AdaGrad

- Optimized for problems with sparse features

- Per-parameter learning rate: smaller updates are made to parameters that get updated frequently

$$w_i \leftarrow w_i + \alpha \frac{1}{\sqrt{\epsilon + \sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t_i}$$

(smoothed) sum of squared gradients from all updates

- Generally more robust than SGD, requires less tuning of learning rate

- Other techniques for optimizing deep models — more later!

Duchi et al. (2011)

# Implementation

- Supposing *k* active features on an instance, gradient is only nonzero on *k* dimensions

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g} \qquad \mathbf{g} = \frac{\partial}{\partial \mathbf{w}} \mathcal{L}$$

- *k* < 100, total num features = 1M+ on many problems
- Be smart about applying updates!
- In PyTorch: applying sparse gradients only works for certain optimizers and sparse updates are very slow.

# Sentiment Analysis

# Sentiment Analysis

*this movie was great! would watch again* +

*the movie was gross and overwrought, but I liked it* +

*this movie was not really very enjoyable* —

- Bag-of-words doesn't seem sufficient (discourse structure, negation)
- There are some ways around this: extract bigram feature for "*not* X" for all X following the *not*

Bo Pang, Lillian Lee, Shivakumar Vaithyanathan (2002)

# Sentiment Analysis

| | Features | # of features | frequency or presence? | NB | ME | SVM |
|---|---|---|---|---|---|---|
| (1) | unigrams | 16165 | freq. | **78.7** | N/A | 72.8 |
| (2) | unigrams | " | pres. | 81.0 | 80.4 | **82.9** |
| (3) | unigrams+bigrams | 32330 | pres. | 80.6 | 80.8 | **82.7** |
| (4) | bigrams | 16165 | pres. | 77.3 | **77.4** | 77.1 |
| (5) | unigrams+POS | 16695 | pres. | 81.5 | 80.4 | **81.9** |
| (6) | adjectives | 2633 | pres. | 77.0 | **77.7** | 75.1 |
| (7) | top 2633 unigrams | 2633 | pres. | 80.3 | 81.0 | **81.4** |
| (8) | unigrams+position | 22430 | pres. | 81.0 | 80.1 | **81.6** |

- Simple feature sets can do pretty well!

Bo Pang, Lillian Lee, Shivakumar Vaithyanathan (2002)

# Sentiment Analysis

| Method | RT-s | MPQA |
|---|---|---|
| MNB-uni | 77.9 | 85.3 |
| MNB-bi | **79.0** | **86.3** |
| SVM-uni | 76.2 | 86.1 |
| SVM-bi | 77.7 | **86.7** |
| NBSVM-uni | **78.1** | 85.3 |
| NBSVM-bi | **79.4** | **86.3** |
| RAE | 76.8 | 85.7 |
| RAE-pretrain | 77.7 | **86.4** |
| Voting-w/Rev. | 63.1 | 81.7 |
| Rule | 62.9 | 81.8 |
| BoF-noDic. | 75.7 | 81.8 |
| BoF-w/Rev. | 76.4 | 84.1 |
| Tree-CRF | 77.3 | 86.1 |
| BoWSVM | – | – |
| **Kim (2014) CNNs** | **81.5** | **89.5** |

Naive Bayes is doing well!

Ng and Jordan (2002) — NB can be better for small data

Before neural nets had taken off — results weren't that great

Wang and Manning (2012)

# Sentiment Analysis

- Stanford Sentiment Treebank (SST) binary classification
- Best systems now: large pretrained networks
- 90 -> 97 with good NN models

| Model | Accuracy | Paper / Source | Code |
|---|---|---|---|
| XLNet-Large (ensemble) (Yang et al., 2019) | 96.8 | XLNet: Generalized Autoregressive Pretraining for Language Understanding | Official |
| MT-DNN-ensemble (Liu et al., 2019) | 96.5 | Improving Multi-Task Deep Neural Networks via Knowledge Distillation for Natural Language Understanding | Official |
| Snorkel MeTaL(ensemble) (Ratner et al., 2018) | 96.2 | Training Complex Models with Multi-Task Weak Supervision | Official |
| MT-DNN (Liu et al., 2019) | 95.6 | Multi-Task Deep Neural Networks for Natural Language Understanding | Official |
| Bidirectional Encoder Representations from Transformers (Devlin et al., 2018) | 94.9 | BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding | Official |
| ... | | | |
| Neural Semantic Encoder (Munkhdalai and Yu, 2017) | 89.7 | Neural Semantic Encoders | |
| BLSTM-2DCNN (Zhou et al., 2017) | 89.5 | Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling | |

https://github.com/sebastianruder/NLP-progress/blob/master/english/sentiment_analysis.md

# Takeaways

- Logistic regression, SVM, and perceptron are closely related; we'll use logistic regression mostly, but the exact loss function doesn't matter much in practice

- All gradient updates: "make it look more like the right thing and less like the wrong thing"

- Next time: multiclass classification