

Report to Project-2

Yuanbo Han 15300180032

November 19, 2017

Contents

1	Logistic Regression	2
1.1	Bayes' Rule	2
1.2	Maximum Likelihood Estimation	3
1.3	L2 Regularization	4
2	Digit Classification	4
2.1	k-Nearest Neighbors	4
2.2	Logistic Regression	5
2.3	Penalized Logistic Regression	7
2.4	Naive Bayes	7
2.5	Compare k-NN, Logistic Regression, and Naive Bayes	8
3	Stochastic Subgradient Methods	8
3.1	Averaging Strategy	8
3.2	Second-Half Averaging Strategy	8
3.3	Stochastic Average Gradient	8
A	Codes for Digit Classification	10
A.1	Codes for k-Nearest Neighbors	10
A.1.1	knn_plot.m	10
A.1.2	knn_test.m	11
A.2	Codes for Logistic Regression	12
A.2.1	logistic_predict.m	12
A.2.2	evaluate.m	12
A.2.3	logistic.m	13
A.2.4	logistic_pen.m	13
A.2.5	plot_cross_entropy.m	14
A.2.6	plot_stats_against_lambda.m	16
A.3	Codes for Naive Bayes	19
A.3.1	train_nb.m	19
A.3.2	test_nb.m	20
A.3.3	run_nb.m	21

B	Codes for Stochastic Subgradient Methods	22
B.1	svmAvg.m	22
B.2	svmHalfAvg.m	24
B.3	svmSAG.m	26

1 Logistic Regression

1.1 Bayes' Rule

$$\begin{aligned}
p(y = 1|x) &= \frac{p(x|y = 1)p(y = 1)}{p(x|y = 1)p(y = 1) + p(x|y = 0)p(y = 0)} \\
&= \frac{\frac{\alpha}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)}}{\frac{\alpha}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)} + \frac{1-\alpha}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)}} \\
&= \frac{1}{1 + \frac{1-\alpha}{\alpha}e^{-\frac{1}{2}[(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) - (x-\mu_1)^T\Sigma^{-1}(x-\mu_1)]}} \\
&= \frac{1}{1 + \frac{1-\alpha}{\alpha}e^{-\frac{1}{2}\sum_{i=1}^D \frac{1}{\sigma_i^2}[(x_i-\mu_{i0})^2 - (x_i-\mu_{i1})^2]}} \\
&= \frac{1}{1 + \frac{1-\alpha}{\alpha}e^{-\frac{1}{2}\sum_{i=1}^D \frac{1}{\sigma_i^2}[2(\mu_{i1}-\mu_{i0})x + \mu_{i0}^2 - \mu_{i1}^2]}} \\
&= \frac{1}{1 + e^{-\sum_{i=1}^D \frac{\mu_{i1}-\mu_{i0}}{\sigma_i^2}x_i - \left(\sum_{i=1}^D \frac{\mu_{i0}^2-\mu_{i1}^2}{2\sigma_i^2} - \log \frac{1-\alpha}{\alpha}\right)}}
\end{aligned}$$

Let $w_i = \frac{\mu_{i1}-\mu_{i0}}{\sigma_i^2}$, $\mathbf{w} = (w_1, w_2, \dots, w_D)^T$, $b = \sum_{i=1}^D \frac{\mu_{i0}^2-\mu_{i1}^2}{2\sigma_i^2} - \log \frac{1-\alpha}{\alpha}$, then

$$p(y = 1|x) = \frac{1}{1 + e^{-\mathbf{w}^T x - b}} = \sigma(\mathbf{w}^T x + b)$$

1.2 Maximum Likelihood Estimation

$$\begin{aligned}
E(\mathbf{w}, b) &= - \sum_{i=1}^N \log p(y^{(i)} | \mathbf{x}^{(i)}) \\
&= - \sum_{i=1}^N \left\{ y^{(i)} \log p(y = 1 | \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log p(y = 0 | \mathbf{x}^{(i)}) \right\} \\
&= - \sum_{i=1}^N \left\{ y^{(i)} \log \frac{p(y = 1 | \mathbf{x}^{(i)})}{p(y = 0 | \mathbf{x}^{(i)})} + \log p(y = 0 | \mathbf{x}^{(i)}) \right\} \\
&= - \sum_{i=1}^N \left\{ (y^{(i)} - 1) (\mathbf{x}^{(i)T} \mathbf{w} + b) - \log (1 + e^{-\mathbf{x}^{(i)T} \mathbf{w} - b}) \right\}
\end{aligned}$$

If we write

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_D^{(1)} & 1 \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_D^{(2)} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_D^{(N)} & 1 \end{bmatrix}$$

$$y = (y^{(1)}, y^{(2)}, \dots, y^{(N)})^T$$

and

$$\mathbf{W} = (w_1, w_2, \dots, w_D, b)^T$$

then

$$E(\mathbf{w}, b) = -\mathbf{W}^T \mathbf{X}^T (y - 1) - \sum_{i=1}^N \log \sigma(\mathbf{X} \mathbf{W}) \quad (1)$$

$$\frac{\partial E(\mathbf{w}, b)}{\partial \mathbf{W}} = -\mathbf{X}^T (y - \sigma(\mathbf{X} \mathbf{W})) \quad (2)$$

1.3 L2 Regularization

$$\begin{aligned}
p(\mathcal{D}|\mathbf{w}, b) &= \prod_{i=1}^N p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}, b) \\
&= e^{\sum_{i=1}^N \log p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}, b)} \\
&= e^{-E(\mathbf{w}, b)} \\
p(\mathbf{w}, b) &= \prod_{i=1}^N \mathcal{N}\left(w_j|0, \frac{1}{\lambda}\right) \cdot \mathcal{N}\left(b|0, \frac{1}{\lambda}\right) \\
&= \frac{1}{(2\pi)^{\frac{D+1}{2}} \left(\frac{1}{\lambda}\right)^{\frac{D+1}{2}}} e^{-\frac{1}{2} \sum_{j=1}^D \frac{w_j^2}{\lambda} - \frac{1}{2} \frac{b^2}{\lambda}} \\
&= \left(\frac{\lambda}{2\pi}\right)^{\frac{D+1}{2}} e^{-\frac{\lambda}{2} (\sum_{j=1}^D w_j^2 + b^2)}
\end{aligned}$$

By Bayes' rule,

$$\begin{aligned}
p(\mathbf{w}, b|\mathcal{D}) &= \frac{p(\mathcal{D}|\mathbf{w}, b)p(\mathbf{w}, b)}{p(\mathcal{D})} \\
&\propto p(\mathcal{D}|\mathbf{w}, b)p(\mathbf{w}, b) \\
&= \left(\frac{\lambda}{2\pi}\right)^{\frac{D+1}{2}} e^{-E(\mathbf{w}, b) - \frac{\lambda}{2} (\sum_{j=1}^D w_j^2 + b^2)} \\
L(\mathbf{w}, b) &= -\log \left\{ \left(\frac{\lambda}{2\pi}\right)^{\frac{D+1}{2}} e^{-E(\mathbf{w}, b) - \frac{\lambda}{2} (\sum_{j=1}^D w_j^2 + b^2)} \right\} \\
&= E(\mathbf{w}, b) + \frac{\lambda}{2} \sum_{j=1}^D w_j^2 + \frac{\lambda}{2} b^2 + C(\lambda)
\end{aligned}$$

where $C(\lambda) = \frac{D+1}{2} \log \frac{2\pi}{\lambda}$.

2 Digit Classification

2.1 k-Nearest Neighbors

The script *knn_plot.m* written by me runs kNN for different values of $k \in \{1, 3, 5, 7, 9\}$, and plots the classification rate on the validation set as a function of k (see Figure 1).

As we can see in the figure, $k = 3, 5, 7$ performs the best, so the final k we will choose is among them. Since 5 is in the middle of $\{3, 5, 7\}$, we can infer that $k = 5$ is more steady to perform well than $k = 3$ or $k = 7$. Therefore, I'd like to choose $k^* = 5$, and the classification rate on the test set is 94%. For comparison,

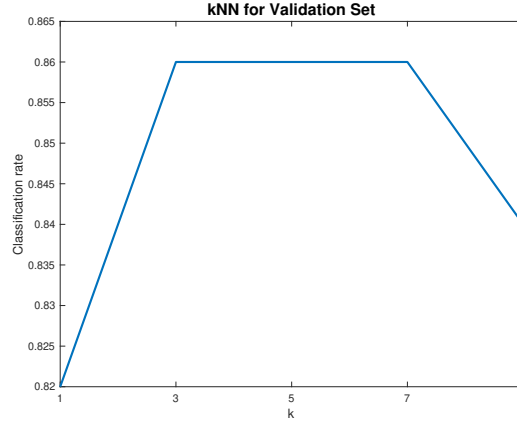


Figure 1: Classification rate against k

I write *plot_knn.m* to compute the rate for $k^* - 2 = 3$ and $k^* + 2 = 7$ as well, and plot them in Figure 2.

As a consequence, the outcome agrees with my analysis.

2.2 Logistic Regression

Codes of *logistic.m*, *logistic_predict.m* and *evaluate.m* are in the Appendix.

It is necessary to mention that, in the original provided script *logistic_regression_template.m*, the learning rate is actually $\frac{\text{hyperparameters.learning_rate}}{N}$, where N is the number of examples in training data. However, I found no good to do this division, and it is even troublesome when comparing performance of different training sets. Since N remains a constant once we have chosen a certain training set, we can simply delete it and use *hyperparameters.learning_rate* as the true learning rate. Through the whole experiment below, I have applied *hyperparameters.learning_rate* instead of $\frac{\text{hyperparameters.learning_rate}}{N}$, and I will not mention this again.

From my experiment, it performs well to set *hyperparameters.learning_rate* = 0.001, *hyperparameters.num_iterations* = 300, and initialize *weights* randomly by standard normal distribution. Running it once on the test set, the final stats is

```

1 ITERATION: 300   NLOGL: 0.05
2 TRAIN_CE:0.054999 TRAIN_FRAC:99.38
3 VALIC_CE:0.432754 VALID_FRAC:88.00
4 TEST_CE:0.294021 TEST_FRAC:88.00

```

My script *plot_cross_entropy.m* shows how the cross entropy changes as training progresses. The result is in Figure 3.

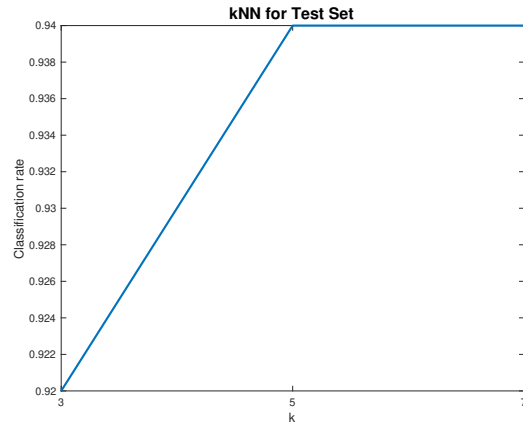


Figure 2: Comparison among my chosen k^* , $k^* - 2$ and $k^* + 2$

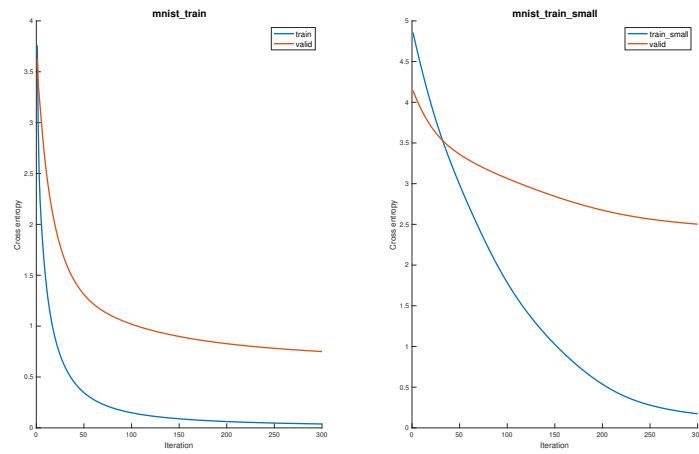


Figure 3: Cross entropy as training progresses

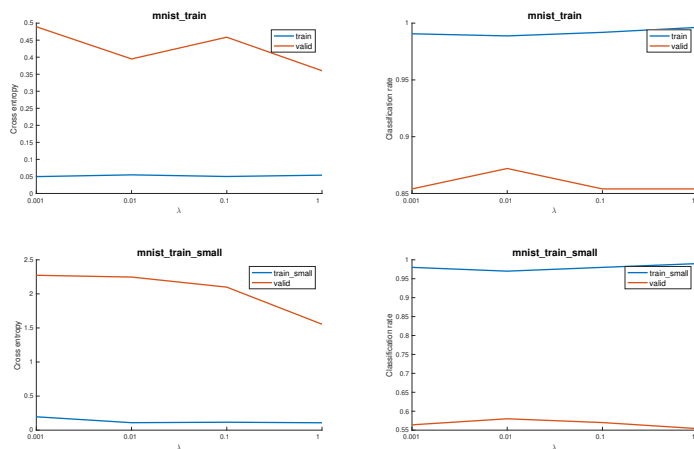


Figure 4: Average cross entropy and classification rate against λ

Running *plot_cross_entropy.m*, the plot does change subtly. That is mainly because the initial *weights* is set randomly. In practical experiment, I first fix all the initial values of weight parameters to be 0. Then I run the program and change *hyperparameters.learning_rate* and *hyperparameters.num_iteration*s several times to make them ideal. And finally I let *weights* randomly initialized by standard normal distribution, which proves to perform well.

2.3 Penalized Logistic Regression

Codes of *logistic_pen.m* are in the Appendix. My *plot_stats_against_lambda.m* plots the average cross entropy and classification rate against λ . The result is in Figure 4. The best λ for both training sets is $\lambda = 0.01$, since the average classification rates of valid data is the highest with $\lambda = 0.01$. And in this situation, the classification rate of test data is 92% for *mnist_train* and 70% for *mnist_train_small*.

For explanation, a small λ is exactly what we want, for it reduces the influence of those unimportant parameters of *weights*. That's why it performs bad when λ is big. However, a quite small λ is too near to 0, and it will be similar to the situation without penalty.

2.4 Naive Bayes

The implementation is *run_nb.m* in the Appendix. The result is in Figure 5.

The visualization of the mean and variance vectors for both classes look respectively like 4s and 9s. The difference is that the visualization for the mean fades from the inner to the outer, but that for the variance fades from the outer to the inner.

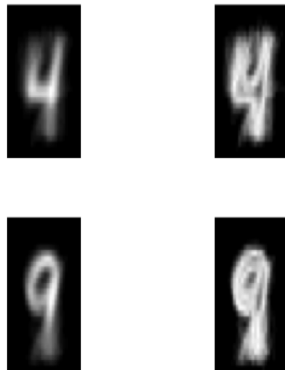


Figure 5: Visualization of the mean and variance vectors

2.5 Compare k-NN, Logistic Regression, and Naive Bayes

The Naive Bayes classifier is the simplest method but performs the worst. k-NN has to choose a right k to perform well, while it is still easy to achieve. The logistic regression is the most complicated method among these three, especially when we add penalty. There are at least 3 kinds of parameters for us to modify. But of course, it pays off with the highest classification rate.

3 Stochastic Subgradient Methods

3.1 Averaging Strategy

Codes of *svmAvg.m* are in the Appendix. The result is in Figure 6.

3.2 Second-Half Averaging Strategy

Codes of *svmHalfAvg.m* are in the Appendix. The result is in Figure 7.

3.3 Stochastic Average Gradient

I write a program to implement the SAG algorithm (stochastic average gradient), which is a kind of stochastic subgradient method (proposed by Le Roux, Schmidt, and Bach, 2012). The result is in Figure 8. Obviously, it performs much better than all the methods discussed above.

Codes of *svmSAG.m* are in the Appendix.

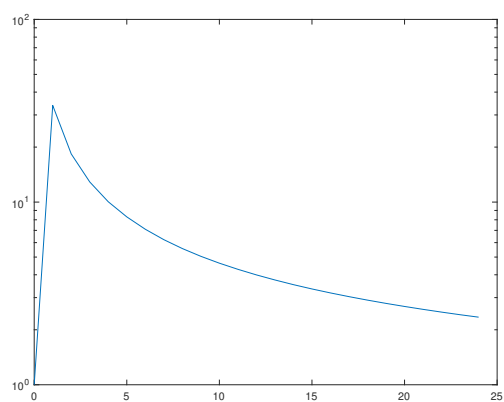


Figure 6: svmAvg

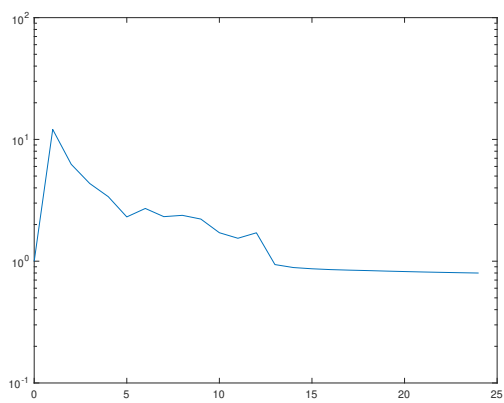


Figure 7: svmHalfAvg

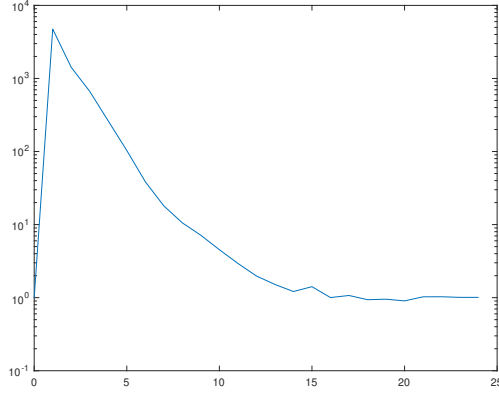


Figure 8: svmSAG

A Codes for Digit Classification

A.1 Codes for k-Nearest Neighbors

A.1.1 knn_plot.m

```

1  % Edited by Yuanbo Han, 2017-11-14.
2
3  % Clear workspace.
4  clear all;
5  close all;
6
7  % Load data.
8  load ../data/mnist_train;
9  load ../data/mnist_valid;
10
11 N = size(valid_inputs, 1);
12 K = 1:2:9; % set of values of k
13 num = length(K); % the number of values of k
14
15 % Compute the classification rates for each k.
16 classification_rate = zeros(num, 1);
17 for i = 1:num
18     valid_labels = run_knn( K(i), train_inputs, train_targets, valid_inputs);
19     classification_rate(i) = sum(valid_labels == valid_targets) / N;
20 end
21
22 % Plot the classification rate against k.

```

```

23 figure;
24 plot(K, classification_rate, 'LineWidth', 2);
25 title('kNN for Validation Set', 'FontSize', 15);
26 xlabel('k', 'FontSize', 12);
27 ylabel('Classification rate', 'FontSize', 12);
28 set(gca, 'XTick', K);
29 set(gca, 'XTickLabel', K);
30
31 clear i;

```

A.1.2 knn_test.m

```

1 % Edited by Yuanbo Han, 2017-11-14.
2
3 % Clear workspace.
4 clear all;
5 close all;
6
7 % Load data.
8 load ../data/mnist_train;
9 load ../data/mnist_test;
10
11 N = size(test_inputs, 1);
12 k = 5; % my chosen value of k
13 K = [k-2, k, k+2];
14
15 % Compute the classification rates for each k.
16 classification_rate = zeros(3, 1);
17 for i = 1:3
18     test_labels = run_knn( K(i), train_inputs, train_targets, test_inputs);
19     classification_rate(i) = sum(test_labels == test_targets) / N;
20 end
21
22 % Plot the classification rate against k.
23 figure;
24 plot(K, classification_rate, 'LineWidth', 2);
25 title('kNN for Test Set', 'FontSize', 15);
26 xlabel('k', 'FontSize', 12);
27 ylabel('Classification rate', 'FontSize', 12);
28 set(gca, 'XTick', K);
29 set(gca, 'XTickLabel', K);
30
31 clear i;

```

A.2 Codes for Logistic Regression

A.2.1 logistic_predict.m

```
1 function [y] = logistic_predict(weights, data)
2 % Compute the probabilities predicted by the logistic classifier.
3 %
4 % Note: N is the number of examples and
5 %       M is the number of features per example.
6 %
7 % Inputs:
8 %   weights: (M+1) x 1 vector of weights, where the last element
9 %             corresponds to the bias (intercepts).
10 %   data:    N x M data matrix where each row corresponds
11 %            to one data point.
12 % Outputs:
13 %   y:       N x 1 vector of probabilities.
14 %            This is the output of the classifier.
15 %
16 % Yuanbo Han, 2017-11-12.
17
18 [N, ~] = size(data);
19 z = [data, ones(N,1)] * weights;
20 y = sigmoid(z);
21 end
```

A.2.2 evaluate.m

```
1 function [ce, frac_correct] = evaluate(targets, y)
2 % Compute evaluation metrics.
3 % Inputs:
4 %   targets : N x 1 vector of binary targets. Values should be either 0 or 1.
5 %   y       : N x 1 vector of probabilities.
6 % Outputs:
7 %   ce           : (scalar) Cross entropy.  $CE(p, q) = E_p[-\log q]$ . Here we
8 %                 want to compute  $CE(targets, y)$ .
9 %   frac_correct : (scalar) Fraction of inputs classified correctly.
10 %
11 % Yuanbo Han, 2017-11-12.
12
13 ce = mean( - targets .* log(y) - (1-targets) .* log(1-y) );
14 frac_correct = ( sum(targets==1 & y>=0.5) + sum(targets==0 & y<0.5) ) / size(y,1)
15 end
```

A.2.3 logistic.m

```
1 function [f, df, y] = logistic(weights, data, targets, ~)
2 % Calculate log likelihood and derivatives with respect to weights.
3 %
4 % Note: N is the number of examples and
5 %       M is the number of features per example.
6 %
7 % Inputs:
8 %       weights: (M+1) x 1 vector of weights, where the last element
9 %               corresponds to bias (intercepts).
10 %      data:    N x M data matrix where each row corresponds
11 %              to one data point.
12 %      targets: N x 1 vector of binary targets. Values should be either
13 %              0 or 1.
14 %      ~:       The hyperparameter structure is omitted.
15 %
16 % Outputs:
17 %      f:       The scalar error value (i.e. negative log likelihood).
18 %      df:      (M+1) x 1 vector of derivatives of error w.r.t. weights.
19 %      y:       N x 1 vector of probabilities.
20 %              This is the output of the classifier.
21 %
22 % Yuanbo Han, 2017-11-12.
23
24 x = [ data, ones( size(data,1), 1 ) ];
25 z = x * weights;
26 y = sigmoid(z);
27 f = - z' * (targets - 1) - sum( log(y) );
28 df = - x' * ( targets - y );
29 end
```

A.2.4 logistic_pen.m

```
1 function [f, df, y] = logistic_pen(weights, data, targets, hyperparameters)
2 % Penalized logistic regression.
3 % Calculate log likelihood and derivatives with respect to weights.
4 %
5 % Note: N is the number of examples and
6 %       M is the number of features per example.
7 %
8 % Inputs:
9 %       weights: (M+1) x 1 vector of weights, where the last element
10 %              corresponds to bias (intercepts).
11 %      data:    N x M data matrix where each row corresponds
```

```

12 %           to one data point.
13 %   targets:   N x 1 vector of targets class probabilities.
14 %   hyperparameters: The hyperparameter structure
15 %
16 % Outputs:
17 %       f:           The scalar error value (i.e. negative log liklihood
18 %                   + lambda/2 * weights(1:M)' * weights(1:M)).
19 %       df:          (M+1) x 1 vector of derivatives of error w.r.t. weights.
20 %       y:           N x 1 vector of probabilities.
21 %                   This is the output of the classifier.
22 %
23 % Yuanbo Han, 2017-11-13.
24
25 x = [ data, ones( size(data,1), 1 ) ];
26 z = x * weights;
27 y = sigmoid(z);
28 weights( length(weights) ) = 0;
29 f = - z' * (targets - 1) - sum( log(y) ) + hyperparameters.weight_regularization
30 df = - x' * ( targets - y ) + hyperparameters.weight_regularization * weights;
31 end

```

A.2.5 plot_cross_entropy.m

```

1 % Edited by Yuanbo Han, 2017-11-12.
2
3 clear all;
4 close all;
5
6 %% Load data.
7 load ../data/mnist_train;
8 load ../data/mnist_train_small;
9 load ../data/mnist_valid;
10
11 %% Initialize hyperparameters.
12 % Learning rate
13 hyperparameters.learning_rate = 0.001;
14 % Weight regularization parameter
15 hyperparameters.weight_regularization = 0;
16 % Number of iterations
17 hyperparameters.num_iterations = 300;
18 % Logistic regression weights
19 % Set random weights.
20 weights = randn( (size(train_inputs,2) + 1), 1 );
21 %weights = zeros( (size(train_inputs,2) + 1), 1 );
22 weights_small = weights;

```

```

23
24 N = size(train_inputs, 1);
25 N_small = size(train_inputs_small, 1);
26
27 cross_entropy_train = zeros( hyperparameters.num_iterations, 1 );
28 cross_entropy_train_small = cross_entropy_train;
29 cross_entropy_valid = cross_entropy_train;
30 cross_entropy_valid_small = cross_entropy_train;
31
32 %% Begin learning with gradient descent.
33 for t = 1:hyperparameters.num_iterations
34
35     % Find the negative log likelihood and derivatives w.r.t. weights.
36     [f, df, predictions] = logistic(weights, ...
37         train_inputs, ...
38         train_targets, ...
39         hyperparameters);
40
41     [f_small, df_small, predictions_small] = logistic(weights_small, ...
42         train_inputs_small, ...
43         train_targets_small, ...
44         hyperparameters);
45
46     % Report the possible errors.
47     if isnan(f) || isinf(f)
48         error('f nan/inf error');
49     end
50
51     if isnan(f_small) || isinf(f_small)
52         error('f_small nan/inf error');
53     end
54
55     % Find the cross entropy and fraction of correctly classified examples of tr
56     [cross_entropy_train(t), frac_correct_train] = evaluate(train_targets, predi
57     [cross_entropy_train_small(t), frac_correct_train_small] = evaluate(train_ta
58
59     % Update weights.
60     weights = weights - hyperparameters.learning_rate .* df;
61     weights_small = weights_small - hyperparameters.learning_rate .* df_small;
62
63     % Find the cross entropy and fraction of correctly classified examples of va
64     predictions_valid = logistic_predict(weights, valid_inputs);
65     predictions_valid_small = logistic_predict(weights_small, valid_inputs);
66
67     [cross_entropy_valid(t), frac_correct_valid] = evaluate(valid_targets, predi
68     [cross_entropy_valid_small(t), frac_correct_valid_small] = evaluate(valid_ta

```

```

69
70     % Print some stats.
71     fprintf(1, 'ITERATION:%4i    NLOGL:%11.2f TRAIN_CE:%16.6f TRAIN_FRAC:%12.2f V
72             t, f/N, cross_entropy_train(t), frac_correct_train*100, cross_entropy_va
73     fprintf( '%17sNLOGL_SMALL:%5.2f TRAIN_SMALL_CE:%10.6f TRAIN_SMALL_FRAC:%6.2f
74             ', f_small/N_small, cross_entropy_train_small(t), frac_correct_train_sm
75
76 end
77
78 %% Plot the cross entropy as training progresses.
79 figure;
80 subplot(1,2,1);
81 hold on;
82 title('mnist\_train', 'FontSize', 15);
83 plot(1:hyperparameters.num_iterations, cross_entropy_train, 'LineWidth', 2);
84 plot(1:hyperparameters.num_iterations, cross_entropy_valid, 'LineWidth', 2);
85 lgd = legend('train', 'valid', 'Location', 'NorthEast');
86 set(lgd, 'FontSize', 12);
87 xlabel('Iteration', 'FontSize', 12);
88 ylabel('Cross entropy', 'FontSize', 12);
89
90 subplot(1,2,2);
91 hold on;
92 title('mnist\_train\_small', 'FontSize', 15);
93 plot(1:hyperparameters.num_iterations, cross_entropy_train_small, 'LineWidth', 2);
94 plot(1:hyperparameters.num_iterations, cross_entropy_valid_small, 'LineWidth', 2);
95 lgd = legend('train\_small', 'valid', 'Location', 'NorthEast');
96 set(lgd, 'FontSize', 12);
97 xlabel('Iteration', 'FontSize', 12);
98 ylabel('Cross entropy', 'FontSize', 12);
99
100 clear t lgd;

```

A.2.6 plot_stats_against_lambda.m

```

1 % Edited by Yuanbo Han, 2017-11-13.
2
3 clear all;
4 close all;
5
6 %% Load data.
7 load ../data/mnist_train;
8 load ../data/mnist_train_small;
9 load ../data/mnist_valid;
10

```



```

11 %% Initialize hyperparameters.
12 % Learning rate
13 hyperparameters.learning_rate = 0.001;
14 % Number of iterations
15 hyperparameters.num_iterations = 300;
16
17 [N, M] = size(train_inputs);
18 [N_small, M_small] = size(train_inputs_small);
19
20 penalty_parameters = logspace(-3,0,4); % values of penalty parameter
21 num = length(penalty_parameters); % the number of values of penalty parameter
22
23 rerun_times = 10;
24
25 cross_entropy_train = zeros(rerun_times, num);
26 cross_entropy_train_small = zeros(rerun_times, num);
27 cross_entropy_valid = zeros(rerun_times, num);
28 cross_entropy_valid_small = zeros(rerun_times, num);
29
30 %% Compute some stats for each penalty parameters.
31 for i = 1:num
32     hyperparameters.weight_regularization = penalty_parameters(i);
33     for r = 1:rerun_times
34         fprintf( '\n\nPENALTY PARAMETER = %.3f    RUN TIME = %d\n\n', hyperparameters.learning_rate, hyperparameters.num_iterations);
35
36         % Randomly initialize the logistic regression weights.
37         weights = randn( M+1, 1 );
38         weights_small = randn( M_small+1, 1 );
39
40         % Begin learning with gradient descent.
41         for t = 1:hyperparameters.num_iterations
42
43             % Find the error value and derivatives w.r.t. weights.
44             [f, df, predictions] = logistic_pen(weights, ...
45                 train_inputs, ...
46                 train_targets, ...
47                 hyperparameters);
48
49             [f_small, df_small, predictions_small] = logistic_pen(weights_small, ...
50                 train_inputs_small, ...
51                 train_targets_small, ...
52                 hyperparameters);
53
54             % Report the possible errors.
55             if isnan(f) || isinf(f)
56                 error( 'f nan/inf error' );

```

```

57         end
58
59         if isnan(f_small) || isinf(f_small)
60             error('f_small nan/inf error');
61         end
62
63         % Find the cross entropy and fraction of correctly classified examples
64         [cross_entropy_train(r,i), frac_correct_train(r,i)] = evaluate(train_inputs,
65         [cross_entropy_train_small(r,i), frac_correct_train_small(r,i)] = evaluate(train_inputs_small,
66
67         % Update weights.
68         weights = weights - hyperparameters.learning_rate .* df;
69         weights_small = weights_small - hyperparameters.learning_rate .* df_small;
70
71         % Find the cross entropy and fraction of correctly classified examples
72         predictions_valid = logistic_predict(weights, valid_inputs);
73         predictions_valid_small = logistic_predict(weights_small, valid_inputs_small);
74
75         [cross_entropy_valid(r,i), frac_correct_valid(r,i)] = evaluate(valid_inputs,
76         [cross_entropy_valid_small(r,i), frac_correct_valid_small(r,i)] = evaluate(valid_inputs_small,
77
78         % Print some stats.
79         fprintf(1, 'ITERATION:%4i    NLOGL:%11.2f TRAIN_CE:%16.6f TRAIN_FRAC:%16.6f\n',
80             t, f/N, cross_entropy_train(r,i), frac_correct_train(r,i)*100, c/N, frac_correct_train(r,i)*100);
81         fprintf(1, '%17sNLOGL_SMALL:%5.2f TRAIN_SMALL_CE:%10.6f TRAIN_SMALL_FRAC:%10.6f\n',
82             ' ', f_small/N_small, cross_entropy_train_small(r,i), frac_correct_train_small(r,i)*100);
83
84         end
85     end
86 end
87
88 %% Plot the cross entropy and classification rate against penalty parameters.
89 figure;
90 subplot(2,2,1);
91 hold on;
92 title('mnist\_train', 'FontSize', 15);
93 plot(1:num, mean(cross_entropy_train, 1), 'LineWidth', 2);
94 plot(1:num, mean(cross_entropy_valid, 1), 'LineWidth', 2);
95 lgd = legend('train', 'valid', 'Location', 'NorthEast');
96 set(lgd, 'FontSize', 12);
97 xlabel('\lambda', 'FontSize', 12);
98 ylabel('Cross entropy', 'FontSize', 12);
99 set(gca, 'XTick', 1:num);
100 set(gca, 'XTickLabel', penalty_parameters);
101
102 subplot(2,2,2);

```

```

103 hold on;
104 title('mnist\_train', 'FontSize', 15);
105 plot(1:num, mean(frac\_correct\_train, 1), 'LineWidth', 2);
106 plot(1:num, mean(frac\_correct\_valid, 1), 'LineWidth', 2);
107 lgd = legend('train', 'valid', 'Location', 'NorthEast');
108 set(lgd, 'FontSize', 12);
109 xlabel('\lambda', 'FontSize', 12);
110 ylabel('Classification rate', 'FontSize', 12);
111 set(gca, 'XTick', 1:num);
112 set(gca, 'XTickLabel', penalty_parameters);
113
114 subplot(2,2,3);
115 hold on;
116 title('mnist\_train\_small', 'FontSize', 15);
117 plot(1:num, mean(cross\_entropy\_train\_small, 1), 'LineWidth', 2);
118 plot(1:num, mean(cross\_entropy\_valid\_small, 1), 'LineWidth', 2);
119 lgd = legend('train\_small', 'valid', 'Location', 'NorthEast');
120 set(lgd, 'FontSize', 12);
121 xlabel('\lambda', 'FontSize', 12);
122 ylabel('Cross entropy', 'FontSize', 12);
123 set(gca, 'XTick', 1:num);
124 set(gca, 'XTickLabel', penalty_parameters);
125
126 subplot(2,2,4);
127 hold on;
128 title('mnist\_train\_small', 'FontSize', 15);
129 plot(1:num, mean(frac\_correct\_train\_small, 1), 'LineWidth', 2);
130 plot(1:num, mean(frac\_correct\_valid\_small, 1), 'LineWidth', 2);
131 lgd = legend('train\_small', 'valid', 'Location', 'NorthEast');
132 set(lgd, 'FontSize', 12);
133 xlabel('\lambda', 'FontSize', 12);
134 ylabel('Classification rate', 'FontSize', 12);
135 set(gca, 'XTick', 1:num);
136 set(gca, 'XTickLabel', penalty_parameters);
137
138 clear i r t lgd;

```

A.3 Codes for Naive Bayes

A.3.1 train_nb.m

```

1 function [log_prior, class_mean, class_var] = train_nb(train_data, train_label)
2 % TRAIN_NB trains a Naive Bayes binary classifier. All conditional
3 % distributions are Gaussian.
4 %

```

```

5 % Usage:
6 % [log_prior, class_mean, class_var] = train_nb(train_data, train_label);
7 %
8 % Inputs:
9 % train_data : n_examples x n_dimensions matrix
10 % train_label : n_examples x 1 binary label vector
11 %
12 % Outputs:
13 % log_prior : 2 x 1 vector, log_prior(i) = log(p(C=i)).
14 % class_mean : 2 x n_dimensions matrix, class_mean(i,:) is the mean
15 % vector for class i.
16 % class_var : 2 x n_dimensions matrix, class_var(i,:) is the variance
17 % vector for class i.
18 %
19 % Modified by Yuanbo Han, 2017-11-14: Omit the unused variable by '~', and
20 % correct the comments.
21
22 SMALL_CONSTANT = 1e-10;
23
24 [~, n_dims] = size(train_data);
25 K = 2;
26
27 prior = zeros(K, 1);
28 class_mean = zeros(K, n_dims);
29 class_var = zeros(K, n_dims);
30
31 for k = 1 : K
32     prior(k) = mean(train_label == (k-1));
33     class_mean(k, :) = mean(train_data(train_label == (k-1), :), 1);
34     class_var(k, :) = var(train_data(train_label == (k-1), :), 0, 1);
35 end
36
37 class_var = class_var + SMALL_CONSTANT;
38 log_prior = log(prior + SMALL_CONSTANT);
39
40 end

```

A.3.2 test_nb.m

```

1 function [prediction, accuracy] = test_nb(test_data, test_label, log_prior, class
2 % TEST_NB tests a learned Naive Bayes classifier.
3 %
4 % Usage:
5 % [prediction, accuracy] = test_nb(test_data, test_label, log_prior, ...
6 % class_mean, class_var);

```

```

7 %
8 % Inputs:
9 %   test_data   : n_examples x n_dimensions matrix
10 %   test_label  : n_examples x 1 binary label vector
11 %   log_prior   : 2 x 1 vector, log_prior(i) = log(p(C=i)).
12 %   class_mean  : 2 x n_dimensions matrix, class_mean(i,:) is the mean
13 %                 vector for class i.
14 %   class_var   : 2 x n_dimensions matrix, class_var(i,:) is the variance
15 %                 vector for class i.
16 %
17 % Outputs:
18 %   prediction  : n_examples x 1 binary label vector
19 %   accuracy    : a real number
20 %
21 % Modified by Yuanbo Han, 2017-11-14: Correct the comments.
22
23 K = length(log_prior);
24 n_examples = size(test_data, 1);
25
26 log_prob = zeros(n_examples, K);
27
28 for k = 1 : K
29     mean_mat = repmat(class_mean(k, :), [n_examples, 1]);
30     var_mat = repmat(class_var(k, :), [n_examples, 1]);
31     log_prob(:, k) = sum(-0.5 * (test_data - mean_mat).^2 ./ var_mat - 0.5 * log
32 end
33
34 [~, prediction] = max(log_prob, [], 2);
35 prediction = prediction - 1;
36 accuracy = mean(prediction == test_label);
37
38 end

```

A.3.3 run_nb.m

```

1 % Learn a Naive Bayes classifier on the digit dataset, evaluate its
2 % performance on training and test sets, then visualize the mean and
3 % variance for each class.
4 % Edited by Yuanbo Han, 2017-11-14.
5
6 % Clear workspace and close figures.
7 clear all;
8 close all;
9
10 % Load data.

```

```

11 load ../data/mnist_train;
12 load ../data/mnist_test;
13
14 % Add your code here (it should be less than 10 lines).
15 [log_prior, class_mean, class_var] = train_nb(train_inputs, train_targets);
16 [prediction_train, accuracy_train] = test_nb(train_inputs, train_targets, log_prior);
17 [prediction_test, accuracy_test] = test_nb(test_inputs, test_targets, log_prior);
18
19 fprintf('Training accuracy = %5.2f%%\n', accuracy_train * 100);
20 fprintf('Test accuracy = %9.2f%%\n', accuracy_test * 100);
21
22 plot_digits(class_mean); % mean visualization
23 plot_digits(class_var); % variance visualization

```

B Codes for Stochastic Subgradient Methods

B.1 svmAvg.m

```

1 function [model] = svmAvg(X,y,lambda,maxIter)
2 % SVM AVG minimizes the SVM objective function by stochastic method based on
3 % the running average of w.
4 %
5 % Yuanbo Han, 2017-11-18.
6
7 % Add the bias variable.
8 [n,d] = size(X);
9 X = [ones(n,1), X];
10
11 % Use the transpose to accelerate the program.
12 Xt = X';
13
14 % Initialize the values of regression parameters.
15 w = zeros(d+1,1);
16
17 % The running average of w
18 w_bar = w;
19
20 % Apply stochastic method based on the running average of w.
21 for t = 1:maxIter
22     if mod(t-1,n) == 0
23         % Plot our progress.
24         % (turn this off for acceleration)
25
26         objValues(1+(t-1)/n) = (1/n)*sum(max(0,1-y.*(X*w_bar))) + (lambda/2)*(w_bar'

```

```

27         semilogy ([0:t/n], objValues);
28         pause(.1);
29     end
30
31     % Pick a random training example.
32     i = randi(n);
33
34     % Compute the i-th subgradient.
35     [~, sg] = hingeLossSubGrad(w, Xt, y, i);
36
37     % Set step size.
38     alpha = 1 / (lambda * t);
39
40     % Take stochastic subgradient step.
41     w = w - alpha * (sg + lambda * w);
42
43     % Renew the running average of w.
44     w_bar = (t-1)/t * w_bar + 1/t * w;
45 end
46
47 model.w = w_bar;
48 model.predict = @predict;
49
50 end
51
52 function [yhat] = predict(model, Xhat)
53 d = size(Xhat, 1);
54 Xhat = [ones(d, 1), Xhat];
55 w = model.w;
56 yhat = sign(Xhat * w);
57 end
58
59 function [f, sg] = hingeLossSubGrad(w, Xt, y, i)
60 % Function value
61 wtx = w' * Xt(:, i);
62 f = max(0, 1 - y(i) * wtx);
63
64 % Subgradient
65 if f > 0
66     sg = - y(i) * Xt(:, i);
67 else
68     sg = zeros(size(Xt, 1), 1);
69 end
70
71 end

```

B.2 svmHalfAvg.m

```
1 function [model] = svmHalfAvg(X,y,lambda,maxIter)
2 % SVM AVG minimizes the SVM objective function by stochastic gradient
3 % method, based on the current w for the first half iterations, and based
4 % on the running average of w for the second half.
5 %
6 % Yuanbo Han, 2017-11-18.
7
8 % Add the bias variable.
9 [n,d] = size(X);
10 X = [ones(n,1), X];
11
12 % Use the transpose to accelerate the program.
13 Xt = X';
14
15 % Initialize the values of regression parameters.
16 w = zeros(d+1,1);
17
18 % The 1st half: based on the current w
19
20 % Apply stochastic gradient method.
21 for t = 1 : (maxIter/2)
22     if mod(t-1,n) == 0
23         % Plot our progress.
24         % (turn this off for acceleration)
25
26         objValues(1+(t-1)/n) = (1/n)*sum(max(0,1-y.*(X*w))) + (lambda/2)*(w'*w);
27         semilogy([0:t/n],objValues);
28         pause(.1);
29     end
30
31     % Pick a random training example.
32     i = randi(n);
33
34     % Renew the i-th and the average subgradient.
35     [~, sg] = hingeLossSubGrad(w,Xt,y,i);
36
37     % Set step size.
38     alpha = 1 / (lambda * t);
39
40     % Take stochastic subgradient step.
41     w = w - alpha * ( sg + lambda * w );
42 end
43
44 % The 2nd half: based on the running average of w
```



```

45
46 % The running average of w
47 w_bar = w;
48
49 % Apply stochastic method based on the running average of w.
50 k = 1;
51 for t = ceil(maxIter/2) : maxIter
52     if mod(t-1,n) == 0
53         % Plot our progress.
54         % (turn this off for acceleration)
55
56         objValues(1+(t-1)/n) = (1/n)*sum(max(0,1-y.*(X*w_bar))) + (lambda/2)*(w_
57         semilogy([0:t/n],objValues);
58         pause(.1);
59     end
60
61     % Pick a random training example.
62     i = randi(n);
63
64     % Compute the i-th subgradient.
65     [~, sg] = hingeLossSubGrad(w,Xt,y,i);
66
67     % Set step size.
68     alpha = 1 / (lambda * t);
69
70     % Take stochastic subgradient step.
71     w = w - alpha * (sg + lambda * w);
72
73     % Renew the running average of w.
74     w_bar = (k-1)/k * w_bar + 1/k * w;
75     k = k + 1;
76 end
77
78 model.w = w_bar;
79 model.predict = @predict;
80
81 end
82
83 function [yhat] = predict(model,Xhat)
84 d = size(Xhat,1);
85 Xhat = [ones(d,1), Xhat];
86 w = model.w;
87 yhat = sign(Xhat * w);
88 end
89
90 function [f,sg] = hingeLossSubGrad(w,Xt,y,i)

```

```

91 % Function value
92 wtx = w' * Xt(:,i);
93 f = max(0, 1 - y(i) * wtx );
94
95 % Subgradient
96 if f > 0
97     sg = - y(i) * Xt(:,i);
98 else
99     sg = zeros(size(Xt,1), 1);
100 end
101
102 end

```

B.3 svmSAG.m

```

1 function [model] = svmSAG(X,y,lambda,maxIter)
2 % SVMAG minimizes the SVM objective function by stochastic average
3 % gradient (SAG) method.
4 %
5 % Yuanbo Han, 2017-11-18.
6
7 % Add the bias variable.
8 [n,d] = size(X);
9 X = [ones(n,1), X];
10
11 % Use the transpose to accelerate the program.
12 Xt = X';
13
14 % Initialize the values of regression parameters.
15 w = zeros(d+1,1);
16
17 % Compute the initial subgradients.
18 sg = zeros(d+1,n);
19 for j=1:n
20     sg(:,j) = - y(j) * Xt(:,j);
21 end
22
23 % The average subgradient
24 m = mean(sg, 2);
25
26 % Apply stochastic average gradient (SAG) method.
27 for t = 1:maxIter
28     if mod(t-1,n) == 0
29         % Plot our progress.
30         % (turn this off for acceleration)

```

```

31
32         objValues(1+(t-1)/n) = (1/n)*sum(max(0,1-y.*(X*w))) + (lambda/2)*(w'*w);
33         semilogy([0:t/n],objValues);
34         pause(.1);
35     end
36
37     % Pick a random training example.
38     i = randi(n);
39
40     % Compute the i-th subgradient, and renew the average subgradient.
41     m = m - sg(:,i) / n;
42     [~, sg(:,i)] = hingeLossSubGrad(w,Xt,y,i);
43     m = m + sg(:,i) / n;
44
45     % Set step size.
46     alpha = 1 / (lambda * t);
47
48     % Take stochastic subgradient step.
49     w = w - alpha * (m + lambda * w);
50 end
51
52 model.w = w;
53 model.predict = @predict;
54
55 end
56
57 function [yhat] = predict(model,Xhat)
58 d = size(Xhat,1);
59 Xhat = [ones(d,1), Xhat];
60 w = model.w;
61 yhat = sign(Xhat * w);
62 end
63
64 function [f,sg] = hingeLossSubGrad(w,Xt,y,i)
65 % Function value
66 wtx = w' * Xt(:,i);
67 f = max(0, 1 - y(i) * wtx);
68
69 % Subgradient
70 if f > 0
71     sg = - y(i) * Xt(:,i);
72 else
73     sg = zeros(size(Xt,1), 1);
74 end
75
76 end

```