

1 Logistic Regression (40 points)**1.1 (10 points) Baye's Rule**

$$p(y = 1) = \alpha$$

$$p(y = 0) = 1 - \alpha$$

$$p(x_i | y = 1) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2}\right)$$

$$\begin{aligned} p(x|y = 1) &= p(x_1|y = 1) * p(x_2|y = 1) * \dots * p(x_D|y = 1) \\ &= \prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2}\right) \end{aligned}$$

$$p(x_i | y = 0) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}\right)$$

$$\begin{aligned} p(x|y = 0) &= p(x_1|y = 0) * p(x_2|y = 0) * \dots * p(x_D|y = 0) \\ &= \prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}\right) \end{aligned}$$

$$p(x) = p(x|y = 0)p(y = 0) + p(x|y = 1)p(y = 1)$$

$$= \prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}\right) * (1 - \alpha) + \alpha * \prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2}\right)$$

$$p(y = 1|x) = \frac{p(x|y = 1) * p(y = 1)}{p(x)}$$

$$\begin{aligned} &= \frac{\prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2}\right) * \alpha}{\prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}\right) * (1 - \alpha) + \prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2}\right) * \alpha} \\ &= \left\{ 1 + \frac{\prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}\right) * (1 - \alpha)}{\prod_{i=1} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2}\right) * \alpha} \right\}^{-1} \end{aligned}$$

$$\begin{aligned}
&= \left\{ 1 + \left(\frac{1-\alpha}{\alpha} \right) \exp \left(- \sum_{i=1} [(x_i - \mu_{i0})^2 - (x_i - \mu_{i1})^2] / (2\sigma_i^2) \right) \right\}^{-1} \\
&= \left\{ 1 + \exp \left(-\log \left(\frac{\alpha}{1-\alpha} \right) - \sum_{i=1} [(x_i - \mu_{i0})^2 - (x_i - \mu_{i1})^2] / (2\sigma_i^2) \right) \right\}^{-1} \\
&= \left\{ 1 + \exp \left(- \sum_i \frac{[x_i^2 - 2x_i\mu_{i0} + \mu_{i0}^2 - x_i^2 + 2x_i\mu_{i1} - \mu_{i1}^2]}{2\sigma_i^2} - \log \left(\frac{\alpha}{1-\alpha} \right) \right) \right\}^{-1} \\
&= \left\{ 1 + \exp \left(- \sum_i \left[\frac{(-2\mu_{i0} + 2\mu_{i1})}{2\sigma_i^2} x_i + \frac{(\mu_{i0}^2 - \mu_{i1}^2)}{2\sigma_i^2} \right] - \log \left(\frac{\alpha}{1-\alpha} \right) \right) \right\}^{-1} \\
&= \frac{1}{1 + \exp \left(- \sum_i \left[\frac{(-\mu_{i0} + \mu_{i1})}{\sigma_i^2} x_i \right] - \left[\sum_i \frac{(\mu_{i0}^2 - \mu_{i1}^2)}{2\sigma_i^2} + \log \left(\frac{\alpha}{1-\alpha} \right) \right] \right)} \\
&= \frac{1}{1 + \exp(-w^T x - b)}
\end{aligned}$$

Thus,

$$w = (w_1, w_2, \dots, w_D), \quad \text{where } w_i = \frac{-\mu_{i0} + \mu_{i1}}{\sigma_i^2}, \quad \forall i \in \{1, 2, \dots, D\}.$$

$$b = \sum_i \left(\frac{\mu_{i0}^2 - \mu_{i1}^2}{2\sigma_i^2} \right) + \log \left(\frac{\alpha}{1-\alpha} \right)$$

1.2 (15 points) Maximum Likelihood Estimation

Let us define the likelihood function as

$$\begin{aligned}
lik &= p(\mathbf{y}|\mathbf{x}, \mathbf{w}, b) \\
&= \prod_{i=1}^N p(y^i = 1|x^i, w, b)^{y^i} * p(y^i = 0|x^i, w, b)^{1-y^i} \\
&= \prod_{i=1}^N \left(\frac{1}{1 + \exp(-(\sum_{j=1}^D w_j x_j^i + b))} \right)^{y^i} \\
&\quad * \left(\frac{\exp(-(\sum_{j=1}^D w_j x_j^i + b))}{1 + \exp(-(\sum_{j=1}^D w_j x_j^i + b))} \right)^{1-y^i}
\end{aligned}$$

$$\text{Let } z = \sum_{j=1}^D w_j x_j^i + b, \quad \zeta^i = \frac{1}{1 + \exp(-z)}.$$

$$E(w, b) = -\log(lik)$$

$$\begin{aligned}
&= -\log\left(\prod_{i=1}^N \zeta^{y^i} * \exp(-z)^{1-y^i} * \zeta^{1-y^i}\right) \\
&= -\sum_{i=1}^N (y^i \log(\zeta^i) + (1-y^i)(-z) + (1-y^i) \log(\zeta^i)) \\
&= -\sum_{i=1}^N ((y^i - 1)z + \log(\zeta^i))
\end{aligned}$$

Plug in $z^i = \sum_{j=1}^D w_j x_j^i + b$, $\zeta^i = \frac{1}{1+\exp(-z^i)}$,

$$\begin{aligned}
E(\mathbf{w}, b) &= -\sum_{i=1}^N ((y^i - 1)z^i + \log(\zeta^i)) \\
&= -\sum_{i=1}^N \left((y^i - 1) \sum_{j=1}^D (w_j x_j^i + b) \right. \\
&\quad \left. + \log\left(\frac{1}{1 + \exp(-\sum_{j=1}^D (w_j x_j^i + b))}\right) \right)
\end{aligned}$$

Then we can get the partial derivatives by applying Chain Rule:

$$\begin{aligned}
\frac{\partial E}{\partial w_j} &= \frac{\partial E}{\partial z^i} \frac{\partial z^i}{\partial w_j} \\
&= \sum_{i=1}^N \left(1 - y^i - \frac{\exp(-\sum_{j=1}^D w_j x_j^i - b)}{1 + \exp(-\sum_{j=1}^D w_j x_j^i - b)} \right) x_j^i \\
&= \sum_{i=1}^N \left(\frac{1}{1 + \exp(-\sum_{j=1}^D w_j x_j^i - b)} - y^i \right) x_j^i \\
\frac{\partial E}{\partial b} &= \frac{\partial E}{\partial z^i} \frac{\partial z^i}{\partial b} \\
&= \sum_{i=1}^N \left(1 - y^i - \frac{\exp(-\sum_{j=1}^D w_j x_j^i - b)}{1 + \exp(-\sum_{j=1}^D w_j x_j^i - b)} \right) 1 \\
&= \sum_{i=1}^N \left(\frac{1}{1 + \exp(-\sum_{j=1}^D w_j x_j^i - b)} - y^i \right)
\end{aligned}$$

1.3 (15 points) L2 Regularization

The probabilities of the parameters are given by

$$p(w_i) = \frac{1}{\sqrt{2\pi\frac{1}{\lambda}}} \exp\left(-\frac{(w_i - 0)^2}{2\left(\frac{1}{\lambda}\right)}\right) = \sqrt{\frac{\lambda}{2\pi}} \exp\left(-\frac{w_i^2 \lambda}{2}\right)$$

$$p(b) = \sqrt{\frac{\lambda}{2\pi}} \exp\left(-\frac{\lambda b^2}{2}\right)$$

Since

$$\textbf{posterior} = \frac{\textbf{class likelihood} * \textbf{prior}}{\textbf{evidence}}$$

$$p(w, b|D) = \frac{p(D|\mathbf{w}, b)p(\mathbf{w}, b)}{p(D)}$$

$$p(D)p(\mathbf{w}, b|D) = p(D|\mathbf{w}, b)p(\mathbf{w}, b) = p(D|\mathbf{w}, b)p(\mathbf{w})p(b)$$

$$\text{Since } p(D) = \text{constant}, \quad p(D|\mathbf{w}, b)p(\mathbf{w})p(b) \propto p(\mathbf{w}, b|D)$$

Since $D = \{(x^1, y^1), \dots, (x^N, y^N)\}$, where x 's are the inputs.

$$p(D|\mathbf{w}, b) = p(\{(x^1, y^1), \dots, (x^N, y^N)\}|\mathbf{w}, b) = p(\mathbf{y}|\mathbf{x}, \mathbf{w}, b)$$

$$p(D|\mathbf{w}, b)p(\mathbf{w})p(b) = \text{lik} * \prod_{j=1}^D p(w_j) * p(b)$$

$$L(w, b) = -\log p(D|\mathbf{w}, b)p(\mathbf{w}, b)$$

$$= -\log(\text{lik} * \prod_{j=1}^D p(w_j) * p(b))$$

$$= -\log \text{lik} - \sum_j \log p(w_j) - \log p(b)$$

$$= E(w, b) - \sum_{j=1}^D \log\left(\sqrt{\frac{\lambda}{2\pi}} \exp\left(-\frac{\lambda w_j^2}{2}\right)\right) - \log\left(\sqrt{\frac{\lambda}{2\pi}} \exp\left(-\frac{\lambda b^2}{2}\right)\right)$$

$$= E(w, b) - \frac{D}{2} \log \frac{\lambda}{2\pi} + \sum_{j=1}^D \left(\frac{\lambda w_j^2}{2}\right) + \frac{\lambda b^2}{2} - \frac{1}{2} \log \frac{\lambda}{2\pi}$$

$$= E(w, b) + \left(\frac{\lambda}{2}\right) \sum_{j=1}^D w_j^2 + \frac{\lambda b^2}{2} + \left(-\frac{D+1}{2} \log \frac{\lambda}{2\pi}\right)$$

$$= E(w, b) + \frac{\lambda}{2} \sum_{j=1}^D w_j^2 + \frac{\lambda}{2} b^2 + C(\lambda), \quad \text{for } C(\lambda) = \left(-\frac{D+1}{2} \log \frac{\lambda}{2\pi} \right)$$

Derivatives with respect to each of the model parameters are,

$$\frac{\partial L}{\partial b} = \frac{\partial E}{\partial b} + \lambda b = \sum_{i=1}^N \left(1 - y^i - \frac{\exp(-\sum_{j=1}^D w_j x_j^i - b)}{1 + \exp(-\sum_{j=1}^D w_j x_j^i - b)} \right) + \lambda b$$

$$\frac{\partial L}{\partial w_j} = \frac{\partial E}{\partial w_j} + \lambda w_j = \sum_{i=1}^N \left(1 - y^i - \frac{\exp(-\sum_{j=1}^D w_j x_j^i - b)}{1 + \exp(-\sum_{j=1}^D w_j x_j^i - b)} \right) (x_j^i) + \lambda w_j$$

2. Digit classification (60 points)

2.1 (10 points) k-Nearest Neighbors

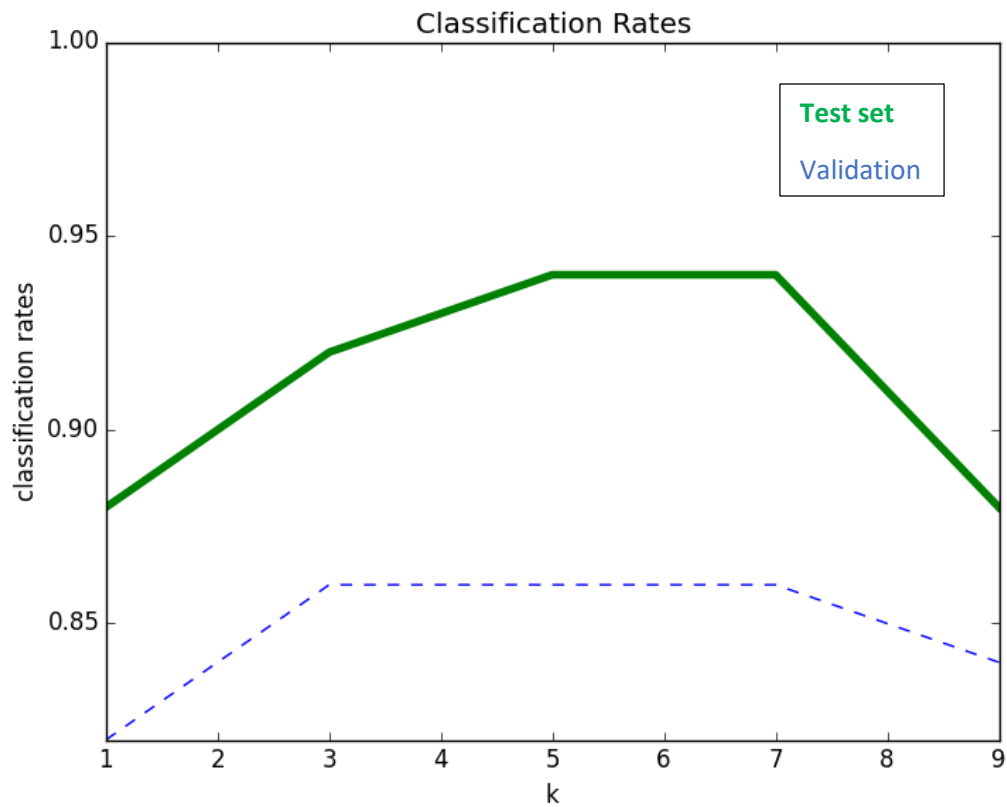
In k-Nearest Neighbors classification, we are going to classify a new data point by computing the distances from it to k nearest labeled data points, where k is some odd number. Then we will label the new data point as the class which has the majority of the k labels.

Since there are only 80 examples in the training set, the model will include too many mislabeled neighbors if k is too large, thus leads to misclassification. This can be shown by calculating classification rates of different values of k, which is shown below. The function of classification rate is non-increasing with respect to k, for k larger than 7. Conversely, the model will not have sufficient amount of data (i.e. labeled neighbors) if k is too small.

According to the output information below, the best k is among 3, 5, 7 for the test data set. $k^* = 5$ may be a better choice because it is more stable than the other two. The classification rates for all k's are given below.

The classification rates correspond to k:

k	Classification rate on Validation set %	Classification rate on Test set %
1	0.82	0.88
3	0.86	0.92
5	0.86	0.94
7	0.86	0.94
9	0.84	0.88



The above information provided above suggests that the test performance for the values of k does correspond to the performance of the validation set. The best values of k can be different depends on the existence of noise terms.

2.2 (10 points) Logistic regression

Since there is no penalty term in this method, the `weight_regularization` parameter will not be used throughout question 2.2. The parameters that can be varied are `learning_rate`, `num_iterations` and the way to initialize weights.

A good model should have low uncertainty and high correctness, thus it should have a small cross entropy and a large fraction of correct. After experimenting on different sets of parameters, and examining on the cross entropy and fraction of correctly classification, we can find then find a best model. (The experiment data and process is in Appendix on the last page pf the write-up.)

❖ Statistics on `minist_train_small`

- The hyerparameter setting used:

Learning rate: 0.01

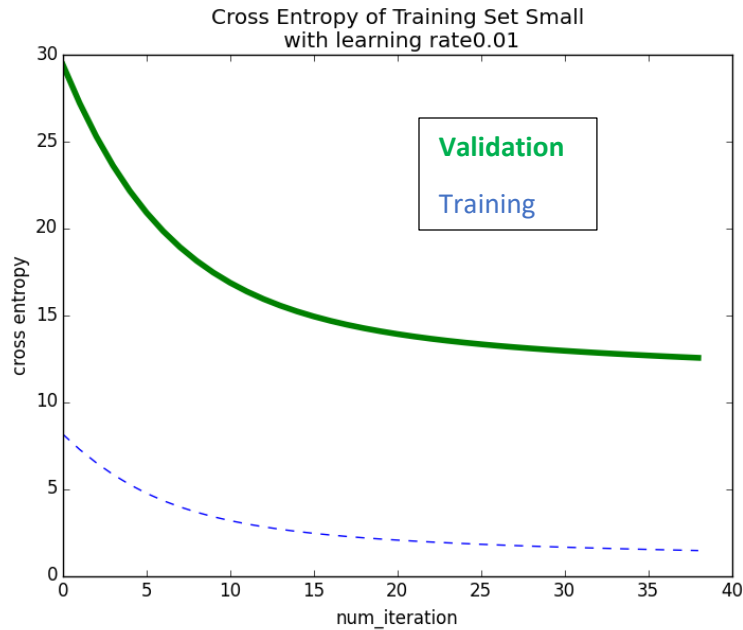
Weight regularization: 1

Number of iterations: 39

Initial weights: follows Gaussian(0, 0.01)

- Statistics on `mnist_train_small`:

Data set	cross_entr	frac_corr%
Mnist_train_small	1.214	90.00
Validation set	11.255	54.00
Test set	9.599	68.00



- **The above information shows:**
 1. Cross entropy curves for both small training set and validation set converges.
 2. Fraction correctness of the small training set is not 100.00, which means the model did not overfit the data set.

❖ **Statistics on mnist_train:**

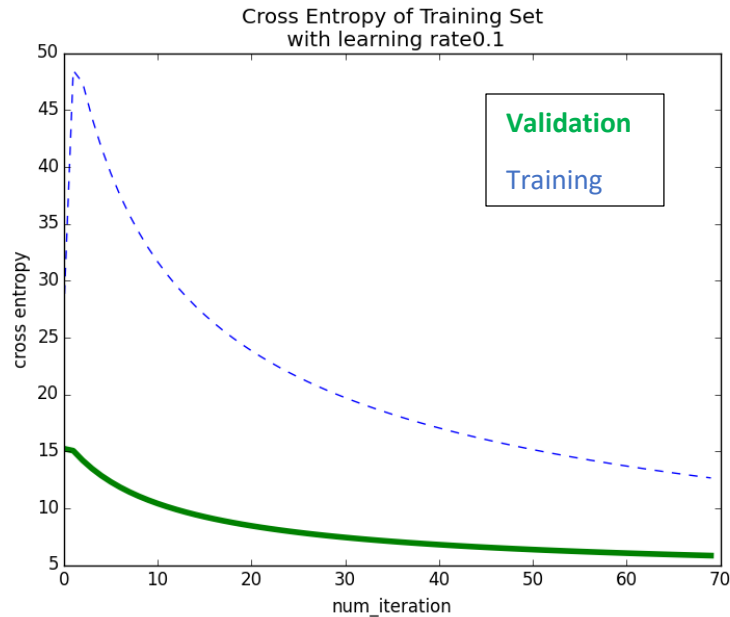
- **Hyperparameters used on this data set:**

Since a larger learning rate is used here for a faster converging, the number of iterations is adjusted as well.

Learning rate: 0.1
 Weight regularization: 1
 Number of iterations: 40
 Initial weights: follows Gaussian(0, 0.01)

- **Statistics:**

Data set	cross_entr	frac_corr%
Mnist_train	17.04	97.5
Validation set	8.0579	84.00
Test set	5.861	90.00



- **The above information shows:**

1. Both curves are converging, which indicates that the model is converging at the minimum of loss function.
2. This is a good model because the cross entropy values for both validation and training sets are low, which means the uncertainty of the classification is low. Also, the fraction correctness is pretty high for all data sets.

❖ **Overall**

Comparing results from both training sets, the larger training set trains the model much better than the smaller one because the cross entropy for both sets are around the same level, but fraction correctness for large training set is much higher than the small. The reason is with more historical data, the model will get a higher accuracy on its classification.

The results will change slightly if we run the program several times. This is because we are assigning the initial weights randomly at each call. If the variance of the Gaussian model is large, the results will differ more. Therefore, we can pick a small variance in order to restrict on the weights, thus changes in results can be reduced. Alternatively, we can run the program several times and take the mean of the data in each run, to get an average on model performance.

2.3 (10 points) Penalized logistic regression

❖ **minist_train**

- This hyperparameter settings is used:

Learning rate: 0.1
Weight regularizations (λ): {0.001, 0.01, 0.1, 1.0}
Number of iterations: 70
Initial weights: follows Gaussian(0, 0.1)

- **Test data on mnist_train corresponding to $\lambda \in [0.001, 0.01, 0.1, 1.0]$**

Training set:

frac_error: [0.02 0.02125 0.019375 0.015625]

ce: [12.94917253 12.95216837 12.73485931 13.13836072]

Validation set:

frac_error: [0.116 0.106 0.104 0.104]

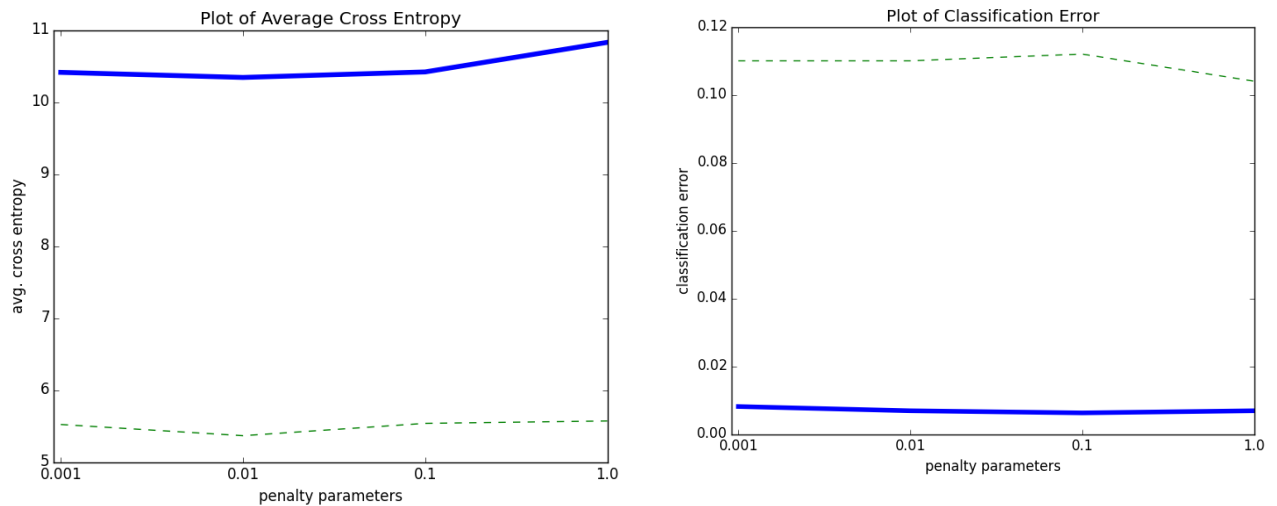
ce: [6.08488291 6.05699394 6.26602178 6.04909971]

Testing set:

frac_error: [0.076 0.096 0.082 0.078]

ce: [3.80139734 4.15496006 4.02944961 3.97020806]

- **Plots of cross entropy against regularization parameters λ :**



Note: The values on the x-axis are not to scale.

Validation

Training

❖ **minist_train_small**

- **Hyperparameter settings** used on this data set:

Learning rate: 0.01
 Weight regularizations (λ): {0.001, 0.01, 0.1, 1.0}
 Number of iterations: 50
 Initial weights: follows Gaussian(0, 0.1)

- **Test data on minist_train_small corresponding to $\lambda \in [0.001, 0.01, 0.1, 1.0]$:**

Training set:

frac_error: [0.02 0.01 0.01 0.]
 ce: [0.87178067 0.8527974 0.90033255 0.86834025]

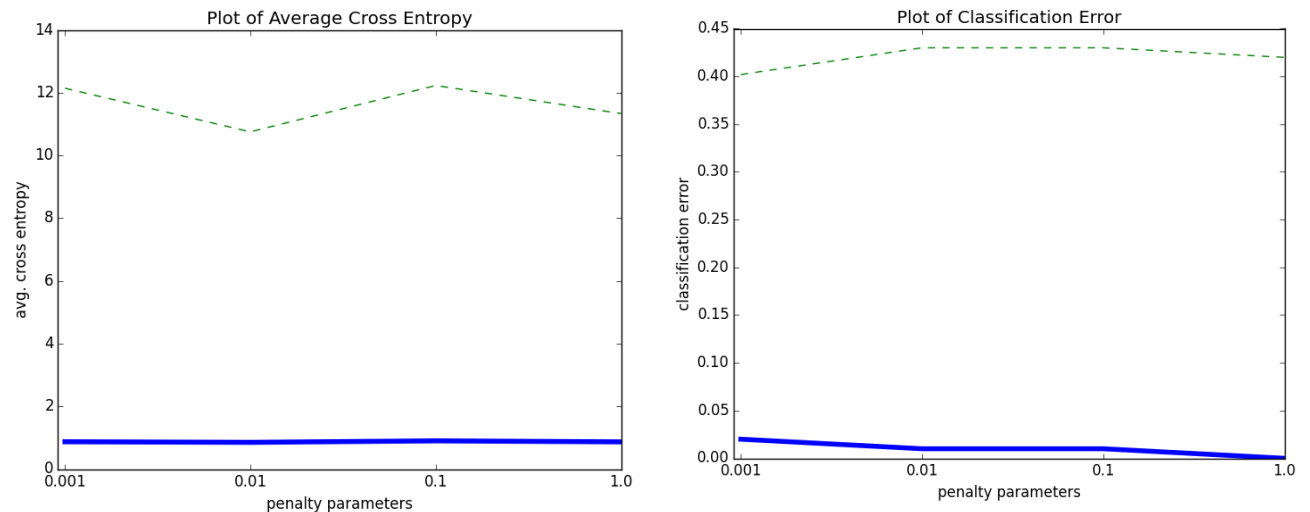
Validation set:

frac_error: [0.402 0.43 0.43 0.42]
 ce: [12.14076983 10.74983207 12.22768712 11.33352384]

Testing set:

frac_error: [0.278 0.294 0.316 0.304]
 ce: [8.91892895 8.329423 9.64514705 8.75605653]

- Plots of cross entropy against regularization parameters λ :



Note: The values on the x-axis are not to scale.

Validation

Training

Comments and observations:

1. Models trained by using mnist_train data set perform better than the ones trained using mnist_train_small. This is shown by a lower entropy and fraction error. Classification error curves for validation sets is higher than the training sets in general.
2. In mnist_train plots, both average cross entropy and classification error curves are quite stable. The best penalty parameter vary if we call the function multiple times. In mnist_train_small plots, the curves for validation set fluctuates, with a minimum average cross entropy at 0.01.
3. The data on penalty parameters {0.001, 0.01, 0.1, 1.0} reflects that as the penalty parameter increases, the classification error and the average cross entropy on the validation set both slightly decreases before $\lambda = 0.01$, and increase afterwards. Therefore, the best λ is 0.01 based on the results.

Explanation:

An appropriate penalty term will help the model to fit the data well without having large weight parameters. If the penalty term is too small, then it will not regularize the weight parameters well. If the penalty term is too large, then there will be

too many 0's in the weight parameter, thus the model will have too little parameters to be predictive.

The test error for $\lambda = 0.01$:

λ	0.1
Mnist_train	0.096
Mnist_train_small	0.294

Compare the results with and without penalty,

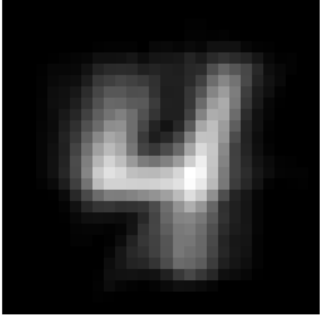
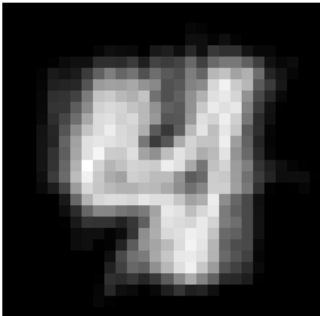
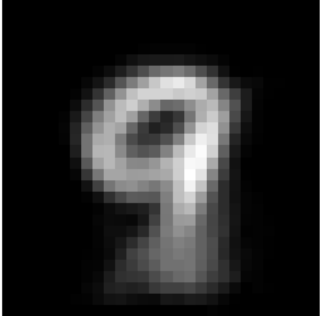
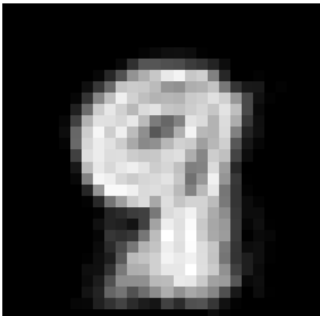
	Training set	Testing set	Cross entropy	Fraction error
Without regularization	Mnist_train	Validation	8.0579	0.16
		test	5.861	0.10
	Mnist_train_small	Validation	11.255	0.46
		test	9.599	0.32
With regularization $\lambda = 0.01$	Mnist_train	Validation	6.0570	0.106
		test	4.1549	0.096
	Mnist_train_small	Validation	10.7498	0.43
		Test	8.3294	0.294

The data suggests that the model with regularization performs better than without regularization. Penalty parameters are used in logistic regression to prevent the models from overfitting due to the existence of some extremely weighted parameters (i.e. some extremely high w_i 's). A penalty term will shrink every weight parameter during the updating process (i.e. gradient descent).

2.4 (15 points) Naïve Bayes

Test accuracy for both validation and testing sets are 0.8, which means 80% of classifications using Naïve Bayes model are correct.

Visualizations of both classes:

mean	variance
	
	

Comments on the visualization results:

The means for both classes are nice and clear, showing that the classification is correct in general. Only the bottom of the 4's and 9's are blurry, indicating that people have different habits in writing the two digits.

The variance represents how far the data of a hand-written digit is from the mean. In another word, the variance shows how far the digits are spread out. In terms of visualization, variance is the narrow area right beside the variance. Therefore, it is reasonable that the variance of the classes look like a bigger and fatter mean but empty in the middle.

2.5 (15 points) Compare k-NN, Logistic Regression, and Naïve Bayes

Test accuracy for each model:

Model	Validation set accuracy	Test set accuracy
k-NN (k=5)	0.86	0.94
Logistic regression without penalty	0.84	0.90
Logistic regression with penalty	0.896	0.918
Naïve Bayes	0.8	0.8

1. The performance **of k-nn** models depends on the sets to be classified, as it is a non-deterministic method. Since k-nn models has piece-wise linear (i.e. nonlinear) decision boundaries, it does NOT depend on whether the data set is linear separable. If the data set does not contain too many noises, then k-nn can be a good model.
2. In **logistic regression**, has linear decision boundary on data sets. Thus, it performs better on data sets that are linear separable. In addition, over-fitting training set can include the noise terms and cause problems.
3. **Naïve bayes model** has simple classifiers and performs well in general. If the conditional independence assumption holds, then the model will converge fast and will classify accurately.

In conclusion, models depends on the data sets we are using. Different models may perform differently on different data sets. Before applying different models, it is worthwhile to think about the features of the data (i.e. linear separability, size, etc.). A large training set is usually better in providing more accurate models (i.e. less prone to overfit), but it may be expensive in training if the training set is too large. Try to train different models and compare the cross entropies and fraction correctness will provide suggestions on which model to use.

Appendix

2.2 Process of selecting the best hyperparameter settings.

Note: **cross_entr and **frac_corr** represent the cross entropy and fraction correct rate on the last iteration on **the validation set**. **

- **Statistics on mnist_train**, where weights follows Gaussian(0, 1):

Learning_rate	num_iter	Weights (w_i)	cross_entr	frac_corr
0.001	100	Gaussian(0, 1)	400.807391	50.00
0.001	1000	Gaussian(0, 1)	54.946939	42.00
0.01	100	Gaussian(0, 1)	57.025413	64.00
0.01	70	Gaussian(0, 0.1)	11.567	76.00
0.01	200	Gaussian(0, 0.1)	10.0654	74
0.01	35	Gaussian(0,1)	12.998	62.00
0.1	100	Gaussian(0, 1)	12.858099	82.00
0.5	100	Gaussian(0, 1)	4.138122	88.00
1	100	Gaussian(0, 1)	15.162320	90.00
1	100	0.001	6.065	88.00
1	100	0.001	6.065	88.00
10	100	Gaussian(0, 1)	65.475161	88.00

Note: the cross entropy and fraction correctness data can differ depends on the initial weights.

With learning rate too small, the model with weights initialized as Gaussian(0, 1) will take longer to converge. Therefore, the cross entropy at the 100th run is still large. For learning rate equals to 0.001, number of iterations for the model to converge will be very large. At the 100th iteration, the cross entropy is large, which indicates that the classification has a very high uncertainty. Thus, we need to higher the number of iterations. However, if we set num_iter to 1000, the maximum frac_corr occurs around the 700th iteration, with a relatively large cross entropy. Cross entropy keeps decreasing before the 1000th iteration, with fraction correctness bouncing below. It will be noticed that the curves are jumping by observing the curves of cross entropy and fraction correctness against the number of iterations.

Reset the variables and repeat the steps to find a model with low cross entropy, high fraction correctness and smooth curves. After many attempts, a reasonable learning rate is found to be around 0.1 on mnist_train, and 0.01 on mnist_train_small.