Solving optimal margin classifier

- Recall our opt problem:
  \[
  \begin{align*}
  \max_{w, b} & \quad \frac{1}{2} \|w\|^2 \\
  \text{s.t.} & \quad y_i(w^T x_i + b) \geq 1, \quad \forall i
  \end{align*}
  \]

- This is equivalent to
  \[
  \begin{align*}
  \min_{w, b} & \quad \frac{1}{2} w^T w \\
  \text{s.t.} & \quad 1 - y_i(w^T x_i + b) \leq 0, \quad \forall i
  \end{align*}
  \]

- Write the Lagrangian:
  \[
  \mathcal{L}(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{m} \alpha_i [y_i(w^T x_i + b) - 1]
  \]

- Recall that (*) can be reformulated as \( \min_{w, b} \max_{\alpha \geq 0} \mathcal{L}(w, b, \alpha) \)

  Now we solve its dual problem: \( \max_{\alpha \geq 0} \min_{w, b} \mathcal{L}(w, b, \alpha) \)
The Dual Problem

\[
\max_{\alpha \geq 0} \min_w \mathcal{L}(w, b, \alpha)
\]

- We minimize \( \mathcal{L} \) with respect to \( w \) and \( b \) first:
  \[
  \nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y_i x_i = 0, \quad (\ast)
  \]
  \[
  \nabla_b \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y_i = 0, \quad (**)
  \]
  Note that \((\ast)\) implies:
  \[
  w = \sum_{i=1}^{m} \alpha_i y_i x_i \quad (***)
  \]
- Plus (***) back to \( \mathcal{L} \), and using (**), we have:
  \[
  \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j)
  \]

The Dual problem, cont.

- Now we have the following dual opt problem:
  \[
  \max_{\alpha} \mathcal{J}(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j)
  \]
  \[
  \text{s.t. } \alpha_i \geq 0, \quad i = 1, \ldots, k \text{ \quad \quad \sum_{i=1}^{m} \alpha_i y_i = 0.}
  \]
  This is, (again,) a quadratic programming problem.
  - A global maximum of \( \alpha \) can always be found.
  - But what’s the big deal??
  - Note two things:
    1. \( w \) can be recovered by \( w = \sum_{i=1}^{m} \alpha_i y_i x_i \)
    See next …
    2. The "kernel" \( x_i^T x_j \)
    More later …
**Support vectors**

- Note the KKT condition --- only a few $\alpha_i$'s can be nonzero!!

$$\alpha_i g_i(w) = 0 \quad i = 1, \ldots, k$$

![Diagram illustrating support vectors](image)

Call the training data points whose $\alpha_i$'s are nonzero the support vectors (SV)

**Support vector machines**

- Once we have the Lagrange multipliers $\{\alpha_i\}$, we can reconstruct the parameter vector $w$ as a weighted combination of the training examples:

$$w = \sum_{i \in SV} \alpha_i y_i x_i$$

- For testing with a new data $z$
  - Compute
    $$w^T z + b = \sum_{i \in SV} \alpha_i y_i (x_i^T z) + b$$

    and classify $z$ as class 1 if the sum is positive, and class 2 otherwise
  - Note: $w$ need not be formed explicitly
Interpretation of support vector machines

- The optimal \( \mathbf{w} \) is a linear combination of a small number of data points. This "sparse" representation can be viewed as data compression as in the construction of kNN classifier.

- To compute the weights \( \{ \alpha_i \} \), and to use support vector machines we need to specify only the inner products (or kernel) between the examples \( \mathbf{x}_i^T \mathbf{x}_j \).

- We make decisions by comparing each new example \( z \) with only the support vectors:

\[
y^* = \text{sign} \left( \sum_{i \in \text{SV}} \alpha_i y_i (\mathbf{x}_i^T z) + b \right)
\]

Non-linearly Separable Problems

- We allow "error" \( \xi_i \) in classification; it is based on the output of the discriminant function \( \mathbf{w}^T \mathbf{x} + b \).

- \( \xi_i \) approximates the number of misclassified samples.
Soft Margin Hyperplane

- Now we have a slightly different opt problem:

\[
\begin{align*}
\text{min}_{w, b} & \quad \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \\
\text{s.t.} & \quad y_i (w^T x_i + b) \geq 1 - \xi_i, \quad \forall i \\
& \quad \xi_i \geq 0, \quad \forall i
\end{align*}
\]

- \( \xi_i \) are “slack variables” in optimization
- Note that \( \xi_i = 0 \) if there is no error for \( x_i \)
- \( \xi_i \) is an upper bound of the number of errors
- \( C \) : tradeoff parameter between error and margin

The Optimization Problem

- The dual of this new constrained optimization problem is

\[
\begin{align*}
\text{max}_\alpha & \quad J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \\
\text{s.t.} & \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, k \\
& \quad \sum_{i=1}^{m} \alpha_i y_i = 0.
\end{align*}
\]

- This is very similar to the optimization problem in the linear separable case, except that there is an upper bound \( C \) on \( \alpha_i \) now
- Once again, a QP solver can be used to find \( \alpha_i \)
Extension to Non-linear Decision Boundary

- So far, we have only considered large-margin classifier with a linear decision boundary
- How to generalize it to become nonlinear?
- Key idea: transform $x_i$ to a higher dimensional space to “make life easier”
  - Input space: the space the point $x_i$ are located
  - Feature space: the space of $\phi(x_i)$ after transformation
- Why transform?
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - Classification can become easier with a proper transformation.

Transforming the Data

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue
The Kernel Trick

- Recall the SVM optimization problem
  \[
  \max_{\alpha} \quad J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\phi(x_i^T) \phi(x_j))
  \]
  s.t. \( 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, k \)
  \( \sum_{i=1}^{m} \alpha_i y_i = 0. \)

- The data points only appear as inner product
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Many common geometric operations (angles, distances) can be expressed by inner products
- Define the kernel function \( K \) by
  \[
  K(x_i, x_j) = \phi(x_i)^T \phi(x_j)
  \]

An Example for feature mapping and kernels

- Consider an input \( x = [x_1, x_2] \)
- Suppose \( \phi(\cdot) \) is given as follows
  \[
  \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1, \sqrt{2}x_1, -\sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2 \end{bmatrix}^T
  \]
- An inner product in the feature space is
  \[
  \left\langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} x_1' \\ x_2' \end{bmatrix}\right) \right\rangle = 1 + 2x_1x_1' + 2x_2x_2' + x_1^2 + x_1'^2 + x_2^2 + x_2'^2 + 2x_1x_1' + x_1x_2' + x_2x_1' + x_2x_2'
  \]
- So, if we define the kernel function as follows, there is no need to carry out \( \phi(\cdot) \) explicitly
  \[
  K(x, x') = \left(1 + x^T x'\right)^2
  \]
More examples of kernel functions

- Linear kernel (we’ve seen it)
  \[ K(x, x') = x^T x' \]

- Polynomial kernel (we just saw an example)
  \[ K(x, x') = (1 + x^T x')^p \]
  where \( p = 2, 3, \ldots \) To get the feature vectors we concatenate all \( p \)th order polynomial terms of the components of \( x \) (weighted appropriately)

- Radial basis kernel
  \[ K(x, x') = \exp\left(-\frac{1}{2}||x - x'||^2\right) \]
  In this case the feature space consists of functions and results in a non-parametric classifier.

Kernelized SVM

- Training:

\[
\max_{\alpha} \quad J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\
\text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, k \\
\quad \sum_{i=1}^{m} \alpha_i y_i = 0.
\]

- Using:

\[
y^* = \text{sign} \left( \sum_{i \in SV} \alpha_i y_i K(x_i, z) + b \right)
\]
SVM examples

Examples for Non Linear SVMs – Gaussian Kernel
Cross-validation error

- The leave-one-out cross-validation error does not depend on the dimensionality of the feature space but only on the number of support vectors!

\[
\text{Leave-one-out CV error} = \frac{\# \text{ support vectors}}{\# \text{ of training examples}}
\]
Rationale: Combination of methods

- There is no algorithm that is always the most accurate
- We can select simple “weak” classification or regression methods and combine them into a single “strong” method
- Different learners use different
  - Algorithms
  - Hyperparameters
  - Representations (Modalities)
  - Training sets
  - Subproblems
- The problem: how to combine them

Some early algorithms

- Boosting by filtering (Schapire 1990)
  - Run weak learner on differently filtered example sets
  - Combine weak hypotheses
  - Requires knowledge on the performance of weak learner
- Boosting by majority (Freund 1995)
  - Run weak learner on weighted example set
  - Combine weak hypotheses linearly
  - Requires knowledge on the performance of weak learner
- Bagging (Breiman 1996)
  - Run weak learner on bootstrap replicates of the training set
  - Average weak hypotheses
  - Reduces variance
Combination of classifiers

- Suppose we have a family of component classifiers (generating ±1 labels) such as decision stumps:
  \[ h(x; \theta) = \text{sign}(wx_k + b) \]
  where \( \theta = (k, w, b) \)

- Each decision stump pays attention to only a single component of the input vector.

Combination of classifiers con’d

- We’d like to combine the simple classifiers additively so that the final classifier is the sign of
  \[ \hat{h}(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m) \]
  where the “votes” \( \{ \alpha_i \} \) emphasize component classifiers that make more reliable predictions than others.

- Important issues:
  - what is the criterion that we are optimizing? (measure of loss)
  - we would like to estimate each new component classifier in the same manner (modularity)
Measurement of error

- Loss function:
  \[ \lambda(y, h(x)) \quad \text{(e.g. } I(y \neq h(x)) \text{)} \]

- Generalization error:
  \[ L(h) = E[\lambda(y, h(x))] \]

- Objective: find \( h \) with minimum generalization error

- Main boosting idea: minimize the empirical error:
  \[ \hat{L}(h) = \frac{1}{N} \sum_{n=1}^{N} \lambda(y_n, h(x_n)) \]

Exponential Loss

- One possible measure of empirical loss is
  \[ \sum_{i=1}^{C} \exp\left\{ -y_i \hat{h}_m(x_i) \right\} \]
  \[ = \sum_{i=1}^{n} \exp\left\{ y_i \hat{h}_{m-1}(x_i) - y_i a_m h(x_i; \theta_m) \right\} \]
  \[ = \sum_{i=1}^{n} \exp\left\{ y_i \hat{h}_{m-1}(x_i) \right\} \exp\left\{ -y_i a_m h(x_i; \theta_m) \right\} \]
  \[ = \sum_{i=1}^{n} W_i^{m-1} \exp\left\{ -y_i a_m h(x_i; \theta_m) \right\} \]

- The combined classifier based on \( m = 1 \) iterations defines a weighted loss criterion for the next simple classifier to add
- each training sample is weighted by its "classifiability" (or difficulty) seen by the classifier we have built so far
Linearization of loss function

- We can simplify a bit the estimation criterion for the new component classifiers (assuming $\alpha$ is small)
  \[ \exp \left\{ -y_i \alpha h(x_i; \theta_m) \right\} \approx 1 - y_i \alpha h(x_i; \theta_m) \]

- Now our empirical loss criterion reduces to
  \[
  \sum_{i=1}^{n} \exp \left\{ -y_i \hat{h}_m(x_i) \right\} \\
  \approx \sum_{i=1}^{n} W_{i}^{m-1} (1 - y_i \alpha h(x_i; \theta_m)) \\
  = \sum_{i=1}^{n} W_{i}^{m-1} - a_n \sum_{i=1}^{n} W_{i}^{m-1} y_i h(x_i; \theta_m) 
  \]

- We could choose a new component classifier to optimize this weighted agreement

A possible algorithm

- At stage $m$ we find $\hat{\theta}_m^*$ that maximize (or at least give a sufficiently high) weighted agreement:
  \[
  \sum_{i=1}^{n} W_{i}^{m-1} y_i h(x_i; \theta_m^*) 
  \]
  - each sample is weighted by its “difficulty” under the previously combined $m - 1$ classifiers,
  - more “difficult” samples received heavier attention as they dominates the total loss

- Then we go back and find the “votes” $\alpha_m^*$ associated with the new classifier by minimizing the original weighted (exponential) loss
  \[
  \sum_{i=1}^{n} W_{i}^{m-1} \exp \left\{ -y_i \alpha_m h(x_i; \theta_m) \right\} 
  \]
Boosting

- We have basically derived a Boosting algorithm that sequentially adds new component classifiers, each trained on reweighted training examples
  - each component classifier is presented with a slightly different problem

- AdaBoost preliminaries:
  - we work with normalized weights $W_i$ on the training examples, initially uniform ($W_i = 1/n$)
  - the weight reflect the "degree of difficulty" of each datum on the latest classifier

The AdaBoost algorithm

- At the $k$th iteration we find (any) classifier $h(x; \theta_k^*)$ for which the weighted classification error:
  $$
  \varepsilon_k = 0.5 \left( \frac{1}{2} \sum_{i=1}^{n} W_i^{k-1} y_i h(x_i; \theta_k^*) \right)
  $$
  is better than change.
  - This is meant to be "easy" --- weak classifier

- Determine how many "votes" to assign to the new component classifier:
  $$
  \alpha_k = 0.5 \log((1 - \varepsilon_k) / \varepsilon_k)
  $$
  - stronger classifier gets more votes

- Update the weights on the training examples:
  $$
  W_i^k = W_i^{k-1} \exp\{-y_i \alpha_k h(x_i; \theta_k^*)\} \}
  $$
The AdaBoost algorithm cont’d

- The final classifier after \( m \) boosting iterations is given by the sign of
  \[
  \hat{h}(x) = \frac{\alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m)}{\alpha_1 + \ldots + \alpha_m}
  \]

- the votes here are normalized for convenience

---

AdaBoost: summary

- **Input:**
  - \( N \) examples \( S_N = \{ (x_1, y_1), \ldots, (x_N, y_N) \} \)
  - a weak base learner \( h = h(x, \theta) \)

- **Initialize:** equal example weights \( w_i = 1/N \) for all \( i = 1..N \)

- **Iterate for** \( t = 1..T \):
  1. train base learner according to weighted example set \( (w_t, x) \) and obtain hypothesis \( h_t = h(x, \theta) \)
  2. compute hypothesis error \( \varepsilon_t \)
  3. compute hypothesis weight \( \alpha_t \)
  4. update example weights for next iteration \( w_{t+1} \)

- **Output:** final hypothesis as a linear combination of \( h_t \)
AdaBoost: dataflow diagram

Boosting: examples
Boosting: example cont’d

Boosting: example cont’d
Base Learners

- Weak learners used in practice:
  - Decision stumps (axis parallel splits)
  - Decision trees (e.g. C4.5 by Quinlan 1996)
  - Multi-layer neural networks
  - Radial basis function networks

- Can base learners operate on weighted examples?
  - In many cases they can be modified to accept weights along with the examples
  - In general, we can sample the examples (with replacement) according to the distribution defined by the weights

Boosting performance

- The error rate of component classifier (the decision stumps) does not improve much (if at all) over time
- But both training and testing error improve over time!
- Even after the training error of the combined classifier goes to zero, boosting iterations can still improve the generalization error!!
Why it is working?

- You will need some learning theory (to be covered in the next two lectures) to understand this fully, but for now let's just go over some high level ideas
- Generalization Error:

  With high probability, Generalization error is less than:

  \[
  \Pr[H(x) \neq y] + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)
  \]

  As \( T \) goes up, our bound becomes worse,
  Boosting should overfit!

Experiments

*The Boosting Approach to Machine Learning*, by Robert E. Schapire
Training Margins

- When a vote is taken, the more predictors agreeing, the more confident you are in your prediction.

- Margin for example:

$$\text{margin}_i(x_i, y_i) = y_i \left[ \frac{\alpha_1 h(x_i; \theta_1) + \ldots + \alpha_m h(x_i; \theta_m)}{\alpha_1 + \ldots + \alpha_m} \right]$$

The margin lies in $[-1, 1]$ and is negative for all misclassified examples.

- Successive boosting iterations improve the majority vote or margin for the training examples.

More Experiments

The Boosting Approach to Machine Learning, by Robert E. Schapire
A Margin Bound

- For any $\gamma$, the generalization error is less than:

$$\Pr(\text{margin}_h(x, y) \leq \gamma) + O\left(\frac{d}{m\gamma^2}\right)$$


- It does not depend on $T$!!!

Summary

- Boosting takes a weak learner and converts it to a strong one
- Works by asymptotically minimizing the empirical error
- Effectively maximizes the margin of the combined hypothesis