CSE 546: Homework 2

Due on October 31, 2016

Brian de Silva

0 Collaborators and Acknowledgements

- I collaborated with Kathleen Champion on all of the problems and I collaborated with Matthew Farrell on problem 3.
- I referenced *Numerical Linear Algebra* by Trefethen and Bau to find the computational complexity of the QR-factorization.

1 Multi-Class Classification using Least Squares

1.1 One vs all classification

1. Recall that if X is the matrix of images, and y is the vector of (binary) labels, the least squares solution is given by

$$\hat{w} = (X^T X)^{-1} X^T y.$$

The approximation is then $\hat{y} = X\hat{w}$. First, computing $X^T X$ costs $\mathcal{O}(nd^2)$ operations (it has d^2 entries, each of which is the dot product of two length n vectors–these each require $2n-1$ floating point operations). Next, $X^T y$ has d entries, each of which is a dot product between two vectors in $\mathbb{R}^n$. Therefore this computation is $\mathcal{O}(nd)$. "Inverting" $X^T X$, or solving $X^T X z = X^y$ costs $\mathcal{O}(d^3)$ operations because $X^T X$ is a $d \times d$ matrix. Hence the total cost of finding the weight vector $\hat{w}$ for one linear regression problem is $\mathcal{O}(nd^2 + d^3 + nd) = \mathcal{O}(nd^2 + d^3)$. Since we expect that $n > d$, this is really $\mathcal{O}(nd^2)$. If we solved all k problems independently the total complexity would be $\mathcal{O}(knd^2)$.

2. The most expensive computations of the problem only need to be performed once. We saw that the cost of solving the linear regression problem is dominated by the computation and inversion of $X^T X$. Each of the k problems, if solved independently, requires these same steps. Rather than carrying out these operations separately, we can simply compute the (reduced) QR-factorization of X once, $X = QR$, at a cost of $\mathcal{O}(nd^2)$ floating point operations, then use it to find the weight vectors of each of the binary regression problems. For the i-th class, let y^i be the binary labels and let $\hat{w}^i$ be the weight vector we wish to find. Then $\hat{w}^i$ can be obtained as the solution to

$$R\hat{w}^i = Q^T y^i.$$

Since R is upper-triangular, solving this system costs only $\mathcal{O}(d^2)$ operations. The multiplication $Q^T y^i$ costs $\mathcal{O}(nd)$ operations. If we store Q and R and use them to solve each of the k problems, then our total cost is $\mathcal{O}(nd^2 + dkn + kd^2) = \mathcal{O}(nd^2 + dkn)$, which is much better than the $\mathcal{O}(knd^2)$ operations required to solve them each on their own.

If we wished to use ridge-regression, we could use the same regularization parameter λ for each problem and instead perform an LU-decomposition on $X^T X + \lambda I$, which we could then re-use for all the binary problems. This raises the cost by a constant times nd^2 since we must also compute $X^T X$ in this case.

3. Using the multiclass algorithm described above with a regularization parameter of $\lambda \approx 0.0835$ we obtain a 0/1 loss of 14.806667 percent incorrect (8884 / 60000 misclassifications) and a mean square loss of 0.391004 on the training set. On the test set our 0/1 loss was 14.660000 percent incorrect (1466 / 10000 misclassifications) and our mean square loss was 0.395191. The code for this problem can be found in `hw2-1.py`.

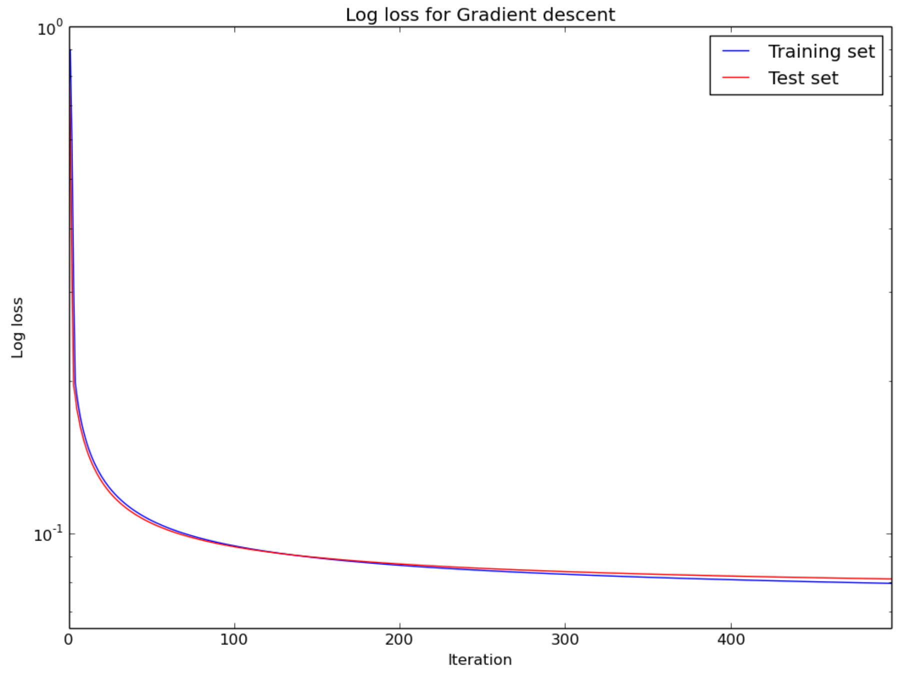

Figure 1: Log loss on training and test sets for binary logistic regression using batch gradient descent

1.2 Neural Nets with a random first layer: using more features

1. Using the same regularization parameter as above, $\lambda \approx 0.0835$, on our training set our 0/1 loss was 0.676667 percent incorrect (406 / 60000 misclassifications) and our mean square loss was 0.079238. On the test set our 0/1 loss was 2.330000 percent incorrect (233 / 10000 misclassifications) and our mean square loss was 0.122706. The code for this problem can be found in `hw2-1.py`.

2 Multi-Class Classification using Logistic Regression and Softmax

2.1 Binary Logistic Regression

1. After lots of experimenting I used a constant learning rate of 10^{-5}. My regularization constant was simply $\lambda = 1$ and I used an unregularized offset weight w_0.

2. Figure 1 shows the decay of the log loss on the training and test sets as a function of iteration of (batch) gradient descent. At first the log loss is lower on the test set than the training set, suggesting that the model is still underfitting the training data. Eventually the log loss on the training data drops below that of the test set.

3. Using a threshold of 0.5 for classification, my final losses on the training set are log loss: 0.079540 and 0/1 loss: 2.228333 percent incorrect (1337 / 60000 misclassifications). The final losses on the test set are log loss: 0.081172 and 0/1 loss: 2.060000 percent incorrect (206 / 10000 misclassifications). The code for this problem can be found in `hw2-2-1.py`.

2.2 Softmax classification: gradient descent

1. The negative log likelihood function consists of a sum of logs of two types of probabilities. We will distinguish between the two cases initially, then it will turn out that we can unify them in one compact formula. For $Y > 0$

$$\log(\Pr(Y=\ell|x,w)) = \log\left(\frac{\exp(w^{(\ell)}\cdot x)}{1+\sum_{i=1}^{k-1}\exp(w^{(i)}\cdot x)}\right) = w^{(\ell)}\cdot x - \log\left(1+\sum_{i=1}^{k-1}\exp(w^{(i)}\cdot x)\right).$$

Similarly, if $Y=0$, then

$$\log(\Pr(Y=0|x,w)) = -\log\left(1+\sum_{i=1}^{k-1}\exp(w^{(i)}\cdot x)\right).$$

Taking derivatives with respect to each of the weight vectors gives, for $Y>0$

$$\begin{aligned}\frac{\partial}{\partial w^{(j)}}\left(\log(\Pr(Y=\ell|x,w))\right) &= \frac{\partial}{\partial w^{(j)}}\left(w^{(\ell)}\cdot x - \log\left(1+\sum_{i=1}^{k-1}\exp(w^{(i)}\cdot x)\right)\right)\\ &= x\mathbf{1}(j=\ell) - \frac{x\exp(w^{(j)})}{1+\sum_{i=1}^{k-1}\exp(w^{(i)}\cdot x}\\ &= x\left(\mathbf{1}(j=\ell) - \frac{\exp(w^{(j)})}{1+\sum_{i=1}^{k-1}\exp(w^{(i)}\cdot x}\right)\\ &= x\left(\mathbf{1}(j=\ell) - P(Y=\ell|x,w)\right).\end{aligned}$$

For $Y=0$, we just lose the indicator function term and get

$$\frac{\partial}{\partial w^{(j)}}\left(\log(\Pr(Y=0|x,w))\right) = -xP(Y=j|x,w).$$

Combining both cases together and taking the full gradient (derivative with respect to $w=(w^{(1)},w^{(2)},\ldots,w^{(k-1)})^T$), we obtain the following

$$\frac{\partial}{\partial w}\left(\log(\Pr(Y=y^{(i)}|x^{(i)},w))\right) = \begin{bmatrix} x^{(i)}\left(\mathbf{1}(y^{(i)}=1)-\Pr(y^{(i)}=1|x^{(i)},w)\right)\\ x^{(i)}\left(\mathbf{1}(y^{(i)}=2)-\Pr(y^{(i)}=2|x^{(i)},w)\right)\\ \vdots\\ x^{(i)}\left(\mathbf{1}(y^{(i)}=k-1)-\Pr(y^{(i)}=k-1|x^{(i)},w)\right)\end{bmatrix}.$$

Hence the gradient of the negative log likelihood function is

$$\begin{aligned}\nabla_w L(w) &= -\frac{1}{N}\sum_{i=1}^{N}\frac{\partial}{\partial w}\log\Pr\left(Y=y^{(i)}|x^{(i)},w\right)\\ &= -\frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix} x^{(i)}\left(\mathbf{1}(y^{(i)}=1)-\Pr(y^{(i)}=1|x^{(i)},w)\right)\\ x^{(i)}\left(\mathbf{1}(y^{(i)}=2)-\Pr(y^{(i)}=2|x^{(i)},w)\right)\\ \vdots\\ x^{(i)}\left(\mathbf{1}(y^{(i)}=k-1)-\Pr(y^{(i)}=k-1|x^{(i)},w)\right)\end{bmatrix}.\end{aligned}$$

2. For my impelmentation of batch gradient descent see `hw2-2-2.py`. For this problem I used a constant learning rate of $10^{-4}/8$, no offset, an initial guess of all zero weights, and a regularization parameter $\lambda = 1$.

3. Figure 2 plots the log loss of my multiclass logistic regression algorithm on the training and test sets as a function of gradient descent iterations. Figure 3 plots the 0/1 loss as gradient descent chugs along.

4. On the training set my final log loss was 0.288508 and my final 0/1 loss was 7.9133 percent incorrect. On the test set my final log loss was 0.288230 and my final 0/1 loss was 7.8300 percent incorrect.

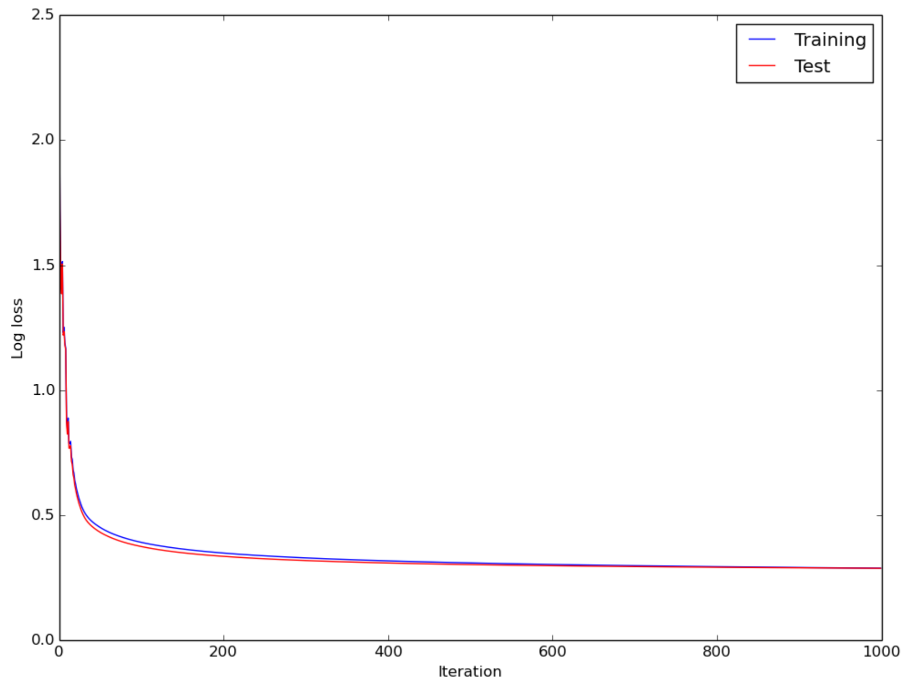

Figure 2: Log loss on training and test sets for multiclass logistic regression using batch gradient descent

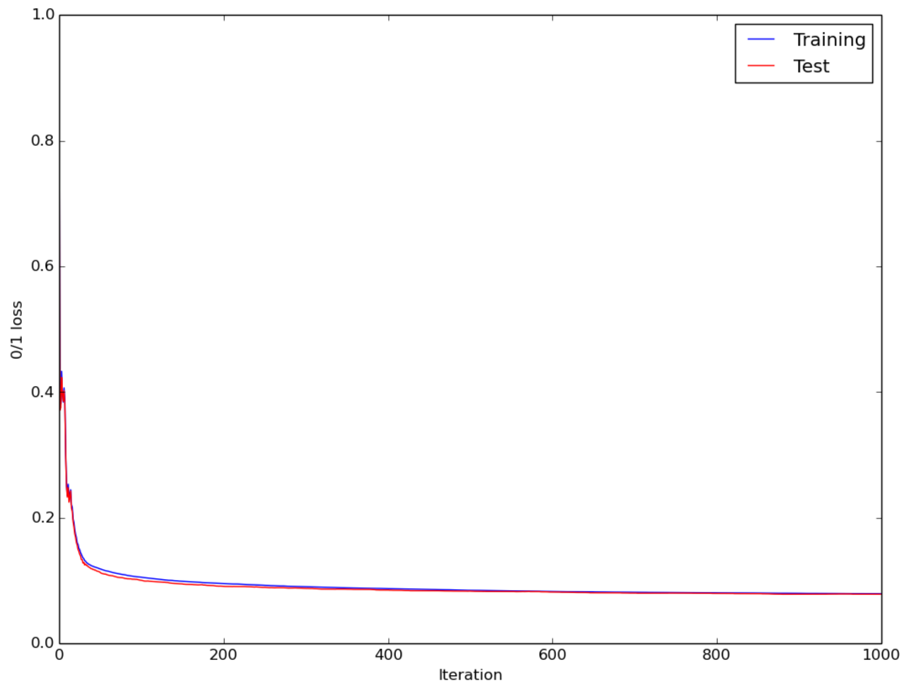

Figure 3: 0/1 loss on training and test sets for multiclass logistic regression using batch gradient descent

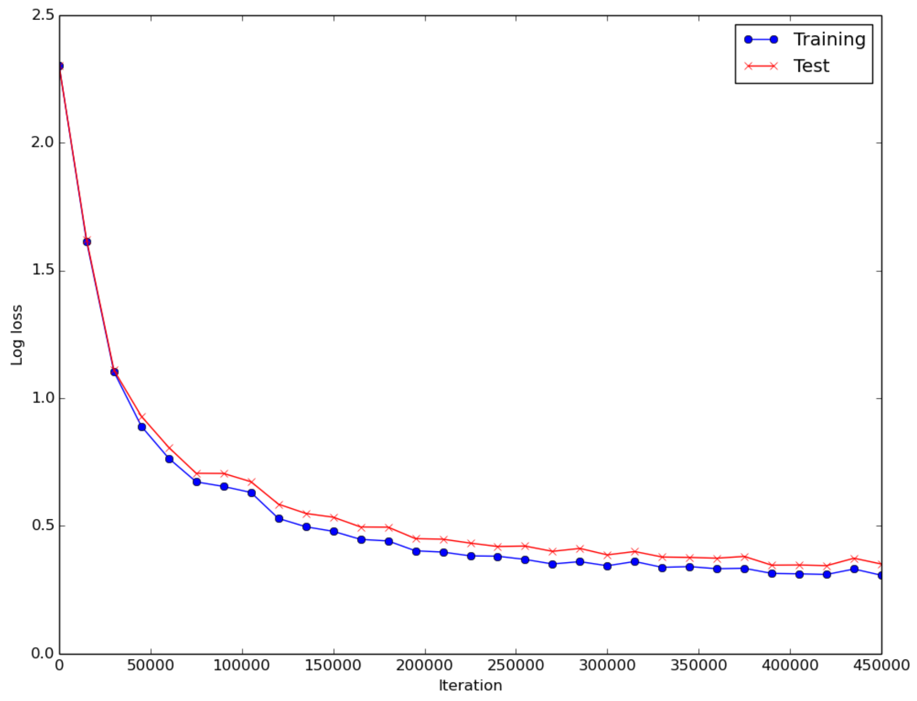

Figure 4: Log loss on training and test sets for multiclass logistic regression using stochastic gradient descent with batch size 1

2.3 Softmax classification: stochastic gradient descent

1. For this problem I used a decaying learning rate of $10^{-4}/\sqrt{t+1}$, where t is the iteration number. I used a regularization parameter of $\lambda = 1$, no offset, and an initial guess of all weights set to zero.

   Figures 4 and 5 show the log and 0/1 losses, respectively, on both the training and testing sets as functions of the numbers of iterations. These losses were sampled every time stochastic gradient descent had sampled an additional 15,000 points. In this case that corresponded to once every 15,000 iterations. After about 450,000 iterations, stochastic gradient descent (with batch size 1) attained similar performance to batch gradient descent, achieving a log loss of 0.306910 and a 0/1 loss of 7.9550 percent on the training set and a log loss of 0.350849 and 0/1 loss of 8.7600 percent on the training set, though these numbers vary a bit for any given run due to the random selection of the point at which to evaluate the gradient.

2. Batch gradient descent required $1,000$ iterations to produce the results given above. Each such iteration costs $n = 60,000$ times as much work as an iteration of stochastic gradient descent with batch size 1. In total, we can approximate the amount of work it used as being $1,000n = 1,000 \times 60,000 = 60,000,000$ times the work of a single iteration of stochastic gradient descent. Since stochastic gradient descent only needs around $450,000$ iterations to get similar performance, it is about $60,000,000/450,000 \approx 133$ times as efficient on this problem.

3. For this problem I used a decaying learning rate of $10^{-3}/(4\sqrt{t+1})$, where t is the iteration number. My regularization parameter was $\lambda = 1$. I did not use an offset and my initial guess was all zeros.

   Figures 6 and 7 show the log and 0/1 losses, respectively, on both the training and testing sets as

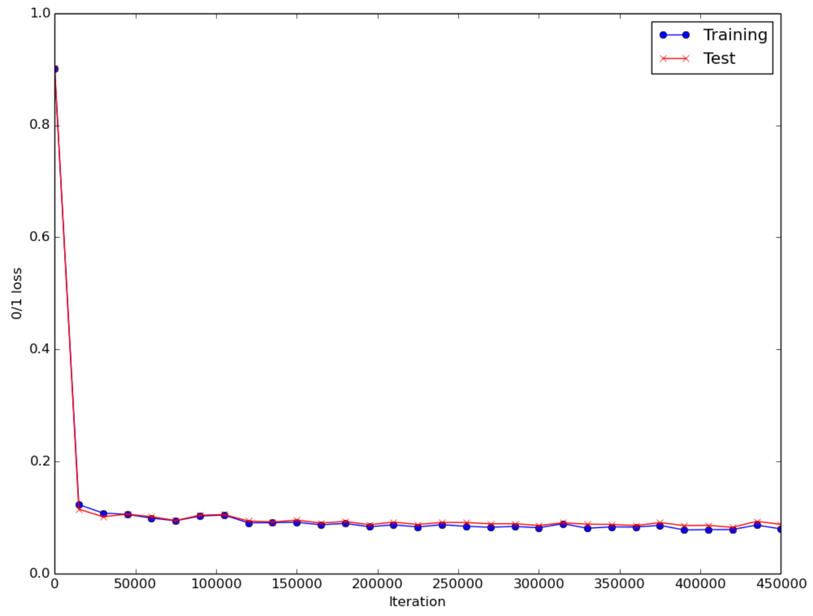

Figure 5: 0/1 loss on training and test sets for multiclass logistic regression using stochastic gradient descent with batch size 1

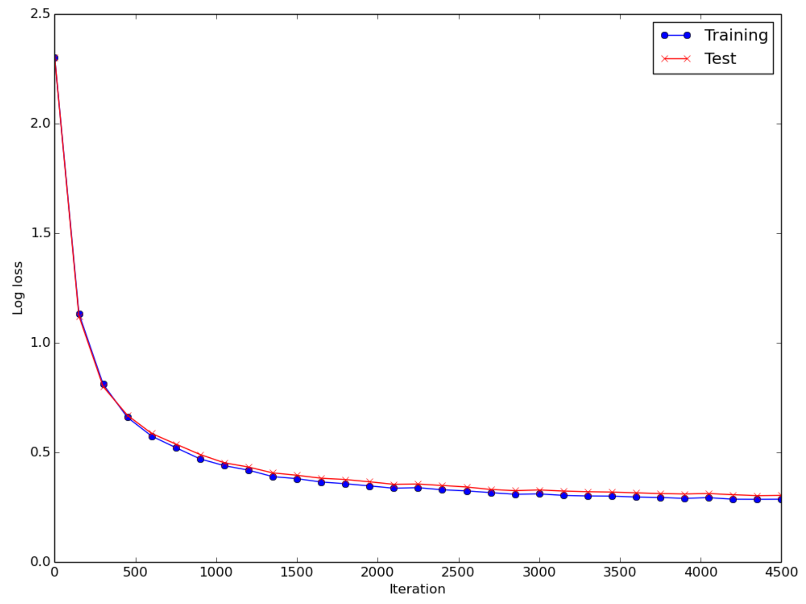

Figure 6: Log loss on training and test sets for multiclass logistic regression using stochastic gradient descent with batch size 100

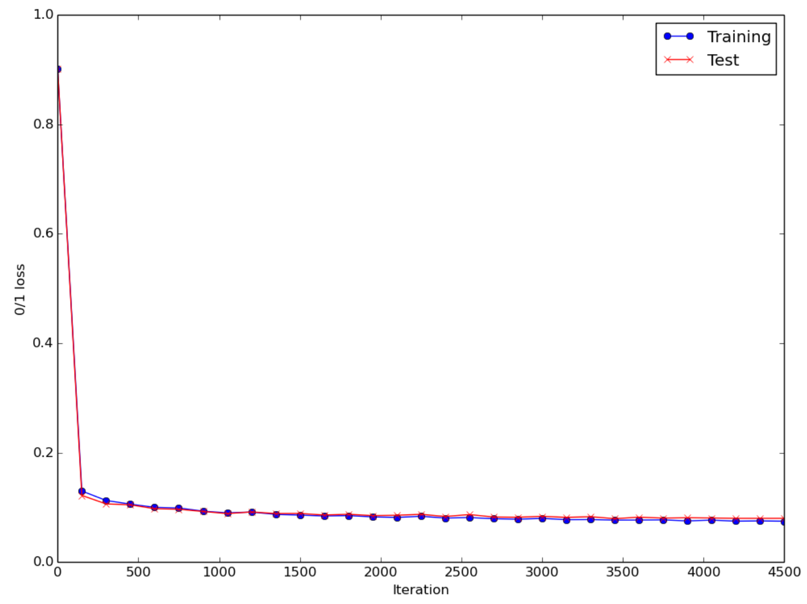

Figure 7: 0/1 loss on training and test sets for multiclass logistic regression using stochastic gradient descent with batch size 100

functions of the numbers of iterations. These losses were sampled every time stochastic gradient descent had sampled an additional 15,000 points. In this case that corresponded to once every 150 iterations. We have plotted out to 4,500 iterations, though the method achieved similar results (with respect to 0/1 loss) to batch gradient descent after only about 3,000 iterations. At this point the log loss on the training set was 0.292830 and the 0/1 loss was 7.6367 percent. On the training set the log loss was 0.307059 and the 0/1 loss was 8.1100 percent.

4. Recall that stochastic gradient descent using only 1 points needed 450,000 iterations to get similar results to stochastic gradient descent with batch size 100. Assuming using stochastic gradient descent with batch size 100 costs 100 times more than using batch size 1, $3,000$ iterations of SGD with batches of 100 are roughly equivalent in cost to $3,000 \times 100 = 300,000$ iterations of SGD with batches of 1. So using batches of 100 is roughly 1.5 times as efficient as using batches of size 1 on this problem!

   The code for this problem can be found in `hw2-2-3.py`.

2.4 Neural Nets with a random first layer: Using more features

1. I chose to use stochastic gradient descent with batch size 100. My learning rate was $10^{-1}/\sqrt{t+1}$, where t is the iteration number. I used a regularization constant $\lambda = 1$, no offset, and an initial guess of all zeros.

   Figure 8 and 9 show the log and 0/1 loss for multiclass logistic regression using stochastic gradient descent as a function of iteration count. The log and 0/1 losses were computed once every 150 iterations. After 30,000 iterations the log loss on the training set was 0.065144 and the 0/1 loss was 1.4817 percent. On the test set the log loss was 0.162889 and the 0/1 loss was 0.033900 percent. Using the same parameters and more iterations we were also able to achieve results slightly better than those in section 1.2, i.e. a 0/1 loss of a little over 0.6 percent on the training set and a bit over 2 percent on the

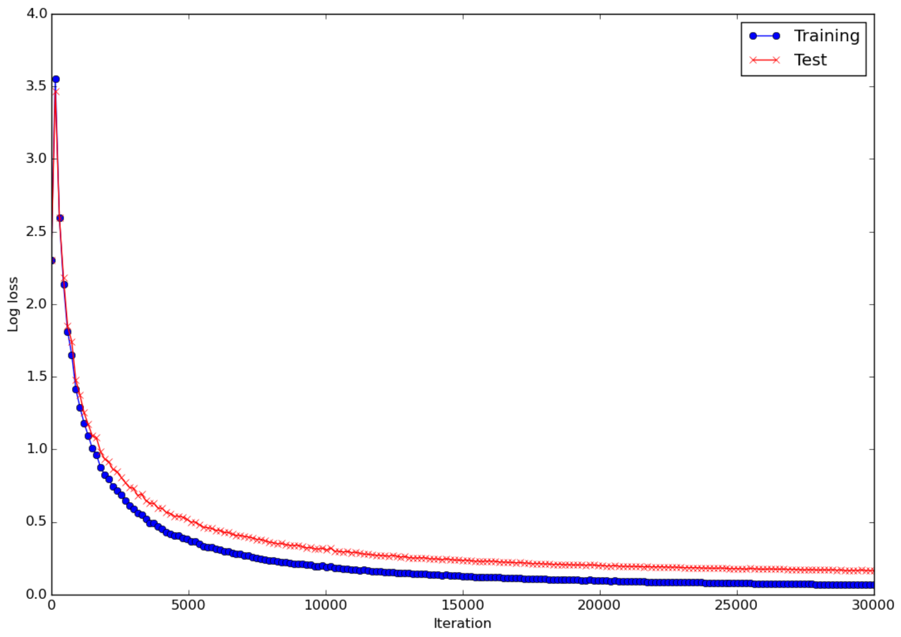

Figure 8: Log loss on training and test sets (with more features) for multiclass logistic regression using stochastic gradient descent with batch size 100

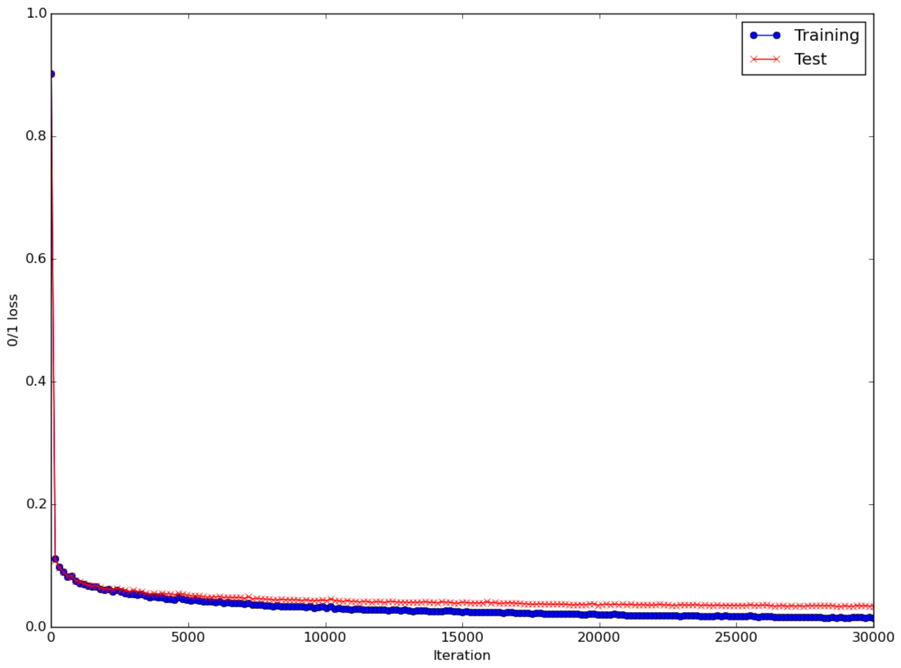

Figure 9: 0/1 loss on training and test sets (with more features) for multiclass logistic regression using stochastic gradient descent with batch size 100

test set. However, we mistakenly neglected to generate plots in this case. The code for this problem can be found in `hw2-2-4.py`.

3 (Baby) Learning Theory: Why is statistical learning possible?

3.1 A confidence interval for a coin

1. Fix $\delta > 0$. Note that if, for some $\epsilon > 0$, $\Pr(|\bar{Z} - \theta| \geq \epsilon) \leq 2e^{-2N\epsilon^2}$, then $\Pr(|\bar{Z} - \theta| < \epsilon) < 1 - 2e^{-2N\epsilon^2}$. So if we take $\delta = 2e^{-2N\epsilon^2}$, we have our desired bound. To this end, we solve for δ in terms of ϵ

$$\begin{aligned} \delta &= 2e^{-2N\epsilon^2} \\ \implies \log\left(\frac{\delta}{2}\right) &= -2N\epsilon^2 \\ \implies \frac{-1}{2N}\log\left(\frac{\delta}{2}\right) &= \epsilon^2 \\ \implies \sqrt{\frac{1}{2N}\log(2/\delta)} &= \epsilon. \end{aligned}$$

Hence, the probability that $|\bar{Z} - \theta| < \sqrt{\frac{1}{2N}\log(2/\delta)}$ is greater than $1 - \delta$.

3.2 Estimating the performance of a given classifier

1. Let us use the following estimator of $L(f)$

$$\hat{L}(f) := \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2.$$

Notice that since f maps from $\mathcal{X}$ to $\{0,1\}$, and $y_i \in \{0,1\}$, each of the terms in the sum is either 0 or 1. $\hat{L}(f)$ simply gives the probability that f misclassifies a point in our set of samples. We can relate this situation with the previous problem as follows: let $Z = (Y - f(X))^2$ be a random variable, taking values in $\{0,1\}$ and denote $Z_i = (y_i - f(x_i))^2$. Observe that Z is a Bernoulli(θ) random variable, i.e. $\Pr(Z = 1) = \theta$ and $\Pr(Z = 0) = 1 - \theta$, for some parameter $\theta \in [0,1]$. Furthermore,

$$L(f) = E_{X,Y}(\mathbf{1}(f(X) \neq Y)) = E_Z(Z) = \theta,$$

and

$$\hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} Z_i = \bar{Z}.$$

Thus, by the previous result, since the samples are i.i.d and f^* is chosen independently of them, the Z_i are i.i.d, and we have

$$\Pr\left(|L(f) - \hat{L}(f)| < \sqrt{\tfrac{1}{2N}\log(2/\delta)}\right) = \Pr\left(|\theta - \bar{Z}| < \sqrt{\tfrac{1}{2N}\log(2/\delta)}\right) > 1 - \delta.$$

3.3 ERM revisited

1. The confidence interval above does hold when applied to f^*. If we take f^* to be the "given" classifier, f, in the previous problem, then all the arguments used there still apply. It is important to notice that f^* is given to separately from the samples upon which we estimate the loss, $\{(x_1, y_1), \ldots, (x_n, y_n)\}$.

2. The confidence interval from the last question does not hold with probability greater than $1-\delta$ for $\hat{f}$. This is because the classifier, $\hat{f}$, we choose depends implicitly on the set of samples $\{(x_1,y_1),\dots,(x_n,y_n)\}$ that are used for training. If we had been given different samples, we may have chosen a different function in $\mathcal{F}$ as our classifier. In the previous part we needed $Z_1,Z_2,\dots,Z_n$ to be i.i.d. However, in this instance they fail to be independent. $\hat{f}$ is chosen so that the sum $\sum_{i=1}^n Z_i(f)$ is minimal, where $Z_i(f)=(f(x_i)-y_i)^2$. Whether or not a particular $Z_i(f)$ is 0 or not for a given f will affect whether or not that f is chosen to be $\hat{f}$, thereby affecting the other $Z_j(f)$ (if a different function is chosen, they can change). Since this i.i.d. assumption is not met, the confidence interval is not valid for $\hat{f}$.

3. From part 3.2, we know that for a given $f\in\mathcal{F}$,
$$\Pr\left(|\hat{L}(f)-L(f)|>B\right)\le 2e^{-2NB^2}.$$
If $\mathcal{F}=\{f_1,f_2,\dots,f_k\}$, then
$$\begin{aligned}&\Pr\left(|\hat{L}(f_1)-L(f_1)|>B \text{ or } |\hat{L}(f_2)-L(f_2)|>B \text{ or}\dots\text{or } |\hat{L}(f_k)-L(f_k)|>B\right)\\ &\le\sum_{i=1}^k\Pr\left(|\hat{L}(f_i)-L(f_i)|>B\right)\\ &\le 2ke^{-2NB^2}.\end{aligned}$$
Hence,
$$\begin{aligned}&\Pr\left(\forall f\in\mathcal{F}, |\hat{L}(f)-L(f)|\le B|\right)\\ &=1-\Pr\left(|\hat{L}(f_1)-L(f_1)|>B \text{ or } |\hat{L}(f_2)-L(f_2)|>B \text{ or}\dots\text{or } |\hat{L}(f_k)-L(f_k)|>B\right)\\ &\ge 1-2ke^{-2NB^2}.\end{aligned}$$
So if we choose B so that $\delta=2ke^{-2NB^2}$, then we get our desired confidence interval. I.e., for fixed $\delta>0$, solving for B in terms of δ in the above, we see that if we choose
$$B=\sqrt{\frac{1}{2N}\log(2k/\delta)},$$
then
$$\Pr\left(\forall f\in\mathcal{F}, |\hat{L}(f)-L(f)|\le B\right)\ge 1-\delta.$$
Notice that the larger k becomes, the worse our confidence interval becomes.

4. Fix $\delta>0$. Since $\hat{f}\in\mathcal{F}$ the bound derived in the previous part certainly holds for it. That is to say, if we choose
$$B=\sqrt{\frac{1}{2N}\log(2k/\delta)}$$
as above, then since the event that $\forall f\in\mathcal{F}, |\hat{L}(f)-L(f)|\le B$ is a subset of the event $|\hat{L}(\hat{f})-L(\hat{f})|\le B$, we have
$$\Pr\left(|\hat{L}(\hat{f})-L(\hat{f})|\le B\right)\ge\Pr\left(\forall f\in\mathcal{F}, |\hat{L}(f)-L(f)|\le B\right)\ge 1-\delta.$$

5. First observe that since $\hat{f}$ minimizes $\hat{L}(f)$, $\hat{L}(\hat{f})-\hat{L}(f^*)\le 0$. Using this we have
$$\begin{aligned}L(\hat{f})-L(f^*)&=L(\hat{f})-\hat{L}(\hat{f})+\hat{L}(\hat{f})-\hat{L}(f^*)+\hat{L}(f^*)-L(f^*)\\ &=\left(L(\hat{f})-\hat{L}(\hat{f})\right)+\left(\hat{L}(\hat{f})-\hat{L}(f^*)\right)+\left(\hat{L}(f^*)-L(f^*)\right)\\ &\le\left(L(\hat{f})-\hat{L}(\hat{f})\right)+\left(\hat{L}(f^*)-L(f^*)\right)\\ &\le\left|L(\hat{f})-\hat{L}(\hat{f})\right|+\left|\hat{L}(f^*)-L(f^*)\right|.\end{aligned}$$

Next, notice that since $\hat{f}, f^* \in \mathcal{F}$, we can use the confidence interval from part 3 to bound both $\left|L(\hat{f}) - \hat{L}(\hat{f})\right|$ and $\left|\hat{L}(f^*) - L(f^*)\right|$ simultaneously with probability greater than $1-\delta$. Specifically, we saw that if we take

$$B = \sqrt{\frac{1}{2N}\log(2k/\delta)}$$

then

$$\Pr\left(\forall f \in \mathcal{F}, |\hat{L}(f) - L(f)| \leq B\right) \geq 1-\delta.$$

Using this, we have

$$\begin{aligned}
&\Pr\left(L(\hat{f}) - L(f^*) \leq \left|L(\hat{f}) - \hat{L}(\hat{f})\right| + \left|\hat{L}(f^*) - L(f^*)\right| \leq 2B\right) \\
&\geq \Pr\left(\forall f \in \mathcal{F}, |\hat{L}(f) - L(f)| \leq B\right) \\
&\geq 1-\delta,
\end{aligned}$$

since the event $\left\{\left|L(\hat{f}) - \hat{L}(\hat{f})\right| \leq B \text{ and } \left|\hat{L}(f^*) - L(f^*)\right| \leq B\right\}$ is a subset of the event $\forall f \in \mathcal{F}, |\hat{L}(f) - L(f)| \leq B$

6. Let $\delta > 0$ be fixed.

   (a) If K is constant then we can learn since our upper bound on our regret is then $R = 2\sqrt{\frac{1}{2N}\log(2K/\delta)} = \mathcal{O}(N^{-1/2})$, which tends to 0 as $N \to \infty$.

   (b) If $K = N^p$, for some constant p, then learning is still possible since

   $$R = 2\sqrt{\frac{1}{2N}\log(2N^p/\delta)} \;=\; 2\sqrt{\frac{1}{2N}\left(p\log(N) + \log(2/\delta)\right)} \;=\; \mathcal{O}\left(\sqrt{\frac{\log(N)}{N}}\right)$$

   which also tends to 0 as $N \to \infty$.

   (c) If we are even more greedy and $K = e^{\sqrt{N}}$ then we can still learn. To see this, we again examine the regret bound:

   $$\begin{aligned}
   R &= 2\sqrt{\frac{1}{2N}\log(2e^{\sqrt{N}}/\delta)} \\
   &= 2\sqrt{\frac{1}{2N}\left(\sqrt{N} + \log(2/\delta)\right)} \\
   &= \mathcal{O}\left(\sqrt{\frac{\sqrt{N}}{N}}\right) \\
   &= \mathcal{O}\left(N^{-1/4}\right).
   \end{aligned}$$

   Hence the regret bound tends to 0 as $N \to \infty$.

   (d) If we choose $K = e^{10N}$, then we are not able to learn since our regret bound is

   $$\begin{aligned}
   R &= 2\sqrt{\frac{1}{2N}\log(2e^{10N}/\delta)} \\
   &= 2\sqrt{\frac{1}{2N}(\log(2) + 10N - \log(\delta))} \\
   &= 2\sqrt{5 + \frac{1}{2N}(\log(2/\delta))}.
   \end{aligned}$$

   This term does not vanish as $N \to \infty$, so no matter how many samples we take, we cannot expect to be able to approach the best possible loss in our hypothesis space.