Elements of Machine Learning

Assignment 2

Xingrong Zong, tnd179

February 27th, 2022

PRML Exercise 8.16

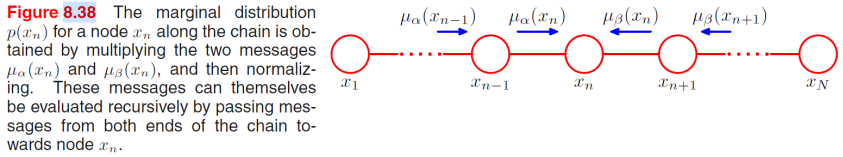

Figure 1: Figure 8.38

To evaluate $p(x_n \mid x_N)$ for every $n = 1, 2, \cdots, N-1$. We will make some observations, and then sketch the algorithm.

Observations:

By using the product rule and then marginalizing over every variable apart from x_n, we obtain the following expression for the conditional probability.

$$
\begin{aligned}
p(x_n \mid x_N) &= \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} \frac{p(x)}{p(x_N)} \\
&= \frac{1}{Z} \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \cdots \frac{\psi_{N-1,N}(x_{N-1}, x_N)}{p(x_N)}
\end{aligned}
$$

The equation above only differs from Equation 8.52 (desired marginal) because

of the marginal distribution $p(x_N)$ in the denominator in the last term. The following is Equation 8.52

$$\begin{aligned} p(x_n) = & \frac{1}{Z} \\ & [\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots [\sum_{x_2} \psi_{2,3}(x_2, x_3)[\sum_{x_1} \psi_{1,2}(x_1, x_2)]] \cdots] \\ & [\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots [\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)] \cdots] \\ & = \frac{1}{Z} \mu_\alpha(x_n) \mu_\beta(x_n) \end{aligned}$$

Assuming that we know $p(x_N)$, we group terms as in Equation 8.52

$$\begin{aligned} p(x_n \mid x_N) &= \frac{1}{Z} \mu_\alpha(x_n) [\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots [\sum_{x_N} \frac{\psi_{N-1,N}(x_{N-1}, x_N)}{p(x_N)}] \cdots] \\ &= \frac{1}{Z} \mu_\alpha(x_n) \mu^*_\beta(x_n) \end{aligned}$$

The difference between the $\mu^*_\beta(x_n)$ here and the $\mu_\beta(x_n)$ in Equation 8.52 is the denominator in the innermost parenthesis.

Algorithm Sketch:

We sketch an $O(NK^2)$ algorithm which uses message passing.

1. First we need to find $p(x_n)$. Since $p(x_N) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n) = \frac{1}{Z}\mu_\alpha(x_n)$, we compute $\mu_\alpha(x_2)$, then $\mu_\alpha(x_3)$, and so forth by using recursive relationship in Equation 8.55

$$\begin{aligned} \mu_\alpha(x_n) &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) [\sum_{x_{n-2}} \cdots] \\ &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_\alpha(x_{n-1}) \end{aligned}$$

   This step is $O(NK^2)$, and every $\mu_\alpha(x_n)$ is stored.

2. Now we obtained $p(x_N)$, we can compute the $\mu^*_\beta(x_n)$ by the following

equations, which give an initial value and a recursive relationship.

$$\mu_\beta^*(x_{N-1}) = \sum_{x_N} \frac{\psi_{N-1,N}(x_{N-1}, x_N)}{p(x_N)}$$

$$\mu_\beta^*(x_n) - \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1})\mu_\beta^*(x_{n+1})$$

This step is also $O(NK^2)$, and every $\mu_\beta^*(x_n)$ is stored.

3. Now we can compute $p(x_n \mid x_N)$ for every $n = 1, \cdots, N-1$ using

$$p(x_n \mid x_N) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta^*(x_n)$$

Since $\sum_{x_n} p(x_n \mid x_N) = 1$, we compute Z as $\sum_{x_n} \mu_\alpha(x_n)\mu_\beta^*(x_n)$.

The algorithm runs in $O(NK^2)$ time, where N is the number of variable nodes and K is the number of states per variable. We propagate messages forward, then backwards, evaluate Z and compute the conditional probabilities $p(x_n \mid x_N)$.

PRML Exercise 8.27

An example is given by

	x=0	x=1	x=2
y=0	0.0	0.1	0.2
y=1	0.0	0.1	0.2
y=2	0.3	0.1	0.0

for which $\hat{x} = 2$ and $\hat{y} = 2$.

EML Exercise 8.4

In fitting a B-spline smoother, the fitted model takes the form

$$\begin{aligned}\hat{f}(x) &= Sy \\ &= H(H^\intercal H)^{-1}H^\intercal y\end{aligned}$$

In this form S is a function of the input x and not the observed response vector y. To use the parametric bootstrap, we simulate n additional samples of the response y by adding noise to the fitted values above as

$$y^* = \hat{f}(x) + \epsilon$$

The above is a vector equation and we do it B times (getting a new vector y^* each time), one for each of the B bagging we are going to aggregate. From the above formula, the fitted model for the j-th parametric bootstrap sample is given by

$$\begin{aligned}\hat{f}(x) &= Sy^* \\ &= S(\hat{f}(x) + \epsilon) \\ &= S^2 y + S\epsilon \\ &= Sy + S\epsilon\end{aligned}$$

Here we have used the fact that S is idempotent so $S^2 = S$. Let's denote the j-th bootstrap fitted model as $\hat{f}_j^*$ and the j-th bootstrap added noise as ϵ_j. If we then bag B simulated sampled our estimate at x would take the form

$$\begin{aligned}\hat{f}_{bag}(x) &= \frac{1}{B}\sum_{j=1}^{B} \hat{f}_j^*(x) \\ &= Sy + S(\frac{1}{B}\sum_{j=1}^{B} \epsilon_j)\end{aligned}$$

Since $\frac{1}{B}\sum_{j=1}^{B} \epsilon_j \to 0$ and $B \to \infty$, so $\hat{f}_{bag}(x) = Sy = \hat{f}(x)$.

EML Exercise 10.1

We know a fixed β the solution $G_m(x)$ is given by

$$G_m = \arg\min_G \sum_{i=1}^{N} \omega_i^m I(y_i \neq G(x_i)$$

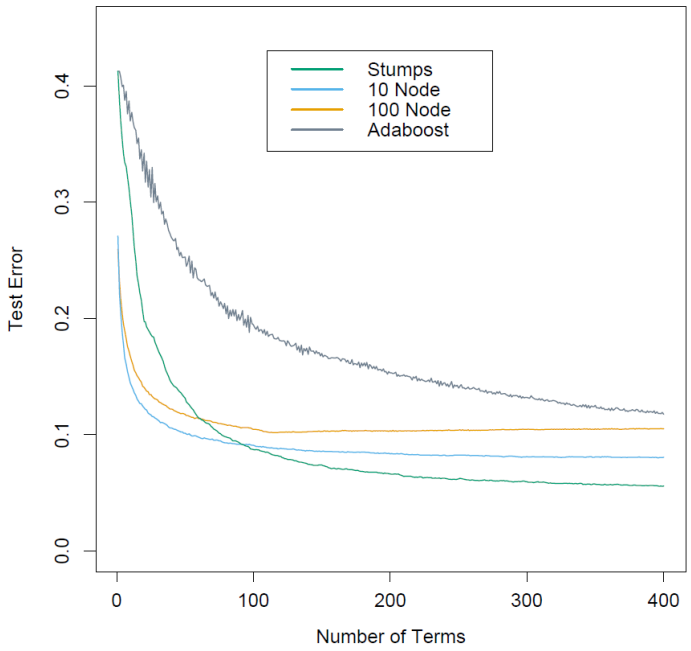

Figure 2: Boosting with different sized trees, applied to the example 10.2 used in Figure 3 (Figure 10.2 from book). Since the generate model is additive, stumps perform the best. The boosting algorithm used the binomial deviance loss in Algorithm 10.3; shown for comparison is the AdaBoost Algorithm 10.1.

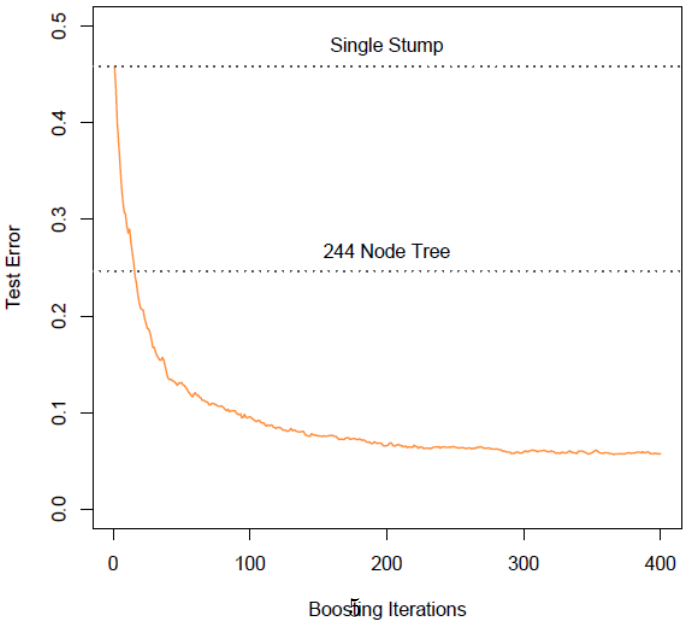

Figure 3: Simulated data: test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Figure 4: AdaBoost Algorithm 10.1.

which states that we should select classifier G_m and $G_m(x_i) = y_i$ for the largest weights ω_i^m values, effectively "null" these values out. In AdaBoost this is done by selecting the training samples according to a discrete distribution ω_i^m specified on the training data. Since $G_m(x)$ is specifically trained to use these samples, we expect it will correctly classify most points. So we select $G_m(x)$ to appropriately minimize the above expression. Once $G_m(x)$ has been selected, we can seek to minimize exponential error with the respect to the β parameter. Then by considering Equation 10.11 with the derived G_m we have

$$(e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} \omega_i^m I(y_i \neq G_m(x_i)) + e^{-\beta} \sum_{i=1}^{N} \omega_i^m$$

Then to minimize this expression with the respect to β, we will take the derivative with respect to β, set the resulting expression equal to zero and solve for β

$$(e^{\beta} + e^{-\beta}) \sum_{i=1}^{N} \omega_i^m I(y_i \neq G_m(x_i)) - \sum_{i=1}^{N} \omega_i^m = 0$$

To facilitate solving for β we multiply the expression by e^{β}

$$(e^{2\beta}+1)\sum_{i=1}^{N}\omega_i^m I(y_i \neq G_m(x_i)) - \sum_{i=1}^{N}\omega_i^m = 0$$

$$e^{2\beta} = \frac{\sum_{i=1}^{N}\omega_i^m - \sum_{i=1}^{N}\omega_i^m I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N}\omega_i^m I(y_i \neq G_m(x_i))}$$

Following the text we can define that the error at the m-th state err_m as

$$err_m = \frac{\sum_{i=1}^{N}\omega_i^m I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N}\omega_i^m}$$

then $e^{2\beta}$ becomes

$$\begin{aligned} e^{2\beta} &= \frac{1}{err_m} - 1 \\ &= \frac{1 - err_m}{err_m} \end{aligned}$$

Finally β is given by

$$\beta = \frac{1}{2}\log(\frac{1-err_m}{err_m})$$

which is the expression Equation 10.12 as desired.

Combining Multiple Learners

Method	Accuracy Avg	Accuracy Std	AUC Avg	AUC Std	Top 3
Decision Tree	0.90695	0.00915	0.90720	0.00964	remove, free, hp

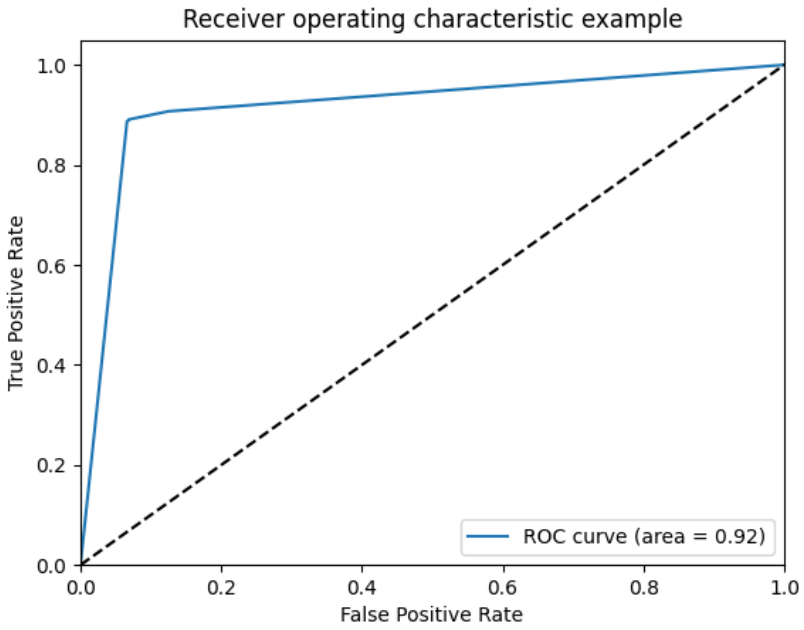

Figure 5: Decision Tree

Method	Accuracy Avg	Accuracy Std	AUC Avg	AUC Std	Top 3(y=ham)	Top 3(y=spam)
Sklearn Gaussian	0.79066	0.01795	0.92464	0.01610	you, will, hp	you, your, will
Sklearn Multinomial	0.87936	0.00703	0.95772	0.00468	you, will, hp	you, your, will
Sklearn Bernoulli	0.89343	0.00617	0.95127	0.00429	you, will, hp	you, your, will
Self Gaussian	0.80875	0.00976	0.87874	0.01061	george, you, hp	you, your, will
Self Multinomial	0.86852	0.00768	0.95187	0.00424	george, you, hp	you, your, will
Self Bernoulli	0.87939	0.00754	0.95803	0.00461	you, will, hp	you, your, will

Gaussian distribution, Multinomial, and Bernoulli, these three methods were tested. The graphs from sklearn and self implementations on these methods appear similar to each other except the very start area when the false positive area is between zero and two. Based on the accuracy, Bernoulli appears to be the best classifier among these methods. However, according to the AUC, Multinomial performs almost the same as Bernoulli.

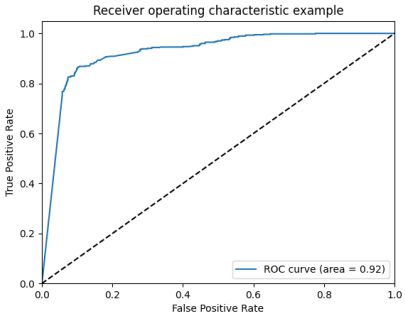

Figure 6: Sklearn Gaussian

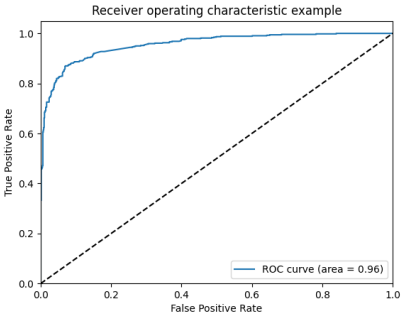

Figure 7: Sklearn Multinomial

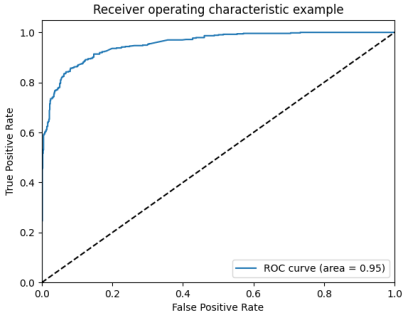

Figure 8: Sklearn Bernoulli

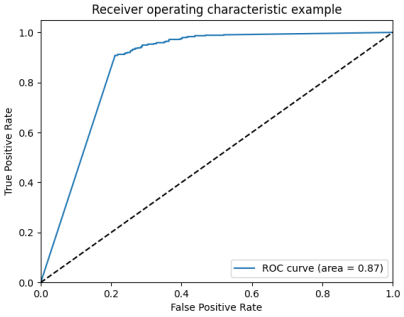

Figure 9: Self Gaussian

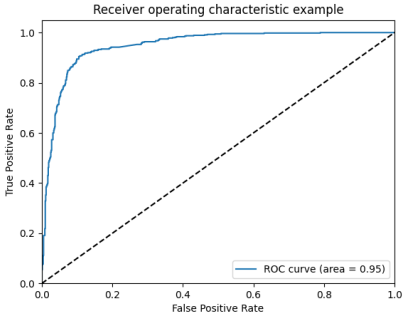

Figure 10: Self Multinomial

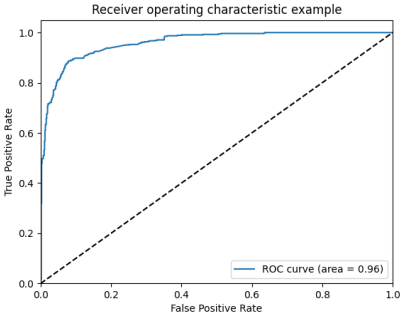

Figure 11: Self Bernoulli