Binary Classification (cont')

Wei Xu

(many slides from Greg Durrett and Vivek Srikumar)

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

- Perceptron guaranteed to eventually separate the data if the data are separable

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

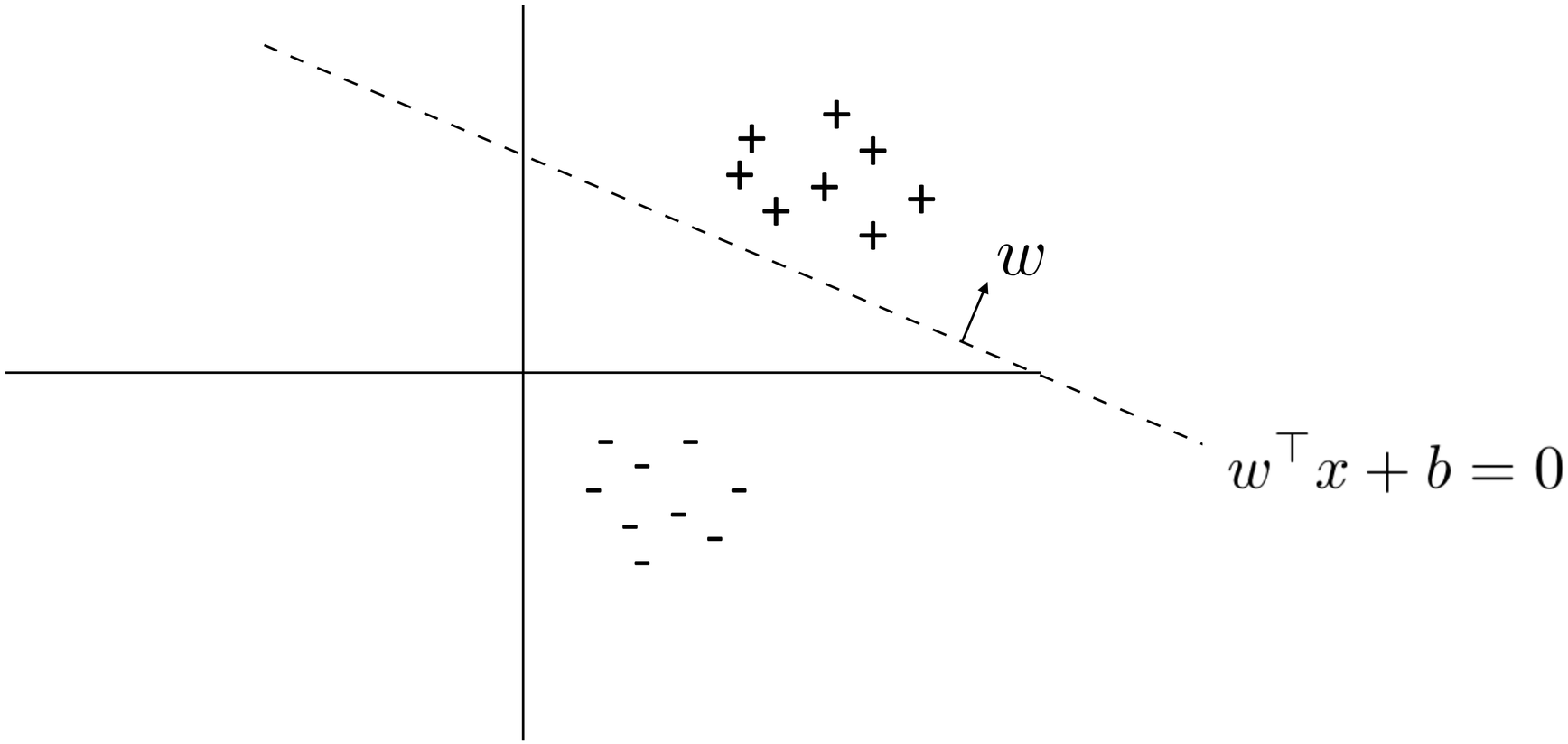
What does “converge” mean?

- 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

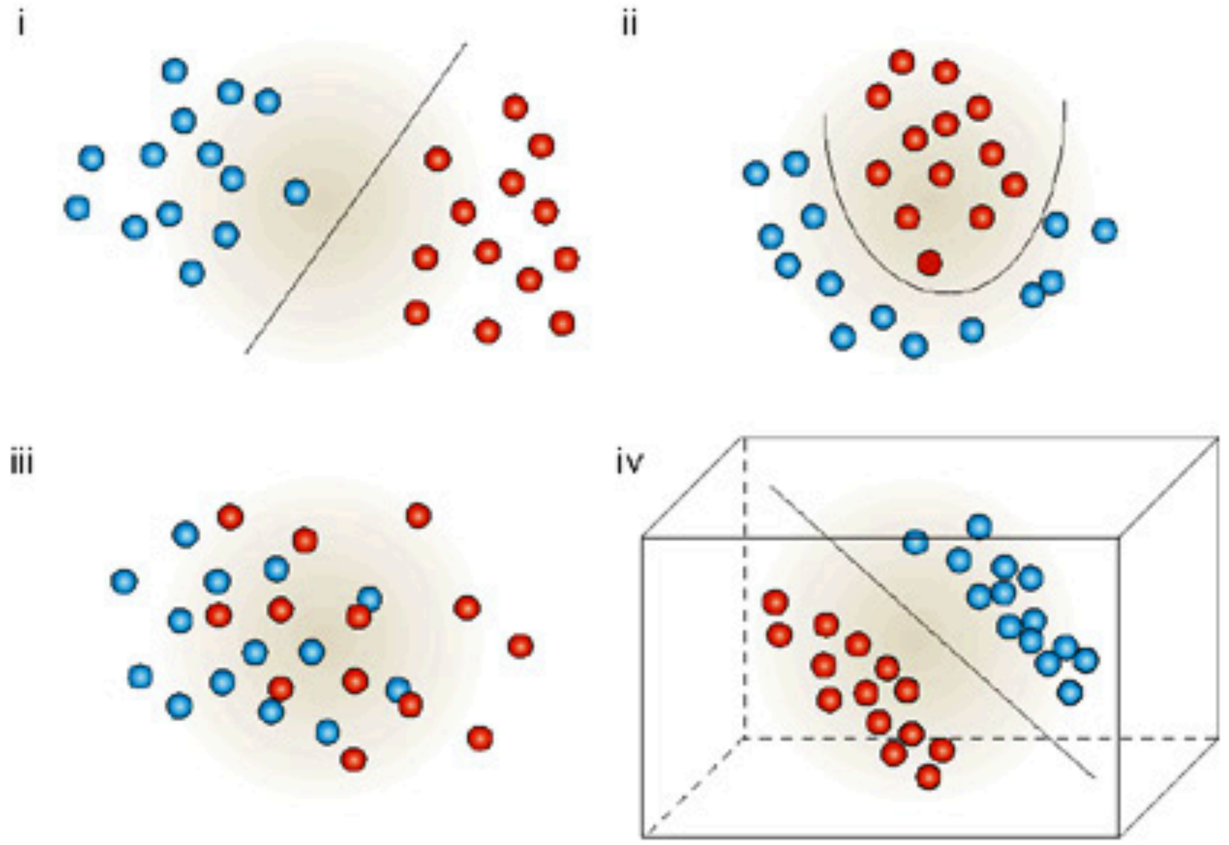
- Separating hyperplane



Two vectors have a zero dot product if and only if they are perpendicular

Linear Separability

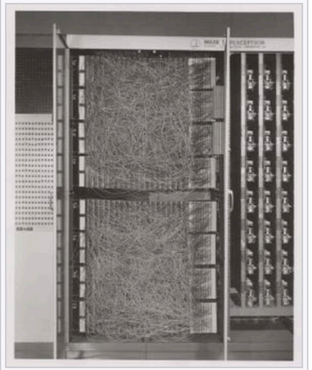
- In general, two groups are linearly separable in n-dimensional space, if they can be separated by an (n-1)-dimensional hyperplane.



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

Perceptron - artificial neuron

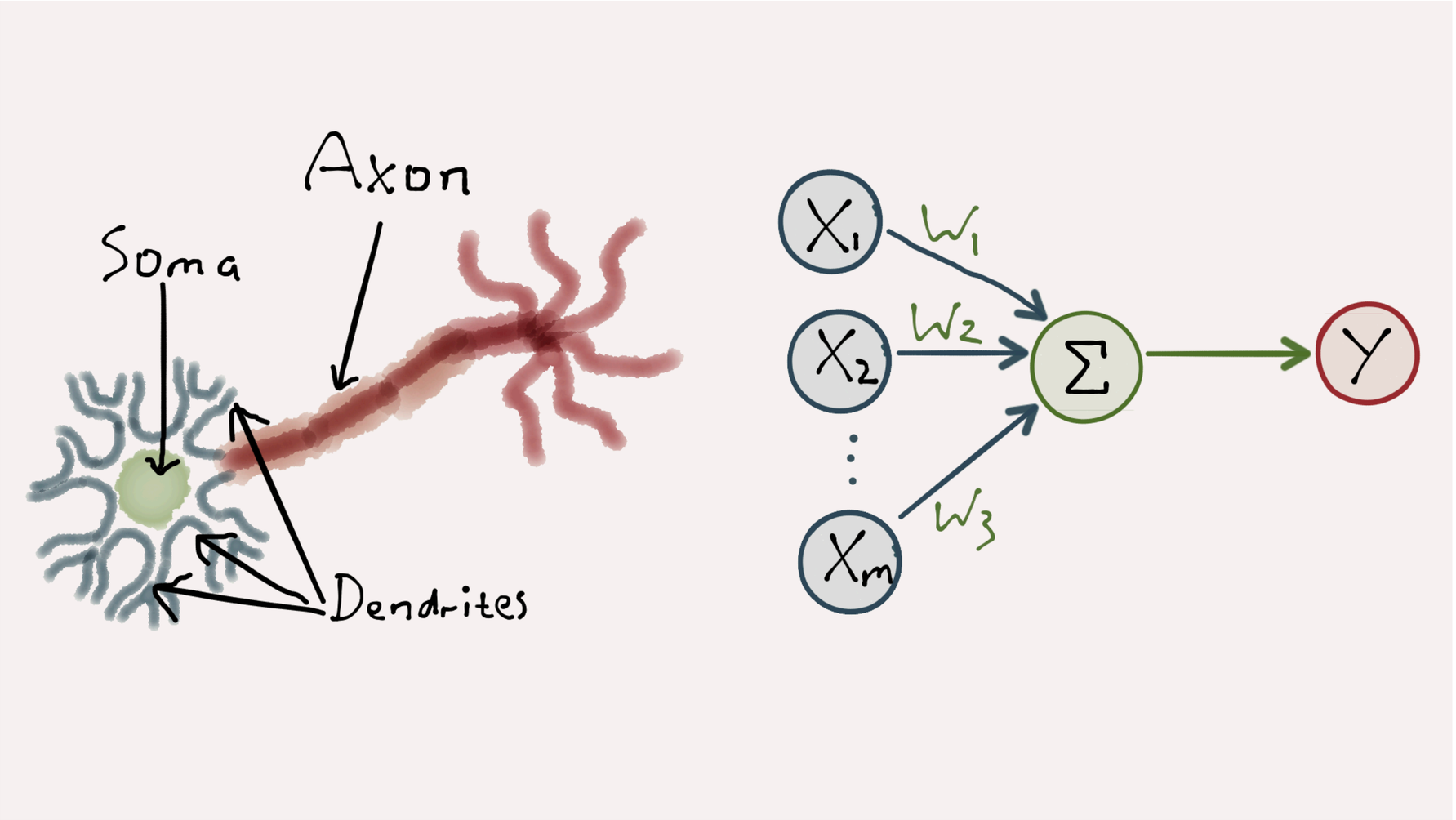
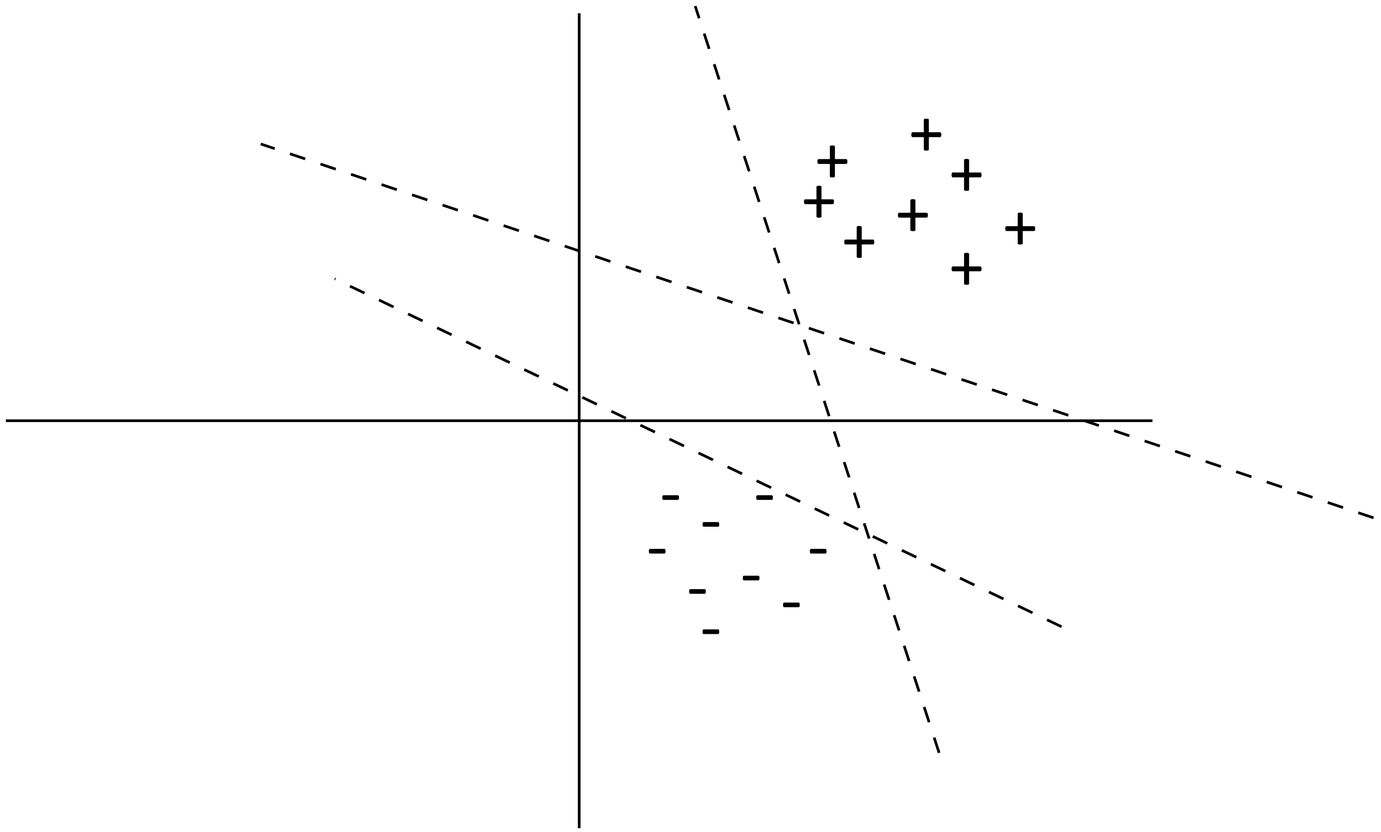


Figure from https://jontysinai.github.io/jekyll/update/2017/11/11/the-perceptron.html

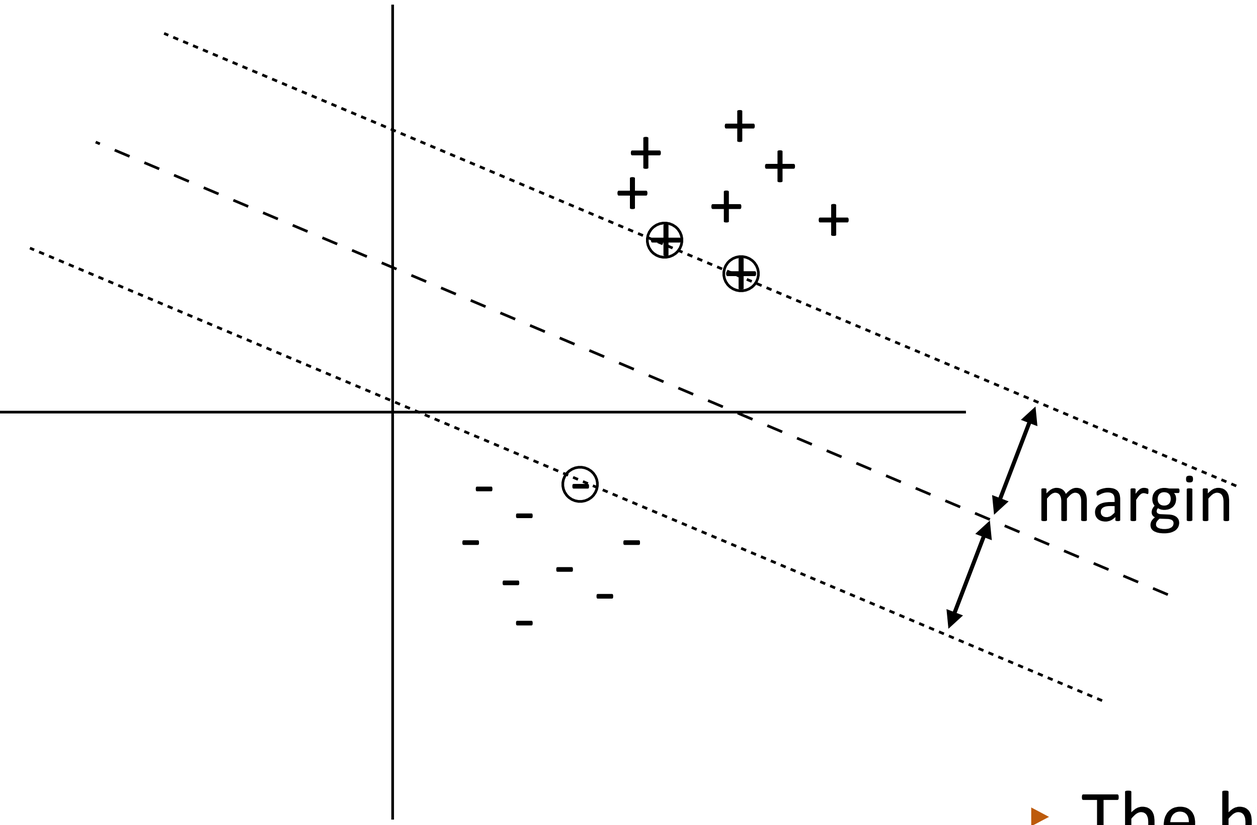
Support Vector Machines

- Many separating hyperplanes — is there a best one?



Support Vector Machines

- Many separating hyperplanes — is there a best one?



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

Support Vector Machines

- 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

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

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

- The ξ_j are a "fudge factor" to make all constraints satisfied

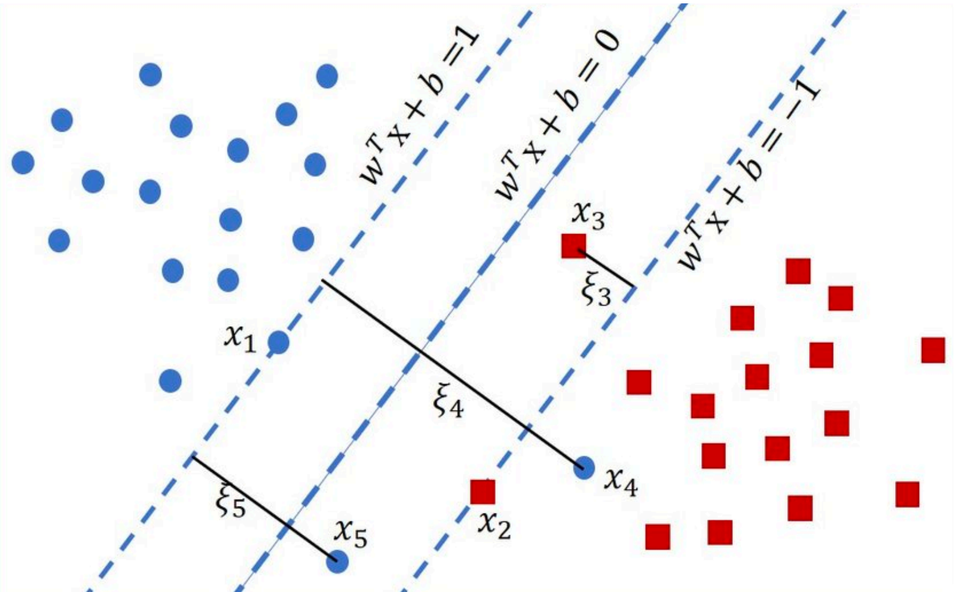


Image credit: Lang Van Tran

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

N-Slack SVMs

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 ξ_j are a "fudge factor" to make all constraints satisfied
- Take the gradient of the objective (flip for maximizing):

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

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

- Looks like the perceptron! But updates more frequently

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

LR, Perceptron, SVM

- 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”

LR, Perceptron, SVM

- Gradients on Positive Examples

Logistic regression

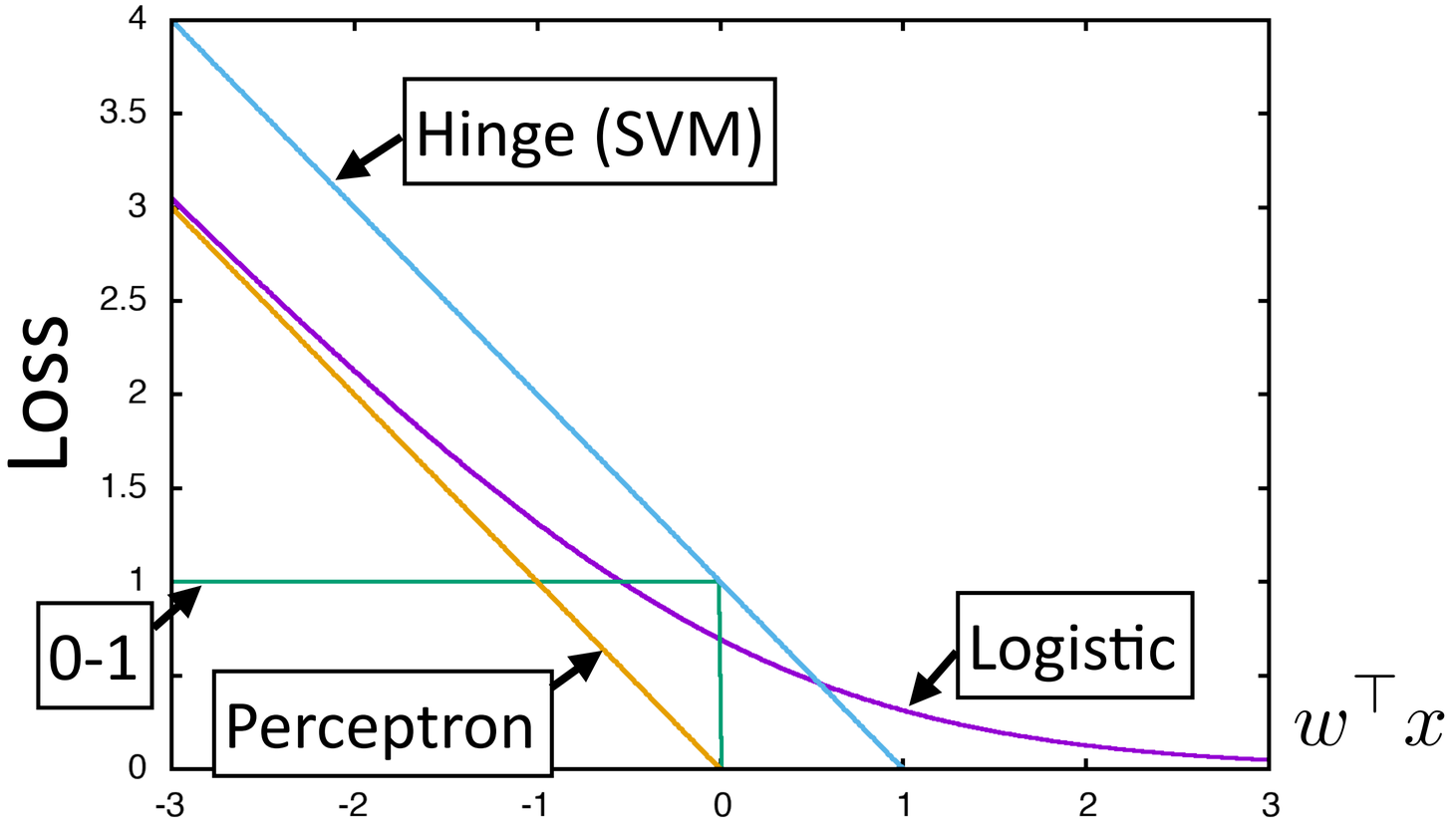
$$x(1 - \text{logistic}(w^\top x))$$

Perceptron

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

SVM (ignoring regularizer)

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



*these 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)

Recap

- 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