

## Lecture 8: SVM &amp; Boosting &amp; Clustering

Lecturer: Liwei Wang

Scribe: Group 7

**Disclaimer:** These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.

## 8.1 Recap: SVM for Linear Separable Case

When data is linear separable, we can find a hyperplane that separates the positive and negative examples with maximum margin.

$$(P) \quad \begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1, \quad i = 1, \dots, n \end{aligned}$$

KKT conditions tell us that for all  $i = 1, \dots, n$ ,  $\lambda_i^*(y_i(w^{*\top} x_i + b^*) - 1) = 0$ , so either  $\lambda_i^* = 0$  or  $y_i(w^{*\top} x_i + b^*) - 1 = 0$ . Then we have:

$$w^* = \sum_{i=1}^n \lambda_i^* y_i x_i = \sum_{i \in \text{SV}} \lambda_i^* y_i x_i$$

SV (Support Vector) =  $\{(x_i, y_i) : (x_i, y_i) \text{ is closest to the hyperplane}\}$

The duality problem of (P) is:

$$(D) \quad \begin{aligned} \min_{\lambda} \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j x_i^\top x_j - \sum_{i=1}^n \lambda_i \\ \text{s.t.} \quad & \lambda_i \geq 0, \quad i = 1, \dots, n \\ & \sum_{i=1}^n \lambda_i y_i = 0 \end{aligned}$$

Let  $G_{ij} = y_i y_j x_i^\top x_j$ , then  $G$  is the Gram matrix. It is known that any Gram matrix is positive semi-definite. Indeed, for any  $\lambda = (\lambda_1, \dots, \lambda_n)^\top \in \mathbb{R}^n$ :

$$\lambda^\top G \lambda = \sum_{i,j} \lambda_i \lambda_j (y_i x_i)^\top (y_j x_j) = \left\| \sum_{i=1}^n \lambda_i y_i x_i \right\|^2 \geq 0.$$

Hence,  $G$  is positive semi-definite, and the quadratic form  $\sum_{i,j} \lambda_i \lambda_j y_i y_j x_i^\top x_j$  is nonnegative for all  $\lambda$ .

## 8.2 SVM for Linear Inseparable Case

When data is not linear separable, we can introduce slack variables  $\varepsilon_i \geq 0$  to allow some misclassification. The optimization problem becomes:

$$(P) \quad \begin{aligned} \min_{w, b, \varepsilon} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \varepsilon_i \quad (C \text{ is tradeoff const}) \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1 - \varepsilon_i, \quad i = 1, \dots, n \\ & \varepsilon_i \geq 0, \quad i = 1, \dots, n \end{aligned}$$

Choosing other minimizing objective such as  $\sum_{i=1}^n \varepsilon_i^2$  or  $\sum_{i=1}^n \mathbb{I}[\varepsilon_i > 0]$  would also make sense. However, the indicator function is not continuous, making it hard to optimize.

The Lagrangian function is:

$$\mathcal{L}(w, b, \varepsilon, \lambda, \mu) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \varepsilon_i - \sum_{i=1}^n \lambda_i [y_i(w^\top x_i + b) - 1 + \varepsilon_i] - \sum_{i=1}^n \mu_i \varepsilon_i$$

KKT conditions tell us that:

$$\begin{aligned} \mu_i, \lambda_i &\geq 0 \\ \frac{\partial \mathcal{L}}{\partial w} = 0 &\Rightarrow w = \sum_{i=1}^n \lambda_i y_i x_i \\ \frac{\partial \mathcal{L}}{\partial b} = 0 &\Rightarrow \sum_{i=1}^n \lambda_i y_i = 0 \\ \frac{\partial \mathcal{L}}{\partial \varepsilon_i} = 0 &\Rightarrow C - \lambda_i - \mu_i = 0 \Rightarrow \lambda_i \leq C \end{aligned}$$

Therefore, we can derive the duality problem of (P):

$$(D) \quad \begin{aligned} \min_{\lambda} \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j y_i y_j x_i^\top x_j - \sum_{i=1}^n \lambda_i \\ \text{s.t.} \quad & 0 \leq \lambda_i \leq C, \quad i = 1, \dots, n \\ & \sum_{i=1}^n \lambda_i y_i = 0 \end{aligned}$$

The only difference between the duality problems of linear separable and inseparable cases is that now we have an upper bound  $C$  for  $\lambda_i$ .

Another way to look at the linear inseparable case is through **hinge loss**, because (P) can be rewritten as:

$$(P') \quad \begin{aligned} \min_{w, b} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n [1 - y_i(w^\top x_i + b)]_+ \\ \text{where} \quad & [u]_+ := \max(0, u) \end{aligned}$$

## 8.3 Boosting (Meta Learning)

The idea of boosting is to combine multiple weak classifiers to form a strong classifier:

1. Assume we already have a “base learning algorithm”.
2. Boosting tries to learn an ensemble of “base classifiers” and “combine” them to form a strong classifier.

While the natural approach “Bagging” effectively creates an ensemble by randomly sampling data (bootstrapping) to train multiple base learners, the following approach aims to promote diversity among them. By designing learners to be complementary rather than merely independent, it seeks to achieve superior ensemble performance.

### 8.3.1 AdaBoost

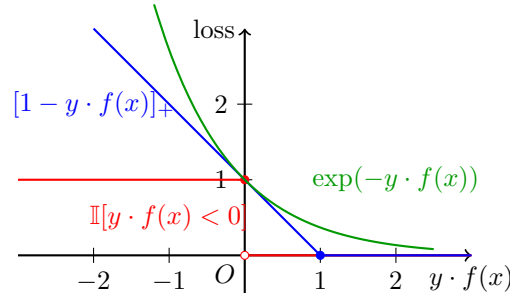
---

#### Algorithm 1 AdaBoost Algorithm

---

- 1: **Input:** training set  $\mathcal{S} = \{(x_1, y_1), \dots, (x_n, y_n)\}$  and a base learning algorithm  $\mathcal{A}$ .
  - 2: **Init:** set weights  $D_1(i) \leftarrow 1/n, i = 1, \dots, n$ .
  - 3: **for**  $i \leftarrow 1$  to  $n$  **do**
  - 4:   Using  $\mathcal{A}$  to train a base classifier  $h_t(\cdot)$  w.r.t.  $D_t$ . ( $h_t(x) \in \{\pm 1\}$ )
  - 5:   Compute the weighted error:  $\varepsilon_t \leftarrow \sum_{i=1}^n D_t(i) \mathbb{I}[h_t(x_i) \neq y_i]$
  - 6:    $\gamma_t \leftarrow 1 - 2\varepsilon_t \in [-1, 1]$
  - 7:    $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1 + \gamma_t}{1 - \gamma_t}$
  - 8:   Multiplicative weights update:  $D_{t+1}(i) \leftarrow \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$  (\*)
  - where  $Z_t$  is the normalization factor  $Z_t = \sum_{i=1}^n D_t(i) \exp(-\alpha_t y_i h_t(x_i))$ .
  - 9: **end for**
  - 10: **Output:**  $\text{sgn}(f(x))$ , where  $f(x) = \sum_{t=1}^T \alpha_t h_t(x)$ .
- 

If  $\alpha_t > 0$ , i.e.  $h_t(\cdot)$  is stronger than random guess, then when  $h_t(x_i) = y_i$ ,  $D_{t+1}(i) < D_t(i)$ ; when  $h_t(x_i) \neq y_i$ ,  $D_{t+1}(i) > D_t(i)$ . So the misclassified examples will get higher weights. If  $\alpha_t < 0$ , i.e.  $h_t(\cdot)$  is weaker than random guess, we can simply flip its output to make it stronger than random guess. Therefore the misclassified examples are actually correctly-classified examples in the flipped version and will get lower weights.  $\alpha_t = 0$  is the worst case, which means  $h_t(\cdot)$  is just random guess, and it contributes nothing to the final classifier.



The reason why AdaBoost works well is that it actually tries to minimize the **exponential loss**. From above figure, we can see that **exponential loss**  $\exp(-y \cdot f(x))$  upper bounds the **hinge loss**  $[1 - y \cdot f(x)]_+$ , and hinge loss upper bounds the **0-1 loss**  $\mathbb{I}[y \cdot f(x) < 0]$ . Hinge loss and exponential loss are both **convex**, and exponential loss is more **smooth** than hinge loss.

We will prove some theoretical results about AdaBoost.

**Proposition 8.1.**

$$\alpha_t = \arg \min_{\alpha} \sum_{i=1}^n D_t(i) \exp(-\alpha y_i h_t(x_i))$$

*Proof.* Firstly, we can rewrite the objective function:

$$\begin{aligned} Z_t &= \sum_{i=1}^n D_t(i) \exp(-\alpha y_i h_t(x_i)) \\ &= \sum_{i: h_t(x_i)=y_i} D_t(i) \exp(-\alpha) + \sum_{i: h_t(x_i) \neq y_i} D_t(i) \exp(\alpha) \\ &= \exp(-\alpha)(1 - \varepsilon_t) + \exp(\alpha)\varepsilon_t \\ &= \exp(-\alpha) + \varepsilon_t(\exp(\alpha) - \exp(-\alpha)) \end{aligned}$$

Then we set the derivative of  $Z_t$  w.r.t.  $\alpha$  to zero:

$$\begin{aligned} \frac{\partial Z_t}{\partial \alpha} &= -\exp(-\alpha) + \varepsilon_t(\exp(\alpha) - \exp(-\alpha)) = 0 \\ &\Rightarrow \varepsilon_t \exp(\alpha) = (1 - \varepsilon_t) \exp(-\alpha) \\ &\Rightarrow \alpha = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t} = \frac{1}{2} \ln \frac{1 + \gamma_t}{1 - \gamma_t} \end{aligned}$$

By the second derivative test, we can verify that this is indeed a minimum. □

**Proposition 8.2.**

$$\prod_{i=1}^T Z_t = \frac{1}{n} \sum_{i=1}^n \exp \left( -y_i \sum_{t=1}^T \alpha_t h_t(x_i) \right) = \frac{1}{n} \sum_{i=1}^n \exp(-y_i f(x_i))$$

*Proof.* By taking telescoping product on both sides of (\*), we have:

$$D_{T+1}(i) = \frac{D_1(i) \exp \left( -y_i \sum_{t=1}^T \alpha_t h_t(x_i) \right)}{\prod_{t=1}^T Z_t} = \frac{\frac{1}{n} \exp \left( -y_i \sum_{t=1}^T \alpha_t h_t(x_i) \right)}{\prod_{t=1}^T Z_t}$$

Since  $\sum_{i=1}^n D_{T+1}(i) = 1$ , by summing both sides over  $i$ , we have:

$$\prod_{t=1}^T Z_t = \frac{1}{n} \sum_{i=1}^n \exp \left( -y_i \sum_{t=1}^T \alpha_t h_t(x_i) \right)$$

□

**Proposition 8.3.** Assume in AdaBoost algorithm,  $\gamma_t \geq \gamma > 0$  for all  $t \in [T]$  (a.k.a.  $\varepsilon_t \leq \varepsilon < 1/2$ ). Then

$$\mathbb{P}_S(y_i f(x_i) < 0) = \frac{1}{n} \sum_{i=1}^n \mathbb{I}[y_i f(x_i) < 0] \leq \frac{1}{n} \sum_{i=1}^n \exp(-y_i f(x_i)) \leq (1 - \gamma^2)^{T/2}$$

*Proof.* Note that exponential loss upper bounds the training error, so we just need to prove the second inequality.

From the previous proposition, we have  $\frac{1}{n} \sum_{i=1}^n \exp(-y_i f(x_i)) = \prod_{t=1}^T Z_t$ . Since we can rewrite  $Z_t$  as:

$$\begin{aligned} Z_t &= \exp(-\alpha_t) + \varepsilon_t(\exp(\alpha_t) - \exp(-\alpha_t)) \\ &= 2\sqrt{\varepsilon_t(1 - \varepsilon_t)} \\ &= \sqrt{1 - \gamma_t^2} \leq \sqrt{1 - \gamma^2} \end{aligned}$$

Thus, we have:

$$\mathbb{P}_S(y_i f(x_i) < 0) = \prod_{t=1}^T Z_t \leq (1 - \gamma^2)^{T/2}$$

□

**Corollary 8.4.** Since training error is a discrete variable in  $\{0, 1/n, 2/n, \dots, 1\}$ , if we want to achieve zero training error, it suffices to let  $T = O(\log n)$ .

**Proposition 8.5.** The error of  $h_t(\cdot)$  w.r.t. distribution  $D_{t+1}$  is

$$\sum_{i=1}^n D_{t+1}(i) \mathbb{I}[h_t(x_i) \neq y_i] = \frac{1}{2}$$

*Proof.*

$$\begin{aligned} \sum_{i=1}^n D_{t+1}(i) \mathbb{I}[h_t(x_i) \neq y_i] &= \sum_{i: h_t(x_i) \neq y_i} \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \\ &= \sum_{i: h_t(x_i) \neq y_i} \frac{D_t(i) \exp(\alpha_t)}{Z_t} = \frac{\exp(\alpha_t)}{Z_t} \varepsilon_t \\ &= \frac{\exp(\alpha_t)}{\exp(-\alpha_t) + \varepsilon_t(\exp(\alpha_t) - \exp(-\alpha_t))} \varepsilon_t \\ &= \frac{\varepsilon_t}{(1 - \varepsilon_t) \exp(-2\alpha_t) + \varepsilon_t} \\ &= \frac{\varepsilon_t}{(1 - \varepsilon_t) \frac{\varepsilon_t}{1 - \varepsilon_t} + \varepsilon_t} = \frac{1}{2} \end{aligned}$$

□

### 8.3.2 Other Boosting Algorithms

Besides AdaBoost, there are many other boosting algorithms, such as Gradient Boosting, XGBoost, LightGBM, etc. They differ in the way of combining base classifiers and updating weights.

## 8.4 Clustering

### 8.4.1 K-Means

Given  $n$  data points  $\{x_1, x_2, \dots, x_n\}$  in  $\mathbb{R}^d$ , the goal of clustering is to partition them into  $k$  clusters  $\{C_1, C_2, \dots, C_k\}$  such that the intra-cluster similarity is high and the inter-cluster similarity is low.

---

**Algorithm 2** K-Means Algorithm
 

---

- 1: **Input:** data points  $\{x_1, x_2, \dots, x_n\}$ , number of clusters  $k$ .
- 2: **Init:** Randomly select  $k$  initial centroids  $\{c_1, c_2, \dots, c_k\}$ .
- 3: **repeat**
- 4:   For each  $x_i$ , find a cluster center  $c_{(i)}$  closest to  $x_i$  and assign  $x_i$  to the corresponding cluster  $C_{(i)}$ .
- 5:   Reallocate cluster center according to the partition

$$c_i \leftarrow \frac{1}{n_i} \sum_{x_j \in C_i} x_j$$

where  $C_i$  means the  $i$ -th cluster and  $n_i = |C_i|$ .

- 6: **until** convergence
- 

Since reassigning  $x_i$  to the closest cluster and recalculating the centers all decrease the sum of the distances, K-Means can monotonically decrease the objective function  $\sum_{i=1}^n \|x_i - c_{(i)}\|^2$ . However, K-Means doesn't have theoretical guarantee.

### 8.4.2 K-Means++

An optimized version of K-Means (we call it K-Means++) just by changing the initialization step can achieve  $O(\log k)$  approximation ratio in expectation.

Intuitively, we want to select initial centroids that are far away from each other:

$$\begin{aligned} c_1 &\sim \text{Uniform}(\{x_1, x_2, \dots, x_n\}) \\ c_2 &: p(x) \propto \|x - c_1\|^2 \\ c_3 &: p(x) \propto \min(\|x - c_1\|^2, \|x - c_2\|^2) \end{aligned}$$

Using this approach would ensure that if there exists an outlier in a very large dataset, then the probability of choosing it as the next centroid would be very low (while a deterministic approach would choose it with probability 1).

---

**Algorithm 3** K-Means++ Initialization
 

---

- 1: **Input:** data points  $\{x_1, x_2, \dots, x_n\}$ , number of clusters  $k$ .
  - 2: Randomly select the first centroid  $c_1$  from the data points.
  - 3: **for**  $i \leftarrow 2$  to  $k$  **do**
  - 4:   For each data point  $x_j$ , compute its distance  $D(x_j)$  to the nearest centroid among  $\{c_1, c_2, \dots, c_{i-1}\}$ .
  - 5:   Select the next centroid  $c_i$  from the data points with probability proportional to  $D(x_j)^2$ .
  - 6: **end for**
-

**Proposition 8.6.** *K-Means++ can achieve:*

$$\frac{\mathbb{E}[\phi_{K\text{-Means}++}]}{OPT} \leq 8(\ln k + 2)$$

Proof of this proposition is omitted here. Please refer to [AV07] for details.

## References

- [AV07] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.