NYCU Pattern Recognition, Homework 2

Ivan K. 0860838

PART 1, CODING

Logistic Regression Model

1. (0%) Show the learning rate, epoch, and batch size that you used.
   Learning rate: 0.0001
   Epochs: 100
   Batch size: 32

```
# For Q1
lr = 0.0001
batch_size = 32
epoch = 100

logistic_reg = MultiClassLogisticRegression()
logistic_reg.fit(X_train, y_train, lr=lr, batch_size=batch_size, epoch=epoch)
```

✓ 0.2s

2. (5%) What's your training accuracy?
   Training acc: 0.877

```
# For Q2
print('Training acc: ', logistic_reg.evaluate(X_train, y_train))
```

✓ 0.2s

```
Training acc:  0.877
```

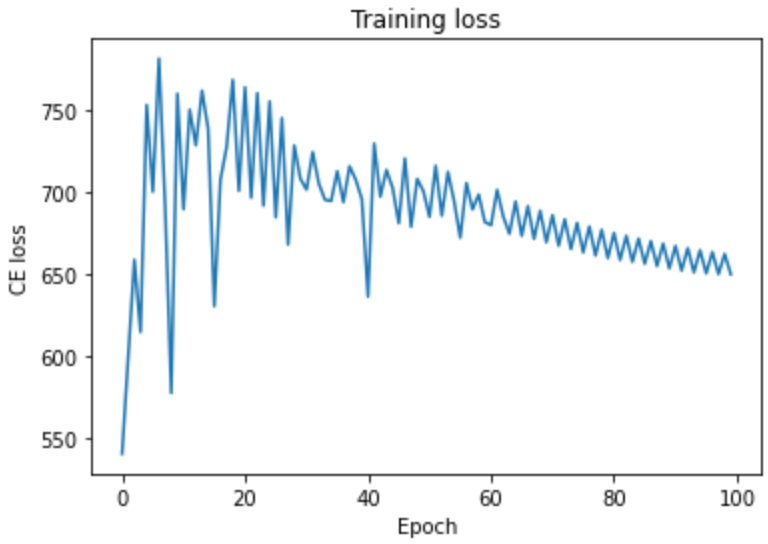
3. (5%) What's your testing accuracy?
   Testing acc: 0.893

```
# For Q3
print('Testing acc: ', logistic_reg.evaluate(X_test, y_test))
```

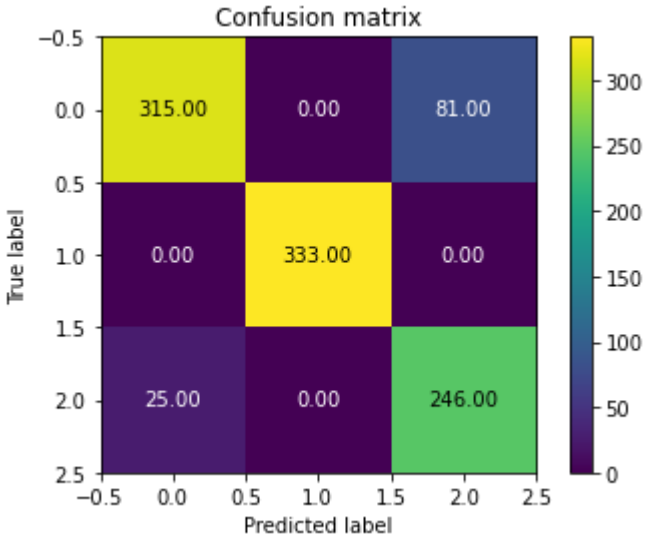
✓ 0.3s

```
Testing acc:  0.893
```

4. (5%) Plot the learning curve of the training. (x-axis=epoch, y-axis=loss)



5. (5%) Show the confusion matrix on testing data



Fisher's Linear Discriminant (FLD) Model

6. (2%) Compute the mean vectors (i=1, 2, 3) of each class on training data.mi

Class mean vector:

[[-4.17505764 6.35526804]
[-9.43385176 -4.87830741]
[-2.54454008 7.53144179]]

```
# For Q6
print("Class mean vector: ", fld.mean_vectors)
```
✓ 0.4s

```
Class mean vector:  [[-4.17505764  6.35526804]
 [-9.43385176 -4.87830741]
 [-2.54454008  7.53144179]]
```

7. (2%) Compute the within-class scatter matrix on training data.SW

Within-class scatter matrix SW:

[[11190.71043049 20368.6120689]
 [20368.6120689 43588.79888566]]

```
# For Q7
print("Within-class scatter matrix SW: ", fld.sw)
```
✓ 0.3s

```
Within-class scatter matrix SW:  [[11190.71043049 20368.6120689 ]
 [20368.6120689  43588.79888566]]
```

8. (2%) Compute the between-class scatter matrix on training data.SB

Between-class scatter matrix SB:

[[28930.94537388 -16146.88066567]
 [-16146.88066567   9011.86435017]]

```
# For Q8
print("Between-class scatter matrix SB: ", fld.sb)
```
✓ 0.3s

```
Between-class scatter matrix SB:  [[ 28930.94537388 -16146.88066567]
 [-16146.88066567   9011.86435017]]
```

9. (4%) Compute the Fisher's linear discriminant on training data.w

Weights: [0.90001598 0.48735181]

```
# For Q9
print("W: ", fld.w)
```
✓ ✓ 0.4s

```
W:  [0.90001598 0.48735181]
```

10. (8%) Project the testing data to get the prediction using the shortest distance to the class mean. Report the accuracy score and draw the confusion matrix on testing data.

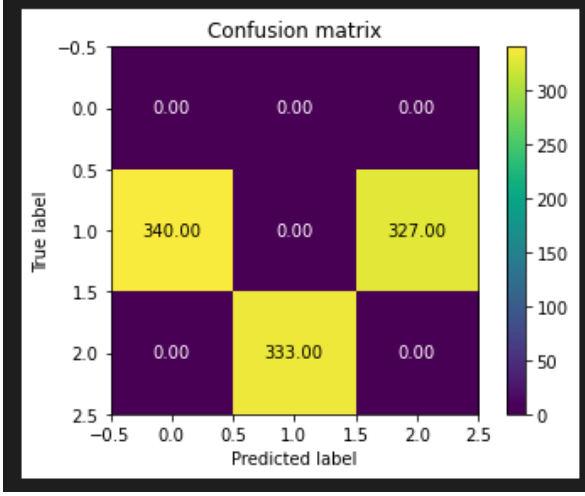
FLD using class mean, accuracy: 0.66

```python
# For Q10
y_pred = fld.predict_using_class_mean(X_train, y_train, X_test)
print("FLD using class mean, accuracy: ", fld.accuracy_score(y_test, y_pred))

fld.show_confusion_matrix(y_test, y_pred)
```

✓ ✓ 0.2s

```
FLD using class mean, accuracy:  0.66
```



11. (8%) Project the testing data to get the prediction using K-Nearest-Neighbor. Compare the accuracy score on the testing data with K values from 1 to 5.

FLD using knn (k=1), accuracy: 0.846
FLD using knn (k=2), accuracy: 0.827
FLD using knn (k=3), accuracy: 0.861
FLD using knn (k=4), accuracy: 0.863
FLD using knn (k=5), accuracy: 0.872

```python
# For Q11
y_pred_k1 = fld.predict_using_knn(X_train, y_train, X_test, k=1)
print("FLD using knn (k=1), accuracy: ", fld.accuracy_score(y_test, y_pred_k1))

y_pred_k2 = fld.predict_using_knn(X_train, y_train, X_test, k=2)
print("FLD using knn (k=2), accuracy: ", fld.accuracy_score(y_test, y_pred_k2))

y_pred_k3 = fld.predict_using_knn(X_train, y_train, X_test, k=3)
print("FLD using knn (k=3), accuracy: ", fld.accuracy_score(y_test, y_pred_k3))

y_pred_k4 = fld.predict_using_knn(X_train, y_train, X_test, k=4)
print("FLD using knn (k=4), accuracy: ", fld.accuracy_score(y_test, y_pred_k4))

y_pred_k5 = fld.predict_using_knn(X_train, y_train, X_test, k=5)
print("FLD using knn (k=5), accuracy: ", fld.accuracy_score(y_test, y_pred_k5))
```

✓ 0.3s

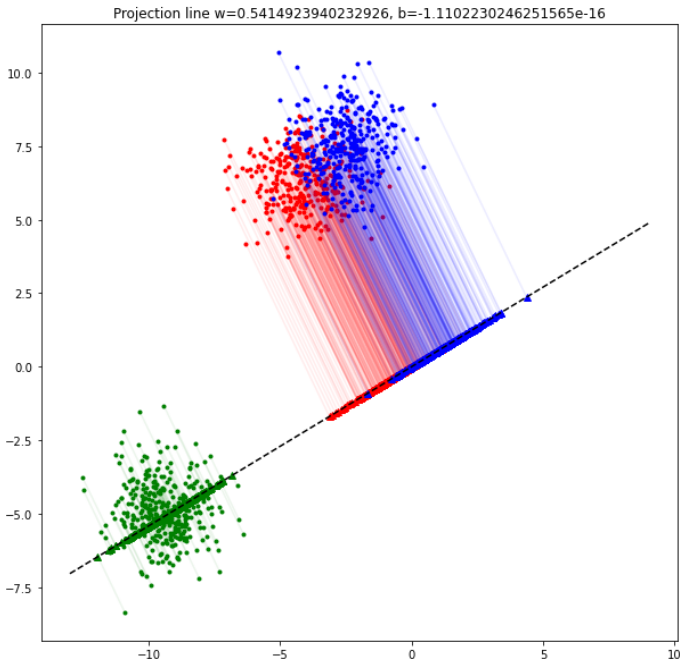
```
FLD using knn (k=1), accuracy:  0.846
FLD using knn (k=2), accuracy:  0.827
FLD using knn (k=3), accuracy:  0.861
FLD using knn (k=4), accuracy:  0.863
FLD using knn (k=5), accuracy:  0.872
```

12. (4%)

1) Plot the best projection line on the training data and show the slope and intercept on the title (you can choose any value of intercept for better visualization

2) colorize the training data with each class

3) project all training data points on your projection line. Your result should look like the below image (This image is for reference, not the answer)

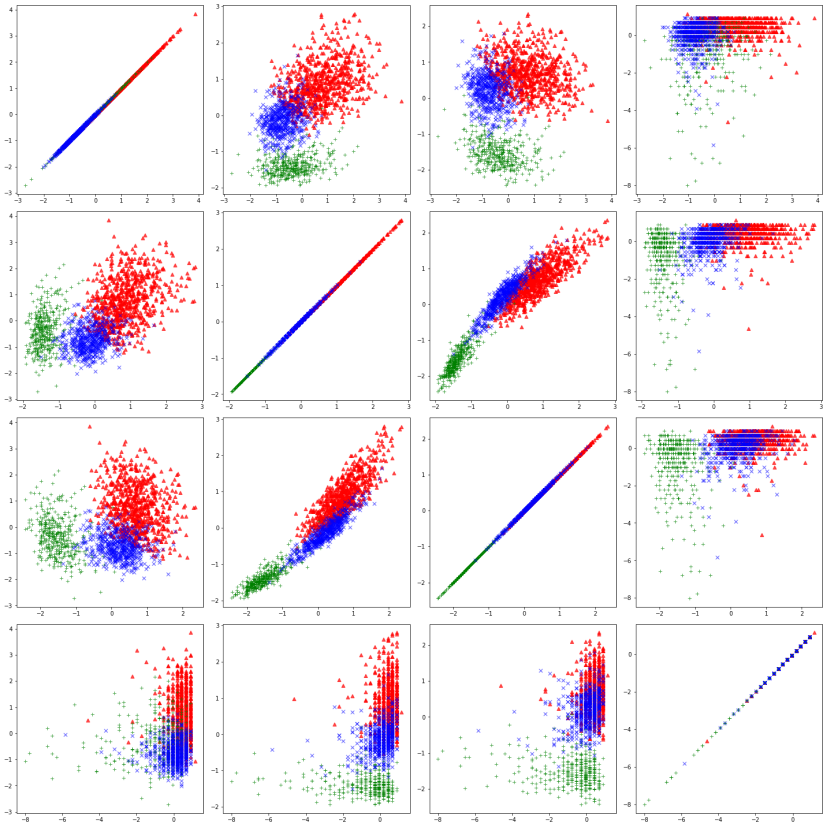


Train your own model

13. Explain how you chose your model and what feature processing you have done in detail. Otherwise, no points will be given.

Point	Accuracy
20	testing acc > 0.921
15	0.91 < testing acc <= 0.921
8	0.9 < testing acc <= 0.91
0	testing acc <= 0.9

For the model I choose Logistic regression because I can change parameters of this model (train it, or make it better). I tried data upsampling and polynomial features first, they don't affect performance in any significant way. At the end I just normalized data and



Training data visualization.

I noticed that if you train model long enough no those features then you can easily achieve above 0.9 accuracy on validation set.

Learning rate = 1e-4

Batch size = 170 (x_train.shape[0]//10)

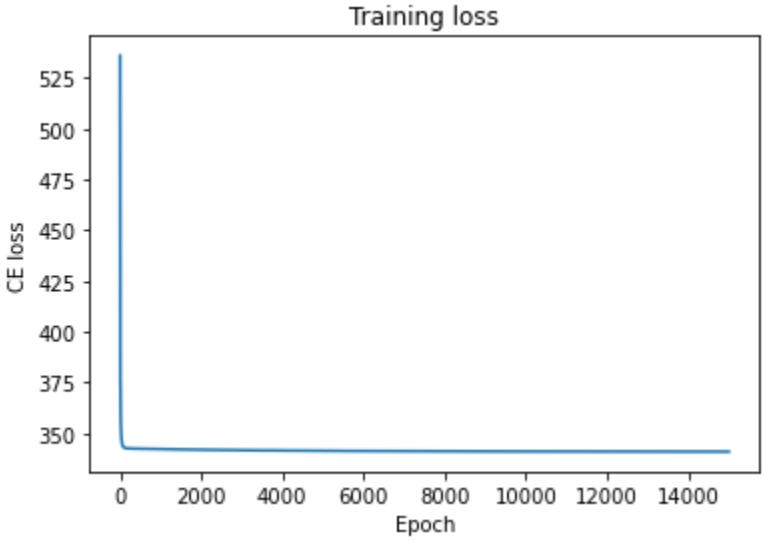
Epoch = 15000

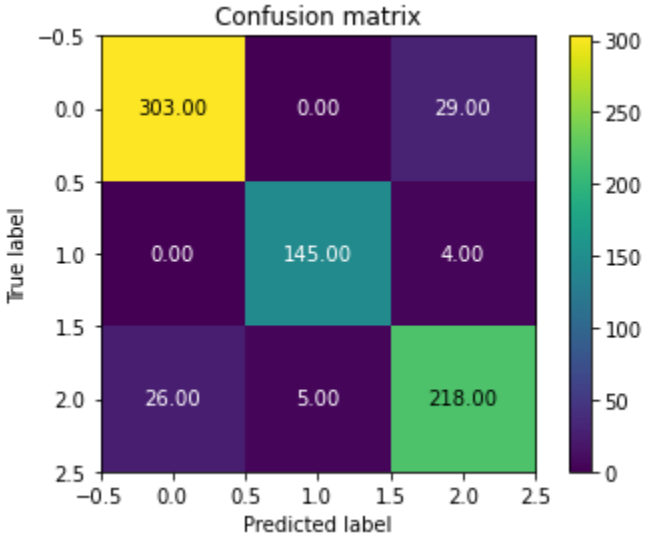
```
lr = 1e-4
batch_size = x_train.shape[0]//10
epoch = 15000

print(batch_size)

logistic_reg = MultiClassLogisticRegression(x_train.shape[-1], y_train.shape[-1])
```

✓ ✓ ✓ 0.3s





Training acc: 0.9365825014679976

Testing acc: 0.9246575342465754

```
1  print('Training acc: ', logistic_reg.evaluate(x_train, y_train))
2  print('Testing acc: ', logistic_reg.evaluate(x_val, y_val))
3  logistic_reg.plot_curve()
4  logistic_reg.show_confusion_matrix(x_val, y_val)
```

✓ 0.2s

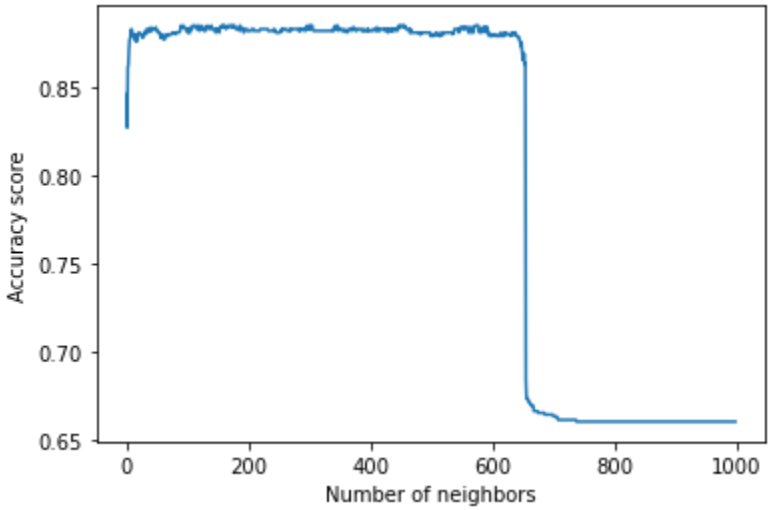
```
Training acc:  0.9365825014679976
Testing acc:  0.9246575342465754
```

Part. 2, Questions

(6%) 1. Discuss and analyze the performance
a) between Q10 and Q11, which approach is more suitable for this dataset. Why?
b) between different values of k in Q11. (Which is better, a larger or smaller k?
Does this always hold?)

a) From the test set (validation, because we have labels) we can see that our accuracy on this dataset is better after using KNN, so this is the reasoning why we should go with it. We can also assume that projected distributions that are used in KNN make decisions on projected(separated) distributions and we make a decision not on one point as we do in the mean class prediction, so based on that we get more predictive power.

b) As you can see from the figure below, at the beginning as we increase as we increase the number of neighbors our accuracy increases as well, but at some point it saturates and doesn't go any higher until we reach more than around 600 neighbors, accuracy starts to plummet down and stays there.



(6%) 2. Compare the sigmoid function and softmax function.

Sigmoid can be used to map any number to the interval between zero and one, it can be used as probability or binary class. Softmax for the other hand maps an often two-dimensional input of numbers to a probability distribution where the sum of probabilities sum up to one for each sample, this function often used for multi-class classification problems. Both of those functions take numbers and try to represent them as probability distributions, but the first one only maps on one distribution when the softmax maps on multiple ones.

(6%) 3. Why do we use cross entropy for classification tasks and mean square error for regression tasks?

Cross-entropy loss is most commonly used for classification tasks because it deals with class distributions and differences between those probabilities. Mean squared error (MSE) is used for regression tasks to measure distance between predicted values and target ones. MSE is not suitable for classification because it does not taking into account probability distributions of predicted classes and can not compute adequate measure of distance between them.

(6%) 4. In Q13, we provide an imbalanced dataset. Are there any methods to improve Fisher Linear Discriminant's performance in handling such datasets?

Classical methods include: oversampling class with least number of samples or undersampling class with overwhelming number of samples; mapping into higher dimensional space where separability can be better.

Using generative adversarial networks (GANs) to generate more data, can be tried as one of the new deep learning based approaches.

(6%) 5. Calculate the results of the partial derivatives for the following equations. (The first one is binary cross-entropy loss, and the second one is mean square error loss followed by a sigmoid function.)

$$\frac{\partial}{\partial x}\left(\mathrm{y} * \ln\left(\sigma(x)\right) + (1-y) * \ln\left(1-\sigma(x)\right)\right)$$

$$\frac{\partial}{\partial x}\left(\left(\mathrm{y}-\sigma(x)\right)^2\right)$$

$$\frac{\partial}{\partial x}(y-\sigma(x))^2 =$$
$$2(y-\sigma(x))\frac{\partial}{\partial x}(y-\sigma(x)) =$$
$$2(y-\sigma(x))(\frac{\partial}{\partial x}y - \frac{\partial}{\partial x}\sigma(x)) =$$
$$2(y-\sigma(x))(-\frac{\partial}{\partial x}\sigma(x)) =$$
$$-2(y-\sigma(x))(\sigma(x)(1-\sigma(x))) =$$
$$-2y\sigma(x)(1-\sigma(x)) + 2\sigma(x)^2(1-\sigma(x))$$

$$\frac{\partial}{\partial x}(y*ln(\sigma(x)) + (1-y)*ln(1-\sigma(x))) =$$
$$\frac{\partial}{\partial x}([1]+[2]) =$$
$$[1]$$
$$\frac{\partial}{\partial x}(y*ln(\sigma(x))) =$$
$$y\frac{\partial}{\partial x}(ln(\sigma(x))) =$$
$$y\frac{1}{\sigma(x)}\frac{\partial}{\partial x}(\sigma(x)) =$$
$$\frac{y\sigma(x)(1-\sigma(x))}{\sigma(x)}$$
$$[2]$$
$$\frac{\partial}{\partial x}(1-y)*ln(1-\sigma(x)) =$$
$$(1-y)\frac{\partial}{\partial x}ln(1-\sigma(x)) =$$
$$(1-y)\frac{1}{1-\sigma(x)}\frac{\partial}{\partial x}(1-\sigma(x)) =$$
$$(1-y)\frac{1}{1-\sigma(x)}(-\sigma(x)(1-\sigma(x))) =$$
$$\frac{-(1-y)\sigma(x)(1-\sigma(x))}{1-\sigma(x)}$$
$$\frac{\partial}{\partial x}([1]+[2]) = \frac{y\sigma(x)(1-\sigma(x))}{\sigma(x)} - \frac{(1-y)\sigma(x)(1-\sigma(x))}{1-\sigma(x)} =$$
$$y(1-\sigma(x)) - (1-y)\sigma(x)$$