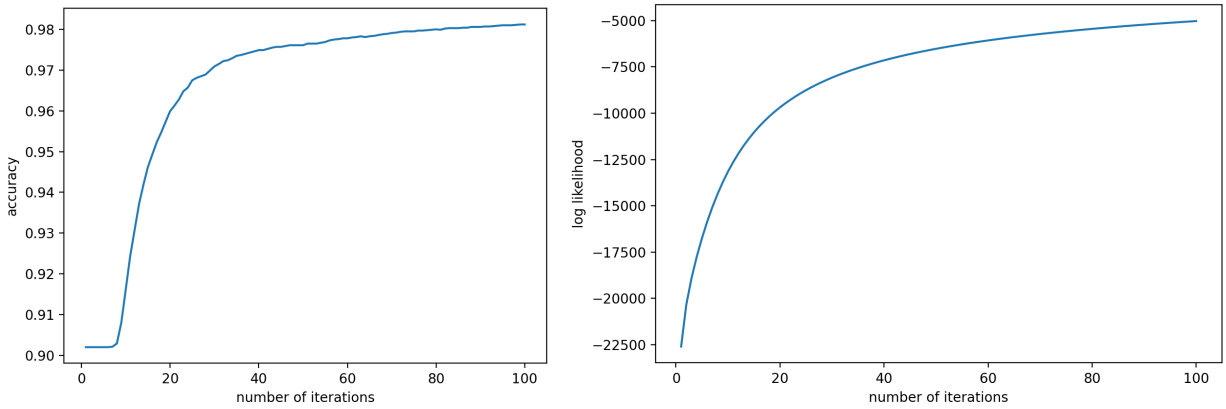
1. Logistic Regression.

The graphs of the Testing **accuracy** v.s. Number of iterations (left) and of the **Log likelihood** v.s. The number of iterations (right) are shown below. And the code deriving the figures is in *logistic_reg.py* in the *src* file.

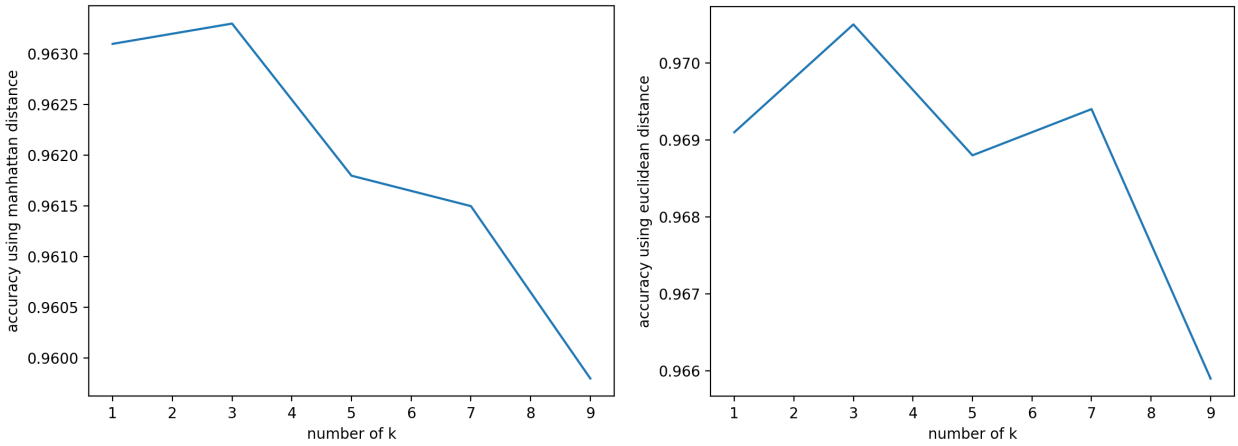


From the graph we can see that as the number of iterations increases, the testing accuracy increases and approaches 98% in the 100th iteration, and the value of log likelihood decreases in magnitude, i.e. increases in value.

This is because logistic regression uses gradient descent (ascent) to find the parameters that maximize the log likelihood, which updates the parameters for each iteration. Thus as the number of iterations increases, the parameters will get closer to the optimal point and so as the value of log likelihood. And the accuracy will increase accordingly.

2. KNN.

The graphs of the prediction accuracy using **Manhattan** distance (left) and of the prediction accuracy using **Euclidean** distance (right) are shown below. And the code deriving the figures is in *knn.py* in the *src* file.

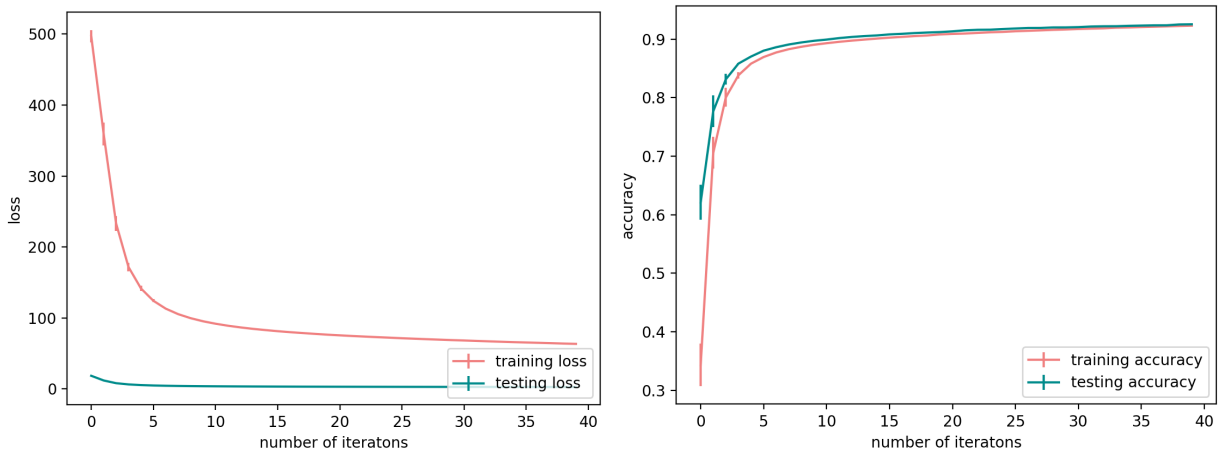


We can observe that the knn algorithm in general performs better when using Euclidean distance, with the highest accuracy of 97%, than using Manhattan distance, with the highest accuracy of 96.3%. Also, the algorithm has the highest accuracy when k = 3 and after this the accuracy tends to decrease as the value of k increases.

When the value of k increases, the model becomes more complex and more biased, which might lead to decreasing accuracy. Also, using Euclidean distance provides better accuracy because it measures the distance between points more accurately.

3. MLP.

The graphs of the Testing and training **loss** v.s. Number of training epochs (left) and of the Training and Testing **accuracy** v.s. The number of training epochs (right) are shown below, with the **curve** representing the **mean** of values in each iteration for the 3 runs and vertical **error bars** representing the **standard deviation** of values in each iteration for the 3 runs. And the code deriving the figures is in *mlp.py* in the *src* file.



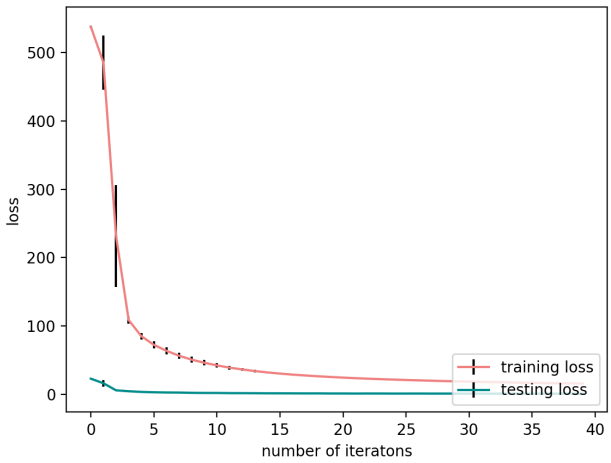
The hyperparameters used are: hidden dimension h = 32, testing_batch_size = 1000, training_batch_size = 256, optimizer = SGD with learning rate lr = 0.01 as recommended.

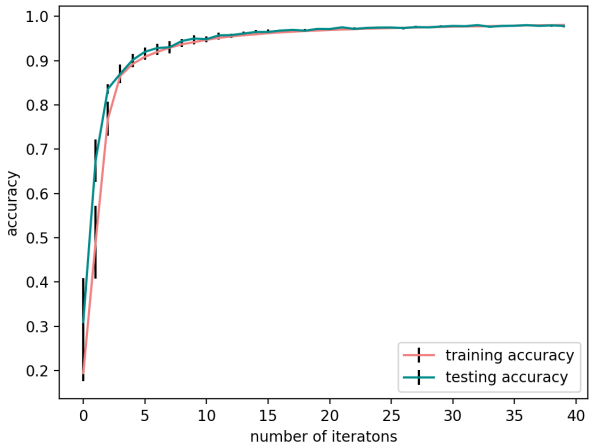
From the graph we can see that as the number of iterations increases, the loss for both testing and training decrease and the accuracy increases. And they both tend to converge at the end of the 40 iterations. Also the standard deviation of the values of the 3 runs tends to get smaller as the number of iterations grows.

Since I used SGD for optimizer, the parameters converge to the optimal value as the number of iterations increase. And the loss decreases and the accuracy increases accordingly during the process. Since in the MLP model I implemented a hidden layer, which helps learn the features better, the loss function and accuracy both converged to desired values in the end. However, since there weren't any other layers such as dropout layers and convolutional layers, the accuracy converges only to around 90% but not higher. Also we can see that the prediction for test data tends to behave better than the training data. This is because the testing loss and accuracy are measured before updating the model in each epoch.

4. CNN.

The graphs of the Testing and training **loss** v.s. Number of training epochs (left) and of the Training and Testing **accuracy** v.s. The number of training epochs (right) are shown below, with the **curve** representing the **mean** of values in each iteration for the 3 runs and vertical **error bars** representing the **standard deviation** of values in each iteration for the 3 runs. And the code deriving the figures is in *cnn.py* in the *src* file.





The hyperparameters used are: testing_batch_size = 1000, training_batch_size = 256, optimizer = SGD with learning rate *lr* = 0.01 as recommended. And the CNN architecture is as shown below.

Layers	
Conv1	[5*5, 10] + ReLU
Pool1	2*2 Max Pooling, Stride 2
Conv2	[5*5, 20] + ReLU
Pool2	2*2 Max Pooling, Stride 2
FC1	50 + ReLU
FC2	10

From the graph we can see that the loss of the training data drops drastically at the first few iterations, and then it smoothly converges to a lower value in the following iterations. For the testing loss, the value also drops in a large value at the beginning, but in general, testing loss is lower than the training loss. The curves for training and testing accuracy have similar values, in which the accuracy grows from around 20% to near 98%. Also, the standard deviation of the values of the 3 runs tends to get smaller as the number of iterations grows.

Because SGD is used for optimizer, the curve will gradually converge to the optimal value. Also since there are two convolutional layers and two fully connected layers in this model, the result will be better compared to the previous model in the last problem. And the accuracy approaches 98% at the end of the iterations.