CS689: Machine Learning - Fall 2019

Homework 1

Sankaran Vaidyanathan (svaidyanatha@cs.umass.edu)

1. Optimal Predictions for Zero-One Loss

a. Using the given loss criterion, we can find the optimal prediction function f_* as follows:

$$\begin{aligned} f_* &= \arg\min_f \mathbb{E}_{P(x,y)}[L_{01}(y, f(x))] \\ &= \arg\min_f \int_{x \in R} \sum_{y \in \{0,1\}} L_{01}(y, f(x)) P(X = x, Y = y) dx \\ &= \arg\min_f \mathbb{E}_{P(x)} \Big[\mathbb{E}_{P(y|x)} \big[L_{01}(y, f(x)) | x \big] \Big] \end{aligned}$$

With respect to the minimizer, all values for f besides that corresponding to the given x will be treated as a constant and be ignored. Hence we can write the optimal prediction for a given x as follows:

$$\begin{aligned} f_*(x) &= \arg\min_f \mathbb{E}_{P(y|x)} \big[L_{01}(y, f(x)) \big] \\ &= \arg\min_f \sum_{y \in \{0,1\}} L_{01}(y, f(x)) P(Y = y | X = x) \end{aligned}$$

Here, $L_{01}(y, f(x)) = \begin{cases} 0 & \text{if } y = f(x) \\ 1 & \text{if } y \neq f(x) \end{cases} = 1 - [y = f(x)]$

$$\implies f_*(x) = \arg\min_f \sum_{y \in \{0,1\}} \big(1 - [y = f(x)]\big) P(Y = y | X = x)$$

b. We have:

$$\begin{aligned} f_*(x) &= \arg\min_f \sum_{y \in \{0,1\}} L_{01}(y, f(x)) P(Y = y | X = x) \\ &= \arg\min_f \sum_{y \in \{0,1\}} (1 - [y = f(x)]) P(Y = y | X = x) \\ &= \arg\min_f \left\{ \sum_{y \in \{0,1\}} P(Y = y | X = x) - \sum_{y \in \{0,1\}} [y = f(x)] P(Y = y | X = x) \right\} \\ &= \arg\min_f \left\{ 1 - \sum_{y \in \{0,1\}} [y = f(x)] P(Y = y | X = x) \right\} \end{aligned}$$

$$
\begin{aligned}
&= \arg\min_f \left\{ \sum_{y\in\{0,1\}} [y = f(x)] - \sum_{y\in\{0,1\}} [y = f(x)]P(Y = y|X = x) \right\} \\
&= \arg\min_f \left\{ \sum_{y\in\{0,1\}} [y = f(x)](1 - P(Y = y|X = x)) \right\} \\
&= \arg\max_f \left\{ \sum_{y\in\{0,1\}} [y = f(x)]P(Y = y|X = x) \right\} \\
&= \arg\max_{y\in\{0,1\}} P(Y = y|X = x)
\end{aligned}
$$

The last expression appears because the term in the sum goes to zero if $y \neq f(x)$; when $y = f(x)$, the term is maximized by the value of y with highest conditional probability $P(Y = y|X = x)$.

---

2. Logistic Regression with Centering:

a. We have the regularized negative conditional log likelihood given by:

$$\mathcal{L}(\mathcal{D}, \theta) = \sum_{n=1}^{N} \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + \lambda\|\mathbf{w}\|_2^2 + \lambda\|\mathbf{c}\|_2^2 + \lambda \cdot b^2$$

$$
\begin{aligned}
\nabla_{\mathbf{w}}\mathcal{L}(\mathcal{D}, \theta) &= \nabla_{\mathbf{w}}\Big\{ \sum_{n=1}^{N} \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + \lambda\|\mathbf{w}\|_2^2 + \lambda\|\mathbf{c}\|_2^2 + \lambda \cdot b^2 \Big\} \\
&= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + \lambda\nabla_{\mathbf{w}}\|\mathbf{w}\|_2^2 \\
&= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + \lambda\nabla_{\mathbf{w}}(\mathbf{w}\mathbf{w}^T) \\
&= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + 2\lambda\mathbf{w} \\
&= \sum_{n=1}^{N} \frac{1}{1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))} \nabla_{\mathbf{w}}(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + 2\lambda\mathbf{w} \\
&= \sum_{n=1}^{N} \frac{\exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))}{1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))} \nabla_{\mathbf{w}}(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b)) + 2\lambda\mathbf{w} \\
&= \sum_{n=1}^{N} \frac{\exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))(-y_n(\mathbf{x}_n - \mathbf{c}))}{1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))} + 2\lambda\mathbf{w} \\
&= \sum_{n=1}^{N} \frac{-y_n \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))}{1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))} (\mathbf{x}_n - \mathbf{c}) + 2\lambda\mathbf{w}
\end{aligned}
$$

$$\nabla_{\mathbf{c}}\mathcal{L}(\mathcal{D}, \theta) = \nabla_{\mathbf{c}}\Big\{ \sum_{n=1}^{N} \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + \lambda\|\mathbf{w}\|_2^2 + \lambda\|\mathbf{c}\|_2^2 + \lambda \cdot b^2 \Big\}$$

$$
\begin{aligned}
&= \sum_{n=1}^{N} \nabla_{\mathbf{c}} \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + \lambda \nabla_{\mathbf{c}} \|\mathbf{c}\|_2^2 \\
&= \sum_{n=1}^{N} \frac{\exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))}{1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))} \nabla_{\mathbf{c}}(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b)) + 2\lambda \mathbf{c} \\
&= \sum_{n=1}^{N} \frac{\exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))(-y_n(-\mathbf{w}))}{1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))} + 2\lambda \mathbf{c} \\
&= \sum_{n=1}^{N} \frac{-y_n \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))}{1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))} (-\mathbf{w}) + 2\lambda \mathbf{c}
\end{aligned}
$$

Though b is a scalar, the ∇ operator is used to denote its partial derivative for consistency of notation.

$$
\begin{aligned}
\nabla_b \mathcal{L}(\mathcal{D}, \theta) &= \nabla_b \Big\{ \sum_{n=1}^{N} \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + \lambda \|\mathbf{w}\|_2^2 + \lambda \|\mathbf{c}\|_2^2 + \lambda \cdot b^2 \Big\} \\
&= \sum_{n=1}^{N} \nabla_b \log(1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))) + \lambda \nabla_b b^2 \\
&= \sum_{n=1}^{N} \frac{-y_n \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))}{1 + \exp(-y_n(\mathbf{w}(\mathbf{x}_n - \mathbf{c})^T + b))} + 2\lambda b
\end{aligned}
$$

b. The main issues encountered when implementing this model were with vectorization and broadcasting in numpy. As a sanity check for my equations, I initially decided to start with a naive iterative implementation for reference. This turned out not to be as helpful as I thought, since the accumulated differences in precision seemed to be enough to be marked incorrectly on the autograders. My vectorized implementation gave the correct shapes but either incorrect gradients or overflow errors. Surprisingly, the final results turned out correctly when I fixed a broadcasting issue in a Hadamard product of a vector with a matrix (line 108 of `augmented_logistic_regression.py`), by adding an axis and following the row-vector convention established in class. The code worked without a need for logsumexp or other numerical stability fixes.

c. The average test set prediction error for my implementation of augmented logistic regression was 0.02465.

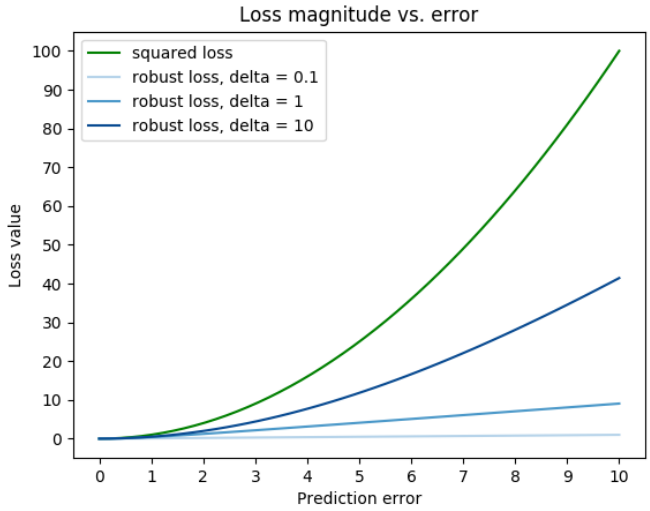
d. The average test set prediction error for sklearn's implementation of logistic regression was 0.02304.

e. The distance of the mean of the data X from the origin $\mathbf{0}$ was 6.795, and the distance of the mean from the new center $\mathbf{c}$ was 6.444. The distance from $\mathbf{c}$ to the origin $\mathbf{0}$ was 1.1246. Switching off the centering by setting $\mathbf{c}$ and its gradient to zero gave a similar prediction error of 0.02358, close to the sklearn model. The model did not learn to mean-center the data; in fact, the center it learned is closer to the origin than the mean.

We can conclude that *the purpose of mean-centering for a model linear in its input is not for the improvement of prediction accuracy*. Because the model is both linear in the weights and the input, the centering of $\mathbf{x}$ has no effect on the slope and simply shifts the intercept, since $y = \mathbf{w}(\mathbf{x} - \mathbf{c})^T + b = \mathbf{w}\mathbf{x}^T + (b - \mathbf{w}\mathbf{c}^T)$ for some constant $\mathbf{c}$. The change in y with respect to change in $\mathbf{x}$ is unaffected by $\mathbf{c}$, and the center is simply a point of reference. Mean-centering could be a way of interpreting the weights as magnitudes of deviation from the reference point for a unit change in $\mathbf{x}$, while leaving the predictions intact.

3. Regression with Outliers

a. The following plot shows the loss value as a function of the prediction error $y - y'$.



For the squared error loss function, we can see that the error exhibits quadratic growth; the loss grows slowly when the prediction error is small, but blows up as the prediction error grows large. This can result in sensitivity to outliers, as the contribution of a single training example with large error to the overall loss can outweigh that of many training examples with low error.
The robust loss function appears to have quadratic growth for small prediction error, albeit scaled down. As the prediction error grows larger, the loss appears to approach a linear curve. This could control the blowup in loss caused by outliers, while preserving the characteristics of the loss curve for small errors.

b. We have the learning objective given by:

$$\mathcal{L}_\delta(\mathcal{D}, \theta) = \sum_{n=1}^{N} L_\delta(y_n, \mathbf{w}\mathbf{x}_n^T + b) = \sum_{n=1}^{N} \delta^2(\sqrt{1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2} - 1)$$

$$\begin{aligned}
\nabla_{\mathbf{w}}\mathcal{L}_\delta(\mathcal{D}, \theta) &= \delta^2 \sum_{n=1}^{N} \nabla_{\mathbf{w}}(\sqrt{1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2} - 1) \\
&= \delta^2 \sum_{n=1}^{N} \frac{1}{2\sqrt{1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2}} \nabla_{\mathbf{w}}(1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2) \\
&= \delta^2 \sum_{n=1}^{N} \frac{1}{2\sqrt{1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2}} \frac{2(y_n - \mathbf{w}\mathbf{x}_n^T - b)}{\delta^2} \nabla_{\mathbf{w}}(y_n - \mathbf{w}\mathbf{x}_n^T - b) \\
&= \sum_{n=1}^{N} \frac{y_n - \mathbf{w}\mathbf{x}_n^T - b}{\sqrt{1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2}} (-\mathbf{x}_n)
\end{aligned}$$

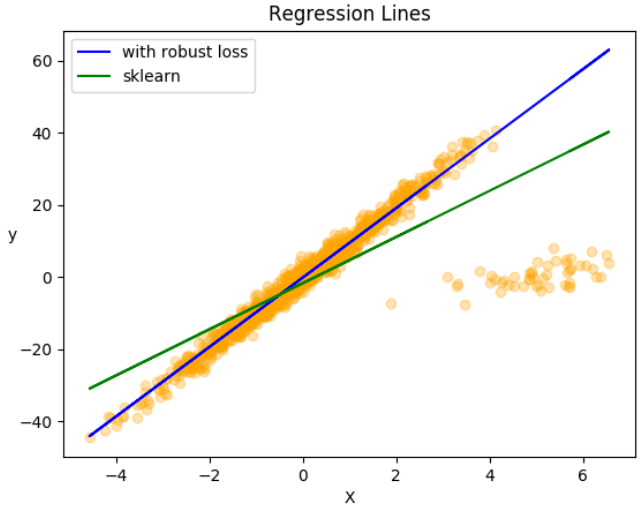
If the bias absorption trick is used, the above equation is sufficient to implement in code.

$$\begin{aligned}\nabla_b \mathcal{L}_\delta(\mathcal{D}, \theta) &= \delta^2 \sum_{n=1}^{N} \nabla_b (\sqrt{1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2} - 1) \\ &= \delta^2 \sum_{n=1}^{N} \frac{1}{2\sqrt{1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2}} \frac{2(y_n - \mathbf{w}\mathbf{x}_n^T - b)}{\delta^2} \nabla_b (y_n - \mathbf{w}\mathbf{x}_n^T - b) \\ &= \sum_{n=1}^{N} \frac{y_n - \mathbf{w}\mathbf{x}_n^T - b}{\sqrt{1 + (y_n - \mathbf{w}\mathbf{x}_n^T - b)^2/\delta^2}} (-1)\end{aligned}$$

c. The issues I faced with this were similar to those in Q2: broadcasting and vectorization. Initially, both the objective and gradient did not work correctly, and as before an iterative reference implementation did not help. Ensuring consistency in row-vector notation, fixing Hadamard products and checking for correct shapes and axes in sums led to a correct implementation.

d. The MSE achieved by the augmented linear regression model on the training set is 114.89058. The MSE achieved by sklearn's linear regression implementation is 76.04311.

e. The following plot shows the regression lines found using the robust model with $\delta = 1$, and sklearn's default implementation of Linear Regression.



f. Examining this plot, it appears that the data is largely clustered along the direction given by the blue line, the center of which is roughly at the intersection of the two lines. There is also a small set of outlier points. Based on the data distribution, it can be assumed that a new datapoint is more likely to appear along the direction of the majority than as an outlier, and hence the line given by the robust model would generalize better. The default linear regression model minimizes the squared error aggregated from *all* the points, and is reoriented by the outlier points. A test point lying along the direction of the majority cluster would end up having greater error as the distance from the cluster center increases. Because of this, the generalization performance would not be as good as that of the robust model.