

Numerical Learning Algorithms

Introduction	2
Naive Bayes	3
Naive Bayes	3
Naive Bayes Example	4
Naive Bayes Example Continued	5
Linear Models	6
Linear Models	6
Example of Numeric Examples	7
Linear Regression	8
Least Squares Gradient Descent	9
Perceptron Learning Rule	10
Perceptrons Continued	11
Example of Perceptron Learning ($\alpha = 1$)	12
The Nearest Neighbor Algorithm	13
The Nearest Neighbor Algorithm	13
Neural Networks	14
Artificial Neural Networks	14
ANN Structure	15
ANN Illustration	16
Illustration	17
Sigmoid Activation	18
Plot of Sigmoid Function	19
Backpropagation	20
Applying The Chain Rule	21
Support Vector Machines	22
Support Vector Machines	22

Example SVM for Separable Examples	23
Example SVM for Nonseparable Examples	24
Example Gaussian Kernel SVM	25
Example Gaussian Kernel, Zoomed In	26
Ensemble Learning	27
Ensemble Learning	27
Boosting	28
Example Boosting Algorithm	29
Example Run of AdaBoost	30
Example Run of AdaBoost, Continued	31

Introduction

- Numerical learning methods learn the parameters or weights of a model, often by optimizing an error function. Examples include:
- Calculate the parameters of a probability distribution.
- Separate positive from negative examples by a decision boundary.
- Find points close to positive but far from negative examples.
- Update parameters to decrease error.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 2

Naive Bayes

3

Naive Bayes

- For class C and attributes X_i , assume:
$$\mathbf{P}(C, X_1, \dots, X_n) = \mathbf{P}(C)\mathbf{P}(X_1|C)\dots\mathbf{P}(X_n|C)$$
- This corresponds to a Bayesian network where C is the sole parent of each X_i .
- Estimate prior and conditional probabilities by counting.
- If an outcome occurs m times out of n trials, Laplace's law of succession recommends the estimate $(m + 1)/(n + k)$ where k is the number of outcomes.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 3

Naive Bayes Example

Using Laplace's law of succession on the 14 examples.:

$$P(\text{pos}) = (9 + 1)/(14 + 2) = 10/16$$

$$P(\text{neg}) = (5 + 1)/(14 + 2) = 6/16$$

$$P(\text{sunny} | \text{pos}) = (2 + 1)/(9 + 3) = 3/12$$

$$P(\text{overcast} | \text{pos}) = (4 + 1)/(9 + 3) = 5/12$$

$$P(\text{rain} | \text{pos}) = (3 + 1)/(9 + 3) = 4/12$$

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 4

Naive Bayes Example Continued

For the first example:

$$\begin{aligned} P(\text{pos} | \text{sunny, hot, high, false}) \\ &= \alpha (10/16) (3/12) (3/12) (4/11) (7/11) \\ &\approx \alpha 0.00904 \end{aligned}$$

$$\begin{aligned} P(\text{neg} | \text{sunny, hot, high, false}) \\ &= \alpha (6/16) (4/8) (3/8) (5/7) (3/7) \\ &\approx \alpha 0.02152 \\ &\approx \frac{0.02152}{0.00904 + 0.02152} \approx 0.704 \end{aligned}$$

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 5

Linear Models

6

Linear Models

- For a *linear model*, the output and each attribute must be numeric.
- The input of an example is a numeric vector $\mathbf{x} = (1.0, x_1, \dots, x_n)$.
- A hypothesis is a weight vector $\mathbf{w} = (w_0, w_1, \dots, w_n)$. w_0 is the *bias weight*.
- The output of a hypothesis is computed by
$$\hat{y} = w_0 + w_1x_1 + \dots + w_nx_n = \mathbf{w} \cdot \mathbf{x}$$
- The *loss* on example (\mathbf{x}, y) is typically one of:
Squared error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$
Absolute error loss: $L_1(y, \hat{y}) = |y - \hat{y}|$
0/1 loss: $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$ else 1

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 6

Example of Numeric Examples

No.	Input Attributes						Output
	Sunny	Rainy	Hot	Cool	Humid	Windy	
1	1	0	1	0	1	0	-1
2	1	0	1	0	1	1	-1
3	0	0	1	0	1	0	1
4	0	1	0	0	1	0	1
5	0	1	0	1	0	0	1
6	0	1	0	1	0	1	-1
7	0	0	0	1	0	1	1
8	1	0	0	0	1	0	-1
9	1	0	0	1	0	0	1
10	0	1	0	0	0	0	1
11	1	0	0	0	0	1	1
12	0	0	0	0	1	1	1
13	0	0	1	0	0	0	1
14	0	1	0	0	1	1	-1

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 7

Linear Regression

- *Linear regression* finds the weights that minimizes loss over the training set.
- *Gradient descent* changes the weights based on the gradient, the derivatives of the loss with respect to the weights. (more on next page)
- The *linear least squares algorithm* calculates the weights by:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where \mathbf{X} is the *data matrix* and \mathbf{y} is the vector of outputs.

- Classification can be performed by
if $\mathbf{w} \cdot \mathbf{x} > 0$ then positive else negative

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 8

Least Squares Gradient Descent

$\mathbf{w} \leftarrow$ zeroes

loop until convergence

for each example (\mathbf{x}_j, y_j)

$$\hat{y}_j \leftarrow \mathbf{w} \cdot \mathbf{x}_j$$

for each w_i in \mathbf{w}

$$w_i \leftarrow w_i + \alpha(y_j - \hat{y}_j)x_{ij}$$

where α is the *learning rate*. This is a small number chosen to tradeoff speed of convergence vs. closeness to optimal weights.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 9

Perceptron Learning Rule

[differs from book] A *perceptron* does gradient descent for absolute error loss (more accurately, “ramp loss”). This assumes each y_j is 1 or -1.

$\mathbf{w} \leftarrow$ zeroes

loop until convergence

for each example (\mathbf{x}_j, y_j)

$$\hat{y}_j \leftarrow \mathbf{w} \cdot \mathbf{x}_j$$

if $(y_j = 1 \wedge \hat{y}_j < 1) \vee (y_j = -1 \wedge \hat{y}_j > -1)$ **then**

for each w_i in \mathbf{w}

$$w_i \leftarrow w_i + \alpha y_j x_{ij}$$

Again, α is the learning rate.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 10

Perceptrons Continued

- The perceptron convergence theorem states that if some \mathbf{w} classifies all the training examples correctly, then the perceptron learning rule will converge to zero error on the training examples.
- Usually, many *epochs* (passes over the training examples) are needed until convergence.
- If zero error is not possible, use $\alpha \approx 0.1/n$, where n is the number of normalized or binary inputs.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 11

Example of Perceptron Learning ($\alpha = 1$)

Using $\alpha = 1$:

Inputs							Weights				
x_1	x_2	x_3	x_4	y	\hat{y}	L	w_0	w_1	w_2	w_3	w_4
0	0	0	1	-1	0	1	-1	0	0	0	0
1	1	1	0	1	-1	2	0	1	1	1	-1
1	1	1	1	1	2	0	0	1	1	1	-1
0	0	1	1	-1	0	1	-1	1	1	0	-2
0	0	0	0	1	-1	2	0	1	1	0	-2
0	1	0	1	-1	-1	0	0	1	1	0	-2
1	0	0	0	1	1	0	0	1	1	0	-2
1	0	1	1	1	-1	2	1	2	1	1	-1
0	1	0	0	-1	2	3	0	2	0	1	-1

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 12

Neural Networks

14

Artificial Neural Networks

An (artificial) neural network consists of units, connections, and weights. Inputs and outputs are numeric.

Biological NN	Artificial NN
soma	unit
axon, dendrite	connection
synapse	weight
potential	weighted sum
threshold	bias weight
signal	activation

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 14

The Nearest Neighbor Algorithm

13

The Nearest Neighbor Algorithm

- The k -nearest neighbor algorithm classifies a test example by finding the k closest training example(s), returning the most common class.
- Suppose 10% noise (best possible test error is 10%).
- With sufficient training exs., a test example will agree with its nearest neighbor with prob. $(.9)(.9) + (.1)(.1) = .82$ (both not noisy or both noisy) and disagree with prob. $(.9)(.1) + (.1)(.9) = .18$.
- In general 1-nearest neighbor converges to less than twice the optimal error (3-NN to less than 32% higher than optimal).

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 13

ANN Structure

- A typical unit j receives inputs a_1, a_2, \dots from other units and performs a weighted sum:

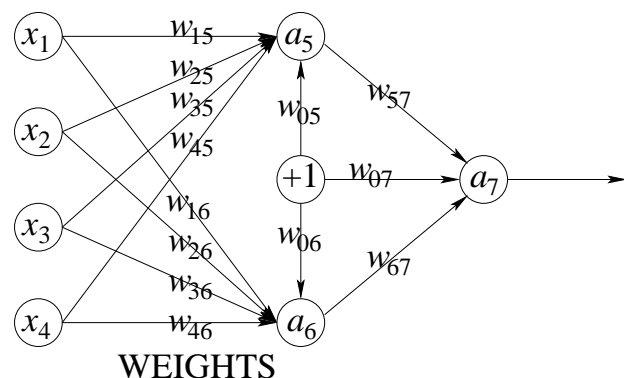
$$in_j = w_0 + \sum_i w_{ij} a_i$$
 and outputs activation $a_j = g(in_j)$.
- Typically, *input units* store the inputs, *hidden units* transform the inputs into an internal numeric vector, and an *output unit* transforms the hidden values into the prediction.
- An ANN is a function $f(\mathbf{x}, \mathbf{W}) = a$, where \mathbf{x} is an example, \mathbf{W} is the weights, and a is the prediction (activation value from output unit).
- Learning is finding a \mathbf{W} that minimizes error.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 15

ANN Illustration

INPUT UNITS HIDDEN UNITS OUTPUT UNIT OUTPUT



WEIGHTS

Sigmoid Activation

- The sigmoid function is defined as:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

- It is commonly used for ANN activation functions:

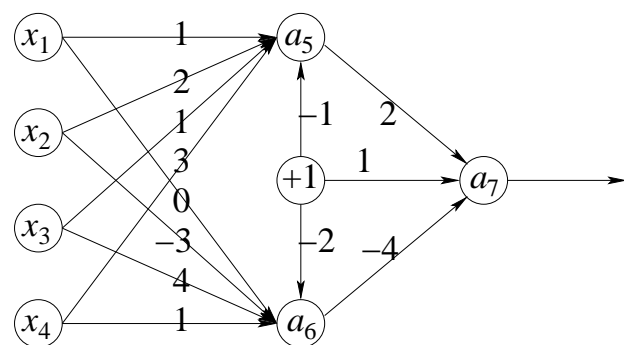
$$a_j = \text{sigmoid}(in_j) \\ = \text{sigmoid}(w_{0j} + \sum_i w_{ij} a_i)$$

- Note that

$$\frac{\partial \text{sigmoid}(x)}{\partial x} = \text{sigmoid}(x)(1 - \text{sigmoid}(x))$$

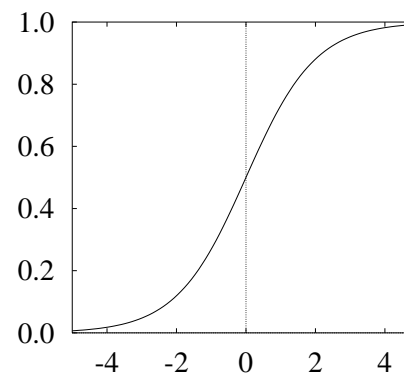
Illustration

INPUT UNITS HIDDEN UNITS OUTPUT UNIT OUTPUT



WEIGHTS

Plot of Sigmoid Function



$\text{sigmoid}(x)$ —

Backpropagation

- One learning method is *backpropagating* the error from the output to all of the weights. It is an application of the *delta rule*.
- Given loss $L(\mathbf{W}, \mathbf{x}, y)$, obtain the *gradient*:

$$\nabla L(\mathbf{W}, \mathbf{x}, y) = \left[\dots, \frac{\partial L}{\partial w_{ij}}, \dots \right]$$

- To decrease error, use the update rule:

$$w_{ij} \leftarrow w_{ij} - \alpha \frac{\partial L}{\partial w_{ij}}$$

where α is the learning rate.

Applying The Chain Rule

- Using $L = (y_k - a_k)^2$ for output unit k :

$$\begin{aligned} \frac{\partial L}{\partial w_{jk}} &= \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} \\ &= -2(y_k - a_k) a_k(1 - a_k) a_j \end{aligned}$$

- For weights from input to hidden units:

$$\begin{aligned} \frac{\partial L}{\partial w_{ij}} &= \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial in_k} \frac{\partial in_k}{\partial a_j} \frac{\partial a_j}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} \\ &= -2(y_k - a_k) a_k(1 - a_k) w_{jk} \\ &\quad a_j(1 - a_j) x_i \end{aligned}$$

Support Vector Machines

Support Vector Machines

- A SVM assigns a weight α_i to each example (\mathbf{x}_i, y_i) (\mathbf{x}_i is an attribute value vector, y_i is either -1 or 1).
- A SVM computes a discriminant by:

$$h(\mathbf{x}) = \text{sign} \left(-b + \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) \right)$$

where K is a *kernel function*.

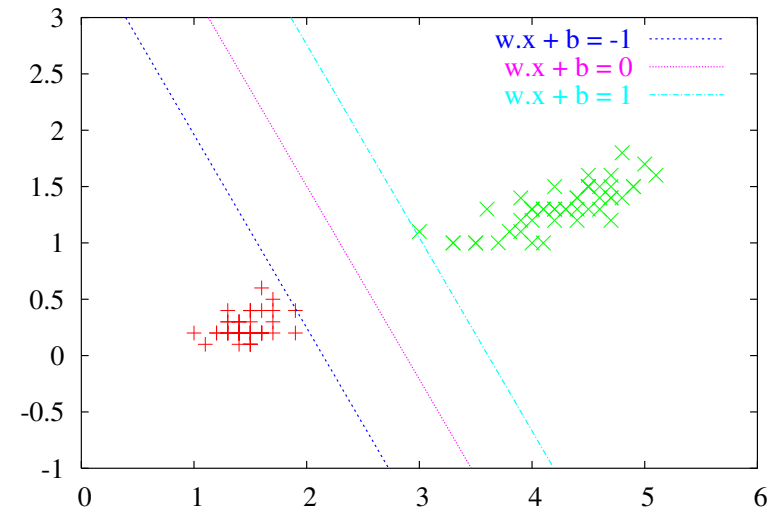
- A SVM learns by optimizing the error function:

$$\text{minimize } \|h\|^2/2 + \sum_i \max(0, 1 - y_i h(\mathbf{x}_i))$$

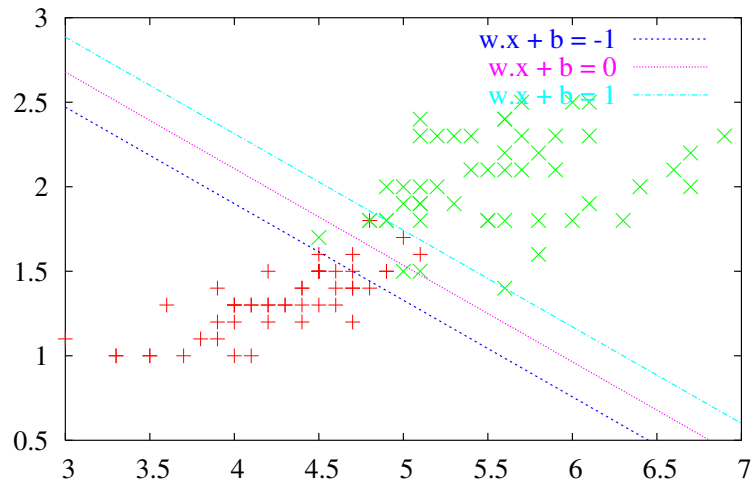
$$\text{subject to } 0 \leq \alpha_i \leq C$$

where $\|h\|$ is the size of h in kernel space

Example SVM for Separable Examples



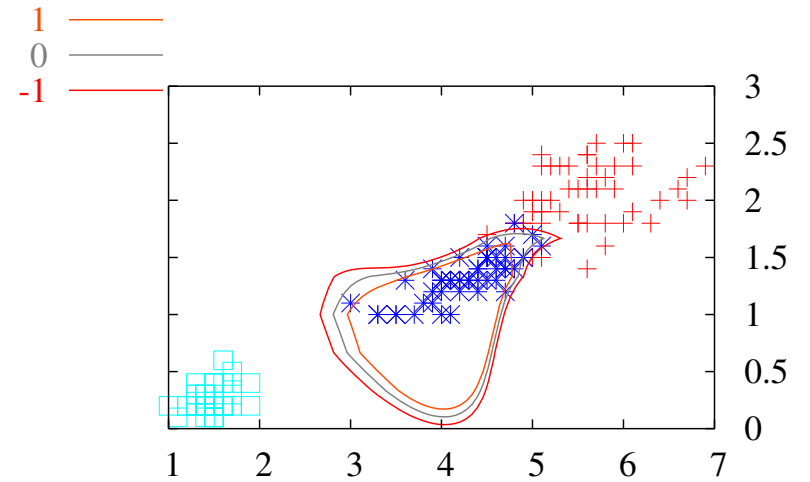
Example SVM for Nonseparable Examples



CS 3793 Artificial Intelligence

Numerical Learning Algorithms - 24

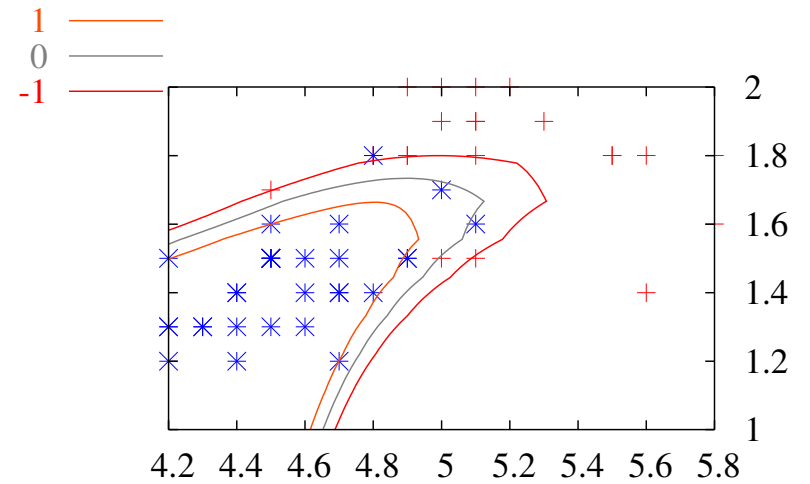
Example Gaussian Kernel SVM



CS 3793 Artificial Intelligence

Numerical Learning Algorithms - 25

Example Gaussian Kernel, Zoomed In



CS 3793 Artificial Intelligence

Numerical Learning Algorithms - 26

Ensemble Learning

27

Ensemble Learning

- There are many algorithms for learning a single hypothesis.
- *Ensemble learning* will learn and combine a collection of hypotheses by running the algorithm on different training sets.
- *Bagging* (briefly mentioned in the book) runs a learning algorithm on repeated subsamples of the training set.
- If there are n examples, then a subsample of n examples is generated by sampling with replacement.
- On a test example, each hypothesis casts 1 vote for the class it predicts.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 27

Boosting

- In *boosting*, the hypotheses are learned in sequence.
- Both hypotheses and examples have weights with different purposes.
- After each hypothesis is learned, its weight is based on its error rate, and the weights of the training examples (initially all equal) are also modified.
- On a test example, when each hypothesis predicts a class, its weight is the size of its vote. The ensemble predicts the class with the highest vote.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 28

Example Boosting Algorithm

AdaBoost(*examples, algorithm, iterations*)

1. $n \leftarrow$ number of examples
2. initialize weights $w[1 \dots n]$ to $1/n$
3. **for** i **from** 1 **to** *iterations*
4. $h[i] \leftarrow$ *algorithm*(*examples*)
5. $error \leftarrow$ sum of exs. misclassified by $h[i]$
6. **for** j **from** 1 **to** n
7. **if** $h[i]$ is correct on example j
8. **then** $w[j] \leftarrow w[j] * error / (1 - error)$
9. normalize $w[1 \dots n]$ so it sums to 1
10. weight of $h[i] \leftarrow \log((1 - error) / error)$
11. **return** $h[1 \dots iterations]$ and their weights

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 29

Example Run of AdaBoost

Using the 14 examples as a training set:

- The hypothesis windy = false \leftrightarrow class = pos is wrong on 5 of the 14 examples.
- The weights of the correctly classified examples are multiplied by $5/9$, then all examples are multiplied by $14/10$ so they sum up to 1 again.
- This hypothesis has a weight of $\log(9/5)$.
- Note that after weight updating, the sum of the correctly classified examples equals the sum of the incorrectly classified examples.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 30

Example Run of AdaBoost, Continued

- The next hypothesis must be different from the previous one to have error less than $1/2$.
- Now the hypothesis outlook = overcast \leftrightarrow class = pos has an error rate of $29/90 \approx 0.322$
- The weights of the correctly classified examples are multiplied times $29/61 \approx 0.475$, then all examples are multiplied by $90/58 \approx 1.55$ so they sum up to 1 again.
- This hypothesis has a weight of $\log(61/29)$.

CS 3793 Artificial Intelligence

Numerical Learning Algorithms – 31