COMP 4211 Programming Assignment 1

Lee Jae Yeol (20308109) | jyleeag@connect.ust.hk

Introduction

For the programming assignment 1, wine quality train & test datasets are provided, each dataset consisting of 10 feature columns and one binary label column for classification of high-quality wine (1) or low quality (0).

The main objective of pa1 is first to be familiar on data importing and preprocessing techniques, and then have hands-on experience with scikit-learn to build supervised learning models. After constructing a linear regression to evaluate the relationship between different features and density, logistic regression and single-hidden-layer neural network models will be implemented and compared for solving the classification task. At part 4, techniques such as hyperparameter tuning and oversampling will be employed for enhancement of model performance.

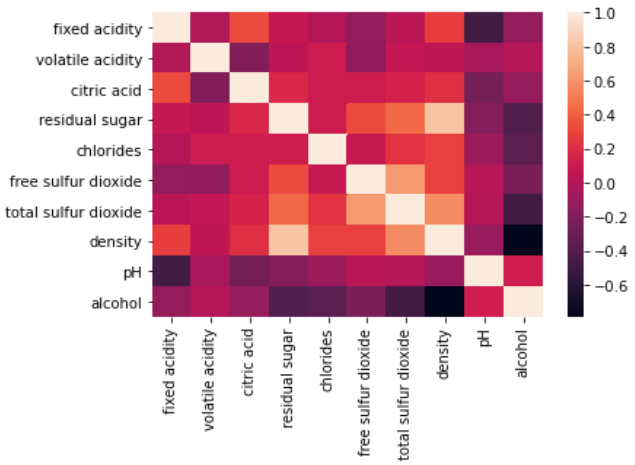
Experiment Environment

CPU	Memory	Python
Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz	8 GB	3.7.9

Part 1: Data Preprocessing

[Q1] Number of remaining records after duplicate removal and median fillna: **1617**

Fig1. heatmap of correlation among features



Part 2: Regression

[Q2]

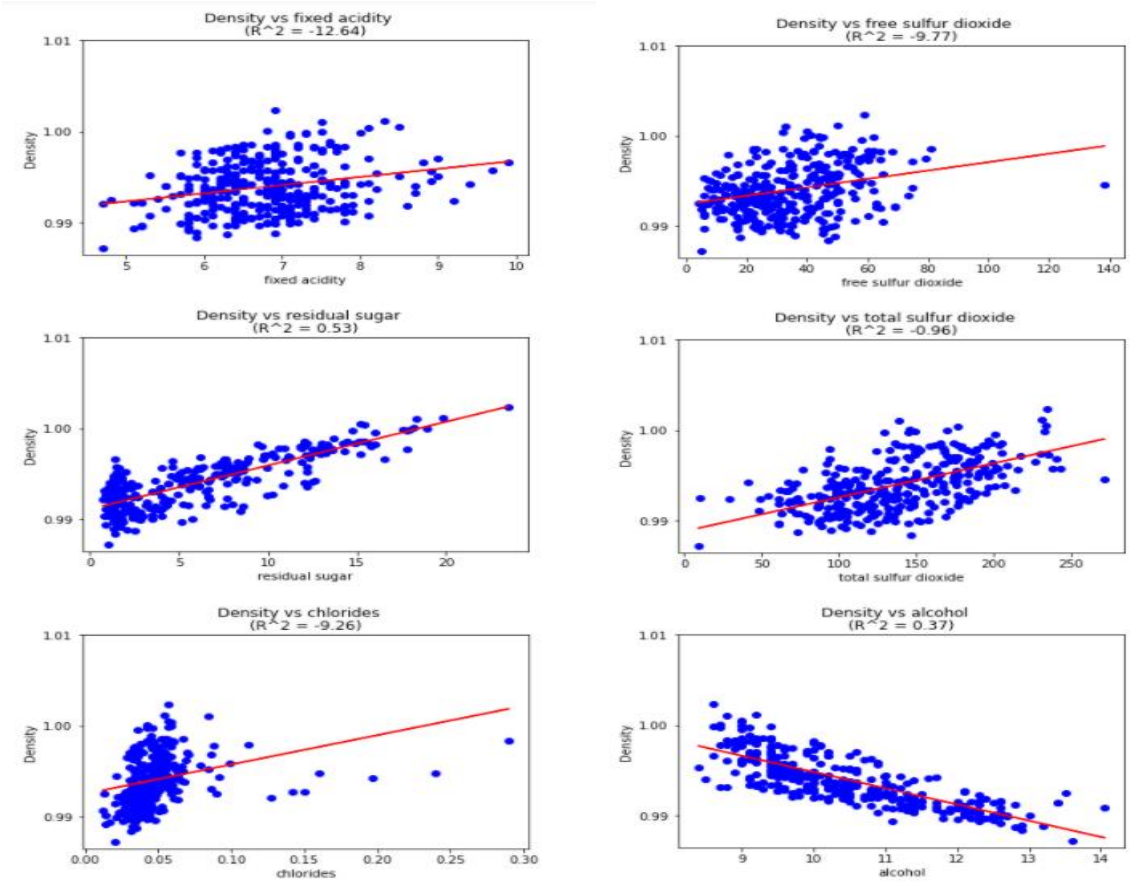
Table1. R^2 score of each features and 'density'

Features	Fixed acidity	Residual sugar	Chlorides	Free sulfur dioxide	Total sulfur dioxide	Alcohol	combination
R^2 score	-12.638	0.5275	-9.2616	-9.7749	-0.9552	0.3652	0.9333

Based on the R^2 score, the observed variation of density can be well most explained by features residual sugar, followed by alcohol. If all these features are combined (i.e. combination), it can explain 93% of the variation of density.

[Q3]

Fig2. Plot of regression line and data points of each features and 'density'



Part 3: Classification

3.1 Feature Selection

[Q4]

Table2. Chi-square score of each feature

features	Chi score
total sulfur dioxide	1315.695788
residual sugar	198.105064
alcohol	70.569785
fixed acidity	3.231314
free sulfur dioxide	2.305043
volatile acidity	2.252629
chlorides	1.160100
pH	0.292732
citric acid	0.025408
density	0.002408

For feature selection, SelectKBest function from sklearn has been used for calculating the chi-square score of each feature in the dataset. Since our objective is to drop two least unimportant features, 'citric acid' and 'density' features will not be further used for model training.

3.2 Logistic Regression

[Q5]

```
from sklearn.linear_model import SGDClassifier
lrelmodel1 = SGDClassifier(loss = 'log', penalty = 'none', random_state = 1, learning_rate = 'constant', eta0 =1, verbose = 1)
```

The baseline of the stochastic gradient descent classifier model parameters that is implemented is as above. Eta0 has been set to 1, 0.1 and 0.05, each setting reiterated 3 times with different random states. Final additional setting was implemented as learning rate value of 'adaptive' and eta0 equal to 1. The mean and standard deviation of the three random states for each setting is as below.

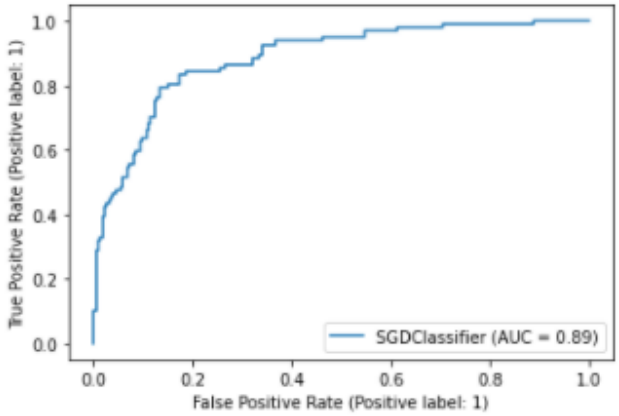
Table3. time, accuracy, f1scores mean and std for different step size

Model	eta0 = 1	eta0 = 0.1	eta0 = 0.05	Adaptive, eta0 =1
time mean	0.003325	0.003314	0.003657	0.010328
Time std	0.000577	0.000586	0.001152	0.000558
Accuracy mean	0.754115	0.812757	0.826132	**0.838477**
Accuracy std	0.057106	0.006425	0.001782	0.001782
F1 mean	0.611629	0.713424	0.725759	**0.740489**
F1 std	0.073482	0.015637	0.013321	0.003612

The model with learning rate as adaptive performed the best in terms of accuracy and f1 statistics.

[Q6]

Fig3. ROC curve using the adaptive learning rate model on validation set



AUC value = 0.89

ROC (receiver operating characteristic curve) provides a graphical illustration of how well the model can distinguish the binary class. The area under the curve, as known as AUC, represents the probability that the model will classify correctly. It also displays the tradeoff of true positive rate and false positive rate under different classification thresholds.

3.2.1 Single-hidden-layer Neural Networks

[Q7]

```
MLPClassifier(hidden_layer_sizes = (num_units,),
              solver = 'sgd', max_iter = 500,
              early_stopping = True,
              random_state = i))
```

The baseline of the single hidden layer neural network. Varying number of hidden layer units and random states are inputted reiteratively.

Table4. time, accuracy, f1scores mean and std for different values of hidden units H

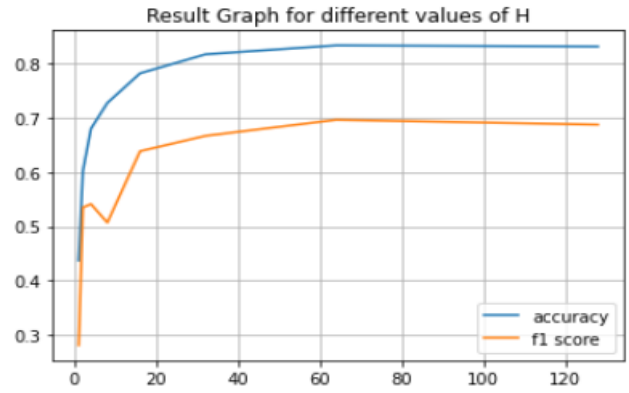
Number of hidden units	1	2	4	8	16	32	64	128
Time mean	0.0389	0.1513	0.1905	0.152924	0.2241	0.2021	0.2334	0.3461
Time std	0.0108	0.1013	0.0923	0.1069	0.0747	0.0558	0.0552	0.1725
Accuracy mean	0.4372	0.6019	0.6800	0.7274	0.7819	0.8169	**0.8333**	0.8313
Accuracy std	0.2229	0.2641	0.1560	0.0767	0.0371	0.0125	0.0193	0.0094
F1 mean	0.2812	0.5348	0.5410	0.5069	0.6385	0.6667	**0.6962**	0.6874
F1 std	0.2460	0.0747	0.0753	0.2362	0.0589	0.0436	0.0578	0.0364

[Q8]

Single hidden layer with 64 units of neural network model performed the best. Comparing with the logistic regression model, the training time has significantly increased from 0.01 sec to 0.34 second. But the accuracy and f1 score has both decreased. This represents that for our dataset, a logistic regression model outperforms a single hidden layer neural network.

[Q9]

Fig4. Accuracy and F1 score for different values of H



Accuracy refers to the measure of all correctly identified cases in classification. F1 score is calculated by the harmonic mean of precision and recall. While the accuracy for our single layer hidden NN is high (i.e. above 80%), the f1_score is bound under 70%.

This represents that while our model is good at predicting the true positives and true negatives, it still has room for enhancement for incorrectly classified cases (i.e. precision and recall).

Also, accuracy might be misleading evaluation method when the datasets are imbalanced, hence metric such as confusion matrix or F1 score might provide an objective evaluation of the model.

[Q10]

There is a trend of rapid increase in the scores as the hidden units increase, but as the number of hidden units go beyond 20, the scores increase at a very slow speed.

This observation might be due to the reason that the model is reaching the ceiling of the maximum possible training with the data set given, especially when H goes over 40 ~ 60. It is plausible that with excess number of hidden units, it could lead to overfitting.

Part 4: Performance Enhancement

4.1 Hyperparameter Tuning

[Q11]

```
from sklearn.model_selection import GridSearchCV

param_dist = {
    'hidden_layer_sizes':[(10,), (50,), (100,)],
    'learning_rate': ['constant', 'adaptive'],
    'learning_rate_init': [1e-3, 5e-2],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'sgd'],
}
# single layer neural network
slnn = MLPClassifier(random_state = 4211, early_stopping = True)
gridcv = GridSearchCV(slnn, param_grid = param_dist, cv =5, verbose = 3)
```

As shown above, there were 48 candidates of parameter combination. Out of those 48 candidates, arbitrary 10 hyperparameter settings are as follows.

1. activation=relu, hidden_layer_sizes=(10,), learning_rate=constant, learning_rate_init=0.001, solver=adam;

2. activation=relu, hidden_layer_sizes=(10,), learning_rate=constant, learning_rate_init=0.001, solver=sgd;

3. activation=relu, hidden_layer_sizes=(10,), learning_rate=constant, learning_rate_init=0.05, solver=sgd

4. activation=relu, hidden_layer_sizes=(10,), learning_rate=adaptive, learning_rate_init=0.001, solver=adam;

5. activation=tanh, hidden_layer_sizes=(10,), learning_rate=constant, learning_rate_init=0.001, solver=adam;

6. activation=tanh, hidden_layer_sizes=(10,), learning_rate=constant, learning_rate_init=0.05, solver=adam;

7. activation=tanh, hidden_layer_sizes=(10,), learning_rate=constant, learning_rate_init=0.05, solver=sgd;

8. activation=tanh, hidden_layer_sizes=(10,), learning_rate=adaptive, learning_rate_init=0.001, solver=adam;

9. ctivation=relu, hidden_layer_sizes=(50,), learning_rate=adaptive, learning_rate_init=0.001, solver=adam;

10. activation=relu, hidden_layer_sizes=(10,), learning_rate=adaptive, learning_rate_init=0.05, solver=sgd;

[Q12]

Top three hyperparameter settings

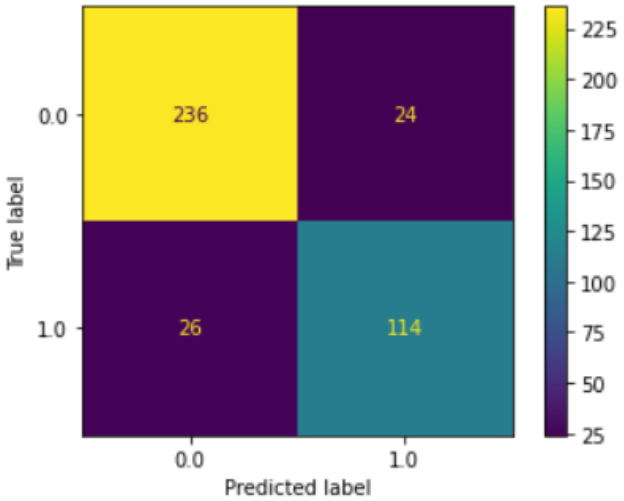
```
Model with rank: 1
Mean validation score: 0.862 (std: 0.032)
Parameters: {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'learning_rate_init': 0.05, 'solver': 'ad
am'}

Model with rank: 1
Mean validation score: 0.862 (std: 0.032)
Parameters: {'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.05, 'solver': 'ad
am'}

Model with rank: 3
Mean validation score: 0.855 (std: 0.025)
Parameters: {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'learning_rate_init': 0.05, 'solver': 'ada
m'}

Model with rank: 3
Mean validation score: 0.855 (std: 0.025)
Parameters: {'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.05, 'solver': 'ada
m'}
```

[Q13]

Accuracy, F1 score and confusion matrix of the test.csv set prediction

	precision	recall	f1-score	support
0.0	0.90	0.91	0.90	260
1.0	0.83	0.81	0.82	140
accuracy			0.88	400
macro avg	0.86	0.86	0.86	400
weighted avg	0.87	0.88	0.87	400

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x238a6cf1488>
```



4.2 Oversampling

[Q14]

Oversampling is a technique used when there exist imbalance data size problems in classification. It adds oversampled data from the minority class and rebalances the inequality.

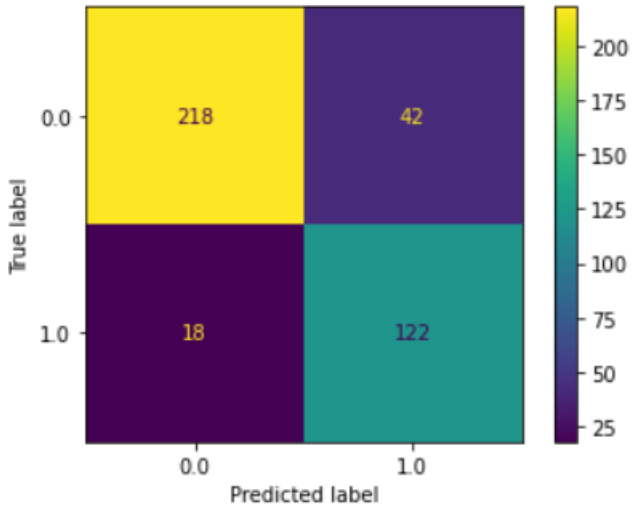
1. Instead of oversampling, we can perform under sampling. If we have sufficient data such that we can discard some, we can under sample some data from the majority class. We can perform under sampling till we have a reasonable proportion of data between classes.

2. Another well-known method for revising imbalanced class classification is to use SMOTE, short for Synthetic Minority Oversampling. SMOTE is performed by selecting k nearest neighbors of data, and then making a new sample within the colinear lines. The implementation has an advantage of synthesizing data points that are feasible.

[Q15]

	precision	recall	f1-score	support
0.0	0.92	0.84	0.88	260
1.0	0.74	0.87	0.80	140
accuracy			0.85	400
macro avg	0.83	0.85	0.84	400
weighted avg	0.86	0.85	0.85	400

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2
```



[Q16]

After performing oversampling, the number of correct predictions for class 1 has increased. However, while the score for recall has improved, the precision has decreased. We can observe that due to the effect of oversampling, our model is guessing more class 1 label than before, which simultaneously inevitably increases the number of incorrect predictions of class 1 label. Therefore, there seems to be a tradeoff between precision and recall and a decrease in F1-score.