NYCU Pattern Recognition, Homework 2

109550090 李以恩

Part. 1, Coding (70%):

In this coding assignment, you are requested to implement 1) Logistic Regression and 2) Fisher's Linear Discriminant by using only Numpy, then train your model on the provided dataset and finally evaluate the performance on testing data. Please train your logistic regression model using Gradient Descent, not the closed-form solution.

(20%) Logistic Regression Model

Requirements:

- Use Gradient Descent
- Use CE (Cross-Entropy) as your loss function.
- Use Softmax for this multiclass classification task.

Criteria:

1. (0%) Show the learning rate, epoch, and batch size that you used.

```
logistic_reg.fit(X_train, y_train, batch_size=16, lr=0.01, n_epoch=1000)
```

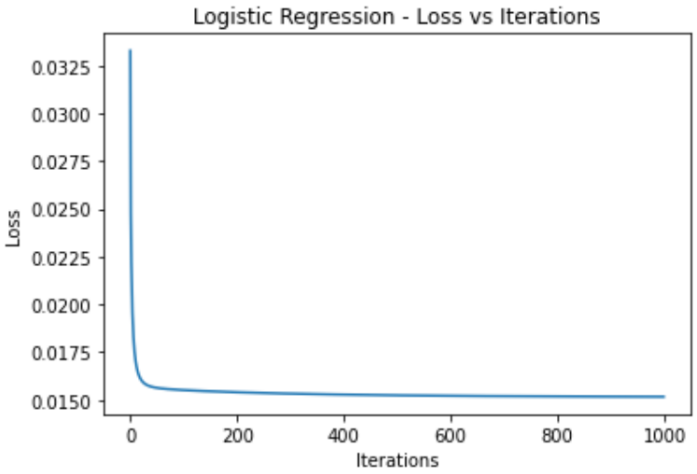
2. (5%) What's your training accuracy?

```
Training acc:  0.897
```

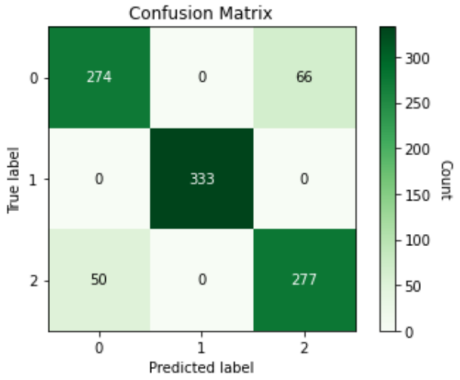
3. (5%) What's your testing accuracy?

```
Testing acc:  0.884
```

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



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



(30%) Fisher's Linear Discriminant (FLD) Model

Requirements:

- Use FLD to reduce the dimension of the data from 2 to 1.

Criteria:

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

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

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

```
[[1052.70745046  -12.5828441 ]
[ -12.5828441   971.29686189]]
```

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

```
[[ 8689.12907035 16344.86572983]
[16344.86572983 31372.93949414]]
```

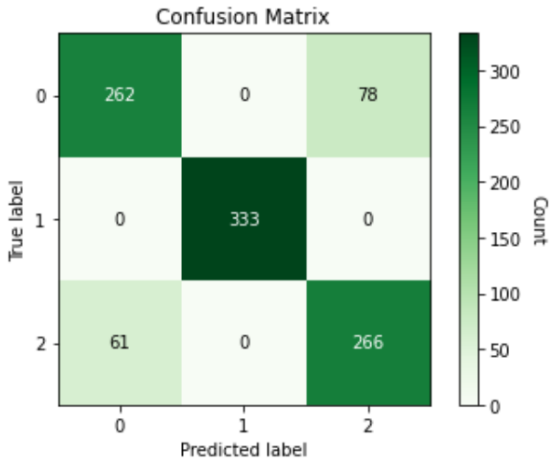
9. (4%) Compute the Fisher's linear discriminant w on training data.

```
[[-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.

```
## Predict_using_class_mean
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
```



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.

```
## Predict_using_knn
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
```

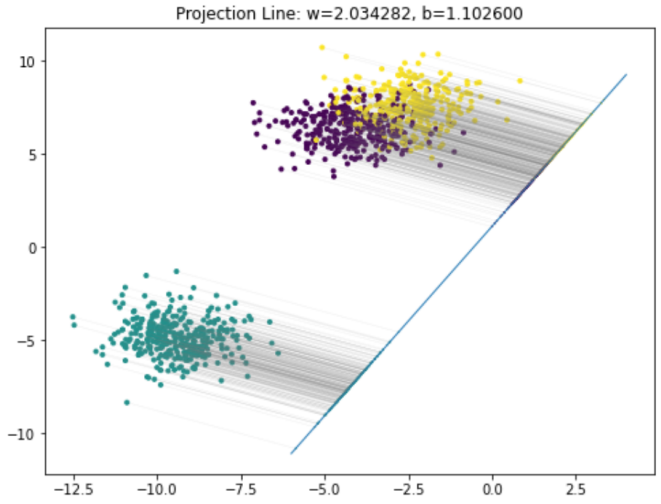
12. (4%)

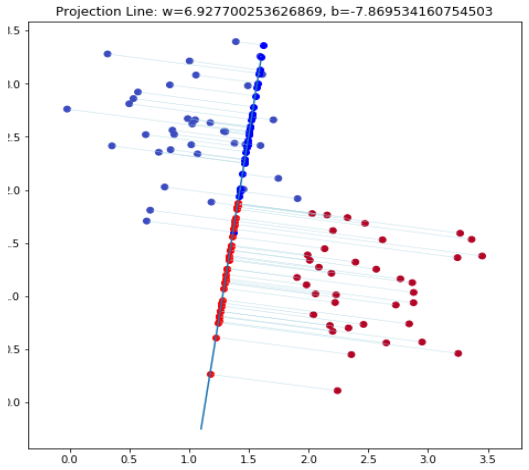
1) Plot the best projection line on the training data and show the slope and intercept on the title *(you can choose any value of intercept for better visualization)*

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)

```
1 fld.plot_projection(X_train, y_train)
```





(20%) Train your own model

Requirements:

- Using another dataset that we provided (a real-world dataset).
- Train your model (FLD or Logistics Regression model that you implemented above).
- Try different parameters and feature engineering to beat the baseline.
- Save your testing predictions in the CSV file.

Criteria:

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

I. Training Data Analysis

a. Statistic info

We can observe that the type of 'Target' attribute is float, and we may want to convert it to int. Also we can see that the range of the feature values differs greatly and requires standardization (Feature1,2 between Feature3,4).

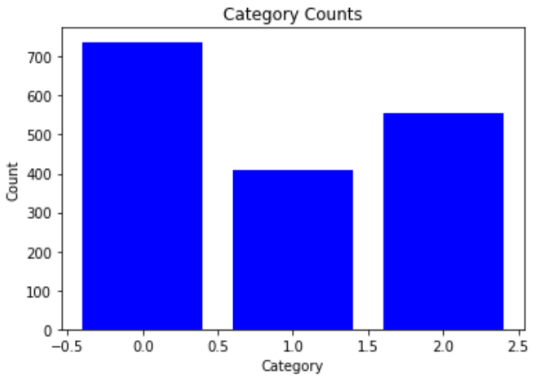
```python
df_train = pd.DataFrame(pd.read_csv("./PR_HW2_train.csv"))
df_val   = pd.DataFrame(pd.read_csv("./PR_HW2_val.csv"))
df_test  = pd.DataFrame(pd.read_csv("./PR_HW2_test.csv"))
print(df_train.head(), "\n")
print(df_train.info(), "\n")
print(df_train.describe())
print("--------------------------------------------------------------------------")
```

```
   Feature1  Feature2  Feature3  Feature4  Target
0   0.00668   0.00192     0.682     0.996     2.0
1   0.00680   0.00106     0.503     0.996     1.0
2   0.00742   0.00106     0.482     0.991     1.0
3   0.00685   0.00178     0.650     0.998     2.0
4   0.00680   0.00163     0.623     0.996     2.0

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1703 entries, 0 to 1702
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Feature1  1703 non-null   float64
 1   Feature2  1703 non-null   float64
 2   Feature3  1703 non-null   float64
 3   Feature4  1703 non-null   float64
 4   Target    1703 non-null   float64
dtypes: float64(5)
memory usage: 66.6 KB
None

          Feature1     Feature2     Feature3     Feature4       Target
count  1703.000000  1703.000000  1703.000000  1703.000000  1703.000000
mean      0.007234     0.001741     0.617571     0.995216     0.893130
std       0.000672     0.000503     0.082357     0.004149     0.865364
min       0.005410     0.000769     0.417000     0.962000     0.000000
25%       0.006720     0.001350     0.567000     0.994000     0.000000
50%       0.007110     0.001780     0.639000     0.996000     1.000000
75%       0.007675     0.002100     0.674000     0.998000     2.000000
max       0.009820     0.003150     0.811000     1.000000     2.000000
--------------------------------------------------------------------------
```

b. Category counts

We can notice that there exists an imbalance between different classes.

```python
data = df_train
counts = data['Target'].value_counts()
fig, ax = plt.subplots()
ax.bar(counts.index, counts.values, color='blue')
ax.set_xlabel('Category')
ax.set_ylabel('Count')
ax.set_title('Category Counts')
plt.show()
```



c. Correlation coefficient

The correlation coefficient is typically used to measure the linear relationship between two continuous variables, while in multiclass classification problems, the target variable is often a discrete categorical variable. Therefore, the correlation coefficient may not capture the nonlinear relationship between the target variable and the features and thus we can't use correlation coefficients to analyze the relationship between target and each feature, just like we do in homework1. However, we can notice that there exists a strong positive correlation between 'Feature 2' and 'Feature 3'.

```
corr = data.corr()
print(corr)
```

```
           Feature1  Feature2  Feature3  Feature4    Target
Feature1   1.000000  0.532584  0.224169  0.214851 -0.657016
Feature2   0.532584  1.000000  0.938422  0.431155 -0.477465
Feature3   0.224169  0.938422  1.000000  0.421574 -0.265032
Feature4   0.214851  0.431155  0.421574  1.000000 -0.215316
Target    -0.657016 -0.477465 -0.265032 -0.215316  1.000000
```

II. Feature Engineering

```
def Feature_engineering(data):
    data = (data - data.mean()) / data.std()
    data['11'] = data['Feature1'] * data['Feature1']
    data['22'] = data['Feature2'] * data['Feature2']
    data['33'] = data['Feature3'] * data['Feature3']
    data['44'] = data['Feature4'] * data['Feature4']
    data['12'] = data['Feature1'] * data['Feature2']
    data['13'] = data['Feature1'] * data['Feature3']
    data['14'] = data['Feature1'] * data['Feature4']
    data['23'] = data['Feature2'] * data['Feature3']
    data['24'] = data['Feature2'] * data['Feature4']
    data['34'] = data['Feature3'] * data['Feature4']
    data['feature1_bin'] = pd.cut(data['Feature1'], bins=5, labels=False)
    data['feature2_bin'] = pd.cut(data['Feature2'], bins=5, labels=False)
    data['feature3_bin'] = pd.cut(data['Feature3'], bins=5, labels=False)
    data['feature4_bin'] = pd.cut(data['Feature4'], bins=5, labels=False)
    data = pd.get_dummies(data)
    return data
```

a. Normalization

Since the range of the feature values differs greatly, we normalized the feature values first.

b. Generating PolynomialFeatures

Since the features we currently have can only achieve 70-85% accuracy when using FLDA or logistic regression model. In order to generate additional features of the dataset, I simulated sklearn.preprocessing.PolynomialFeatures and generated polynomial and interaction features of degree=2, including 1*1, 2*2, 3*3, 4*4, 1*2, 1*3, 1*4, 2*3, 2*4, 3*4.

c. Feature Discretization

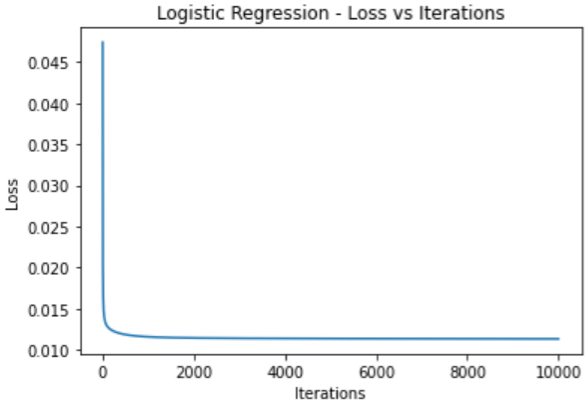
Similar to homework1, I generated additional features by splitting it into multiple new features which can compile significant interactions to our feature matrix.

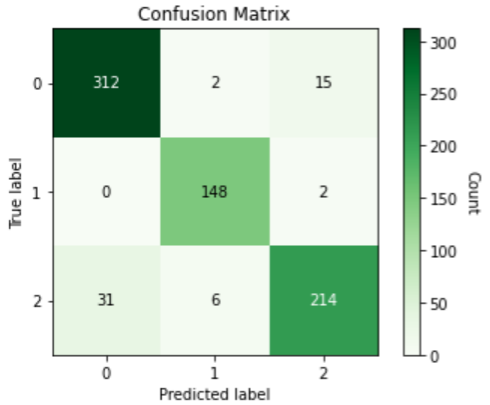
III. Model Selection

After trying logistic regression and FLDA, the performance of the logistic regression model is better. Therefore, I choose logistic regression as my final model.

```
logistic_reg = MulticlassLogisticRegression()
logistic_reg.fit(X_train, y_train, batch_size=16, lr=0.005, n_epoch=10000)
print('Training acc: ', logistic_reg.evaluate(X_train, y_train))
print('Validation acc: ', logistic_reg.evaluate(X_val, y_val))
logistic_reg.plot_curve()
logistic_reg.show_confusion_matrix(X_val, y_val)
```

```
Training acc:  0.9201409277745156
Validation acc:  0.9232876712328767
```





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?

b) between different values of k in Q11.

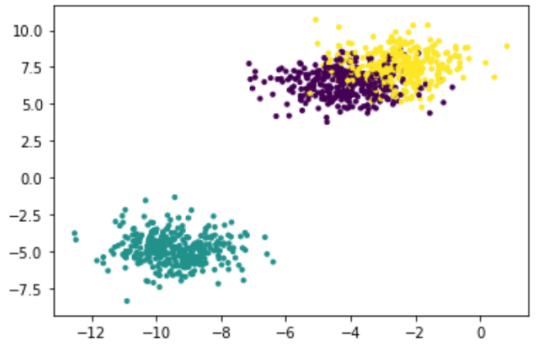
(Which is better, a larger or smaller k? Does this always hold?)

a) Predict by KNN (k=5) is more suitable since it has higher prediction accuracy (86.2%). In FLDA, we projected data to a one-dimension space. However, when the distinction between categories is not significant enough, the performance of FLDA may be affected. For this dataset and the scatter plot below, we can see that yellow points and purple points are mixing together and close to each other, which may lead to errors in the prediction. If using predict_using_class_mean( ), their means may be too close and if we only refer to the mean point it may cause error. On the other hand, if using predict_using_knn( ), though the two data distributions are similar, referencing more sample points for prediction may lead to greater accuracy since we can access more information about the data.

```
8  print("X_train", X_train.shape)
9  print("X_test",  X_test.shape)
10 print("y_train", y_train.shape)
11 print("y_test",  y_test.shape)
12
13 plt.scatter(X_train[:,0], X_train[:,1], c=y_train, marker='.')
```

```
X_train (1000, 2)
X_test (1000, 2)
y_train (1000,)
y_test (1000,)
```

```
<matplotlib.collections.PathCollection at 0x7ff765b5ceb8>
```



b) For k in range 1 to 5, the performance of the largest k (k=5) is better than the smallest k (k=1). However, this doesn't always hold. If the value of k is too large, it will increase the number of nearest neighbors, which may lead to bias in the algorithm's prediction because it may be influenced by more unrelated samples. For example, in Q11 we can see that as k increases, the accuracy doesn't always increase. (I tried k=1 to k=10.)

```
k=1 -> Accuracy of test-set 0.822
k=2 -> Accuracy of test-set 0.819
k=3 -> Accuracy of test-set 0.843
k=4 -> Accuracy of test-set 0.84
k=5 -> Accuracy of test-set 0.862
k=6 -> Accuracy of test-set 0.855
k=7 -> Accuracy of test-set 0.853
k=8 -> Accuracy of test-set 0.856
k=9 -> Accuracy of test-set 0.863
k=10 -> Accuracy of test-set 0.862
```

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

- **Sigmoid function**: **σ(x) = 1 / (1 + exp(-x))**
  The sigmoid function is a logistic function that maps any input value to a value between 0 and 1. It is often used in binary classification problems to convert a real-valued input to a probability score indicating the likelihood of the input belonging to the positive class.
- **Softmax function**: **σ(z) = exp(z) / Σ(exp(z))**
  The softmax function is a generalization of the sigmoid function that is used in multi-class classification problems to convert a vector of real-valued inputs to a probability distribution over multiple classes.

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

Cross-entropy loss is commonly used for classification tasks, where the output is discrete and categorical, and the accuracy and recall are important, because it measures the difference between predicted class probabilities and true class probabilities, which aims to optimize the goal of predicting a probability distribution over classes.

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

Mean squared error is commonly used for regression tasks, where the output is continuous and numerical, and the mean absolute error and coefficient of determination are important, because it measures the difference between predicted continuous values and true continuous values, which aims to optimize the goal of predicting a continuous variable.

$$MSE = \frac{1}{n}\Sigma\left(y - \widehat{y}\right)^2$$

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

When dealing with an imbalanced dataset, we can try the following methods to improve FLDA's performance.

- Data resampling: Try oversampling the minority class or undersampling the majority class.
- Regularization: Regularization can be used to prevent overfitting and reduce the bias towards the majority class.
- Class weighting: We can try to assign different weights to each class during training which can be done by setting higher weights for the minority class and lower weights for the majority class.

- Data augmentation: Generate new samples for the minority class to increase the size of the dataset, which can be done by applying various transformations to the existing samples, such as rotations, translations, or flipping.

(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.)

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

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

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

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

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

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

$= y \frac{1}{\sigma(x)} \sigma'(x) + (1-y) \frac{-1}{1-\sigma(x)} \sigma'(x)$

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

$= y(1-\sigma(x)) - (1-y)\sigma(x)$

$= y \frac{e^{-x}}{1+e^{-x}} - (1-y) \frac{1}{1+e^{-x}}$

$= \frac{y e^{-x} + y - 1}{1+e^{-x}}$ #

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

$= -2(y-\sigma(x)) \cdot \sigma'(x)$

$= -2(y-\sigma(x)) \cdot \sigma(x)(1-\sigma(x))$

$= \left[-2y + 2\sigma(x)\right]\left[\sigma(x) - \sigma^2(x)\right]$

$= -2y\sigma(x) + 2y\sigma^2(x) + 2\sigma^2(x) - 2\sigma^3(x)$

$= \frac{-2y}{1+e^{-x}} + \frac{2y+2}{(1+e^{-x})^2} - \frac{2}{(1+e^{-x})^3}$ #