COMPSCI 617 Spring 2019
Homework 3 Solutions

1 Probit Classification

Consider a classifier that operates in a similar way as logistic regression, but with a different link function, that is we use:

$$Y \sim Bernoulli(p), \tag{1}$$

which is the same as logistic regression, except that in this case:

$$p = \Phi(\mathbf{x}\beta), \tag{2}$$

where $\Phi(x)$ is the **Standard Normal CDF**. As usual, suppose the data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ are all IID draws from the same distribution, with $y_i \in \{0, 1\}$ and $\mathbf{x} \in \mathbb{R}^p$.

a) Suppose we introduce a latent variable as follows for each training point:

$$Z = \mathbf{x}\beta + \epsilon, \quad \epsilon \overset{iid}{\sim} \mathcal{N}(0, 1) \tag{3}$$

and then use the following model:

$$Y = \begin{cases} 1 & \text{if } Z > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Show that $\Pr(Y = 1|\mathbf{x}) = \Phi(\mathbf{x}\beta)$ under this model, and that, therefore it is equivalent to the model introduced in (1) and (2).

solution

$$\begin{aligned} \Pr(Y = 1|x) &= \Pr(Z > 0|x) && \text{(By Definition of } Y\text{)} \\ &= \Pr(x\beta + \epsilon > 0) \\ &= \Pr(\epsilon > -x\beta) \\ &= 1 - \Phi(-x\beta) && \text{(Because } \epsilon \sim \mathcal{N}(0, 1)\text{)} \\ &= \Phi(x\beta) && \text{(Symmetry of standard normal CDF).} \end{aligned}$$

b) Write down a likelihood function for the observed data under the model in (1) and (2), and show that the log-likelihood has form:

$$\log \mathcal{L}(\beta) = \sum_{i=1}^{N} y_i \log \Phi(\mathbf{x}_i\beta) + (1 - y_i) \log(\Phi(-\mathbf{x}_i\beta)),$$

where log is the natural logarithm function.

solution From the model in (1) and (2) observed data has likelihood:

$$\begin{aligned}\mathcal{L}(\beta) &= \prod_{i=1}^{N} \Pr(Y_i = 1|x_i)^{y_i}(1 - \Pr(Y_i = 1|x_i))^{1-y_i} \\ &= \prod_{i=1}^{N} \Phi(x_i\beta)^{y_i}(1 - \Phi(x_i\beta))^{1-y_i},\end{aligned}$$

and, therefore:

$$\begin{aligned}\log \mathcal{L}(\beta) &= \log \prod_{i=1}^{N} \Phi(x_i\beta)^{y_i}(1 - \Phi(x_i\beta))^{1-y_i} \\ &= \sum_{i=1}^{N} \log(\Phi(x_i\beta)^{y_i}) + \log((1 - \Phi(x_i\beta))^{1-y_i}) \\ &= \sum_{i=1}^{N} y_i \log(\Phi(x_i\beta)) + (1 - y_i) \log(\Phi(-x_i\beta)).\end{aligned}$$

c) Show that $-\log \Phi(a)$ is convex for all $a \in \mathbb{R}$.

solution We can do this for arbitrary a by showing that $\frac{\partial^2 - \log \Phi(a)}{\partial a^2} > 0$. Start with computing the first derivative:

$$\frac{\partial - \log(\Phi(a))}{\partial a} = -\frac{\phi(a)}{\Phi(a)},$$

and the second:

$$\begin{aligned}\frac{\partial^2 \log(\Phi(a))}{\partial a^2} &= -\frac{-a\phi(a)\Phi(a) - \phi(a)^2}{\Phi(a)^2} \\ &= \frac{\phi(a)}{\Phi(a)^2}[a\Phi(a) + \phi(a)].\end{aligned}$$

Since $\frac{\phi(a)}{\Phi(a)^2} \geq 0$ as it is the ratio of two probabilities, we only need to show that $a\Phi(a) + \phi(a) > 0$ to prove that $\frac{\partial^2 - \log \Phi(a)}{\partial a^2} > 0$. This can be done by noticing first that:

$$\frac{\partial}{\partial a} a\Phi(a) + \phi(a) = \Phi(a) > 0,$$

which implies that $a\Phi(a) + \phi(a)$ is an increasing function. Since $a \in \mathbb{R}$ we just need to show that the function stays positive as $a \to -\infty$ to show that it is always positive:

$$\begin{aligned}
\lim_{a\to-\infty} a\Phi(a) + \phi(a) &= \lim_{a\to-\infty} a\Phi(a) && \text{(Because } \lim_{a\to-\infty} \phi(a) = 0\text{)} \\
&= \lim_{a\to-\infty} \frac{\Phi(a)}{\frac{1}{a}} \\
&= \lim_{a\to-\infty} a^2\phi(a) && \text{(By l'Hopital's rule)} \\
&= \lim_{a\to-\infty} a^2 \frac{1}{\sqrt{2\pi}} e^{-\frac{a^2}{2}} && \text{(By def. of } \phi(a)\text{)} \\
&= \lim_{a\to-\infty} \frac{a^2}{\sqrt{2\pi} e^{\frac{a^2}{2}}} \\
&= \lim_{a\to-\infty} \frac{2a}{\sqrt{2\pi} a e^{\frac{a^2}{2}}} && \text{(By L'Hopital's rule)} \\
&= \lim_{a\to-\infty} 2\phi(a) = 0.
\end{aligned}$$

Because of this, we can conclude that $a\Phi(a) + \phi(a) > 0 \, \forall a \in \mathbb{R}$, and, therefore, that $\frac{\partial^2 - \log \Phi(a)}{\partial a^2} > 0$, which shows convexity of $-\log \Phi(a)$.

d) Since $-\log \mathcal{L}(\beta)$ is convex we can use gradient descent to find $\beta^* \in \arg\min_{\beta} [-\mathcal{L}(\beta)]$. Derive the needed gradient.

solution Start with the negative likelihood expression:

$$\begin{aligned}
\frac{\partial \log \mathcal{L}(\beta)}{\partial \beta} &= \sum_{i=1}^{N} \frac{y_i \frac{\partial \Phi(x\beta)}{\partial \beta}}{\Phi(x\beta)} - \frac{(1-y_i)\frac{\partial \Phi(-x\beta)}{\partial \beta}}{\Phi(-x\beta)} \\
&= \sum_{i=1}^{N} \frac{y_i x_i^T \phi(x\beta)}{\Phi(x\beta)} + \frac{(1-y_i) x_i^T \phi(x\beta)}{\Phi(-x\beta)}.
\end{aligned}$$

Misc. hints

1. If $\Phi(a)$ is the standard normal CDF then $\phi(a)$ is the standard normal pdf evaluated at a point a and $\frac{\partial \Phi(a)}{\partial a} = \phi(a)$.
2. $\frac{\partial \phi(a)}{\partial a} = -a\phi(a)$.
3. $1 - \Phi(a) = \Phi(-a)$.
4. Show convexity by showing that $\frac{\partial^2 - \log(\Phi(a))}{\partial a^2} \geq 0$.
5. You can use the following fact without proof: $a\Phi(a) + \phi(a) > 0 \, \forall a$.

2 Adaboost with the L_2 loss

Consider a regression problem in which you are given data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, with $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$, as well as a collection of m weak classifiers $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$, each taking $\mathbf{x}$ as input and producing a real number as output. We would like to develop a version of AdaBoost that can work on this problem. To do so we choose to employ the squared L_2 loss:

$$L(\boldsymbol{\lambda}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda})^2, \tag{5}$$

where $h(\mathbf{x}_i) = \begin{bmatrix} h_1(\mathbf{x}_i) & \ldots & h_m(\mathbf{x}_i) \end{bmatrix}$, and $\boldsymbol{\lambda} \in \mathbb{R}^m$ is a vector of weights, one for each of the weak classifiers. We need to come up with coordinate descent to minimize this loss.

a) Following the derivation of the AdaBoost updates in the lecture notes, we start by choosing a optimal direction to move in. Show that:

$$\arg\max_{j=1,\ldots,m}\left[-\frac{\partial}{\partial\alpha}L(\boldsymbol{\lambda}+\alpha\mathbf{e}_j)\right]\Big|_{\alpha=0} = \arg\max_{j=1,\ldots,m} h_j^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\boldsymbol{\lambda}),$$

where: $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$, $h_j(\mathbf{X}) = \begin{bmatrix} h_j(\mathbf{x}_1) \\ \vdots \\ h_j(\mathbf{x}_n) \end{bmatrix}$, $H(\mathbf{X}) = \begin{bmatrix} h_1(\mathbf{x}_1) & \ldots & h_m(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_n) & \ldots & h_m(\mathbf{x}_n) \end{bmatrix}$, and $\mathbf{e}_j$ is a vector that is 0 in all entries and 1 in the j^{th}.

Solution :

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\left[-L(\boldsymbol{\lambda}+\alpha\mathbf{e}_j)\right]\Big|_{\alpha=0} &= \frac{\partial}{\partial\alpha}\left[-\frac{1}{n}\sum_{i=1}^{n}(y_i - h(\mathbf{x}_i)(\boldsymbol{\lambda}+\alpha\mathbf{e}_j))^2\right]\Big|_{\alpha=0} \\
&= \frac{\partial}{\partial\alpha}\left[-\frac{1}{n}\sum_{i=1}^{n}(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda})^2 - 2\alpha h_j(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda}) + \alpha^2 h_j^2(\mathbf{x}_i)\right]\Big|_{\alpha=0} \\
&= -\frac{1}{n}\sum_{i=1}^{n} -2h_j(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda}) + 2\alpha h_j^2(\mathbf{x}_i)\Big|_{\alpha=0} \\
&= \frac{1}{n}\sum_{i=1}^{n} 2h_j(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda}),
\end{aligned}$$

then:

$$\begin{aligned}
\arg\max_{j=1,\ldots,m}\left[-L(\boldsymbol{\lambda}+\alpha\mathbf{e}_j)\right]\Big|_{\alpha=0} &= \arg\max_{j=1,\ldots,m}\frac{1}{n}\sum_{i=1}^{n} 2h_j(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda}) \\
&= \arg\max_{j=1,\ldots,m}\sum_{i=1}^{n} h_j(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda}) \\
&= \arg\max_{j=1,\ldots,m} h_j^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\boldsymbol{\lambda}).
\end{aligned}$$

b) After choosing $j^{(t)}$, the optimal direction to move in at iteration t, which we found in part (a), we need to choose the optimal amount of shift in the weight in this direction. Show that this is given by:

$$\alpha^{(t)} = \frac{h_{j^{(t)}}^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\boldsymbol{\lambda})}{\sum_{i=1}^n h_{j^{(t)}}^2(\mathbf{x}_i)}.$$

solution The optimal update for weight $j^{(t)}$ can be obtained by plugging $j^{(t)}$ into the loss equation and then by maximizing it with respect to α, that is:

$$\alpha^{(t)} = \arg\min_{\alpha} L(\boldsymbol{\lambda} + \alpha \mathbf{e}_{j^{(t)}}).$$

This is easy to solve by setting the derivative of the loss derived in part (a) equal to 0 and solving for α:

$$\begin{aligned}
0 &= \frac{\partial}{\partial \alpha} L(\boldsymbol{\lambda} + \alpha \mathbf{e}_{j^{(t)}}) \\
&= -\frac{1}{n}\sum_{i=1}^n 2h_{j^{(t)}}(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda}) + 2\alpha h_{j^{(t)}}^2(\mathbf{x}_i) && \text{(Plugging } j^{(t)} \text{ into the steps in part a)} \\
&= -\sum_{i=1}^n h_{j^{(t)}}(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda}) + \sum_{i=1}^n \alpha h_{j^{(t)}}^2(\mathbf{x}_i) \\
&= -\frac{\sum_{i=1}^n h_{j^{(t)}}(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda})}{\sum_{i=1}^n h_{j^{(t)}}^2(\mathbf{x}_i)} + \alpha \\
\implies \alpha &= \frac{\sum_{i=1}^n h_{j^{(t)}}(\mathbf{x}_i)(y_i - h(\mathbf{x}_i)\boldsymbol{\lambda})}{\sum_{i=1}^n h_{j^{(t)}}^2(\mathbf{x}_i)} = \frac{h_{j^{(t)}}^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\boldsymbol{\lambda})}{\sum_{i=1}^n h_{j^{(t)}}^2(\mathbf{x}_i)}.
\end{aligned}$$

c) Using the results obtained in parts a and b write down the pseudocode for a version of Adaboost with the squared L_2 loss given in (5). State the formula for making predictions using the new algorithm.

solution AdaBoostL2

- Initialize $\boldsymbol{\lambda}^{(0)T} = [0, \ldots, 0]$
- For $t = 1, \ldots, T$ iterations, repeat.
  - $j^{(t)} := \arg\max_{j=1,\ldots,m} h_j^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\boldsymbol{\lambda})$
  - $\alpha^{(t)} := \frac{h_{j^{(t)}}^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X}))}{\sum_{i=1}^n h_{j^{(t)}}^2(\mathbf{x}_i)}$
  - $\boldsymbol{\lambda}^{(t+1)} := \boldsymbol{\lambda}^{(t)} + \alpha^{(t)} \mathbf{e}_{j^{(t)}}$
- For a new point $\tilde{x}$, predict $\hat{\tilde{y}} = H(\tilde{x})\boldsymbol{\lambda}^{(T)}$

3 Comparing AdaBoost and Logistic Regression

In this question we'll compare the performance of AdaBoost to the performance of logistic regression on the same real-world data. Download the congressional records data from sakai. This dataset contains voting records of members of Congress in the 1984 parliament on 16 key bills, as well as the party each congressperson belonged to. We are going to use party as label, and votes on bills as binary features. That is, we will predict which party the congressperson belonged to based on his/her voting record.

a) Implement the version of AdaBoost introduced in the lecture notes, using the features themselves as weak classifiers, *i.e.* $h_j(\mathbf{x}_i) = x_{ij}$.

solution

```
def AdaBoost(HX, y, n_iter = 1000):
    N = len(y)
    P = np.shape(HX)[1]
    d = np.repeat(1.0/N, N)
    Lambda = np.repeat(0.0, P)
    M = np.array([y * HX[:, j] for j in xrange(P)]).transpose()
    for t in xrange(n_iter):
        jt = np.argmax(np.dot(d, M))
        d_ = sum(d[M[:, jt] == -1])
        alpha = 0.5 * np.log((1 - d_)/d_)
        Lambda[jt] = Lambda[jt] + alpha
        d = np.exp(-np.dot(M, Lambda)) / sum(np.exp(-np.dot(M, Lambda)))
    return Lambda

def AdaBoost_predict_prob(HX, Lambda):
    return np.exp(2 * np.dot(HX, Lambda))/ (1 + np.exp(2 * np.dot(HX, Lambda)))

## Train Model
Lambda = AdaBoost(X, y, n_iter =3000)

## Predict
ada_yhat = AdaBoost_predict_prob(X_train, Lambda)
ada_yhat_test = AdaBoost_predict_prob(X_test, Lambda)
```

b) Implement logistic regression for the Congress voting data by maximizing the log-likelihood. You don't have to implement your own maximization procedure, you can use any optimizer you want.

solution

```
def logistic_loss(theta, *args):
    X = args[0]
    y = args[1]
    return np.sum(np.log(1 + np.exp(-y * np.dot(X, theta))))
```

```
def logistic_gradient(theta, X, y):
    N = - y * X.transpose() * np.exp(-y * np.dot(X, theta))
    D = 1 + np.exp(-y * np.dot(X, theta))
    return np.sum(N/D, axis=1)

def gradient_descent(X, y, loss, gradient, lr, epsilon = 1e-5, n_iter=1000):
    P = np.shape(X)[1]
    theta = np.repeat(0.01, P)
    i = 0
    prev = np.repeat(0.0, P)
    while i < n_iter and np.sum(np.abs(theta - prev)) > epsilon:
        l = loss(theta, X, y)
        i += 1
        prev = theta
        theta = theta - lr * gradient(theta, X, y)
    return theta

def logit_predict_prob(X, theta):
    return 1.0/(1 + np.exp(-np.dot(X, theta)))

## Train
theta = gradient_descent(X, y, logistic_loss, logistic_gradient, 0.01)

## Predict
logit_yhat = logit_predict_prob(X_train, Lambda)
logit_yhat_test = logit_predict_prob(X_test, Lambda)
```

c) Compare the algorithms' performance on the data. Use evaluation methods we have covered in class to answer this question.

solution The two methods perform similarly on the data: logistic regression seems to have a slight advantage on the training set, but they both do equally as well on the test set. This is shown by the ROC curves plotted below:

d) Which bills seem to be more important for predicting partisanship in Congress? Justify your answer.

Misc. Hints

- Most languages come with packages that implement optimization methods for generic convex functions. For example, in R, this is the optim function, in Matlab, the function fminsearch or the cvx toolbox can be used, and in Python the minimize function in the scipy.optimize package does the trick.
- You don't necessarily have to maximize the log likelihood, you can optimize any related quantity as long as the solution is the same as maximizing the log-likelihood.

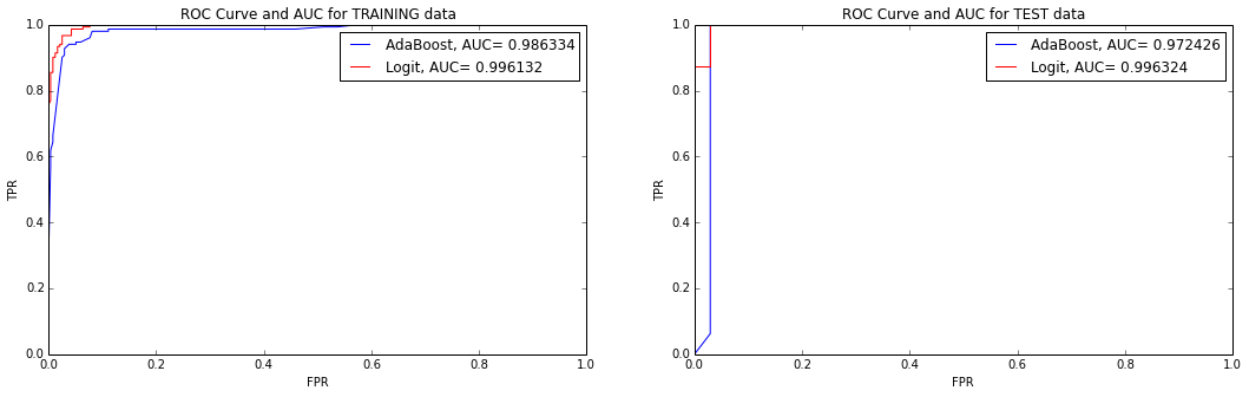

Figure 1: ROC curves for the Congress data