NYCU Pattern Recognition, Homework 2

311551059, 陳昱丞

Part. 1, Coding (70%):

You should type the answer and also screenshot at the same time. Otherwise, no points will be given. The screenshot and the figures we provided below are just examples. **The results below are not guaranteed to be correct.** Please convert it to a pdf file before submission. You should use English to answer the questions. After reading this paragraph, you can delete it.

Logistic Regression Model

1. (0%) Show the learning rate, epoch, and batch size that you used.
   learning rate: 0.001
   epoch: 2500
   batch size: len(X_train)=1000

```
# For Q1
lr = 0.001
batch_size = len(X_train)
epoch = 2500

logistic_reg = MultiClassLogisticRegression(3)
logistic_reg.fit(X_train, y_train, lr=lr, batch_size=batch_size, epoch=epoch)
```

2. (5%) What's your training accuracy?
   training accuracy: 0.894

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

```
Training acc:  0.894
```

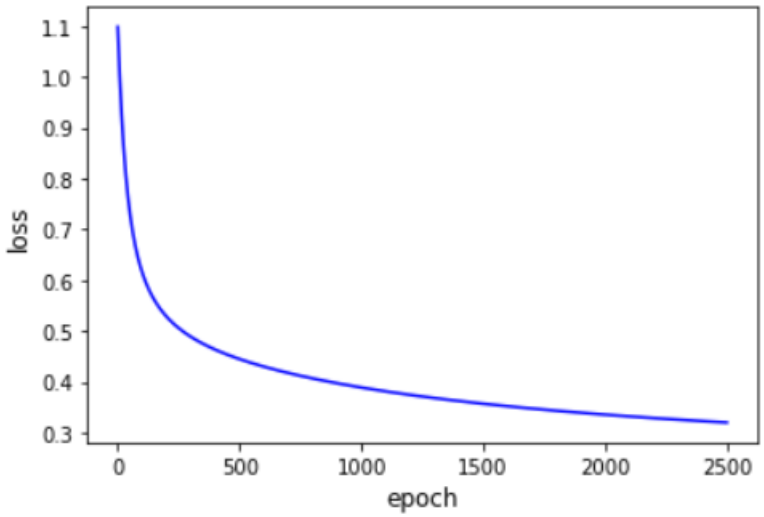
3. (5%) What's your testing accuracy?
   testing accuracy: 0.884

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

```
Testing acc:  0.884
```

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

```
# For Q4
logistic_reg.plot_curve()
```



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

```
# For Q5
logistic_reg.show_confusion_matrix(X_test, y_test)
```

```
[[281   0  59]
 [  0 333   0]
 [ 57   0 270]]
```

confusion matrix:
[[281 0 59]
[0 333 0]
[57 0 270]]

Fisher's Linear Discriminant (FLD) Model

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

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

```
Class mean vector:  [array([-4.17505764,  6.35526804]), array([-9.43385176, -4.87830741]), array([-2.54454008,  7.53144179])]
```

m1: [-4.17505764, 6.35526804]
m2: [-9.43385176, -4.87830741]
m3: [-2.54454008, 7.53144179]

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

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

```
Within-class scatter matrix SW:  [[1052.70745046  -12.5828441 ]
 [ -12.5828441   971.29686189]]
```

SW: [[1052.70745046, -12.5828441],
[-12.5828441, 971.29686189]]

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

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

```
Between-class scatter matrix SB:  [[ 8689.12907035 16344.86572983]
 [16344.86572983 31372.93949414]]
```

SB: [[8689.12907035, 16344.86572983],
[16344.86572983, 31372.93949414]]

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

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

```
W:  [[-0.44115384]
 [-0.8974315 ]]
```

W: [[-0.44115384],
[-0.8974315]]

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?

```
# 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)
```

```
FLD using class mean, accuracy:  0.861
[[262   0  78]
 [  0 333   0]
 [ 61   0 266]]
```

accuracy score: 0.861
confusion matrix:
[[262 0 78]
[0 333 0]
[61 0 266]]

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.

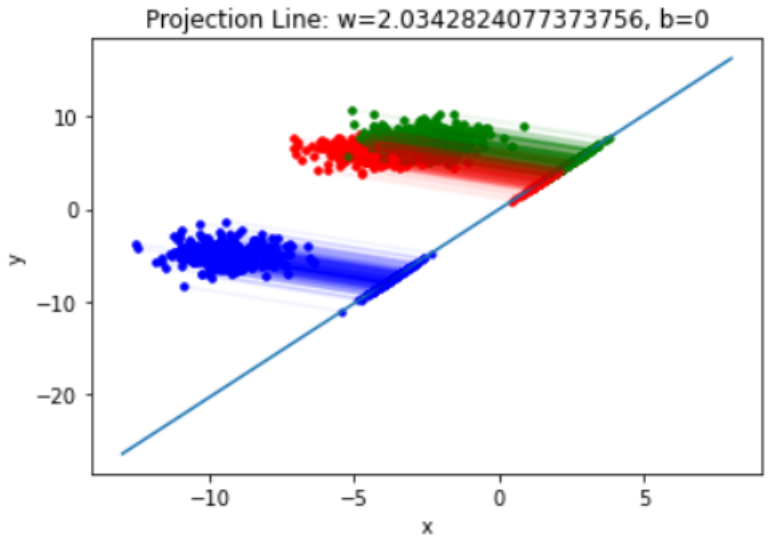
```
# For Q11
y_pred_k1 = fld.predict_using_knn(X_train, y_train, X_test, k=1)
print("FLD using knn (k=1), accuracy: ", fld.accuracy_score(y_test, y_pred_k1))

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))
```

```
FLD using knn (k=1), accuracy:  0.822
FLD using knn (k=2), accuracy:  0.819
FLD using knn (k=3), accuracy:  0.843
FLD using knn (k=4), accuracy:  0.84
FLD using knn (k=5), accuracy:  0.862
```

According to picture above, we can see that accuracy increase while k goes up. Some slightly decrease is because of some points have same number of votes, my implementation will select the one who has the smallest index.

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.

```
# For Q12, using only training data
fld.plot_projection(X_train, y_train)
```



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.

I preprocessed the data by applying L2 norm and implementing a quantile transformer using Numpy. I fitted the transformer with the training data and used it to transform the validation and testing data. Implementation code below:

```
class QuantileTransformer:
    def __init__(self, n_quantiles=100):
        self.n_quantiles = n_quantiles

    def fit(self, X):
        self.quantiles_ = np.linspace(0, 1, self.n_quantiles)
        self.references_ = np.quantile(X, self.quantiles_)
        return self

    def transform(self, X):
        X_transformed = np.interp(X, self.references_, self.quantiles_)
        return X_transformed

    def fit_transform(self, X):
        return self.fit(X).transform(X)

def l2_norm(x):
    return x / np.sqrt(np.sum(x * x))


x_train = np.array([l2_norm(x_train[i]) for i in range(len(x_train))])
x_val = np.array([l2_norm(x_val[i]) for i in range(len(x_val))])
x_test = np.array([l2_norm(x_test[i]) for i in range(len(x_test))])

quantile_transformer = QuantileTransformer(500)
x_train = quantile_transformer.fit_transform(x_train)
x_val = quantile_transformer.transform(x_val)
x_test = quantile_transformer.transform(x_test)
```

By applying these non-linear transformations to data, we can get better performance. I applied FLD with knn. The accuracy with different k is shown below. We can see that with k=40 and k=50, the validation accuracy exceeds 90%.

```python
# Try FLD
my_fld = FLD(3)
my_fld.fit(x_train, y_train)

y_pred = my_fld.predict_using_class_mean(x_train, y_train, x_val)
print("FLD using class mean, accuracy: ", my_fld.accuracy_score(y_val, y_pred))

y_pred_k1 = my_fld.predict_using_knn(x_train, y_train, x_val, k=1)
print("FLD using knn (k=1), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k1))

y_pred_k2 = my_fld.predict_using_knn(x_train, y_train, x_val, k=2)
print("FLD using knn (k=2), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k2))

y_pred_k3 = my_fld.predict_using_knn(x_train, y_train, x_val, k=3)
print("FLD using knn (k=3), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k3))

y_pred_k4 = my_fld.predict_using_knn(x_train, y_train, x_val, k=4)
print("FLD using knn (k=4), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k4))

y_pred_k5 = my_fld.predict_using_knn(x_train, y_train, x_val, k=5)
print("FLD using knn (k=5), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k5))

y_pred_k10 = my_fld.predict_using_knn(x_train, y_train, x_val, k=10)
print("FLD using knn (k=10), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k10))

y_pred_k20 = my_fld.predict_using_knn(x_train, y_train, x_val, k=20)
print("FLD using knn (k=20), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k20))

y_pred_k30 = my_fld.predict_using_knn(x_train, y_train, x_val, k=30)
print("FLD using knn (k=30), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k30))

y_pred_k40 = my_fld.predict_using_knn(x_train, y_train, x_val, k=40)
print("FLD using knn (k=40), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k40))

y_pred_k50 = my_fld.predict_using_knn(x_train, y_train, x_val, k=50)
print("FLD using knn (k=50), accuracy: ", my_fld.accuracy_score(y_val, y_pred_k50))
```

```
FLD using class mean, accuracy:  0.8780821917808219
FLD using knn (k=1), accuracy:  0.8438356164383561
FLD using knn (k=2), accuracy:  0.8493150684931506
FLD using knn (k=3), accuracy:  0.873972602739726
FLD using knn (k=4), accuracy:  0.8780821917808219
FLD using knn (k=5), accuracy:  0.8794520547945206
FLD using knn (k=10), accuracy:  0.8808219178082192
FLD using knn (k=20), accuracy:  0.8890410958904109
FLD using knn (k=30), accuracy:  0.8945205479452055
FLD using knn (k=40), accuracy:  0.9054794520547945
FLD using knn (k=50), accuracy:  0.9041095890410958
```

Part. 2, Questions (30%):

(6%) 1. Discuss and analyze the performance.

a) between Q10 and Q11, which approach is more suitable for this dataset. Why?

Based on the performance shown above, I believe that knn is more suitable for this dataset. When an appropriate k value is selected, the performance of k-NN exceeds that of the class mean method.

b) between different values of k in Q11. (Which is better, a larger or smaller k? Does this always hold?)

A smaller value of k can improve the sensitivity of classification and the expressive power of local features, but it may also lead to overfitting. A larger value of k can reduce the risk of overfitting, but it may also lead to underfitting, which means the inability to capture the features of the data effectively. So it is important to select an appropriate k, not always larger or smaller.

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

Apparently, their function is different.

sigmoid(x) = 1 / (1 + exp(-x))

softmax(x_i) = exp(x_i) / sum(exp(x_j))

Secondly, the sigmoid function is often used in binary classification problems, while the softmax function is typically used in multiclass classification problems. The sigmoid function outputs a value between 0 and 1, while the softmax function outputs a set of probabilities that sum up to 1.

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

Because in classification tasks, we measure the difference between the predicted probability distribution and the true probability distribution of the classes. Cross-entropy loss is a good choice because it is sensitive to small differences between predicted and actual probabilities.

On the other hand, we measure the difference between the predicted and actual continuous values in regression tasks. MSE is appropriate for regression tasks because it gives more weight to larger errors and provides a smooth and continuous measure of the quality of predictions.

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

a. Resampling: We balance the dataset by oversampling the minority class or undersampling the majority class.
b. Cost-sensitive learning: We assign different misclassification costs to different classes. This means that the classifier is penalized more heavily for misclassifying

instances of the minority class than for misclassifying instances of the majority class. Penalized-LDA is a popular approach based on this.

(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.)

(1) $L = y \cdot \ln(\sigma(x)) + (1-y)\ln(1-\sigma(x))$

Let $a = \sigma(x) = \frac{1}{1+e^{-x}}$,

then $L = y \ln a + (1-y)\ln(1-a)$

$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial x}$ (by chain rule)

$\frac{\partial L}{\partial a} = \frac{y}{a} - \frac{1-y}{1-a}$

$\frac{\partial a}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = a(1-a)$

Then $\frac{\partial L}{\partial x} = \left(\frac{y}{a} - \frac{1-y}{1-a}\right) \cdot a(1-a) = y(1-a) - a(1-y)$

$= y - ay - a + ay = y - a = y - \frac{1}{1+e^{-x}}$ ✱

(2) $L = (y - \sigma(x))^2$

Let $a = \sigma(x) = \frac{1}{1+e^{-x}}$

then $L = (y-a)^2$

$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial x}$ (by chain rule)

$= 2(a-y) \cdot a(1-a)$

$= 2\left(\frac{1}{1+e^{-x}} - y\right)\left(\frac{e^{-x}}{(1+e^{-x})^2}\right)$ ✱