Haiyang Wang
November 28, 2020
hw1927@nyu.edu

Foundations of Machine Learning
Homework 3

Acknowledgements and other stuff

I was about to read some papers for details of logitboost, yet I find them quite unhelpful to me. So, no Acknowledgements.

1 Kernel Methods

1.1 Graph Kernel

If there are more than 1 edges from vertex p to vertex q, we can merge the edges into one with weights of sum of the merged edges. And this will not effect the computation of the kernel. Thus, with out loss of generality, we can assume there is at most one edge between 2 vertices.

Now we can describe an edge e by its 2 ends (p, q). And we use the notation $w[e] = w_{p,q} = w_{q,p}$, where the last equation uses the undirectedness of the graph. More generally, if there is no edge between vertices p, q we let $w_{p,q} = 0$. This gives us a $|\mathcal{V}| \times |\mathcal{V}|$ matrix $W = (w_{p,q})$ The Kernel is symmetric because of undirectedness of the graph:

$$K(p,q) = \sum_{i\in\mathcal{V}} w_{p,i} \cdot w_{i,q} = \sum_{i\in\mathcal{V}} w_{q,i} \cdot w_{i,p} = K(q,p)$$

To prove K is SPD, we need to prove that: for am arbitrary sequence of $v_1, v_2, \cdots, v_t \in \mathcal{V}$, the matrix below is SPSD:

$$\boldsymbol{K} = (K(v_i, v_j))_{i,j\in[t]} = ((W \times W^T)_{v_i,v_j})_{i,j\in[t]}$$

This can be considered as a submatrix of $W \times W^T$ of rows and columns being $v_1, v_2, \cdots, v_t$. Thus $\boldsymbol{K}$'s SPSD can be infered from the SPSD of $W \times W^T$. [1]

Now I prove $W \times W^T$ is SPSD: for $x = (x_1, \cdots, x_{|\mathcal{V}|}) \in \mathbb{R}^{|\mathcal{V}|}$

$$x \times (W \times W^T) \times x^T = (x \times W) \times (W^T \times x^T) = (x \times W) \times (x \times W)^T = |x \times W|^2_{L^2} \geq 0$$

This completes the proof.

1.2 Pixel Kernel

1.2.1 PDSness of a kernel

By definition of PDS, we need to prove: for an arbitrary sequence $z_1, z_2, \cdots, z_n$ in $\mathbb{R}$, the matrix K defined in following way is SPSD:

$$K_{i,j} = S(z_i, z_j)$$

An easy observation might help us proving K is SPSD is that $S(z, z') = \max(0, \min(z, z'))$.

K is symmetric can be seen as:

$$K_{i,j} = S(z_i, z_j) = \max(0, \min(z_i, z_j)) = \max(0, \min(z_j, z_i)) = S(z_j, z_i) = K_{j,i}$$

[1] A short explaination: The bilinear form with respect to $\boldsymbol{K}$ can be considered as a bilinear form with respect to $W \times W^T$ with unrelevant coordinates equals to zero. Thus the SPSDness of $W \times W^T$ implies SPSDness of K

Also, we need to have a few other observations for our proof to continue:

- if z_i is replaces by $\tilde{z_i} = \max(0, z_i)$, K remains unchanged. Thus we can assume $z_i \geq 0$ with out loss of generality. With this assumption, we can say $K_{i,j} = S(z_i, z_j) = \min(z_i, z_j)$
- Permutation of $(z_i)_{i=1,\cdots,n}$ permutes the rows and columns of K in a coordinate change style, so does not effect the SPSD-ness of K. Thus, we can assume $z_1 \leq z_2 \leq \cdots \leq z_n$ [2]

Let $w_1 = z_1$, $w_i = z_i - z_{i-1}, \forall i > 1$. Let W be the matrix with diagonal being : $W_{i,i} = w_i$, and $W_{i,j} = 0$ for $i \neq j$. Let J be an $n \times n$ matrix with ones in upper triangle and diagonal, zeroes in the lower triangle. Then we can see that $K = J^T W J$. W is certainly a SPSD, as it is a diagonal matrix with non-negative diagonal. And binear-form with respect to K can be understood as a bilinear form with respect to W composing with a coordinate change J. Thus K is also SPSD, as coordinate change does not effect the symmetric semipositive definitive-ness[3]

1.2.2 PDSness of the K_μ Kernel

What we are using here is that PDS kernel is closed under [4]:

- Taking a power series of non-negative coefficients
- Product

Another thing to mention is that the procedure of proving $S'(z, z') = S(|z|^\mu, |z'|^\mu)$ is a PDS kernel is completely similar of proving $S : (z, z') \mapsto \int_0^{+\infty} 1_{t\in[0,z]} 1_{t\in[0,z']} dt$ is a PDS kernel in previous subsubsection. This is because we used nothing more than non-negativity of $S(\cdot, \cdot)$ and we can switch K's rows and columns in certain coordinate change way.

Now we can step by step show that K_μ is PSD:

$\exp(S'(|x_k|^\mu, |x'_k|^\mu)) = \sum_{i=0}^{\infty} \frac{S'(|x_k|^\mu, |x'_k|^\mu)^n}{n!}$. By the fact that composition with positive coefficients power series preserve PSD-ness. $\exp(S'(|x_k|^\mu, |x'_k|^\mu))$ is PSD.

$K_\mu = \prod_{i=1}^{k} \exp(S'(|x_k|^\mu, |x'_k|^\mu))$ is thus product of PSD kernels, this tells that K_μ is a PSD.

2 Logistic Loss Boosting

2.1 Φ's simple properties

The first order derivative $\Phi'(u) = -\frac{1}{\log 2}(\frac{e^{-\mu}}{1+e^{-\mu}}) < 0$ tells us that Φ is a decreasing function.

The second order derivative $\Phi''(u) = \frac{e^{-u}}{\log 2 \cdot (1+e^{-u})^2} > 0$ tells us that Φ is convex.

For $u \leq 0$, $\Phi(u) - 1_{u\leq 0} = \Phi(u) - 1$ achieves the minimal value at $u = 0$ by the decreasing nature of Φ. Thus in this domain $\Phi(u) - 1_{u\leq 0} \geq \Phi(0) - 1 = 0$.

For $u > 0$, $\Phi(u) - 1_{u\leq 0} = \Phi(u) > 0$.

In conclusion, Φ is a convex decreasing function upper-bounding $1_{u\leq 0}$

2.2 Gradient descent with Φ: the objective function and reweighting

The objective function defined with respect to Φ, analogue to the object function of adaboost, should be:

$$F(\overline{\boldsymbol{\alpha}}) = \frac{1}{m}\sum_{i=1}^{m} \Phi(y_i f(x_i)) = \frac{1}{m}\sum_{i=1}^{m} \log_2(1 + \exp(-y_i f(x_i))) = \frac{1}{m}\sum_{i=1}^{m} \log_2(1 + \exp(-y_i \sum_{j=1}^{T} \overline{\alpha}_j h_j(x_i))) \tag{1}$$

For a sample $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$ and T iterations.

[2]This is standard techniques and can be found in https://mathoverflow.net/questions/264120/prove-that-matrix-is-positive-definite or some other standard linear algebra textbooks.

[3]You can find this sort of propositions in any linear algebra textbook, don't deduct my points for not explaining it.

[4]this result can be seen in the textbook theorem 6.10

Now I need to establish convexity of F to make sure gradient descent is legit, for $\lambda \in (0,1)$:

$$\begin{aligned}
&\lambda F(\alpha) + (1-\lambda)F(\beta) && (2)\\
&= \frac{1}{m}\sum_{i=1}^{m} \lambda\Phi\left(y_i \sum_{j=1}^{T} \alpha_j h_j(x_i)\right) + (1-\lambda)\Phi\left(y_i \sum_{j=1}^{T} \beta_j h_j(x_i)\right) && (3)\\
&\leq \frac{1}{m}\sum_{i=1}^{m} \Phi\left(\lambda y_i \sum_{j=1}^{T} \alpha_j h_j(x_i) + (1-\lambda) y_i \sum_{j=1}^{T} \beta_j h_j(x_i)\right) && (4)\\
&= \frac{1}{m}\sum_{i=1}^{m} \Phi\left(y_i \sum_{j=1}^{T} (\lambda\alpha_j + (1-\lambda)\beta_j) h_j(x_i)\right) && (5)\\
&= F(\lambda\alpha + (1-\lambda)\beta) && (6)
\end{aligned}$$

where (4) to (5) is by convexity of Φ.

2.3 Direction of coordinate descent

There are some familiar notions in the AdaBoost framework that we need to establish for Logistic loss boosting:

- D_t stands for the probability distribution of S we using for the t-th iteration.
- Z_t stands for normalization factor in reweighting
- $\epsilon_{t,k}$ is the empirical error of h_k with respect to the distribution D_t

And below is the definitions of these notions in Logistic Loss Boosing:

- $D_1(i) = \frac{1}{m}$
- $f_t = \sum_{s=1}^{t} \alpha_s h_s$ is the model we got from first k iterations.
- $Z_t = \sum_{i=1}^{m} \Phi(y_i f_{t-1}(x_i))$, for $t > 1$
- $D_t(i) = \Phi(y_i f_{t-1}(x_i))/Z_t$, for $t > 1$
- $\epsilon_{t,k} = \mathbb{E}_{i\sim D_t}[1_{y_i h_k(x_i) \leq 0}]$

where t represents the current iteration and $t-1$ the previous iteration. Once we can find the best direction and best step size in iteration t, we can inductively went to iteration $t+1$. In this sense, we do not have to worry about what is the result of iteration $t-1$ exactly, as it is mere a step of an induction procudure, with initial condition settled.

By definition we have:

$$F(\alpha^{t-1} + \eta e_k) = \frac{1}{m}\sum_{i=1}^{m} \Phi(y_i \sum_{j=1}^{N} \alpha_j^{t-1} h_j(x_i) + \eta y_i h_k(x_i))$$

where the superscript in α^{t-1} means the coefficients of f_{t-1}.

This helps us taking differential of F at the direction of h_k:

$$\begin{aligned}
&F'(\alpha^{t-1}, e_k) && (7)\\
&= \lim_{\eta \to 0} \frac{F\left(\alpha^{t-1} + \eta e_k\right) - F\left(\alpha^{t-1}\right)}{\eta} && (8)\\
&= \frac{1}{m} \sum_{i=1}^{m} y_i h_k(x_i) \Phi'(y_i \sum_{j=1}^{N} \alpha_j^{t-1} h_j(x_i)) && (9)\\
&= \frac{1}{m} \sum_{i=1}^{m} y_i h_k(x_i) \Phi'(y_i f_{t-1}(x_i)) && (10)\\
&= \frac{1}{m \log 2} \sum_{i=1}^{m} y_i h_k(x_i) (\frac{1}{1 + \exp(-y_i f_{t-1}(x_i))} - 1) && (11)\\
&= \frac{1}{m \log 2} \sum_{i=1}^{m} y_i h_k(x_i) (2^{-\Phi(y_i f_{t-1}(x_i))} - 1) && (12)\\
&\text{using Taylor expansion around zero of: } 2^{-x} - 1 = -(\log 2)x + o(x) && (13)\\
&= \frac{-1}{m} \sum_{i=1}^{m} y_i h_k(x_i) (\Phi(y_i f_{t-1}(x_i)) + o(\Phi(y_i f_{t-1}(x_i)))) && (14)\\
&= \frac{-Z_t}{m} \sum_{i=1}^{m} y_i h_k(x_i) (D_t(i) + o(D_t(x_i))) && (15)\\
&= \frac{-Z_t}{m} \mathbb{E}_{i \sim D_t}\left[1_{y_i h_k(x_i)=1} - 1_{y_i h_k(x_i)=1} + o(1)\right] && (16)\\
&= \frac{Z_t}{m} (2\epsilon_{t,k} - 1 + o(1)) && (17)
\end{aligned}$$

Notice that this $o(1)$ can be simply bounded in terms of $\Phi(y_i f_{t-1}(x_i))$, the selection of h_k might alter this small term but not significantly. Thus we can ignore this term in making decision of h_k. So we choose the one with the smallest error to be our direction.

2.4 An inequality

With the facts of $\Phi' < 0$ and fundamental theorem of calculus, we can see the stated inequality is equivalent to :

$$\int_{u}^{u+v} \frac{\Phi'(s)}{\Phi'(u)} ds \geq 1 - e^{-v}$$

If v is positive, then in the LSH integral we always have $s \geq v$, which will gives us:

$$\int_{u}^{u+v} \frac{\Phi'(s)}{\Phi'(u)} ds = \int_{u}^{u+v} (\frac{e^{-s}}{e^{-u}})(\frac{1 + e^{-u}}{1 + e^{-s}}) ds \geq \int_{u}^{u+v} \frac{e^{-s}}{e^{-u}} ds = 1 - e^{-v}$$

If v is nagetive, similarly:

$$\int_{u}^{u+v} \frac{\Phi'(s)}{\Phi'(u)} ds = -\int_{u+v}^{u} (\frac{e^{-s}}{e^{-u}})(\frac{1 + e^{-u}}{1 + e^{-s}}) ds \geq -\int_{u+v}^{u} \frac{e^{-s}}{e^{-u}} ds = 1 - e^{-v}$$

In conclusion, we have:

$$\Phi(u + v) - \Phi(u) \leq \Phi'(u)(1 - e^{-v}), \forall (u, v) \in \mathbb{R}^2$$

2.5 A useful bound

$$F\left(\alpha^{t-1}+\eta\mathbf{e}_k\right)-F\left(\alpha_{t-1}\right) \tag{18}$$
$$=\frac{1}{m}\sum_{i=1}^{m}\Phi(y_if_{t-1}(x_i)+\eta y_ih_k(x_i))-\Phi(y_if_{t-1}(x_i)) \tag{19}$$
$$\leq\frac{-1}{m}\sum_{i=1}^{m}\Phi'(y_if_{t-1}(x_i))(\exp(-\eta y_ih_k(x_i))-1) \tag{20}$$
$$=\frac{-1}{m\log 2}\sum_{i=1}^{m}(2^{-\Phi(y_if_{t-1}(x_i))}-1)(\exp(-\eta y_ih_k(x_i))-1) \tag{21}$$
$$=\frac{1}{m\log 2}\sum_{i=1}^{m}(1-2^{-Z_tD_t(i)})(\exp(-\eta y_ih_k(x_i))-1) \tag{22}$$
$$=\frac{1}{m}\sum_{i=1}^{m}Z_tD_t(i)(\exp(-\eta y_ih_k(x_i))-1) \tag{23}$$

where the last step is by an additional inequality: $\forall t\geq 0, \frac{1-2^{-t}}{\log 2}\leq t$.[5]

2.6 The best step in a given direction

As the description of the problem indicated, I should find η such that $T(\eta)=\sum_{i=1}^{m}Z_tD_t(i)(\exp(-\eta y_ih_k(x_i))-1)$ is minimized. It's natural to try differentiation:

$$\frac{dT}{d\eta} \tag{24}$$
$$=-\sum_{i=1}^{m}D_t(i)Z_ty_ih_k(x_i)\exp(-\eta y_ih_k(x_i)) \tag{25}$$
$$=\sum_{i=1}^{m}D_t(i)Z_t\exp(\eta)1_{y_ih_k(x_i)=-1}-\sum_{i=1}^{m}D_t(i)Z_t\exp(-\eta)1_{y_ih_k(x_i)=1} \tag{26}$$
$$=e^{\eta}\epsilon_{t,k}-e^{-\eta}(1-\epsilon_{t,k}) \tag{27}$$
$$=(e^{\eta}+e^{-\eta})\epsilon_{t,k}-e^{-\eta} \tag{28}$$

It is even more nature to take second derivative, which will imply the convexity of T:

$$T''(\eta)=\frac{d}{d\eta}\frac{dT}{d\eta}=\frac{d}{d\eta}((e^{\eta}+e^{-\eta})\epsilon_{t,k}-e^{-\eta}) \tag{29}$$
$$=(e^{\eta}-e^{-\eta})\epsilon_{t,k}+e^{-\eta}=e^{\eta}\epsilon_{t,k}+(1-\epsilon_{t,k})e^{-\eta}>0 \tag{30}$$

Since we have the convexity of T, we are sure T achieves its minimizing value at point of first order derivative being zero. Now, we only need to solve:

$$0=T'(\eta)=(e^{\eta}+e^{-\eta})\epsilon_{t,k}-e^{-\eta} \tag{31}$$
$$\eta=\frac{1}{2}\log\frac{1-\epsilon_{t,k}}{\epsilon_{t,k}} \tag{32}$$

This is syntactically similar to the results we have for adaboost.

[5]This inequality can be proven by taking 1st order derivative on both sides.

2.7 Full pseudocode

Algorithm 1 Pseudocode of Logistic loss boosting

```
Input: A sample of size $m$: $S = ((x_1, y_1), \cdots, (x_m, y_m))$
 1: for $i \longleftarrow 1$ to $m$ do
 2:     $D_1(i) = \frac{1}{m}$
 3: end for
 4: for $t \longleftarrow 1$ to $T$ do
 5:     $h_t =$ base classifier in $H$ with small error $\epsilon_t = \mathbb{P}_{i\sim D_t}[h_t(x_i) \neq y_i]$
 6:     $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
 7:     $f_t = \sum_{s=1}^{t} \alpha_s h_s$
 8:     $Z_{t+1} = F(\alpha^t) = \sum_{i=1}^{m} \Phi(\alpha_t y_i f_t(x_i))$
 9:     for $i \longleftarrow 1$ to $m$ do
10:         $D_{t+1}(i) = \Phi(\alpha_t y_i f_t(x_i))/Z_{t+1}$
11:     end for
12: end for
13: Return $f_T$
```

2.8 Margin-based generalization bound

Bounding the generalization error by the empirical error and VC-dimensions has has been done in the textbook at page 158 corollary 7.6 :

Theorem 1 *Let H be a family of functions taking values in $\{-1, 1\}$ with VC dimension d. Fix $\rho > 0$. For any $\delta > 0$, with probability at least $1 - \delta$, the following holds for all $h \in conv(H)$*

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\sqrt{\frac{2d\log\frac{em}{d}}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}$$

Next, I need to bound the $\widehat{R}_\rho(h)$ term:

$$\widehat{R}_\rho(f_T) = \frac{1}{m}\sum_{i=1}^{m} 1_{y_i f_T(x_i) - \rho|\alpha|_{L^2} \leq 0} \leq \frac{1}{m}\sum_{i=1}^{m} \Phi(y_i f_T(x_i) - \rho|\alpha|_{L^2}) = \frac{e^{\rho|\alpha|}}{m}\sum_{i=1}^{m} \Phi(y_i f_T(x_i)) \tag{33}$$

where with the assumption that $\alpha_t \geq 0$, $|\alpha| = \sum_{i=1}^{T} \alpha_t$. Then the next thing to do is bounding $F(\alpha^T) = \frac{1}{m}\sum_{i=1}^{m} \Phi(y_i f_T(x_i))$. Recalling the previous results of **5. A useful bound**, we can have:

$$F(\alpha^t) - F(\alpha^{t-1}) \tag{34}$$

$$\leq \frac{1}{m}\sum_{i=1}^{m} Z_t D_t(i)(\exp(-\alpha_t y_i h_t(x_i)) - 1) \tag{35}$$

$$= \frac{Z_t}{m}\sum_{i=1}^{m} D_t(i) 1_{y_i h_t(x_i)=1}(\exp(-\alpha_t) - 1) + D_t(i) 1_{y_i h_t(x_i)=-1}(\exp(\alpha_t) - 1) \tag{36}$$

$$= \frac{Z_t}{m}\mathbb{E}_{i\sim D_t}[1_{y_i h_t(x_i)=1}(\exp(-\alpha_t) - 1) + 1_{y_i h_t(x_i)=-1}(\exp(\alpha_t) - 1)] \tag{37}$$

$$= \frac{Z_t}{m}\epsilon_t(e^{\alpha_t} - 1) + (1 - \epsilon_t)(e^{-\alpha_t} - 1) \tag{38}$$

$$= \frac{Z_t}{m}(\sqrt{\epsilon_t(1 - \epsilon_t)} - 1) \tag{39}$$

$$= \frac{F(\alpha^{t-1})}{m}(\sqrt{\epsilon_t(1 - \epsilon_t)} - 1) \tag{40}$$

Thus, $F(\alpha^t) = F(\alpha^t) - F(\alpha^{t-1}) + F(\alpha^{t+1}) \le F(\alpha^{t+1})(\frac{m-1}{m} + \frac{\sqrt{\epsilon_t(1-\epsilon_t)}}{m})$

Doing induction we can have:

$$F(\alpha^T) \le \prod_{t=1}^{T}(\frac{m-1}{m} + \frac{\sqrt{\epsilon_t(1-\epsilon_t)}}{m}) = \prod_{t=1}^{T}(\frac{m-1}{m} + \frac{\sqrt{1-(1-2\epsilon_t)^2}}{2m})$$

Now add an assumption that $0.5 - \epsilon_t > \gamma$, we can have $F(\alpha^T) < (\frac{m-1+e^{-\gamma^2}}{m})^T < e^{-T\gamma^2/m}$

Notice that each term in the product is less than 1. This tells us the log error is indeed decreasing as iteration goes. In conclustion, we have:

$$\begin{aligned} R(f_T) &\le \widehat{R}_\rho(f_T) + \frac{2}{\rho}\sqrt{\frac{2d\log\frac{em}{d}}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}} && (41)\\ &\le \frac{e^{\rho|\alpha|}}{m}F(\alpha^T) + \frac{2}{\rho}\sqrt{\frac{2d\log\frac{em}{d}}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}} && (42)\\ &\le \frac{e^{\rho|\alpha| - \frac{T\gamma^2}{m}}}{m} + \frac{2}{\rho}\sqrt{\frac{2d\log\frac{em}{d}}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}} && (43)\\ &\le \frac{\exp(\frac{1}{2}T(\log\frac{1+2\gamma}{1-2\gamma}) - \frac{T\gamma^2}{m})}{m} + \frac{2}{\rho}\sqrt{\frac{2d\log\frac{em}{d}}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}} && (44) \end{aligned}$$

Remark 2 *It seems that the logitboost does not converge its error exponentially fast. But it has another advantage of insensitive to 'noisy' samples. In this sense, it's better than adaboost.*

2.9 Empirical

The jupyter notebook I coded to do this problem can be found in the folder hw3 in the github projects `https://github.com/WangHaiYang874/machine_learning2020`.

Figures below is my results.

The shape of all the graphs are volatile. I believe this is because my self-implemented boosting algorithm is having certain issues of precision of float numbers and the details in my implementation on finding a best stump. But still you can observe that log-boost is less noisy compare to adaboost. This property is especially more apperant in the cross-validions, which I believe is because taking mean value makes the curve smooth.

From the graph I believe that we should use log-boost with $T = 1000$ as the best parameter for our boosting algorithm on this sample.

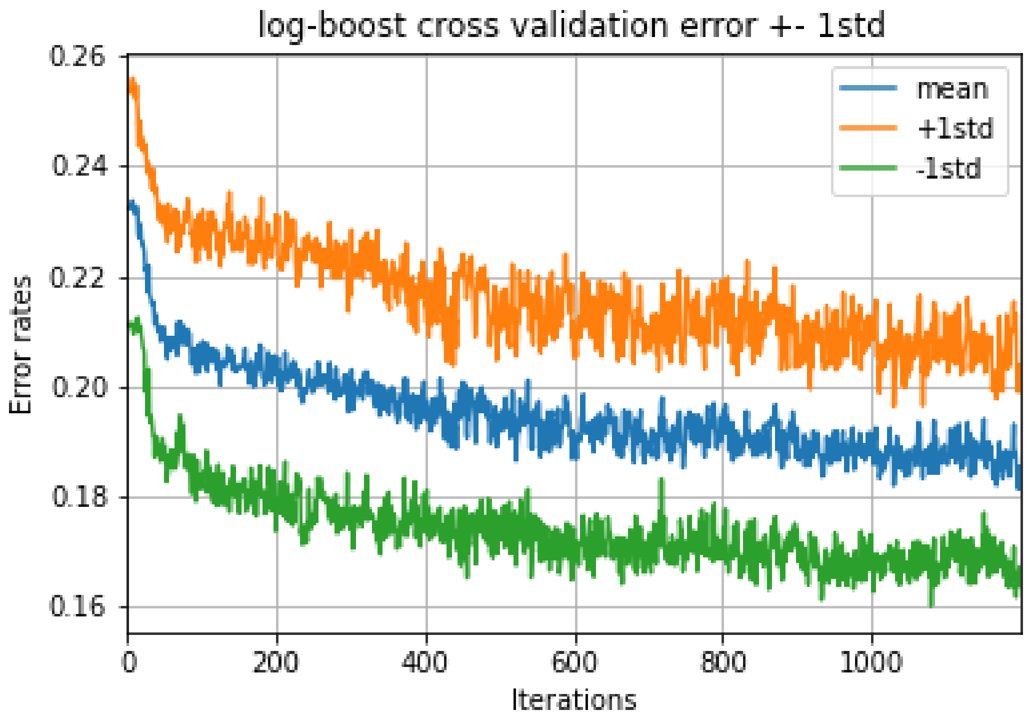

Figure 1: Log-boost cross validation

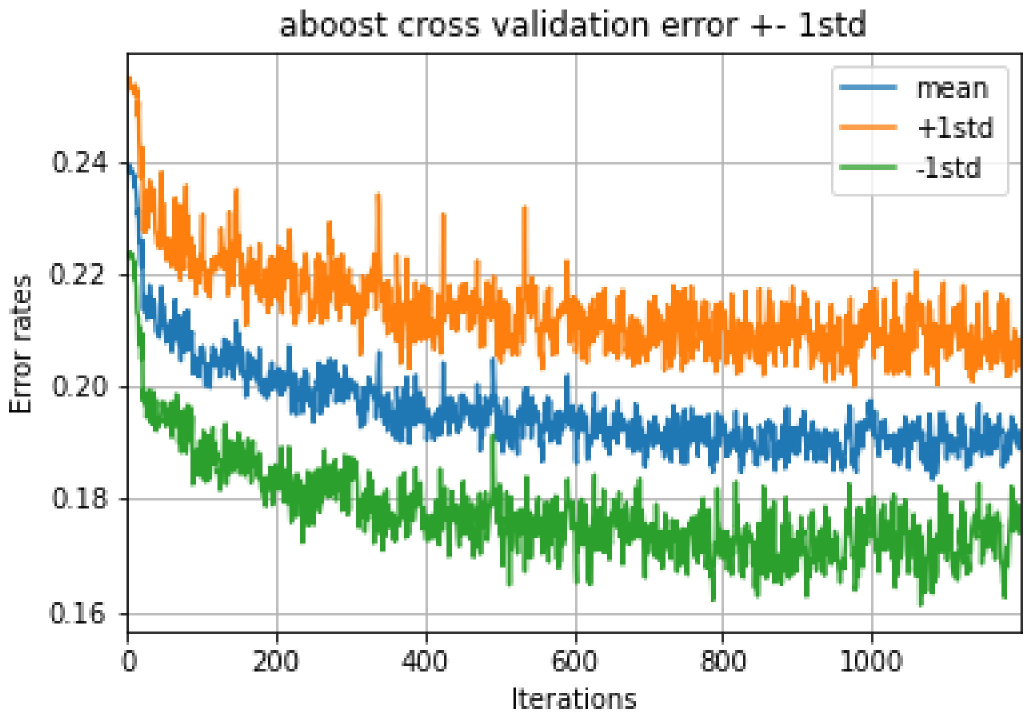

Figure 2: Adaboost cross validation

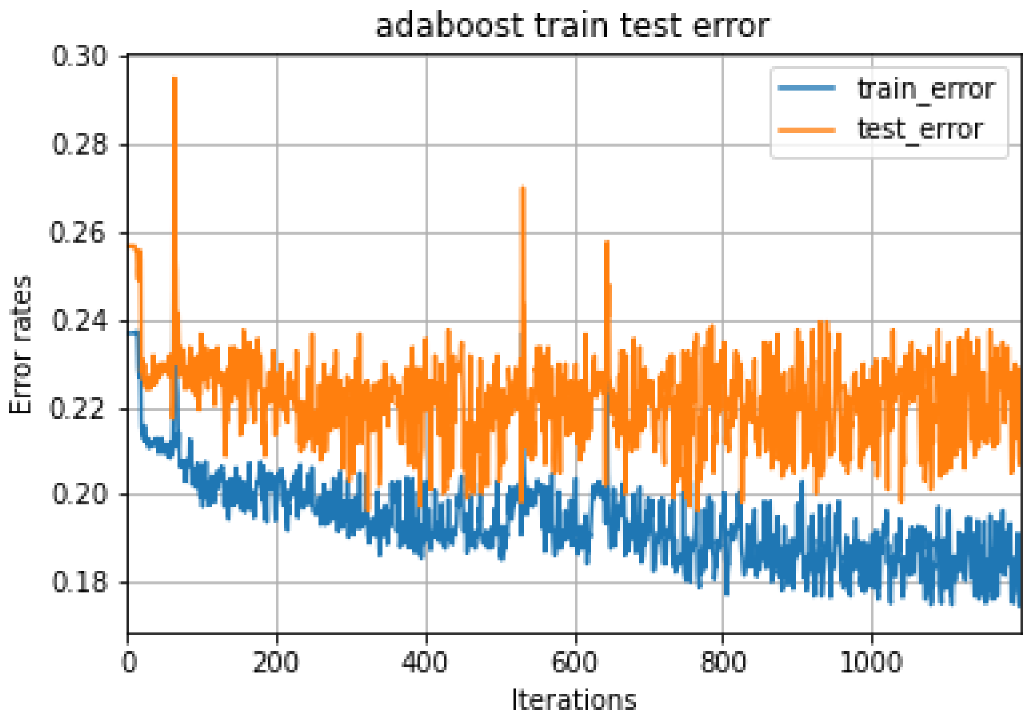

Figure 3: adaboost train test error

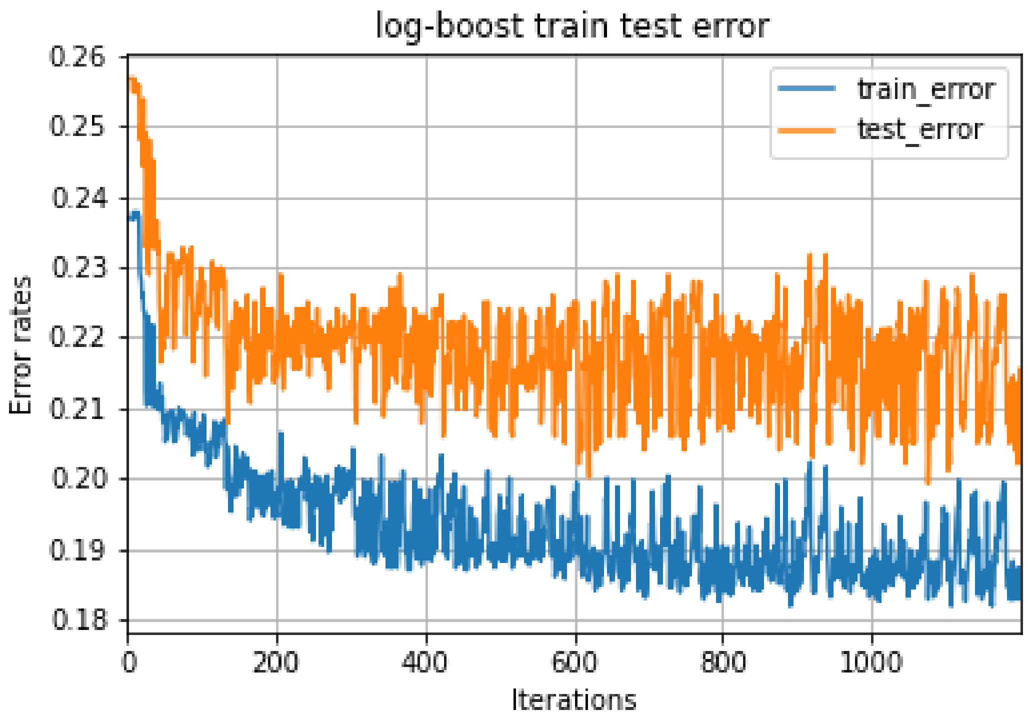

Figure 4: log-boost train test error.png