MATP 6600 Programming Project

Gabe Mancino-Ball

Fall 2018 - RPI

1 Problem

Given a set of training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where $y_i \in \{-1, 1\}$ for each $i = 1, 2, \ldots, m$, logistic regression can be used as a binary classifier. In this project, the methods of Newton, steepest gradient descent, and stochastic gradient descent are used on the datasets `gisette.mat` and `spamData.mat` to perform such classification.

2 Theoretical Approach

The model is formulated as

$$\min_{\mathbf{w},b} f(\mathbf{w}, b) := \frac{1}{m}\sum_{i=1}^{m} \log\left(1 + \exp[-y_i(\mathbf{w}^T\mathbf{x}_i + b)]\right) + \frac{\lambda_1}{2}\|\mathbf{w}\|_2^2 + \frac{\lambda_2}{2}b^2 \tag{1}$$

where $\lambda_1 \geq 0, \lambda_2 \geq 0, b \in \mathbb{R}, y_i \in \{-1, 1\}$, and $\mathbf{x}_i, \mathbf{w} \in \mathbb{R}^n$ for $i = 1, 2, \ldots, m$. Once a suitable $(\bar{\mathbf{w}}, \bar{b})$ have been found for (1), the hyperplane $\bar{\mathbf{w}}^T\mathbf{x}_{new} + \bar{b}$ is used to classify any new data.

2.1 Gradient and Hessian

The gradient and the Hessian matrix are necessary for the algorithms used. Using (1), the components of the gradient are formulated as

$$\begin{aligned}
\frac{\partial}{\partial w_j} f &= \frac{1}{m}\sum_{i=1}^{m} \frac{-y_i x_{(i,j)}}{\exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)] + 1} + \lambda_1 w_j \\
\frac{\partial}{\partial b} f &= \frac{1}{m}\sum_{i=1}^{m} \frac{-y_i}{\exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)] + 1} + \lambda_2 b
\end{aligned}$$

for $j = 1, 2, \ldots, n$. Similarly, the components of the Hessian are formulated as

$$\begin{aligned}
\frac{\partial^2}{\partial w_j \partial w_k} f &= \frac{1}{m}\sum_{i=1}^{m} \frac{x_{(i,j)}x_{(i,k)} \exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)]}{\left(\exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)] + 1\right)^2} \\
\frac{\partial^2}{\partial w_j \partial b} f &= \frac{\partial^2}{\partial b \partial w_j} = \frac{1}{m}\sum_{i=1}^{m} \frac{x_{(i,j)} \exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)]}{\left(\exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)] + 1\right)^2} \\
\frac{\partial^2}{\partial w_j^2} f &= \frac{1}{m}\sum_{i=1}^{m} \frac{x_{(i,j)}^2 \exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)]}{\left(\exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)] + 1\right)^2} + \lambda_1 \\
\frac{\partial^2}{\partial b^2} f &= \frac{1}{m}\sum_{i=1}^{m} \frac{\exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)]}{\left(\exp[y_i(\mathbf{w}^T\mathbf{x}_i + b)] + 1\right)^2} + \lambda_2.
\end{aligned}$$

3 Numerical Results

Using the method of steepest gradient descent, Newton's method, and stochastic gradient descent, numerical experiments are implemented in MATLAB. See Listings (5), (6), and (7) for the code for these methods, respectively. Steepest gradient descent algorithm uses the method of *backtracking* coupled with Armijo's Rule, which is outlined in [1], for determining the step size at each iteration; here $\sigma = 0.5$ and $\alpha_{new} = 0.75\alpha_{old}$. Pure Newton's Method is used here (i.e. $\alpha = 1$). For the stochastic gradient descent method, a minibatch of 100 samples is taken for each iteration; the step size is given by $\alpha_k = \left(\frac{5}{6}\right)\frac{1}{k}$. Using the data sets `spamData.mat` and `gisette.mat` with the following termination rule:

$$\|\nabla(f(\mathbf{w}^{(k)}, b^{(k)}))\|_2 \le \epsilon \max\{1, \|(\mathbf{w}^{(k)}, b^{(k)})\|_2\}$$

the algorithms are tested with $\epsilon \in \{10^{-2}, 10^{-4}, 10^{-6}\}$. For these experiments, $\lambda_1 = \lambda_2 = 0.001$ (see section 3.2 for results with varied λ_i, $i = 1, 2$). The initial guesses for $(\bar{\mathbf{w}}, \bar{b})$ are given by $(\mathbf{w}^{(0)}, b^{(0)}) = (\mathbf{0}, 0)$ for all of the experiments. The results are summarized in the following tables below.

Table 1: Results for the Spam dataset

Method	Tolerance	Total Iteration Number	Total Run Time	Classification Accuracy
Steepest GD	10^{-2}	10000	138.66 seconds	87.19%
Newton's	10^{-2}	7	0.5 seconds	94.34%
Steepest GD	10^{-4}	25000	368.72 seconds	90.18%
Newton's	10^{-4}	9	0.64 seconds	94.34%
Steepest GD	10^{-6}	50000	782.35 seconds	91.18%
Newton's	10^{-6}	9	0.67 seconds	94.34%
Stochastic GD	-	2000	8.5 seconds	91.51%

Table 2: Results for the Gisette dataset

Method	Tolerance	Total Iteration Number	Total Run Time	Classification Accuracy
Steepest GD	10^{-2}	183	8.21 seconds	93.3%
Newton's	10^{-2}	6	13.07 seconds	93.4%
Steepest GD	10^{-4}	952	27.85 seconds	93.4%
Newton's	10^{-4}	10	25.05 seconds	93.3%
Steepest GD	10^{-6}	5814	128.2 seconds	93.3%
Newton's	10^{-6}	11	26.25 seconds	93.3%
Stochastic GD	-	5000	136.87 seconds	92%

Note

The functions in Listings (1), (2), and (3) had originally been made with nested `for` loops; thanks to Robben Teufel, they are now vectorized. The MATLAB file `RunMe.m` will output the results for $\epsilon = 10^{-2}$ in about 5 minutes.

3.1 Plots of Objective Function

The following plots visually display how (1) is decreasing with each iteration with respect to the given method. Each plot shows the log of the difference between the computed objective value and the optimal objective value which is computed with Newton's method (with $\epsilon = 10^{-6}$). The plots below only show the results for $\epsilon = 10^{-6}$, since these plots will be the most informative.

Figure 1: Comparison of Algorithms on Spam Data

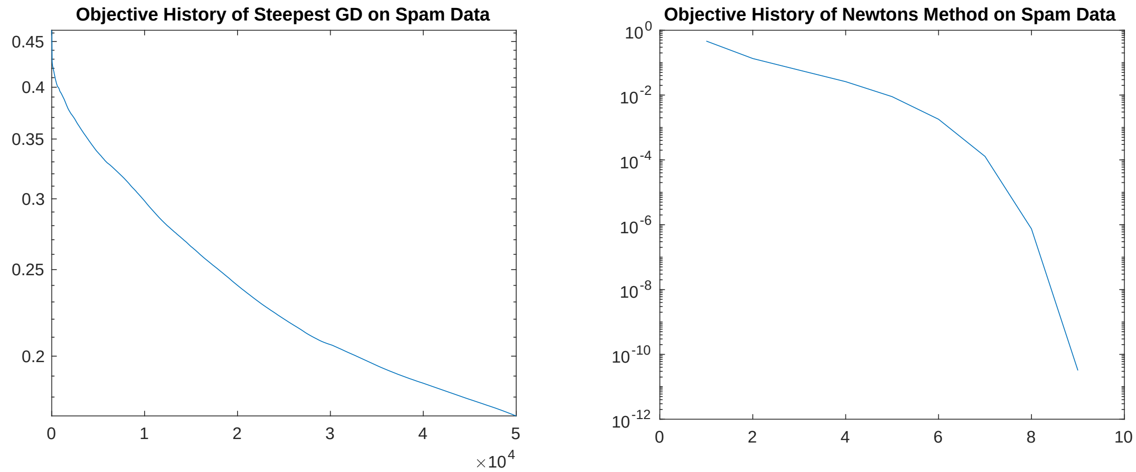


Figure 2: Comparison of Algorithms on Gisette Data

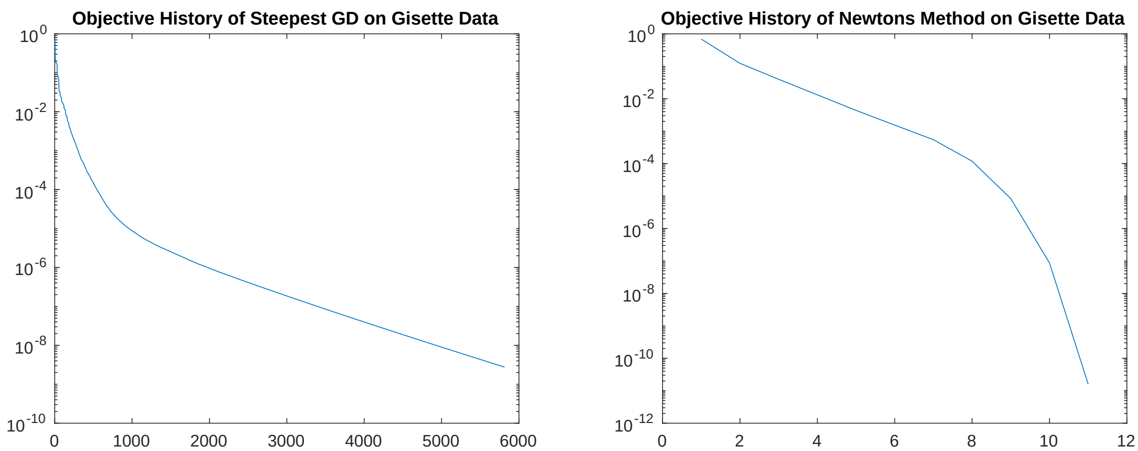
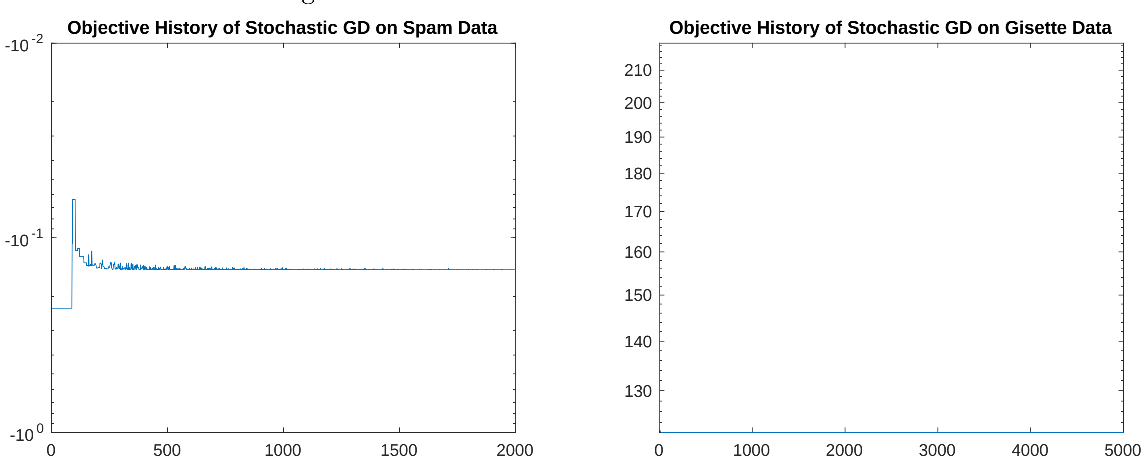


Figure 3: Stochastic Gradient Descent on each Dataset



3.2 Varying λ_1 and λ_2: Impact on Classification Accuracy

The parameters $\lambda_i, \;\; i = 1, 2$, are varied in this section for the gradient descent method and Newton's method. For each test, the initial guess of $(\mathbf{w}^{(0)}, b^{(0)}) = (\mathbf{0}, 0)$ is made along with a tolerance of $\epsilon = 10^{-2}$. The resulting change in classification accuracy is summarized in the tables below.

Table 3: Results for the Spam dataset

Method	λ_1	λ_2	Run Time	Classification Accuracy
Steepest GD	0.1	0.1	45.48 seconds	84.52%
Newton's	0.1	0.1	0.5 seconds	91.51%
Steepest GD	0.005	0.05	40.6 seconds	84.52%
Newton's	0.005	0.05	0.43 seconds	93.84%
Steepest GD	1	0.0075	45.33 seconds	84.02%
Newton's	1	0.0075	0.53 seconds	79.36%

Table 4: Results for the Gisette dataset

Method	λ_1	λ_2	Run Time	Classification Accuracy
Steepest GD	0.1	0.1	9.28 seconds	93.1%
Newton's	0.1	0.1	14.53 seconds	93.4%
Steepest GD	0.005	0.05	8.5 seconds	93.3%
Newton's	0.005	0.05	12.96 seconds	93.6%
Steepest GD	1	0.0075	7.55 seconds	92.3%
Newton's	1	0.0075	11.63 seconds	92.3%

Using the tables above, it is clear that subtle changes in λ_1 and λ_2 had a greater impact on classification accuracy of the Spam dataset, for both steepest gradient descent and Newton's method. The Gisette dataset was more resilient to the affect of changing λ_1 and λ_2. For plots of how (1) is changing, see Figure (3) at the end of this document.

4 Observations and Conclusion

In terms of performance, Newton's method was the superior algorithm; setting $\epsilon = 10^{-16}$ yielded a numerically optimal solution to (1) for both data sets. The steepest gradient descent algorithm did not converge for the Spam dataset, even after 50,000 iterations, this could be due to the step size at each iteration. For the stochastic gradient descent, the maximum number of iterations was set to a low value since the testing accuracy was above 90% for all trials. Overall, the methods have their advantages and disadvantages, but a large part of this is due to the given dataset. Comparison of algorithms, as done in this project, is necessary for determining an optimal algorithm to use on new data.

Note

Please note that the `Warning: Negative data ignored` comes from plotting the objective values on a logarithmic scale.

References

[1] Mokhtar Bazaraa, Hanif Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms.* John Wiley & Sons, Inc., 2006.

MATLAB Code

Listing 1: Code for Objective Function

```
%-----------------------------------------------------------------------
% Objective Function for LR
% (vectorized)
%
% Inputs:
% X(i,:) — ith data point
% y — vector of classification results
% w — normal vector to hyperplane
% b — scalar in hyperplane equation
% lambda1 — tuning parameter
% lambda2 — tuning parameter
%
% Outputs:
% f — function value
%-----------------------------------------------------------------------
function f = UpdatedObjLR(X, y, w, b, lambda1, lambda2)
    [m,n] = size(X);
    f = 1/m*(sum(log(1+exp(-y.*(X*w+b)))))+0.5*lambda1*(w'*w)+0.5*lambda2*b^2;
end
```

Listing 2: Code for Gradient of Objective Function

```
%-----------------------------------------------------------------------
% Gradient of Objective Function for LR
% (vectorized)
%
% Inputs:
% X(i,:) — ith data point
% y — vector of classification results
% w — normal vector to hyperplane
% b — scalar in hyperplane equation
% lambda1 — tuning parameter
% lambda2 — tuning parameter
%
% Outputs:
% gradf — gradient of function evaluated at w and b
%-----------------------------------------------------------------------
function gradf = UpdatedGradLR(X, y, w, b, lambda1, lambda2)
    [m,n] = size(X);
    gradf = zeros(n+1,1);
    gradf(1:n,1) = 1/m*sum((-y.*X./(exp(y.*(X*w+b))+1)))' + lambda1*w;
    gradf(n+1,1) = 1/m*sum((-y./(exp(y.*(X*w+b))+1))) + lambda2*b;
end
```

Listing 3: Code for Hessian of Objective Function

```
%------------------------------------------------
% Hessian of Objective Function for LR
% (vectorized)
%
% Inputs:
% X(i,:) - ith data point
% y - vector of classification results
% w - normal vector to hyperplane
% b - scalar in hyperplane equation
% lambda1 - tuning parameter
% lambda2 - tuning parameter
%
% Outputs:
% H - hessian matrix of f
%------------------------------------------------
function H = UpdatedHessLR(X, y, w, b, lambda1, lambda2)
    [m,n] = size(X);
    H = zeros(n+1,n+1);
    evec = exp(y.*(X*w+b))./(1+exp(y.*(X*w+b))).^2;
    diage = diag(evec);
    H(1:n,1:n) = (1/m)*(X'*diage)*X + eye(n)*lambda1;
    H(1:n,n+1) = (1/m)*sum(X.*evec)';
    H(n+1,1:n) = H(1:n,n+1)';
    H(n+1,n+1) = (1/m)*sum(evec) + lambda2;
end
```

Listing 4: Code for Classifying New Data

```
%------------------------------------------------
% Classification Function for Logistic Regression
%
% Inputs:
% X(i,:) - ith data point
% w - normal vector for hyperplane
% b - scalar for hyperplane
%
% Outputs:
% y - vector with -1 and 1 as components
%------------------------------------------------
function y = ClassLR(X, w, b)
    [m,n] = size(X);
    y = zeros(m,1);
    for i=1:m
       if X(i,:)*w+b >= 0
           y(i,1) = 1;
       else
           y(i,1) = -1;
       end
    end
end
```

Listing 5: Steepest Gradient Descent

```
%-------------------------------------
% Steepest Gradient Descent for Logistic Regression
%
% Inputs:
% X(i,:) - ith data point as a row vector
% y - {-1, +1} classifier
% w - initial guess for w
% b - initial guess for b
% lambda1 - tuning parameter
% lambda2 - tuning parameter
% maxit - max number of iteration
% tol - tolerance
%
% Outputs:
% w - normal vector for hyperplane
% b - scalar for hyperplane
% hist_obj - history of objective value
% iter - number of iterations
%-------------------------------------
function [w, b, iter, hist_obj] =...
    SteepGD(X, y, w, b, lambda1, lambda2, maxit, tol)
    [m,n] = size(X);
    iter = 1;
    grad = UpdatedGradLR(X, y, w, b, lambda1, lambda2);
    obj = UpdatedObjLR(X, y, w, b, lambda1, lambda2);
    hist_obj = obj;
    while iter < maxit && norm(grad(1:n,1))+norm(grad(n+1,1))...
            >= tol*max(1, norm(w)+norm(b))

        % Loop for alpha using backtracking
        alpha = 1;
        while UpdatedObjLR(X, y, w - alpha*grad(1:n,1),...
                b - alpha*grad(n+1,1), lambda1, lambda2) - ...
                UpdatedObjLR(X, y, w, b, lambda1, lambda2) >= 0.5*alpha*...
                grad'*(-grad)
            alpha = 0.75*alpha;
        end

        % Update w and b
        w = w - alpha*grad(1:n,1);
        b = b - alpha*grad(n+1,1);

        % Update gradient
        grad = UpdatedGradLR(X, y, w, b, lambda1, lambda2);

        % Update objective
        obj = UpdatedObjLR(X, y, w, b, lambda1, lambda2);
        hist_obj = [hist_obj; obj];
        iter = iter + 1;
    end
end
```

Listing 6: Newton's Method

```
%------------------------------------------------
% Newton's Method for Logistic Regression
%
% Inputs:
% X(i,:) - ith data point as a row vector
% y - {-1, +1} classifier
% w - initial guess for w
% b - initial guess for b
% lambda1 - tuning parameter
% lambda2 - tuning parameter
% maxit - max number of iteration
% tol - tolerance
%
% Outputs:
% w - normal vector for hyperplane
% b - scalar for hyperplane
% hist_obj - history of objective value
% iter - number of iterations
%------------------------------------------------
function [w, b, iter, hist_obj] =...
    Newton(X, y, w, b, lambda1, lambda2, maxit, tol)
    [m,n] = size(X);
    iter = 1;
    grad = UpdatedGradLR(X, y, w, b, lambda1, lambda2);
    H = UpdatedHessLR(X, y, w, b, lambda1, lambda2);
    obj = UpdatedObjLR(X, y, w, b, lambda1, lambda2);
    hist_obj = obj;
    % Pure Newton's
    alpha = 1;

    while iter < maxit && norm(grad(1:n,1))+norm(grad(n+1,1))...
            >= tol*max(1, norm(w)+norm(b))

        % Find descent direction
        d = linsolve(H, grad);

        % Update w and b
        w = w - alpha*d(1:n,1);
        b = b - alpha*d(n+1,1);

        % Update hessian and gradient
        H = UpdatedHessLR(X, y, w, b, lambda1, lambda2);
        grad = UpdatedGradLR(X, y, w, b, lambda1, lambda2);

        % Update objective
        obj = UpdatedObjLR(X, y, w, b, lambda1, lambda2);
        hist_obj = [hist_obj; obj];
        iter = iter + 1;
    end
end
```

Listing 7: Stochastic Gradient Descent

```matlab
%-----------------------------------
% Stochastic Gradient Descent for Logistic Regression
%
% Inputs:
% X(i,:) - ith data point as a row vector
% y - {-1, +1} classifier
% w - initial guess for w
% b - initial guess for b
% lambda1 - tuning parameter
% lambda2 - tuning parameter
% maxit - max number of iteration
% tol - tolerance
% batch - batchsize
%
% Outputs:
% w - normal vector for hyperplane
% b - scalar for hyperplane
% hist_obj - history of objective value
% iter - number of iterations
%-----------------------------------
function [w, b, iter, hist_obj] =...
    SGD(X, y, w, b, lambda1, lambda2, maxit, batch)
    [m,n] = size(X);
    hist_obj = 0;
    iter = 1;
    while iter < maxit
    for i=1:(m/batch)
        % Pick random entries
        r = randi([1,m], batch, 1);
        Xup = X(r,:);
        yup = y(r,1);
        sgrad = UpdatedGradLR(Xup, yup, w, b, lambda1, lambda2);

        % Set alpha
        alpha = (5/6)/(iter);

        % Update w and b
        w = w - alpha*sgrad(1:n,1);
        b = b - alpha*sgrad(n+1,1);

    end
    % Update objective only if obj != Inf
    obj = UpdatedObjLR(X, y, w, b, lambda1, lambda2);
    if obj~=Inf
    hist_obj = [hist_obj; (sum(hist_obj) + obj)/iter];
    else
    hist_obj = [hist_obj; hist_obj(length(hist_obj))];
    end
    iter = iter + 1;
    end
end
```

Figure 4: Comparison of Algorithms with Varied λ_i

$\lambda_1 = 0.1$ $\lambda_2 = 0.1$ | $\lambda_1 = 0.005$ $\lambda_2 = 0.05$ | $\lambda_1 = 1$ $\lambda_2 = 0.0075$