

CSC 2515: Introduction to Machine Learning

Lecture 5: SVMs and Ensembles: Boosting

Amir-massoud Farahmand¹

University of Toronto and Vector Institute

¹Credit for slides goes to many members of the ML Group at the U of T, and beyond, including (recent past): Roger Grosse, Murat Erdogdu, Richard Zemel, Juan Felipe Carrasquilla, Emad Andrews, and myself.

Table of Contents

- 1 Support Vector Machines
 - Optimal Separating Hyperplane
 - Non-Separable Data Points
- 2 Kernel Methods
 - Dual Formulation of SVM
 - From Inner Products to Kernels
- 3 Boosting
 - AdaBoost
 - Additive Models
 - Additive Models with Exponential Loss

Today

- We have seen cross-entropy loss for classification. Any other choices?
- We introduce two new loss functions for classification
 - ▶ One of them is derived based on a geometric approach
- Kernel methods: A new way to feature expansion without explicitly doing so
- We become familiar with the concept of weak learners, additive models, and boosting
- Skills to Learn
 - ▶ Support Vector Machine
 - ▶ Boosting
 - ▶ Kernel Method

Binary Classification with a Linear Model

- Classification: Predict a discrete-valued target
- Binary classification: Targets $t \in \{-1, +1\}$ (This is different than previous lectures where we had $t \in \{0, +1\}$).
- Linear model:

$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$y = \text{sign}(z)$$

- Question: How should we choose \mathbf{w} and b ?

Zero-One Loss

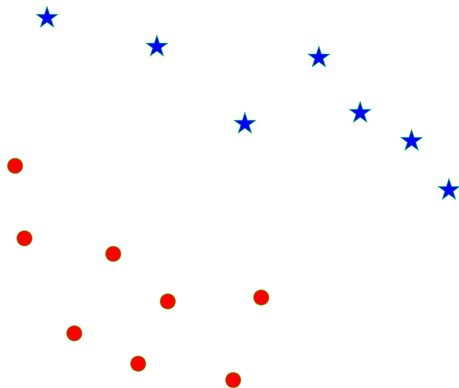
- We can use the 0 – 1 loss function, and find the weights that minimize the sum of loss functions over training data points.

$$\begin{aligned}\mathcal{L}_{0-1}(y, t) &= \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases} \\ &= \mathbb{I}\{y \neq t\}.\end{aligned}$$

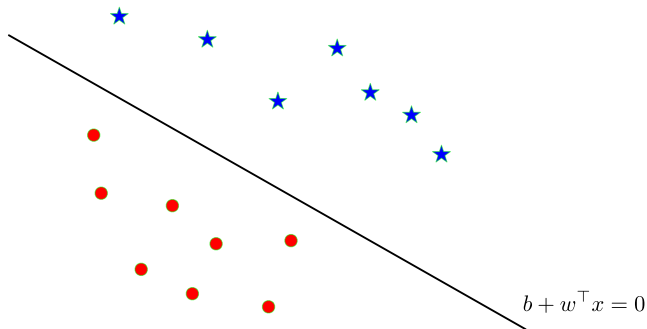
- But minimizing this loss is computationally difficult, and it cannot distinguish different hypotheses that achieve the same accuracy.
- We investigated some other loss functions that are easier to minimize, e.g., logistic regression with the cross-entropy loss \mathcal{L}_{CE} .
- Let's consider a different approach, starting from the [geometry](#) of binary classifiers.

Separating Hyperplanes

Suppose that we are given these data points from two different classes and want to find a linear classifier that separates them.

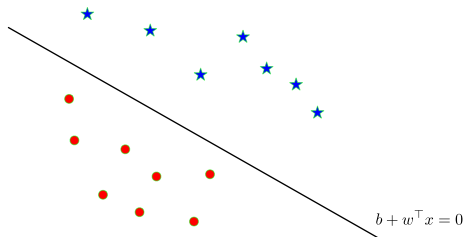


Separating Hyperplanes



- The decision boundary looks like a line because $\mathbf{x} \in \mathbb{R}^2$, but think of it as a $D - 1$ dimensional hyperplane when $\mathbf{x} \in \mathbb{R}^D$.
- Recall that a hyperplane is described by points $\mathbf{x} \in \mathbb{R}^D$ such that $f(\mathbf{x}) = \mathbf{w}^T x + b = 0$.

Separating Hyperplanes

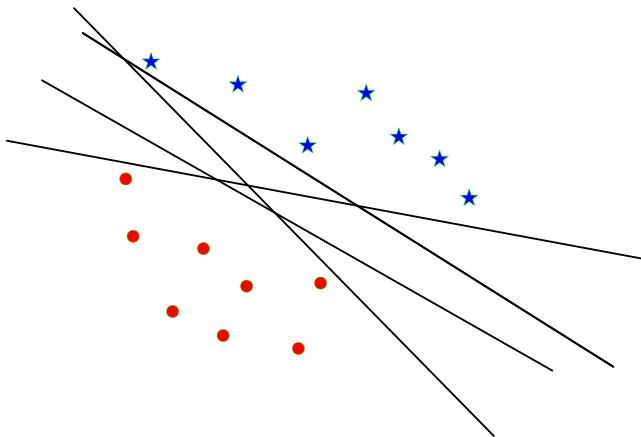


How can we find a separating hyperplane?

- Data points with **positive** label should satisfy $f(\mathbf{x}) = \mathbf{w}^T x + b > 0$.
- Data points with **negative** label should satisfy $f(\mathbf{x}) = \mathbf{w}^T x + b < 0$.
- We can formulate it as a **Linear Program**:

$$\begin{aligned} & \min_{\mathbf{w}, b} 1 \\ & \text{s.t. } (\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq +1 \quad \forall i \text{ with } t^{(i)} = +1 \\ & \quad (\mathbf{w}^T \mathbf{x}^{(i)} + b) \leq -1 \quad \forall i \text{ with } t^{(i)} = -1 \end{aligned}$$

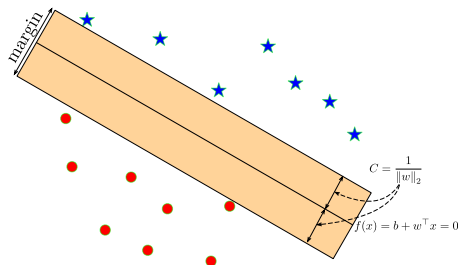
Separating Hyperplanes



- There are more than one separating hyperplane. They are described by different \mathbf{w} s.
- Which one should we choose?

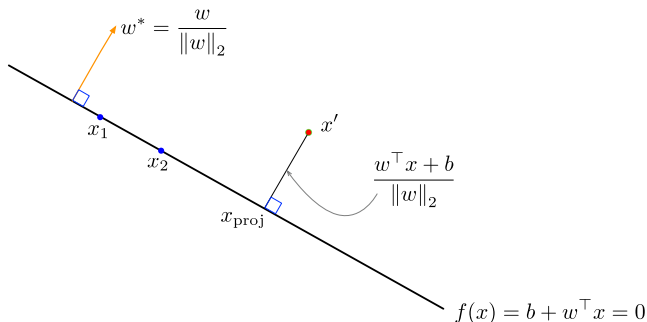
Optimal Separating Hyperplane

Optimal Separating Hyperplane: A hyperplane that separates two classes and maximizes the distance to the closest point from either class, i.e., maximize the **margin** of the classifier.



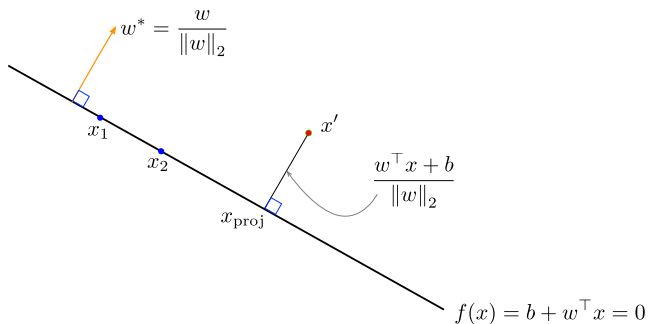
Intuitively, ensuring that a classifier is not too close to any data points leads to better generalization on the test data.

Geometry of Points and Planes



- Recall that the decision hyperplane is orthogonal (perpendicular) to \mathbf{w} .
- The vector $\mathbf{w}^* = \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$ is a unit vector pointing in the same direction as \mathbf{w} .
- The same hyperplane could equivalently be defined in terms of \mathbf{w}^* .

Geometry of Points and Planes



The (signed) distance of a point \mathbf{x}' to the hyperplane is

$$\frac{\mathbf{w}^T \mathbf{x}' + b}{\|\mathbf{w}\|_2}$$

Maximizing Margin as an Optimization Problem

- Recall: the classification for the i -th data point is correct when

$$\text{sign}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) = t^{(i)}$$

- This can be rewritten as

$$t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) > 0$$

- Enforcing a margin of C :

$$t^{(i)} \cdot \underbrace{\frac{(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2}}_{\text{signed distance}} \geq C$$

Maximizing Margin as an Optimization Problem

- The distance of the i -th point to the hyperplane described by \mathbf{w} and b :

$$d_i = \frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2}$$

- **Maximizing Margin:** Maximize the minimum distance of points to the hyperplane (by optimizing w.r.t. \mathbf{w} and b):

$$\max_{\mathbf{w}, b} \min_{i=1, \dots, N} d_i$$

- Or equivalently,

$$\begin{aligned} & \max_{\mathbf{w}, b} C \\ \text{s.t.} & \frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C \quad i = 1, \dots, N \end{aligned}$$

Maximizing Margin as an Optimization Problem

Max-margin objective:

$$\begin{aligned} & \max_{\mathbf{w}, b} C \\ \text{s.t.} & \frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C \quad i = 1, \dots, N \end{aligned}$$

We have

$$\frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C \iff t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq C \|\mathbf{w}\|_2.$$

If we multiply \mathbf{w} and b by any positive constant, the inequality is still satisfied. So, the norm of \mathbf{w} does not matter. We arbitrary set it equal to $\|\mathbf{w}\|_2 = \frac{1}{C}$. Plug in $C = 1/\|\mathbf{w}\|_2$ and simplify:

$$\underbrace{\frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq \frac{1}{\|\mathbf{w}\|_2}}_{\text{geometric margin constraint}} \iff \underbrace{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1}_{\text{algebraic margin constraint}}$$

Maximizing Margin as an Optimization Problem

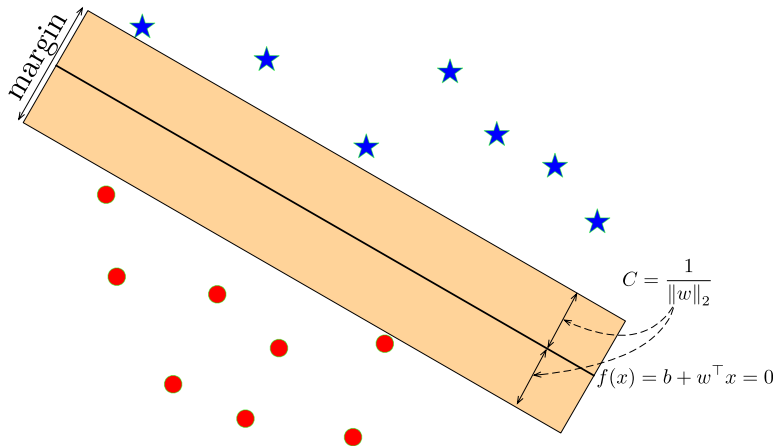
Max-margin objective:

$$\begin{aligned} & \max_{\mathbf{w}, b} C \\ & \text{s.t. } \frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C \quad i = 1, \dots, N \end{aligned}$$

As maximizing $C = 1/\|\mathbf{w}\|_2$ is equivalent to minimizing $\|\mathbf{w}\|_2$, we have

$$\begin{aligned} & \min_{\mathbf{w}, b} \|\mathbf{w}\|_2^2 \\ & \text{s.t. } t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \quad i = 1, \dots, N \end{aligned}$$

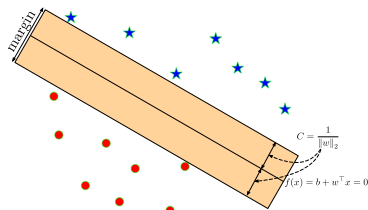
Maximizing Margin as an Optimization Problem



Maximizing Margin as an Optimization Problem

Algebraic max-margin objective:

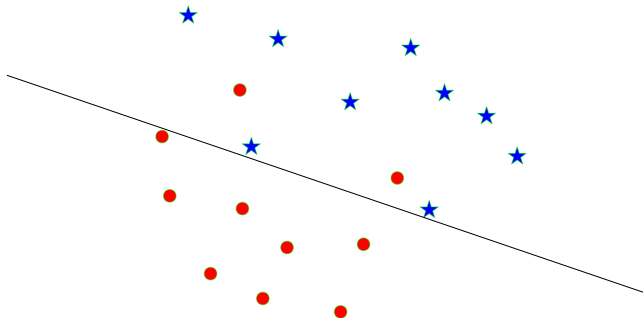
$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \|\mathbf{w}\|_2^2 \\ \text{s.t.} \quad & t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \quad i = 1, \dots, N \end{aligned}$$



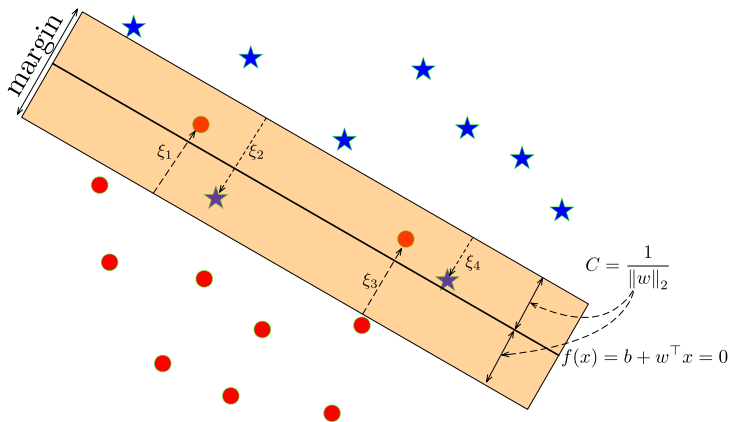
- This is a **Quadratic Program**: Quadratic objective + Linear inequality constraints.
- Observe: if the margin constraint is not tight for $\mathbf{x}^{(i)}$, we could remove it from the training set and the optimal \mathbf{w} would be the same.
- The important training examples are the ones with algebraic margin 1, and are called **support vectors**.
- Hence, this algorithm is called the (hard) **Support Vector Machine (SVM)** (or Support Vector Classifier (SVC) – SV “Machine” refers to when we **kernelize** SV Classifier).
- SVM-like algorithms are often called **max-margin** or **large-margin**.

Non-Separable Data Points

How can we apply the max-margin principle if the data are **not** linearly separable?



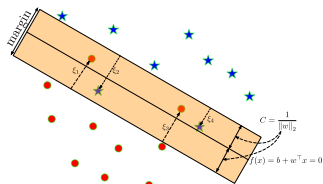
Maximizing Margin for Non-Separable Data Points



Main Idea:

- Allow some points to be within the margin or even be misclassified; we represent this with **slack variables** ξ_i .
- But constrain or penalize the total amount of slacks.

Maximizing Margin for Non-Separable Data Points



- Soft margin constraint:

$$\frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C(1 - \xi_i),$$

for $\xi_i \geq 0$.

- Penalize $\sum_i \xi_i$

Maximizing Margin for Non-Separable Data Points

Soft-margin SVM objective:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 - \xi_i \quad i = 1, \dots, N \\ & \xi_i \geq 0 \quad i = 1, \dots, N \end{aligned}$$

- γ is a hyperparameter that trades off the margin with the amount of slack.
 - ▶ For $\gamma = 0$, we'll get $\mathbf{w} = 0$. (Why?)
 - ▶ As $\gamma \rightarrow \infty$ we get the hard-margin objective.
- Note: It is also possible to constrain $\sum_i \xi_i$ instead of penalizing it.

From Margin Violation to Hinge Loss

Let's simplify the soft margin constraint by eliminating ξ_i . Recall:

$$\begin{aligned}t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) &\geq 1 - \xi_i & i = 1, \dots, N \\ \xi_i &\geq 0 & i = 1, \dots, N\end{aligned}$$

- We would like to find a smallest slack variable ξ_i that satisfy both $\xi_i \geq 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)$ and $\xi_i \geq 0$.
 - ▶ **Case 1:** $1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \leq 0$
 - ▶ The smallest non-negative ξ_i that satisfies the constraint is $\xi_i = 0$.
 - ▶ **Case 2:** $1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) > 0$
 - ▶ The smallest ξ_i that satisfies the constraint is $\xi_i = 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)$.
- Hence, $\xi_i = \max\{0, 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\}$.
- Therefore, the slack penalty can be written as

$$\sum_{i=1}^N \xi_i = \sum_{i=1}^N \max\{0, 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\}.$$

- We sometimes write $\max\{0, z\} = (z)_+$

From Margin Violation to Hinge Loss

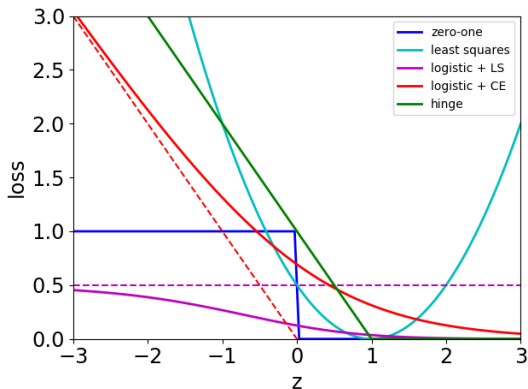
If we write $z^{(i)}(\mathbf{w}, b) = \mathbf{w}^\top \mathbf{x}^{(i)} + b$, the optimization problem can be written as

$$\min_{\mathbf{w}, b} \sum_{i=1}^N \left(1 - t^{(i)} z^{(i)}(\mathbf{w}, b)\right)_+ + \frac{1}{2\gamma} \|\mathbf{w}\|_2^2$$

- The loss function $\mathcal{L}_H(z, t) = (1 - tz)_+$ is called the **hinge loss**.
- The second term is the L_2 -norm of the weights.
- Hence, the soft-margin SV Classifier can be seen as a **linear classifier with hinge loss and an L_2 regularizer**.

Revisiting Loss Functions for Classification

Hinge loss compared with other loss functions

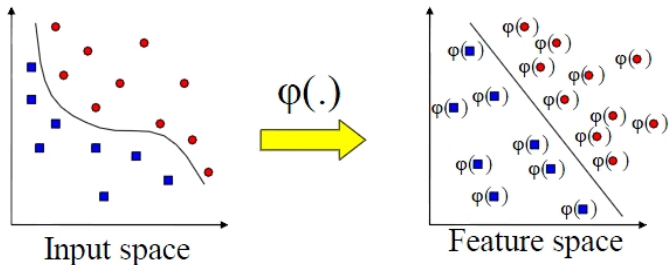


[Figure credit: ESL, Section 10.6]

Kernel Methods

Nonlinear Decision Boundaries

- SV Classifier: Margin maximizing **linear** classifier.
- Linear models are restricted.
- Q: How do we get non-linear decision boundaries?
- Feature mapping $\mathbf{x} \mapsto \phi(\mathbf{x})$



- How do we find good features?
- If features are in a high dimension, the computation cost might be large. Can we avoid it?

Feature Maps

- Let's say that we want a quadratic decision boundary
- What feature mapping do we need?
- One possibility (ignore $\sqrt{2}$ for now)

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \dots, \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \dots, \sqrt{2}x_{d-1}x_d, x_1^2, \dots, x_d^2)$$

Pairwise is over $i < j$

- We have $\dim(\phi(\mathbf{x})) = \mathcal{O}(d^2)$, which is problematic for large d .
- How can this be addressed?
- Let us take a closer look at SVM.

From Primal to Dual Formulation of SVM

- Recall that the SVM is defined using the following constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 - \xi_i \quad i = 1, \dots, N \\ & \xi_i \geq 0 \quad i = 1, \dots, N \end{aligned}$$

- This is called the **primal** formulation.
- We can instead solve a **dual** optimization problem to obtain \mathbf{w} .
 - We do not derive it here in detail. The basic idea is to form the following **Lagrangian**, find \mathbf{w} as a function of α (and other variables), and express the Lagrangian only in terms of the dual variables:

$$L(\mathbf{w}, b, \xi, \alpha, \mu) \triangleq \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^N \xi_i + \sum_{i=1}^N \alpha_i \left[1 - \xi_i - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \right] - \sum_{i=1}^N \mu_i \xi_i$$

From Primal to Dual Formulation of SVM

- Primal Optimization Problem:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 - \xi_i \quad i = 1, \dots, N \\ & \xi_i \geq 0 \quad i = 1, \dots, N \end{aligned}$$

- Dual Optimization Problem:

$$\begin{aligned} \max_{\alpha_i \geq 0} \quad & \left\{ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)\top} \mathbf{x}^{(j)}) \right\} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C; \quad \sum_{i=1}^N \alpha_i t^{(i)} = 0 \end{aligned}$$

- The weights become $\mathbf{w} = \sum_{i=1}^N \alpha_i t^{(i)} \mathbf{x}^{(i)}$, which is a function of the dual variables $(\alpha_i)_{i=1}^N$.
- The prediction of SVM is $y(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x})$.
- **Observation:** The input data only appears in the form of inner products $\mathbf{x}^{(i)\top} \mathbf{x}^{(j)}$.

SVM in Feature Space

- If instead of using input \mathbf{x} , we first map it to a feature space as $\phi(\mathbf{x})$, we get a similar dual optimization problem.
 - ▶ The difference is that we have to substitute the inner product in the input space $\mathbf{x}^{(i)\top} \mathbf{x}^{(j)}$ with the inner product in the feature space $\phi(\mathbf{x}^{(i)})^\top \phi(\mathbf{x}^{(j)})$.

$$\max_{\alpha_i \geq 0} \left\{ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N t^{(i)} t^{(j)} \alpha_i \alpha_j (\phi(\mathbf{x}^{(i)})^\top \phi(\mathbf{x}^{(j)})) \right\}$$

subject to $0 \leq \alpha_i \leq C; \quad \sum_{i=1}^N \alpha_i t^{(i)} = 0$

- The weight vector would be $\mathbf{w} = \sum_{i=1}^N \alpha_i t^{(i)} \phi(\mathbf{x}^{(i)})$.
- The prediction of SVM would be

$$y(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \phi(\mathbf{x})) = \text{sign} \left(\sum_{i=1}^N \alpha_i t^{(i)} \phi(\mathbf{x}^{(i)})^\top \phi(\mathbf{x}) \right).$$

- **Observation:** The input data only appears in the form of inner products $\phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2)$ for some of \mathbf{x}_1 and \mathbf{x}_2 .

From Inner Products to Kernels

- SVMs (and many other “linear” algorithms) are based on the inner product computation.
- For high-dimensional features maps, the explicit computation of $\phi(\mathbf{x})$ and then computing the inner product can be expensive.
 - ▶ Our previous example:

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \dots, \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \dots, \sqrt{2}x_{d-1}x_d, x_1^2, \dots, x_d^2)$$

- ▶ This has $O(d^2)$ computation and memory cost.
- What if we could compute the inner product without explicitly computing $\phi(\mathbf{x})$?
- What is the inner product $\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$?

From Inner Products to Kernels

- What is the inner product $\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$?

$$\begin{aligned}\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle &= 1 + 2 \sum_{i=1}^d x_i y_i + \sum_{i,j=1}^d x_i x_j y_i y_j \\ &= \left(1 + \sum_{i=1}^d x_i y_i\right)^2 = (1 + \langle \mathbf{x}, \mathbf{y} \rangle)^2\end{aligned}$$

- This is $O(d)$ in memory and compute time.
- We call the inner product in the feature space their **kernel** and denote it by $K(\mathbf{x}, \mathbf{y}) \triangleq \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$.
 - ▶ Technically: The feature space Φ together with its inner product define a **Hilbert** space (after some technical considerations). The kernel is simply its inner product.
 - ▶ We may use a kernel to define a **reproducing kernel Hilbert space (RKHS)**, which is a Hilbert space with some nice properties. We do not get into such details.
- We can define kernels without **explicitly** defining the feature space.

- Examples:

1. Polynomial

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + 1)^p,$$

where d is the degree of the polynomial, e.g., $p = 2$ for quadratic

2. Gaussian/RBF

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|_2^2}{2\sigma^2}\right)$$

3. Sigmoid

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\beta \mathbf{x}^\top \mathbf{y} + a)$$

- A kernel measures the **similarity** of two points \mathbf{x} and \mathbf{y} .
- Each kernel computation corresponds to an inner product in the feature space (and not the input space). The calculation is based on an implicitly mapping to a potentially high-dimensional space.
- Kernel functions can be defined for non-vectorized data, e.g., string kernel, graph kernel, etc.

Kernel Functions

- What functions $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ are proper kernels?
 - ▶ Proper in the sense that they are an inner product
 $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$ for some feature map ϕ .
- Any **symmetric** and **positive semidefinite function** $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a proper kernel.
- Positive semidefinite functions are those whose **Gram matrix** is positive semidefinite: For any choice of $\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \in \mathcal{X}$ for $i = 1, \dots, n$, the Gram matrix (or Grammian)

$$[\mathbf{K}]_{ij} = K(\mathbf{x}^{(i)}, \mathbf{y}^{(j)})$$

satisfies $z^\top \mathbf{K} z \geq 0$ for any $z \in \mathbb{R}^n$.

- We can build complicated kernels so long as they are positive semidefinite.
- We can combine simple kernels together to make more complicated ones

Basic Kernel Properties

- Positive constant function is a kernel: for $\alpha \geq 0$, $K'(x_1, x_2) = \alpha$
- Positively weighted linear combinations of kernels are kernels: if $\forall i, \alpha_i \geq 0$, $K'(x_1, x_2) = \sum_i \alpha_i K_i(x_1, x_2)$
- Products of kernels are kernels: $K'(x_1, x_2) = K_1(x_1, x_2)K_2(x_1, x_2)$
- The above transformations preserve positive semidefinite functions
- We can use kernels as building blocks to construct complicated feature mappings

Kernel Feature Space

- Kernels let us express very large feature spaces
 - ▶ polynomial kernel $(1 + \mathbf{x}^{(i)\top} \mathbf{x}^{(j)})^p$ corresponds to feature space with the dimension exponential in p .
 - ▶ Gaussian kernel has infinitely dimensional features.
- Linear separators in these super high-dimensional spaces correspond to **highly non-linear decision boundaries** in the input space

- Dual formulation of SVM only depends on the inner product in the feature space $\phi(\mathbf{x}^{(i)})^\top \phi(\mathbf{x}^{(j)})$. This inner product is the kernel between two inputs, i.e., $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$. So we write

$$\max_{\alpha_i \geq 0} \left\{ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N t^{(i)} t^{(j)} \alpha_i \alpha_j (\phi(\mathbf{x}^{(i)})^\top \phi(\mathbf{x}^{(j)})) \right\} =$$
$$\left\{ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N t^{(i)} t^{(j)} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \right\}$$

subject to $0 \leq \alpha_i \leq C; \quad \sum_{i=1}^N \alpha_i t^{(i)} = 0$

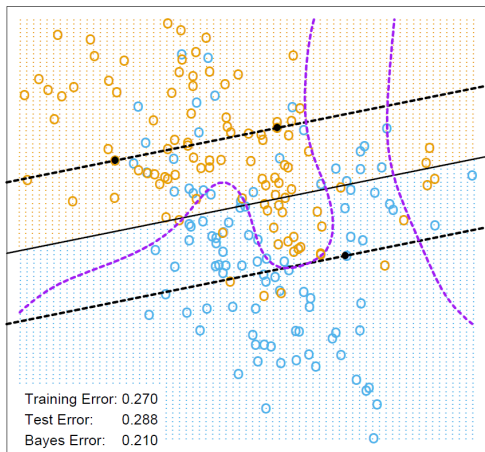
Kernelizing SVMs

- The prediction of SVM would be

$$\begin{aligned}y(\mathbf{x}) &= \text{sign}(\mathbf{w}^\top \phi(\mathbf{x})) = \text{sign}\left(\sum_{i=1}^N \alpha_i t^{(i)} \phi(\mathbf{x}^{(i)})^\top \phi(\mathbf{x})\right) \\ &= \text{sign}\left(\sum_{i=1}^N \alpha_i t^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x})\right).\end{aligned}$$

- The computation is all expressed in terms of kernels between different points. We do not need to compute the feature mapping $\phi(x)$ explicitly.
- This is called the [kernel trick](#).

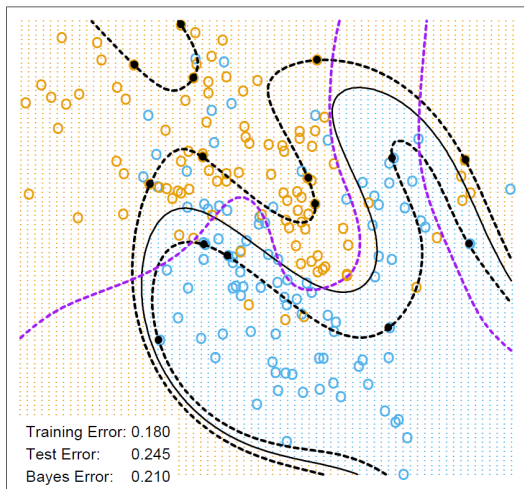
Example: Linear SVM



- Solid line - decision boundary. Dashed - $+1/-1$ margin. Purple - Bayes optimal
- Solid dots - Support vectors on margin

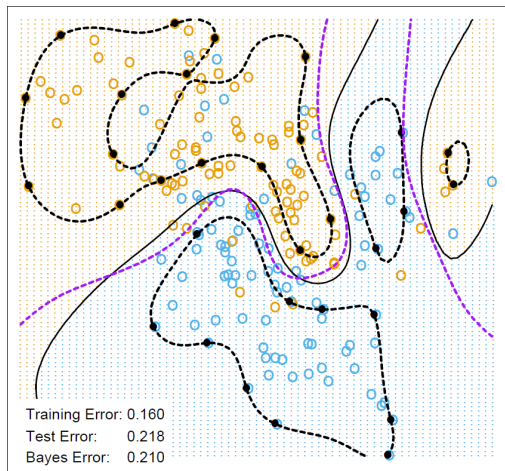
[Image credit: "Elements of statistical learning"]

Example: Degree-4 Polynomial Kernel SVM



[Image credit: "Elements of statistical learning"]

Example: Gaussian Kernel SVM



[Image credit: "Elements of statistical learning"]

Kernel Methods Beyond SVMs

- Kernel method is not limited to SVMs.
- When can we apply the kernel trick?

Representer Theorem

If \mathbf{w}^* is defined as

$$\mathbf{w}^* = \arg \min \sum_{i=1}^N L \left(\langle \mathbf{w}, \phi(\mathbf{x}^{(i)}) \rangle, t^{(i)} \right) + \lambda \|\mathbf{w}\|^2,$$

then $\mathbf{w}^* \in \text{span}\{\phi(x_1), \dots, \phi(x_N)\}$, i.e., $\mathbf{w}^* = \sum_{i=1}^N \alpha_i \phi(x_i)$ for some $\alpha \in \mathbb{R}^N$.

- We assume we can predict using inner product computation.

Optimization

- We can compute

$$\langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \left\langle \sum_{i=1}^N \alpha_i \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \right\rangle = \sum_{i=1}^N \alpha_i \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \rangle = \sum_{i=1}^N \alpha_i K(\mathbf{x}^{(i)}, \mathbf{x})$$

- Similarly for the regularizer

$$\begin{aligned} \|\mathbf{w}\|^2 &= \left\langle \sum_{i=1}^N \alpha_i \phi(\mathbf{x}^{(i)}), \sum_{j=1}^N \alpha_j \phi(\mathbf{x}^{(j)}) \right\rangle = \sum_{i,j=1}^N \alpha_i \alpha_j \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle \\ &= \sum_{i=1}^N \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \end{aligned}$$

- We can optimize without computing $\phi(\mathbf{x})$.

$$\alpha = \operatorname{argmin}_{\alpha \in \mathbb{R}^N} \sum_{i=1}^N L \left(\sum_{j=1}^N \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}), t^{(i)} \right) + \lambda \sum_{i=1}^N \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Kernel Logistic regression
 - ▶ We can think of logistic regression as minimizing $\log(1 + \exp(-t^{(i)} \mathbf{w}^T \mathbf{x}^{(i)}))$
 - ▶ If you use L_2 regularization (Gaussian prior) this fits the representer theorem.
 - ▶ Performance is close to SVM

- The kernel trick is not limited to SVM, but is most common with it.
- Why do the kernel trick and SVM work well together?
- Generalization:
 - ▶ The kernel trick allows us to work in very high dimensions.
 - ▶ Regularization allows us to control the complexity of the high dimensional space.
- Computation:
 - ▶ In general, \mathbf{w}^* is a linear combination of the training data N .
 - ▶ This might become an issue for large N .

Boosting

Ensembles: Boosting

- Recall that an *ensemble* is a set of predictors whose individual decisions are combined in some way to classify new examples.
- (Previously) **Bagging**: Train classifiers independently on random subsamples of the training data.
- (This lecture) **Boosting**: Train classifiers sequentially, each time focusing on training data points that were previously misclassified.
- Let's start with the concepts of **weighted training sets** and **weak learner/classifier** (or base classifiers).

Weighted Training Set

- The misclassification rate $\frac{1}{N} \sum_{n=1}^N \mathbb{I}[h(x^{(n)}) \neq t^{(n)}]$ weighs each training example equally.
- Key idea: We can learn a classifier using different cost (aka weight) for each example.
 - ▶ Classifier “tries harder” on examples with higher cost
- Change cost function:

$$\sum_{n=1}^N \frac{1}{N} \mathbb{I}[h(x^{(n)}) \neq t^{(n)}] \quad \text{becomes} \quad \sum_{n=1}^N w^{(n)} \mathbb{I}[h(x^{(n)}) \neq t^{(n)}]$$

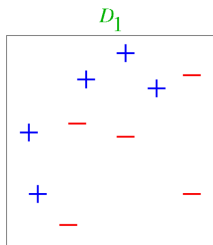
- Usually require each $w^{(n)} > 0$ and $\sum_{n=1}^N w^{(n)} = 1$

Weak Learner/Classifier

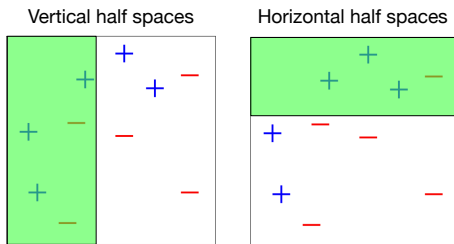
- (Informal) Weak learner is a learning algorithm that outputs a hypothesis (i.e., a classifier) that performs slightly better than chance, e.g., it predicts the correct label with probability 0.51 in binary label case.
 - ▶ It gets slightly less than 0.5 error rate (the worst case is 0.5)
- We are interested in weak learners that are *computationally* efficient.
 - ▶ Decision trees
 - ▶ Even simpler: **Decision Stump**: A decision tree with a single split

[Formal definition of weak learnability has quantifiers such as “for any distribution over data” and the requirement that its guarantee holds only probabilistically.]

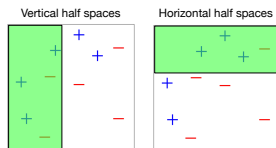
Weak Classifiers



These weak classifiers, which are decision stumps, consist of the set of horizontal and vertical half spaces.



Weak Classifiers



- A *single* weak classifier is not capable of making the training error very small. It only performs slightly better than chance, i.e., the error of **classifier** h according to the given weights $\{w^{(1)}, \dots, w^{(N)}\}$ (with $\sum_{n=1}^N w^{(n)} = 1$ and $w^{(n)} \geq 0$)

$$\text{err} = \sum_{n=1}^N w^{(n)} \mathbb{I}[h(\mathbf{x}^{(n)}) \neq t^{(n)}]$$

is at most $\frac{1}{2} - \gamma$ for some small $\gamma > 0$.

- Can we combine a set of weak classifiers in order to make a better ensemble of classifiers?

AdaBoost (Adaptive Boosting)

- **Boosting**: Train classifiers sequentially, each time assigning higher weights to training data points that were previously misclassified.
- Key steps of **AdaBoost**:
 1. At each iteration we re-weight the training samples by assigning larger weights to samples (i.e., data points) that were classified incorrectly.
 2. We train a new weak classifier based on the re-weighted samples.
 3. We add this weak classifier to the ensemble of weak classifiers. This ensemble is our new classifier.
 4. We repeat the process many times.
- The weak learner needs to minimize weighted error.
- AdaBoost reduces **bias** by making each classifier focus on previous mistakes.

- Input: Data $\mathcal{D}_N = \{\mathbf{x}^{(n)}, t^{(n)}\}_{n=1}^N$ where $t^{(n)} \in \{-1, +1\}$
 - ▶ This is different from previous lectures where we had $t^{(n)} \in \{0, +1\}$
 - ▶ It is for notational convenience; otherwise, it is equivalent.
- A classifier or hypothesis $h : \mathbf{x} \rightarrow \{-1, +1\}$
- 0-1 loss: $\mathbb{I}[h(x^{(n)}) \neq t^{(n)}] = \frac{1}{2}(1 - h(x^{(n)}) \cdot t^{(n)})$

AdaBoost Algorithm

- Input: Data \mathcal{D}_N , weak classifier WeakLearn (a classification procedure that returns a classifier h , e.g., best decision stump, from a set of classifiers \mathcal{H} , e.g., all possible decision stumps), number of iterations T
- Output: Classifier $H(x)$
- Initialize sample weights: $w^{(n)} = \frac{1}{N}$ for $n = 1, \dots, N$
- For $t = 1, \dots, T$
 - ▶ Fit a classifier to data using weighted samples ($h_t \leftarrow \text{WeakLearn}(\mathcal{D}_N, \mathbf{w})$), e.g.,

$$h_t \leftarrow \operatorname{argmin}_{h \in \mathcal{H}} \sum_{n=1}^N w^{(n)} \mathbb{I}\{h(\mathbf{x}^{(n)}) \neq t^{(n)}\}$$

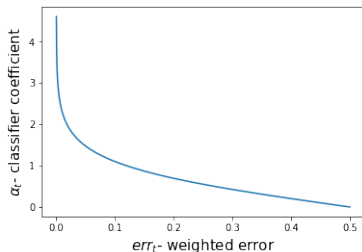
- ▶ Compute weighted error $\text{err}_t = \frac{\sum_{n=1}^N w^{(n)} \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^N w^{(n)}}$
- ▶ Compute classifier coefficient $\alpha_t = \frac{1}{2} \log \frac{1 - \text{err}_t}{\text{err}_t} \quad (\in (0, \infty))$
- ▶ Update data weights ($n = 1, \dots, N$):

$$w^{(n)} \leftarrow w^{(n)} \exp\left(-\alpha_t t^{(n)} h_t(\mathbf{x}^{(n)})\right) \left[\equiv w^{(n)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right) \right]$$

- Return $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x})\right)$

Weighting Intuition

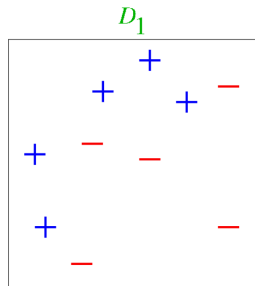
- Recall: $H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$ where $\alpha_t = \frac{1}{2} \log \frac{1 - \text{err}_t}{\text{err}_t}$



- Weak classifiers which get lower weighted error get more weight in the final classifier
- Also: $w^{(n)} \leftarrow w^{(n)} \exp \left(2\alpha_t \mathbb{I} \{ h_t(\mathbf{x}^{(n)}) \neq t^{(n)} \} \right)$
 - ▶ If $\text{err}_t \approx 0$, α_t high so misclassified examples get more attention
 - ▶ If $\text{err}_t \approx 0.5$, α_t low so misclassified examples are not emphasized
- Q: What happens to the weight of correctly classifier data?

AdaBoost Example

- Training data

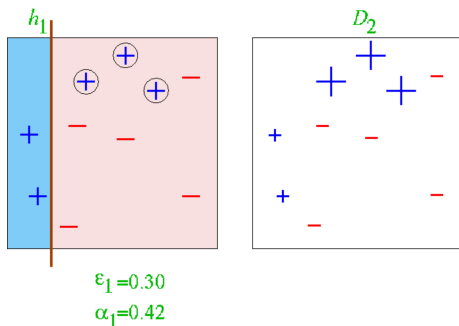


- \mathcal{H} : decision trees with a single split (decision stumps)

[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 1

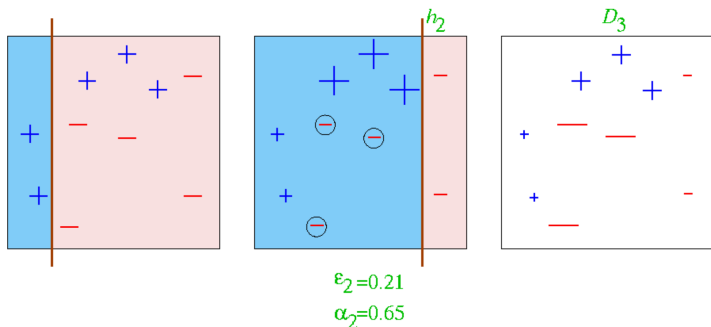


$$\mathbf{w} = \left(\frac{1}{10}, \dots, \frac{1}{10} \right) \Rightarrow \text{Train a classifier (using } \mathbf{w} \text{)} \Rightarrow \text{err}_1 = \frac{\sum_{n=1}^{10} w^{(n)} \mathbb{I}[h_1(\mathbf{x}^{(n)}) \neq t^{(n)}]}{\sum_{n=1}^{10} w^{(n)}} = \frac{3}{10}$$
$$\Rightarrow \alpha_1 = \frac{1}{2} \log \frac{1 - \text{err}_1}{\text{err}_1} = \frac{1}{2} \log \left(\frac{1}{0.3} - 1 \right) \approx 0.42 \Rightarrow H(\mathbf{x}) = \text{sign}(\alpha_1 h_1(\mathbf{x}))$$

[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 2



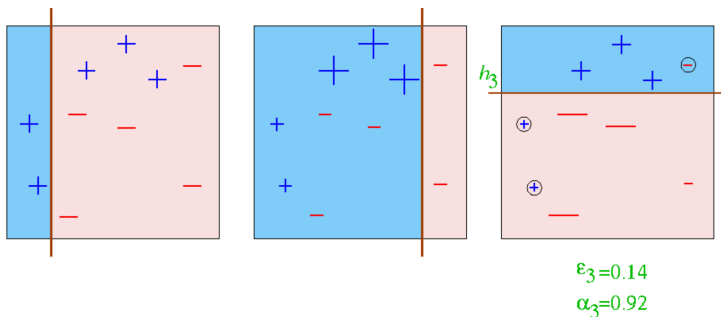
$$\mathbf{w} \leftarrow \text{new weights} \Rightarrow \text{Train a classifier (using } \mathbf{w} \text{)} \Rightarrow \text{err}_2 = \frac{\sum_{n=1}^{10} w^{(n)} \mathbb{I}\{h_2(\mathbf{x}^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^{10} w^{(n)}} = 0.21$$

$$\Rightarrow \alpha_2 = \frac{1}{2} \log \frac{1 - \text{err}_2}{\text{err}_2} = \frac{1}{2} \log \left(\frac{1}{0.21} - 1 \right) \approx 0.66 \Rightarrow H(\mathbf{x}) = \text{sign}(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}))$$

[Slide credit: Verma & Thrun]

AdaBoost Example

- Round 3



$$\epsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

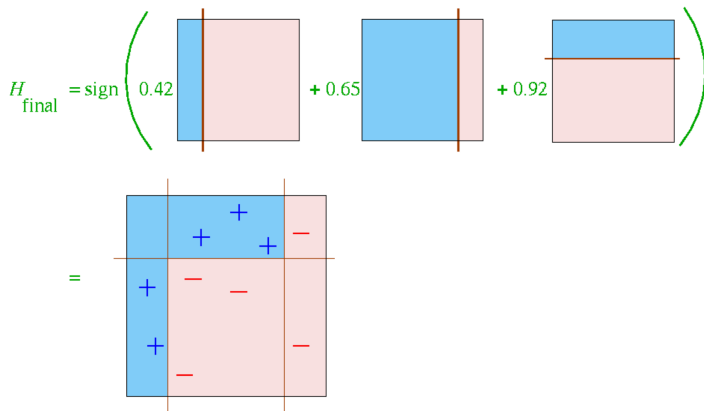
$$\mathbf{w} \leftarrow \text{new weights} \Rightarrow \text{Train a classifier (using } \mathbf{w} \text{)} \Rightarrow \text{err}_3 = \frac{\sum_{n=1}^{10} w^{(n)} \mathbb{I}\{h_3(\mathbf{x}^{(n)}) \neq t^{(n)}\}}{\sum_{i=1}^{10} w^{(i)}} = 0.14$$

$$\Rightarrow \alpha_3 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log \left(\frac{1}{0.14} - 1 \right) \approx 0.91 \Rightarrow H(\mathbf{x}) = \text{sign}(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x}))$$

[Slide credit: Verma & Thrun]

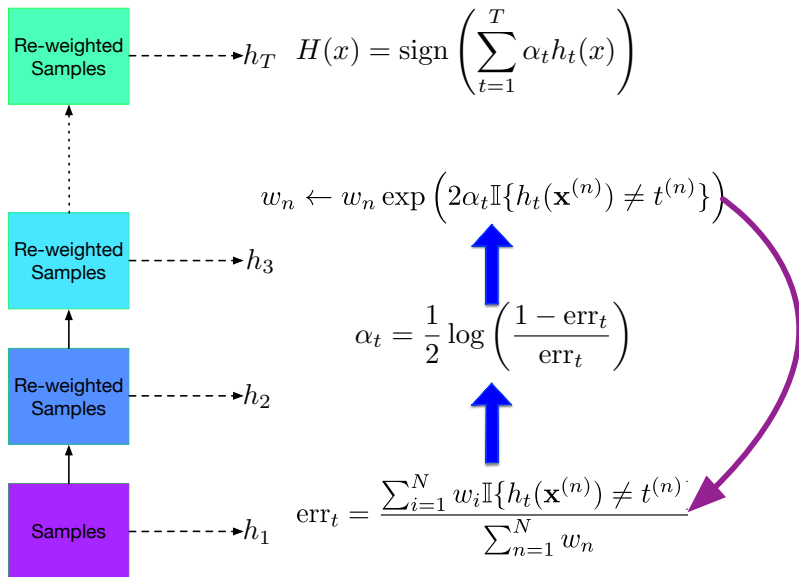
AdaBoost Example

- Final classifier



[Slide credit: Verma & Thrun]

AdaBoost Algorithm



AdaBoost Minimizes the Training Error

Theorem

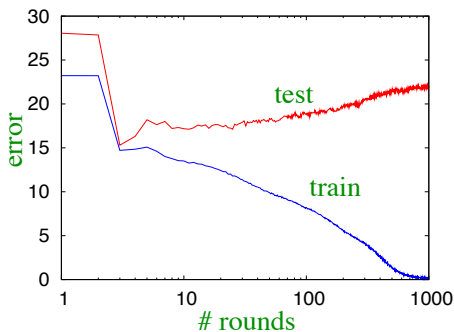
Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error $\text{err}_t \leq \frac{1}{2} - \gamma$ for all $t = 1, \dots, T$ with $\gamma > 0$. The training error of the output hypothesis $H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$ is at most

$$L_N(H) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}\{H(\mathbf{x}^{(n)}) \neq t^{(n)}\} \leq \exp(-2\gamma^2 T).$$

- This is under the simplifying assumption that each weak learner is γ -better than a random predictor.
- This is called **geometric convergence**. It is fast!

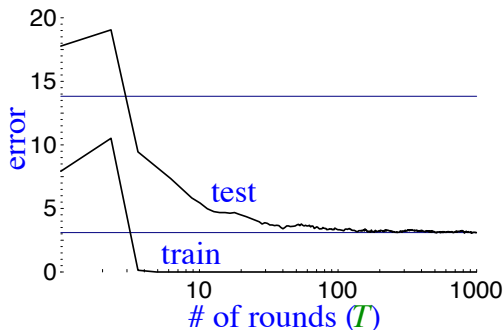
Generalization Error of AdaBoost

- AdaBoost's training error (loss) converges to zero. What about the test error of H ?
- As we add more weak classifiers, the overall classifier H becomes more “complex”.
- We expect more complex classifiers overfit.
- If one runs AdaBoost long enough, it can in fact overfit.



Generalization Error of AdaBoost

- But often it does not!
- Sometimes the test error decreases even after the training error is zero!



- How does that happen?
- Next, we provide an alternative viewpoint on AdaBoost.

[Slide credit: Robert Shapire's Slides,
<http://www.cs.princeton.edu/courses/archive/spring12/cos598A/schedule.html>]

Additive Models

Next, we interpret AdaBoost as a way of fitting an additive model.

- Consider a hypothesis class \mathcal{H} with each $h_i : \mathbf{x} \mapsto \{-1, +1\}$ within \mathcal{H} , i.e., $h_i \in \mathcal{H}$. These are the “weak learners”, and in this context they are also called **bases**.
- An **additive model** with m terms is given by

$$H_m(x) = \sum_{i=1}^m \alpha_i h_i(\mathbf{x}),$$

where $(\alpha_1, \dots, \alpha_m) \in \mathbb{R}^m$ (generally $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$).

- Observe that we are taking a linear combination of base classifiers $h_i(\mathbf{x})$, just like in boosting.

Additive Models

Additive model:

$$H_m(x) = \sum_{i=1}^m \alpha_i h_i(\mathbf{x}),$$

- How can we learn it? Two ways to learn additive models:
 1. Learn all m hypotheses h_i and α_i at the same time:

$$\min_{\{h_i \in \mathcal{H}, \alpha_i\}_{i=1}^m} \sum_{n=1}^N \mathcal{L} \left(H_m(\mathbf{x}^{(n)}), t^{(n)} \right) = \sum_{n=1}^N \mathcal{L} \left(\sum_{i=1}^m \alpha_i h_i(\mathbf{x}^{(n)}), t^{(n)} \right).$$

2. Learn them one by one, i.e., learn h_{m+1} while fixing h_1, \dots, h_m .

Stagewise Training of Additive Models

A greedy approach to fitting additive models, known as **stagewise training**:

1. Initialize $H_0(x) = 0$
2. For $m = 1$ to T :
 - ▶ Compute the m -th hypothesis $H_m = H_{m-1} + \alpha_m h_m$, i.e., h_m and α_m , assuming previous additive model H_{m-1} is fixed:

$$(h_m, \alpha_m) \leftarrow \operatorname{argmin}_{h \in \mathcal{H}, \alpha} \sum_{i=1}^N \mathcal{L} \left(H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)}), t^{(i)} \right)$$

- ▶ Add it to the additive model

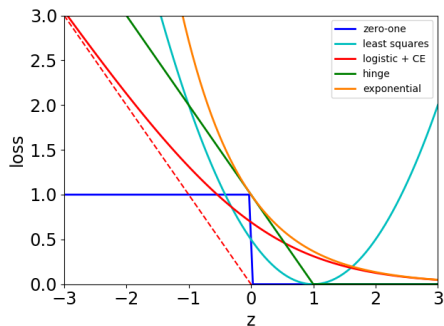
$$H_m = H_{m-1} + \alpha_m h_m$$

Additive Models with Exponential Loss

Consider the exponential loss

$$\mathcal{L}_E(z, t) = \exp(-tz).$$

We want to see how the stagewise training of additive models can be done.



Additive Models with Exponential Loss

Consider the exponential loss

$$\mathcal{L}_E(z, t) = \exp(-tz).$$

We want to see how the stagewise training of additive models can be done.

$$\begin{aligned}(h_m, \alpha_m) &\leftarrow \operatorname{argmin}_{h \in \mathcal{H}, \alpha} \sum_{i=1}^N \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)})\right] t^{(i)}\right) \\ &= \sum_{i=1}^N \exp\left(-H_{m-1}(\mathbf{x}^{(i)})t^{(i)} - \alpha h(\mathbf{x}^{(i)})t^{(i)}\right) \\ &= \sum_{i=1}^N \exp\left(-H_{m-1}(\mathbf{x}^{(i)})t^{(i)}\right) \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right) \\ &= \sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right).\end{aligned}$$

Here we defined $w_i^{(m)} \triangleq \exp\left(-H_{m-1}(\mathbf{x}^{(i)})t^{(i)}\right)$ (doesn't depend on h, α).

Additive Models with Exponential Loss

We want to solve the following minimization problem:

$$(h_m, \alpha_m) \leftarrow \operatorname{argmin}_{h \in \mathcal{H}, \alpha} \sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right).$$

- If $h(\mathbf{x}^{(i)}) = t^{(i)}$ (correct), we have $\exp(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}) = \exp(-\alpha)$.
- If $h(\mathbf{x}^{(i)}) \neq t^{(i)}$ (incorrect), we have $\exp(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}) = \exp(+\alpha)$.

(recall that we are in the binary classification case with $\{-1, +1\}$ output values). We can decompose the summation above into two parts:

$$\sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) = e^{-\alpha} \underbrace{\sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\}}_{\text{correct predictions}} + e^{\alpha} \underbrace{\sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}}_{\text{incorrect predictions}}$$

Additive Models with Exponential Loss

We now add and subtract $e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}$:

$$\begin{aligned} \sum_{i=1}^N w_i^{(m)} \exp(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}) &= \underbrace{e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\}}_{\text{correct predictions}} + \underbrace{e^{\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}}_{\text{incorrect predictions}} \\ &= e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\} + e^{\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} \\ &\quad + e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} - e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} \\ &= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + \\ &\quad e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \underbrace{\left[\mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\} \right]}_{=1}. \end{aligned}$$

Additive Models with Exponential Loss

$$\begin{aligned}\sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right) &= (e^\alpha - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + \\ &\quad e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \left[\mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + \mathbb{I}\{h(\mathbf{x}^{(i)}) = t^{(i)}\} \right] \\ &= (e^\alpha - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^N w_i^{(m)}.\end{aligned}$$

Let us first optimize h : The second term on the RHS does not depend on h .

So we get

$$h_m \leftarrow \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)})t^{(i)}\right) \equiv \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}.$$

This means that h_m is the minimizer of the weighted 0 – 1-loss.

Now that we obtained h_m , we want to find α : Define the weighted classification error:

$$\text{err}_m = \frac{\sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^N w_i^{(m)}}$$

With this definition, and $h_m = \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \exp(-\alpha h(\mathbf{x}^{(i)})t^{(i)})$

$$\begin{aligned} & \min_{\alpha} \min_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \exp(-\alpha h(\mathbf{x}^{(i)})t^{(i)}) = \\ & \min_{\alpha} \left\{ (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\} + e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \right\} \\ & = \min_{\alpha} \left\{ (e^{\alpha} - e^{-\alpha}) \text{err}_m \sum_{i=1}^N w_i^{(m)} + e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \right\} \end{aligned}$$

By taking derivative w.r.t. α and set it to zero, we get

$$e^{2\alpha} = \frac{1 - \text{err}_m}{\text{err}_m} \Rightarrow \alpha = \frac{1}{2} \log \left(\frac{1 - \text{err}_m}{\text{err}_m} \right).$$

Additive Models with Exponential Loss

The updated weights for the next iteration is

$$\begin{aligned}w_i^{(m+1)} &= \exp\left(-H_m(\mathbf{x}^{(i)})t^{(i)}\right) \\&= \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha_m h_m(\mathbf{x}^{(i)})\right]t^{(i)}\right) \\&= \exp\left(-H_{m-1}(\mathbf{x}^{(i)})t^{(i)}\right) \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right) \\&= w_i^{(m)} \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right)\end{aligned}$$

Additive Models with Exponential Loss

To summarize, we obtain the additive model $H_m(x) = \sum_{i=1}^m \alpha_i h_i(\mathbf{x})$ with

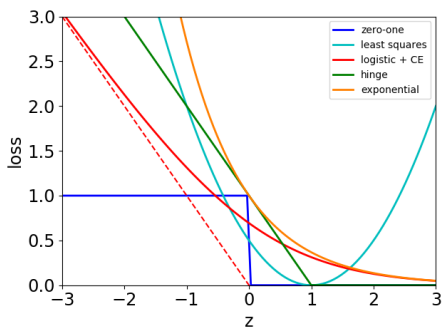
$$h_m \leftarrow \operatorname{argmin}_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\},$$

$$\alpha_m = \frac{1}{2} \log \left(\frac{1 - \operatorname{err}_m}{\operatorname{err}_m} \right), \quad \text{where } \operatorname{err}_m = \frac{\sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^N w_i^{(m)}},$$

$$w_i^{(m+1)} = w_i^{(m)} \exp \left(-\alpha_m h_m(\mathbf{x}^{(i)}) t^{(i)} \right).$$

We derived the AdaBoost algorithm!

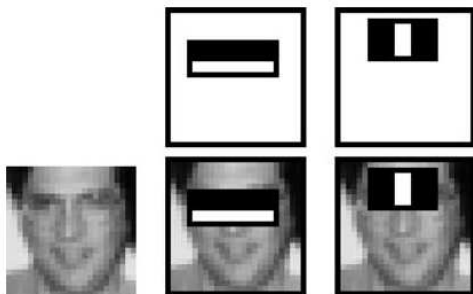
Revisiting Loss Functions for Classification



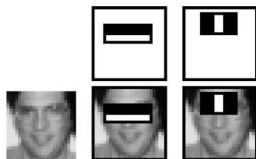
- If AdaBoost is minimizing exponential loss, what does that say about its behavior (compared to, say, logistic regression)?
- This **stagewise training of additive model** interpretation allows boosting to be generalized to lots of other loss functions.

AdaBoost for Face Detection

- Famous application of boosting: detecting faces in images
- Viola and Jones created a very fast face detector that can be scanned across a large image to find the faces.
- A few twists on standard algorithm
 - ▶ Change loss function for weak learners: false positives less costly than misses
 - ▶ Smart way to do inference in real-time (in 2001 hardware)



AdaBoost for Face Recognition



- The base classifier/weak learner just compares the total intensity in two rectangular pieces of the image and classifies based on comparison of this difference to some threshold.
 - ▶ There is a neat trick for computing the total intensity in a rectangle in a few operations.
 - ▶ So it is easy to evaluate a huge number of base classifiers and they are very fast at runtime.
 - ▶ The algorithm adds classifiers greedily based on their quality on the weighted training cases
 - ▶ Each classifier uses just one feature

AdaBoost Face Detection Results



Summary: Boosting

- Boosting reduces bias by generating an ensemble of weak classifiers.
- Each classifier is trained to reduce errors of previous ensemble.
- It is quite resilient to overfitting, though it can overfit.
- Loss minimization viewpoint of AdaBoost allows us to derive other boosting algorithms for regression, ranking, etc.

Summary: Ensembles

- Ensembles combine classifiers to improve performance
- Boosting
 - ▶ Reduces bias
 - ▶ Increases variance (large ensemble can cause overfitting)
 - ▶ Sequential
 - ▶ High dependency between ensemble elements
- Bagging
 - ▶ Reduces variance (large ensemble cannot cause overfitting)
 - ▶ Bias is not changed (much)
 - ▶ Parallel
 - ▶ Want to minimize correlation between ensemble elements.

Summary: SVM and Boosting

- Support Vector Machine (or Classifier) is formulated as the regularized empirical risk minimizer problem with the choice of **hinge** loss and the ℓ_2 regularizer.
- SVM finds the optimal separating hyperplane (separable data) and has a margin maximization property.
- AdaBoost is the solution of stagewise training of an additive model using the **exponential** loss.