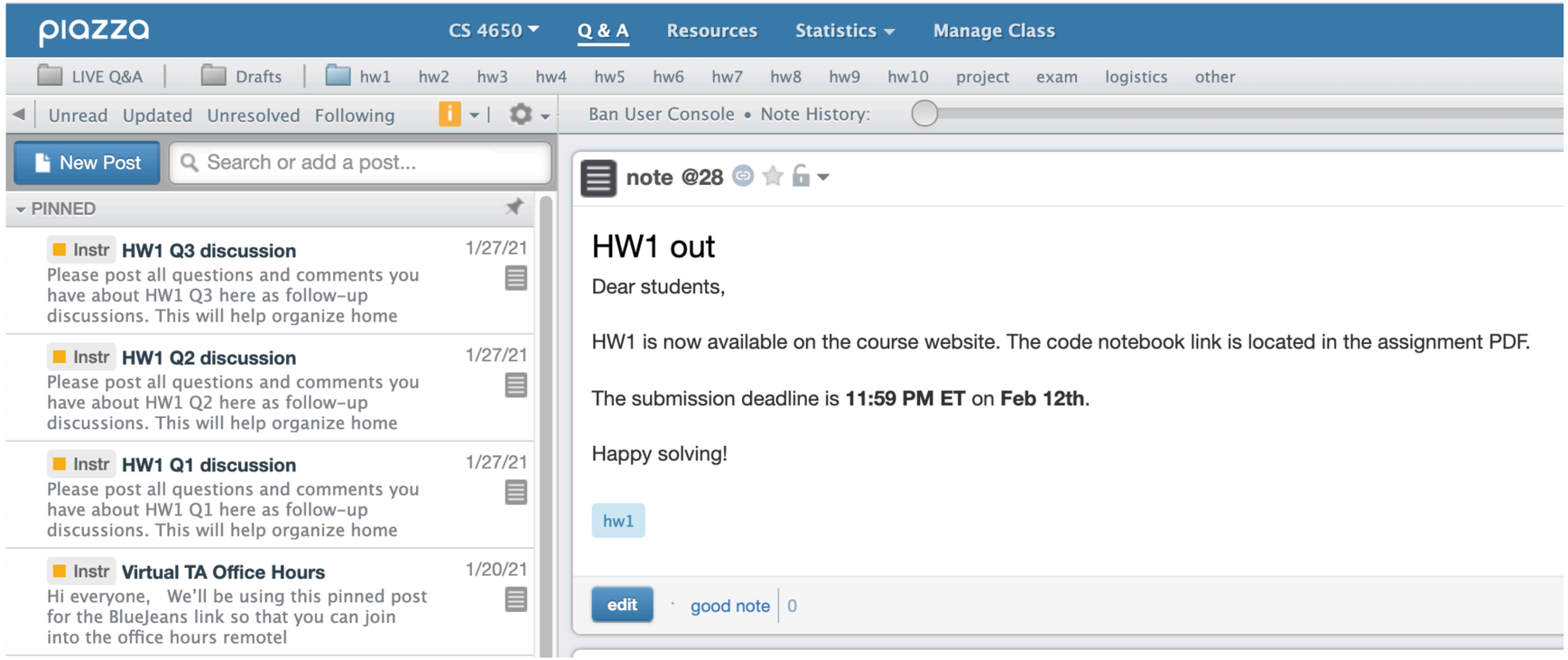
# Binary Classification

Wei Xu

(many slides from Greg Durrett and Vivek Srikumar)

# Administrivia

- Homework 1 is released (due 2/12)
  - Two written questions
  - One programming tast: Logistic Regression for Text Classification (Hate Speech)



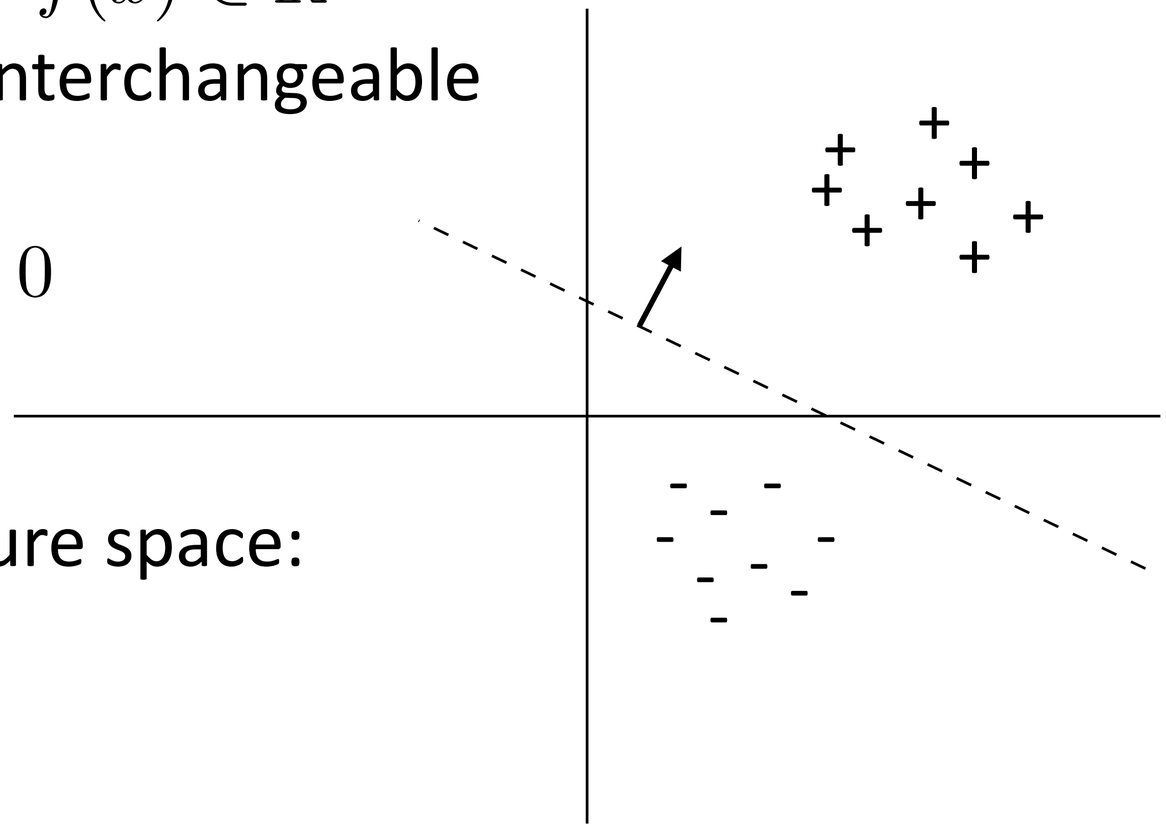
# This Lecture

- Linear classification fundamentals
- (Monday) Naive Bayes, maximum likelihood in generative models
- (Today) Three discriminative models: logistic regression, perceptron, SVM
  - Different motivations but very similar update rules / inference!

# Classification (recap)

- Datapoint $x$ with label $y \in \{0, 1\}$
- Embed datapoint in a feature space $f(x) \in \mathbb{R}^n$
  but in this lecture $f(x)$ and $x$ are interchangeable
- Linear decision rule: $w^\top f(x) + b > 0$

$$w^\top f(x) > 0$$



- Can delete bias if we augment feature space:

$f(x)$ = [0.5, 1.6, 0.3]

↓

[0.5, 1.6, 0.3, **1**]

# Naive Bayes (recap)

- Data point $x = (x_1, ..., x_n)$, label $y \in \{0, 1\}$
- Model: $P(x, y) = P(y) \prod_{i=1}^{n} P(x_i|y)$ ← conditional independence
- Inference: Compute $P(y|x)$, predict $\text{argmax}_y P(y|x)$ to classify

Bayes rule

$$P(y|x) = \frac{P(x, y)}{P(x)} \propto P(x, y) = P(y)P(x|y) = P(y) \sum_{j=i}^{n} P(x_i|y)$$

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

- Learning: Find values of $P(y)$, $P(x_i|y)$ that maximize data likelihood
  - i.e. maximize $P(x, y)$ by reading counts off the data

# Maximum Likelihood for Naive Bayes (recap)

*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* —

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

prior

$$P(\text{great}|+) = \frac{1}{2} \qquad P(\text{great}|-) = \frac{1}{4}$$

word likelihood

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

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

This is Bernoulli (binary features) document model!

# Naive Bayes

## Text Classification using Naive Bayes

Hiroshi Shimodaira*

10 February 2015

Text classification is the task of classifying documents by their content: that is, by the words of which they are comprised. Perhaps the best-known current text classification problem is email *spam filtering*: classifying email messages into spam and non-spam (ham).

### 1 Document models

Text classifiers often don't use any kind of deep representation about language: often a document is represented as a *bag of words*. (A bag is like a set that allows repeating elements.) This is an extremely simple representation: it only knows which words are included in the document (and how many times each word occurs), and throws away the word order!

Consider a document $D$, whose class is given by $C$. In the case of email spam filtering there are two classes $C = S$ (spam) and $C = H$ (ham). We classify $D$ as the class which has the highest posterior probability $P(C|D)$, which can be re-expressed using Bayes' Theorem:

$$P(C|D) = \frac{P(D|C)\,P(C)}{P(D)} \propto P(D|C)\,P(C)\,. \tag{1}$$

We shall look at two probabilistic models of documents, both of which represent documents as a bag of words, using the Naive Bayes assumption. Both models represent documents using feature vectors whose components correspond to word types. If we have a vocabulary $V$, containing $|V|$ word types, then the feature vector dimension $d = |V|$.

**Bernoulli document model:** a document is represented by a feature vector with binary elements taking value 1 if the corresponding word is present in the document and 0 if the word is not present.

**Multinomial document model:** a document is represented by a feature vector with integer elements whose value is the frequency of that word in the document.

**Example:** Consider the vocabulary:

$$V = \{blue, red, dog, cat, biscuit, apple\}\,.$$

In this case $|V| = d = 6$. Now consider the (short) document "the blue dog ate a blue biscuit". If $\mathbf{d}^B$ is the Bernoulli feature vector for this document, and $\mathbf{d}^M$ is the multinomial feature vector, then we would have:

$$\mathbf{d}^B = (1, 0, 1, 0, 1, 0)^T$$
$$\mathbf{d}^M = (2, 0, 1, 0, 1, 0)^T$$

To classify a document we use equation (1), which requires estimating the likelihoods of the document given the class, $P(D|C)$ and the class prior probabilities $P(C)$. To estimate the likelihood, $P(D|C)$, we use the Naive Bayes assumption applied to whichever of the two document models we are using.

*Heavily based on notes inherited from Steve Renals and Iain Murray.



### 2 The Bernoulli document model

As mentioned above, in the Bernoulli model a document is represented by a binary vector, which represents a point in the space of words. If we have a vocabulary $V$ containing a set of $|V|$ words, then the $t$ th dimension of a document vector corresponds to word $w_t$ in the vocabulary. Let $\mathbf{b}_i$ be the feature vector for the $i$ th document $D_i$; then the $t$ th element of $\mathbf{b}_i$, written $b_{it}$, is either 0 or 1 representing the absence or presence of word $w_t$ in the $i$ th document.

Let $P(w_t|C)$ be the probability of word $w_t$ occurring in a document of class $C$; the probability of $w_t$ not occurring in a document of this class is given by $(1 - P(w_t|C))$. If we make the naive Bayes assumption, that the probability of each word occurring in the document is independent of the occurrences of the other words, then we can write the document likelihood $P(D_i \mid C)$ in terms of the individual word likelihoods $P(w_t|C)$:

$$P(D_i|C) \sim P(\mathbf{b}_i|C) = \prod_{t=1}^{|V|} [b_{it}P(w_t|C) + (1 - b_{it})(1 - P(w_t|C))]\,. \tag{2}$$

This product goes over all words in the vocabulary. If word $w_t$ is present, then $b_{it} = 1$ and the required probability is $P(w_t|C)$; if word $w_t$ is not present, then $b_{it} = 0$ and the required probability is $1 - P(w_t|C)$. We can imagine this as a model for generating document feature vectors of class $C$, in which the document feature vector is modelled as a collection of $|V|$ weighted coin tosses, the $t$ th having a probability of success equal to $P(w_t|C)$.

The *parameters* of the likelihoods are the probabilities of each word given the document class $P(w_t|C)$; the model is also parameterised by the prior probabilities, $P(C)$. We can learn (estimate) these parameters from a training set of documents labelled with class $C = k$. Let $n_k(w_t)$ be the number of documents of class $C = k$ in which $w_t$ is observed; and let $N_k$ be the total number of documents of that class. Then we can estimate the parameters of the word likelihoods as,

$$\hat{P}(w_t \mid C = k) = \frac{n_k(w_t)}{N_k}\,, \tag{3}$$

the relative frequency of documents of class $C = k$ that contain word $w_t$. If there are $N$ documents in total in the training set, then the prior probability of class $C = k$ may be estimated as the relative frequency of documents of class $C = k$:

$$\hat{P}(C = k) = \frac{N_k}{N}\,. \tag{4}$$

Thus given a training set of documents (each labelled with a class), and a set of $K$ classes, we can estimate a Bernoulli text classification model as follows:



http://socialmedia-class.org/slides_AU2017/Shimodaira_note07.pdf

# Problems with Naive Bayes

*the film was beautiful, stunning cinematography and gorgeous sets, but boring* —

$$P(x_{\text{beautiful}}|+) = 0.1 \qquad P(x_{\text{beautiful}}|-) = 0.01$$
$$P(x_{\text{stunning}}|+) = 0.1 \qquad P(x_{\text{stunning}}|-) = 0.01$$
$$P(x_{\text{gorgeous}}|+) = 0.1 \qquad P(x_{\text{gorgeous}}|-) = 0.01$$
$$P(x_{\text{boring}}|+) = 0.01 \qquad P(x_{\text{boring}}|-) = 0.1$$

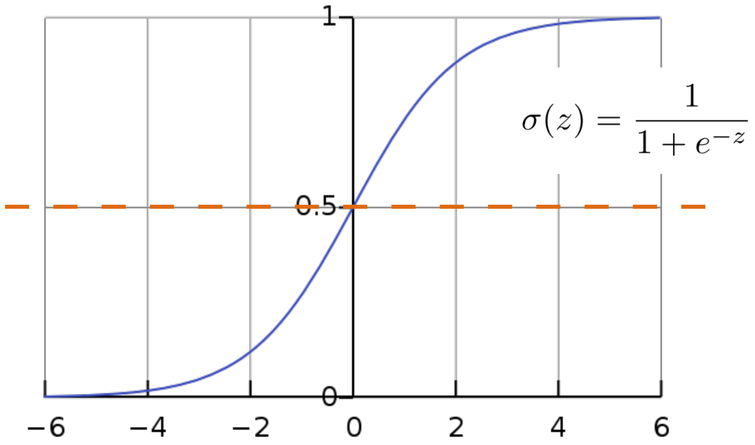
- Correlated features compound: *beautiful* and *gorgeous* are not independent!
- Naive Bayes is naive, but another problem is that it's *generative*: spends capacity modeling P(x, y), when what we care about is P(y|x)
- Discriminative models model P(y|x) directly (Logistic regression, SVMs, most neural networks, …)

# Logistic Regression

# Logistic Regression

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

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




- Decision rule: $P(y = +|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$
- To learn weights: maximize discriminative log likelihood of data P(y|x)

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

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

sum over features

# Logistic Regression

chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} = \frac{\partial f(g)}{\partial g}\frac{\partial g(x)}{\partial x}$$

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

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

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

deriv. of log

$$\frac{\partial \log \mathrm{x}}{\partial \mathrm{x}} = \frac{1}{\mathrm{x}}$$

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

deriv. of exp

$$\frac{\partial e^{\mathrm{x}}}{\partial \mathrm{x}} = e^{\mathrm{x}}$$

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

# Logistic Regression

- Recall that $y_j = 1$ for positive instances, $y_j = 0$ for negative instances.
- Gradient of $w_i$ on positive example $= x_{ji}(y_j - P(y_j = +|x_j))$

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

  Otherwise make $w_i$ look more like $x_{ji}$, which will increase P(+)

- Gradient of $w_i$ on negative example $= x_{ji}(-P(y_j = +|x_j))$

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

  Otherwise make $w_i$ look less like $x_{ji}$, which will decrease P(+)

- Can combine these gradients as $\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w} = x_j(y_j - P(y_j = 1|x_j))$

# Gradient Decent

log likelihood of data P(y|x)

data points (*j*)

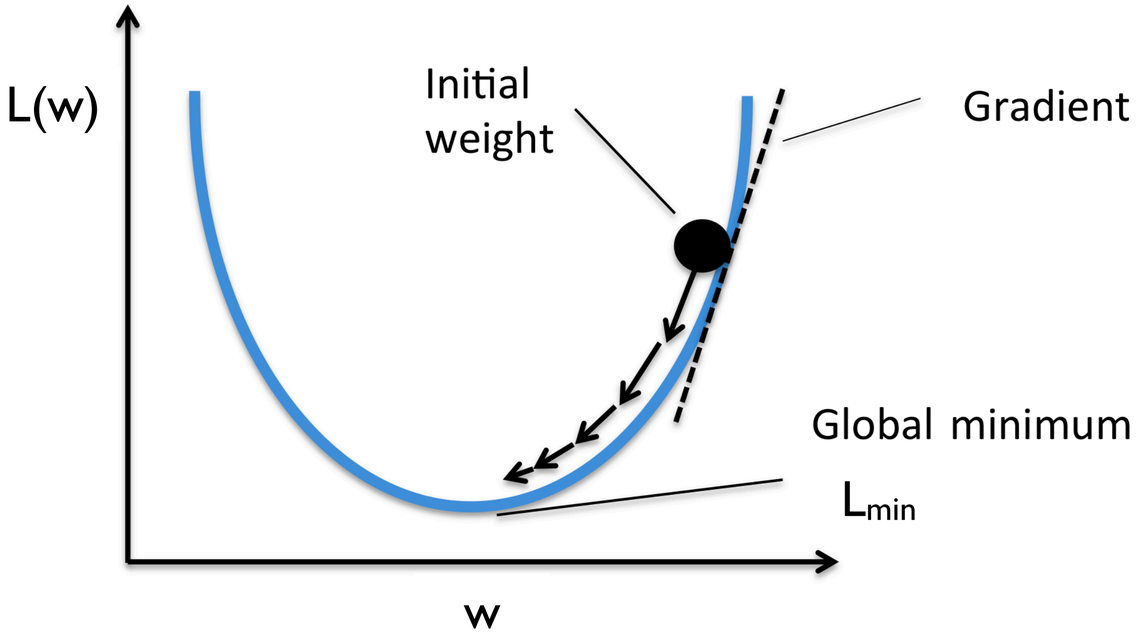
- Can combine these gradients as $\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w} = x_j(y_j - P(y_j = 1|x_j))$

- Training set log-likelihood: $\mathcal{L}(w) = \frac{1}{m}\sum_{j=1}^{m} \mathcal{L}(x_j, y_j)$

- Gradient vector: $\frac{\partial \mathcal{L}(w)}{\partial w} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_n}\right)$

# Gradient Decent

- Gradient decent (or ascent) is an iterative optimization algorithm for finding the minimum (or maximum) of a function.



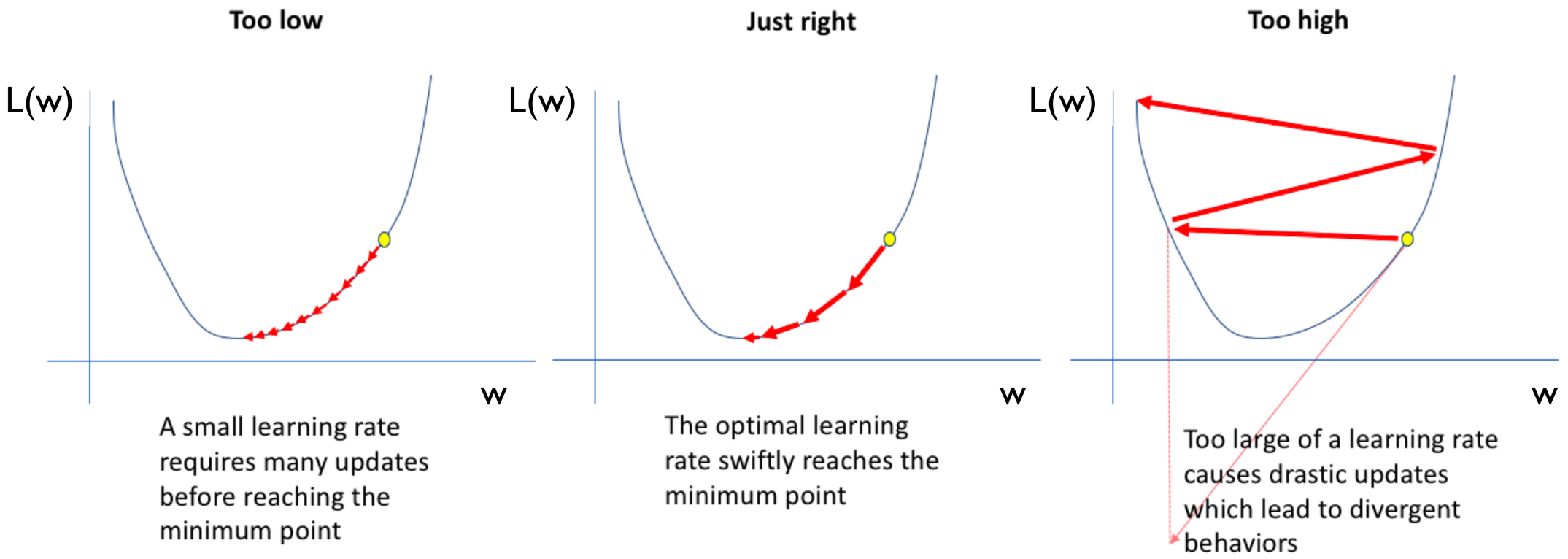

Repeat until convergence {

$$w := w - \alpha \frac{\partial \mathcal{L}(w)}{\partial w}$$

}

learning rate (step size)

# Learning Rate



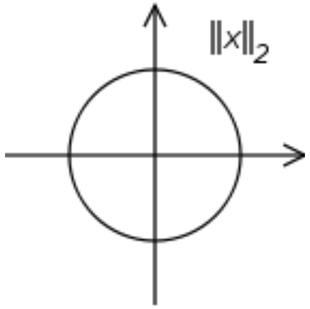



Credit: Jeremy Jordan

# 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
  - Early stopping
  - 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:

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

# Logistic Regression vs. Naive Bayes

- Both are (log) linear models $w^\top f(x)$
- Logistic regression doesn't assume conditional independence of features
  - Weights are trained independently
  - Can handle highly correlated overlapping features
- Naive Bayes assume conditional independence of features
  - Weights are trained jointly

# Perceptron/SVM

# Perceptron

- Simple error-driven learning approach similar to logistic regression
- Decision rule: $w^\top x > 0$
  - If incorrect: if positive, $w \leftarrow w + x$

    if negative, $w \leftarrow w - x$

Logistic Regression

$$w \leftarrow w + x(1 - P(y = 1|x))$$

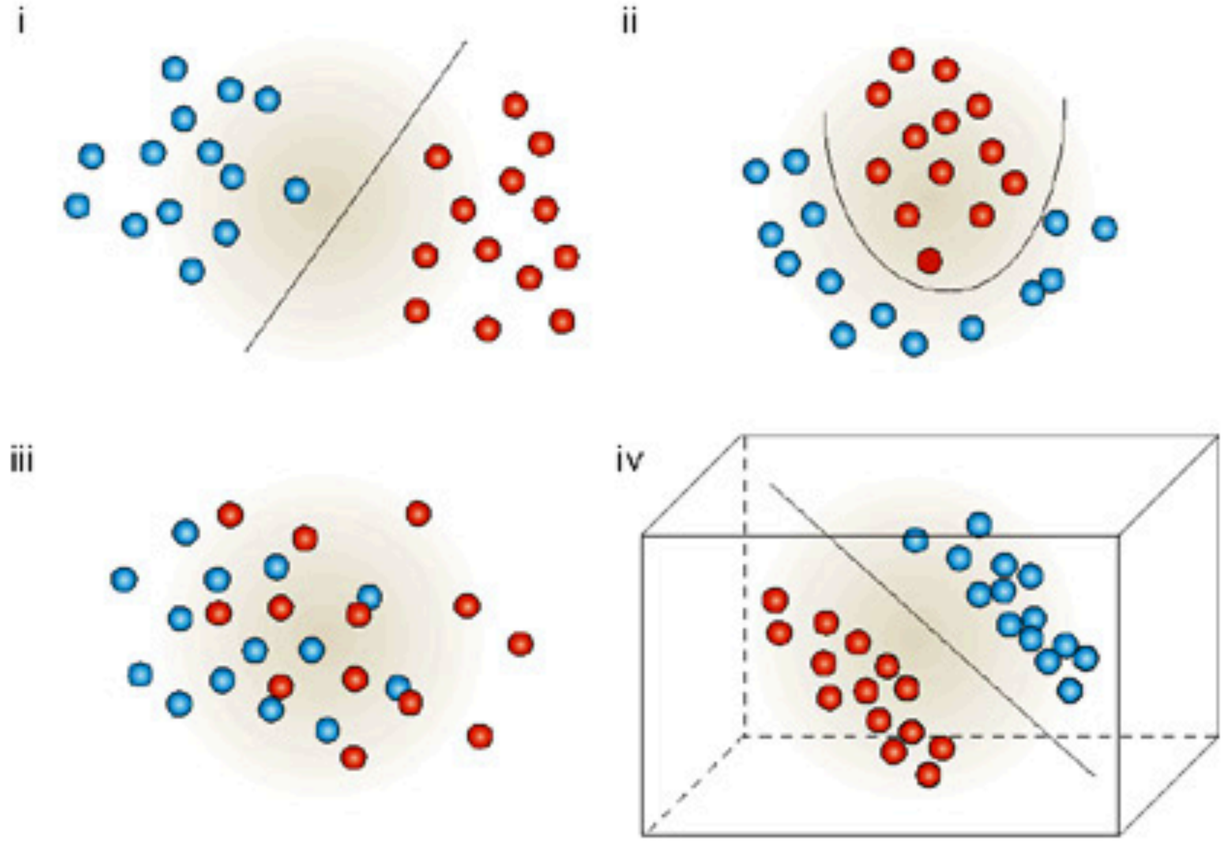
$$w \leftarrow w - xP(y = 1|x)$$

- Algorithm is very similar to logistic regression.
  - Only hyper-parameter is max number of iterations (LR uses learning rate)
  - Guaranteed to eventually separate the data if the data are separable (LR always converge)

http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

# Linear Separability

- In general, two groups are linearly separable in n-dimensional space, if they can be separated by an (n-1)-dimensional hyperplane.



# What does “converge” mean?

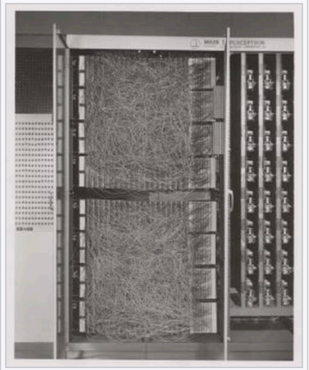
- It means that it can make an entire pass through the training data without making any more updates.
- In other words, Perceptron has correctly classified every training example.
- Geometrically, this means that it was found some hyperplane that correctly segregates the data into positive and negative examples

http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

# Perceptron

## History [ edit ]

V • T • E



Mark I Perceptron machine, the first implementation of the perceptron algorithm. It was connected to a camera with 20×20 cadmium sulfide photocells to make a 400-pixel image. The main visible feature is a patch panel that set different combinations of input features. To the right, arrays of potentiometers that implemented the adaptive weights.[2]:213

*See also: History of artificial intelligence § Perceptrons and the attack on connectionism, and AI winter § The abandonment of connectionism in 1969*

The perceptron algorithm was invented in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt,[3] funded by the United States Office of Naval Research.[4]

The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron". This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.[2]:193

In a 1958 press conference organized by the US Navy, Rosenblatt made statements about the perceptron that caused a heated controversy among the fledgling AI community; based on Rosenblatt's statements, *The New York Times* reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."[4]

Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns. This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had greater processing power than perceptrons with one layer (also called a single layer perceptron).

Single layer perceptrons are only capable of learning linearly separable patterns. For a classification task with some step activation function a single node will have a single line dividing the data points forming the patterns. More nodes can create more dividing lines, but those lines must somehow be combined to form more complex classifications. A second layer of perceptrons, or even linear nodes, are sufficient to solve a lot of otherwise non-separable problems.

In 1969 a famous book entitled *Perceptrons* by Marvin Minsky and Seymour Papert showed that it was impossible for these classes of network to learn an XOR function. It is often believed (incorrectly) that they also conjectured that a similar result would hold for a multi-layer perceptron network. However, this is not true, as both Minsky and Papert already knew that multi-layer perceptrons were capable of producing an XOR function. (See the page on *Perceptrons (book)* for more information.) Nevertheless, the often-miscited Minsky/Papert text caused a significant decline in interest and funding of neural network research. It took ten more years until neural network research experienced a resurgence in the 1980s. This text was reprinted in 1987 as "Perceptrons - Expanded Edition" where some errors in the original text are shown and corrected.

The kernel perceptron algorithm was already introduced in 1964 by Aizerman et al.[5] Margin bounds guarantees were given for the Perceptron algorithm in the general non-separable case first by Freund and Schapire (1998),[1] and more recently by Mohri and Rostamizadeh (2013) who extend previous results and give new L1 bounds.[6]

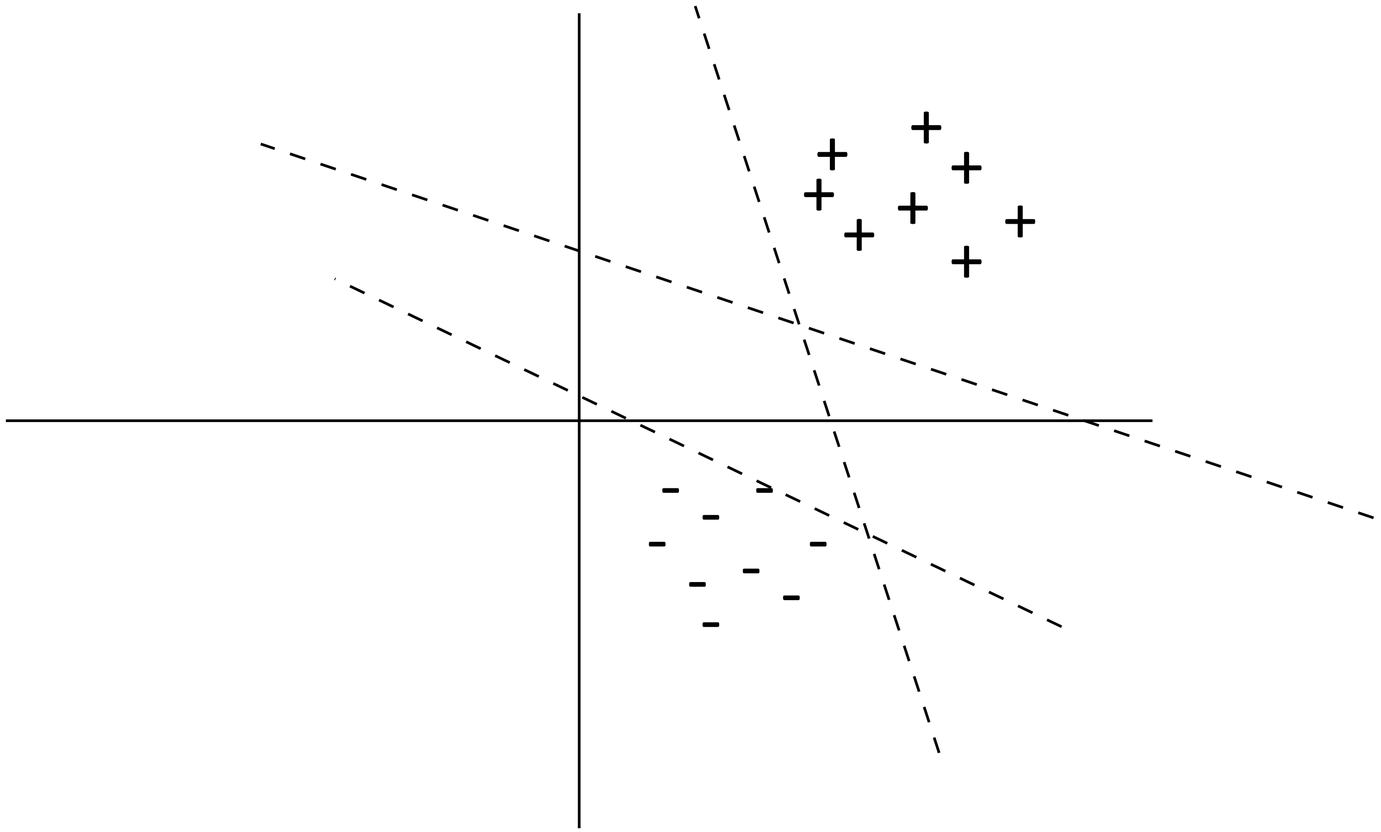
The perceptron is a simplified model of a biological neuron. While the complexity of biological neuron models is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons.[7]


Frank Rosenblatt (1928-1971)

PhD 1956 from Cornell

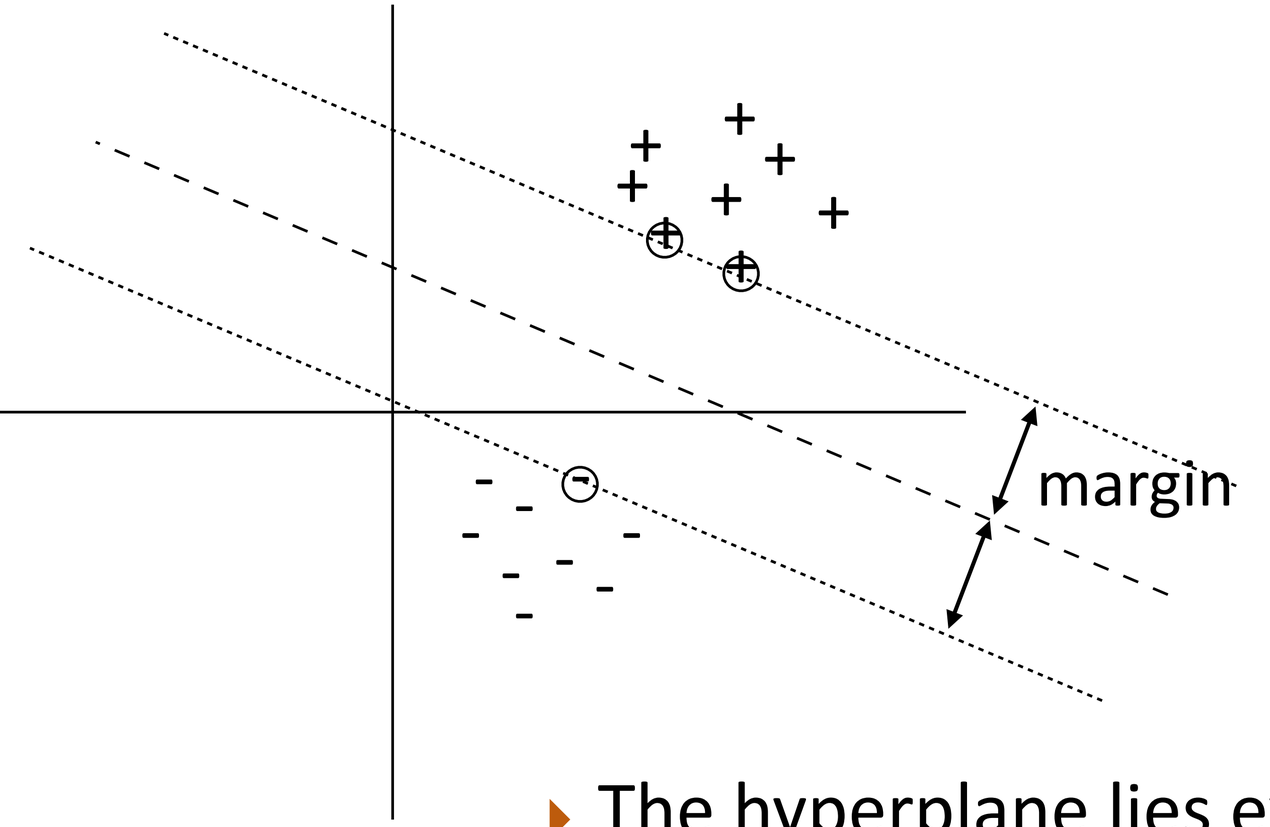
# Support Vector Machines (extracurricular)

- Many separating hyperplanes — is there a best one?



# Support Vector Machines (extracurricular)

- Many separating hyperplanes — is there a best one?




  - The hyperplane lies exactly halfway between the nearest positive point and nearest negative point.

# Support Vector Machines (extracurricular)

- Constraint formulation: find *w* via following quadratic program:

Minimize $\|w\|_2^2$

s.t. $\forall j \quad w^\top x_j \geq 1 \text{ if } y_j = 1$

$w^\top x_j \leq -1 \text{ if } y_j = 0$

minimizing norm with fixed margin <=> maximizing margin

As a single constraint:

$$\forall j \quad (2y_j - 1)(w^\top x_j) \geq 1$$

- Generally no solution (data is generally non-separable) — need slack!

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

# N-Slack SVMs (extracurricular)

Minimize $\lambda \|w\|_2^2 + \sum_{j=1}^{m} \xi_j$

s.t. $\forall j \quad (2y_j - 1)(w^\top x_j) \geq 1 - \xi_j \qquad \forall j \quad \xi_j \geq 0$

- The $\xi_j$ are a "fudge factor" to make all constraints satisfied
- Take the gradient of the objective:

$$\frac{\partial}{\partial w_i}\xi_j = 0 \text{ if } \xi_j = 0 \qquad \frac{\partial}{\partial w_i}\xi_j = (2y_j - 1)x_{ji} \text{ if } \xi_j > 0$$

$$= x_{ji} \text{ if } y_j = 1, \ -x_{ji} \text{ if } y_j = 0$$

- Looks like the perceptron! But updates more frequently

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

# LR, Perceptron, SVM

- Gradients on Positive Examples

**Logistic regression**

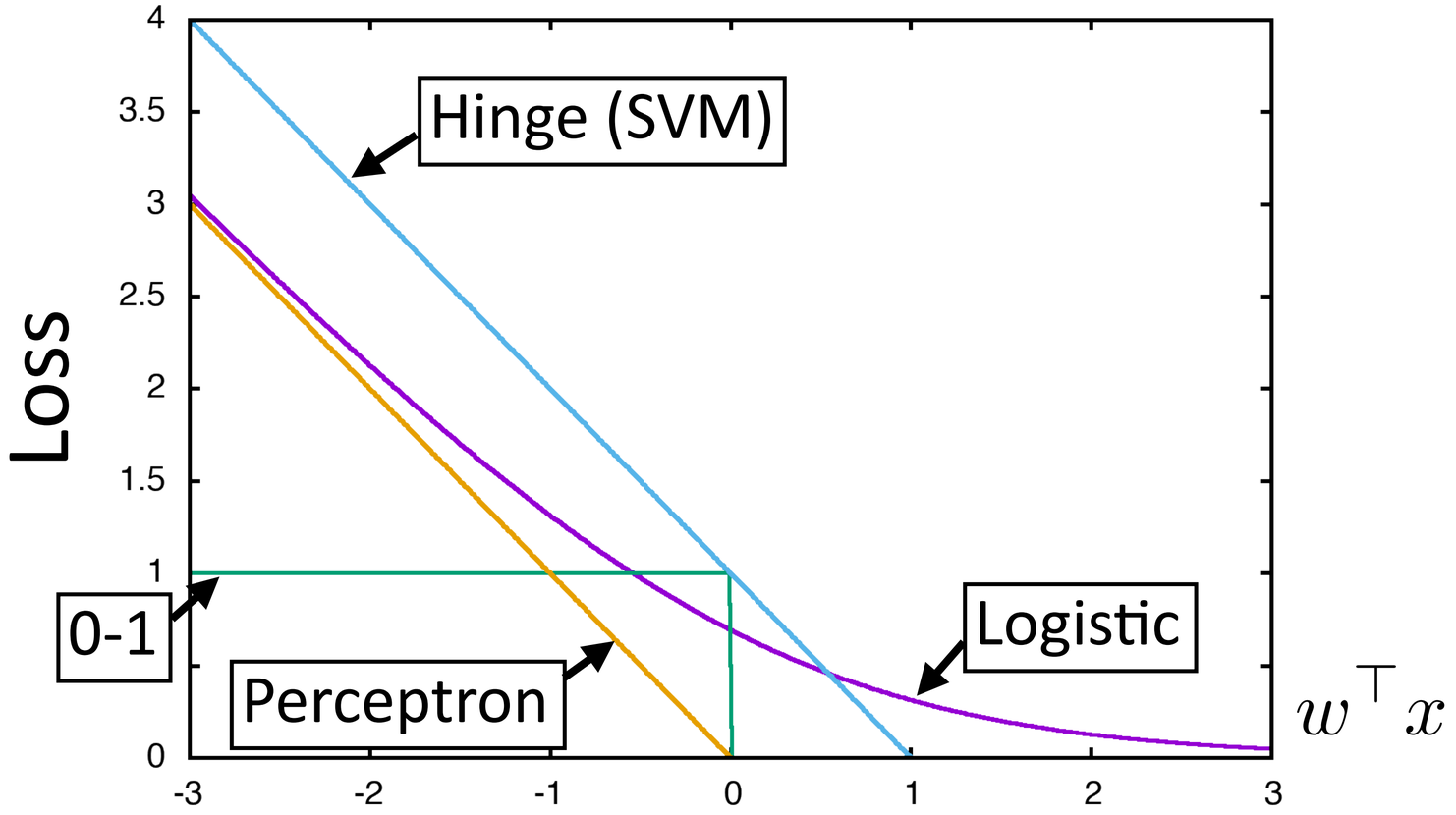
$$x(1 - \text{logistic}(w^\top x))$$

**Perceptron**

$$x \text{ if } w^\top x < 0, \text{ else } 0$$

**SVM (ignoring regularizer)**

$$x \text{ if } w^\top x < 1, \text{ else } 0$$




*gradients are for maximizing things, which is why they are flipped

http://ciml.info/dl/v0_99/ciml-v0_99-ch07.pdf

# 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)

# Summary

- Logistic regression: $P(y = 1|x) = \frac{\exp\left(\sum_{i=1}^{n} w_i x_i\right)}{\left(1 + \exp\left(\sum_{i=1}^{n} w_i x_i\right)\right)}$

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

  Gradient (unregularized): $x(y - P(y = 1|x))$

- Logistic regression, perceptron, and SVM are closely related
- All gradient updates: “make it look more like the right thing and less like the wrong thing”

# Optimization — next …

- Range of techniques from simple gradient descent (works pretty well) to more complex methods (can work better), e.g., Newton's method, Quasi-Newton methods (LBFGS), Adagrad, Adadelta, etc.

- Most methods boil down to: take a gradient and a step size, apply the gradient update times step size, incorporate estimated curvature information to make the update more effective

# QA Time