Topics of This Lecture

- Recap: Nonlinear Support Vector Machines
- Ensembles of classifiers
  - Bagging
  - Bayesian Model Averaging
- AdaBoost
  - Intuition
  - Algorithm
  - Analysis
  - Extensions

Recap: Support Vector Machine (SVM)

- Basic idea
  - The SVM tries to find a classifier which maximizes the margin between pos. and neg. data points.
  - Up to now: consider linear classifiers

\[
\mathbf{w}^T \mathbf{x} + b = 0
\]

- Formulation as a convex optimization problem

\[
\arg \min_{\mathbf{w}, b} \frac{1}{2} \| \mathbf{w} \|^2
\]

under the constraints

\[
t_n (\mathbf{w}^T \mathbf{x}_n + b) \geq 1 \quad \forall n
\]

based on training data points \( \mathbf{x}_n \) and target values \( t_n \in \{-1, 1\} \)

Recap: Nonlinear SVMs

- General idea: The original input space can be mapped to some higher-dimensional feature space where the training set is separable:

\[
\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})
\]
Recap: The Kernel Trick

- Important observation
  - $\phi(x)$ only appears in the form of dot products $\phi(x)^T\phi(y)$:
    
    $$y(x) = w^T\phi(x) + b$$
    
    $$= \sum_{n=1}^{N} a_n t_n \phi(x_n)^T\phi(x) + b$$
  - Define a so-called kernel function $k(x,y) = \phi(x)^T\phi(y)$.
  - Now, in place of the dot product, use the kernel instead:
    
    $$\text{The kernel function implicitly maps the data to the higher-dimensional space (without having to compute } \phi(x) \text{ explicitly)!}$$

Recap: Nonlinear SVM – Dual Formulation

- SVM Dual: Maximize
  
  $$L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m)$$
  
  under the conditions
  
  $$0 \cdot a_n \cdot C$$
  
  $$\sum_{n=1}^{N} a_n t_n = 0$$
  
- Classify new data points using
  
  $$y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b$$

Recap: SVM Loss Function

- Traditional soft-margin formulation
  
  $$\min_{w \in \mathbb{R}^D, \xi \in \mathbb{R}^+} \frac{1}{2} ||w||^2 + C \sum_{n=1}^{N} \xi_n$$
  
  “Maximize the margin”
  
  subject to the constraints
  
  $$t_n y(x_n) \geq 1 - \xi_n$$
  
  “Most points should be on the correct side of the margin”
  
- Different way of looking at it
  
  We can reformulate the constraints into the objective function.
  
  $$\min_{w \in \mathbb{R}^D, \xi \in \mathbb{R}^+} \frac{1}{2} ||w||^2 + C \sum_{n=1}^{N} [1 - t_n y(x_n)]_+$$
  
  L2 regularizer “Hinge loss”

  where $[x]_+ := \max(0,x)$.

Recap: Hinge Loss Analysis

- “Hinge error” used in SVMs
  
  - Zero error for points outside the margin ($z_n > 1$) ⇒ sparsity
  
  - Linear penalty for misclassified points ($z_n < 1$) ⇒ robustness
  
  - Not differentiable around $z_n = 1$ ⇒ Cannot be optimized directly.

So Far…

- We’ve seen already a variety of different classifiers
  
  - $k$-NN
    
  - Bayes classifiers

  - Linear discriminants

  - SVMs

- Each of them has their strengths and weaknesses…
  
  - Can we improve performance by combining them?
Ensembles of Classifiers

- **Intuition**
  - Assume we have $K$ classifiers.
  - They are independent (i.e., their errors are uncorrelated).
  - Each of them has an error probability $p < 0.5$ on training data.
    - Why can we assume that $p$ won’t be larger than 0.5?
  - Then a simple majority vote of all classifiers should have a lower error than each individual classifier…

- **Constructing Ensembles**
  - **Bagging** = “Bootstrap aggregation” (Breiman 1996)
    - In each run of the training algorithm, randomly select $M$ samples with replacement from the full set of $N$ training data points.
    - If $M = N$, then on average, 63.2% of the training points will be represented. The rest are duplicates.
  - **Injecting randomness**
    - Many (iterative) learning algorithms need a random initialization (e.g. k-means, EM).
    - Perform multiple runs of the learning algorithm with different random initializations.

- **Bayesian Model Averaging**
  - **Model Averaging**
    - Suppose we have $H$ different models $h = 1, \ldots, H$ with prior probabilities $p(h)$.
    - Construct the marginal distribution over the data set
      $$ p(X) = \sum_{h=1}^{H} p(X|h)p(h) $$
  - **Interpretation**
    - Just one model is responsible for generating the entire data set.
    - The probability distribution over $h$ just reflects our uncertainty which model that is.
    - As the size of the data set increases, this uncertainty reduces, and $p(X|h)$ becomes focused on just one of the models.

- **Model Averaging: Expected Error**
  - Combine $M$ predictors $y_m(x)$ for target output $h(x)$.
    - E.g. each trained on a different bootstrap data set by bagging.
    - The committee prediction is given by
      $$ y_{COM}(x) = \frac{1}{M} \sum_{m=1}^{M} y_m(x) $$
  - The output can be written as the true value plus some error.
    $$ y(x) = h(x) + \epsilon(x) $$
  - Thus, the expected sum-of-squares error takes the form
    $$ E_x = \left( \sum_{m=1}^{M} (y_m(x) - h(x))^2 \right) = E_x \left[ \epsilon_m(x)^2 \right] $$
Model Averaging: Expected Error

- Average error of individual models
  \[ \mathbb{E}_{AV} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{x} [\epsilon_m(x)^2] \]

- Average error of committee
  \[ \mathbb{E}_{COM} = \mathbb{E}_{x} \left[ \frac{1}{M} \sum_{m=1}^{M} [y_m(x) - h(x)] \right] = \mathbb{E}_{x} \left[ \frac{1}{M} \sum_{m=1}^{M} \epsilon_m(x) \right] \]

- Assumptions
  - Errors have zero mean: \( \mathbb{E}_{x} [\epsilon_m(x)] = 0 \)
  - Errors are uncorrelated: \( \mathbb{E}_{x} [\epsilon_m(x) \epsilon_j(x)] = 0 \)

- Then:
  \[ \mathbb{E}_{COM} = \frac{1}{M} \mathbb{E}_{AV} \]

AdaBoost – “Adaptive Boosting”

- Main idea
  - Iteratively select an ensemble of component classifiers
  - After each iteration, reweight misclassified training examples.
    - Increase the chance of being selected in a sampled training set.
    - Or increase the misclassification cost when training on the full set.

- Components
  - \( h_m(x) \): “weak” or base classifier
    - Condition: <50% training error over any distribution
  - \( H(x) \): “strong” or final classifier

- AdaBoost:
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:
    \[ H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right) \]
**AdaBoost – Formalization**

- 2-class classification problem
  - Given: training set $X = \{x_1, ..., x_N\}$ with target values $T = \{t_1, ..., t_N\}, t_n \in \{-1, 1\}$.
  - Associated weights $W = \{w_1, ..., w_N\}$ for each training point.
- Basic steps
  - In each iteration, AdaBoost trains a new weak classifier $h_m(x)$ based on the current weighting coefficients $W(m)$.
  - We then adapt the weighting coefficients for each point
    - Increase $w_n$ if $x_n$ was misclassified by $h_m(x)$.
    - Decrease $w_n$ if $x_n$ was classified correctly by $h_m(x)$.
  - Make predictions using the final combined model

$$H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right)$$

**AdaBoost – Algorithm**

1. Initialization: Set $w(1)^n = \frac{1}{N}$ for $n = 1, ..., N$.
2. For $m = 1, ..., M$ iterations
   a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W(m)$ by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w(n)^m I(h_m(x) \neq t_n)$$

$\{ I : 1, \text{ if } A \text{ is true} \}
\{ 0, \text{ otherwise} \}$

b) Estimate the weighted error of this classifier on $X$:

$$\epsilon_m = \sum_{n=1}^{N} w(n)^m I(h_m(x) \neq t_n)$$

$$\sum_{n=1}^{N} w(n)^m$$

c) Calculate a weighting coefficient for $h_m(x)$:

$$\alpha_m = ?$$

d) Update the weighting coefficients:

$$w(n)^{m+1} = ?$$

**AdaBoost – Historical Development**

- Originally motivated by Statistical Learning Theory
  - AdaBoost was introduced in 1996 by Freund & Schapire.
  - It was empirically observed that AdaBoost often tends not to overfit. (Breiman 96, Cortes & Drucker 97, etc.)
  - As a result, the margin theory (Schapire et al. 98) developed, which is based on loose generalization bounds.
    - Note: margin for boosting is not the same as margin for SVM.
    - A bit like retrotitling the theory...
  - However, those bounds are too loose to be of practical value.
- Different explanation (Friedman, Hastie, and Tibshirani, 2000)
  - Interpretation as sequential minimization of an exponential error function ("Forward Stagewise Additive Modeling").
  - Explains why boosting works well.
  - Improvements possible by altering the error function.

**AdaBoost – Minimizing Exponential Error**

- Sequential Minimization
  - Suppose that the base classifiers $h_1(x), ..., h_m(x)$ and their coefficients $\alpha_1, ..., \alpha_m$ are fixed.
  - Only minimize with respect to $\alpha_m$ and $h_m(x)$.

$$E = \sum_{n=1}^{N} \exp \{-t_n f_m(x_n)\} \quad \text{with} \quad f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l h_l(x)$$

$$= \sum_{n=1}^{N} \exp \left\{ -t_n \sum_{l=1}^{m} \alpha_l h_l(x) \right\}$$

$$= \sum_{n=1}^{N} \exp \left\{ -t_n \alpha_m h_m(x_n) \right\}$$

$$= \sum_{n=1}^{N} w(n)^m \exp \left\{ -\frac{1}{2} \alpha_m h_m(x_n) \right\}$$

**AdaBoost – Minimizing Exponential Error**

$$E = \sum_{n=1}^{N} w(n)^m \exp \left\{ -\frac{1}{2} \alpha_m h_m(x_n) \right\}$$

- Observation:
  - Correctly classified points: $t_n h_m(x_n) = +1 \Rightarrow \text{collect in } T_m$
  - Misclassified points: $t_n h_m(x_n) = -1 \Rightarrow \text{collect in } F_m$
- Rewrite the error function as

$$E = e^{-\alpha_m/2} \sum_{n \in T_m} w(n)^m + e^{\alpha_m/2} \sum_{n \in F_m} w(n)^m$$

$$= \left( e^{\alpha_m/2} \right)^N$$

$$\sum_{n=1}^{N} w(n)^m I(h_m(x_n) \neq t_n)$$
AdaBoost – Minimizing Exponential Error

$$E = \frac{1}{2} \sum_{n=1}^{N} w^{(m)}_n \exp \left\{ -\frac{1}{2} \alpha_m h_m(x_n) \right\}$$

- Observation:
  - Correctly classified points: $t_h_m(x_n) = +1 \Rightarrow$ collect in $T_m$
  - Misclassified points: $t_h_m(x_n) = -1 \Rightarrow$ collect in $F_m$
- Rewrite the error function as
  $$E = \sum_{n \in F_m} w_n^{(m)} + \sum_{n \in T_m} e^{-\alpha_m/2} w_n^{(m)}$$

$$= \sum_{n \in F_m} e^{-\alpha_m} w_n^{(m)} + \sum_{n \in T_m} e^{-\alpha_m/2} w_n^{(m)}$$

$$\Rightarrow$$ Update for the $\alpha$ coefficients:

$$\alpha_m = \ln \left\{ \frac{1 - e_m}{e_m} \right\}$$

AdaBoost – Final Algorithm

1. Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$ iterations:
   a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W^{(m)}$ by minimizing the weighted error function $J_m$:

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n)$$

b) Estimate the weighted error of this classifier on $X$:

$$e_m = \frac{1}{\sum_{n=1}^{N} w_n^{(m)}} \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n)$$

c) Calculate a weighting coefficient for $h_m(x)$:

$$\alpha_m = \ln \left\{ \frac{1 - e_m}{e_m} \right\}$$

d) Update the weighting coefficients:

$$w_{n}^{(m+1)} = w_{n}^{(m)} \exp \{ \alpha_m I(h_m(x_n) \neq t_n) \}$$

AdaBoost – Analysis

- Result of this derivation:
  - We now know that AdaBoost minimizes an exponential error function in a sequential fashion.
  - This allows us to analyze AdaBoost’s behavior in more detail.
  - In particular, we can see how robust it is to outlier data points.
Recap: Error Functions

\[ t_n \in \{-1, 1\} \]

\[ E(z_n) \]

\[ z_n = t_n y(x_n) \]

- Ideal misclassification error function (black)
  - This is what we want to approximate,
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.
  - We cannot minimize it by gradient descent.

Squared error used in Least-Squares Classification

- Very popular, leads to closed-form solutions.
- However, sensitive to outliers due to squared penalty.
- Penalizes "too correct" data points
  - Generally does not lead to good classifiers.

"Hinge error" used in SVMs

- Zero error for points outside the margin (\( z_n > 1 \)) ⇒ sparsity
- Linear penalty for misclassified points (\( z_n < 1 \)) ⇒ robustness
- Not differentiable around \( z_n = 1 \) ⇒ Cannot be optimized directly.

Exponential error used in AdaBoost

- No penalty for too correct data points, fast convergence.
- Disadvantage: exponential penalty for large negative values!
  - Less robust to outliers or misclassified data points!
- Similar to exponential error for \( z > 0 \).
- Only grows linearly with large negative values of \( z \).
  - Make AdaBoost more robust by switching to this error function.
  - "GentleBoost"

"Cross-entropy error" used in Logistic Regression
Summary: AdaBoost

- Properties
  - Simple combination of multiple classifiers.
  - Easy to implement.
  - Can be used with many different types of classifiers.
    - None of them needs to be too good on its own.
    - In fact, they only have to be slightly better than chance.
  - Commonly used in many areas.
  - Empirically good generalization capabilities.

- Limitations
  - Original AdaBoost sensitive to misclassified training data points.
    - Because of exponential error function.
    - Improvement by GentleBoost
  - Single-class classifier
    - Multiclass extensions available

References and Further Reading

- More information on Classifier Combination and Boosting can be found in Chapters 14.1-14.3 of Bishop’s book.

Christopher M. Bishop
Pattern Recognition and Machine Learning
Springer, 2006

- A more in-depth discussion of the statistical interpretation of AdaBoost is available in the following paper: