ML Homework 1

Samuel Pegg

1 Mathematics Basics

1.1 Optimisation

Use the Lagrange multiplier method to solve the following problem:

$$\begin{aligned} &\min_{x_1,x_2} x_1^2 + x_2^2 - 1 \\ \text{s.t.} \quad & x_1 + x_2 - 1 = 0 \\ & x_1 - 2x_2 \geq 0 \end{aligned} \tag{1}$$

Let

$$\mathcal{L}(\boldsymbol{x}, \lambda, \mu) = x_1^2 + x_2^2 - 1 + \lambda(x_1 + x_2 - 1) + \mu(2x_2 - x_1)$$

where the Lagrange multiplier $\mu \geq 0$. Next we minimise with respect to x:

$$\nabla_{\boldsymbol{x}} \mathcal{L} = \mathbf{0} \qquad \iff \qquad \begin{pmatrix} 2x_1 + \lambda - \mu \\ 2x_2 + \lambda + 2\mu \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \implies \begin{cases} x_1 = \frac{\mu - \lambda}{2} \\ x_2 = -\frac{\lambda + 2\mu}{2} \end{cases}$$

$$\implies \mathcal{L}(\lambda, \mu) = \left(\frac{\mu - \lambda}{2}\right)^2 + \left(\frac{\lambda + 2\mu}{2}\right)^2 - 1 + \lambda\left(\left(\frac{\mu - \lambda}{2}\right) - \left(\frac{\lambda + 2\mu}{2}\right) - 1\right) - \mu\left(2\left(\frac{\lambda + 2\mu}{2}\right) + \left(\frac{\mu - \lambda}{2}\right)\right)$$

Thus we have eliminated $\boldsymbol{x}$.

$$\implies \mathcal{L}(\lambda, \mu) = -\frac{1}{4}\left(5\mu^2 + 2\lambda\mu + 2\lambda^2 + 4\lambda + 4\right)$$

Note that complementary slackness requires $0 = \mu(2x_2 - x_1) = -\frac{\mu}{2}(\lambda + 5\mu) \implies$ either $\mu = 0$ or $\lambda = -5\mu$. We can now minimise with respect to the Lagrange multipliers.

$$\frac{\partial}{\partial \mu}\mathcal{L}(\lambda, \mu) = -\frac{1}{4}(10\mu + 2\lambda) = 0 \qquad \implies \lambda = -5\mu$$

$$\frac{\partial}{\partial \lambda}\mathcal{L}(\lambda, \mu) = -\frac{1}{4}(4\lambda + 2\mu + 4) = 0 \qquad \implies \mu = -2 - 2\lambda$$

$$\implies \mu = -2 + 10\mu \implies \mu = \frac{2}{9} \implies \lambda = -\frac{10}{9}$$

$$\implies x_1 = \frac{2}{3}, \qquad x_2 = \frac{1}{3}$$

This gives a result of $x_1^2 + x_2^2 - 1 = -\frac{4}{9}$.

1.2 Stochastic Process

Compute the expected number of tosses we need to observe a first time occurrence of the following consecutive pattern

$$H, T, T, \cdots, T \qquad \text{where } T \text{ is repeated } k \text{ times}$$

To obtain the expected number of a pattern, you take the inverse of the probability of the pattern, plus the inverse of the probability of any patterns that repeat at the beginning and end (source). Since there are no patterns that repeat at the beginning and end, the answer is simply 2^{k+1}.

2 SVM

Let $\epsilon > 0$ be a fixed small value. For training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, derive the dual of the following SVM:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\hat{\boldsymbol{\xi}}} \frac{1}{2}\|\boldsymbol{w}\|^2 + &C\sum_{i=1}^N(\xi_i + \hat{\xi}_i) \\ \text{s.t. } \forall i = 1,...,N: \qquad y_i \leq &\boldsymbol{w}^T\boldsymbol{x}_i + b + \epsilon + \xi_i \\ y_i \geq &\boldsymbol{w}^T\boldsymbol{x}_i + b - \epsilon - \hat{\xi}_i \\ \xi_i \geq &0 \\ \hat{\xi}_i \geq &0 \end{aligned}$$

Firstly we write down the Lagrangian:

$$\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\xi},\hat{\boldsymbol{\xi}},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^N(\xi_i+\hat{\xi}_i) + \sum_{i=1}^N\alpha_i(y_i - \boldsymbol{w}^T\boldsymbol{x}_i - b - \epsilon - \xi_i) + \sum_{i=1}^N\beta_i(\boldsymbol{w}^T\boldsymbol{x}_i + b - \epsilon - \hat{\xi}_i - y_i) - \sum_{i=1}^N\gamma_i\xi_i - \sum_{i=1}^N\delta_i\hat{\xi}_i$$

$$= \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^N\Big((C-\gamma_i-\alpha_i)\xi_i + (C-\delta_i-\beta_i)\hat{\xi}_i\Big) + \sum_{i=1}^N(\alpha_i-\beta_i)(y_i - \boldsymbol{w}^T\boldsymbol{x}_i - b) - \sum_{i=1}^N\epsilon(\alpha_i+\beta_i)$$

where the Lagrange multipliers satisfy $\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta} \geq \boldsymbol{0}$. Next we minimise with respect to the original variables:

$$\begin{aligned} \nabla_{\boldsymbol{w}}\mathcal{L} = \boldsymbol{w} - \sum_{i=1}^N\alpha_i\boldsymbol{x}_i + \sum_{i=1}^N\beta_i\boldsymbol{x}_i = 0 \qquad &\implies \qquad \boldsymbol{w} = \sum_{i=1}^N(\alpha_i-\beta_i)\boldsymbol{x}_i \\ \nabla_b\mathcal{L} = \sum_{i=1}^N\beta_i - \sum_{i=1}^N\alpha_i = 0 \qquad &\implies \qquad \sum_{i=1}^N\beta_i = \sum_{i=1}^N\alpha_i \\ \nabla_{\boldsymbol{\xi}}\mathcal{L} = C\boldsymbol{1} - \boldsymbol{\alpha} - \boldsymbol{\gamma} = 0 \qquad &\implies \qquad \alpha_i + \gamma_i = C \quad \forall i = 1,...,N \\ \nabla_{\hat{\boldsymbol{\xi}}}\mathcal{L} = C\boldsymbol{1} - \boldsymbol{\beta} - \boldsymbol{\delta} = 0 \qquad &\implies \qquad \beta_i + \delta_i = C \quad \forall i = 1,...,N \end{aligned}$$

Due to the non-negativity constraints on $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$, the last two lines can be rewritten

$$\begin{aligned} \gamma_i = C - \alpha_i \geq 0, \quad 0 \leq \alpha_i \leq C \qquad \forall i = 1,...,N \\ \delta_i = C - \beta_i \geq 0, \quad 0 \leq \beta_i \leq C \qquad \forall i = 1,...,N \end{aligned}$$

Substituting this information back into the Lagrangian, we obtain

$$\mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) = \frac{1}{2}\sum_{i,j=1}^N(\alpha_i-\beta_i)(\alpha_j-\beta_j)\boldsymbol{x}_i^T\boldsymbol{x_j} + \sum_{i=1}^N 0\times(\xi_i+\hat{\xi}_i) + \sum_{i=1}^N y_i(\alpha_i-\beta_i) - \sum_{i=1}^N(\alpha_i-\beta_i)\sum_{j=1}^N(\alpha_j-\beta_j)\boldsymbol{x}_j^T\boldsymbol{x}_i - 0\times b - \sum_{i=1}^N\epsilon(\alpha_i+\beta_i)$$

$$\mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_{i=1}^N y_i(\alpha_i-\beta_i) - \frac{1}{2}\sum_{i,j=1}^N(\alpha_i-\beta_i)(\alpha_j-\beta_j)\boldsymbol{x}_i^T\boldsymbol{x_j} - \sum_{i=1}^N\epsilon(\alpha_i+\beta_i)$$

which can be rewritten

$$\mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) = (\boldsymbol{\alpha}-\boldsymbol{\beta})^T\boldsymbol{y} - \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\beta})^TG(\boldsymbol{\alpha}-\boldsymbol{\beta}) - (\boldsymbol{\alpha}+\boldsymbol{\beta})^T\boldsymbol{\epsilon}$$

where $\boldsymbol{y} = (y_1,...,y_N)$, $G_{ij} = \boldsymbol{x}_i^T\boldsymbol{x}_j$ is the Gram Matrix, and $\boldsymbol{\epsilon} = (\epsilon,...,\epsilon)$. Thus the dual of the linear SVM given is

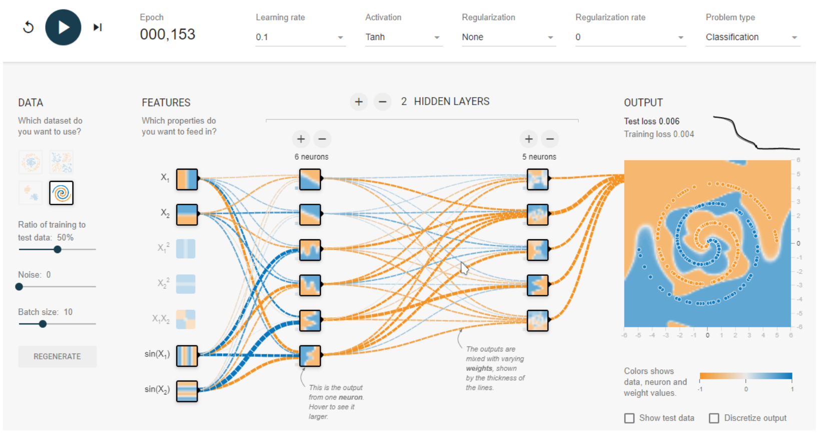
$$\begin{aligned} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \quad (\boldsymbol{\alpha}-\boldsymbol{\beta})^T\boldsymbol{y} - \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\beta})^TG(\boldsymbol{\alpha}-\boldsymbol{\beta}) - (\boldsymbol{\alpha}+\boldsymbol{\beta})^T\boldsymbol{\epsilon}& \\ \text{s.t.} \qquad 0 \leq \alpha_i,\beta_i \leq &C \quad \forall i = 1,...,N \\ \sum_{i=1}^N(\alpha_i-\beta_i) = &0 \end{aligned}$$

Complimentary slackness gives us

$$\begin{aligned} \alpha_i(y_i - \boldsymbol{w}^T\boldsymbol{x}_i - b - \epsilon - \xi_i) = &0 \quad \forall i = 1,...,N \\ \beta_i(\boldsymbol{w}^T\boldsymbol{x}_i + b - \epsilon - \hat{\xi}_i - y_i) = &0 \quad \forall i = 1,...,N \\ (C-\alpha_i)\xi_i = &0 \quad \forall i = 1,...,N \\ (C-\beta_i)\hat{\xi}_i = &0 \quad \forall i = 1,...,N \end{aligned}$$

3 Deep Neural Networks: Have a Try

3.1



Epoch 000,153

Learning rate 0.1

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

2 HIDDEN LAYERS

6 neurons

5 neurons

Test loss 0.006

Training loss 0.004

Which dataset do you want to use?

Which properties do you want to feed in?

This is the output from one ***neuron***. Hover to see it larger.

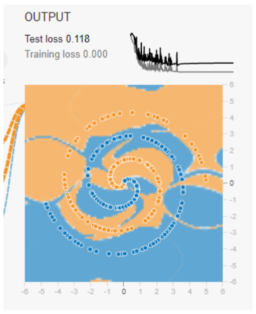
The outputs are mixed with varying ***weights***, shown by the thickness of the lines.

Colors shows data, neuron and weight values.

3.2

With everything set as in the above image, I changed each hyper parameter and recorded the effects.

1.
   - Decreasing the learning rate resulted in a much longer convergence time. However, the final performance was the same, and the convergence becomes more smooth - test loss monotonic decreasing. Decrease the learning rate too much actually stops the network converging (or it just takes so long it's not worth considering). This happened at 0.003.
   - Increasing the learning rate made the convergence much faster. However, the convergence was also less smooth - test loss increased and decreased in "zig-zag" like patterns. However, final performance was the same. Increasing the learning rate further, the zig-zags in test loss became so large the network sometimes stopped converging. Even when it did converge, performance was often lower. When the learning rate was 3, it stopped converging altogether.
2.
   - Changing the activation function to ReLU increased convergence time and decreased performance. The spiral shapes didn't seem to want to form, and were very jagged compared to the figure above. In addition, during convergence the test loss oscillated a lot.
   - Changing the activation function to Sigmoid did not change the performance of the final network, but convergence was much slower.
   - Changing the activation function to linear meant the network no longer converged.
3.
   - Increasing the number of layers resulted in the convergence being less smooth. In addition, the test loss oscillates with very small amplitude for a very long time for finally becoming stable. The final performance was slightly worse too. Increasing the number of layers further results in even slower convergence, with huge oscillations. There was also a lot of evidence for over-fitting. Consider the figure below:



Test loss 0.118

Training loss 0.000

Notice the circular blue lines to the left hand side, and the orange arc on the right. These are not patterns present in the original data. In addition, convergence was quite random. There were occasions where it converged very quickly, and other times it got stuck for ages before converging. Over-fitting meant the training loss reached 0, but the test lost plateaued at 0.1, which is a big decrease in performance. If you try several times, you randomly get a perfect algorithm, but more often than not it over-fits and attains the mediocre result seen in the image above.

- Decreasing the number of hidden layers made the network take longer to converge, and the final performance was often worse. Continuing to remove nodes and layers, the network wasn't complex enough to fit the data, and had very poor performance. This happened at around 1 layer, 5 nodes.

4.
- Under L1 regularisation, increasing the regularisation rate slows convergence and reduces performance. Often, when the network looked like it was about to converge, instead it just oscillated continuously at a low test loss. A regularisation rate of 0.01 or higher stopped convergence completely.
- L2 regularisation had similar results to L1, but less drastic. Convergence stopped at a regularisation rate of 0.03, and rates below this value didn't have as poor performance as with L1. There was also less oscillation in test loss.

4 IRLS for Logistic Regression

For a binary classification problem $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ ($\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \{0,1\}$), the probabilistic decision rule according to "logistic regression" is

$$P_{\boldsymbol{w}}(y|\boldsymbol{x}) = \frac{\exp(y\boldsymbol{w}^T\boldsymbol{x})}{1+\exp(\boldsymbol{w}^T\boldsymbol{x})} \tag{2}$$

And hence the log-likelihood is

$$\mathcal{L}(w) = \log\left(\Pi_{i=1}^N P_{\boldsymbol{w}}(y_i|\boldsymbol{x}_i)\right) \tag{3}$$

$$= \sum_{i=1}^N y_i \boldsymbol{w}^T\boldsymbol{x}_i - \log(1+\exp(\boldsymbol{w}^T\boldsymbol{x}_i)) \tag{4}$$

Please implement the IRLS algorithm to estimate the parameters of logistic regression

$$\max_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) \tag{5}$$

and the L2-norm regularised logistic regression

$$\max_{\boldsymbol{w}} \phi(\boldsymbol{w}) \quad \text{where} \quad \phi(\boldsymbol{w}) = -\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \mathcal{L}(\boldsymbol{w}) \tag{6}$$

where λ is the positive regularisation constant. From the lecture notes, we get the update equation for (5) as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - H^{-1}\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t) = (X^TR_tX)^{-1}X^TR_t(X\boldsymbol{w}_t + R_t^{-1}(\boldsymbol{y}-\boldsymbol{\mu}_t))$$

Note that I have used the standard design matrix, rather than the ones from lectures (so $X_{lectures} = X^T$). For the update equation for (6), we firstly need $\nabla_{\boldsymbol{w}}\phi(\boldsymbol{w}_t)$.

$$\nabla_{\boldsymbol{w}}\phi(\boldsymbol{w}_t) = X^T(\boldsymbol{y}-\boldsymbol{\mu}_t) - \lambda\boldsymbol{w}_t$$

and secondly we need the Hessian:

$$H = -X^TR_tX - \lambda I$$

Substituting these to into the update equation we obtain

$$\begin{aligned}\boldsymbol{w}_{t+1} =& \boldsymbol{w}_t + (X^TR_tX+\lambda I)^{-1}\left(X^T(\boldsymbol{y}-\boldsymbol{\mu}_t) - \lambda\boldsymbol{w}_t\right)\\ =& (X^TR_tX+\lambda I)^{-1}\left((X^TR_tX+\lambda I)\boldsymbol{w}_t + X^T(\boldsymbol{y}-\boldsymbol{\mu}_t) - \lambda\boldsymbol{w}_t\right)\\ =& (X^TR_tX+\lambda I)^{-1}\left(X^TR_tX\boldsymbol{w}_t + X^T(\boldsymbol{y}-\boldsymbol{\mu}_t)\right)\\ =& (X^TR_tX+\lambda I)^{-1}X^TR_t\left(X\boldsymbol{w}_t + R_t^{-1}(\boldsymbol{y}-\boldsymbol{\mu}_t)\right)\end{aligned}$$

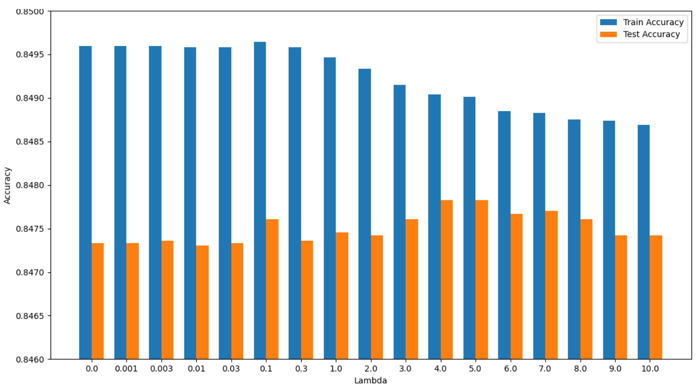
where the differences between this update equation and the non-regularised version have been highlighted in red. I implemented this code in python, and iterated until convergence. I defined convergence as the point in which the training accuracy of the previous iteration is no different to the training accuracy of the current iteration. We achieve the following results.

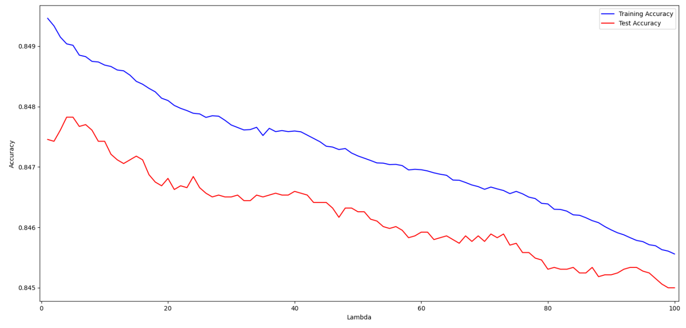
Comments	λ	Train Acc	Test Acc.	L2 Norm	Log-Likelihood	Convergence
Non-Regularised	0	0.849145	0.849948	13.137408	-10504.884970	7
Cross-Validation Suggested λ	4	0.848838	0.849764	5.175971	-10575.729121	7
Highest Test Accuracy	54	0.847241	0.851668	3.649683	-11003.807244	6
Fastest Convergence	184	0.843924	0.847184	2.710754	-11656.975607	3

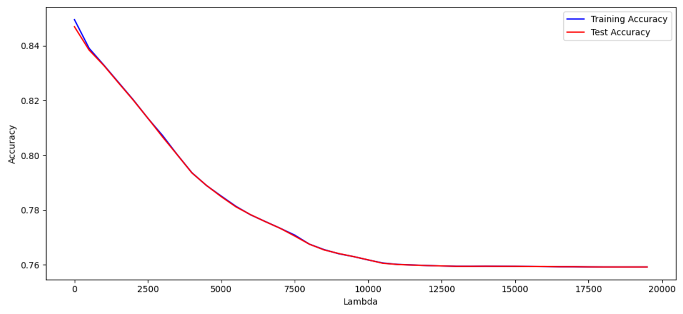
The standard IRLS achived a test accuracy of 0.849948. For the L2-regularised version, using 10-fold cross validation on the training data, the top five "best" integer values for lambda were

$$\begin{bmatrix} \text{Accuracy} & \lambda \\ 0.84782424 & 4 \\ 0.84782423 & 5 \\ 0.84770139 & 7 \\ 0.84767066 & 6 \\ 0.84760926 & 3 \end{bmatrix}$$

There is very little difference between the accuracy for different values of λ. During cross-validation, we only really consider the test accuracy (orange bars), but I included the training accuracy for completeness (blue bars). The maximum test accuracy value (0.84782424) occurred at $\lambda = 4$. Taking inspiration from the neural network playground, I tested the values in the figure to the right, which confirms 4 to be a good choice. Bottom left shows the general trend of test accuracy (red) decreasing as regularisation increases, after the initial peak at $\lambda = 4$. However, I am not too confident in this peak being an actual trend – the accuracy oscillates a lot and so this peak could be due to random chance. However, this kid of oscillation is to be expected as the differences in accuracy are so tiny, the slightest changes due to variance in the data are blown out of proportion. Considering extreme values, bottom right shows the accuracy with large values of λ. Performance decreases linearly up to about $\lambda = 10,000$, where performance appears unable to decrease further. This is because the identity classifier (that classifies all observations as 0) has an accuracy of 76%.







The best value, as seen above, was $\lambda = 4$. However, this did not perform particularly well on the test data set, a9a.t. The test accuracy was 0.849764, which is actually lower than the non-regularised result. However, taking the second best value, $\lambda = 5$, does give us a modest test accuracy increase of 0.000368.

λ	Train Acc	Test Acc.	L2 Norm	Log-Likelihood	Convergence
4	0.849022	0.850316	5.032041	-10588.738759	7

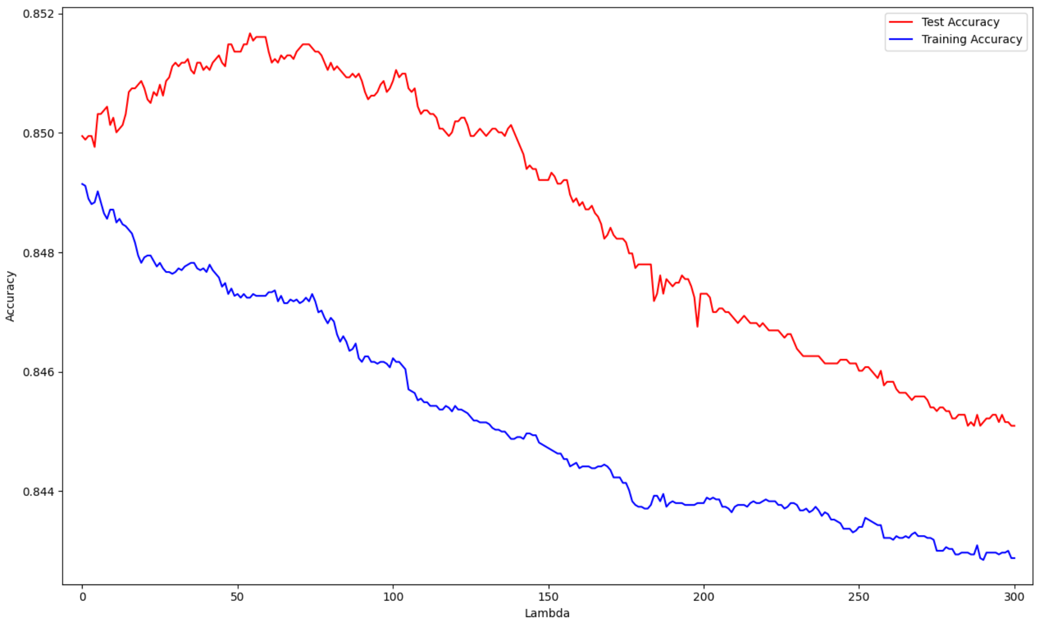
By trying many different λ on the a9a.t data-set, I found a $\lambda = 54$ to be the most effective. The test accuracy was 0.851668, an increase in accuracy of 0.00172 over the non-regularised implementation.

There were several values of λ that achieved the minimum number of iterations to converge, 3. The smallest of these values was $\lambda = 184$. The test accuracy for this λ value was 0.847184.

As a general rule, as λ increases $\|\hat{\boldsymbol{w}}\|_2$ decreases, so there is little value in discussing the $\boldsymbol{w}$ with the smallest L2 norm.

Below is a plot of λ vs test and train accuracy, where we can clearly see the peak in test accuracy at $\lambda = 54$. A key feature to note is that the training accuracy decreases approximately linearly, whereas the test accuracy increases before it decreases. It seems that we were quite unlucky getting the suggestion $\lambda = 4$ as nearly all $\lambda \in (0, 100)$ perform better on the a9a.t test data. Since most $\lambda \in (0, 100)$ give a better test accuracy, we could say that the non-regularised approach over-fits the data more than the regularised approach, and therefore performs worse on the test data. This supports using regularisation on this data.

However, on the flip side, we could also claim that regularisation is not needed on this data. The non-regularised implementation does not have a crazy high accuracy. In addition, the test accuracy is practically the same as the training accuracy (difference is 000803). This is pretty strong evidence against over fitting. Further, regularisation barely increases the test accuracy at all (increase is 0.00172). I find these reasons more compelling, so believe that on this data regularisation is likely not needed, but we could argue either way.

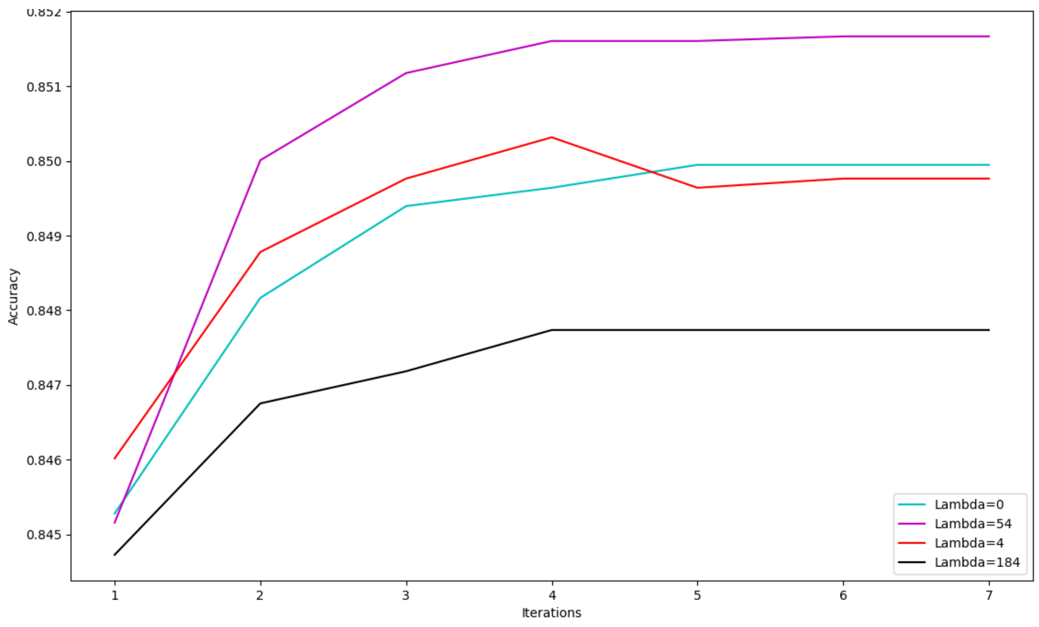


Some variables had effectively no non-zero entries. For example, variable 123 only has one entry out of 32561. I considered removing columns like this that would contribute very little to the results, and the best improvement I saw was removing all columns that had 500 entries or lower. This helped reduce over fitting, and so for $\lambda = 56$, this gave a test accuracy 0.851913, the highest yet. However, we should also take into account the fact that the non-regularised test accuracy also increases to 0.850378.

The table below summarises these findings. After removing rows with 500 entries or fewer, the results were

Comments	λ	Train Acc	Test Acc.	L2 Norm	Log-Likelihood	Convergence
Non-Regularised	0	0.848346	0.850378	5.625195	-10560.127954	7
Highest Test Accuracy	56	0.846503	0.851913	3.601462	-11034.486914	6

The final plot below shows the convergence of non-regularised logistic regression, L2 norm regularised logistic regression with $\lambda = 54$ (best performance), L2 norm regularised logistic regression with $\lambda = 4$ (cross-validation suggested λ value) and L2 norm regularised logistic regression with $\lambda = 184$. Interestingly, the $\lambda = 4$ regularised implementation performs better than the non regularised version up to 4 epochs. Perhaps we are allowing too many epochs.



4.1 Test Accuracy is greater than Training Accuracy?

From several online sources, it appears fairly normal that training accuracy is lower than test accuracy when a regularisation has been applied. However, when $\lambda = 0$ this should not be the case unless for some reason the test data in a9a.t contains "easier" examples than in the test data, which is either pure luck or some fault in the way the a9a and a9a.t data sets were split originally. This strange phenomenon of test accuracy exceeding training accuracy is likely why the result of cross validation ($\lambda = 4$) is different from the best value we found for λ ($\lambda = 54$).

To deal with this strange result, I would probably combine the test and training data, and then split the data into new testing and training sets again myself, and then run the above analysis again.

Another question to consider is why the best regularisation for this data, 54, is so large. Normally one would expect regularisation terms to be 10 or below, as seen on the neural network playground. This is very unusual.

Interestingly, the first thing we notice when trying to apply the iterations on $\boldsymbol{w}$ is that $X^T X$, and hence $X^T R X$ is invertible. To circumvent this issue, I had to use the (Moore-Penrose) pseudo-inverse. However, since the L2 regularised version adds λ to the diagonal, this was not an issue for $\lambda > 0$. It is unusual for $X^T X$ to be invertible. On this data, both the training data matrix and the test data matrix were invertible.