CSE 446: Machine Learning
Winter 2018

Assignment 3 Bonus
with bonus questions

from
Lukas Nies
University of Washington

02/26/18

Contents

0 Policies

0.1 List of Collaborators

My collaborator was Edith Heiter (discussed parts of Problem 1, 4, and 5). The development of the answers and code though was completely independent and individually.

0.2 List of Acknowledgments

None.

0.3 Policies

I have read and understood these policies.

0.4 Note: Bonus included!

In this version some of the bonus questions are included.

1 Problem: Linear Regression on MNIST

1.1 Closed Form Estimator

1. If one runs the Closed Form Estimator with $\lambda = 0$ one encounters trying to invert a singular matrix $(X^T X)$ which is not possible per definition since the determinant is $\det(X^T X) = 0$. The matrix is therefore not invertible. To avoid this we introduce a regularization by adding the term $\lambda \mathbb{1}_d$. This is intuitively clear by considering the data itself: one digit consists of 28×28 pixels where most pixels (at the edges and in the corners) don't carry any information about the digit itself (matrix is sparse). When calculating $X^T X$ we get the same result: we have more "dimensions" than information for those "dimensions". In mathematical terms: $X^T X$ is under-determined.

2. For this part a grid search was implemented to search for different values of λ and the threshold to optimize the performance on the development set:

   (a) The best result was found with $\lambda = 0.02$ and a threshold of 0.5. The grid search ran for λ from 0.01 to 1 with steps of 0.01, the treshold ran from 0.1 to 1.0 in steps of 0.1.

   (b) The average squared error using the parameters stated above is as follows:
   - Training error = 0.013045
   - Development error = 0.01420
   - Test error = 0.01626

   (c) The misclassification error using the parameters stated above is as follows:
   - Training error = 0.93%
   - Development error = 1.08%
   - Test error = 1.76%

3. Samples with large values (far off the mean of the rest of the data points) have a strong influence on linear polynomial functions fitted through regression. This leads to large misclassification on most of the data points. A better model would be using a higher order polynomial to fit those samples more efficiently.

1.2 Linear regression using gradient descent

1. The proof is as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}_\lambda}{\partial w} &= \frac{\partial}{\partial w}\left(\frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}\left(y_n - w^T x_n\right)^2 + \frac{\lambda}{2}\|w\|^2\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\left(-\frac{2x_n}{2}\right)\left(y_n - w^T x_n\right) + \left(\frac{2\lambda}{2}\mathbf{w}\right) \\
&= -\frac{1}{N}\sum_{n=1}^{N}\left(y_n - \hat{y}_n\right)x_n + \lambda\mathbf{w}
\end{aligned}$$

2. We can rewrite this as a matrix expression:

$$\frac{\partial \mathcal{L}_w}{\partial w} = -\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)\,x_n + \lambda \mathbf{w} = -\frac{1}{N}X^T \cdot \left(Y - \hat{Y}\right) + \lambda \mathbf{w}$$

3. Stepsizes $-10^{-2} \leq \eta < -10^{-1}$ worked well for this problem. For the error rate see figure 1. For generating the plots, $\eta = \frac{1}{4} \times 10^{-1}$ and $\lambda = 10^{-2}$ were chosen. The lowest error I achieved is comparable to the closed form estimator, with

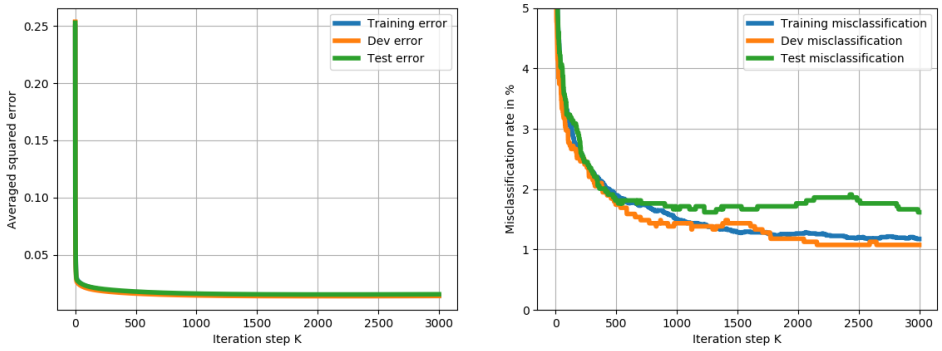

Figure 1: Plot of averaged squared errors (left) and misclassification loss in percent (right) for the gradient descent algorithm. For generating the plots, $\eta = \frac{1}{4} \times 10^{-1}$ and $\eta = 10^{-2}$ were chosen.

1.08% on the development set.

1.3 Linear Regression Using Stochastic Gradient Descent

1. The stochastic gradient descent diverges in this case with a learning rate for about $\eta = -0.1$ at $\lambda = 0.005$. For too large values of η the algorithm might never find the global minimum and therefore the gradient gets larger and larger which leads to divergence.

2. Stepsizes $\eta \leq -10^{-1}$ worked well for this problem. For the error rate see figure 2. For generating the plots, constant $\eta = \frac{1}{4} \times 10^{-1}$ and $\eta = 0.005$ were chosen. The lowest error I achieved is comparable to the closed form estimator, with 1.03% on the development set.

1.4 BONUS: Mini-batch stochastic gradient descent

1. (a) The stochastic gradient descent diverges in this case with a learning rate for about $\eta = -0.01$ at $\lambda = 0.005$. This is ten times less than the stochastic gradient descent with $m = 1$.

   (b) Stepsizes $-10^{-3} \leq \eta < -10^{-2}$ worked well for this problem. For the error rate see figure 3. For generating the plots, $\eta = \frac{1}{4} \times 10^{-2}$ and $\lambda = 0.005$ were chosen. The lowest error I achieved was roughly 2.2% on the development set.

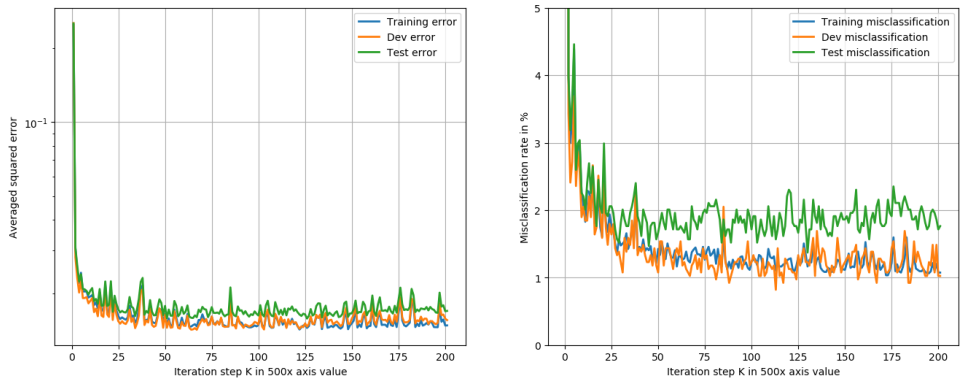

Figure 2: Plot of averaged squared errors (left, note the logarithmic vertical axis) and misclassification loss in percent (right) for the stochastic gradient descent algorithm. The horizontal axis shows the iteration steps for every 500th step. For generating the plots, $\eta = \frac{1}{4} \times 10^{-1}$ and $\eta = 0.005$ were chosen.

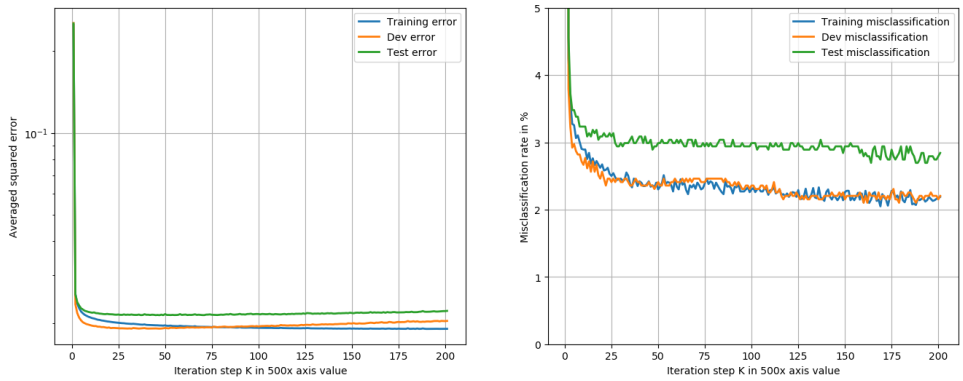

Figure 3: Plot of averaged squared errors (left, note the logarithmic vertical scale) and misclassification loss in percent (right) for the mini batch stochastic gradient descent algorithm with $m = 100$. For generating the plots, $\eta = \frac{1}{4} \times 10^{-2}$ and $\lambda = 0.005$ were chosen. The horizontal axis shows the iteration steps for every 500th step.

2. By comparing plot 2 and 3 we find that the performance for the mini-batch algorithm is slightly worse than the normal stochastic gradient descent. The error rates increase by roughly 1%. In general the curves are more smooth since the gradient gets updated based on 100 random samples and not only on one random sample.

1.5 BONUS: Using polynomial features

1. We can calculate the dimensionality of the $\phi(x)$ vector by looking at at generic matrix M of dimension d. The matrix consists of d^2 elements, where d are diagonal elements and $d^2 - d$ are non-diagonal elements. According to the problem, we need only the diagonal elements and half of the non-diagonal

elements (to avoid counting pairs twice), so:

$$d_{\phi(x)} = \frac{d^2 - d}{2} + d = \frac{d^2 + d}{2} \tag{1}$$

2. If we set $d = 784$ then we would get $d_{\phi(x)} = 307720$ features per sample. This isn't a good idea since the computation time for N=10000 samples with that amount of features would increase a lot because this results in roughly 10^9 data points, respectively entries in the matrix. To work witch such a matrix with numpy arrays would fill the RAM with $8 \times 10^9 = 8$GB of memory which probably throws an memory error.

3. For $d = 40$ features we get $d_{\phi(x)} = 820$ features per sample which yields in a massive reduction in computation time and use of memory.

4. Running with standard stochastic gradient descent, stepsizes $-10^{-6} \leq \eta < -10^{-4}$ with $\lambda = 10^{-4}$ worked well for this problem. For the error rate see figure 4. For generating the plots, $\eta = \times 10^{-5}$ and $\lambda = 0.0001$ were chosen. The lowest error I achieved was roughly 0.4%(!) on the development set.

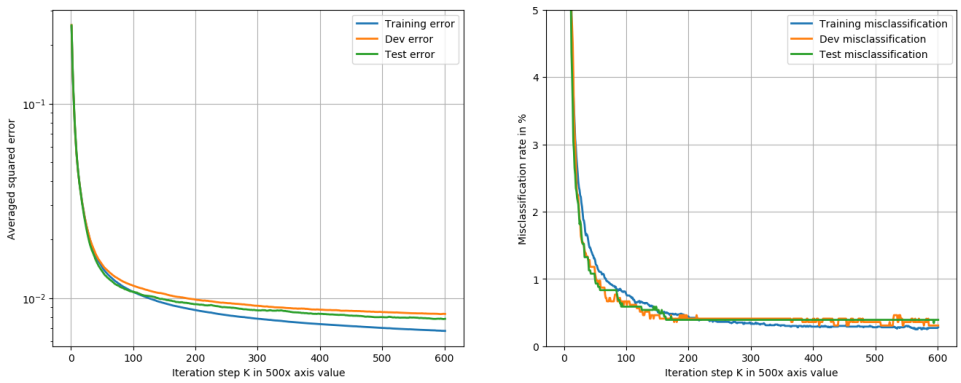

Figure 4: Plot of averaged squared errors (left, note the logarithmic vertical scale) and misclassification loss in percent (right) for the normal stochastic gradient descent algorithm with polynomial features. For generating the plots, $\eta = \times 10^{-5}$ and $\lambda = 0.0001$ were chosen. The horizontal axis shows the iteration steps for every 500th step.

2 Binary Classification with Logistic Regression

1. For proofing this, we look at the cases $y_n = 1$ and $y_n = 0$ separately.
   Case $y_n = 1$:

$$\begin{aligned}
\frac{\partial \mathcal{L}_\lambda}{\partial w} &= \frac{\partial}{\partial w}\left(-\frac{1}{N}\sum_{n=1}^{N}\log p_w(y_n = 1|x_n) + \frac{\lambda}{2}\|\mathbf{w}\|^2\right) \\
&= \frac{\partial}{\partial w}\left(-\frac{1}{N}\sum_{n=1}^{N}\log\frac{1}{1+\exp(-wx_n)} + \frac{\lambda}{2}\|\mathbf{w}\|^2\right) \\
&= \frac{\partial}{\partial w}\left(-\frac{1}{N}\sum_{n=1}^{N}[\log(1) - \log(1+\exp(-wx_n)] + \frac{\lambda}{2}\|\mathbf{w}\|^2\right) \\
&= -\frac{1}{N}\sum_{n=1}^{N}\left[\frac{x_n\exp(-wx_n)}{1+\exp(-wx_n)}\right] + \lambda\mathbf{w} \\
&= -\frac{1}{N}\sum_{n=1}^{N}\left[x_n(\hat{y}_n^{-1} - 1)\hat{y}_n\right] + \lambda\mathbf{w} = -\frac{1}{N}\sum_{n=1}^{N}(1-\hat{y}_n)x_n + \lambda\mathbf{w}
\end{aligned}$$

   Case $y_n = 0$:

$$\begin{aligned}
\frac{\partial \mathcal{L}_\lambda}{\partial w} &= \frac{\partial}{\partial w}\left(-\frac{1}{N}\sum_{n=1}^{N}\log p_w(y_n = 0|x_n) + \frac{\lambda}{2}\|\mathbf{w}\|^2\right) \\
&= \frac{\partial}{\partial w}\left(-\frac{1}{N}\sum_{n=1}^{N}\log\frac{1}{1+\exp(+wx_n)} + \frac{\lambda}{2}\|\mathbf{w}\|^2\right) \\
&= \frac{\partial}{\partial w}\left(-\frac{1}{N}\sum_{n=1}^{N}[\log(1) - \log(1+\exp(+wx_n)] + \frac{\lambda}{2}\|\mathbf{w}\|^2\right) \\
&= -\frac{1}{N}\sum_{n=1}^{N}\left[\frac{-x_n\exp(+wx_n)}{1+\exp(+wx_n)}\right] + \lambda\mathbf{w} = -\frac{1}{N}\sum_{n=1}^{N}\left[\frac{-x_n}{\exp(-wx_n)+1}\right] + \lambda\mathbf{w} \\
&= -\frac{1}{N}\sum_{n=1}^{N} -x_n\hat{y}_n + \lambda\mathbf{w} = -\frac{1}{N}\sum_{n=1}^{N}(0-\hat{y}_n)x_n + \lambda\mathbf{w}
\end{aligned}$$

2. We can rewrite this as a matrix expression:

$$\frac{\partial \mathcal{L}_\lambda}{\partial w} = -\frac{1}{N}\sum_{n=1}^{N}\left(y_n - \hat{y}_n\right)x_n + \lambda\mathbf{w} = -\frac{1}{N}X^T\cdot\left(Y - \hat{Y}\right) + \lambda\mathbf{w}$$

3. Properties of logistic regression

   (a) Suppose the data is linear separable and $\lambda = 0$. In order to fit the data best, one would like to optimize the sigmoid function $\frac{1}{1+\exp(-wx)}$. Since the data is linear separable all, data points with $y_n = 0$ are left of $x = 0$ and all points with $y_n = 1$ are to the right. In this case, the

optimal fit would be the Heaviside function (step function, 0 for $x < 0$, 1 for $x > 0$). In order to optimize the sigmoid function to approach the Heaviside function, $w \to \inf$. Hence, our weight vector would diverge.

(b) If we suppose that $d > n$ ($\lambda = 0$) then the data matrix is sparse and several features will carry no information (equal 0). To fit a larger accumulation of features with value 0 the logistic function must approach, similar to previous question, the Heaviside function. Therefore the weight vector will diverge.

(c) To avoid the divergence of the weight vector one can introduce regularization which avoids sparsity and linear separability such that the algorithm stops early enough to give a good estimation without diverging too fast. If one does not consider this the algorithm might overfit which influences the true error.

2.1 BONUS: Log loss, applied

1. For tackling this problem I chose a batch size of $m = 100$, a step size of $\eta = \frac{1}{4} \times 10^{-2}$ and a parameterization factor of $\lambda = 0.01$.

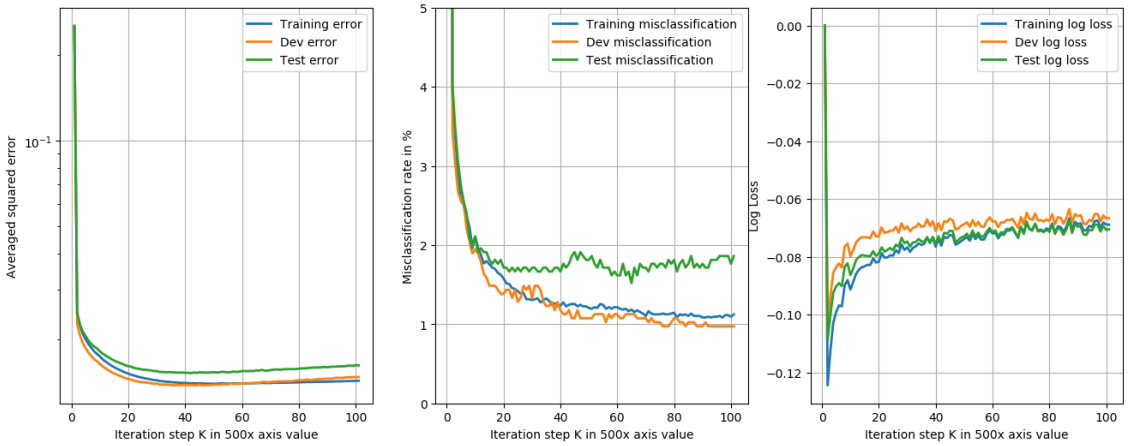

Figure 5: Plot of averaged squared errors (left, note the logarithmic vertical scale), the misclassification loss in percent (middle) and the log loss (right) for the mini batch stochastic gradient descent algorithm with normal features. For generating the plots, $\eta = \times\frac{1}{4} \times 10^{-2}$ and $\lambda = 0.01$ were chosen. The horizontal axis shows the iteration steps for every 500th step.

2. Again, I chose a batch size of $m = 100$, a step size of $\eta = \frac{1}{4} \times 10^{-2}$ and a parameterization factor of $\lambda = 0.01$.

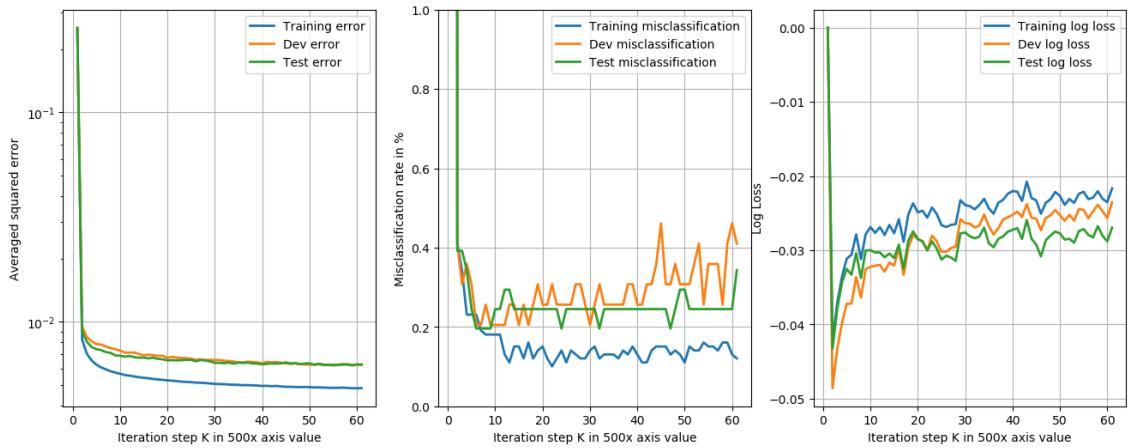

Figure 6: Plot of averaged squared errors (left, note the logarithmic vertical scale), the misclassification loss in percent (middle) and the log loss (right) for the mini batch stochastic gradient descent algorithm with normal features. For generating the plots, $\eta = \times\frac{1}{4} \times 10^{-2}$ and $\lambda = 0.01$ were chosen. The horizontal axis shows the iteration steps for every 500th step.

3 Multi-Class classification using Least Squares

3.1 "One vs. all Classification" with Linear Regression

1. We can rewrite this as a matrix expression:

$$\frac{\partial \mathcal{L}_W}{\partial W} = -\frac{1}{N}\sum_{n=1}^{N} x_n \left(y_n - \hat{y}_n\right)^T + \lambda \mathbf{W} = -\frac{1}{N} X^T \cdot \left(Y - \hat{Y}\right) + \lambda \mathbf{W}$$

2. Stepsizes $-10^{-2} \leq \eta \leq -10^{-1}$ worked well for this problem using the stochastic gradient descent. For the error rate see figure 7. For generating the plots, $\eta = \frac{2}{4} \times 10^{-1}$ and $\eta = 10^{-2}$ were chosen. The lowest error I achieved is roughly

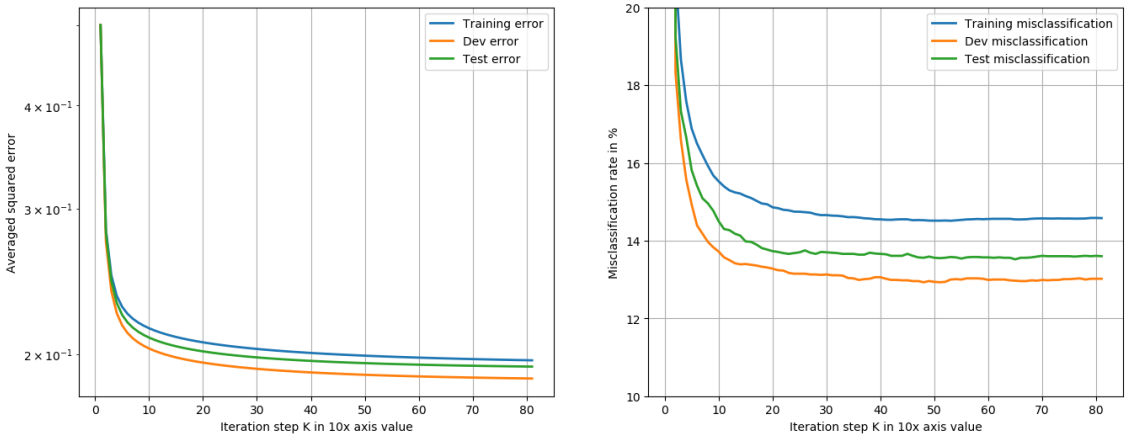

Figure 7: Plot of averaged squared errors (left) and misclassification loss in percent (right) for the gradient descent algorithm. For generating the plots, $\eta = \frac{1}{4} \times 10^{-1}$ and $\eta = 10^{-2}$ were chosen.

13% on the development set.

3.2 BONUS: Matrix derivative

Following the matrix derivative:

$$\begin{aligned}
\mathcal{L}_W &= \frac{1}{2N}\sum_{n=1}^{N} \|y_n - W^T x_n\|_2^2 + \frac{\lambda}{2}\|W\|_2^2 = \frac{1}{2N}\|Y - X \cdot W\|_2^2 + \frac{\lambda}{2}\|W\|_2^2 \\
&= \frac{1}{2N}\left[(Y - X \cdot w)^T (Y - X \cdot w)\right] + \frac{\lambda}{2} W^T W \\
\frac{\partial \mathcal{L}_W}{\partial W} &= \frac{1}{2N}\left[-2X^T(Y - X \cdot W) + \lambda W\right] = -\frac{1}{N} X^T (Y - X \cdot W) + \lambda W \\
&= -\frac{1}{N}\sum_{n=1}^{N} x_n (y_n - \hat{y}_n)^T + \lambda W \qquad (2)
\end{aligned}$$

3.3 BONUS: Softmax

Calculate the derivative according to previous problem parts as a general case of the one hot encoding with k possible different label and the softmax probability distribution. Therefore:

$$\frac{\partial \mathcal{L}_W}{\partial W} = -\frac{1}{N}\sum_{n=1}^{N} x_n(y_n - \hat{y}_n)^T + \lambda W = -\frac{1}{N}X^T(Y - \hat{Y}) + \lambda W \tag{3}$$

The dimensions are

$$X \in \mathbb{R}^{n\times d} \qquad Y \in \mathbb{R}^{n\times k} \qquad \hat{Y} \in \mathbb{R}^{n\times k} \qquad W \in \mathbb{R}^{d\times k}$$

For our dataset $k = 10$.

4 Probability and Maximum Likelihood Estimation

4.1 Probability Review

1. (a) Since the disease is quite rare it is likely that one does not have the disease even if one is tested positive. The amount of healthy persons in a group is larger than the number of sick persons and even if the test is highly accurate it's much more likely that one is healthy given the test is incorrect.

   (b) The probability of being tested positive (P) given having the disease ($\bar{H}$) is given by:

$$P(P|\bar{H}) = \frac{P(\bar{H}|P)P(P)}{P(\bar{H})} = 0.99. \tag{4}$$

   By reversing this Bayesian theorem we can yield the probability having the disease given being tested positive:

$$P(P|\bar{H}) = \frac{P(\bar{H}|P)P(\bar{H})}{P(P)}, \tag{5}$$

   where $P(P)$ is the total probability being tested positive, which is the sum of the probabilities of being tested positive and being sick, or being tested positive and being healthy:

$$P(\bar{H}|P) = \frac{P(P|\bar{H})P(\bar{H})}{P(P)} = \frac{P(P|\bar{H})P(\bar{H})}{P(P|\bar{H})P(\bar{H}) + P(P|H)P(H)} \tag{6}$$

$$= \frac{0.99 \times 10^{-4}}{0.99 \times 10^{-4} + 0.01 \times (1 - 10^{-4})} = \frac{1}{102} = 0.98\% \tag{7}$$

2. We can rewrite the table as follows: It follows:

	S=0	S=1	
C=1	P(S=0 ∩ C=1)	P(S=1 ∩ C=1)	P(C=1)
C=0	P(S=0 ∩ C=0)	P(S=1 ∩ C=0)	P(C=0)
	P(S=0)	P(S=1)	1

	S=0	S=1	
C=1	$\frac{23}{151}$	$\frac{34}{151}$	$\frac{57}{151}$
C=0	$\frac{41}{151}$	$\frac{53}{151}$	$\frac{94}{151}$
	$\frac{64}{151}$	$\frac{87}{151}$	1

(a) $\hat{p}(C=1,S=1) = P(C=1 \cap S=1) = \frac{34}{151} = 22.52\%$

(b) $\hat{p}(C=1|S=1) = \frac{P(C=1 \cap S=1)}{P(S=1)} = \frac{34}{87} = 39.08\%$

(c) $\hat{p}(C=0|S=0) = \frac{P(C=0 \cap S=0)}{P(S=0)} = \frac{41}{64} = 64.06\%$

3. A hat over a parameter denotes an estimator of the parameter. The estimator estimates a probability value of an event based on a limited amount of samples whereas the actual probability of an event is given by looking at all samples.

4.2 Maximum Likelihood Estimation

1. If the observations are independent then the likelihood function is given by the product of the probability mass function which is the Poisson distribution in this case:

$$l(\lambda, G_1, \ldots, G_N) = \prod_{i=1}^{N} \frac{\lambda^{G_i}}{G_i!} \exp(-\lambda) \tag{8}$$

Therefore the log-likelihood function is given applying the (natural) logarithm:

$$\begin{aligned} L(\lambda, G_1, \ldots, G_N) &= \ln\left(\prod_{i=1}^{N} \frac{\lambda^{G_i}}{G_i!} \exp(-\lambda)\right) = \sum_{i=1}^{N} \left[\ln\left(\frac{\lambda^{G_i}}{G_i!} \exp(-\lambda)\right)\right] \\ &= \sum_{i=1}^{N} \left[\ln(\lambda^{G_i}) - \ln(G_i!) + \ln(\exp(-\lambda))\right] \\ &= \sum_{i=1}^{N} \left[G_i \ln(\lambda) - \ln(G_i!) - \lambda\right] \end{aligned} \tag{9}$$

2. In order to find the MLE for λ we need to solve

$$\hat{\lambda} = \arg\max_{\lambda}(L(\lambda, G_1, \ldots, G_N)) \tag{10}$$

by taking the derivative in respect to λ:

$$\frac{\partial}{\partial \lambda}\left(L(\lambda, G_1, \ldots, G_N)\right) = 0. \tag{11}$$

$$\begin{aligned} &\Leftrightarrow 0 = \frac{\partial}{\partial \lambda}\left(\sum_{i=1}^{N} \left[G_i \ln(\lambda) - \ln(G_i!) - \lambda\right]\right) \\ &\Leftrightarrow 0 = \sum_{i=1}^{N} \left[\frac{G_i}{\hat{\lambda}} - 1\right] = -N + \frac{1}{\lambda}\sum_{i=1}^{N} G_i \\ &\Leftrightarrow \hat{\lambda} = \frac{1}{N}\sum_{i=1}^{N} G_i. \end{aligned} \tag{12}$$

3. The MLE for λ using the observed G is therefore:

$$\hat{\lambda} = \frac{1}{N}\sum_{i=1}^{N} G_i. = \frac{1}{8}(6+4+2+7+5+1+2+5) = 4 \tag{13}$$

5 BONUS: State of the art on MNIST

Disclaimer: As stated bt the instructor in the problem set we do not have a dev set. To create a small dev set I chopped of 30 data points from the test set to roughly tune my hyperparameters on this small dev set. This size is chosen to lie within the $3\sigma = 99.7\%$ confidence interval to guaranty a good estimation of the test error even by looking a little bit on the test set. As it appears after finishing this problem, this was a good choice to estimate the squared average error but not the misclassification rate.

A word to bias: after failing by adding bias before mapping the features I added bias after mapping as one data point in the mini batch sample. This leads to some improvement in the overall results.

5.1 BONUS: Start small ($k = 5000$)

1. I am aware that for different choices of λ the algorithm diverges sooner or later. For very small λ one can choose higher step sizes without diverging too early, for larger λ one must use smaller step size to avoid diverging. For $\eta = 0.0001$ and $\lambda = 0.01$ I found good results. The learning rate was constant. For this given λ the algorithm diverges for $\eta = 10^{-2}$.

2. See figure 8.

3. See figure 8.

4. See figure 8. Comment: The norm of the weight vector appears to look like a root function or logarithm function. Since neither of those two function families converges for large iterations I can not make an assumption about convergence of the norm. But to me it looks like for large iterations the norm might be upper bounded by one.

5.
   - Lowest training avg. sq. error: 0.05756
   - Lowest test avg. sq. error: 0.07140

6.
   - Lowest training miscl. error: 1.30%
   - Smallest # of total mistakes for training set: 780/60000
   - Lowest test miscl. error: 2.40%
   - Smallest # of total mistakes for test set: 239/10000

   Comment on overfitting: It clearly can be seen that the misclassification error for the training set and test set separate for early iteration steps (around step 500 to 1000). For late iterations, both error rates roughly converge with a difference of about 1%.
   Comment on choice of dev set: As stated before, the (tiny) dev set gives a good estimation for the squared error but is a little off for estimating the misclassification rate. Though it gave some intuition about the order of magnitude of the errors.

5.2 BONUS: Go big ($k = 60000$)

For evaluating the errors in this part I chose a subset of 3000 random data points in test and training set. The dev set was chosen like the one in previous part.

1. For $\lambda = 0.01$ and $\eta = 10^{-3}$ I found good results. The learning rate was constant. For this given λ the algorithm diverges for $\eta = 10^{-2}$.
2. See figure 9.
3. See figure 9.
4. See figure 9. Comment: Again one can find a root like or logarithmic like behavior but compared to the smaller feature size the slope for later iterations is steeper.
5. In order to get good run times and to prevent running out of memory, the errors were calculated on a 3k subset.
   - Lowest training avg. sq. error: 0.00687
   - Lowest dev avg. sq. error: 0.03129
   - Lowest test avg. sq. error: 0.03812
6. In order to get good run times and to prevent running out of memory, the errors were calculated on a 3k subset.
   - Lowest training miscl. error: 0.00%
   - Smallest # of total mistakes for training set: 0/60000
   - Lowest dev miscl. error: 0.00%
   - Smallest # of total mistakes for dev set: 0/30
   - Lowest test miscl. error: 1.30%
   - Smallest # of total mistakes for test set: 129/10000
7. Comment on overfitting: In the case for using all the features we can clearly see the curse of overfitting. While the training error literally goes to zero for later iterations, the test error separates after around 500 to 1000 iteration steps and converges to roughly 1.4%. The trained algorithm does no mistakes on the training set but still does about 130 mistakes out of 10000 predictions. Now looking at the average squared error one can see that the test error has a minimum around 2500 iterations and then slowly rises again whereas the training error still falls. To avoid overfitting and to get a good error estimation one should consider stopping training early, around the first 5000 iterations. Comment on choice of dev set: See previous part.

References

/

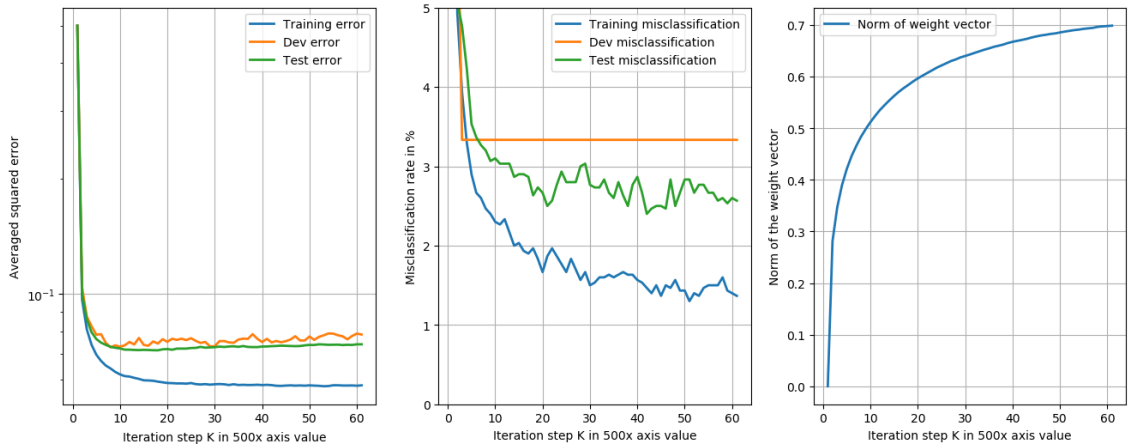

Figure 8: Case $k = 5000$. Plot of averaged squared errors (left), the norm of the weight vector (right) and misclassification loss in percent (middle). For generating the plots, $\eta = 10^{-3}$ and $\lambda = 10^{-2}$ were chosen.

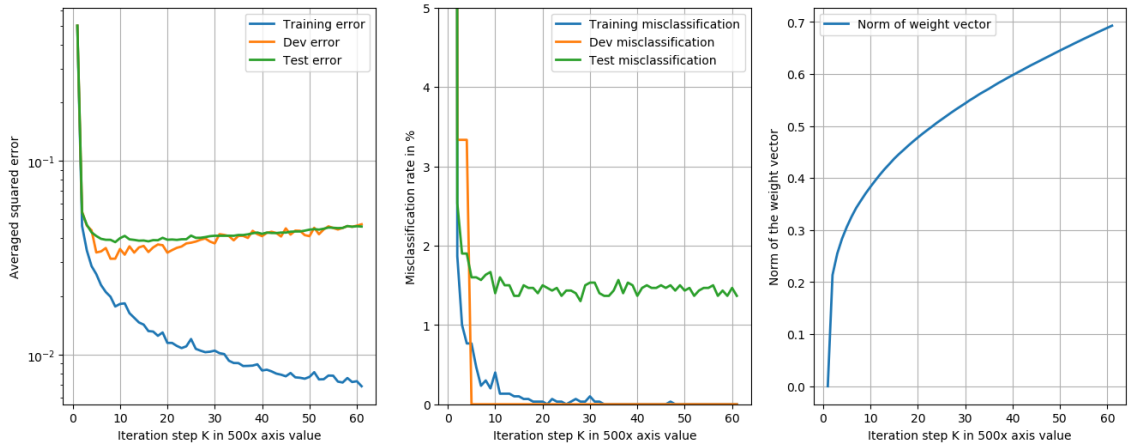

Figure 9: Case $k = 60000$. Plot of averaged squared errors (left), the norm of the weight vector (right) and misclassification loss in percent (middle). For generating the plots, $\eta = 10^{-3}$ and $\lambda = 10^{-2}$ were chosen.