Machine Learning **Spring 2021**

Lecture 6: Boosting

Lecturer: Liwei Wang *Scribe: Qin Han, Zixuan Huo, Zijian Lan, Zezhou Wang, Zhengbo Xu*

6.1 Soft Margin SVM

Under many circumstances, training data is not linear separable. We can deal with this problem by allowing the SVM makes mistakes on some of the data. Specifically, we introduce *Soft-Margin SVM*:

$$\begin{aligned} (P) \quad \min_{w,b,\xi} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \\ s.t. \quad & y_i(w^T x_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, \forall 1 \leq i \leq n \end{aligned} \tag{6.1}$$

Now we try to gain a deeper understanding of soft margin SVM. Denote $w^T x_1 + b$ as $f(x_i)$.

Obviously, (P) is equivalent to the following problem:

$$\min_{w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}[1 - y_i(w^T x_i + b)]_+$$

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

Before further discussion, we introduce some important loss functions. We know that

$$y_i f(x_i) > 0 \Leftrightarrow \text{the classification is correct}$$

So we can define:

Definition 1 *0-1 loss*

$$L_{0-1} = \begin{cases} 1, & if \quad y_i f(x_i) > 0 \\ 0, & if \quad y_i f(x_i) < 0 \end{cases}$$

Definition 2 *Hinge loss*

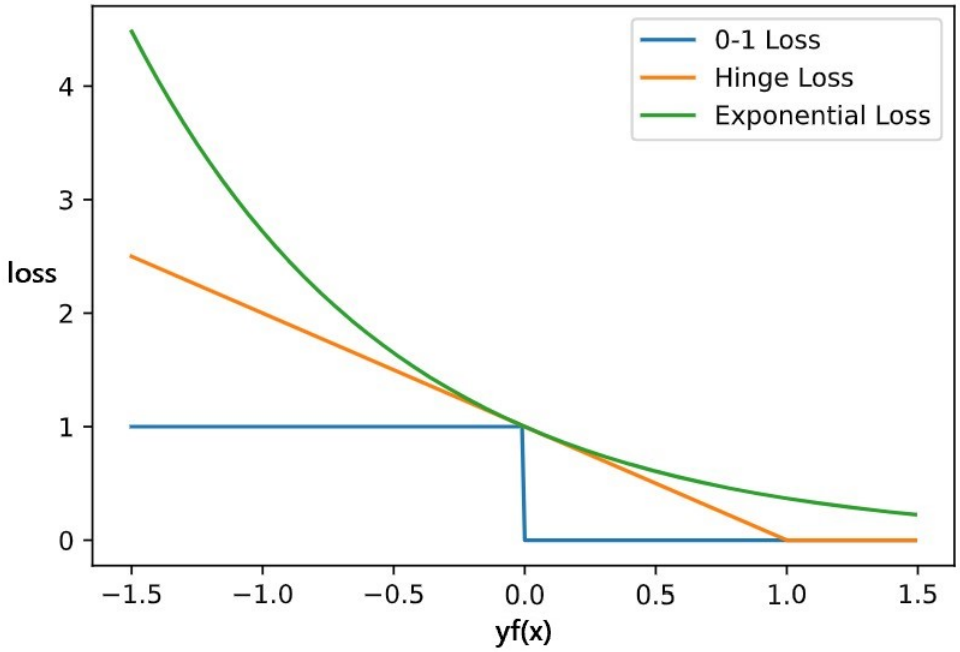
$$L_{hinge} = \begin{cases} 1 - y_i f(x_i), & if \quad y_i f(x_i) \leq 1 \\ 0, & if \quad y_i f(x_i) > 1 \end{cases}$$

Definition 3 *Exponential loss*

$$L_{exp} = e^{-y_i f(x_i)}$$

There are several important properties:

1. Hinge loss and exponential loss are both upper bounds for the 0-1 loss, which means when we minimize these two losses, 0-1 loss can be bounded.
2. Hinge loss and exponential loss are convex and continuous.
3. Once a data is classified correctly, its 0-1 loss reaches zero and no longer changes, causing the learning process to stop. This means $yf(x)$ is near zero. Meanwhile, the hinge loss will push $yf(x)$ to be above 1 and the exponential loss will push $yf(x)$ to infinity, which greatly improve generalization and give the model much robustness.



Back to our problem: $\min_{w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}[1 - y_i(w^T x_i + b)]_+$.

We can see $C\sum_{i=1}^{n}[1 - y_i(w^T x_i + b)]_+$ is the hinge loss of the classifier. The other term $\frac{1}{2}||w||^2$ is the $L2$ norm of the parameter w, so this is exactly $L2$ regularization. And now we have another perspective towards the *Soft-Margin SVM*: **Minimize Surrogate Loss function + Regularization**.

6.2 Boosting

Last week we have given the algorithm of AdaBoost.

Some propositions about the algorithm can help to understand why it performs well in practice.

Proposition 6.1 *Let* $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$, *we have:*

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

Where Z_t is normalization factor.

Proof:

$$\begin{aligned} Z_T &= \sum_{i=1}^{n} D_T(i) exp(-y_i \alpha_T h_T(x_i)) \\ &= \sum_{i=1}^{n} \frac{D_{T-1}(i) exp(-y_i \alpha_{T-1} h_{T-1}(x_i))}{Z_{T-1}} exp(-y_i \alpha_T h_T(x_i)) \\ \implies Z_{T-1} Z_T &= \sum_{i=1}^{n} D_{T-1}(i) exp\left(-y_i \alpha_{T-1} h_{T-1}(x_i) - y_i \alpha_T h_T(x_i)\right) \end{aligned}$$

Likewise, we can finally get

$$\begin{aligned} \prod_{t=1}^{T} Z_T &= \sum_{i=1}^{n} D_1(i) exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)\right) \\ &= \frac{1}{n}\sum_{i=1}^{n} exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)\right) \end{aligned}$$

■

Proposition 6.2 *For α_t and Z_t in the algorithm, $t \in [T]$, we have:*

$$\alpha_t = arg\min_{\alpha} Z_t$$

Proof:

$$\min_{\alpha} Z_t = \min_{\alpha} \sum_{i=1}^{n} D_t(i)(e^{-y_i h_t(x_i)})^{\alpha}$$

According to the definition of classifier, $y_i h_t(x_i) = \begin{cases} 1 & y_i = h_t(x_i) \\ -1 & y_i \neq h_t(x_i) \end{cases}$

So we have

$$\sum_{i=1}^{n} D_t(i)(e^{-y_i h_t(x_i)})^{\alpha} = \epsilon_t e^{\alpha} + (1-\epsilon_t)e^{-\alpha} \geq 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

The equality holds if and only if

$$\epsilon_t e^{\alpha} = (1-\epsilon_t)e^{-\alpha} \iff \alpha = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t} = \frac{1}{2} ln \frac{1+\gamma_t}{1-\gamma_t}$$

So the proposition is proved, and we greedily assign values to α_t and Z_t as follows in our AdaBoost algorithm:

$$\alpha_t = \frac{1}{2} ln \frac{1+\gamma_t}{1-\gamma_t}$$

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

■

In Proposition 6.2, we notice that the selection of α_t is to minimize Z_t. Furthermore, Proposition 6.1 shows that the product of Z_t reflects the exponential loss $e^{-yf(x)}$ of classifier.

Given that the exponential loss is upper bound of the 0-1 loss, if we minimize the exponential loss, then the 0-1 loss decreases accordingly, which means the classifier may work better on the training set.

Then we consider the connections between the updated classifier and the original classifier.

Proposition 6.3

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

Proof:

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

The selection of α_t is to minimize z_t and so we have $\epsilon_t \ exp(\alpha_t) = (1-\epsilon_t) \ exp(-\alpha_t)$, thus

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

■

In Proposition 6.3, we notice that the performance of the updated classifier is as same as a random classifier on the original distribution, so we infer that the optimization directions of different classifiers are orthogonal.

Beyond the intuition, the error rate of the classifier and the speed of algorithm can be estimated.

Proposition 6.4 *Assume $\gamma_t \geq \gamma > 0$, for $t \in [T]$, then the following inequality holds:*

$$P_s(yf(x) \leq 0) \leq (1-\gamma^2)^{\frac{T}{2}}$$

Proof: We have known exponential loss $e^{-yf(x)}$ upper bounds 0-1 loss, so

$$\begin{aligned} P_s(yf(x) \leq 0) &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[y_i f(x_i) \leq 0] \\ &\leq \frac{1}{n}\sum_{i=1}^{n} exp(-y_i f(x_i)) \\ &= \prod_{t=1}^{T} z_t = \prod_{i=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \end{aligned}$$

From the constraint $\gamma_t \geq \gamma > 0$, we can infer that $0 \leq \epsilon_t \leq \frac{1-\gamma}{2} < \frac{1}{2}$, so

$$\begin{aligned} 2\sqrt{\epsilon_t(1-\epsilon_t)} &\leq 2\sqrt{\frac{1-\gamma}{2}(1-\frac{1-\gamma}{2})} \\ &= \sqrt{1-\gamma^2} \end{aligned}$$

Then we have

$$\begin{aligned} P_s(yf(x) \leq 0) &\leq \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \\ &\leq \prod_{t=1}^{T} \sqrt{1-\gamma^2} \\ &= (1-\gamma^2)^{\frac{T}{2}} \end{aligned}$$

■

Notice that the minimum positive value of $P_s(yf(x) \leq 0)$ is $\frac{1}{n}$. So if $(1-\gamma^2)^{\frac{T}{2}} < \frac{1}{n}$, we have $P_s(yf(x) \leq 0) = 0$, which means the classifier works totally correct on training set $\mathcal{S}$.

To reach that, we only need $O(log\ n)$ rounds, namely $T = O(log\ n)$, in the case of $\gamma > 0$.

We have discussed in the Proposition 6.3 that the choice of α indicates that each time h_t learns some orthogonal features. By normalizing α, learned f can be seen as the distance of these features to a hyperplane, which gives a new perspective of AdaBoost that when minimizing exponential loss it maximizes margin like SVM does.

Proposition 6.5 *Suppose (x,y) is a sample from sample space, denote α = (α_1,..., α_T) (with $\sum_{i=1}^{T} \alpha_i = 1$ and $\alpha_i > 0$) as normal vector of a hyperplane, and h(x) = ($h_1(x)$,...,$h_T(x)$) $\in R^T$ as a data point, then y f(x) gives a kind of distance from h(x) to the hyperplane $\{\xi : \alpha\xi = 0\}$.*

Hint: L_∞ distance.

6.3 Term Project

6.3.1 Project I: Teaching Dimension of Neural Network

1. **Without-Teaching Model**: Given a data space X, a hypothesis space $\mathcal{F}$. We can find the VC-Dimension of $\mathcal{F}$.

2. **Teaching Model**: Given a data space X, a hypothesis space $\mathcal{F}$. Teacher chooses classifier $f \in \mathcal{F}$ and test on $\Omega \subseteq X$. Learner can specify f by Ω.

There are some formal definitions for a hypothesis space C and it's data space X.

Definition 4 *Teaching Dimension of a classifier c in C*

$$TD(c, C) = \min_{b \in B_c} |b|$$

where

$$B_c = \{b \subseteq X | \forall c' \in C, c' = c \text{ or } c'|_b \neq c|_b\}$$

Definition 5 *Worst-case Teaching Dimension*

$$TD(C) = \max_{c \in C} TD(c, C)$$

Definition 6 *Best-case Teaching Dimension*

$$TD_{min}(C) = \min_{c \in C} TD(c, C)$$

Definition 7 *Recursive Teaching Dimension*

$$RTD(C) = \max_{0 \leq t \leq T} TD_{min}(C_t)$$

where

$$C_0 = C, \quad C_{t+1} = C_t - \{c \in C_t | TD(c, C_t) = TD_{min}(C_t)\}$$

and

$$T = \arg\min_t C_t = \emptyset$$

Assignment: Try to calculate teaching dimension of a deep neural network.

References

[1] Quadratic Upper Bound for Recursive Teaching Dimension of Finite VC Classes

6.3.2 Project II: Certified Robustness

Adversarial examples are inputs to machine learning models designed to intentionally fool them or to cause mispredictions. One of the threats of adversaries can be modeled as Gradient access.

So far, some countermeasures have been proposed. One of the popular methods is adversarial training, which repeatedly generates adversarial samples for model training. However, its shortcomings are obvious. It can only protect against specific attacks, and it also lacks theoretical guarantees.

Assignment: Design an algorithm to reach the state of the art on some data set, for example, MINIST, FASHION-MINIST, and CIFAR.

References

[1] Towards Certifying l_∞ Robustness using Neural Networks with l_∞-dist Neurons

[2] https://www.microsoft.com/en-us/research/blog/adversarial-robustness-as-a-prior-for-better-trans

6.4 Bagging (Booststrap Aggregating)

6.4.1 Algorithm

- **Step 1:** Given a standard training set $\mathcal{S} = \{x_1, x_2, ..., x_n\}$.
- **Step 2:** Generate a new training set $\mathcal{S}_i = \{x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}\}$ by drawing with replacement from $\mathcal{S}$; train the model with $\mathcal{S}_i$ to obtain a new classifier h_i.
- **Step 3:** Repeat **Step 2** N times. Combine the classifiers by mean aggregator: $\frac{1}{N}\sum_{i=1}^{N} h_i(x)$.

We can easily notice that Booststrap Aggregating is more effective in the case that the base classifier is unstable, which means performance varies greatly with data.

References

[1] MOHRI.M, ROSTAMIZADEH.A and TALWALKAR.A, Foundations of Machine Learning, MIT Press (2012)

[2] ZHIHUA ZHOU, Machine Learning, Tsinghua University Press (2016)

[3] https://en.wikipedia.org/wiki/Hinge_loss

[4] https://en.wikipedia.org/wiki/AdaBoost

[5] https://en.wikipedia.org/wiki/Bootstrap_aggregating

[6] https://en.wikipedia.org/wiki/Random_forest