Binary classification using logistic regression

Practice#2-1

Report by: 완(2019007901)

For this assignment, we will train a model for binary classification using logistic regression similar to the previous assignment and we will use the same datasets as in practice#1-2. But this time, our network will be a double layered network (1 hidden layer and an output layer). Therefore, the aim for this assignment is to change the learning algorithm accordingly and creating additional parameters (another w and b) for each layer.

Estimated unknown function W & b

For our experiments, we will set the number of samples according to the assignment specifications (M=10000, N = 1000) and the number of iterations K will be set to 5000. We realized that for only 5000 iterations, the program would not yield the best accuracy (only ~50%). Therefore, we tried to increase the number of iterations to 500000 and the result is as below.

```
[w1, b1, w2, b2] = [[[-1813.69878086]], [[330901.03284608]], [[8.1351926]], [[-4.34741014]]]
Cost: -0.019030
Cost with n test samples =  -0.2803685484873077
Accuracy for 'm' train samples: 99.31%
Accuracy for 'n' test samples: 99.2%

Process finished with exit code 0
```

To further prove this, we tried to comment out all the training part in the code and directly assign the value of parameters w1,b1,w2,b2 on the accuracy testing part and the parameters are able to yield high accuracy as below.

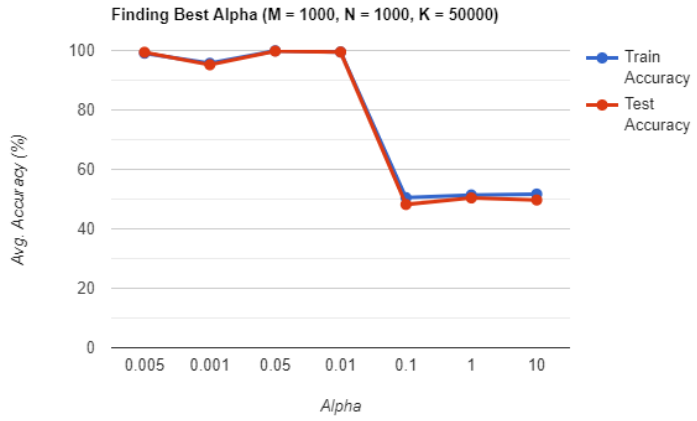
```
# === STEP 2-4 ===
w1 = -1813.69878086
w2 = 8.1351926
b1 = 330901.03284608
b2 = -4.34741014
correct_predict_train = 0
Z1 = np.dot(w1, x_train) + b1
A1 = sigmoid(Z1)
Z2 = np.dot(w2, A1) + b2
A2 = sigmoid(Z2)
for i in range(m):
    if A2[0,i] > 0.5 and y_train[0,i]
        correct_predict_train += 1
    elif A2[0,i] < 0.5 and y_train[0,
        correct_predict_train += 1
train_accuracy = correct_predict_trai
print("Accuracy for 'm' train samples
```

```
for i in range(m) › if A2[0,i] > 0.5 and y_train[0,...
practice2-1 ×
C:\Users\ausba\AppData\Local\Microsoft\Wi
D:\Users\ausba\2022_1_ITE4053\Assignment
  return 1 / (1 + np.exp(-z + 1e-10))
Accuracy for 'm' train samples: 99.42%
Accuracy for 'n' test samples: 99.1%

Process finished with exit code 0
```

Empirically determined (best) hyper parameter, α

To find the alpha value, we will use the same test-and-run method in previous assignment and observe which one yields the highest average accuracy on the training and test samples. The graph below shows the result.



From the graph, we can conclude that when α is between 0.05 and 0.01, it gives the best accuracy for both train and test accuracy. Another observation that we made was, when α is more than 0.1, the changes in parameters w and b are very little after every iteration, which causes the dramatic drop in accuracy to ~50%.

Accuracy

	m=10, n=1000, K=5000	m=100, n=1000, K=5000	m=10000, n=1000, K=5000
Accuracy ('m' train samples)	60.0%	95.0%	93.17%
Accuracy ('n' test samples)	51.6%	94.89 %	94.89%

	m=10000, n=1000, K=10	m=10000, n=1000, K=100	m=1000, n=1000, K=5000
Accuracy ('m' train set)	50.73%	51.03%	95.1%
Accuracy ('n' test samples)	50.4%	49.0%	96.0%

Discussion

We can see that after using a layered network, our accuracy has been improved compared to the last assignment. This is because, by adding more hidden layers, there will be more additional parameters to the model. Hence, this allows the model to fit to more complex functions. Another observation that we saw was the more the number of iteration (K), the higher the accuracy. At K=5000, the accuracy wasn't consistent where it was around 50% to 90%. And as mentioned before, we tested this program at K=500000 and the program consistently output a very high accuracy (~99%) which is obvious, the more the learning time, the better the prediction. We also found out that the performance of learning at m=10000,K=5000 in this program is way faster than previous assignment. In summary, multi-layered network performs way better at handling complex functions. Not to mention the network in this assignment only has a single node for every layer. Therefore, we can see better performance by increasing the number of nodes per layer and number of layers of the network.