ML Homework #1

Yoke Kai Wen, 2020280598

October 22, 2020

1 Mathematics Basics

1.1 Optimization

Set up Lagrange problem, introducing Lagrange multipliers λ, μ:

$$L(x_1, x_2, \lambda, \mu) = x_1^2 + x_2^2 + \lambda(x_1 + x_2 - 1) + \mu(2x_2 - x_1), \quad \mu \geq 0$$

Apply KKT conditions:

$$\nabla_{x_1} L = 2x_1 + \lambda - \mu = 0 \tag{1}$$

$$\nabla_{x_2} L = 2x_2 + \lambda + 2\mu = 0 \tag{2}$$

From constraints:

$$x_1 + x_2 - 1 = 0 \quad \rightarrow \quad x_1 = 1 - x_2 \tag{3}$$

$$\mu(2x_2 - x_1) = 0 \quad \rightarrow \quad \mu = 0 \quad or \quad x_1 = 2x_2 \tag{4}$$

Solve system of linear equations, assume $\mu > 0$:
Sub (4) into (3): $2x_2 = 1 - x_2 \quad \rightarrow \quad x_1 = \frac{2}{3}, x_2 = \frac{1}{3}$
Sub x_1, x_2 into (1), (2):

$$\frac{4}{3} + \lambda - \mu = 0 \tag{5}$$

$$\frac{2}{3} + \lambda + 2\mu = 0 \tag{6}$$

(6) - (5): $-\frac{2}{3} + 3\mu = 0 \quad \rightarrow \quad \mu = \frac{2}{9}, \lambda = -\frac{10}{9}$
Then, min value of expression given constraints is $\frac{2}{3}^2 + \frac{1}{3}^2 - 1 = -\frac{4}{9}$ when $x_1 = \frac{2}{3}, x_2 = \frac{1}{3}, \lambda = -\frac{10}{9}, \mu = \frac{2}{9}$

1.2 Stochastic Process

Let t = expected no. of tosses to observe k Ts consecutively.
We note that if we see a H between any consecutive T tosses, it breaks the streak and we have to restart again for an expected number of t tosses to achieve k consecutive Ts.

Case	**Description**	**Probability**	**No. of tosses**
0	No wasted toss	$\frac{1}{2^k}$	k
1	1 wasted toss , last occurrence of H in first toss	1/2	t+1
2	2 wasted tosses, last occurrence of H in 2nd toss	1/4	t+2
3	3 wasted tosses, last occurrence of H in 3rd toss	1/8	t+3
k	k wasted tosses, last occurrence of H in the kth trial,	$\frac{1}{2^k}$	t+k

Table 1: Table showing probability and expected number of tosses for each coin toss case

From the cases in Table 1.2, the sum of the number of tosses weighted by its respective probability for each case will give the expected number of tosses t for k consecutive Ts.

$$t = \frac{1}{2^k}k + \frac{1}{2}(t+1) + \frac{1}{4}(t+2) + \frac{1}{8}(t+3) + .. + \frac{1}{2^k}(t+k)$$

Rearranging,

$$t = t(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{2^k}) + (\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + ... \frac{k}{2^k}) + \frac{k}{2^k}$$

Recognising the first term as geometric sequence and second term as arithmetico geometric sequence,

$$t = t(1 - (\frac{1}{2})^k) + ((2 - (\frac{1}{2})^{k-1} - \frac{k}{2^k})) + \frac{k}{2^k}$$

Solving,

$$t = 2^{k+1} - 2$$

Now, we want to calculate expected number of tosses t' to get a H before k consecutive Ts. Again, we list out the possible cases as before: H in the first toss, H in the second toss, ... H in the infiniteth toss before the t tosses that will get us k consecutive Ts.
Then,

$$t' = t + \sum_{r=1}^{\infty} \frac{r}{2^r} = (2^{k+1} - 2) + 2 = 2^{k+1}$$

Therefore, expected number of tosses to achieve HTTT..TT is 2^{k+1}.

2 SVM

Form Lagrangian function, introducing dual variables $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}$:

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \hat{\boldsymbol{\xi}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) = \frac{1}{2}|\boldsymbol{w}|^2 + C\sum_{i=1}^{N}(\xi_i + \hat{xi}_i) \\ + \sum_{i=1}^{N}\alpha_i(y_i - \boldsymbol{w}^T\boldsymbol{x_i} - b - \epsilon - \xi_i) \\ + \sum_{i=1}^{N}\beta_i(-y_i + \boldsymbol{w}^T\boldsymbol{x_i} + b - \epsilon - \xi_i) \\ - \sum_{i=1}^{N}\gamma_i\xi_i \\ - \sum_{i=1}^{N}\lambda_i\hat{\xi}_i \end{aligned} \tag{7}$$

Form Lagrange problem and express primal variables in terms of dual variables.

$$\min_{\boldsymbol{w}, b, \boldsymbol{\xi}, \hat{\boldsymbol{\xi}}} \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \hat{\boldsymbol{\xi}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}), \quad \textit{where} \quad \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda} \geq \boldsymbol{0}$$

Solve by applying KKT conditions:

$$\nabla_{\boldsymbol{w}} L = \boldsymbol{w} - \sum_{i=1}^{N}\alpha_i\boldsymbol{x_i} + \sum_{i=1}^{N}\beta_i\boldsymbol{x_i} = 0 \quad \rightarrow \quad \boldsymbol{w} = \sum_{i=1}^{N}(\alpha_i - \beta_i)\boldsymbol{x_i} \tag{8}$$

$$\nabla_b L = -\sum_{i=1}^{N}\alpha_i + \sum_{i=1}^{N}\beta_i = 0 \quad \rightarrow \quad \sum_{i=1}^{N}(\alpha_i - \beta_i) = 0 \tag{9}$$

$$\nabla_{\boldsymbol{\xi}} L = C\boldsymbol{1} - \boldsymbol{\alpha} - \boldsymbol{\gamma} = \boldsymbol{0} \quad \rightarrow \quad \boldsymbol{\alpha} + \boldsymbol{\gamma} = C\boldsymbol{1} \quad \rightarrow \quad 0 \leq \alpha_i \leq C \tag{10}$$

$$\nabla_{\hat{\boldsymbol{\xi}}} L = C\boldsymbol{1} - \boldsymbol{\beta} - \boldsymbol{\lambda} = \boldsymbol{0} \quad \rightarrow \quad \boldsymbol{\beta} + \boldsymbol{\lambda} = C\boldsymbol{1} \quad \rightarrow \quad 0 \leq \beta_i \leq C \tag{11}$$

From constraints,

$$\alpha_i(y_i - \boldsymbol{w}^T\boldsymbol{x_i} - b - \epsilon - \xi_i) = 0 \quad \rightarrow \quad \alpha_i = 0 \quad \textit{or} \quad \xi_i = y_i - \boldsymbol{w}^T\boldsymbol{x_i} - b - \epsilon \tag{12}$$

$$\beta_i(-y_i + \boldsymbol{w}^T\boldsymbol{x_i} + b - \epsilon - \hat{\xi}_i) = 0 \quad \rightarrow \quad \beta_i = 0 \quad \textit{or} \quad \hat{\xi}_i = -y_i + \boldsymbol{w}^T\boldsymbol{x_i} + b - \epsilon \tag{13}$$

$$\gamma_i\xi_i = 0 \quad \rightarrow \quad \gamma_i = 0 \quad \textit{or} \quad \xi_i = 0 \tag{14}$$

$$\lambda_i\hat{\xi}_i = 0 \quad \rightarrow \quad \lambda_i = 0 \quad \textit{or} \quad \hat{\xi}_i = 0 \tag{15}$$

Replace primal variables $\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\hat{\xi}}$ with dual variables $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}$ and apply constraints in equation (7):

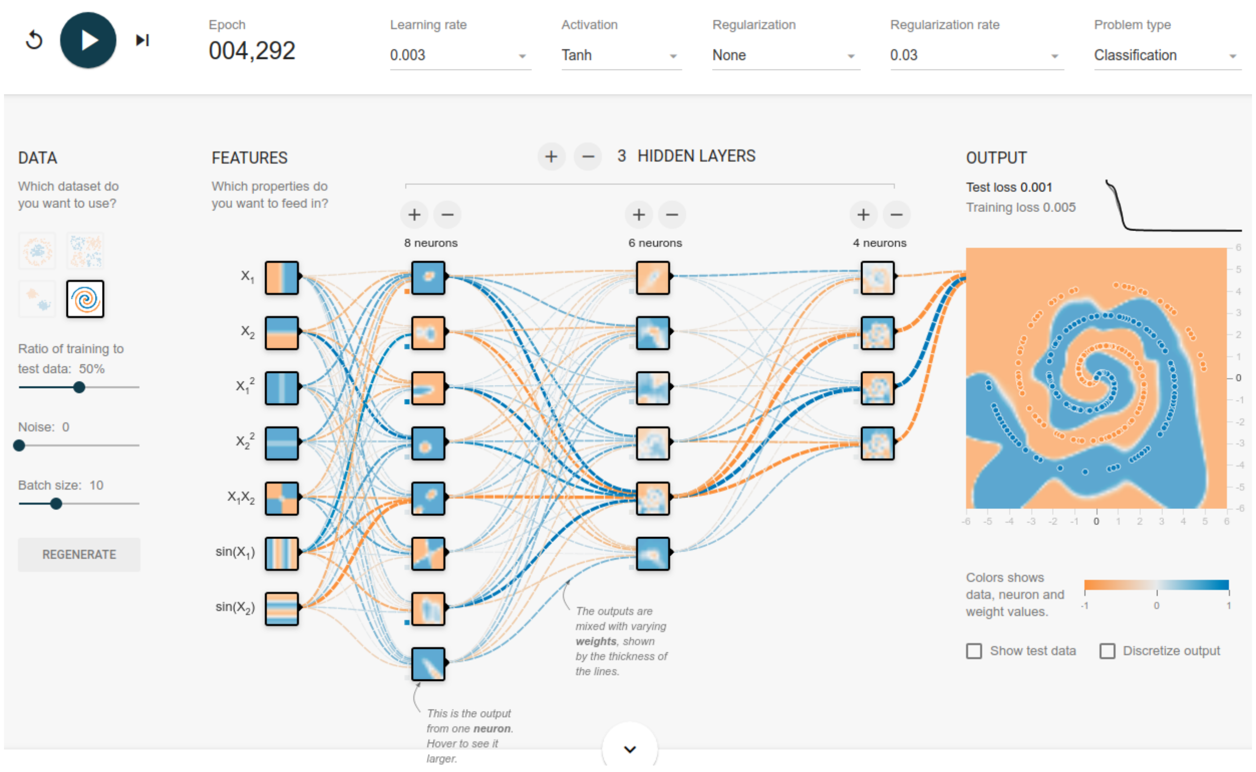
$$
\begin{aligned}
&\max_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\lambda}} \frac{1}{2}(\sum_{i=1}^{N}(\alpha_i-\beta_i)\boldsymbol{x_i})^T(\sum_{j=1}^{N}(\alpha_j-\beta_j)\boldsymbol{x_j}) + \boldsymbol{(\alpha+\gamma)^T\xi} + \boldsymbol{(\beta+\lambda)^T\hat{\xi}} \\
&= \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \frac{1}{2}(\sum_{i=1}^{N}(\alpha_i-\beta_i)\boldsymbol{x_i})^T(\sum_{j=1}^{N}(\alpha_j-\beta_j)\boldsymbol{x_j}) + \boldsymbol{\alpha^T\xi} + \boldsymbol{\beta^T\hat{\xi}} \\
&\quad (since \quad \gamma_i, \lambda_i = 0 \quad or \quad \xi_i, \hat{\xi}_i = 0 \quad from \quad equations \quad 14, 15) \\
&= \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \frac{1}{2}(\sum_{i=1}^{N}(\alpha_i-\beta_i)\boldsymbol{x_i})^T(\sum_{j=1}^{N}(\alpha_j-\beta_j)\boldsymbol{x_j}) + \sum_{i=1}^{N}\alpha_i(y_i-\boldsymbol{w}^T\boldsymbol{x_i}-b-\epsilon) + \sum_{i=1}^{N}\beta_i(-y_i+\boldsymbol{w}^T\boldsymbol{x_i}+b-\epsilon) \\
&\quad (since \quad \xi_i = y_i - \boldsymbol{w}^T\boldsymbol{x_i} - b - \epsilon, \quad \hat{\xi}_i = -y_i + \boldsymbol{w}^T\boldsymbol{x_i} + b - \epsilon \quad from \quad equations \quad 10, 11) \\
&= \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \frac{1}{2}(\sum_{i=1}^{N}(\alpha_i-\beta_i)\boldsymbol{x_i})^T(\sum_{j=1}^{N}(\alpha_j-\beta_j)\boldsymbol{x_j}) + \sum_{i=1}^{N}(\alpha_i-\beta_i)y_i + \sum_{i=1}^{N}(\beta_i-\alpha_i)\boldsymbol{w^T x_i} + \\
&\qquad\qquad b\sum_{i=1}^{N}(\beta_i-\alpha_i) - \epsilon\sum_{i=1}^{N}(\alpha_i+\beta_i) \\
&= \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} -\frac{1}{2}(\sum_{i=1}^{N}(\alpha_i-\beta_i)\boldsymbol{x_i})^T(\sum_{j=1}^{N}(\alpha_j-\beta_j)\boldsymbol{x_j}) + \sum_{i=1}^{N}(\alpha_i-\beta_i)y_i - \epsilon\sum_{i=1}^{N}(\alpha_i+\beta_i) \\
&\qquad\qquad s.t. \quad \sum_{i=1}^{N}(\beta_i-\alpha_i) = 0, \quad 0 \le \alpha_i, \beta_i \le C \\
&\quad (since \quad \sum_{i=1}^{N}(\beta_i-\alpha_i)\boldsymbol{w^T x_i} = \sum_{i=1}^{N}(\beta_i-\alpha_i)(\sum_{j=1}^{N}(\alpha_j-\beta_j)\boldsymbol{x_j})^T\boldsymbol{x_i} \quad from \quad equation \quad 8, \\
&\qquad\qquad b\sum_{i=1}^{N}(\beta_i-\alpha_i) = 0 \quad from \quad equation \quad 9)
\end{aligned}
\tag{16}
$$

Thus, we end up with the dual problem with two dual variables remaining.

3 Deep Neural Networks: Have a Try

3.1 Best configuration for spiral dataset

Figure 1 shows my best configuration for the spiral dataset in terms of achieving the lowest test error.


Epoch 004,292

Learning rate 0.003

Activation Tanh

Regularization None

Regularization rate 0.03

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

FEATURES

Which properties do you want to feed in?

OUTPUT

Test loss 0.001

Training loss 0.005

Colors shows data, neuron and weight values.

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

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

Figure 1: Best configuration parameters

3.2 Effect of hyper-parameters

3.2.1 Learning rate

In general, learning rate should not be too high or too low. If it is too high, the performance with each epoch becomes unstable as observed by an oscillating test/training loss curve, and the model tends to overshoot and might fail to converge. If it is too low, the model takes too many epochs to converge and may also get stuck in local minima. Therefore, we should choose an optimal learning rate such that the convergence rate is reasonable and the classification accuracy is also good. For this dataset, I found that a maximum learning rate of 0.03 can be used such that learning will converge the fastest, while a minimum learning rate of 0.003 has slower (but reasonable) convergence and manages to achieve slightly lower test loss eventually compared to learning rate of 0.03.

3.2.2 Activation function

There are four activation functions available, ReLU, linear, tanh, sigmoid. For this problem, linear and sigmoid both worked extremely poorly. ReLU allows faster convergence than tanh, but ultimately tanh was able to achieve more stable and more accurate classification performance.
In general, ReLU is a more popular activation function because it avoids the vanishing gradient problem which occurs in sigmoid and tanh, furthermore ReLU is simple and fast to implement as it is simply a max function, while sigmoid and tanh requires further computation. However, the reason why ReLU did not work as well in this problem could be due to blowing up of parameters and the dying neuron problem. I found that while my model with ReLU activation was able to

achieve 0.005 test loss in less than a hundred epochs, the test loss starts to increase with more epochs.On the other hand, while my model with tanh activation took slightly longer to achieve a test loss of 0.005 (200 epochs), with more epochs, the test loss decreases gradually. In general, the type of activation suitable depends on the task at hand.

3.2.3 Number of hidden layers

In general, a model with more layers performs better than a shallower model. I tried two model architectures, one with two layers (5 neurons, 2 neurons), the deeper one with three layers (8, 6, 4 neurons). I found that the shallower model only worked well when I used fewer number of input features ($x1, x2, sin(x), cos(x)$) and failed to achieve good accuracy when I input all 7 features. Conversely, the deeper model worked well only when I input all 7 features. I think this happens because a shallower network has lower capacity to learn, so the user has to pre-select features that will be useful for the task, otherwise the network gets confused over bad features. For a deeper network, it has stronger ability to decide what features are useful and form more complex relationships between each feature.
I also note that the shallower network converges much faster and was able to achieve very good test loss (0.002) but the decision boundaries are not very clean, whereas the deeper network takes more epochs to achieve good test loss with clean decision boundaries, and was also able to eventually outperform the shallower network.

3.2.4 Regularisation

I found that no regularisation performed the best. L2 regularisation takes longer to converge but the loss curve was smoother and the decision boundaries were cleaner as well, but only if the regularisation parameter is not too big (0.03). If the parameter is too big, convergence does not occur (underfitting), and if it is too small, it doesn't have a significant stabilising effect (akin to having no regularisation). L1 regularisation with any parameter could not converge for this problem. In general, regularisation prevents overfitting and weights from exploding.

4 IRLS for Logistic Regression

4.1 Derivation of iterative update equations

IRLS is based on Newton's method, that uses the first derivative of $f(x)$ to find the root $x*$ of $f(x) = 0$:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

For logistic regression, we want to find the $\boldsymbol{w}*$ such that the loss function $\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = 0$, so now we have $f(x_t) = \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$ and $f'(x_t) = \nabla^2_{\boldsymbol{w}} J(\boldsymbol{w})$.
Adding L2-norm regularisation parameter λ, the loss function $J(\boldsymbol{w})$ now becomes the log-likelihood constrained by regularisation:

$$\begin{aligned} J(\boldsymbol{w}) &= L(\boldsymbol{w}) - \frac{\lambda}{2}|\boldsymbol{w}|_2^2 \\ &= \sum_{i=1}^{N}(y_i \boldsymbol{w^T x_i} - log(1 + exp(\boldsymbol{w^T x_i}))) - \frac{\lambda}{2}|\boldsymbol{w}|_2^2 \end{aligned} \tag{17}$$

For weight update, $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\nabla_w J(\boldsymbol{w})}{\nabla^2_{\boldsymbol{w}} J(\boldsymbol{w})} = \boldsymbol{H}^{-1}\nabla_{\boldsymbol{w}} J(\boldsymbol{w})$, where $\boldsymbol{H} = \nabla^2_w J(w)$
Compute first derivative of loss function:

$$\begin{aligned}
\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) &= (\sum_i y_i \boldsymbol{x}_i - \boldsymbol{x}_i \psi(\boldsymbol{w^T x_i})) - \lambda \boldsymbol{w} \\
&= [\sum_i (y_i - \mu_i)\boldsymbol{x_i}] - \lambda \boldsymbol{w}, \quad \mu_i = \psi(\boldsymbol{w^T x_i}) \\
&= \boldsymbol{X(y - \mu)} - \lambda \boldsymbol{w}
\end{aligned} \tag{18}$$

Compute Hessian matrix which is the second derivative of loss function:

$$\begin{aligned}
\boldsymbol{H} &= \nabla^2_{\boldsymbol{w}} J(\boldsymbol{w}) \\
&= -(\sum_i \mu_i(1-\mu_i)\boldsymbol{x_i^T x_i}) - \lambda \boldsymbol{I} \\
&= -\boldsymbol{XRX^T} - \lambda \boldsymbol{I}, \quad R_{ii} = \mu_i(1-\mu_i)
\end{aligned} \tag{19}$$

Therefore, weight equation to be implemented is:

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - [-\boldsymbol{XRX^T} - \lambda \boldsymbol{I}]^{-1}[\boldsymbol{X(y-\mu)} - \lambda \boldsymbol{w}]$$

4.2 Results of model on UCI a9a dataset

I implemented the IRLS Logistic Regression model according to the above derivation in Python. When the L2 regularisation parameter λ is set to zero, the model is the same as being not regularised. In Figure 2, I show various performance metrics to investigate the performance of the model at different λ values using the default train-test-split provided in the dataset. In Figure 3, I show the same metrics averaged over 10-fold cross validation.

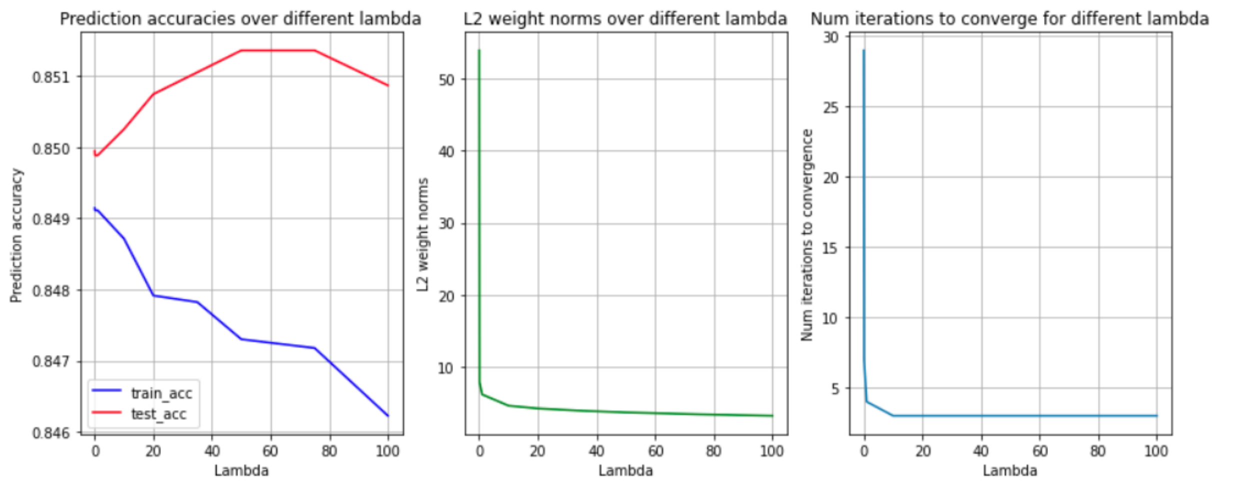


Figure 2: Performance metrics across different regularisation parameters

For the default train-test split, the optimum value for λ seems to be about 50, which achieves 0.851 test accuracy, which seems surprisingly big. In general, regularisation seems to improve test accuracy over having no regularisation, speeds up convergence significantly, and also prevents weights from getting too large. Regularisation also decreases train (in-sample) accuracy as λ increases, this makes sense because regularisation prevents the model from overfitting i.e. memorising the training data too much. However, if λ gets too large, test accuracy starts dropping because the model now underfits. Therefore, care must be taken to ensure the optimum regularisation parameter is chosen.

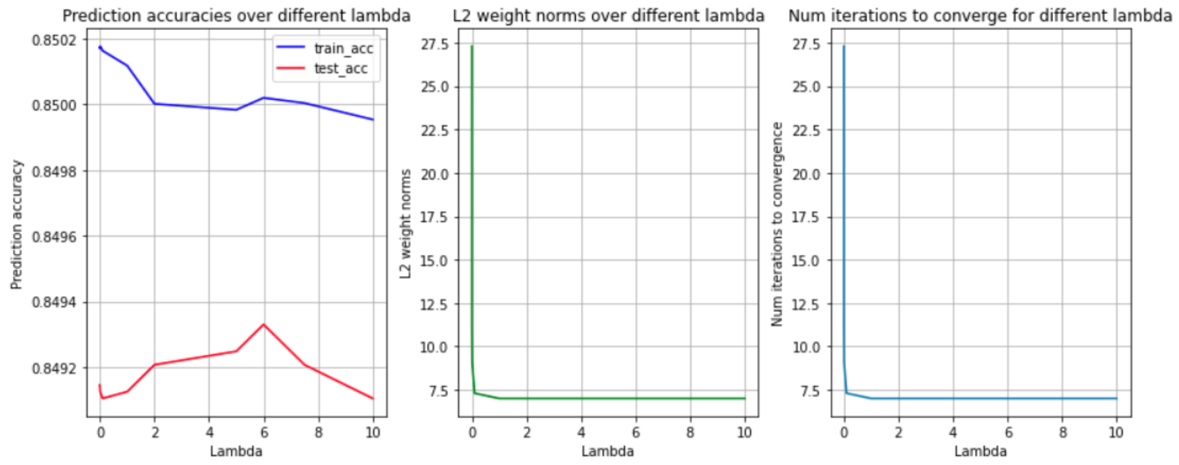


Figure 3: Performance metrics across different regularisation parameters with 10-fold cross validation

Figure 3 shows results after performing 10-fold cross validation, where the entire dataset was divided into ten parts, with each model associated with a λ trained on 9 parts and tested on the remaining part, repeating with each testing part changing, and averaging the performance over the ten train-test splits. I found that the optimum λ now decreases to a more reasonable value of 6 to get a maximum of 0.8493 test prediction accuracy. The other trends and explanations are largely the same as when I used the default train-test split. This reflects that how we choose the train-test split significantly affects our evaluation results, and in general, k-fold cross validation is more reliable than choosing only one train-test split.