

Midterm Exam Review

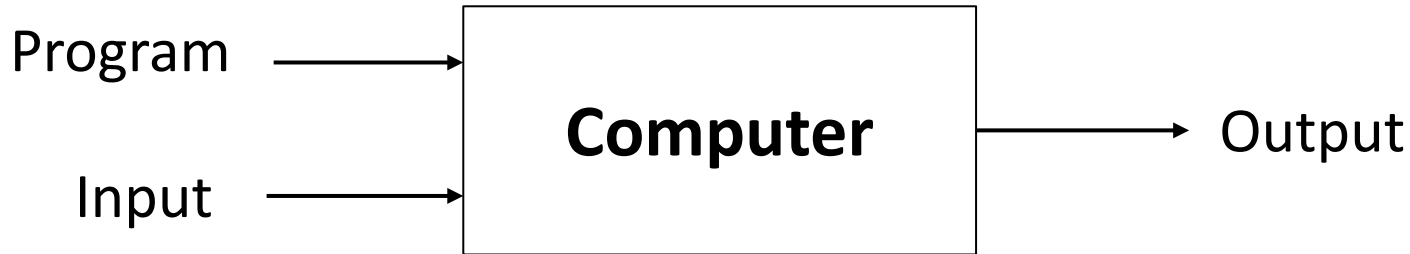
Janardhan Rao (Jana) Doppa

School of EECS, Washington State University

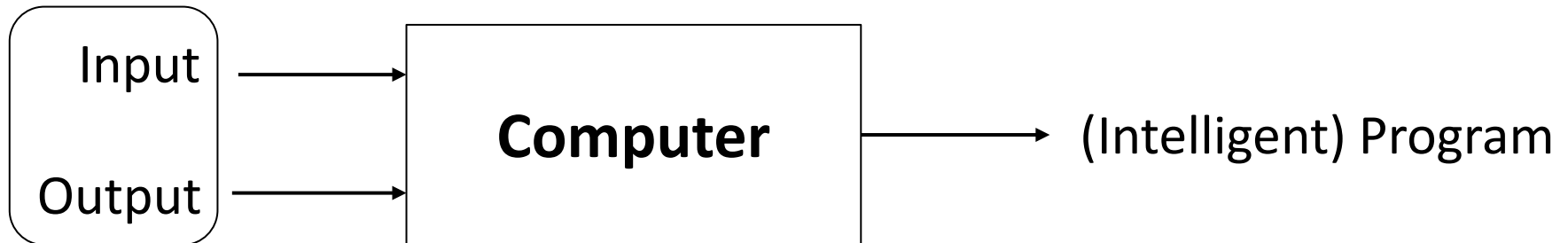
What is Machine Learning?

- Machine learning = Automating Automation

Traditional Programming



Machine Learning

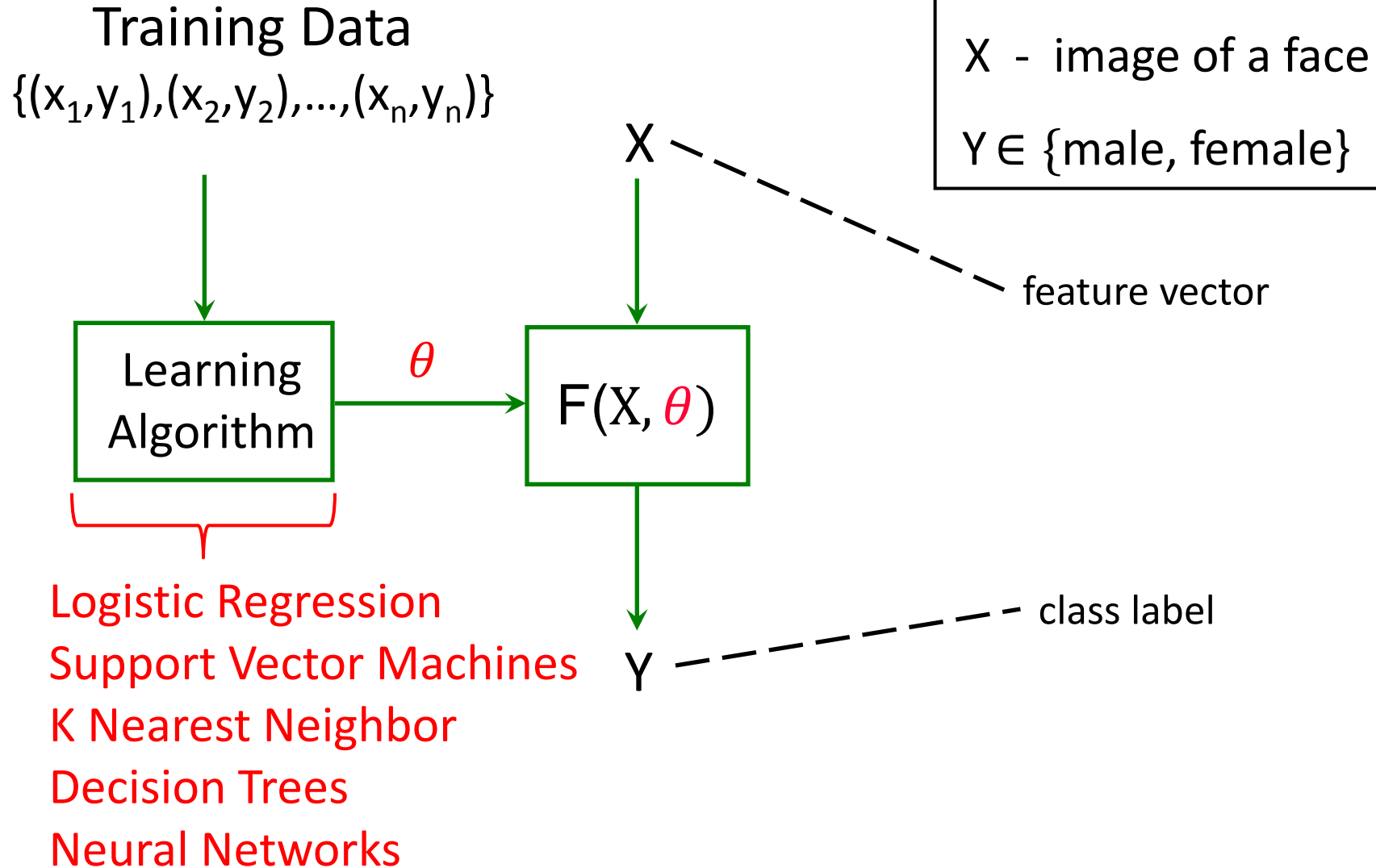


Training data

Learning Paradigms

- **Supervised Learning** – main focus of this course
- **Semi-Supervised Learning**
- **Unsupervised Learning**
- **Active Learning**
- **Reinforcement Learning**

Supervised Learning for Simple Outputs



Overview of ML Algorithms

- There are lot of machine learning algorithms
- Every machine learning algorithm has three components
 - ▲ **Representation**
 - ▲ **Evaluation**
 - ▲ **Optimization**

Representation: Examples

- Linear hyper-planes
- Decision trees
- Sets of conjunctive / logical rules
- Graphical models (Bayes/Markov nets)
- Neural Networks
- ...

Evaluation: Examples

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Cost / Utility
- Margin
- Entropy
- ...

Optimization: Examples

- **Combinatorial Optimization**
 - ▲ greedy search, dynamic programming
- **Convex Optimization**
 - ▲ gradient descent, co-ordinate descent
- **Constrained Optimization**
 - ▲ linear programming, quadratic programming
- ...

Machine Learned Programs: Errors

- **Bayes Error**
 - ▲ Error of the best possible classifier
- **Approximation Error**
 - ▲ Error due to restricted hypothesis class (representation)
- **Estimation Error**
 - ▲ Error due to finite training samples
- **Optimization Error**
 - ▲ Error due to not finding a global optimum to the optimization problem

Generative vs. Discriminative Learning

- Training data: $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ drawn from a joint distribution $P(X, Y)$
- **Generative learning:** learn the distribution $P(X, Y)$
 - ▲ “what controls the rise and fall of the stock prices?”
- **Discriminative learning:** learn the **conditional** distribution $P(Y|X)$
 - ▲ “will there be a rise in the stock prices today evening?”

$$P(Y|X) = \frac{P(X, Y)}{P(X)}$$

Parametric vs. Non-Parametric Learning

- **Parametric learning**

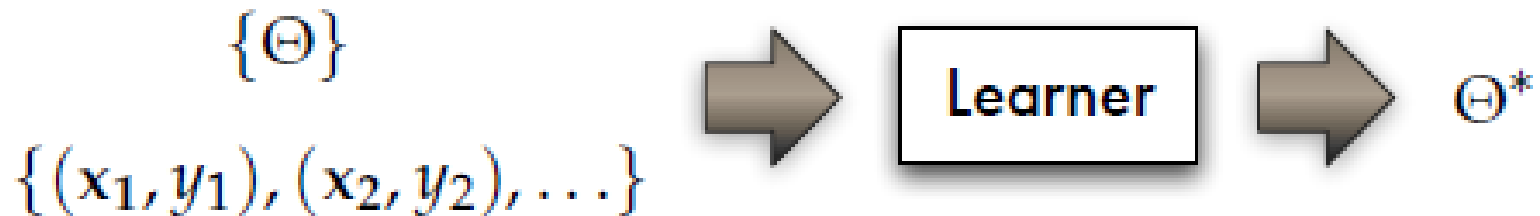
- ▶ define a space of models parameterized by a fixed number of parameters
- ▶ find model that best fits the data (by searching over the parameters)
- ▶ Example: logistic regression

- **Non-Parametric learning**

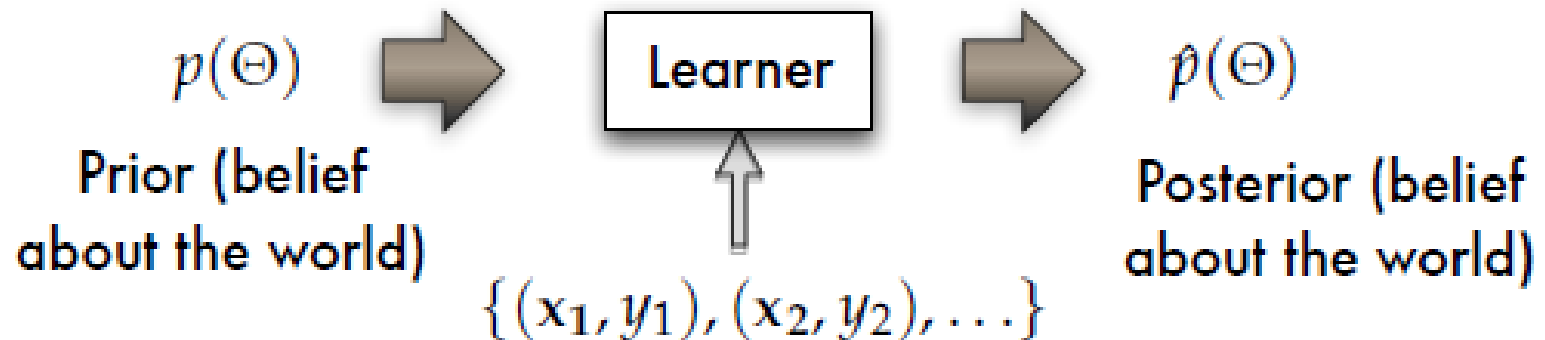
- ▶ define a space of models that can grow in size with data
- ▶ find model that best fits the data
- ▶ “Non-parametric” means “Not-fixed”
- ▶ Example: decision trees

Non-Bayesian vs. Bayesian Learning

- Non-Bayesian learning



- Bayesian learning



Θ^* is a point estimate.

$\hat{p}(\Theta)$ is a distribution over possible worlds

Recap of Last Lecture

- **Online Learning Framework: Overview**
 - ▲ A iterative game between the teacher and the student (online learner)
 - ▲ The goal of the student is to minimize the no. of mistakes made
- **Design Principles of Online Learning Algorithms**
 - ▲ Trade-off corrective (reduce loss) and conservative (reduce change in weights)
- **The Perceptron and PA algorithms**
 - ▲ simple additive weight updates

Online Learning Framework

- Initialize Classifier $f_1(\mathbf{x})$
- Algorithm works in rounds $t = 1 \dots T \dots$
- On round t the online algorithm :
 - Receives an input instance \mathbf{x}_t
 - Outputs a prediction $f_t(\mathbf{x}_t) = \hat{y}_t$
 - Receives a feedback label y_t
 - Computes loss $\ell(\hat{y}_t, y_t)$
 - Updates the prediction rule $f_t \rightarrow f_{t+1}$
- Goal :
 - Suffer small cumulative loss $\sum_t \ell(\hat{y}_t, y_t)$

Update Rules

- Online algorithms are based on an update rule which defines f_{t+1} from f_t (and possibly other information)
- **Linear Classifiers** : find \mathbf{w}_{t+1} from \mathbf{w}_t based on the input (\mathbf{x}_t, y_t)
- **Some Update Rules** :
 - Perceptron (Rosenblatt)
 - ALMA (Gentile)
 - ROMMA (Li & Long)
 - NORMA (Kivinen et. al)
 - MIRA (Crammer & Singer)
 - EG (Littlestone and Warmuth)
 - Bregman Based (Warmuth)
 - CWL (Dredze et. al)

Design Principles of Algorithms

- If the learner suffers non-zero loss at any round, then we want to balance two goals:
 - **Corrective:** Change weights so that we **don't make this error again**
 - **Conservative:** **Don't change the weights too much**

$$d(w_{t+1}, w_t) + \eta L(y_t, w_{tt})$$

The Perceptron Algorithm

- If the learner suffers non-zero loss at any round, then we want to balance two goals:

$$d(w_{t+1}, w_t) + \eta L(y_t, w_{tt})$$

- **Euclidean distance**
- **Hinge loss**
- **Stochastic Gradient Descent (SGD)**

The Perceptron Algorithm ($\eta = 1$)

- If No-Mistake $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) > 0$

- Do nothing $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$

- If Mistake $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) \leq 0$

- Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$

Voted and Averaged Perceptron

- **Voted Perceptron**

- ▲ Maintain multiple classifiers and take a weighted-majority vote to make predictions

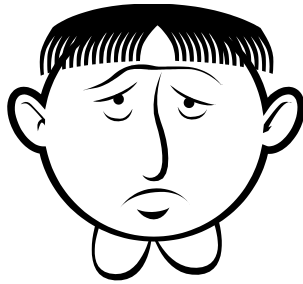
- **Averaged Perceptron**

- ▲ Average of all the different weight-vectors seen during training
- ▲ Can be seen as one form of regularization

PA Algorithm: Motivation

- **Perceptron:** No guaranties of margin *after* the update
- **PA:** Enforce a minimal non-zero margin after the update
- In particular :
 - ▲ If the margin is large enough (1), then do nothing
 - ▲ If the margin is less then unit, update such that the margin *after* the update is *enforced* to be unit

Passive-Aggressive Update



$$\mathbf{w}_{t+1} = \mathbf{w}_t + \tau y_t \mathbf{x}_t$$

$$y_t(\mathbf{w}_t \cdot \mathbf{x}_t) \geq 1$$

$$y_t(\mathbf{w}_t \cdot \mathbf{x}_t) < 1$$

$$\tau = 0$$

$$\tau = \frac{1 - y_t(\mathbf{w}_t \cdot \mathbf{x}_t)}{\|\mathbf{x}_t\|^2}$$

Input Space vs. Version Space

- **Input Space :**

- ▶ Points are input data $y_t \mathbf{X}_t$
- ▶ One constraint is induced by weight vector \mathbf{W}
- ▶ Primal space
- ▶ Half space = all input examples that are classified correctly by a given predictor (weight vector)

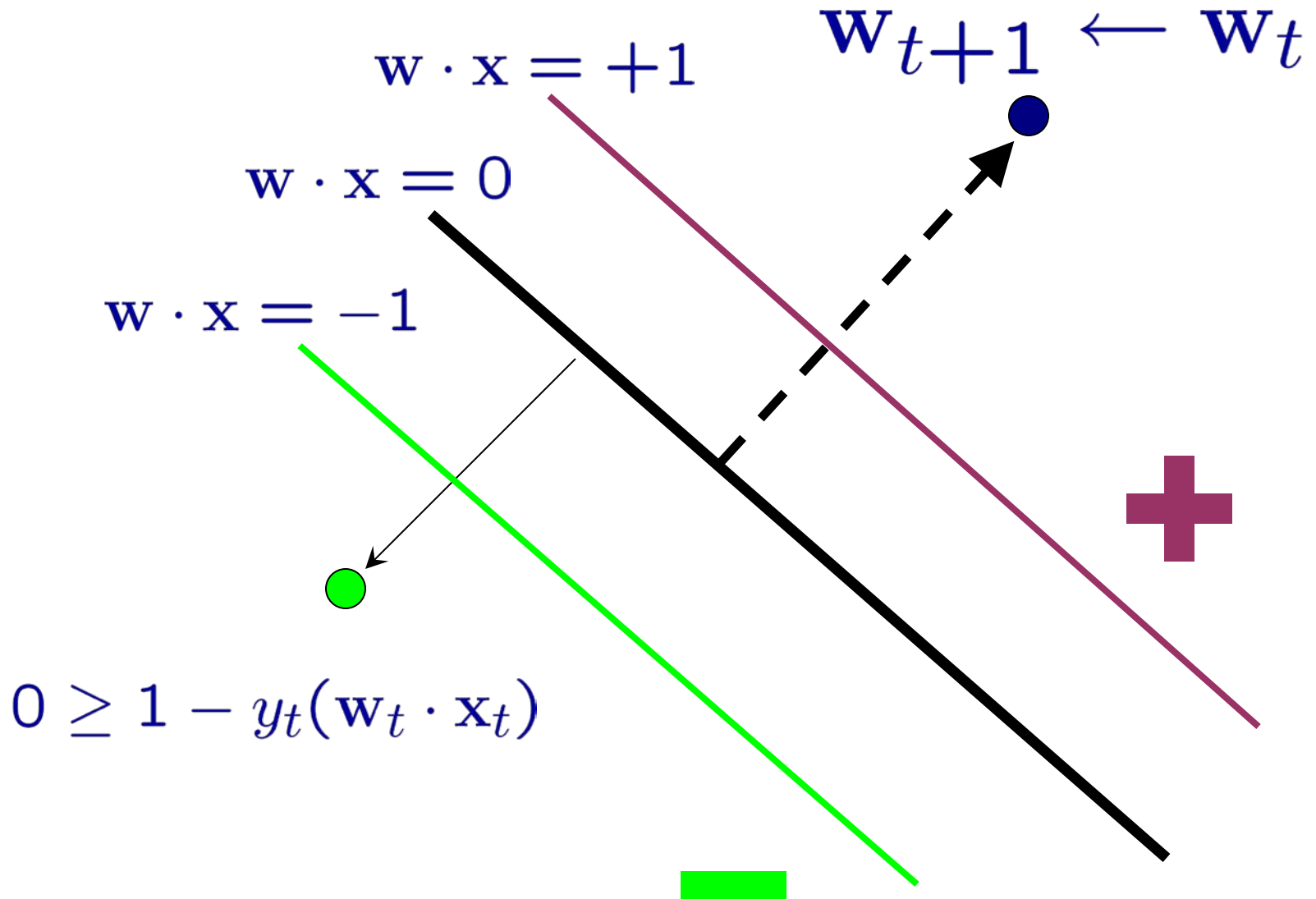
$$\{y\mathbf{x} : \mathbf{w} \cdot (y\mathbf{x}) \geq 0\}$$

- **Version Space :**

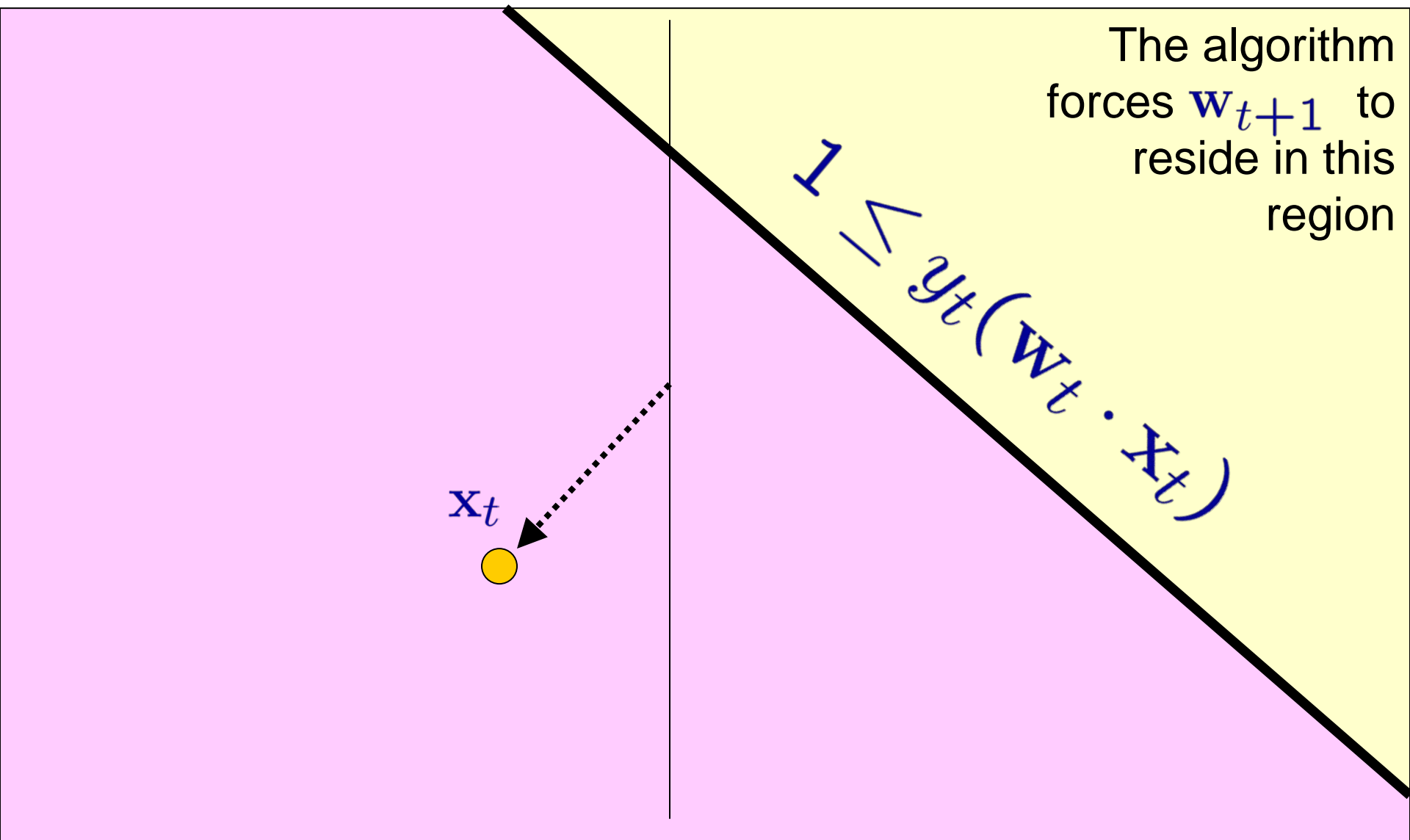
- ▶ Points are weight vectors \mathbf{W}
- ▶ One constraints is induced by input data $y_t \mathbf{X}_t$
- ▶ Dual space
- ▶ Half space = all predictors (weight vectors) that classify correctly a given input example

$$\{\mathbf{w} : \mathbf{w} \cdot (y\mathbf{x}) \geq 0\}$$

Input Space

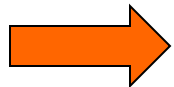


Weight vector (Version) Space

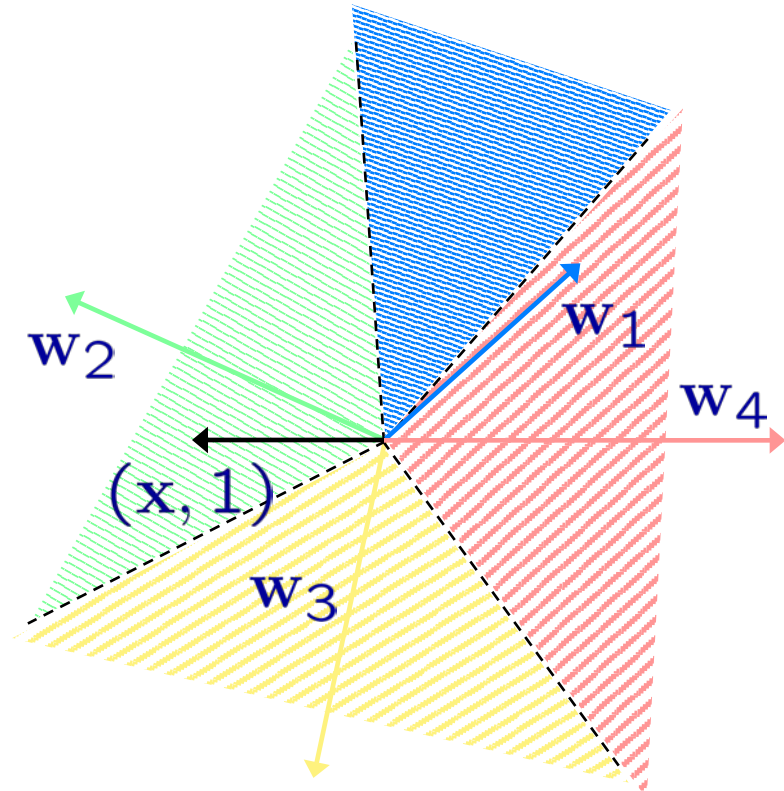


Multi-Class: Representation-I

- k Prototypes $w_1, w_2 \dots w_k$
- New instance x
- Compute $\text{Score}(r) = w_r \cdot x$



Class r	$w_r \cdot x$
1	-1.08
2	1.66
3	0.37
4	-2.09



- Prediction:
The class achieving the highest Score

Multi-Class Representation-II

- Weight-vector per class (Representation I)
 - ▲ Intuitive
- Single weight-vector (Representation II)
 - ▲ Generalizes representation I

$$F(x,4) = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & x & 0 \\ \hline \end{array}$$

- Predict label with highest score (Inference)

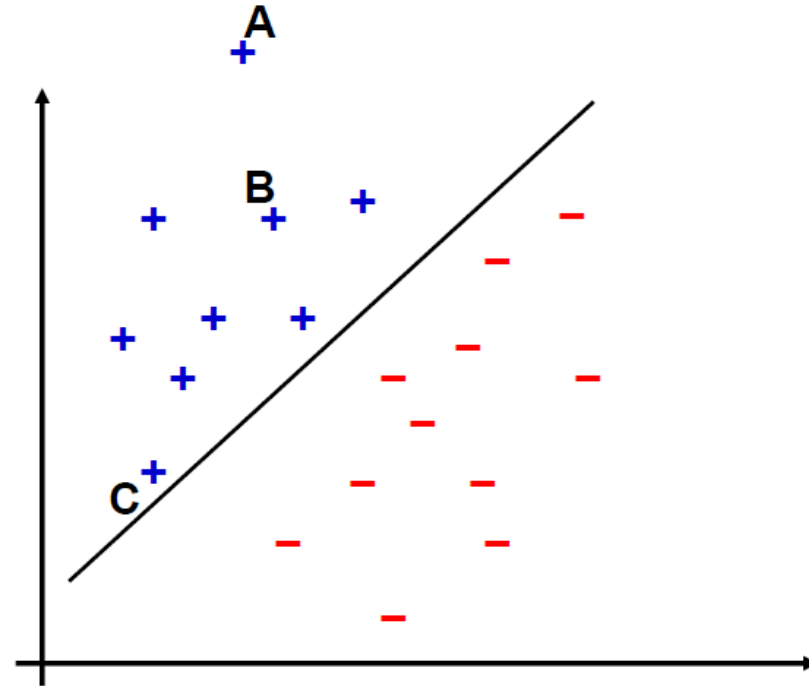
$$\arg \max_z F(x, z) \cdot w$$

Summary of Online Learning

- **Online learning**
 - ▲ Iterative game between teacher and learner
- **Design principles of online learning**
 - ▲ Trade-off amount of change (conservative) and reduction in loss (corrective)
- **Online learning algorithms**
 - ▲ Perceptron (fixed learning rate for all examples)
 - ▲ Passive-Aggressive (fixed learning rate for each example)
 - ▲ Confidence-weighted classifier (fixed learning for each feature and each example)

Intuition of Margin

- Consider points A, B, and C
- We are quite confident in our prediction for A because it is far from the decision boundary
- In contrast, we are not so confident in our prediction for C because a slight change in the decision boundary may flip the decision



Given a training set, we would like to make all predictions correct and confident! This leads to the concept of margin.

Maximum Margin Classifier: Formulation

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to: } & y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1, \quad i = 1, \dots, N \end{aligned}$$

- We will see that it is useful to first formulate an equivalent dual optimization problem and solve it instead
 - ▲ This requires a bit of machinery!

Dual Problem

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to $\alpha_i \geq 0$, $\sum_{i=1}^n \alpha_i y_i = 0$

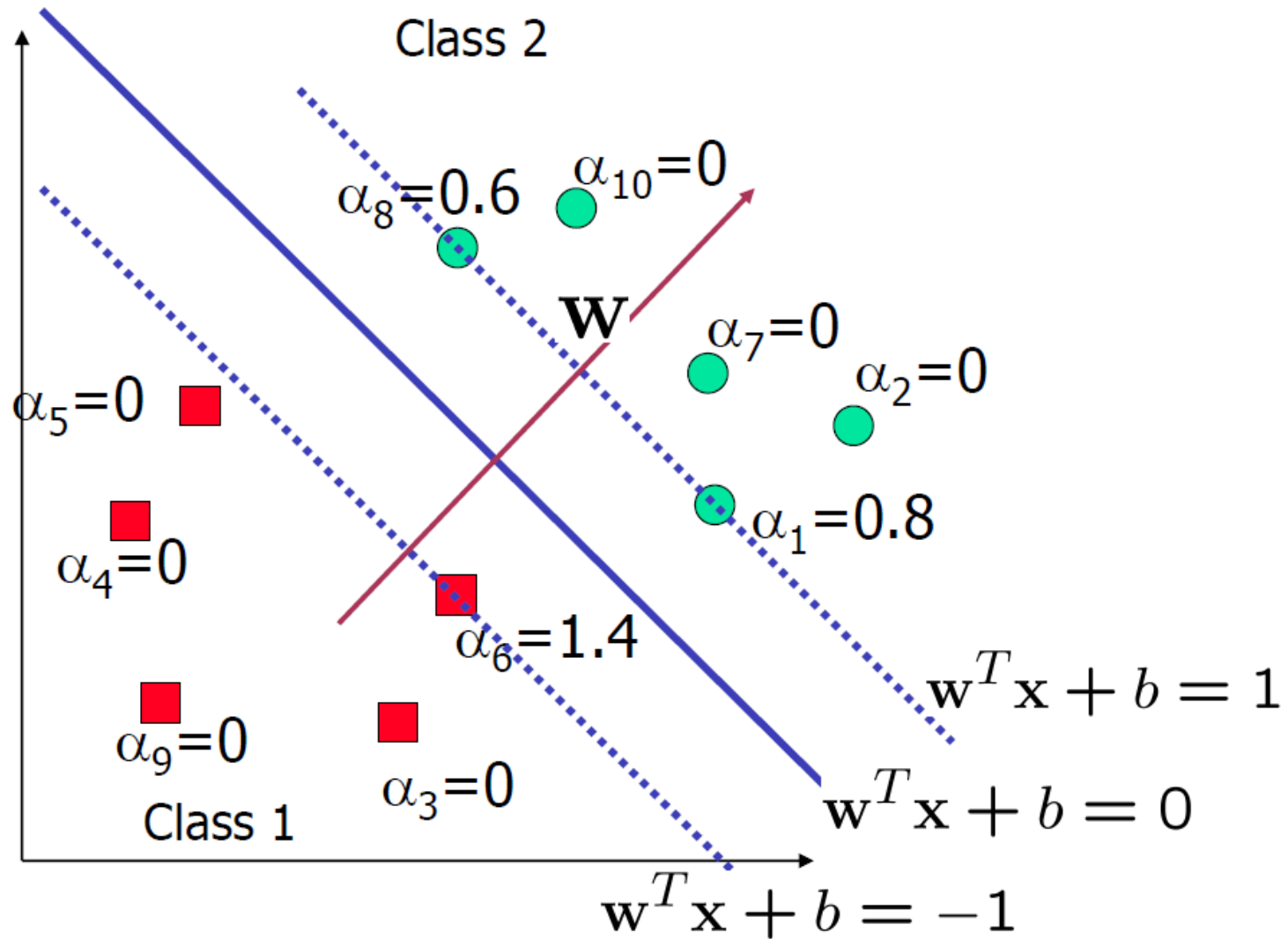
- This is also a QP problem
 - ▲ A global maxima of α' s can always be found
- Weights w can be recovered by
$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$
- b can also be recovered (we will skip the details)

Characteristics of Solution

- Many of the α_i 's are zero
 - ▲ Weights \mathbf{w} is a linear combination of small number of data points
- \mathbf{x}_i with non-zero α_i are called support vectors (SVs)
 - ▲ The decision boundary is determined only by the SVs
 - ▲ Let $t_j (j = 1, 2, \dots, s)$ be the indices of the s support vectors. We can write

$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

A Geometric Interpretation



Characteristics of Solution

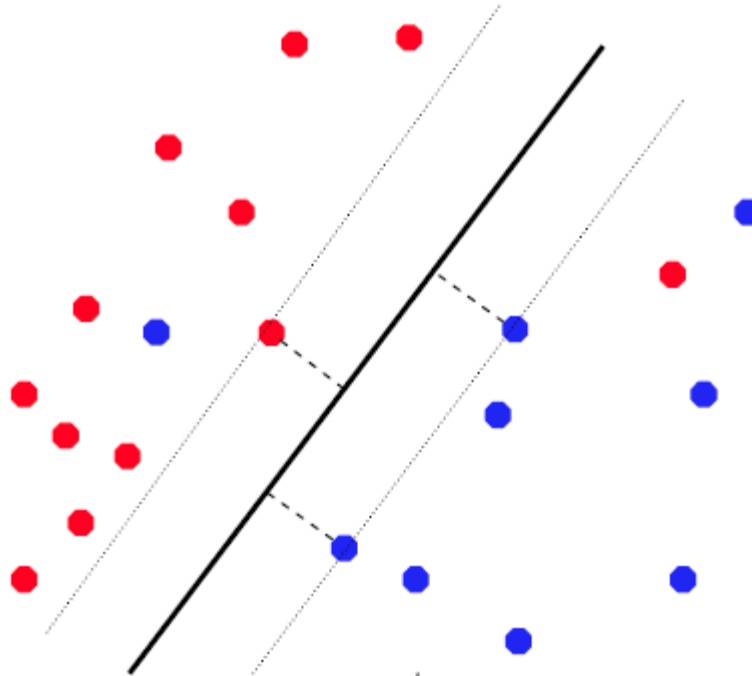
- For classifying a new input example \mathbf{z} , compute

$$\mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$$

- ▲ Classify \mathbf{z} as positive if the sum is positive, and negative otherwise
- ▲ **Note:** \mathbf{w} need not be formed explicitly, rather we can classify \mathbf{z} by taking inner products with the support vectors (useful when we generalize the notion of inner product later)

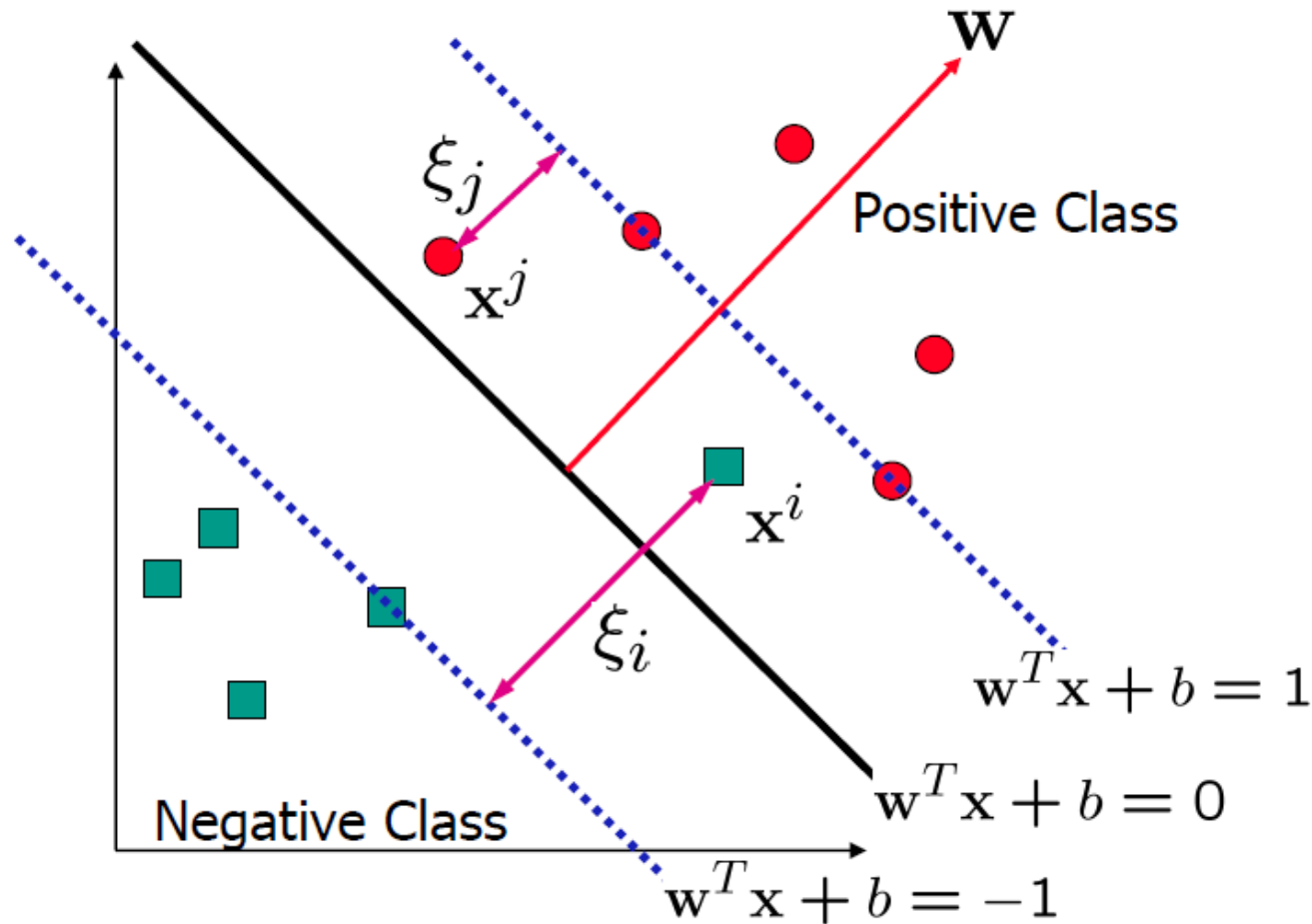
Non-Separable data and Noise

- What if the data is not linearly separable?
- We may have noise in data, and maximum margin classifier is not robust to noise!



Hard Margin \rightarrow Soft Margin

- Allow functional margins to be less than 1
 - ▲ But we will charge a penalty



Soft Margin Maximization

Hard margin

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

subject to: $y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1, \quad i = 1, \dots, N$



Soft margin

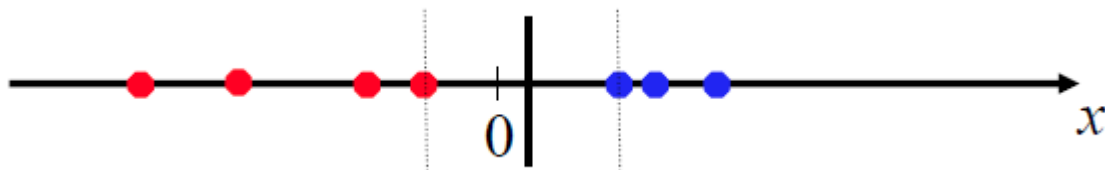
$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + c \sum_{i=1}^N \xi_i$$

subject to: $y^i (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1 - \xi_i, \quad i = 1, \dots, N$
 $\xi_i \geq 0, \quad i = 1, \dots, N$

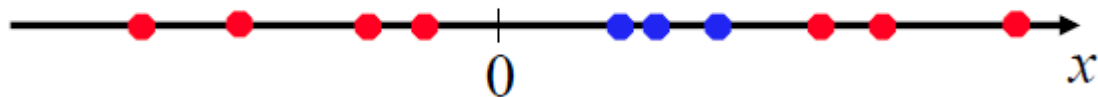
- Introduce **slack variables** ξ_i to allow functional margins to be smaller than 1

Non-Linear SVMs

- Datasets that are linearly separable with some noise work out great

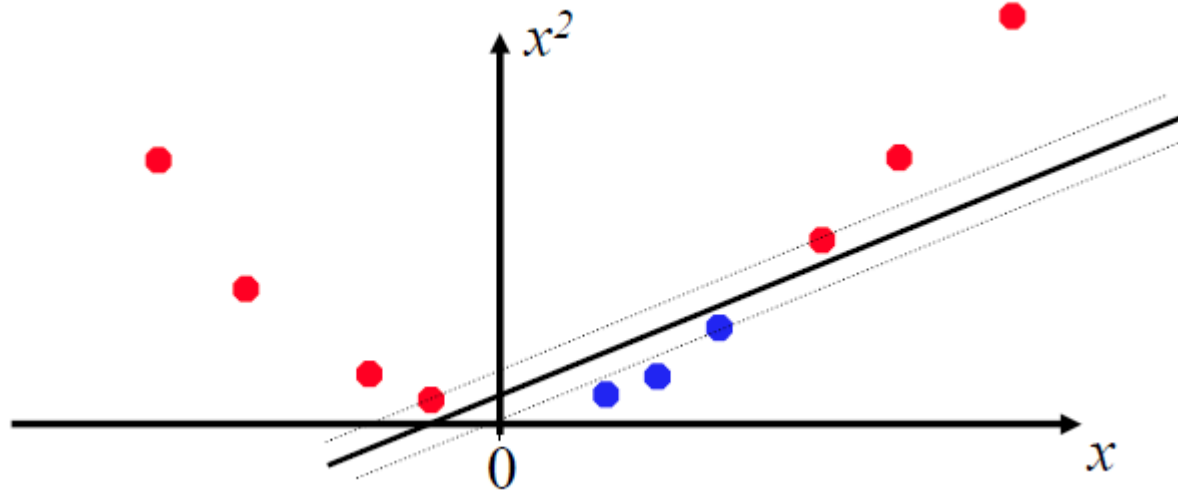


- But what are we going to do if the dataset is just too hard?



Non-Linear SVMs

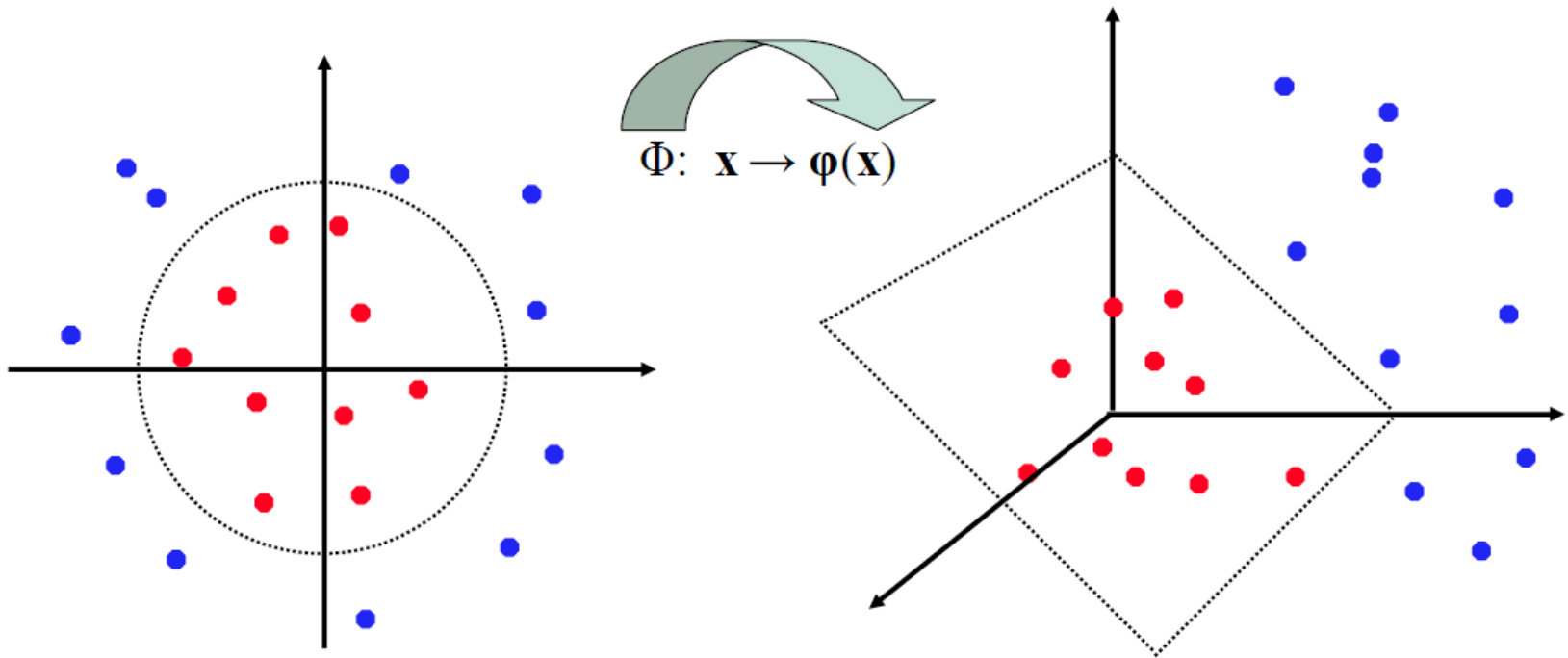
- How about mapping the data to a higher dimensional space?



Non-Linear SVMs: Feature Spaces

- **General idea**

- ▲ For any data set, the original feature space can always be mapped to some higher-dimensional feature space such that the data is linearly separable



Kernel Functions

- The linear classifier relies on inner product between vectors: $K(x_i, x_j) = \langle x_i \cdot x_j \rangle$
- If every data point is mapped into high-dimensional space via some transformation $\Phi: x \rightarrow \phi(x)$, the inner product becomes $K(x_i, x_j) = \langle \phi(x_i) \cdot \phi(x_j) \rangle$
- A **kernel function** is a function that is equivalent to an inner product in some feature space
 - ▲ Example: $K(x_i, x_j) = (x_i \cdot x_j + 1)^2$
 - ▲ This is equivalent to mapping to the quadratic space!

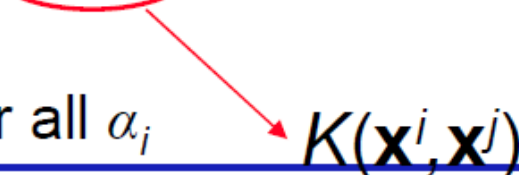
Quadratic Kernel

- A kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\phi(x)$ explicitly)
- Computing inner product of quadratic features is $O(m^2)$ time vs. $O(m)$ time for kernel

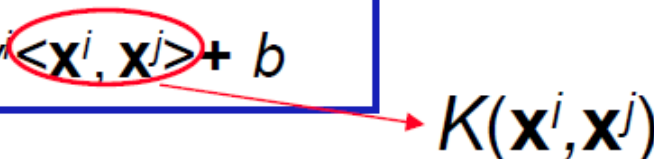
Non-Linear SVMs

- Dual problem formulation

Find $\alpha_1 \dots \alpha_N$ such that
 $\sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y^i y^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle$ is maximized and
(1) $\sum \alpha_i y^i = 0$
(2) $0 \leq \alpha_i \leq c$ for all α_i



- To classify a new point, we compute

$$f(\mathbf{x}) = \sum \alpha_i y^i \langle \mathbf{x}^i, \mathbf{x}^j \rangle + b$$


- Optimization techniques to find α_i 's remain the same
- This shows the utility of the dual formulation

Kernelized Perceptron

Primal Form

update **weights**

$$w \leftarrow w + y_i \phi(x_i)$$

classify

$$f(k) = w \cdot \phi(x)$$

Dual Form

update **linear coefficients**

$$\alpha_i \leftarrow \alpha_i + y_i$$

implicitly equivalent to:

$$w = \sum_{i \in I} \alpha_i \phi(x_i)$$



- Nothing happens if classified correctly
- Weight vector is a linear combination $w = \sum_{i \in I} \alpha_i \phi(x_i)$
- Classifier is a linear combination of inner products

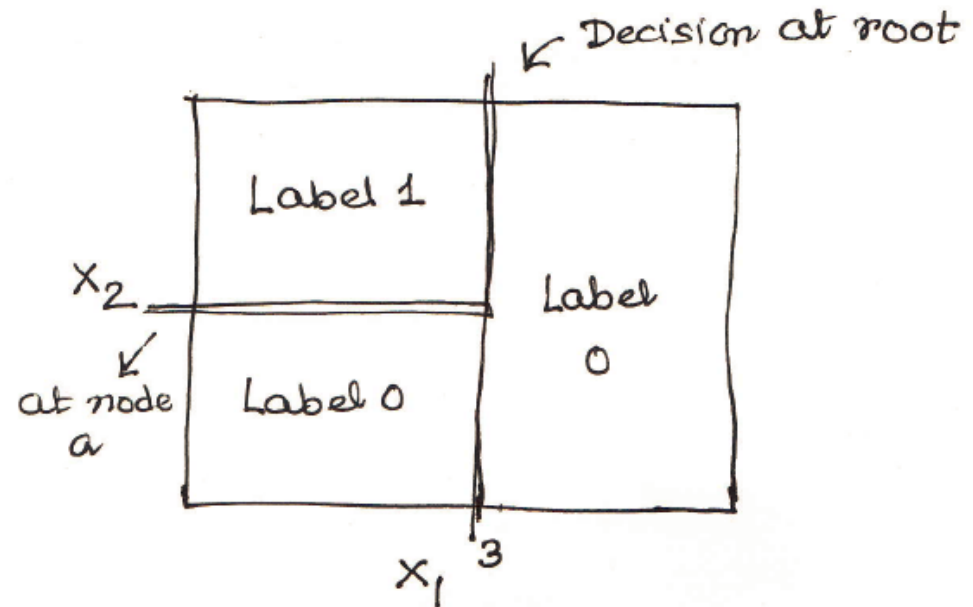
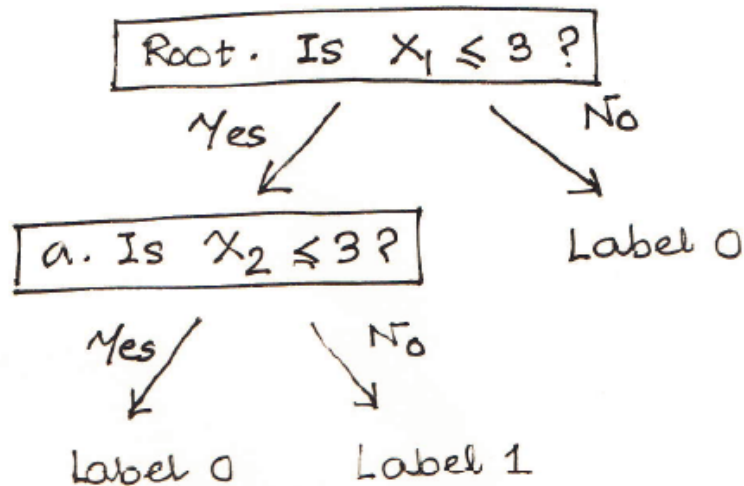
$$f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle = \sum_{i \in I} \alpha_i k(x_i, x)$$

K-Nearest Neighbor Classifier

- K-NN is a non-parametric classifier
 - ▶ no training needed! Just store all the training data 😊
 - ▶ simple and easy to implement
 - ▶ flexible hypothesis space: infinitely complex
- Disadvantages
 - ▶ Classification time is high (NN computation!)
 - ▶ Space requirement is high
 - ▶ Doesn't work very well in high dimensions

Decision Tree Classifier: Examples

Example 1: Features x_1, x_2



ID3 Decision Tree Learning Algorithm

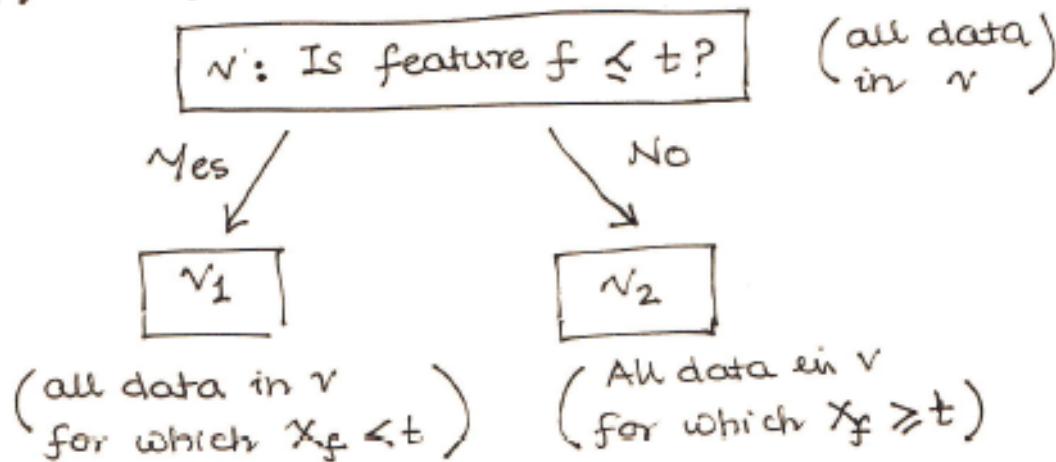
1. Initially, Whole training data is at root.

2. While there is an impure node:

(a) Pick any impure node v

(b) Pick a feature f and threshold t along which to "split" the data at v . Done according to "Splitting Rule" (to be described later)

(c) Modify tree as:



If any of v_1 or v_2 is pure (i.e. has data of only one label), then ~~predict~~ make it a leaf that predicts this label.

Impure Node: Node with data points with multiple labels

Each node in the algorithm corresponds to a subset of the training data.

Splitting Rule via Information Gain

$$\text{Information Gain}(Z) = H(X) - H(X|Z)$$

(essentially, how much entropy of X is reduced because we know Z).

It can be shown that information gain (IG) is ≥ 0 (always)

What is Over-fitting?

Data comes from some underlying true distribution \mathcal{D} .

Training data, test data \rightarrow all samples from \mathcal{D} .

Training error of a classifier:

$$\hat{err}(h) = \frac{1}{n} \sum_{i=1}^n 1(h(x_i) \neq y_i)$$

$1(\cdot)$ means an indicator variable which is 1 when the condition is true, 0 o/w.

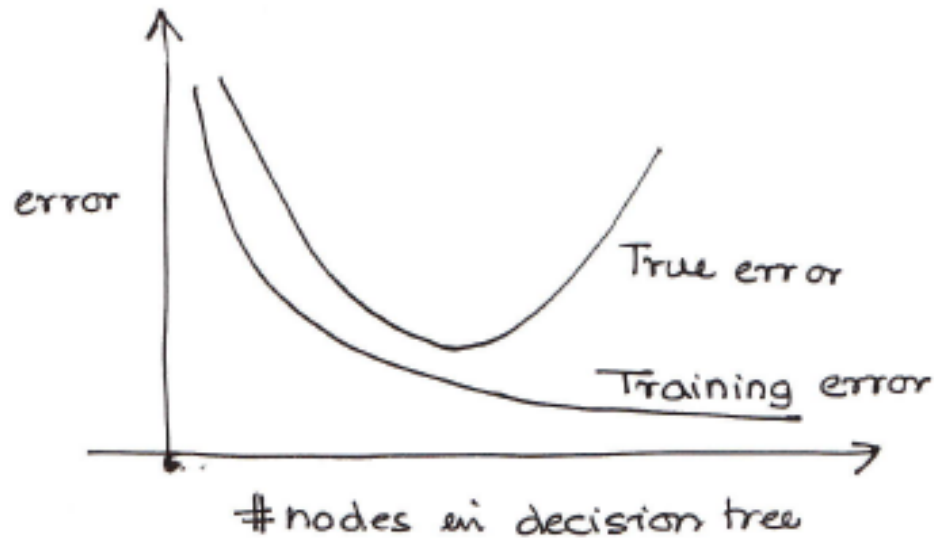
True error of a classifier:

$$err(h) = \Pr_{(x,y) \sim \mathcal{D}}(h(x) \neq y)$$

For a fixed classifier, with more data, training error should approach true error.

Over-fitting in Decision Trees

Overfitting happens when:



As we make the tree more and more complicated, training error decreases. At some point, true error stops improving and may get worse.

True data distribution has some structure, which we would like to capture. Training data captures some of this structure, but may have some "chance" structure of its own, which we should not model into a classifier.

Probabilistic Classifier

- Given an instance \mathbf{x} , what does a probabilistic classifier do differently compared to, say, perceptron?
- It does not directly predict y
- Instead, it first computes the probability that the instance belongs to different classes, i.e., $P(y|\mathbf{x})$ – the posterior probability of y given \mathbf{x}
- Given $p(y|\mathbf{x})$, it then makes a prediction using decision theory

Decision Theory

- **Goal 1:** Minimizing the probability of mistake
 - ▶ $y^* = \arg \max_y P(y|\mathbf{x})$
 - ▶ i.e., predict the class that has the maximum posterior probability
- **Goal 2:** minimizing the expected loss

» Given a cost matrix specifying the cost of different types of mistakes

True label→ Predicted ↓	Spam	Non-spam
Spam	0	10
Non-spam	1	0

$$y^* = \arg \min_y \sum_{y'} \underbrace{L(y, y') P(y'|\mathbf{x})}_{\text{Expected cost if we predict } y}$$

Expected cost if we predict y

A Simple Example

- Suppose our probabilistic spam-filter gives the following posterior for an incoming email \mathbf{x} :

▲ $P(y = \text{spam}|\mathbf{x}) = 0.6$

True label→ Predicted ↓	Spam	Non-spam
Spam	0	10
Non-spam	1	0

- The expected cost if predict spam?
 - If it is a spam: no cost (0.6 prob)
 - If it is not – cost of 10 (0.4 prob)
 - $0.6 \times 0 + 0.4 \times 10 = 4$
- What if we predict non-spam?
 - $0.6 \times 1 + 0.4 \times 0 = 0.6$

$$y^* = \arg \min_y \sum_{y' \in \{s, ns\}} L(y, y') P(y'|\mathbf{x}) = \text{non-spam}$$

Two Main Approaches

- To learn a probabilistic classifier, there are two types of approaches

- **Generative:**

- ▶ Learn $P(y)$ and $P(\mathbf{x}|y)$
- ▶ Compute $P(y|\mathbf{x})$ using Bayes rule

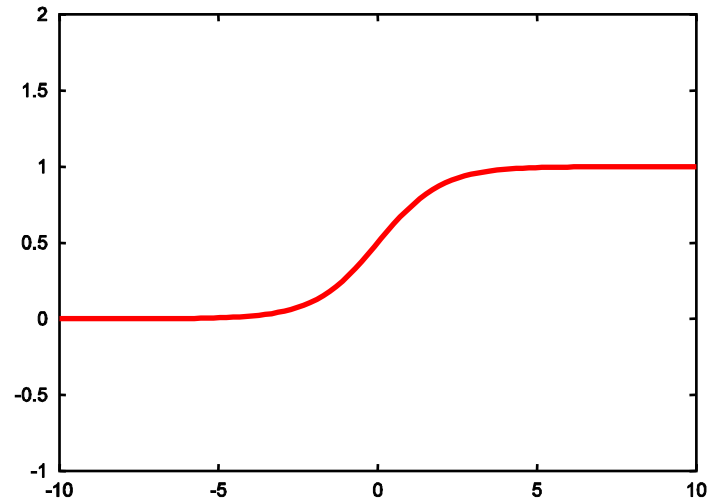
$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} = \frac{P(\mathbf{x}|y)P(y)}{\sum_y P(\mathbf{x}, y)}$$

- **Discriminative:**

- ▶ Learn $P(y|\mathbf{x})$ directly
- ▶ Logistic regression is one of such techniques

The Logistic (Sigmoid) Function

$$g(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$



- The output of a linear function $\mathbf{w}^T \mathbf{x}$ has range $(-\infty, \infty)$. A logistic sigmoid function transforms the value of $\mathbf{w}^T \mathbf{x}$ into a range between 0 and 1

Logistic Regression: Learning Setup

- Given a set of training examples:
 $(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^N, y^N)$
- We assume that \mathbf{x} and y are probabilistically related (parameterized by \mathbf{w})

$$P(y = 1|\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

- **Goal:** learn \mathbf{w} from the training data using Maximum Likelihood Estimation (MLE)

Likelihood Function

- We assume each training example (\mathbf{x}^i, y^i) is drawn **IID** from the same (but unknown) distribution $P(\mathbf{x}, y)$:

$$\log P(D; \mathbf{w}) = \log \prod_i P(\mathbf{x}^i, y^i; \mathbf{w}) = \sum_i \log P(\mathbf{x}^i, y^i; \mathbf{w})$$

- Joint distribution $P(a, b)$ can be factored as $P(a | b)P(b)$

$$\begin{aligned} \arg \max_{\mathbf{w}} \log P(D; \mathbf{w}) &= \arg \max_{\mathbf{w}} \sum_i \log P(\mathbf{x}^i, y^i; \mathbf{w}) \\ &= \arg \max_{\mathbf{w}} \sum_i \log P(y^i | \mathbf{x}^i; \mathbf{w}) P(\mathbf{x}^i; \mathbf{w}) \end{aligned}$$

- Further, $P(\mathbf{x}; \mathbf{w})$ can be dropped because it does not depend on \mathbf{w} :

$$\arg \max_{\mathbf{w}} \log P(D; \mathbf{w}) = \arg \max_{\mathbf{w}} \sum_i \log P(y^i | \mathbf{x}^i; \mathbf{w})$$

Batch Gradient Ascent for LR

Given : training examples (\mathbf{x}^i, y^i) , $i = 1, \dots, N$

Let $\mathbf{w} \leftarrow (0, 0, 0, \dots, 0)$

Repeat until convergence

$\mathbf{d} \leftarrow (0, 0, 0, \dots, 0)$

For $i = 1$ to N do

$$\hat{y}^i \leftarrow \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}^i}}$$

$$error = y^i - \hat{y}^i$$

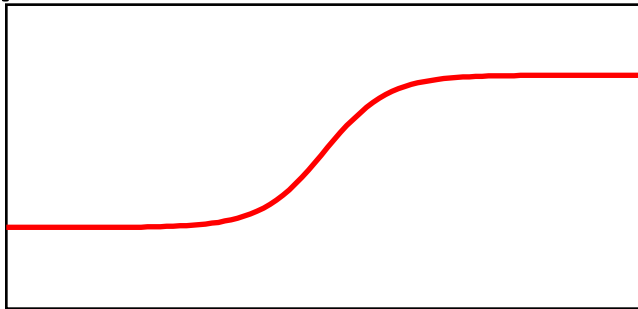
$$\mathbf{d} = \mathbf{d} + error \cdot \mathbf{x}^i$$

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{d}$$

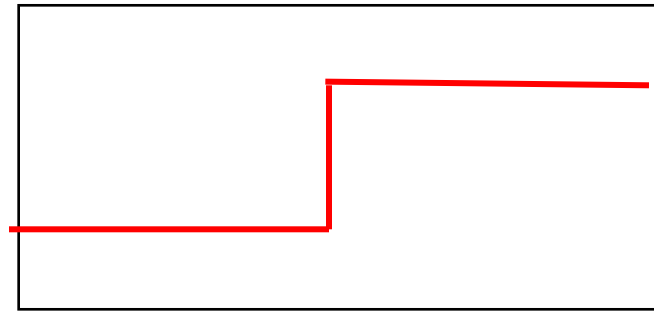
- **Online** gradient ascent algorithm can be easily constructed

Connection between Logistic Regression and Perceptron Algorithm

- Both methods learn a linear function of the input features
- LR uses the logistic function, Perceptron uses a step function



$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$



$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Both algorithms take a similar update rule:

$$\mathbf{w} = \mathbf{w} + \eta(y^i - h_{\mathbf{w}}(\mathbf{x}^i))\mathbf{x}^i$$

Bayes Classifier

- Generative model learns $p(y)$ and $p(\mathbf{x}|y)$

- Prediction is made by

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|y)p(y)}{\sum_y p(\mathbf{x}, y)}$$

- Often referred to as the Bayes Classifier due to using the **Bayes rule**

Learning a joint distribution

Build a JD table in which the probabilities are unspecified

A	B	C	Prob
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

Fraction of all records in which
A and B are True but C is False

Then fill in each row with

$$\hat{P}(\text{row}) = \frac{\text{records matching row}}{\text{total number of records}}$$

A	B	C	Prob
0	0	0	0.30
0	0	1	0.05
0	1	0	0.10
0	1	1	0.05
1	0	0	0.05
1	0	1	0.10
1	1	0	0.25
1	1	1	0.10

Learning Joint Distribution and Overfitting

- Let \mathbf{x} be a d -dimensional binary vector, and $y \in \{1, 2, \dots, k\}$
- Learning the joint distribution $P(\mathbf{x}|y = i)$ for $i = 1, \dots, k$ involves estimating $k \times (2^d - 1)$ parameters
 - ▲ For large d , this number is prohibitively large
 - ▲ Not enough data to estimate the joint distribution accurately
 - ▲ Common to encounter the situation where no training examples have the exact $\mathbf{x} = [u_1, \dots, u_d]^T$ value combination
 - ▲ Then $P(\mathbf{x} = [u_1, \dots, u_d]^T | y = i) = 0$ for all values of i
 - ▲ This will lead to severe overfitting

Naïve Bayes Assumption

- **Assumption:** each feature is independent from one another given the class label
- **Definition:** x is **conditionally independent** of y given z , if the probability distribution governing x is independent of the value of y , given the value of z
$$\forall i, j, k \ P(x = i | y = j, z = k) = P(x = i | z = k)$$

Often denoted as $p(x|y, z) = p(x|z)$

- **Example:**

$$\begin{aligned} p(thunder|raining, lightning) \\ = p(thunder|lightning) \end{aligned}$$

Naïve Bayes Classifier

- Under Naïve Bayes assumption, we have:

$$p(\mathbf{x}|y) = \prod_{i=1}^d p(x_i|y)$$

- No need to estimate the joint distribution
- We only need to estimate $p(x_i|y)$ for each feature i
- **Example:** with d binary features and k classes, we reduce the number of parameters from $k(2^d - 1)$ to kd
 - ▲ Significantly reduces overfitting

MLE for Naïve Bayes with Bernoulli Model

- Suppose our training set contains N emails, maximum likelihood estimate of the parameters are:

$$P(y = 1) = \frac{N_1}{N}, \text{ where } N_1 \text{ is the number of spam emails}$$

$$P(x_i = 1 \mid y = 1) = \frac{N_{i|1}}{N_1},$$

i.e., the fraction of spam emails where x_i appeared

$$P(x_i = 1 \mid y = 0) = \frac{N_{i|0}}{N_0}$$

i.e., the fraction of nonspam emails where x_i appeared

Laplace Smoothing

- Suppose we estimate a probability $P(z)$ and we have n_0 examples where $z = 0$ and n_1 examples where $z = 1$ MLE estimate is

$$P(z = 1) = \frac{n_1}{n_0 + n_1}$$

- **Laplace Smoothing:** Add 1 to the numerator and 2 to the denominator

$$P(z = 1) = \frac{n_1 + 1}{n_0 + n_1 + 2}$$

If we don't observe any examples, we expect $P(z=1) = 0.5$, but our belief is weak (equivalent to seeing one example of each outcome).