CSCI 5525 Machine Learning HW2

1 General Knowledge

1.
When the expected loss is the squared loss, we have the loss as

$$E_{(x,y)}[\ell(f(x), y)] = \iint (y - f(\mathbf{x}))^2 p(\mathbf{x}, y) d\mathbf{x} dy$$

The optimal solution to minimize the loss is $h(\mathbf{x}) = \int y p(y|\mathbf{x}) = E[y|\mathbf{x}]$. Then we can rewrite the loss as

$$E_{(x,y)}[\ell(f(x), y)] = \int (f(x) - h(\mathbf{x}))^2 p(x) dx + \iint (h(\mathbf{x}) - y)^2 p(x, y) dx dy$$

Taking expectation of the first term on a particular dataset D, we have

$$\begin{aligned} &E_D\left[\{f(\mathbf{x}; D) - E_D[f(\mathbf{x}; D)] + E_D[f(\mathbf{x}; D)] - h(\mathbf{x})\}^2\right] \\ &= \left(E_D[f(\mathbf{x}; D)] - h(\mathbf{x})\right)^2 + E_D\left[\left(f(x; D) - E_D[f(x; D)]\right)^2\right] \end{aligned}$$

Substitute the above expectation into the loss, we can have following the decomposition of the expected squared loss

$$\begin{aligned} &E_{(x,y)}[\ell(f(x), y)] \\ &= \int \left(E_D[f(\mathbf{x}; D)] - h(x)\right)^2 p(\mathbf{x}) d\mathbf{x} + \int E_D\left[\left(f(\mathbf{x}; D) - E_D[f(\mathbf{x}; D)]\right)^2\right] p(\mathbf{x}) d\mathbf{x} + \iint (h(x) - y)^2 p(x, y) dx dy \\ &= \text{bias}^2 + \text{variance} + \text{noise} \end{aligned}$$

2.
Suppose that we generated the data from the true distribution, and we know the optimal classifier $h(\mathbf{x})$ based on the true distribution. The method to estimate bias, variance, and noise using cross-validation is as follows

1. Split the data into L training folds and one test fold with size N
2. Learn a prediction function $f^{(l)}(x)$ with fold l ($l = 1, \cdots, L$) respectively
3. Evaluate the prediction functions on the test data

The terms are given by

$$\begin{aligned}
\text{bias}^2 &= \frac{1}{N}\sum_{n=1}^{N}\left\{\frac{1}{L}\sum_{l=1}^{L} f^{(l)}(x_n) - h\left(x_n\right)\right\}^2 \\
\text{variance} &= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{L}\sum_{l=1}^{L}\left\{f^{(l)}\left(x_n\right) - \frac{1}{L}\sum_{l=1}^{L} f^{(l)}(x_n)\right\}^2 \\
\text{loss} &= \sum_{n=1}^{N}\left\{y_n - \frac{1}{L}\sum_{l=1}^{L} f^{(l)}(x_n)\right\}^2 \\
\text{noise} &= \text{loss} - \text{bias}^2 - \text{variance}
\end{aligned}$$

3.
(1) Incorrect data point in the training data would cause the learned classifier different from the true one, thus causing the bias.
(2) The incorrect data point in the training data would possibly fall on the wrong side of the boundary in the SVM. The hinge loss uses a linear function (in $yf(x)$) $1 - yf(x)$ to represent the loss caused by such mis-classification.
(3) The penalty parameter C affects the bias on test data. It is analogous to the inverse of a regularization coefficient because it controls the trade-off between minimizing training errors and controlling model complexity. Thus increasing C would decrease the regularization effect and thus increase the model capacity, and finally decrease the bias. Conversely decreasing C would increase the bias.
4.
The classification is based on the posterior probabilities. For class C_k, its posterior probability is

$$p\left(C_k|\mathbf{x}\right) = \frac{p(\mathbf{x}|C_k)p\left(C_k\right)}{p(\mathbf{x})}$$

so we should compute the score for each class ($k = 1, \cdots, K$)

$$\begin{aligned}
a_k &= \log p(\mathbf{x}_{\text{test}}|C_k) + \log p\left(C_k\right) \\
&= \log \sum_{j=1}^{p} \frac{1}{\sqrt{2\pi\sigma_{jk}^2}} e^{-\frac{(x_j-\mu_{jk})^2}{2\sigma_{jk}^2}} + \log p\left(C_k\right)
\end{aligned}$$

then the model would choose the class with maximum score.

2 Experiments

The best procedure to finalize training model is (b) weight the predictions of each model by $w_i = \exp(-e_i)$. The reason is as follows

- Option (a) is usually used in practice, while simply choosing h_k just take full advantage of one fold of the training data. And it's still possible to come up with a better model by leveraging all other folds altogether.
- Option (b) is kind of like a community machine, i.e., the models are combined together by weighted voting and the single model with a lower error would have a much higher weight

because of exponential function. Compared to (a), it might lose a little training accuracy but potentially gains a lot of generalization capacity. For example, if the single model with minimum error actually fits some noise data point, then option (b) would lessen the effect of noise data point by also considering other models. So it's more robust to noise.

- Option (c) is not really reasonable. Updating θ_k on the held-out data would lead to some θ that mainly fits on the held-out data. It's meaningless to first find a optimal θ_k with the folds. And we're not really sure about how new θ would behave on the future new data.

3 Kernels

1.
The original objective function is equivalent to

$$\frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \mathbf{w}^T\mathbf{w}$$

The gradient of the above objective function w.r.t. $\mathbf{w}$ is

$$\nabla_{\mathbf{w}} = \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + 2\lambda\mathbf{w}$$

Setting the gradient to 0, we have

$$(\mathbf{X}^T\mathbf{X} + 2\lambda\mathbf{I}_p)\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

When $\mathbf{X}^T\mathbf{X}^T + 2\lambda\mathbf{I}_p$ is full rank, we have the solution

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + 2\lambda\mathbf{I}_p)^{-1}\mathbf{X}^T\mathbf{y}$$

When $n < p$, whether the solution is valid depends on the invertibility of the matrix, and it can be either rank-deficient or full-rank due to the regularization term $2\lambda\mathbf{I}_p$. In general the regularization term helps to stabilize the inverse numerically and the solution should be still valid.
2.
Using the matrix inversion lemma, we can convert the above solution

$$\begin{aligned}
\mathbf{w}^* &= (\frac{1}{2\lambda}\mathbf{I}_p)\mathbf{X}^T\left[\mathbf{X}(\frac{1}{2\lambda}\mathbf{I}_p)\mathbf{X}^T + \mathbf{I}_n\right]^{-1}\mathbf{y} \\
&= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + 2\lambda\mathbf{I}_n)^{-1}\mathbf{y} \\
&= \sum_i^n a_i\mathbf{x}_i
\end{aligned}$$

where a_i is the i-th element of the vector $(\mathbf{X}\mathbf{X}^T + 2\lambda\mathbf{I}_n)^{-1}\mathbf{y}$ and $\mathbf{x}_i$ is the row vector of $\mathbf{X}$. Next we replace the data vector with the feature vector: $\mathbf{x}_i \to k_i = k(\mathbf{x}_i, \cdot)$, and we can kernelize the solution

$$\mathbf{w}^* = \sum_i^n \alpha_i k(\mathbf{x}_i, \cdot)$$

where $\boldsymbol{\alpha} = (\mathbf{K} + 2\lambda\mathbf{I}_n)^{-1}\mathbf{y}$ and α_i is its i-th element, and $\mathbf{K}$ is the $n \times n$ Gram matrix and $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

3.
To fit polynomials to order m, we use a polynomial kernel function

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + 1)^m$$

We substitute the kernel function into $\mathbf{w}^*$, then fit the polynomials as

$$\begin{aligned} f(\mathbf{z}) &= \mathbf{z}^T\mathbf{w}^* \\ &= \sum_i^n \alpha_i k(\mathbf{x}_i, \mathbf{z}) \\ &= \sum_i^n \alpha_i (\mathbf{x}_i^T\mathbf{z} + 1)^m \end{aligned}$$

where $\boldsymbol{\alpha} = (\mathbf{K} + 2\lambda\mathbf{I}_n)^{-1}\mathbf{y}$ and α_i is its i-th element, and $\mathbf{K}$ is the $n \times n$ Gram matrix and $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j + 1)^m$.

4 Gradient Descent

1.

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\left\{-y_i\mathbf{w}^T\mathbf{x_i} + \log\left(1 + \exp\left(-\mathbf{w}^T\mathbf{x_i}\right)\right)\right\} + \lambda G(\mathbf{w})$$

The subgradient w.r.t $\mathbf{w}$ is

$$\partial\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left\{y_i\mathbf{x}_i + \frac{\exp(-\mathbf{w}^T\mathbf{x}_i)}{1 + \exp(-\mathbf{w}^T\mathbf{x}_i)}\mathbf{x}_i\right\} + \partial G$$

where the i-th element of ∂G is

$$\partial_i G = \begin{cases} \inf, & \mathbf{w}_i = 0 \\ a\lambda(a\left|w_i\right|^{\frac{1}{2}} + b)\dfrac{x}{\left|x\right|^{\frac{3}{2}}}, & \text{otherwise} \end{cases}$$

2.
The subgradient is

$$\partial\mathcal{L}(\mathbf{w}) = -\frac{1}{n}\sum_{1}^{n}\left\{y_i\mathbf{x}_i + \frac{\exp(-\mathbf{w}^T\mathbf{x}_i)}{1 + \exp(-\mathbf{w}^T\mathbf{x}_i)}\mathbf{x}_i\right\} + \text{sign}(\mathbf{w})$$

where sign() is the element-wise sign function. Since $\mathcal{L}(\mathbf{w})$ is convex in $\mathbf{w} \in \mathbb{R}^p$, we have the subgradient stochastic gradient descent algorithm for the case as Algorithm 1 .

Algorithm 1: Subgradient Stochastic Gradient Method for Logistic Regression with L_1 Regularization

Input: initial $\mathbf{w}_0$, data with size m
For $t = 0, \cdots, T-1$
  randomly draw $i_t \in \{1, \cdots, m\}$
  compute the subgradient $g_{i_t} = \partial \mathcal{L}_i(\mathbf{w}_t)$
  $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t g_{i_t}$
Output: $\bar{\mathbf{w}}_T = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_T$

3.
$\mathcal{L}(\mathbf{w})$ is a smooth function, so the expected run time is $\mathcal{O}(\frac{n}{\epsilon})$ for gradient descent and $\mathcal{O}(\frac{1}{\epsilon^2})$ for stochastic gradient descent.

5 Boosting

1.
Algorithm 2 is the AdaBoost algorithm.

Algorithm 2: AdaBoost

Input: training set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$
Initialize $\mathbf{w}_1(i) = 1/N$
For $t = 1, \cdots, T$
  Train a weak learner using distribution $\mathbf{w}_t$
  Get weak hypothesis G_t with error $\epsilon_t = \mathrm{P}_{\mathbf{r}_\mathbf{x} \sim \mathbf{w}_t}\left[G_t(\mathbf{x}) \neq y\right]$
  Choose $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  Update $\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\exp(-\alpha_t y_i G_t(\mathbf{x}_i))}{Z_t}$ where $Z_t = \sum_{i=1}^{N} \mathbf{w}_t(i)\exp\left(-\alpha_t y_i G_t\left(x_i\right)\right)$
Output: the classifier $g(\mathbf{x}) = \mathrm{sign}\left[\sum_{t=1}^{T} \alpha_t G_t(\mathbf{x})\right] = \mathrm{sign}[G(\mathbf{x})]$

2.
α_t is selected to minimize the upper bound of the training error. The training error of the final classifier is bounded by

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left(G\left(\mathbf{x}_i\right) \neq y_i\right) \leq \prod_{t=1}^{T} Z_t$$

where

$$Z_t = \sum_{i=1}^{N} \mathbf{w}_t(i)\exp\left(-\alpha_t y_i G_t\left(\mathbf{x}_i\right)\right)$$

We rewrite Z_t to minimize it

$$\begin{aligned} Z_t &= \sum_{i=1}^{N} \mathbf{w}_t(i) \exp\left(-\alpha y_i G_t\left(\mathbf{x}_i\right)\right) \\ &= e^{-\alpha} \sum_{y_i = G_t(\mathbf{x}_i)} \mathbf{w}_t(i) + e^{\alpha} \sum_{y_i \neq G_t(\mathbf{x}_i)} \mathbf{w}_t(i) \\ &= \left(e^{\alpha} - e^{-\alpha}\right) \sum_{i=1}^{N} \mathbf{w}_t(i) \mathbb{1}\left(y_i \neq G_t\left(\mathbf{x}_i\right)\right) + e^{-\alpha} \sum_{i=1}^{n} \mathbf{w}_t(i) \\ &= \sum_{i=1}^{n} \mathbf{w}_t(i) \left\{\left(e^{\alpha} - e^{-\alpha}\right) \epsilon_t + e^{-\alpha}\right\} \end{aligned}$$

Recall that the error rate at t

$$\epsilon_t = \frac{\sum_{i=1}^{N} w_t(i) \mathbb{1}\left(y_i \neq G_t\left(\mathbf{x}_i\right)\right)}{\sum_{t=1}^{N} w_t(i)}$$

Minimizing Z_t over α gives

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

3.
In principal we can choose the hinge loss as the upper bound function because the hinge loss is also an upper bound of the $0 - 1$ loss.
But in practice we shouldn't use it because the nonsmoothness of the hinge loss would disable the manner of the sequential optimization, which is one of the most important properties of the AdaBoost. Recall that in AdaBoost, we decompose the error function of the learners committee at stage t $f_t()$ into the error function of $f_{t-1}()$ and the error function of current weak learner $G_t()$

$$\begin{aligned} E &= \sum_{i=1}^{N} \exp\left\{-y_i f_{t-1}\left(\mathbf{x}_i\right) - y_i \alpha_t G_t\left(\mathbf{x}_i\right)\right\} \\ &= \sum_{n=1}^{N} w_i^{(t)} \exp\left\{-y_i \alpha_t G_t\left(\mathbf{x}_i\right)\right\} \end{aligned}$$

Then we get α_t, $G_t()$ by treating the first term as constant and simply minimizing the second term. And the decomposition relies on the smoothness of the exponential function. While for hinge loss function $C(x) = \max(0, 1 - x)$, we cannot know $C(a + b)$ even we know $C(a)$ and $C(b)$. Thus we have to use global optimization instead of sequential optimization.

6 Adaboost

1.
The weak learner can label examples better than random guessing, so we just need to explore the classification accuracy of each feature.
Potty Training Prep: classify 3 as yes and 1,2 as no, and we can achieve accuracy $P = \frac{10}{12}$. So it's eligible for assignment to weak learners.

Table 1: Feature Price

	0.01	0.01	8.99	8.99	8.99	12.49	13	13.65	33.33	33.33	92.50	92.50
Adoptable	no	no	no	yes	yes	no	yes	yes	no	no	no	no

Price: reorder the price feature and the target as Table 1. Classify the doggie as yes if price < 30 and no otherwise, then we can achieve accuracy $P = \frac{8}{12}$. So it's eligible for assignment to weak learners.
Carpet Damage: classify 0 as yes and 1,2,3 as no, and we can achieve accuracy $P = \frac{11}{12}$. So it's eligible for assignment to weak learners.
Color: classify Brown/White as yes and Yellow as no, and we can achieve accuracy $P = \frac{8}{12}$. So it's eligible for assignment to weak learners.
In conclusion all features are eligible for assignment to weak learners.
2.
We implemented the AdaBoost with 4 Binary classifiers as weak learners, and each of them would just take one feature and classify the data based on a simple threshold. Since the data volume is limited, we simply use the feature in the order of importance we think: Potty Training Prep (weeks), Carpet Damage, Color and Price. The training error becomes zero after 4 stages.
We input the Joey's data, and the prediction result is adoptable.
3.
We list the classification criterion for the first three learners
first learner: weight 0.8047

$$G_1(\text{training weeks}) = \begin{cases} -1, & \text{if training weeks} \leq 2 \\ 1, & \text{otherwise} \end{cases}$$

second learner: weight 1.4722

$$G_2(\text{carpet damage}) = \begin{cases} 1, & \text{if carpet damage} = 0 \\ -1, & \text{otherwise} \end{cases}$$

third learner: weight 1.07

$$G_3(\text{color}) = \begin{cases} 1, & \text{if color is Brown/White} \\ -1, & \text{color is Yellow} \end{cases}$$

If we present the three learners as a decision tree, then there will be eight leaf nodes and the label of leaf nodes are decided by the prediction of each learner and its weight. The tree is shown in Fig. 1.

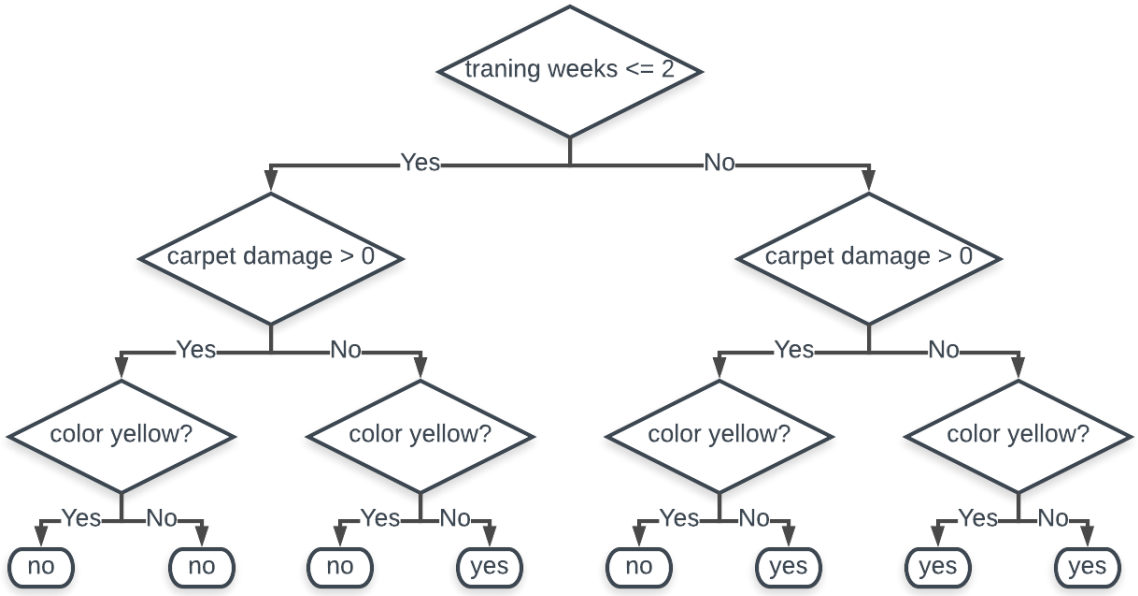

Figure 1: Description of the first 3 stages as a decision tree

4.
We cannot get many extra benefits by adding more learners. Since the data volume is limited and easy to fit, the training error is already zero with only four weak learners. But we can indeed adjust the classification criterion a little on each feature and may gain some generalization benefits.