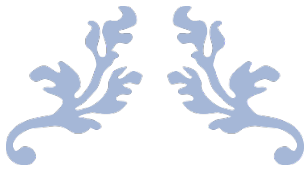

CS5014 PRACTICAL 1

200010781

MARCH 2, 2022

Contents

Part 1 Data Processing

(a) How did you load and clean the data, and why?

Read_csv() provided by pandas helps load the data we want. Since there is no header in the provided csv file, I use header=None parameter to load the data properly.

There are 32 columns in the dataset. According to the spec, the features locate in the 2 to 13 columns, and the target locates at the 30 columns. Therefore, other 19 columns are not helpful in the training process, and they should be dropped. I use drop() method to delete the first index column, 14-29 columns and the last two columns.

I checked the data with missing values and duplicated rows. Since there are not these issues, I did not take actions to deal with them.

Before training the model, I checked the distribution of data classification as shown in Figure 1. It is clear that most samples belong to CL6 class, about 600. The second most is the CL0 category, about 430. Besides, there are only 100 samples belong to CL4 category.

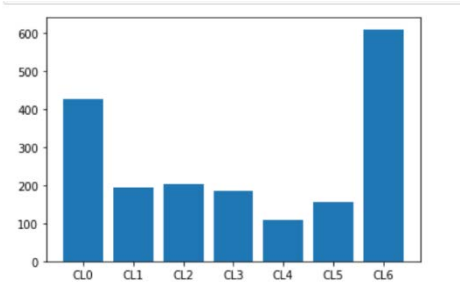

Figure 1 Class distribution of the data

I also use pairplot to show the data distribution of each pair of features, as shown in Figure 2. Different colours represent different categories.

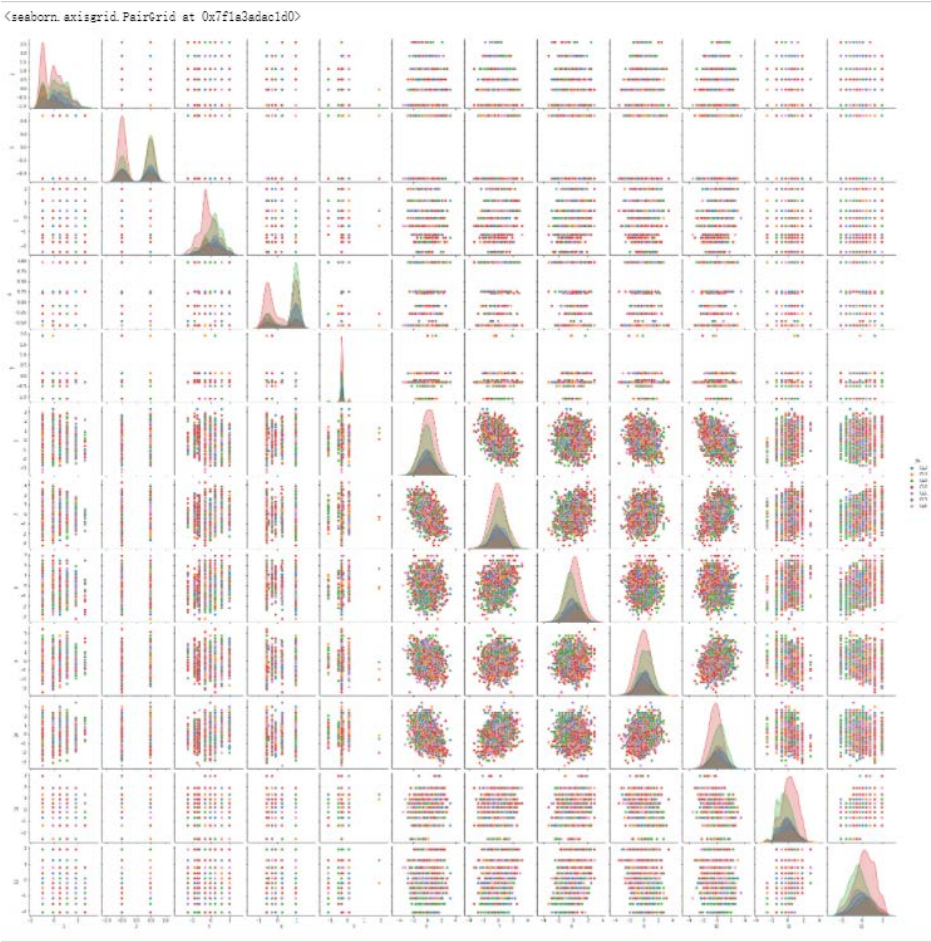

Figure 2 Pairplot of the data

(b) How did you split the data into test/train set and why?

I use train_test_split() to split data into train, validation, and test data. At first, I use parameters test_size= 0.2, random_state=0 and stratify=outputData to split training and testing data. Test_size = 0.2 means that there are 20% test samples among all samples. Random_state=0 means I can get different samples when repeating the experiment. Stratify = outputData means follow the target class frequencies in the test and train dataset. Then I use test_size= 0.25 to split 20% of training data as validation set. The final shape of each set can be seen as Figure 3 below. The number of testing and validation data are the same, stand for 377 samples. There are 1131 samples as training data with 12 features, which is three times of other two sets.

```
x_train shape: (1131, 12)
x_test shape: (377, 12)
x_val shape: (377, 12)
y_train shape: (1131,)
y_test shape: (377,)
y_val shape: (377,)
```

Figure 3 Shape of each set

(c) How did you process the data including encoding, conversion, and scaling, and why?

I use df.types() and df.isnull().sum() to check types of each columns and the number of NaN in the data set. The result shows that all features are float type and there is no NaN value in the dataset. Therefore, I did nothing for encoding and conversion.

I use StandardScaler() for feature scaling. At first, I employ fit() on training data, and then I transform training, validation and testing data using the same object.

(d) How did you ensure that there is no data leakage?

Firstly, I split the data into training, testing and validation data after data cleaning. Secondly, I only fit training data when scaling and not using testing data in this process. Lastly, all training processes are running on training. I only use validation data to check the performance before final evaluation where testing data is used. Therefore, I did not use any information in the testing data in the training process.

Part 2 Training

(a) Train using penalty='none' and class_weight=None. What is the best and worst classification accuracy you can expect and why?

Parameter Class_weight can be used in unbalanced classification problem. Class_weight can be used to increase the weight of small samples classification, which makes the model more sensible.

In this practical, when I use Class_weight=None, I get about 39.70% of accuracy score for training set, but slightly lower in validation set, about 39.25%. For testing set, I get the similar accuracy score, about 40.5% (all scores are shown in Figure 4).

```
Model for training set with None class_weight
Accuracy score:  0.39699381078691426
Model for validation set with None class_weight
Accuracy score:  0.3925729442970822
Model for testing set with None class_weight
Accuracy score:  0.40583554376657827
```

Figure 4 Accuracy scores on various sets with 'None' class_weight

The worst situation is having overfitting issue, which means we get good accuracy on training set, but worse on testing set.

I assume the reason for getting less than 50% accuracy score is because we use linear model, which is not quite suitable for the dataset we use.

(b) Explain each of the following parameters used by LogisticRegression in your own words: penalty, tol, max_iter.

Penalty is a form of regularization. The default penalty is 'l2'. It helps model to deal with overfitting issue and handle errors.

Tol refers to the condition when the model stops. It can be used for the criteria of stopping. The default tol is 1e-4, which means we stop the computation when reaching 1e-4. It helps model to handle noises.

Max_iter represents the maximum iteration of the model to converge. The gradient descent happens step by step. Therefore, this parameter is used to define when to stop the gradient descent process.

(c) Train using balanced class weights (setting class weight='balanced'). What does this do and why is it useful?

Parameter class_weight considers the weight of each classification. When setting it as None, model do not consider the classification weight. Class_weight can be used in highly unbalanced samples or crucial classification problem. In unbalanced data, for example, only 5 belongs to A class and the rest belongs to B class among 10000 samples, class_weight can be used to increase the weight of class A. In crucial classification problem, for example, classifying legal users and illegal users, we tend to classify legal users as illegal users instead of recognizing illegal users as legal users. Class_weight can be used to increase the weight of illegal users.

Class_weight=balanced means the model calculates weight of each classification using formula below.

$$\frac{n\_samples}{n\_classes * np.bincount(y)}$$

$$where\ n\_samples\ refers\ to\ the\ number\ of\ samples, n\_calsses\ refers\ to\ the\ number\ of\ classification, and\ np.bincount(y)\ is\ the\ number\ of\ samples\ in\ each\ class.$$

When I use Class_weight=balanced, I get 29.53% of accuracy score for training set, but slightly lower in validation set, about 26.52%. For testing set, I get about 25% of accuracy score. (all scores are shown in Figure 5).

```
Model for training set with balanced class_weight
Accuracy score:  0.2953138815207781
```

```
Model for validation set with balanced class_weight
Accuracy score:  0.26525198938992045
```

```
Model for testing set with balanced class_weight
Accuracy score:  0.2493368700265252
```

Figure 5 Accuracy scores on training and validation set with 'balanced' class_weight

For this practical, when using the balanced class_weight, the accuracy socre is lower than using class_weight=none. I assume this is because the dataset we use is not quite unbalanced, which decreases the accuracy score.

(d) LogisticRegression provides three functions to obtain classification results. Explain the relationship between the vectors returned by predict(), decision_function(), and predict proba().

Predict(): This gets the final predicted output label of a sample.

Predict_proba(): This returns the probabilities of outputs in each class. The class with highest probability represents the prediction result.

Decision_function(): According to the document of scikit-learn, this function gets the confidence score, that is the signed distance of a sample to the hyperplane.

The functional form of logistic regression is

$$f(x) = \frac{1}{1 + e^{-(\beta 0 + \beta 1x1 + \cdots \ldots + \beta nxn)}}$$

For predict_proba(), it returns f(x). For predict(), it returns a class according to the rule of $f(x) > 0.5$. For decision_function, it returns $d(x) = \beta 0 + \beta 1x1 + \cdots \ldots + \beta nxn$. Hyperplane is $\beta 0 + \beta 1x1 + \cdots \ldots + \beta nxn = 0$

Part 3 Evaluation

(a) What is the classification accuracy of your model and how is it calculated? Give the formula.

When using "penalty='none',class_weight=None" to train the LogisticRegression model, I get about 40% accuracy score in testing data. When using "penalty='none',class_weight=balanced" to train the model, I get about 25% accuracy score in testing set. They are being calculated according to the following formula:

$$Accuracy\ Score = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP refers to the true positive, TN means true negative, FP represents false positive, and FN is false negative.

(b) What is the balanced accuracy of your model and how is it calculated? Give the formula.

When using "penalty='none', class_weight=None" to train the LogisticRegression model, I get 22.25% balanced accuracy score for testing set. When using "penalty = 'none', class_weight = balanced" to train the model, I get higher balanced accuracy score, about 26.73%. These are being calculated according to the following formula:

$$Balanced\ Accuracy = \frac{1}{2} \times (\frac{TP}{TP + FN} + \frac{TN}{TN + FP})$$

also known as

$$Balanced\ Accuracy = \frac{1}{2} \times (Sensitivity + Specificity)$$

Balanced accuracy can be used as a metrics for imbalanced data. It accounts for both the positive and negative outcome classes and does not mislead with imbalanced data.

(c) Show the confusion matrix for your classifier for both unbalanced (2a) and balanced (2b) cases. Discuss any differences.

For (2a) using unbalanced parameter, the confusion matrix for testing set can be seen as Figure 6 below. There are about 50 and 95 samples to be predicted correctly for CL0 and CL6 level. Other levels from CL1 to CL5 have little correct prediction. Besides, lots of samples tend to be predicted as CL0 or CL6. I assume this is because the numbers of samples with CL6 and CL0

```
confusion_matrix on testing set:
 [[51  7  1  0  0  1 26]
 [17  6  0  0  0  0 16]
 [17  2  0  1  0  0 21]
 [ 7  1  0  0  0  0 29]
 [ 6  1  0  1  0  0 13]
 [ 7  0  0  0  0  1 23]
```

Figure 6 Confusion_matrix on testing set using unbalanced parameter

label are the highest two among all classes.

For (2b) using balanced parameter, the confusion matrix for testing set is shown as Figure 7 below. Different from using class_weight=none, there is no 0 shown in the diagonal of matrix, which means more data tends to be predicted correctly. This shows that the class_weight='balanced' can somehow improve the accuracy of classification with less samples number.

```
confusion_matrix on testing set:
 [[26 27  8  4 10  6  5]
 [10 13  4  3  2  1  6]
 [ 8  8  6  7  4  3  5]
 [ 4  3  3  7  8  5  7]
 [ 0  3  1  1  8  8  0]
 [ 2  2  2  3  8 10  4]
 [15 18  8 14 20 23 24]]
```

Figure 7 Confusion_matrix on testing set using balanced parameter

(d) Show the precision and recall of your algorithm for each class, as well as the micro and macro averages. Explain the difference between the micro and macro averages.

The formula of precision_score is $\frac{TP}{TP+FP}$ .The formula of recall score is $\frac{TP}{TP+FN}$.

Using unbalanced parameter, the precision score and recall score with average of none, micro or macro can be seen as Figure 8 below. Using average=None, classes CL0 and CL6 have the highest precision scores and recall scores, while the classes of CL2 to CL4 all have 0 precision and recall scores. Precision score and recall score with micro averages are the same, about 40.58%. Precision score with macro average 20.91%, while recall score with macro average is 22.25%.

```
precision_score(average=None):
 [0.4047619  0.3        0.         0.         0.         0.33333333
 0.42600897]
precision_score(average=micro):
 0.40583554376657827
precision_score(average=macro):
 0.20915774381501478
recall_score(average=None):
 [0.59302326 0.15384615 0.         0.         0.         0.03225806
 0.77868852]
recall_score(average=micro):
 0.40583554376657827
recall_score(average=macro):
 0.22254514268091435
```

Figure 9 Precision and recall scores using unbalanced parameter

```
precision_score(average=None):
 [0.4        0.17567568 0.1875     0.17948718 0.13333333 0.17857143
 0.47058824]
precision_score(average=micro):
 0.2493368700265252
precision_score(average=macro):
 0.24645083605167642
recall_score(average=None):
 [0.30232558 0.33333333 0.14634146 0.18918919 0.38095238 0.32258065
 0.19672131]
recall_score(average=micro):
 0.2493368700265252
recall_score(average=macro):
 0.2673491292745124
```

Figure 8 Precision and recall scores using balanced parameter

Scores become different when using balanced parameter, which is shown in Figure 9. Using average=None, class CL0 and CL6 also have the highest precision scores. But for recall scores, class CL4 has the highest percentages. Different from previous result, there is no 0 scores for all classes. Precision score and recall score with micro averages are still the same, about 24.93%. Scores for both precision and recall with macro average are slightly higher than using unbalanced parameter. Precision score with macro average 24.64%, while recall score with macro average is 26.73%.

According to the documentations, using average='micro' means calculating metrics globally by counting the total true positives, false negatives, and false positives. This means it focuses the contributions of all classes to compute the average metric, which is the reason why we can get the same result for precision and recall scores. Besides, I find that micro average gets the same result of accuracy score.

Macro average calculates score of each class independently, and then take the average. It treats all classes equally. In this assignment, when calculating the precision score with macro average, it will calculate the precision score of all 7 classes, as shown in precision_score (average=None), and then take the average of all these 7 scores.

Part 4 Advanced Tasks

(a) Set the penalty parameter in LogisticRegression to 'l2'. Give the equation of the cost function used by LogisticRegression as the result. Derive the gradient of this l2- regularised cost.

The loss function of LogisticRegression is

$$J(\theta) = -lnL(\theta) = \sum_{1}^{n} -[y^n lnf(x^n) + (1 - y^n)\ln\,(1 - f(x^n))]$$

The target of logistic regression is to find parameter θ so that $J(\theta)$ is minimum. We can use gradient descent to minimize the loss function. Regularization is a technique to handle overfitting issue in a model by penalizing the loss function. Using 'l2' penalty will change the loss function into

$$J(\theta) = \sum_{1}^{n} -\left[y^n lnf(x^n) + (1 - y^n)\ln\left(1 - f(x^n)\right)\right] + \frac{\gamma}{2n}\sum_{i=1}^{n}\theta_i^2$$

where γ is regularization parameter. It helps to reach a balance between fitting training data well and avoiding overfitting.

Therefore, the gradient of $J(\theta)$ is

$$\frac{\partial}{\partial\theta_i}J(\theta) = \sum_{i=1}^{n}(f\left(x^i\right) - y^i)x^i + \gamma\theta_i$$

When there is a large θ, γ will heavily penalize it to avoid overfitting.[1]

(b) Implement a 2nd degree polynomial expansion on the dataset. Explain how many dimensions this produces and why.

There are four parameters of PolynomialFeatures. Degree is used to set the degree of polynomial. The default value is 2. Interaction_only is used to control whether produce polynomial that features combining themselves. For example, there is two features a and b. if we set interaction_only=True, we will not get a^2 and b^2 in the result. Include_bias is used to control whether keep the bias column. If we set it as True, we will get the column that all items are 1.

For this practical, there are 12 features. Using 2nd degree can produce 91 features, which can be counted using n_output_features_ method of PolynomialFeatures. The produced new data is shown as Figure 10 below.

```
 Polynomial Features:

[[ 1.          1.17485754  1.02779552 ...  0.04713316  0.11346972
   0.27317026]
 [ 1.          1.99456499 -0.97295617 ...  0.31056612  0.23840696
   0.18301377]
 [ 1.          1.17485754  1.02779552 ...  0.53209282  0.14936762
   0.04193007]
 ...
 [ 1.         -1.1305387  -0.97295617 ...  0.04713316 -0.17384751
   0.64122496]
 [ 1.         -1.1305387   1.02779552 ...  0.53209282  0.38125048
   0.27317026]
 [ 1.         -1.1305387  -0.97295617 ...  0.53209282 -0.58411574
   0.64122496]]
Number of New Features: 91
```

Figure 10 Polynomial expansion on the dataset

(c) Compare the results of regularised and unregularised classifiers on the expanded data and explain any differences.

After expanding dataset, I use L1 (with liblinear solver) and L2 regularisation to train the model. In order to see changes clearly, all accuracy scores are summarised in the Table 1 below. Compared with training on original data with no regularisation (as shown in figure 4), it is clear that using L1 regularised method and improve the accuracy to some extent.

Table 1 Accuracy score of new produced data with regularisation

class_Weight& regularization	**training (new)**	**validation (New)**	**Testing (New)**
L1, none	**46.77%**	**38.46%**	**36.87%**
L1, balanced	**44.56%**	**27.05%**	**25.72%**
L2, none	**47.83%**	**36.60%**	**32.89%**
L2, balanced	**40.22%**	**22.81%**	**21.48%**

Reference

[1] Wikipedia
(https://en.wikipedia.org/wiki/Regularization_(mathematics)#:~:text=In%20mathematics%2C%20statistics%2C%20and%20computer,problem%20or%20to%20prevent%20overfitting.)