CSCE 636: Deep learning(FALL 2022)

Assignment 1

Student name: Siri Namburi, 331007601

Linear Models for handwritten digits classification: In this assignment, you will implement the binary logistic regression model and multi-class logistic regression model on a partial dataset from MNIST

1. Data Preprocessing Code: DataReader.py,main.py

(a) Explain what the function train valid split does and why we need this step

Usage: This function takes a dataset as input and splits it into two parts, according the value of split index. the value of the split index should be less than or equal to the maximum index of the dataset. In the current code , this function is used to split the original training data set into new training data set and the validation data set. After the split it gives 4 outputs namely new training data, new training labels, validation data and validation data labels. The training data would be used to train the model while the validation data is used to estimate a possible accuracy. we wont use validation data for training purposes. This helps us to evaluate the model in advance without the test data and tune our model appropriately. Some uses are

- if error on validation data is large, it means our model is not trained properly and can indicate over fitting. Hence the testing on validation data can help us identify these cases
- it can be used to fine tune values of hyperparameters
- if test data is not available, validation data can be used as an expected measure of the accuracy of the model

This function helps in splitting the input data we have into training data and validation data , and for the above listed reasons we need this step.

(b)Before testing, is it correct to re-train the model on the whole training set? Explain your answer.

As explained in the above answer, validation data is used to tune hyper-parameters and estimate a possible accuracy and performance of the model. It is also used as an indicator to avoid over fitting. Now retraining the model on the whole training data set invalidates all these. Hence the new model may over fit on the training data or can perform worse than the original model because the hyper-parameters are not tuned properly. Hence I feel that it is bad to retrain the model on the whole training data before testing.

d)In the function prepare X, there is a third feature which is always 1. Explain why we need it.

The third feature is always unity, and is used for including the bias term while doing the matrix multiplication of input vectors and weights. It is mainly used for simplifying the math as by adding this feature the original equation $\theta(bias + \mathbf{w^T x})$ becomes $\theta(\mathbf{w^T x})$. Here the new weight vector includes the bias term as well and it is multiplied by one as other weights are multiplied to their respective input features.

f) Visualise the train features (only the first two features) using the training data containing class1 and class 2 data points only Here the two classes are visibly separable using the two input features we considered

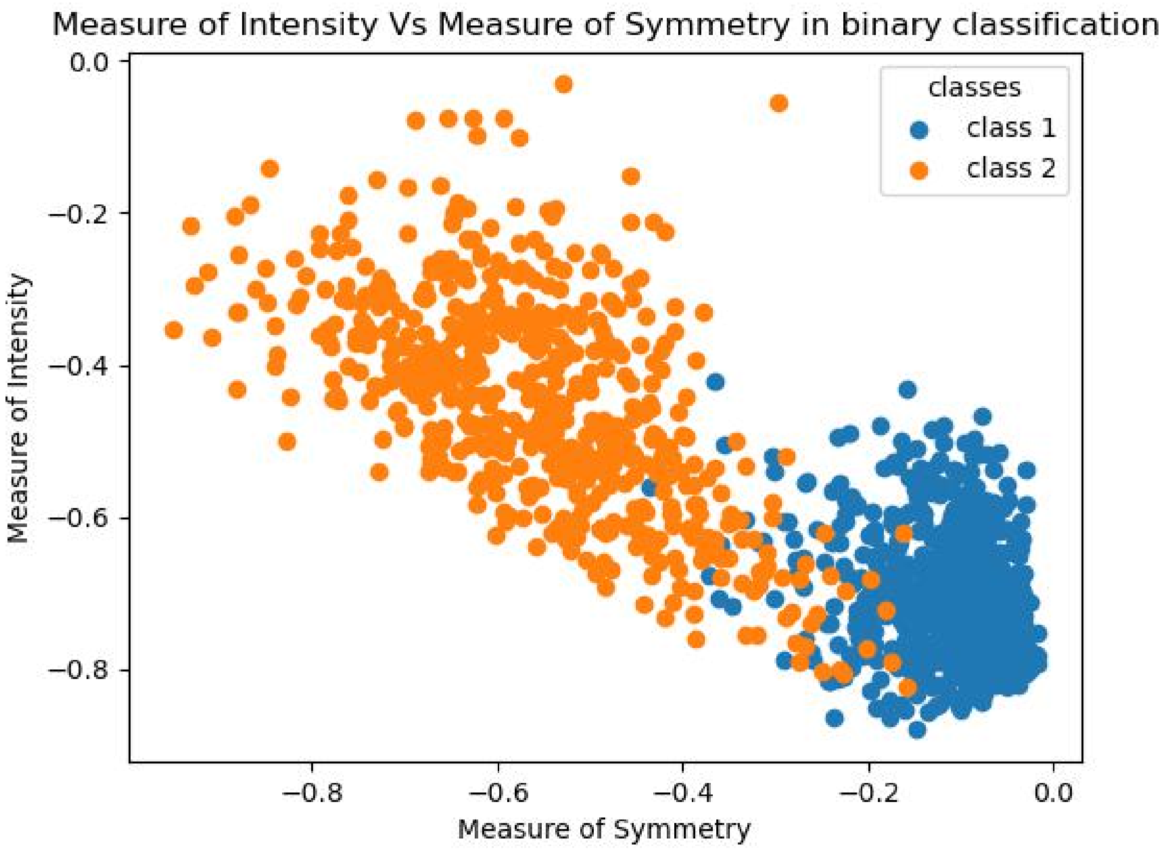


Figure 1: train features

2. Cross-entropy loss : no code modified

(a) Write the loss function E(w) for one training data sample (x, y). Note that the binary labels are 1 and 1.

Assuming weight vector $\mathbf{w}$, the loss function for one training sample is

$$E(W)_{x,y} = \ln\left(1 + e^{-yw^T x}\right)$$

here labels are 1,-1. The loss function needs to be minimised to arrive at our solution.

(b)Compute the gradient $\nabla E(w)$. Please provide intermediate steps of derivation

Let $\mathbf{x}$ be the input vector and y be the target value

$$E(W) = \ln\left(1 + e^{-yw^T\mathbf{x}}\right)$$

$$\nabla E(W) = \nabla \ln\left(1 + e^{-yw^T\mathbf{x}}\right)$$

$$= \frac{1}{1 + e^{-yw^T\mathbf{x}}} \nabla(1 + e^{-yw^T\mathbf{x}})$$

$$= \frac{\nabla e^{-yw^T\mathbf{x}}}{1+e^{-yw^T\mathbf{x}}}$$
$$= \frac{e^{-yw^T\mathbf{x}}}{1+e^{-yw^T\mathbf{x}}}\nabla(-yw^T\mathbf{x})$$
$$= \frac{-ye^{-yw^T\mathbf{x}}}{1+e^{-yw^T\mathbf{x}}}\nabla(w^T\mathbf{x})$$
$$= \frac{-y\mathbf{x}e^{-yw^T\mathbf{x}}}{1+e^{-yw^T\mathbf{x}}}$$
$$= \frac{-y\mathbf{x}}{1+e^{yw^T\mathbf{x}}}$$

Hence $E(w) = \frac{-y\mathbf{x}}{1+e^{yw^T\mathbf{x}}}$

c)Once the optimal w is obtained, it can be used to make predictions as follows, where predicted class of x = 1 if $\theta(w^Tx) >= 0.5$ and -1 otherwise. here function used is $\theta(z) = \frac{1}{1+e^{-z}}$. However, this is not the most efficient way since the decision boundary is linear. Why? Explain it. When will we need to use the sigmoid function in prediction?

Here as the threshold is set to 0.5, the above decision boundary translates as following. Prediction condition for class 1 is $\theta(z) >= 0.5$, i.e , $z >= 0$, and here $z = w^Tx$. Hence the decision condition translates to this: The predicted class of x is 1 if $w^Tx >= 0$ else it is -1. Here w^Tx represents the said linear boundary. As, we can get to the answer by calculating w^Tx itself, calculating the sigmoid on top of it is inefficient as it requires extra time as well as computing power. We use sigmoid function in prediction because

- It has a nice derivative and hence it makes finding the gradient easier for training the model
- Use cases where we require a regression rather than classification,i.e, when we need to find the answer in probabilities that a data point may belong to a particular class. One example is fraud detection where it can compute the probability of a transaction being fraudulent.

d) Prediction rule changed to: predicted class of x is 1 if $\theta(w^Tx) >= 0.9$ and -1 otherwise. Is the decision boundary still linear? Justify briefly
Yes. It can be seen in the following derivation:
Here the threshold =0.9, Hence decision condition for class 1 translates to

$$\theta(z) = \frac{1}{1+e^{-z}} >= 0.9$$

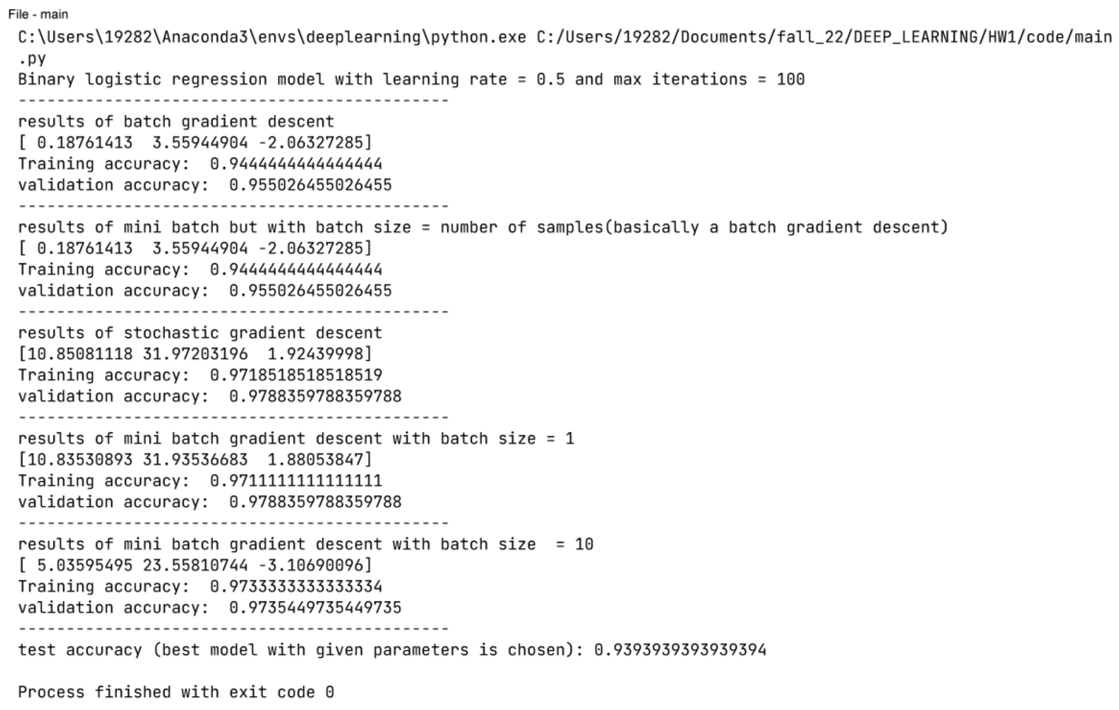
This gives

$$z >= -\ln(\frac{1}{9})$$
$$w^Tx >= \ln(9)$$

The above equation represents a linear equation. Hence, in this case the decision boundary is linear.

e)In light of your answers to the above two questions, what is the essential property of logistic regression that results in the linear decision boundary?
From the above two scenarios it can be seen that for every decision point, $\theta(w^Tx) = threshold$, there is an equivalent decision point $w^Tx >= newthreshold$. As w^Tx is linear , the decision boundary is also linear. In conclusion, the essential property of logistic regression that results in a linear decision boundary is the fact that the signal,i.e, w^Tx is a linear function.

3.Sigmoid logistic regression Code: LogisticRegression.py,main.py
Result for the original hyper parameters given are:



```
File - main
C:\Users\19282\Anaconda3\envs\deeplearning\python.exe C:/Users/19282/Documents/fall_22/DEEP_LEARNING/HW1/code/main
.py
Binary logistic regression model with learning rate = 0.5 and max iterations = 100
---------------------------------------------
results of batch gradient descent
[ 0.18761413  3.55944904 -2.06327285]
Training accuracy:  0.9444444444444444
validation accuracy:  0.955026455026455
---------------------------------------------
results of mini batch but with batch size = number of samples(basically a batch gradient descent)
[ 0.18761413  3.55944904 -2.06327285]
Training accuracy:  0.9444444444444444
validation accuracy:  0.955026455026455
---------------------------------------------
results of stochastic gradient descent
[10.85081118 31.97203196  1.92439998]
Training accuracy:  0.9718518518518519
validation accuracy:  0.9788359788359788
---------------------------------------------
results of mini batch gradient descent with batch size = 1
[10.83530893 31.93536683  1.88053847]
Training accuracy:  0.9711111111111111
validation accuracy:  0.9788359788359788
---------------------------------------------
results of mini batch gradient descent with batch size  = 10
[ 5.03595495 23.55810744 -3.10690096]
Training accuracy:  0.9733333333333334
validation accuracy:  0.9735449735449735
---------------------------------------------
test accuracy (best model with given parameters is chosen): 0.9393939393939394

Process finished with exit code 0
```

Figure 2: Binary logistic regression results with the provided hyperparameters

d)Visualize the results after training by using the function. include the figure in submission

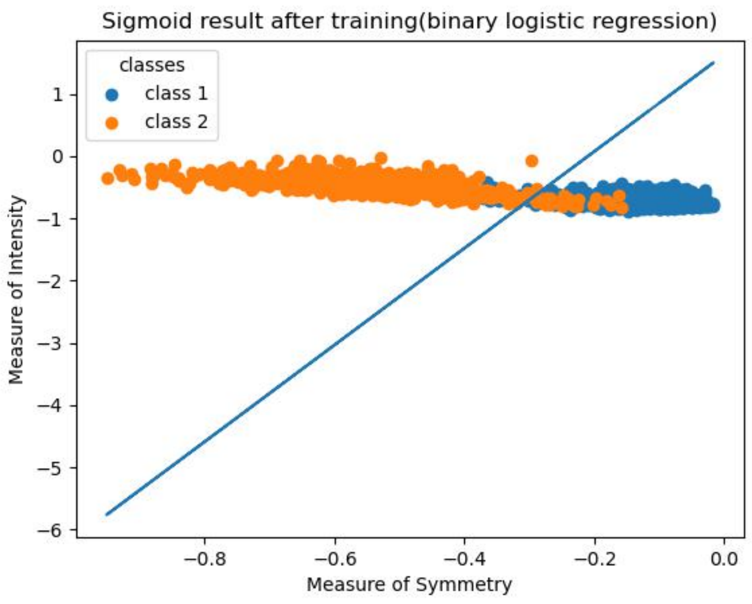
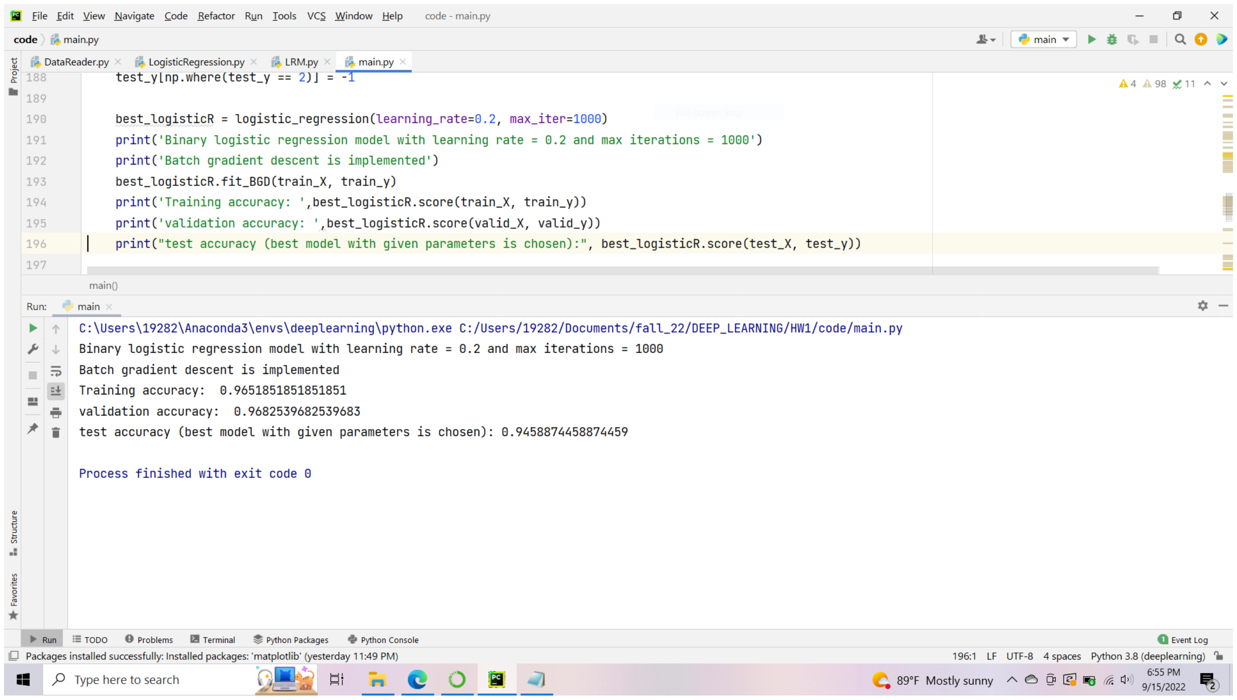


Figure 3: Binary logistic regression model result

e)Implement the testing process and report the test accuracy of your best logistic regression model

The batch gradient descent model has the best accuracy with learning rate = 0.2 and maximum iterations = 1000. Its test accuracy came out to be around 0.94589



```
test_y[np.where(test_y == 2)] = -1

best_logisticR = logistic_regression(learning_rate=0.2, max_iter=1000)
print('Binary logistic regression model with learning rate = 0.2 and max iterations = 1000')
print('Batch gradient descent is implemented')
best_logisticR.fit_BGD(train_X, train_y)
print('Training accuracy: ',best_logisticR.score(train_X, train_y))
print('validation accuracy: ',best_logisticR.score(valid_X, valid_y))
print("test accuracy (best model with given parameters is chosen):", best_logisticR.score(test_X, test_y))
```

```
C:\Users\19282\Anaconda3\envs\deeplearning\python.exe C:/Users/19282/Documents/fall_22/DEEP_LEARNING/HW1/code/main.py
Binary logistic regression model with learning rate = 0.2 and max iterations = 1000
Batch gradient descent is implemented
Training accuracy:  0.9651851851851851
validation accuracy:  0.9682539682539683
test accuracy (best model with given parameters is chosen): 0.9458874458874459

Process finished with exit code 0
```

Figure 4: Best Binary logistic regression model result

4. Softmax logistic regression Code: LRM.py,main.py

d)visualize the results after training by using the function visualize results multi. Include the figure in your submission.

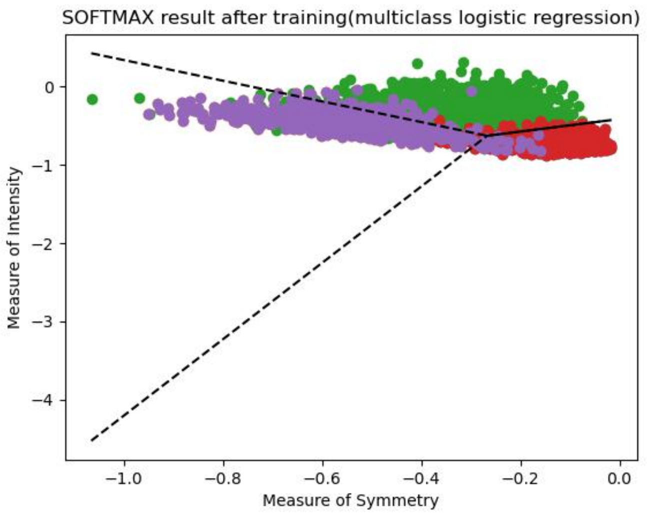
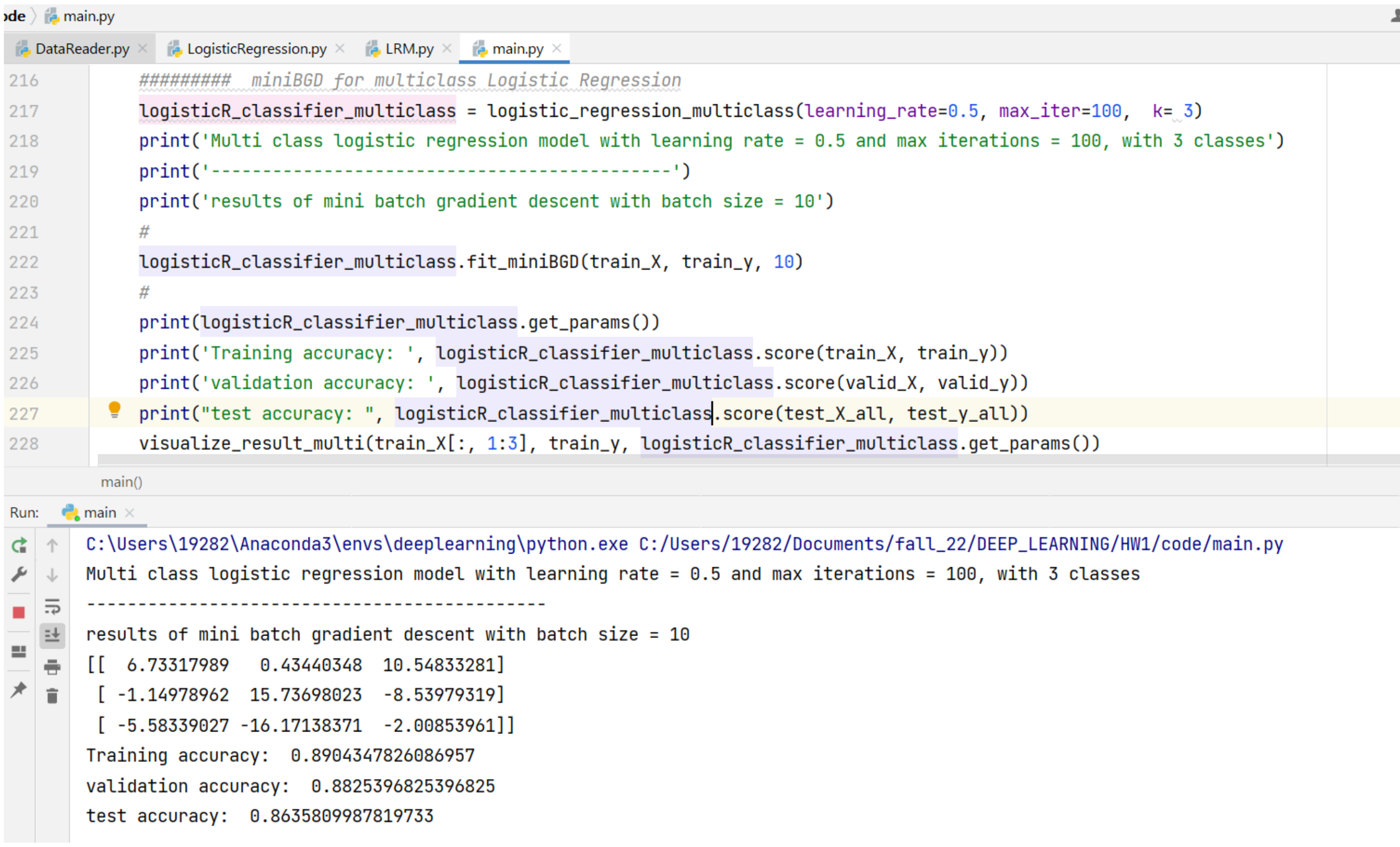


Figure 5: Multiclass logistic regression(softmax)

Accuracy results with given hyperparameters are:

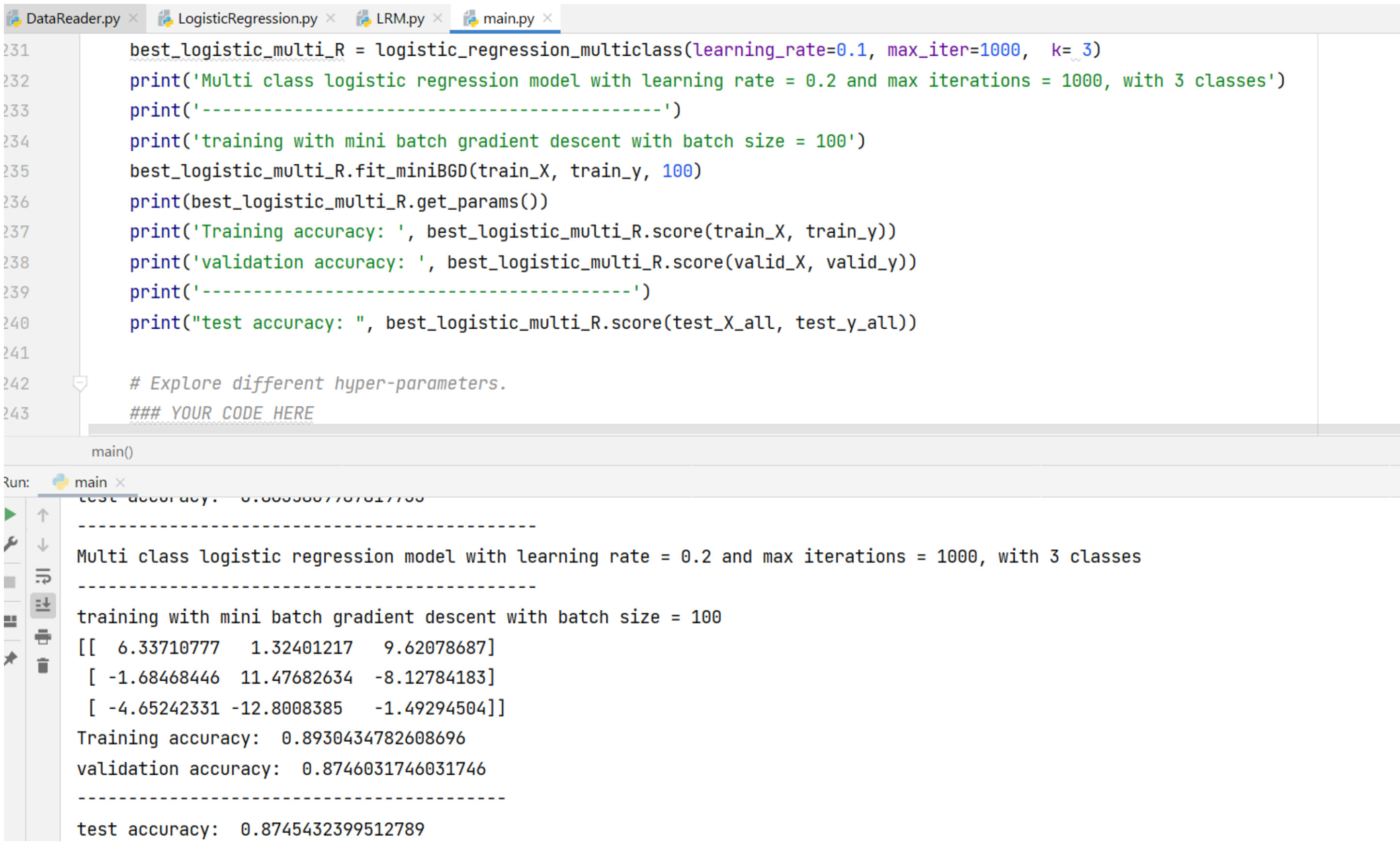


```
    #########  miniBGD for multiclass Logistic Regression
    logisticR_classifier_multiclass = logistic_regression_multiclass(learning_rate=0.5, max_iter=100,  k= 3)
    print('Multi class logistic regression model with learning rate = 0.5 and max iterations = 100, with 3 classes')
    print('--------------------------------------------')
    print('results of mini batch gradient descent with batch size = 10')
    #
    logisticR_classifier_multiclass.fit_miniBGD(train_X, train_y, 10)
    #
    print(logisticR_classifier_multiclass.get_params())
    print('Training accuracy: ', logisticR_classifier_multiclass.score(train_X, train_y))
    print('validation accuracy: ', logisticR_classifier_multiclass.score(valid_X, valid_y))
    print("test accuracy: ", logisticR_classifier_multiclass.score(test_X_all, test_y_all))
    visualize_result_multi(train_X[:, 1:3], train_y, logisticR_classifier_multiclass.get_params())
```

```
C:\Users\19282\Anaconda3\envs\deeplearning\python.exe C:/Users/19282/Documents/fall_22/DEEP_LEARNING/HW1/code/main.py
Multi class logistic regression model with learning rate = 0.5 and max iterations = 100, with 3 classes
--------------------------------------------
results of mini batch gradient descent with batch size = 10
[[  6.73317989   0.43440348  10.54833281]
 [ -1.14978962  15.73698023  -8.53979319]
 [ -5.58339027 -16.17138371  -2.00853961]]
Training accuracy:  0.8904347826086957
validation accuracy:  0.8825396825396825
test accuracy:  0.8635809987819733
```

Figure 6: Multiclass logistic regression(softmax)- given hyperparams result

e)) Implement the testing process and report the test accuracy of your best logistic regression model. The minibatch gradient descent model has the best accuracy with learning rate = 0.2 and maximum iterations = 1000 ans batch size = 100. Its test accuracy came out to be around 0.8745 i.e around 87.5 percent.



```
    best_logistic_multi_R = logistic_regression_multiclass(learning_rate=0.1, max_iter=1000,  k= 3)
    print('Multi class logistic regression model with learning rate = 0.2 and max iterations = 1000, with 3 classes')
    print('-------------------------------------------')
    print('training with mini batch gradient descent with batch size = 100')
    best_logistic_multi_R.fit_miniBGD(train_X, train_y, 100)
    print(best_logistic_multi_R.get_params())
    print('Training accuracy: ', best_logistic_multi_R.score(train_X, train_y))
    print('validation accuracy: ', best_logistic_multi_R.score(valid_X, valid_y))
    print('-----------------------------------------')
    print("test accuracy: ", best_logistic_multi_R.score(test_X_all, test_y_all))

    # Explore different hyper-parameters.
    ### YOUR CODE HERE
```

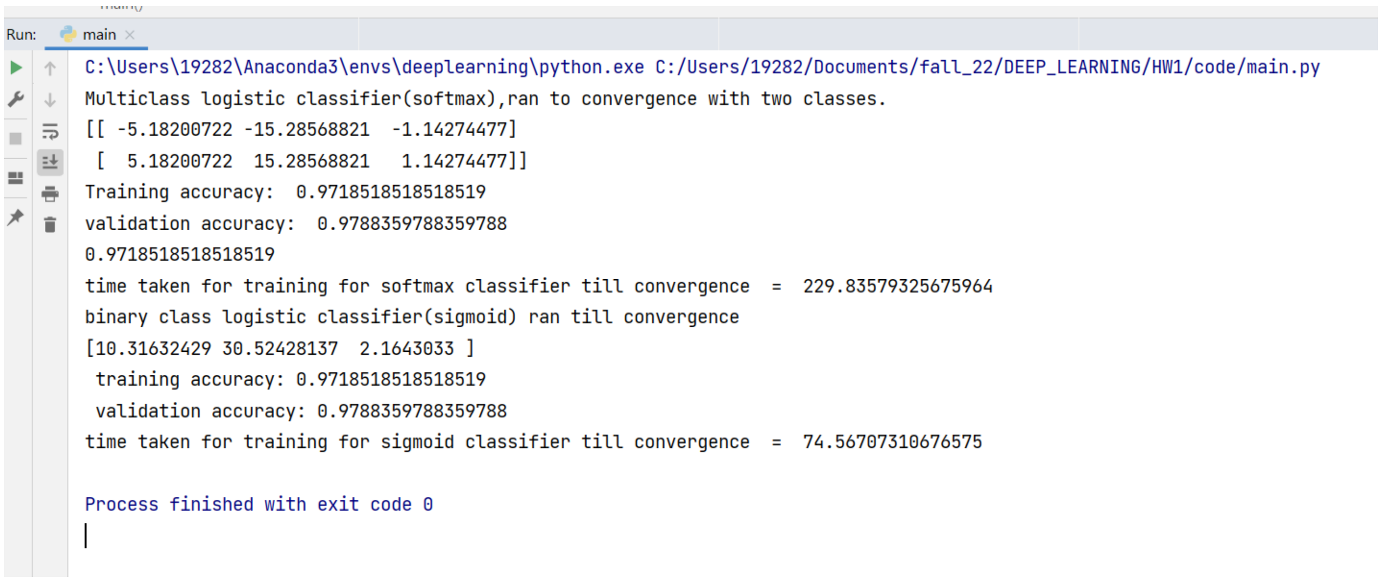
```
--------------------------------------------
Multi class logistic regression model with learning rate = 0.2 and max iterations = 1000, with 3 classes
--------------------------------------------
training with mini batch gradient descent with batch size = 100
[[  6.33710777   1.32401217   9.62078687]
 [ -1.68468446  11.47682634  -8.12784183]
 [ -4.65242331 -12.8008385   -1.49294504]]
Training accuracy:  0.8930434782608696
validation accuracy:  0.8746031746031746
-----------------------------------------
test accuracy:  0.8745432399512789
```

Figure 7: best Multiclass logistic regression(softmax)- result

5.Softmax logistic vs Sigmoid logistic Code:main.py

a)Train the softmax logistic classifier and the sigmoid logistic classifier using the same data until convergence. Compare these two classifiers and report your observations and insights.

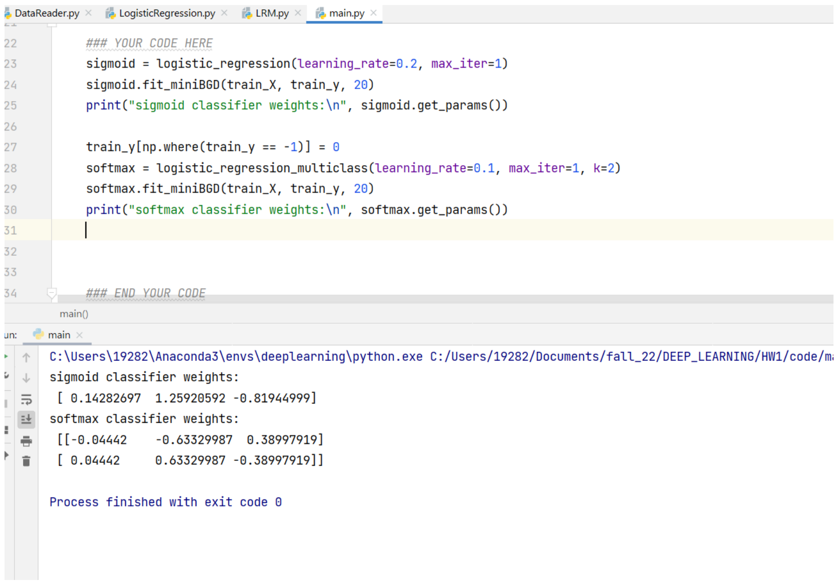
after convergence , both the models are resulting in same training and validation accuracy.(for convergence max iterations are set to 10000). The accuracy and the weight parameters of the two models after convergence are below. One interesting observation is that w2−w1 is almost equal to w where w2 and w1 are weights corresponding to the two classes in the softmax classifier and w is weight vector corresponding to the sigmoid classifier(binary logistic regression).


```
C:\Users\19282\Anaconda3\envs\deeplearning\python.exe C:/Users/19282/Documents/fall_22/DEEP_LEARNING/HW1/code/main.py
Multiclass logistic classifier(softmax),ran to convergence with two classes.
[[ -5.18200722 -15.28568821  -1.14274477]
 [  5.18200722  15.28568821   1.14274477]]
Training accuracy:  0.9718518518518519
validation accuracy:  0.9788359788359788
0.9718518518518519
time taken for training for softmax classifier till convergence  =  229.83579325675964
binary class logistic classifier(sigmoid) ran till convergence
[10.31632429 30.52428137  2.1643033 ]
 training accuracy: 0.9718518518518519
 validation accuracy: 0.9788359788359788
time taken for training for sigmoid classifier till convergence  =  74.56707310676575

Process finished with exit code 0
```

Figure 8: convergence results of softmax and sigmoid classifiers

b)Explore the training of these two classifiers and monitor the gradients/weights. How can we set the learning rates so that w1 −w2 = w holds for all training steps? if we set the learning rate of the sigmoid classifier as double of that of soft-max classifier we can achieve the above condition for all training steps. This is observed through experimentation as well.


```
    ### YOUR CODE HERE
    sigmoid = logistic_regression(learning_rate=0.2, max_iter=1)
    sigmoid.fit_miniBGD(train_X, train_y, 20)
    print("sigmoid classifier weights:\n", sigmoid.get_params())

    train_y[np.where(train_y == -1)] = 0
    softmax = logistic_regression_multiclass(learning_rate=0.1, max_iter=1, k=2)
    softmax.fit_miniBGD(train_X, train_y, 20)
    print("softmax classifier weights:\n", softmax.get_params())



    ### END YOUR CODE
```

```
C:\Users\19282\Anaconda3\envs\deeplearning\python.exe C:/Users/19282/Documents/fall_22/DEEP_LEARNING/HW1/code/m
sigmoid classifier weights:
 [ 0.14282697  1.25920592 -0.81944999]
softmax classifier weights:
 [[-0.04442    -0.63329987  0.38997919]
 [ 0.04442     0.63329987 -0.38997919]]

Process finished with exit code 0
```

Figure 9: sigmoid and softmax classifier results single epoch