CS534: Machine Learning

Thomas G. Dietterich
221C Dearborn Hall
tgd@cs.orst.edu
http://www.cs.orst.edu/~tgd/classes/534

Course Overview

- Introduction:
  - Basic problems and questions in machine learning. Example applications
- Linear Classifiers
- Five Popular Algorithms
  - Decision trees (C4.5)
  - Neural networks (backpropagation)
  - Probabilistic networks (Naïve Bayes; Mixture models)
  - Support Vector Machines (SVMs)
  - Nearest Neighbor Method
- Theories of Learning:
  - PAC, Bayesian, Bias-Variance analysis
- Optimizing Test Set Performance:
  - Overfitting, Penalty methods, Holdout Methods, Ensembles
- Sequential and Spatial Data
  - Hidden Markov models, Conditional Random Fields; Hidden Markov SVMs
- Problem Formulation
  - Designing Input and Output representations

Supervised Learning

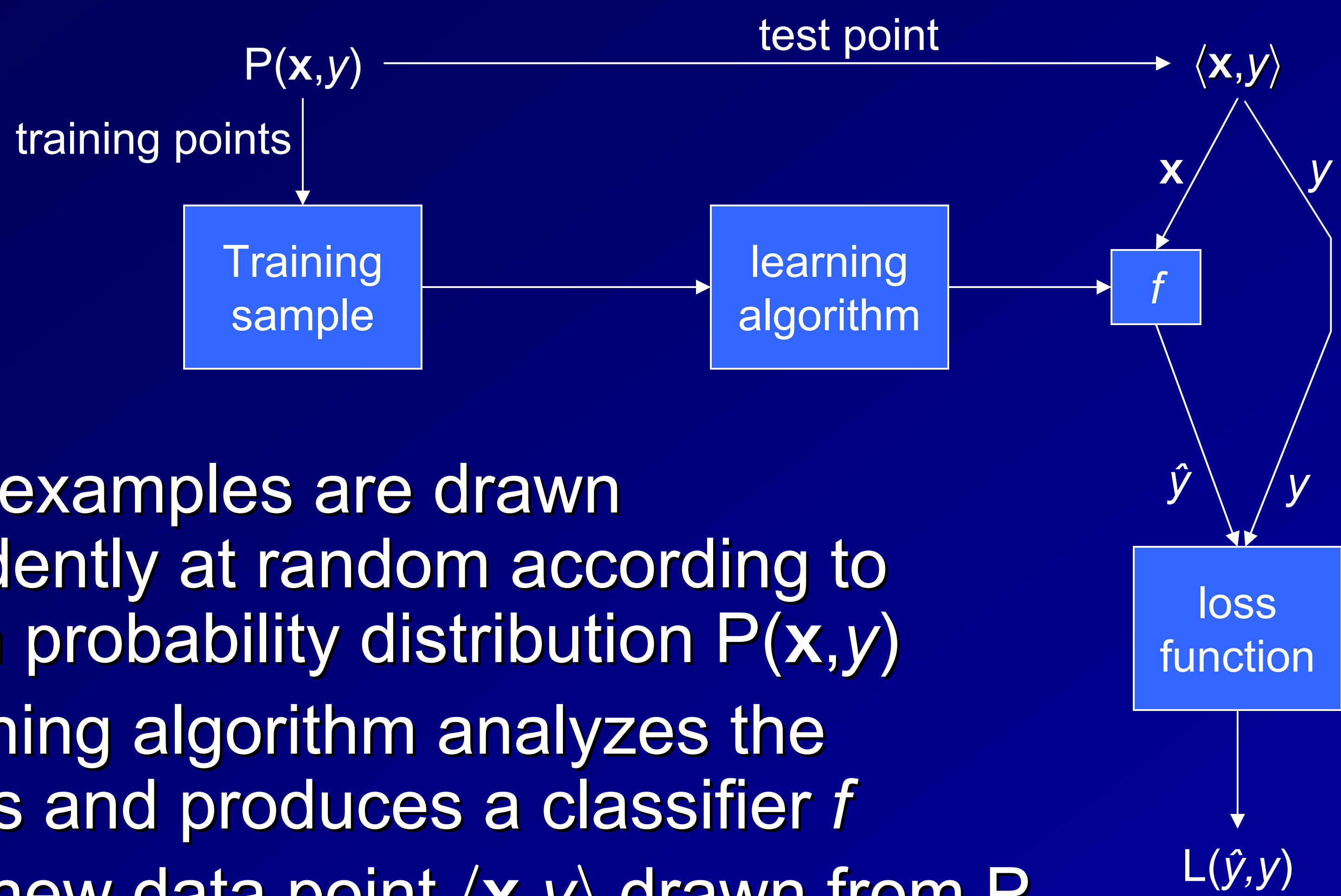
- Given: Training examples $\langle \mathbf{x}, f(\mathbf{x}) \rangle$ for some unknown function f.
- Find: A good approximation to f.

- Example Applications
  - Handwriting recognition
    - x: data from pen motion
    - f(x): letter of the alphabet
  - Disease Diagnosis
    - x: properties of patient (symptoms, lab tests)
    - f(x): disease (or maybe, recommended therapy)
  - Face Recognition
    - x: bitmap picture of person’s face
    - f(x): name of person
  - Spam Detection
    - x: email message
    - f(x): spam or not spam

Appropriate Applications for Supervised Learning

- Situations where there is no human expert
  - x: bond graph of a new molecule
  - f(x): predicted binding strength to AIDS protease molecule
- Situations were humans can perform the task but can’t describe how they do it
  - x: bitmap picture of hand-written character
  - f(x): ascii code of the character
- Situations where the desired function is changing frequently
  - x: description of stock prices and trades for last 10 days
  - f(x): recommended stock transactions
- Situations where each user needs a customized function *f*
  - x: incoming email message
  - f(x): importance score for presenting to the user (or deleting without presenting)

Formal Setting



- Training examples are drawn independently at random according to unknown probability distribution P(**x**,*y*)
- The learning algorithm analyzes the examples and produces a classifier *f*
- Given a new data point ⟨**x**,*y*⟩ drawn from P, the classifier is given **x** and predicts $\hat{y} = f(\mathbf{x})$
- The loss $L(\hat{y},y)$ is then measured
- Goal of the learning algorithm: Find the *f* that minimizes the *expected loss*

Formal Version of Spam Detection

- P(**x**,*y*): distribution of email messages **x** and their true labels *y* ("spam" or "not spam")
- training sample: a set of email messages that have been labeled by the user
- learning algorithm: what we study in this course!
- *f*: the classifier output by the learning algorithm
- test point: A new email message **x** (with its true, but hidden, label *y*)
- loss function $L(\hat{y},y)$:

predicted label $\hat{y}$	true label y: spam	true label y: not spam
spam	0	10
not spam	1	0

Three Main Approaches to Machine Learning

- Learn a classifier: a function f.
- Learn a conditional distribution: a conditional distribution $P(y \mid \mathbf{x})$
- Learn the joint probability distribution: $P(\mathbf{x}, y)$
- In the first two weeks, we will study one example of each method:
  - Learn a classifier: The LMS algorithm
  - Learn a conditional distribution: Logistic regression
  - Learn the joint distribution: Linear discriminant analysis

Infering a classifier *f* from P(*y* | **x**)

- Predict the $\hat{y}$ that minimizes the expected loss:

$$\begin{aligned} f(\mathbf{x}) &= \operatorname*{argmin}_{\widehat{y}} E_{y|\mathbf{x}}[L(\widehat{y}, y)] \\ &= \operatorname*{argmin}_{\widehat{y}} \sum_{y} P(y|\mathbf{x}) L(\widehat{y}, y) \end{aligned}$$

Example: Making the spam decision

- Suppose our spam detector predicts that P(*y*="spam" | **x**) = 0.6. What is the optimal classification decision *ŷ*?
- Expected loss of *ŷ* = "spam" is 0 * 0.6 + 10 * 0.4 = 4
- Expected loss of *ŷ* = "no spam" is 1 * 0.6 + 0 * 0.4 = 0.6
- Therefore, the optimal prediction is "no spam"

predicted label ŷ	true label *y*	
	spam	not spam
spam	0	10
not spam	1	0
P(*y*|**x**)	0.6	0.4

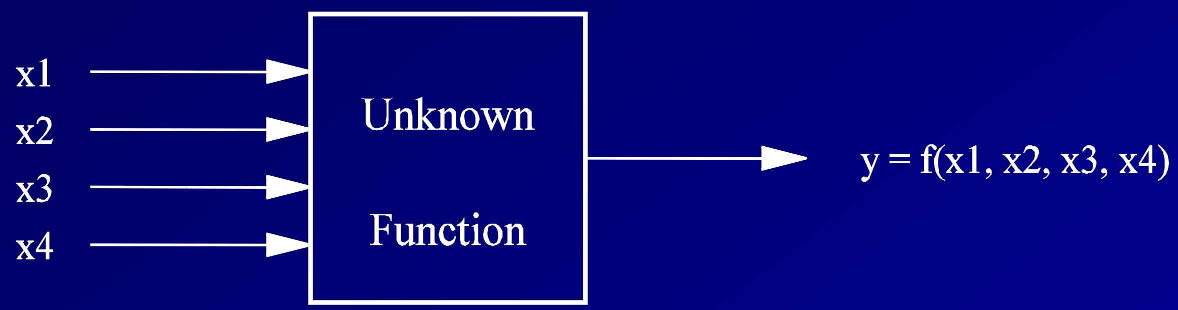
Inferring a classifier from the joint distribution P($\mathbf{x}$,y)

- We can compute the conditional distribution according to the definition of conditional probability:

$$P(y = k|\mathbf{x}) = \frac{P(\mathbf{x}, y = k)}{\sum_j P(\mathbf{x}, y = j)}.$$

- In words, compute P($\mathbf{x}$, y=k) for each value of k. Then normalize these numbers.
- Compute $\hat{y}$ using the method from the previous slide

Fundamental Problem of Machine Learning: It is ill-posed



Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Learning Appears Impossible

- There are $2^{16} = 65536$ possible boolean functions over four input features. We can't figure out which one is correct until we've seen every possible input-output pair. After 7 examples, we still have 2^9 possibilities.

x_1	x_2	x_3	x_4	y
0	0	0	0	?
0	0	0	1	?
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	?
1	0	0	0	?
1	0	0	1	1
1	0	1	0	?
1	0	1	1	?
1	1	0	0	0
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

Solution: Work with a restricted hypothesis space

- Either by applying prior knowledge or by guessing, we choose a space of hypotheses *H* that is smaller than the space of all possible functions:
  - simple conjunctive rules
  - *m*-of-*n* rules
  - linear functions
  - multivariate Gaussian joint probability distributions
  - etc.

Illustration: Simple Conjunctive Rules

- There are only 16 simple conjunctions (no negation)
- However, no simple rule explains the data. The same is true for simple clauses

Rule	Counterexample
$\text{true} \Leftrightarrow y$	1
$x_1 \Leftrightarrow y$	3
$x_2 \Leftrightarrow y$	2
$x_3 \Leftrightarrow y$	1
$x_4 \Leftrightarrow y$	7
$x_1 \wedge x_2 \Leftrightarrow y$	3
$x_1 \wedge x_3 \Leftrightarrow y$	3
$x_1 \wedge x_4 \Leftrightarrow y$	3
$x_2 \wedge x_3 \Leftrightarrow y$	3
$x_2 \wedge x_4 \Leftrightarrow y$	3
$x_3 \wedge x_4 \Leftrightarrow y$	4
$x_1 \wedge x_2 \wedge x_3 \Leftrightarrow y$	3
$x_1 \wedge x_2 \wedge x_4 \Leftrightarrow y$	3
$x_1 \wedge x_3 \wedge x_4 \Leftrightarrow y$	3
$x_2 \wedge x_3 \wedge x_4 \Leftrightarrow y$	3
$x_1 \wedge x_2 \wedge x_3 \wedge x_4 \Leftrightarrow y$	3

A larger hypothesis space: *m*-of-*n* rules

- At least *m* of the *n* variables must be true
- There are 32 possible rules
- Only one rule is consistent!

	Counterexample			
variables	1-of	2-of	3-of	4-of
$\{x_1\}$	3	–	–	–
$\{x_2\}$	2	–	–	–
$\{x_3\}$	1	–	–	–
$\{x_4\}$	7	–	–	–
$\{x_1, x_2\}$	3	3	–	–
$\{x_1, x_3\}$	4	3	–	–
$\{x_1, x_4\}$	6	3	–	–
$\{x_2, x_3\}$	2	3	–	–
$\{x_2, x_4\}$	2	3	–	–
$\{x_3, x_4\}$	4	4	–	–
$\{x_1, x_2, x_3\}$	1	3	3	–
$\{x_1, x_2, x_4\}$	2	3	3	–
$\{x_1, x_3, x_4\}$	1	***	3	–
$\{x_2, x_3, x_4\}$	1	5	3	–
$\{x_1, x_2, x_3, x_4\}$	1	5	3	3

Two Views of Learning

- View 1: **Learning is the removal of our remaining uncertainty**
  - Suppose we *knew* that the unknown function was an *m*-of-*n* boolean function. Then we could use the training examples to *deduce* which function it is.
- View 2: **Learning requires <u>guessing</u> a good, small hypothesis class**
  - We can start with a very small class and enlarge it until it contains an hypothesis that fits the data

We could be wrong!

- Our prior “knowledge” might be wrong
- Our guess of the hypothesis class could be wrong
  – The smaller the class, the more likely we are wrong

Two Strategies for Machine Learning

- Develop Languages for Expressing Prior Knowledge
  - Rule grammars, stochastic models, Bayesian networks
  - (Corresponds to the Prior Knowledge view)
- Develop Flexible Hypothesis Spaces
  - Nested collections of hypotheses: decision trees, neural networks, cases, SVMs
  - (Corresponds to the Guessing view)
- In either case we must develop algorithms for finding an hypothesis that fits the data

Terminology

- Training example. An example of the form $\langle \mathbf{x}, y \rangle$. $\mathbf{x}$ is usually a vector of features, y is called the class label. We will index the features by j, hence x_j is the j-th feature of $\mathbf{x}$. The number of features is n.
- Target function. The true function f, the true conditional distribution $P(y \mid \mathbf{x})$, or the true joint distribution $P(\mathbf{x}, y)$.
- Hypothesis. A proposed function or distribution h believed to be similar to f or P.
- Concept. A boolean function. Examples for which $f(\mathbf{x})=1$ are called positive examples or positive instances of the concept. Examples for which $f(\mathbf{x})=0$ are called negative examples or negative instances.

Terminology

- Classifier. A discrete-valued function. The possible values $f(\mathbf{x}) \in \{1, \ldots, K\}$ are called the classes or class labels.
- Hypothesis space. The space of all hypotheses that can, in principle, be output by a particular learning algorithm.
- Version Space. The space of all hypotheses in the hypothesis space that have not yet been ruled out by a training example.
- Training Sample (or Training Set or Training Data): a set of N training examples drawn according to $P(\mathbf{x}, y)$.
- Test Set: A set of training examples used to evaluate a proposed hypothesis h.
- Validation Set: A set of training examples (typically a subset of the training set) used to guide the learning algorithm and prevent overfitting.

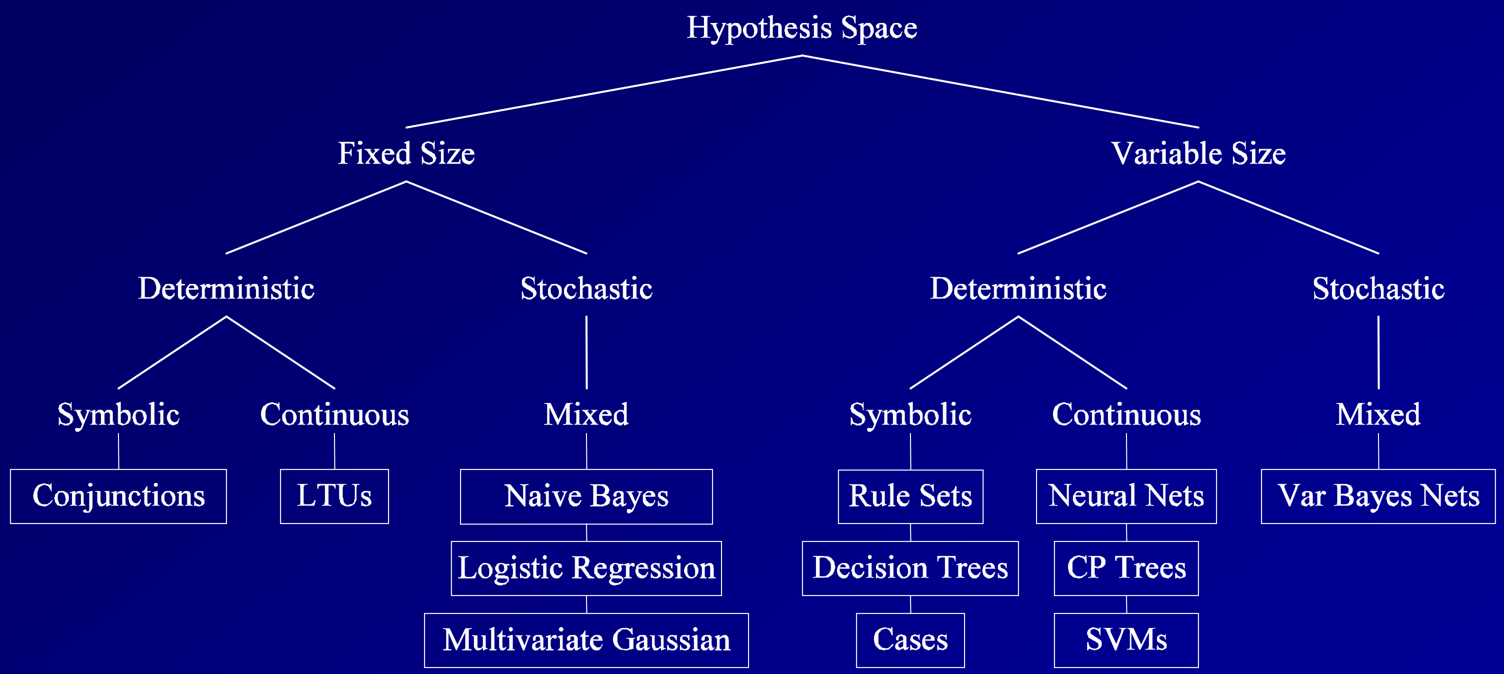
Key Issues in Machine Learning

- What are good hypothesis spaces?
  - which spaces have been useful in practical applications?
- What algorithms can work with these spaces?
  - Are there general design principles for learning algorithms?
- How can we optimize accuracy on future data points?
  - This is related to the problem of "overfitting"
- How can we have confidence in the results? (the statistical question)
  - How much training data is required to find an accurate hypotheses?
- Are some learning problems computational intractable? (the computational question)
- How can we formulate application problems as machine learning problems? (the engineering question)

A framework for hypothesis spaces

- Size: Does the hypothesis space have a fixed size or a variable size?
  - fixed-sized spaces are easier to understand, but variable-sized spaces are generally more useful. Variable-sized spaces introduce the problem of overfitting
- Stochasticity. Is the hypothesis a classifier, a conditional distribution, or a joint distribution?
  - This affects how we evaluate hypotheses. For a deterministic hypothesis, a training example is either *consistent* (correctly predicted) or *inconsistent* (incorrectly predicted). For a stochastic hypothesis, a trianing example is *more likely* or *less likely*.
- Parameterization. Is each hypothesis described by a set of symbolic (discrete) choices or is it described by a set of continuous parameters? If both are required, we say the space has a mixed parameterization.
  - Discrete parameters must be found by combinatorial search methods; continuous parameters can be found by numerical search methods

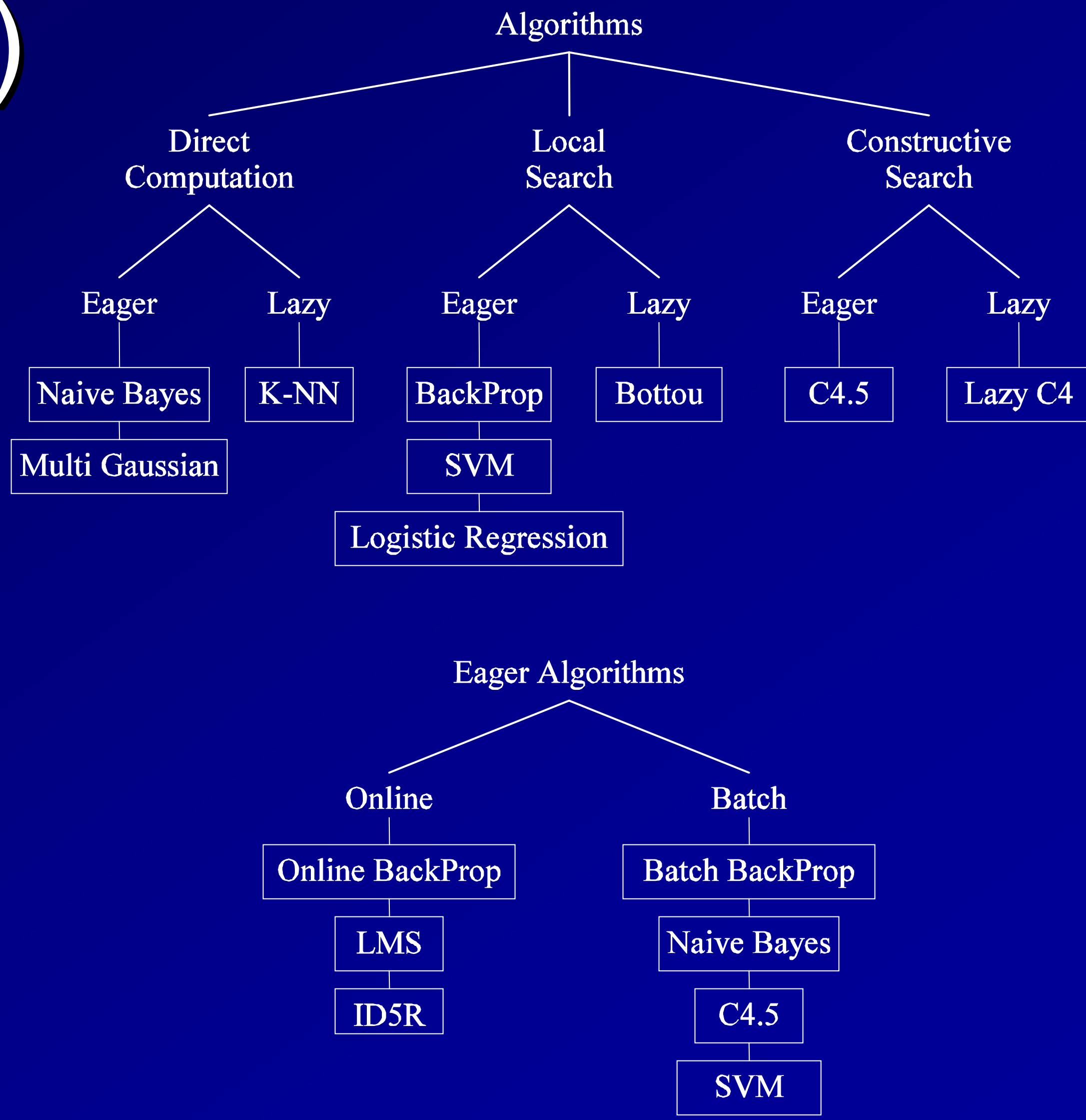
A Framework for Hypothesis Spaces (2)



A Framework for Learning Algorithms

- Search Procedure
  - Direct Computation: solve for the hypothesis directly
  - Local Search: start with an initial hypothesis, make small improvements until a local maximum
  - Constructive Search: start with an empty hypothesis, gradually add structure to it until a local optimum
- Timing
  - Eager: analyze training data and construct an explicit hypothesis
  - Lazy: store the training data and wait until a test data point is presented, then construct an ad hoc hypothesis to classify that one data point
- Online vs. Batch (for eager algorithms)
  - Online: analyze each training example as it is presented
  - Batch: collect examples, analyze them in a batch, output an hypothesis

A Framework for Learning Algorithms (2)



Linear Threshold Units

$$h(\mathbf{x}) = \begin{cases} +1 & \text{if} \quad w_1 x_1 + \ldots + w_n x_n \geq w_0 \\ -1 & \text{otherwise} \end{cases}$$

- We assume that each feature x_j and each weight w_j is a real number (we will relax this later)
- We will study three different algorithms for learning linear threshold units:
  - Perceptron: classifier
  - Logistic Regression: conditional distribution
  - Linear Discriminant Analysis: joint distribution

What can be represented by an LTU:

- Conjunctions

$$x_1 \wedge x_2 \wedge x_4 \Leftrightarrow y$$
$$1 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 + 1 \cdot x_4 \geq 3$$

- At least *m*-of-*n*

$$\text{at-least-2-of}\{x_1, x_3, x_4\} \Leftrightarrow y$$
$$1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 2$$

Things that cannot be represented:

- Non-trivial disjunctions:

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \Leftrightarrow y$$

$1 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 2$ predicts $f(\langle 0110 \rangle) = 1$.

- Exclusive-OR:

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \Leftrightarrow y$$

A canonical representation

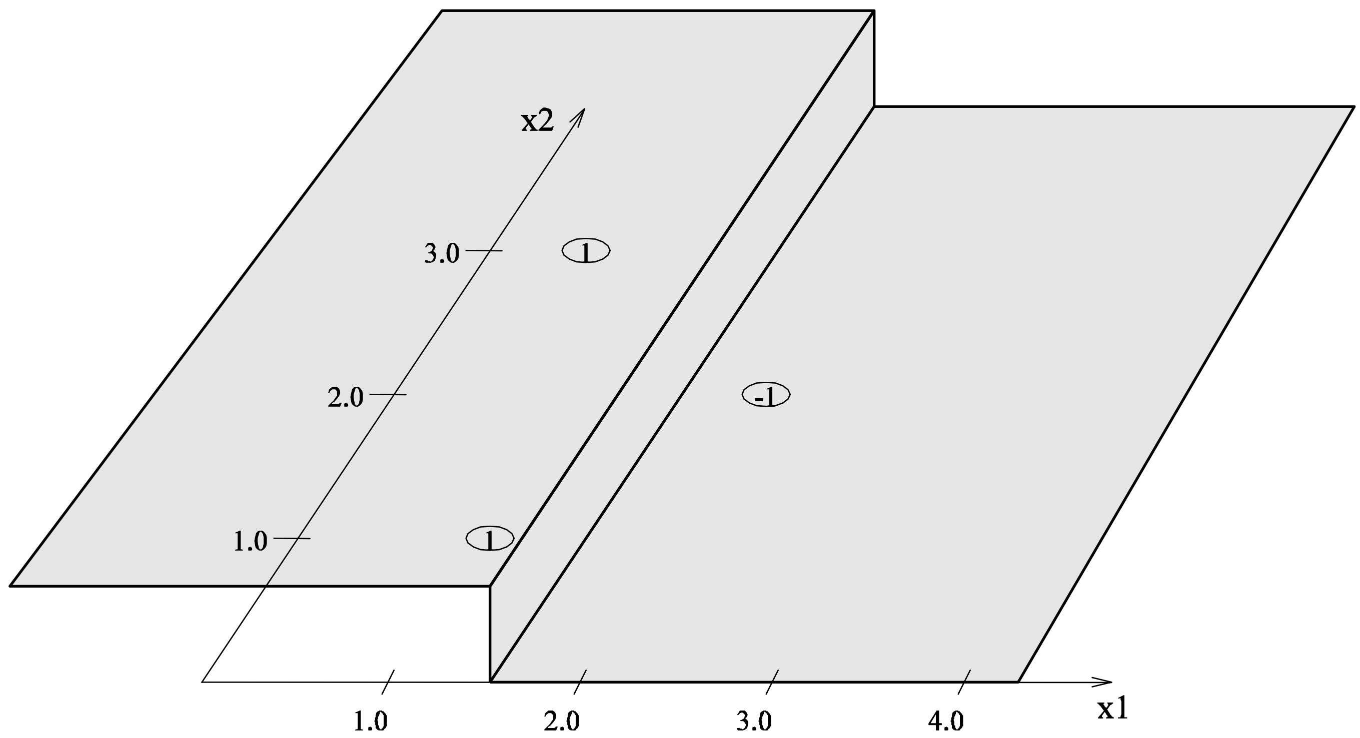
- Given a training example of the form
  $(\langle x_1, x_2, x_3, x_4 \rangle, y)$
- transform it to
  $(1, x_1, x_2, x_3, x_4 \rangle, y)$
- The parameter vector will then be
  $\mathbf{w} = \langle w_0, w_1, w_2, w_3, w_4 \rangle$.
- We will call the *unthresholded* hypothesis $u(\mathbf{x},\mathbf{w})$
  $u(\mathbf{x},\mathbf{w}) = \mathbf{w} \cdot \mathbf{x}$
- Each hypothesis can be written
  $h(\mathbf{x}) = \mathrm{sgn}(u(\mathbf{x},\mathbf{w}))$
- Our goal is to find **w**.

The LTU Hypothesis Space

- Fixed size: There are $O\left(2^{n^2}\right)$ distinct linear threshold units over *n* boolean features
- Deterministic
- Continuous parameters

Geometrical View

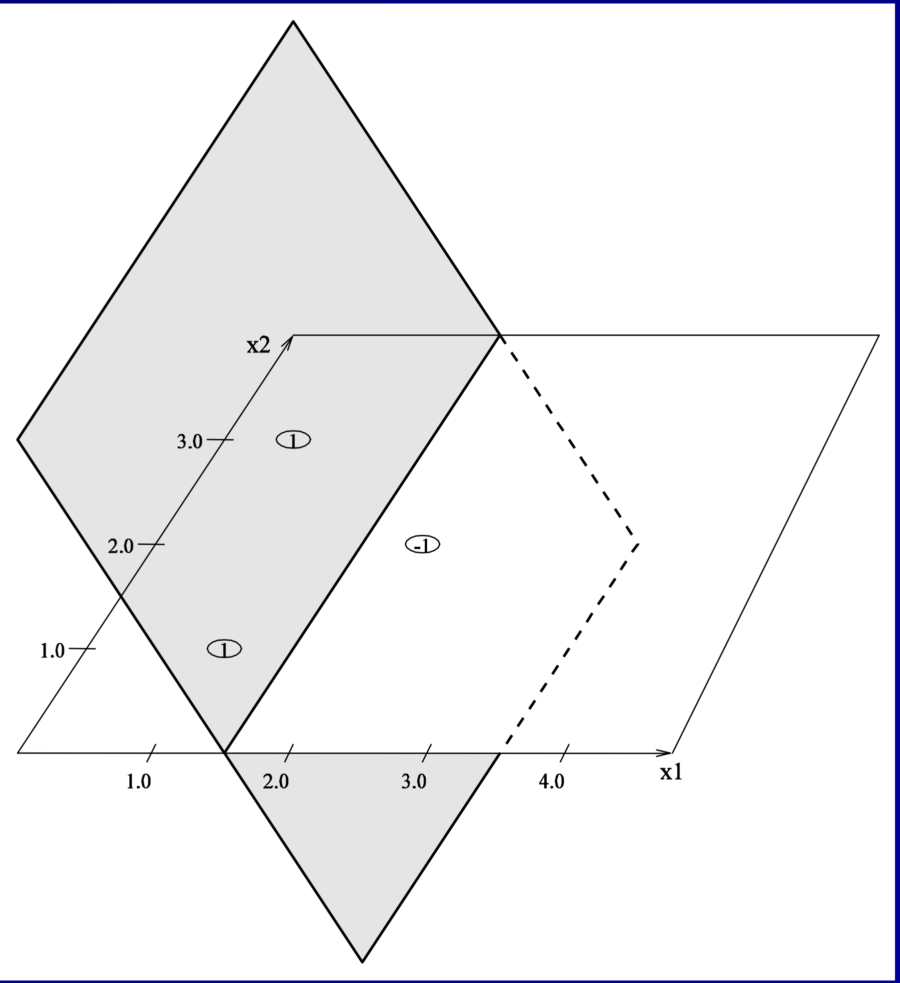
- Consider three training examples: $(\langle 1.0, 1.0\rangle, +1)$
  $(\langle 0.5, 3.0\rangle, +1)$
  $(\langle 2.0, 2.0\rangle, -1)$
- We want a classifier that looks like the following:



The Unthresholded Discriminant Function is a Hyperplane

- The equation
  $u(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$
  is a plane

$$\hat{y} = \begin{cases} +1 & \text{if } u(\mathbf{x}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$



Machine Learning and Optimization

- When learning a classifier, the natural way to formulate the learning problem is the following:
  - Given:
    - A set of *N* training examples
      $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$
    - A loss function *L*
  - Find:
    - The weight vector **w** that minimizes the expected loss on the training data

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} L(\mathrm{sgn}(\mathbf{w} \cdot \mathbf{x}_i), y_i).$$

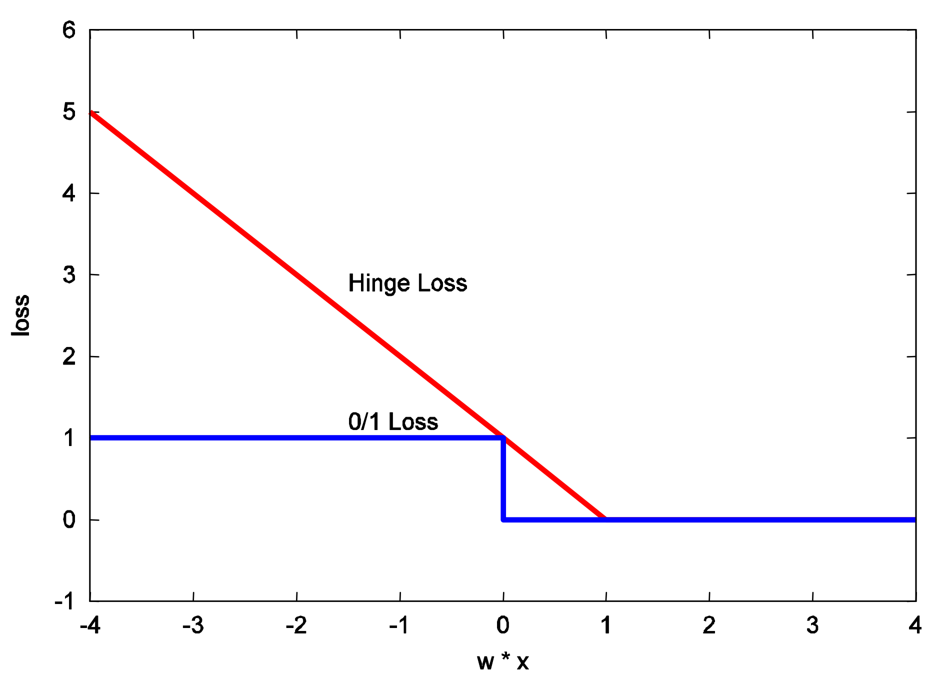
- In general, machine learning algorithms apply some optimization algorithm to find a good hypothesis. In this case, *J* is <u>piecewise constant</u>, which makes this a difficult problem

Approximating the expected loss by a smooth function

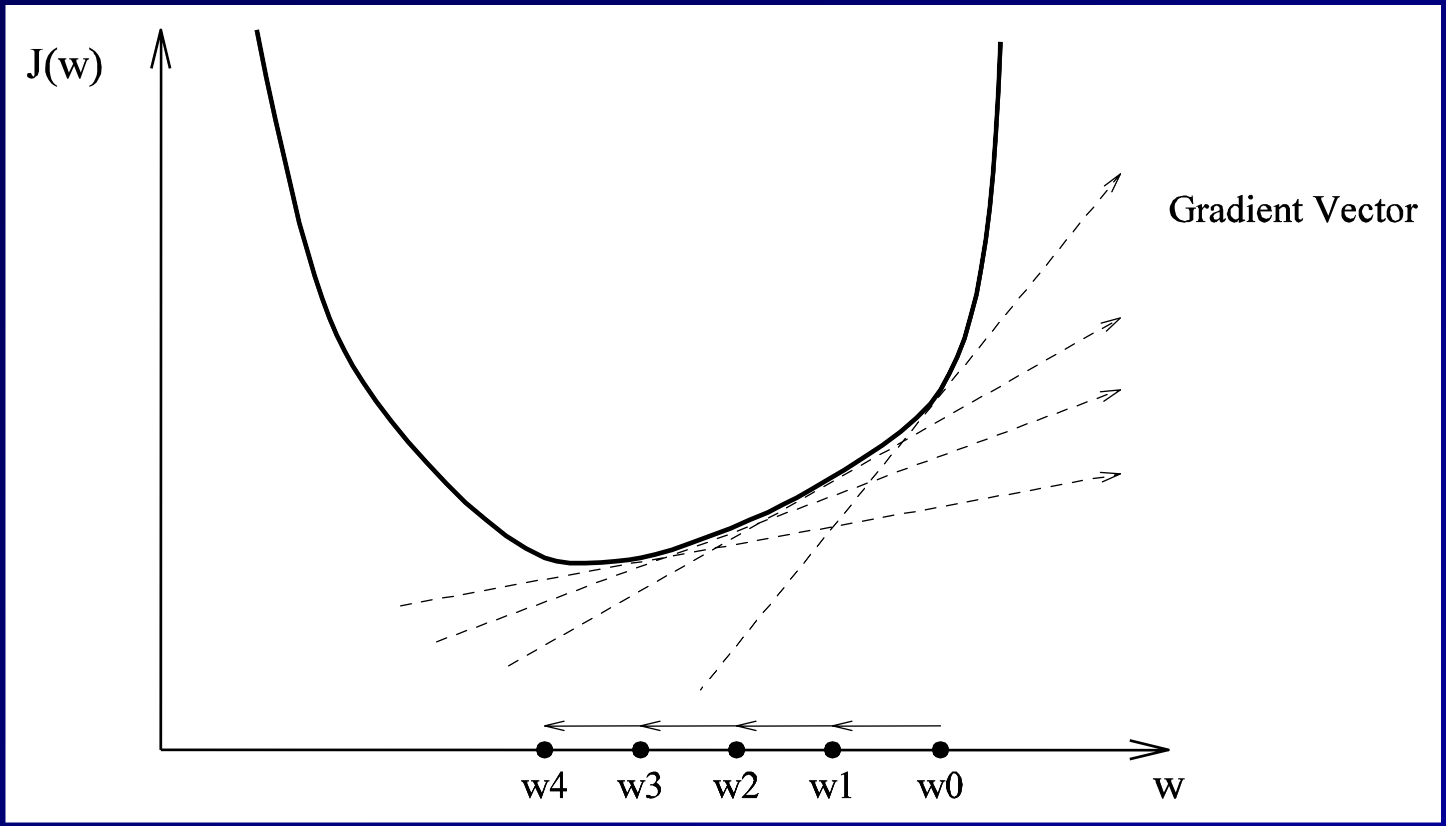
- Simplify the optimization problem by replacing the original objective function by a smooth, differentiable function. For example, consider the *hinge loss*:

$$\tilde{J}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} max(0, 1 - y_i\mathbf{w}\cdot\mathbf{x}_i)$$

When $y = 1$



Minimizing $\tilde{J}$ by Gradient Descent Search



- Start with weight vector $\mathbf{w}_0$
- Compute gradient $$\nabla \tilde{J}(\mathbf{w}_0) = \left(\frac{\partial \tilde{J}(\mathbf{w}_0)}{\partial w_0}, \frac{\partial \tilde{J}(\mathbf{w}_0)}{\partial w_1}, \ldots, \frac{\partial \tilde{J}(\mathbf{w}_0)}{\partial w_n} \right)$$
- Compute $\mathbf{w}_1 = \mathbf{w}_0 - \eta \nabla \tilde{J}(\mathbf{w}_0)$
  where η is a "step size" parameter
- Repeat until convergence

Computing the Gradient

$$\text{Let } \tilde{J}_i(\mathbf{w}) = \max(0, -y_i \mathbf{w} \cdot \mathbf{x}_i)$$

$$\frac{\partial \tilde{J}(\mathbf{w})}{\partial w_k} = \frac{\partial}{\partial w_k} \left(\frac{1}{N} \sum_{i=1}^{N} \tilde{J}_i(\mathbf{w}) \right) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\partial}{\partial w_k} \tilde{J}_i(\mathbf{w}) \right)$$

$$\frac{\partial \tilde{J}_i(\mathbf{w})}{\partial w_k} = \frac{\partial}{\partial w_k} \max \left(0, -y_i \sum_j w_j x_{ij} \right) = \begin{cases} 0 & \text{if } y_i \sum_j w_j x_{ij} > 0 \\ -y_i x_{ik} & \text{otherwise} \end{cases}$$

Batch Perceptron Algorithm

Given: training examples $(\mathbf{x}_i, y_i)$, $i = 1 \ldots N$

Let $\mathbf{w} = (0, 0, 0, 0, \ldots, 0)$ be the initial weight vector.

Let $\mathbf{g} = (0, 0, \ldots, 0)$ be the gradient vector.

Repeat until convergence

For $i = 1$ **to** N **do**

$u_i = \mathbf{w} \cdot \mathbf{x}_i$

If $(y_i \cdot u_i < 0)$

For $j = 1$ **to** n **do**

$g_j = g_j - y_i \cdot x_{ij}$

$\mathbf{g} := \mathbf{g}/N$

$\mathbf{w} := \mathbf{w} - \eta \mathbf{g}$

Simplest case: $\eta = 1$, don't normalize g: "Fixed Increment Perceptron"

Online Perceptron Algorithm

Let $\mathbf{w} = (0, 0, 0, 0, \ldots, 0)$ be the initial weight vector.

Repeat forever

  Accept training example i: $\langle \mathbf{x}_i, y_i \rangle$

  $u_i = \mathbf{w} \cdot \mathbf{x}_i$

  If $(y_i u_i < 0)$

    For $j = 1$ **to** n **do**

      $g_j := y_i \cdot x_{ij}$

    $\mathbf{w} := \mathbf{w} + \eta \mathbf{g}$

This is called stochastic gradient descent because the overall gradient is approximated by the gradient from each individual example

Learning Rates and Convergence

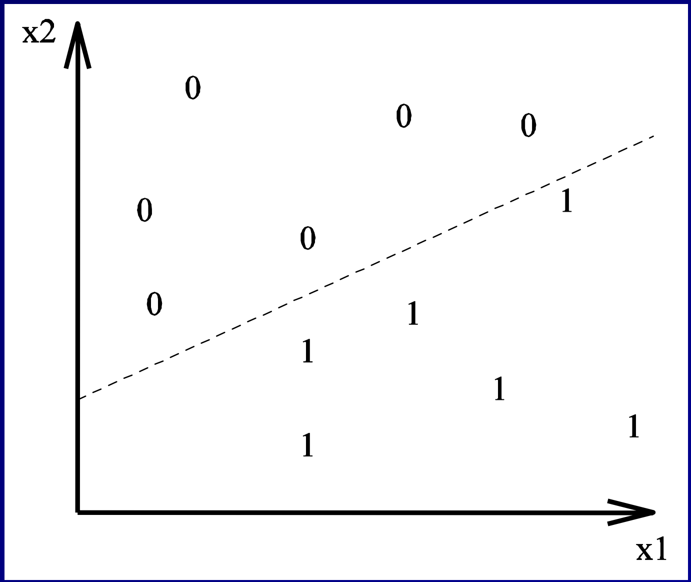
- The learning rate η must decrease to zero in order to guarantee convergence. The online case is known as the Robbins-Munro algorithm. It is guaranteed to converge under the following assumptions:

$$\begin{aligned} \lim_{t \to \infty} \eta_t &= 0 \\ \sum_{t=0}^{\infty} \eta_t &= \infty \\ \sum_{t=0}^{\infty} \eta_t^2 &< \infty \end{aligned}$$

- The learning rate is also called the <u>step size</u>. Some algorithms (e.g., Newton's method, conjugate gradient) choose the stepsize automatically and converge faster
- There is only one "basin" for linear threshold units, so a local minimum is the global minimum. Choosing a good starting point can make the algorithm converge faster

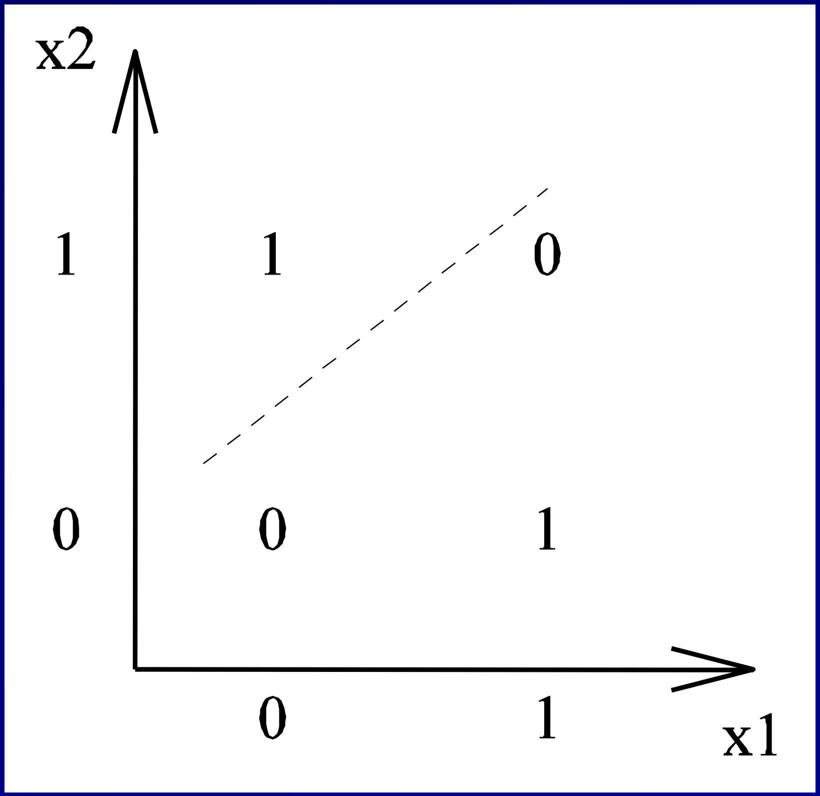
Decision Boundaries

- A classifier can be viewed as partitioning the input space or feature space X into decision regions



- A linear threshold unit always produces a linear decision boundary. A set of points that can be separated by a linear decision boundary is said to be linearly separable.

Exclusive-OR is Not Linearly Separable



Extending Perceptron to More than Two Classes

- If we have K > 2 classes, we can learn a separate LTU for each class. Let $\mathbf{w}_k$ be the weight vector for class k. We train it by treating examples from class $y = k$ as the positive examples and treating the examples from all other classes as negative examples. Then we classify a new data point $\mathbf{x}$ according to

$$\hat{y} = \operatorname*{argmax}_k \mathbf{w}_k \cdot \mathbf{x}.$$

Summary of Perceptron algorithm for LTUs

- Directly Learns a Classifier
- Local Search
  - Begins with an initial weight vector. Modifies it iterative to minimize an error function. The error function is loosely related to the goal of minimizing the number of classification errors
- Eager
  - The classifier is constructed from the training examples
  - The training examples can then be discarded
- Online or Batch
  - Both variants of the algorithm can be used

Logistic Regression

- Learn the conditional distribution P(y | **x**)
- Let $p_y(\mathbf{x}; \mathbf{w})$ be our estimate of P(y | **x**), where **w** is a vector of adjustable parameters. Assume only two classes y = 0 and y = 1, and

$$p_1(\mathbf{x}; \mathbf{w}) = \frac{\exp \mathbf{w} \cdot \mathbf{x}}{1 + \exp \mathbf{w} \cdot \mathbf{x}}.$$

$$p_0(\mathbf{x}; \mathbf{w}) = 1 - p_1(\mathbf{x}; \mathbf{w}).$$

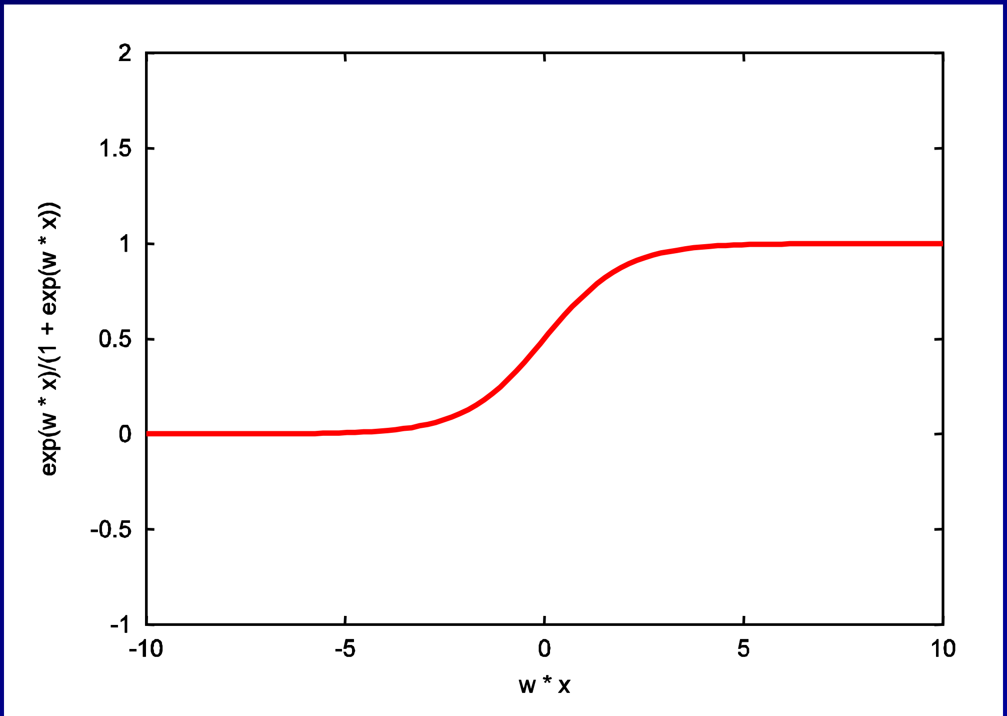
- On the homework, you will show that this is equivalent to

$$\log \frac{p_1(\mathbf{x}; \mathbf{w})}{p_0(\mathbf{x}; \mathbf{w})} = \mathbf{w} \cdot \mathbf{x}.$$

- In other words, the log odds of class 1 is a linear function of **x**.

Why the exp function?

- One reason: A linear function has a range from [$-\infty$, ∞] and we need to force it to be positive and sum to 1 in order to be a probability:



Deriving a Learning Algorithm

- Since we are fitting a conditional probability distribution, we no longer seek to minimize the loss on the training data. Instead, we seek to find the probability distribution *h* that is most likely given the training data
- Let S be the training sample. Our goal is to find *h* to maximize P(*h* | S):

$$\begin{aligned}
\operatorname*{argmax}_h P(h|S) &= \operatorname*{argmax}_h \frac{P(S|h)P(h)}{P(S)} && \text{by Bayes' Rule} \\
&= \operatorname*{argmax}_h P(S|h)P(h) && \text{because } P(S) \text{ doesn't depend on } h \\
&= \operatorname*{argmax}_h P(S|h) && \text{if we assume } P(h) = \text{uniform} \\
&= \operatorname*{argmax}_h \log P(S|h) && \text{because log is monotonic}
\end{aligned}$$

The distribution P(S|*h*) is called the likelihood function. The log likelihood is frequently used as the objective function for learning. It is often written as $\ell(\mathbf{w})$.

The *h* that maximizes the likelihood on the training data is called the maximum likelihood estimator (MLE)

Computing the Likelihood

- In our framework, we assume that each training example $(\mathbf{x}_i, y_i)$ is drawn from the same (but unknown) probability distribution $P(\mathbf{x}, y)$. This means that the log likelihood of S is the sum of the log likelihoods of the individual training examples:

$$\begin{aligned} \log P(S|h) &= \log \prod_i P(\mathbf{x}_i, y_i|h) \\ &= \sum_i \log P(\mathbf{x}_i, y_i|h) \end{aligned}$$

Computing the Likelihood (2)

- Recall that *any* joint distribution P(a,b) can be factored as P(a|b) P(b). Hence, we can write

$$\begin{aligned}\operatorname*{argmax}_h \log P(S|h) &= \operatorname*{argmax}_h \sum_i \log P(\mathbf{x}_i, y_i|h) \\ &= \operatorname*{argmax}_h \sum_i \log P(y_i|\mathbf{x}_i, h) P(\mathbf{x}_i|h)\end{aligned}$$

- In our case, P(**x** | *h*) = P(**x**), because it does not depend on *h*, so

$$\begin{aligned}\operatorname*{argmax}_h \log P(S|h) &= \operatorname*{argmax}_h \sum_i \log P(y_i|\mathbf{x}_i, h) P(\mathbf{x}_i|h) \\ &= \operatorname*{argmax}_h \sum_i \log P(y_i|\mathbf{x}_i, h)\end{aligned}$$

Log Likelihood for Conditional Probability Estimators

- We can express the log likelihood in a compact form known as the cross entropy.
- Consider an example $(\mathbf{x}_i, y_i)$
  – If $y_i = 0$, the log likelihood is $\log [1 - p_1(\mathbf{x}; \mathbf{w})]$
  – if $y_i = 1$, the log likelihood is $\log [p_1(\mathbf{x}; \mathbf{w})]$
- These cases are mutually exclusive, so we can combine them to obtain:

$$\ell(y_i; \mathbf{x}_i, \mathbf{w}) = \log P(y_i \mid \mathbf{x}_i, \mathbf{w}) = (1 - y_i) \log[1 - p_1(\mathbf{x}_i; \mathbf{w})] + y_i \log p_1(\mathbf{x}_i; \mathbf{w})$$

- The goal of our learning algorithm will be to find $\mathbf{w}$ to maximize

$$J(\mathbf{w}) = \sum_i \ell(y_i; \mathbf{x}_i, \mathbf{w})$$

Fitting Logistic Regression by Gradient Ascent

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_i \frac{\partial}{\partial w_j}\ell(y_i; \mathbf{x}_i, \mathbf{w})$$

$$\begin{aligned}
\frac{\partial}{\partial w_j}\ell(y_i; \mathbf{x}_i, \mathbf{w}) &= \frac{\partial}{\partial w_j}\left((1-y_i)\log[1-p_1(\mathbf{x}_i;\mathbf{w})] + y_1 \log p_1(\mathbf{x}_i;\mathbf{w})\right) \\
&= (1-y_i)\frac{1}{1-p_1(\mathbf{x}_i;\mathbf{w})}\left(-\frac{\partial p_1(\mathbf{x}_i;\mathbf{w})}{\partial w_j}\right) + y_i \frac{1}{p_1(\mathbf{x}_i;\mathbf{w})}\left(\frac{\partial p_1(\mathbf{x}_i;\mathbf{w})}{\partial w_j}\right) \\
&= \left[\frac{y_i}{p_1(\mathbf{x}_i;\mathbf{w})} - \frac{(1-y_i)}{1-p_1(\mathbf{x}_i;\mathbf{w})}\right]\left(\frac{\partial p_1(\mathbf{x}_i;\mathbf{w})}{\partial w_j}\right) \\
&= \left[\frac{y_i(1-p_1(\mathbf{x}_i;\mathbf{w})) - (1-y_i)p_1(\mathbf{x}_i;\mathbf{w})}{p_1(\mathbf{x}_i;\mathbf{w})(1-p_1(\mathbf{x}_i;\mathbf{w}))}\right]\left(\frac{\partial p_1(\mathbf{x}_i;\mathbf{w})}{\partial w_j}\right) \\
&= \left[\frac{y_i - p_1(\mathbf{x}_i;\mathbf{w})}{p_1(\mathbf{x}_i;\mathbf{w})(1-p_1(\mathbf{x}_i;\mathbf{w}))}\right]\left(\frac{\partial p_1(\mathbf{x}_i;\mathbf{w})}{\partial w_j}\right)
\end{aligned}$$

Gradient Computation (continued)

- Note that p_1 can also be written as

$$p_1(\mathbf{x}_i; \mathbf{w}) = \frac{1}{(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])}.$$

- From this, we obtain:

$$\begin{aligned}
\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} &= -\frac{1}{(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])^2} \frac{\partial}{\partial w_j}(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i]) \\
&= -\frac{1}{(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])^2} \exp[-\mathbf{w} \cdot \mathbf{x}_i] \frac{\partial}{\partial w_j}(-\mathbf{w} \cdot \mathbf{x}_i) \\
&= -\frac{1}{(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])^2} \exp[-\mathbf{w} \cdot \mathbf{x}_i](-x_{ij}) \\
&= p_1(\mathbf{x}_i; \mathbf{w})(1 - p_1(\mathbf{x}_i; \mathbf{w}))x_{ij}
\end{aligned}$$

Completing the Gradient Computation

- The gradient of the log likelihood of a single point is therefore

$$\begin{aligned}\frac{\partial}{\partial w_j}\ell(y_i;\mathbf{x}_i,\mathbf{w}) &= \left[\frac{y_i - p_1(\mathbf{x}_i;\mathbf{w})}{p_1(\mathbf{x}_i;\mathbf{w})(1-p_1(\mathbf{x}_i;\mathbf{w}))}\right]\left(\frac{\partial p_1(\mathbf{x}_i;\mathbf{w})}{\partial w_j}\right) \\ &= \left[\frac{y_i - p_1(\mathbf{x}_i;\mathbf{w})}{p_1(\mathbf{x}_i;\mathbf{w})(1-p_1(\mathbf{x}_i;\mathbf{w}))}\right] p_1(\mathbf{x}_i;\mathbf{w})(1-p_1(\mathbf{x}_i;\mathbf{w}))x_{ij} \\ &= (y_i - p_1(\mathbf{x}_i;\mathbf{w}))x_{ij}\end{aligned}$$

- The overall gradient is

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_i (y_i - p_1(\mathbf{x}_i;\mathbf{w}))x_{ij}$$

Batch Gradient Ascent for Logistic Regression

Given: training examples $(\mathbf{x}_i, y_i)$, $i = 1 \ldots N$

Let $\mathbf{w} = (0, 0, 0, 0, \ldots, 0)$ be the initial weight vector.

Repeat until convergence

  Let $\mathbf{g} = (0, 0, \ldots, 0)$ be the gradient vector.

  For $i = 1$ **to** N **do**

    $p_i = 1/(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])$

    $\text{error}_i = y_i - p_i$

    For $j = 1$ **to** n **do**

      $g_j = g_j + \text{error}_i \cdot x_{ij}$

  $\mathbf{w} := \mathbf{w} + \eta \mathbf{g}$ step in direction of increasing gradient

- An online gradient ascent algorithm can be constructed, of course
- Most statistical packages use a second-order (Newton-Raphson) algorithm for faster convergence. Each iteration of the second-order method can be viewed as a weighted least squares computation, so the algorithm is known as Iteratively-Reweighted Least Squares (IRLS)

Logistic Regression Implements a Linear Discriminant Function

- In the 2-class 0/1 loss function case, we should predict ŷ = 1 if

$$
\begin{aligned}
E_{y|\mathbf{x}}[L(0,y)] &> E_{y|\mathbf{x}}[L(1,y)] \\
\sum_y P(y|\mathbf{x})L(0,y) &> \sum_y P(y|\mathbf{x})L(1,y) \\
P(y=0|\mathbf{x})L(0,0) + P(y=1|\mathbf{x})L(0,1) &> P(y=0|\mathbf{x})L(1,0) + P(y=1|\mathbf{x})L(1,1) \\
P(y=1|\mathbf{x}) &> P(y=0|\mathbf{x}) \\
\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} &> 1 \quad \text{if } P(y=0|X) \neq 0 \\
\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} &> 0 \\
\mathbf{w} \cdot \mathbf{x} &> 0
\end{aligned}
$$

- A similar derivation can be done for arbitrary L(0,1) and L(1,0).

Extending Logistic Regression to K > 2 classes

- Choose class K to be the "reference class" and represent each of the other classes as a logistic function of the odds of class *k* versus class K:

$$
\begin{aligned}
\log \frac{P(y=1|\mathbf{x})}{P(y=K|\mathbf{x})} &= \mathbf{w}_1 \cdot \mathbf{x} \\
\log \frac{P(y=2|\mathbf{x})}{P(y=K|\mathbf{x})} &= \mathbf{w}_2 \cdot \mathbf{x} \\
&\vdots \\
\log \frac{P(y=K-1|\mathbf{x})}{P(y=K|\mathbf{x})} &= \mathbf{w}_{K-1} \cdot \mathbf{x}
\end{aligned}
$$

- Gradient ascent can be applied to simultaneously train all of these weight vectors $\mathbf{w}_k$

Logistic Regression for K > 2 (continued)

- The conditional probability for class k ≠ K can be computed as

$$P(y = k|\mathbf{x}) \;=\; \frac{\exp(\mathbf{w}_k \cdot \mathbf{x})}{1 + \sum_{\ell=1}^{K-1} \exp(\mathbf{w}_\ell \cdot \mathbf{x})}$$

- For class K, the conditional probability is

$$P(y = K|\mathbf{x}) \;=\; \frac{1}{1 + \sum_{\ell=1}^{K-1} \exp(\mathbf{w}_\ell \cdot \mathbf{x})}$$

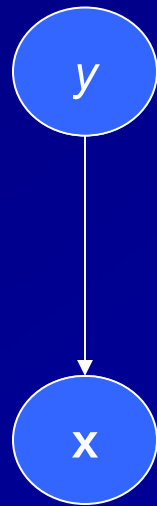
Summary of Logistic Regression

- Learns conditional probability distribution $P(y \mid \mathbf{x})$
- Local Search
  - begins with initial weight vector. Modifies it iteratively to maximize the log likelihood of the data
- Eager
  - the classifier is constructed from the training examples, which can then be discarded
- Online or Batch
  - both online and batch variants of the algorithm exist

Linear Discriminant Analysis

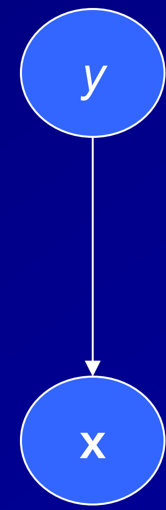
- Learn P(**x**,*y*). This is sometimes called the generative approach, because we can think of P(**x**,*y*) as a model of how the data is generated.
  - For example, if we factor the joint distribution into the form

    P(**x**,*y*) = P(*y*) P(**x** | *y*)
  - we can think of P(*y*) as "generating" a value for *y* according to P(*y*). Then we can think of P(**x** | *y*) as generating a value for **x** given the previously-generated value for *y*.
  - This can be described as a Bayesian network



Linear Discriminant Analysis (2)

- P(*y*) is a discrete multinomial distribution
  - example: P(*y* = 0) = 0.31, P(*y* = 1) = 0.69 will generate 31% negative examples and 69% positive examples
- For LDA, we assume that P(**x** | *y*) is a multivariate normal distribution with mean μ_k and covariance matrix Σ



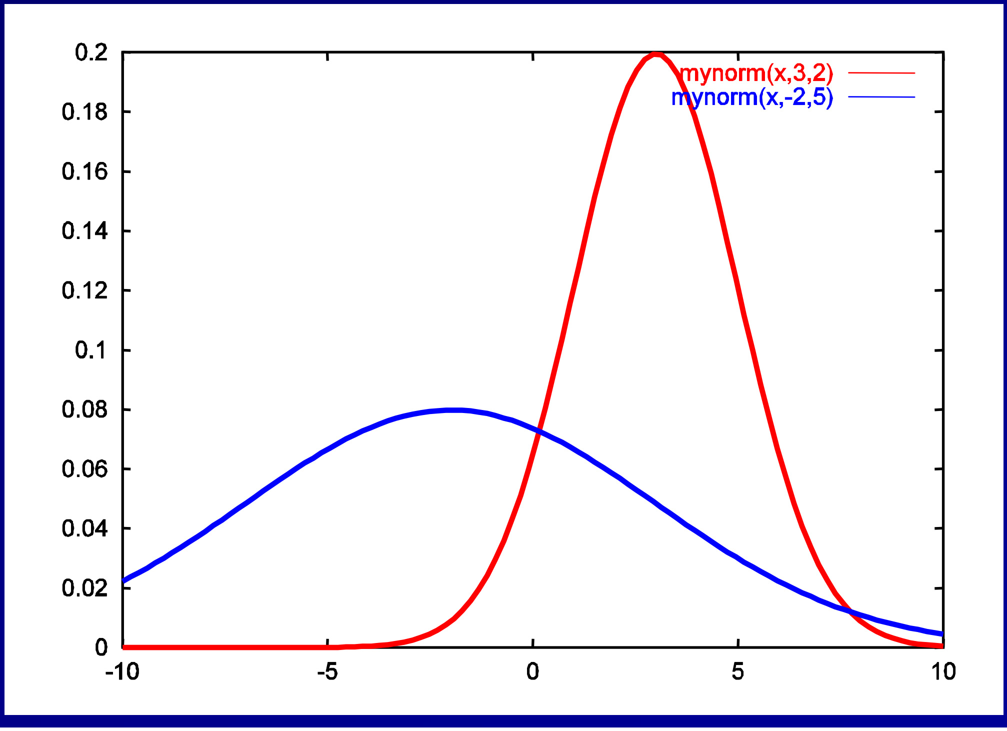
$$P(\mathbf{x}|y=k) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}[\mathbf{x}-\mu_k]^T \Sigma^{-1} [\mathbf{x}-\mu_k]\right)$$

Multivariate Normal Distributions: A tutorial

- Recall that the univariate normal (Gaussian) distribution has the formula

$$p(x) = \frac{1}{(2\pi)^{1/2}\sigma} \exp\left[-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right]$$

- where μ is the mean and σ^2 is the variance
- Graphically, it looks like this:



The Multivariate Gaussian

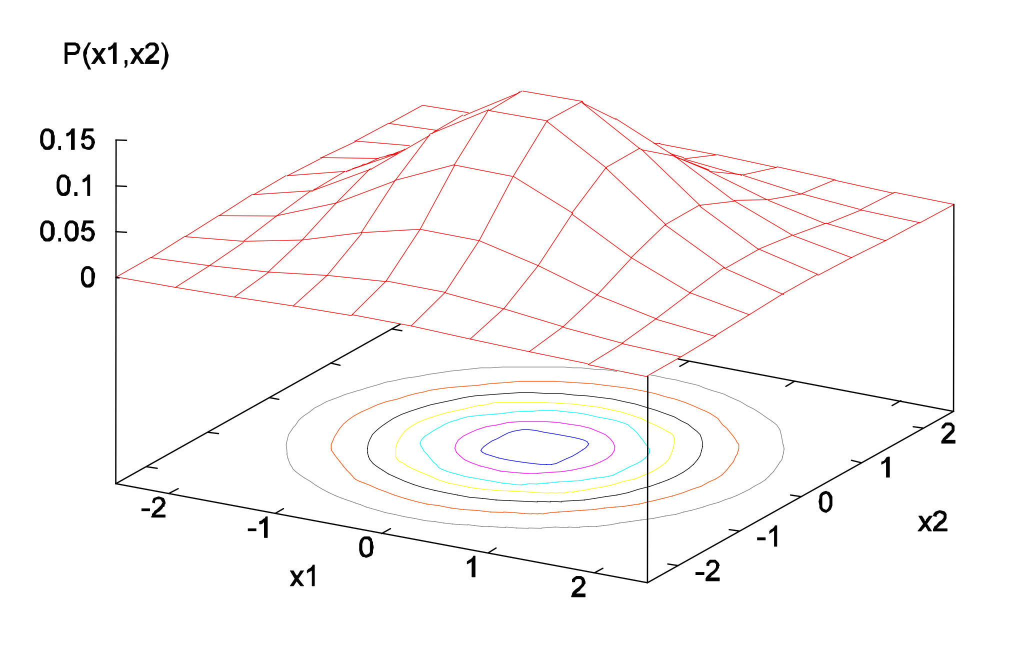
- A 2-dimensional Gaussian is defined by a mean vector $\mu = (\mu_1, \mu_2)$ and a covariance matrix

$$\Sigma = \begin{bmatrix} \sigma^2_{1,1} & \sigma^2_{1,2} \\ \sigma^2_{1,2} & \sigma^2_{2,2} \end{bmatrix}$$

- where $\sigma^2_{i,j} = E[(x_i - \mu_i)(x_j - \mu_j)]$ is the variance (if $i = j$) or co-variance (if $i \neq j$). Σ is symmetrical and positive-definite.

The Multivariate Gaussian (2)

- If Σ is the identity matrix $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\mu = (0, 0)$, we get the standard normal distribution:

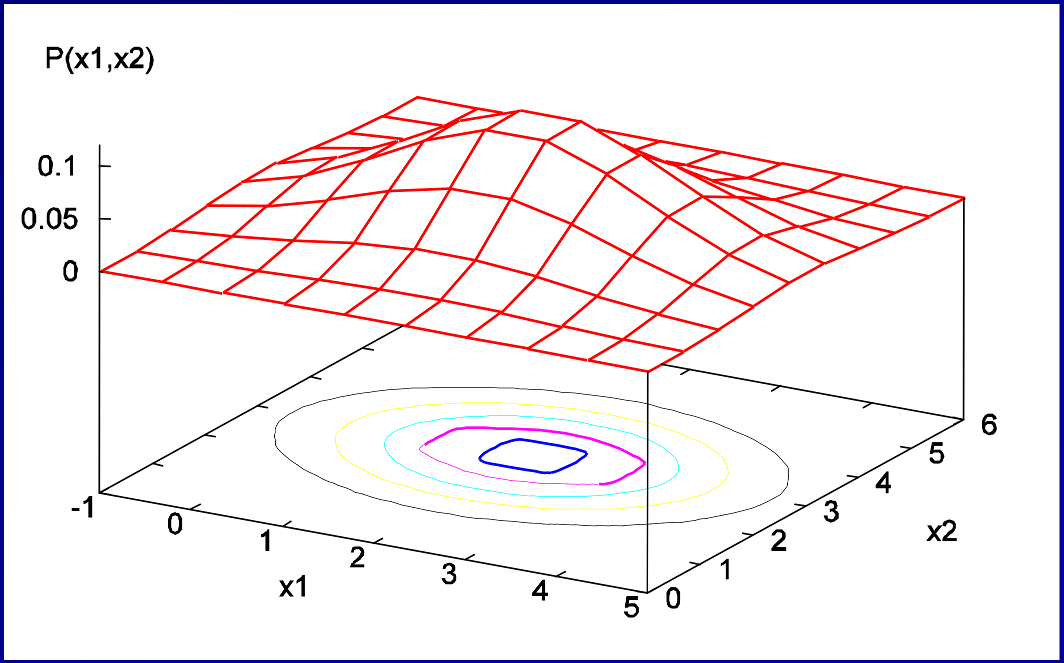


The Multivariate Gaussian (3)

- If Σ is a diagonal matrix, then x_1, and x_2 are independent random variables, and lines of equal probability are ellipses parallel to the coordinate axes. For example, when

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

$$\mu = (2, 3)$$ we obtain

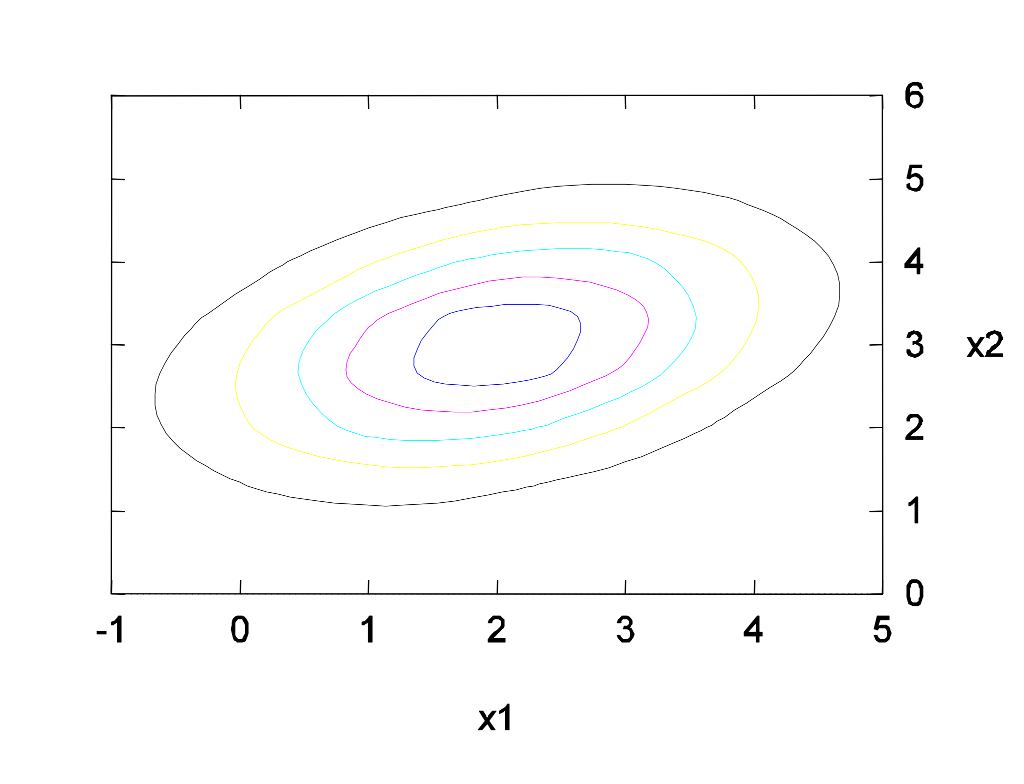


The Multivariate Gaussian (4)

- Finally, if Σ is an arbitrary matrix, then x_1 and x_2 are dependent, and lines of equal probability are ellipses tilted relative to the coordinate axes. For example, when

$$\Sigma = \begin{bmatrix} 2 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$ and

$$\mu = (2, 3)$$ we obtain



Estimating a Multivariate Gaussian

- Given a set of N data points {$\mathbf{x}_1$, …, $\mathbf{x}_N$}, we can compute the maximum likelihood estimate for the multivariate Gaussian distribution as follows:

$$\hat{\mu} = \frac{1}{N}\sum_i \mathbf{x}_i$$

$$\hat{\Sigma} = \frac{1}{N}\sum_i (\mathbf{x}_i - \hat{\mu}) \cdot (\mathbf{x}_i - \hat{\mu})^T$$

- Note that the dot product in the second equation is an outer product. The outer product of two vectors is a matrix:

$$\mathbf{x} \cdot \mathbf{y}^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \cdot [y_1 \; y_2 \; y_3] = \begin{bmatrix} x_1 y_1 & x_1 y_2 & x_1 y_3 \\ x_2 y_1 & x_2 y_2 & x_2 y_3 \\ x_3 y_1 & x_3 y_2 & x_3 y_3 \end{bmatrix}$$

- For comparison, the usual dot product is written as $\mathbf{x}^T \cdot \mathbf{y}$

The LDA Model

- Linear discriminant analysis assumes that the joint distribution has the form

$$P(\mathbf{x}, y) = P(y) \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \text{ exp} \left(-\frac{1}{2} [\mathbf{x} - \mu_y]^T \Sigma^{-1} [\mathbf{x} - \mu_y] \right)$$

where each μ_y is the mean of a multivariate Gaussian for examples belonging to class *y* and Σ is a single covariance matrix <u>shared by all classes</u>.

Fitting the LDA Model

- It is easy to learn the LDA model in a single pass through the data:
  - Let $\widehat{\pi}_k$ be our estimate of P(*y* = *k*)
  - Let N_k be the number of training examples belonging to class *k*.

$$\widehat{\pi}_k = \frac{N_k}{N}$$

$$\widehat{\mu_k} = \frac{1}{N_k} \sum_{\{i:y_i=k\}} \mathbf{x}_i$$

$$\hat{\Sigma} = \frac{1}{N} \sum_i (\mathbf{x}_i - \widehat{\mu}_{y_i}) \cdot (\mathbf{x}_i - \widehat{\mu}_{y_i})^T$$

- Note that each $\mathbf{x}_i$ is subtracted from its corresponding $\widehat{\mu}_{y_i}$ prior to taking the outer product. This gives us the "pooled" estimate of Σ

LDA learns an LTU

- Consider the 2-class case with a 0/1 loss function. Recall that

$$
\begin{aligned}
P(y=0|\mathbf{x}) &= \frac{P(\mathbf{x}, y=0)}{P(\mathbf{x}, y=0) + P(\mathbf{x}, y=1)} \\
P(y=1|\mathbf{x}) &= \frac{P(\mathbf{x}, y=1)}{P(\mathbf{x}, y=0) + P(\mathbf{x}, y=1)}
\end{aligned}
$$

- Also recall from our derivation of the Logistic Regression classifier that we should classify into class ŷ = 1 if

$$\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} > 0$$

- Hence, for LDA, we should classify into ŷ = 1 if

$$\log \frac{P(\mathbf{x}, y=1)}{P(\mathbf{x}, y=0)} > 0$$

because the denominators cancel

LDA learns an LTU (2)

$$P(\mathbf{x}, y) = P(y)\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}[\mathbf{x}-\mu_y]^T\Sigma^{-1}[\mathbf{x}-\mu_y]\right)$$

$$\frac{P(\mathbf{x}, y=1)}{P(\mathbf{x}, y=0)} = \frac{P(y=1)\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}[\mathbf{x}-\mu_1]^T\Sigma^{-1}[\mathbf{x}-\mu_1]\right)}{P(y=0)\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}[\mathbf{x}-\mu_0]^T\Sigma^{-1}[\mathbf{x}-\mu_0]\right)}$$

$$\frac{P(\mathbf{x}, y=1)}{P(\mathbf{x}, y=0)} = \frac{P(y=1)\ \exp\left(-\frac{1}{2}[\mathbf{x}-\mu_1]^T\Sigma^{-1}[\mathbf{x}-\mu_1]\right)}{P(y=0)\ \exp\left(-\frac{1}{2}[\mathbf{x}-\mu_0]^T\Sigma^{-1}[\mathbf{x}-\mu_0]\right)}$$

$$\log\frac{P(\mathbf{x}, y=1)}{P(\mathbf{x}, y=0)} = \log\frac{P(y=1)}{P(y=0)} - \frac{1}{2}\left([\mathbf{x}-\mu_1]^T\Sigma^{-1}[\mathbf{x}-\mu_1] - [\mathbf{x}-\mu_0]^T\Sigma^{-1}[\mathbf{x}-\mu_0]\right)$$

LDA learns an LTU (3)

- Let's focus on the term in brackets:

$$\left([\mathbf{x} - \mu_1]^T \Sigma^{-1} [\mathbf{x} - \mu_1] - [\mathbf{x} - \mu_0]^T \Sigma^{-1} [\mathbf{x} - \mu_0] \right)$$

- Expand the quadratic forms as follows:

$$[\mathbf{x} - \mu_1]^T \Sigma^{-1} [\mathbf{x} - \mu_1] = \mathbf{x}^T \Sigma^{-1} \mathbf{x} - \mathbf{x}^T \Sigma^{-1} \mu_1 - \mu_1^T \Sigma^{-1} \mathbf{x} + \mu_1^T \Sigma^{-1} \mu_1$$

$$[\mathbf{x} - \mu_0]^T \Sigma^{-1} [\mathbf{x} - \mu_0] = \mathbf{x}^T \Sigma^{-1} \mathbf{x} - \mathbf{x}^T \Sigma^{-1} \mu_0 - \mu_0^T \Sigma^{-1} \mathbf{x} + \mu_0^T \Sigma^{-1} \mu_0$$

- Subtract the lower from the upper line and collect similar terms. Note that the quadratic terms cancel! This leaves only terms linear in **x**.

$$\mathbf{x}^T \Sigma^{-1} (\mu_0 - \mu_1) + (\mu_0 - \mu_1) \Sigma^{-1} \mathbf{x} + \mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0$$

LDA learns an LTU (4)

$$\mathbf{x}^T\Sigma^{-1}(\mu_0-\mu_1)+(\mu_0-\mu_1)\Sigma^{-1}\mathbf{x}+\mu_1^T\Sigma^{-1}\mu_1-\mu_0^T\Sigma^{-1}\mu_0$$

- Note that since Σ^{-1} is symmetric $\mathbf{a}^T\Sigma^{-1}\mathbf{b} = \mathbf{b}^T\Sigma^{-1}\mathbf{a}$ for any two vectors **a** and **b**. Hence, the first two terms can be combined to give

$$2\mathbf{x}^T\Sigma^{-1}(\mu_0 - \mu_1) + \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0.$$

- Now plug this back in…

$$\log\frac{P(\mathbf{x}, y = 1)}{P(\mathbf{x}, y = 0)} = \log\frac{P(y = 1)}{P(y = 0)} - \frac{1}{2}\left[2\mathbf{x}^T\Sigma^{-1}(\mu_0 - \mu_1) + \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0\right]$$

$$\log\frac{P(\mathbf{x}, y = 1)}{P(\mathbf{x}, y = 0)} = \log\frac{P(y = 1)}{P(y = 0)} + \mathbf{x}^T\Sigma^{-1}(\mu_1 - \mu_0) - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0$$

LDA learns an LTU (5)

$$\log \frac{P(\mathbf{x}, y=1)}{P(\mathbf{x}, y=0)} = \log \frac{P(y=1)}{P(y=0)} + \mathbf{x}^T \Sigma^{-1}(\mu_1 - \mu_0) - \frac{1}{2}\mu_1^T \Sigma^{-1} \mu_1 + \frac{1}{2}\mu_0^T \Sigma^{-1} \mu_0$$

Let

$$\begin{aligned} \mathbf{w} &= \Sigma^{-1}(\mu_1 - \mu_0) \\ c &= \log \frac{P(y=1)}{P(y=0)} - \frac{1}{2}\mu_1^T \Sigma^{-1} \mu_1 + \frac{1}{2}\mu_0^T \Sigma^{-1} \mu_0 \end{aligned}$$

Then we will classify into class $\hat{y} = 1$ if

$$\mathbf{w} \cdot \mathbf{x} + c > 0.$$

This is an LTU.

Two Geometric Views of LDA View 1: Mahalanobis Distance

- The quantity $D_M(\mathbf{x},\mathbf{u})^2 = (\mathbf{x}-\mathbf{u})^T\Sigma^{-1}(\mathbf{x}-\mathbf{u})$ is known as the (squared) Mahalanobis distance between **x** and **u**. We can think of the matrix Σ^{-1} as a linear distortion of the coordinate system that converts the standard Euclidean distance into the Mahalanobis distance
- Note that

$$\log P(\mathbf{x}|y=k) \;\propto\; \log \pi_k - \frac{1}{2}[(\mathbf{x}-\mu_k)^T\Sigma^{-1}(\mathbf{x}-\mu_k)]$$

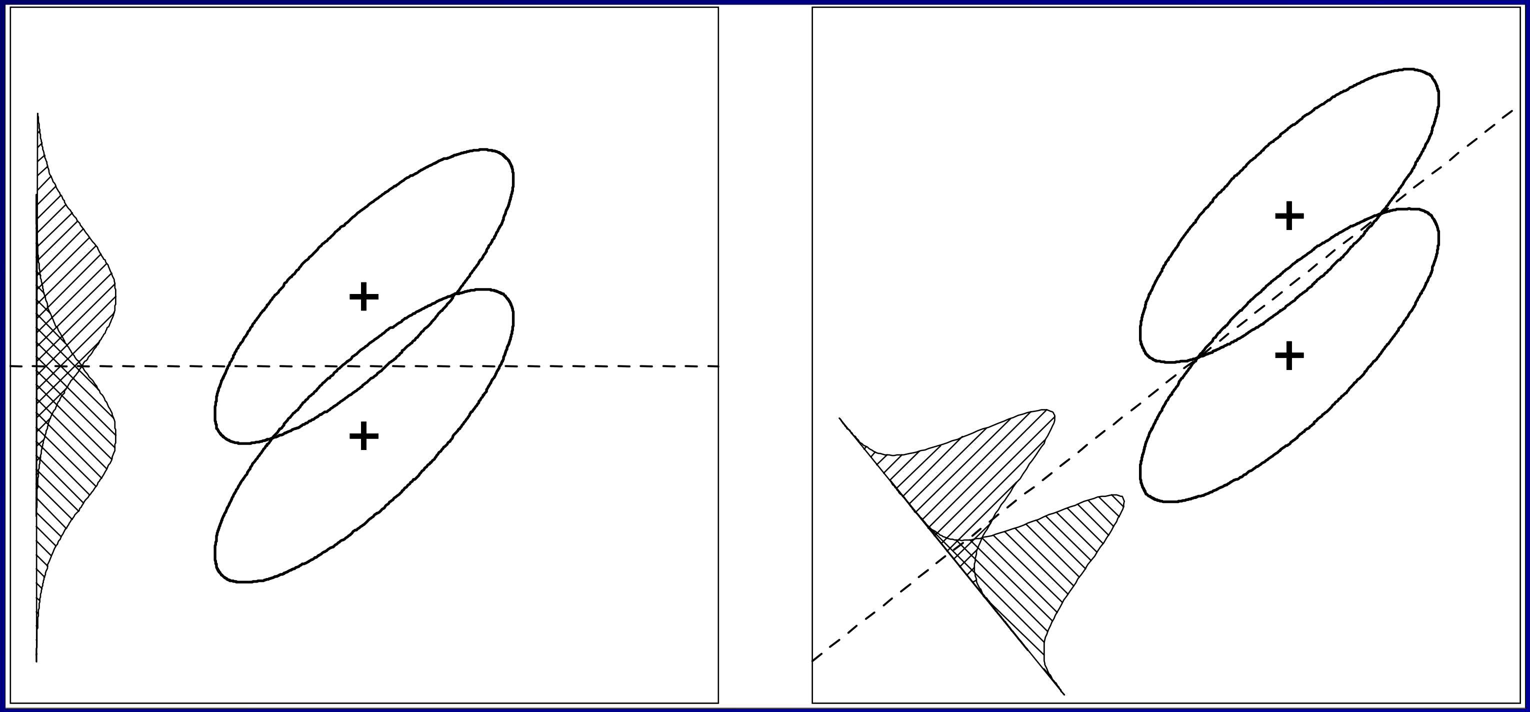
$$\log P(\mathbf{x}|y=k) \;\propto\; \log \pi_k - \frac{1}{2}D_M(\mathbf{x},\mu_k)^2$$

- Therefore, we can view LDA as computing
  - $D_M(\mathbf{x},\mu_0)^2$ and $D_M(\mathbf{x},\mu_1)^2$

  and then classifying **x** according to which mean μ_0 or μ_1 is closest in Mahalanobis distance (corrected by log π_k)

View 2: Most Informative Low-Dimensional Projection

- LDA can also be viewed as finding a hyperplane of dimension K – 1 such that **x** and the $\{\mu_k\}$ are projected down into this hyperplane and then **x** is classified to the nearest μ_k using Euclidean distance inside this hyperplane



Generalizations of LDA

- General Gaussian Classifier
  - Instead of assuming that all classes share the same Σ, we can allow each class *k* to have its own Σ_k. In this case, the resulting classifier will be a quadratic threshold unit (instead of an LTU)
- Naïve Gaussian Classifier
  - Allow each class to have its own Σ_k, but require that each Σ_k be diagonal. This means that *within* each class, any pair of features x_{j1} and x_{j2} will be assumed to be statistically independent. The resulting classifier is still a quadratic threshold unit (but with a restricted form)

Summary of Linear Discriminant Analysis

- Learns the joint probability distribution P(**x**, *y*).
- Direct Computation. The maximum likelihood estimate of P(**x**,*y*) can be computed from the data without search. However, inverting the Σ matrix requires $O(n^3)$ time.
- Eager. The classifier is constructed from the training examples. The examples can then be discarded.
- Batch. Only a batch algorithm is available. An online algorithm could be constructed if there is an online algorithm for incrementally updated Σ^{-1}. [This is easy for the case where Σ is diagonal.]

Comparing Perceptron, Logistic Regression, and LDA

- How should we choose among these three algorithms?
- There is a big debate within the machine learning community!

Issues in the Debate

- Statistical Efficiency. If the generative model P(**x**,*y*) is correct, then LDA usually gives the highest accuracy, particularly when the amount of training data is small. If the model is correct, LDA requires 30% less data than Logistic Regression in theory
- Computational Efficiency. Generative models typically are the easiest to learn. In our example, LDA can be computed directly from the data without using gradient descent.

Issues in the Debate

- Robustness to changing loss functions. Both generative and conditional probability models allow the loss function to be changed at run time without re-learning. Perceptron requires re-training the classifier when the loss function changes.
- Robustness to model assumptions. The generative model usually performs poorly when the assumptions are violated. For example, if $P(\mathbf{x} \mid y)$ is very non-Gaussian, then LDA won't work well. Logistic Regression is more robust to model assumptions, and Perceptron is even more robust.
- Robustness to missing values and noise. In many applications, some of the features x_{ij} may be missing or corrupted in some of the training examples. Generative models typically provide better ways of handling this than non-generative models.