ASSIGNMENT 1A

CAB420 Machine Learning, Semester 1 2022

Student: Dac Duy Anh Nguyen (Rodo Nguyen)

Student ID: 10603280

Table of Contents

Problem 1: Regression

1 Data

1.1 Data Exploration

The data information file states that the numeric data has been normalized into the decimal range from 0.00 to 1.00 with extreme values (outliers) being set to either 0 if more than 3 standard deviation below the mean or 1 if more than 3 standard deviation above the mean. The final goal of this dataset is predicting the percentage of violent crimes per population - ‘ViolentCrimesPerPop’.

It is acknowledged in the information file that some entries are omitted or missing. However, those columns have been removed from the dataset given in this assignment.

Correlation heatmap shows that many features have a strong relationship (as expressed by values near 1 and -1). This mean they are modelling the same feature and may be redundant (See Appendixes/ 1)

Training, validation, testing have shapes of (298, 101), (298, 101), and (299, 101) respectively. This is not ideal split ratio as training dataset should take up a large portion compared to the total dataset for the model to learn effectively and guarantees a variety of training scenarios.

1.2 Data Pre-processing

Since all values have the same scale, standardisation is not necessary in this case. And no other alteration is made to the data.

2 Models

In this assignment, sklearn.linear_model module is used to implement 3 models in Problem 1 while statsmodels.regression.linear_model.OLS is also used in Linear Regression to utilize available summary method for deeper inspection in the model.

2.1 Hyper-parameter selection

2.1.1 Linear Regression

In this Linear Regression model, the only hyper-parameter we need to consider is fit_intercept. Since our data has only been normalized, **fit_intercept will be set to True**. Additionally, as our data values range from 0 to 1, it is expected that the performance difference will be a very small portion (See Appendixes/ 2)

2.1.2 Ridge Regression

In this Ridge Regression model, 2 hyper-parameters to be selected are fit_intercept and lambda (also called as alpha).

In the first round, we trained the model in a wide range of lambda values (from 0 to 500 with interval of 2) to make sure we cover the best lambda, and then refine the interval and the search range around the best first lambda found. The best lambda is selected by the lowest RMSE that model has.

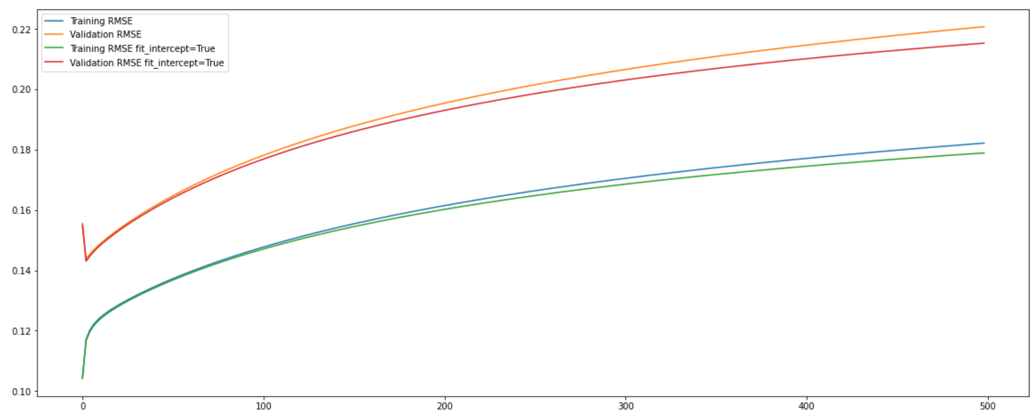


FIGURE 1 RMSE PLOT IN ROUND 1

As can be seen from the figure above, if we continue to increase lambda above 500, the validation RMSE will only get greater and the model's prediction ability will get worse. So our first search range has successfully cover the optimal lambda value.

The best lambda found through programming method is 2. Based on the same ground in 2.1.1 Linear Regression, **fit_intercept** is set to **True** and it indeeds produces slightly lower validation RMSE compared to model with fit_intercept=False.

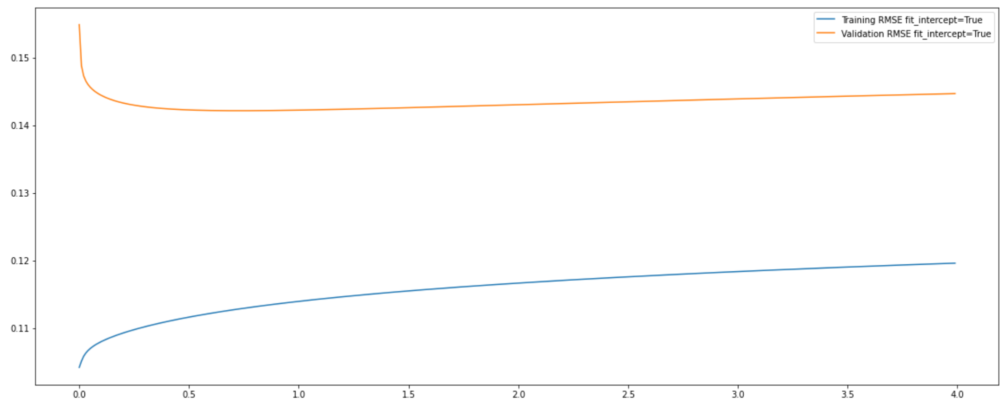


FIGURE 2 RMSE PLOT IN ROUND 2

The second round with lambda ranges from 0 to 4 with interval of 0.01 reveals the **best lambda of 0.73** through the same evaluation method.

2.1.3 Lasso Regression

One of the Lasso hyper-parameters – fit_intercept – is set to True due to the same reason discussed in previous models.

For lambda hyper-parameter, the process of selecting the best lambda is similar to Ridge Regression with the first range from 0-100 with interval of 1. The best lamda associated with smallest validation RMSE is 0. The RMSE value also converges at the top of the graph so we can be certain that the best lambda is within this range.

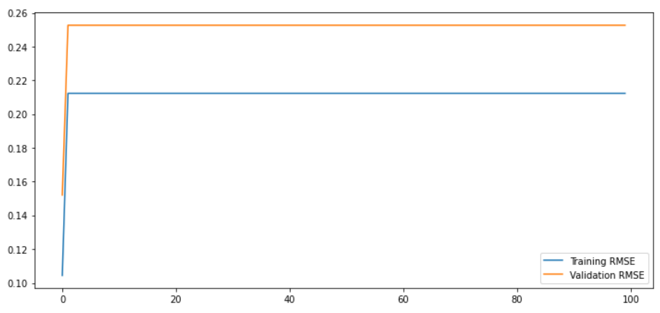


FIGURE 3 RMSE RESULT IN ROUND 1

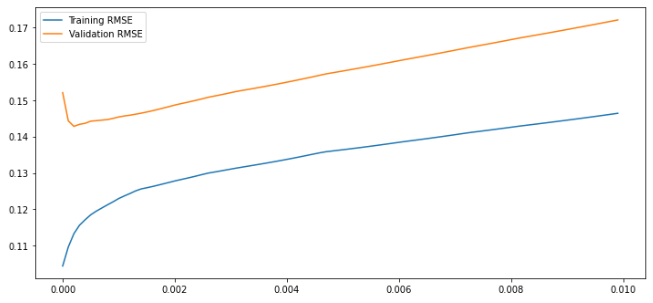


FIGURE 4 RMSE RESULT IN ROUND 3

The second range I examined is from 0-1 with interval of 0.01. The best lambda found still 0. Therefore, I continue to shrink down the interval as well as the range.

The third and final round is from 0-0.01 with interval of 0.0001. The best lambda found and chosen for testing is 0.0002.

2.2 Comparision & Evaluation

2.2.1 Prediction results and other metrics

RMSE_train	0.1042
RMSE_val	0.1549
RMSE_test	0.1533
R_squared	0.7591
Predictors with p value > 0.05	94 out of 100

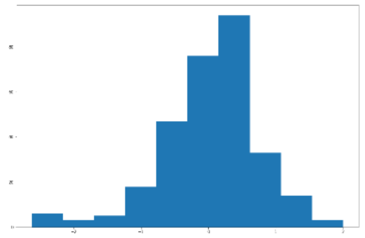


TABLE 1 LINEAR REGRESSION MODEL'S RESULTS AND OTHER METRICS

Best Lambda	0.73
RMSE_train	0.1129
RMSE_val	0.1421
RMSE_test	0.13
R_squared	0.7174

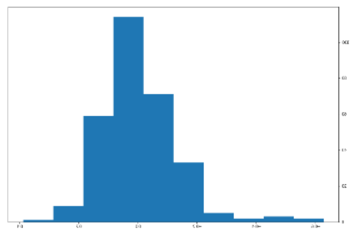


TABLE 2 RIDGE REGRESSION MODEL'S RESULTS AND OTHER METRICS

Best Lambda	0.0006
RMSE_train	0.1195
RMSE_val	0.145
RMSE_test	0.1294
R_squared	0.683
Number of Coefficients = 0	59 out of 100

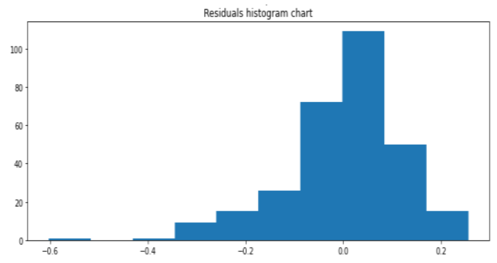


TABLE 3 LASSO REGRESSION MODEL'S RESULTS AND OTHER METRICS

2.2.2 Models Evaluation and Comparison

Looking at the RMSE values in 3 datasets for train, validation, and test dataset, their predictions have been pretty good with RMSEs of ~0.13 in Ridge and Lasso Regression models and ~0.15 in Linear Regression model given the data range is from 0 to 1. Their R_squared values suggest that from 0.68-0.76 of training data is captured, which is fairly high. Residuals histograms for test dataset have normal distribution which is a healthy sign for our models.

However, we can observe that the majority of predictors (94 out of 100) are not significant given that their p-values are above 0.05 in the Linear Regression model or 59 out of 100 coefficients are set to 0 in Lasso Regression model. Additionally, RMSE values in Ridge and Lasso Regression models are higher than Linear Regression model's while their R_squared is smaller. These abnormalities can be best explained by the high correlation between predictors.

As discussed in 1.1 Data Exploration, some have high correlation with other predictors which means they are modelling the same things and may lead to p-value abnormality in Linear Regression model. With that reason, we can try removing some similar predictors and examine the result.

2.2.3 Further Discussion

The predictors were chosen to be removed are the ones with high correlation (near 1 or -1) with the others in the same category, such as: Number of children: ' PctKids2Par ', ' PctYoungKids2Par '; Immigrants: ' PctImmigRec8 ', ' NumImmig ', ' PctImmigRec5 ': Rent: ' RentLowQ ', ' RentHighQ '. However, I still leaved one / some unique predictors (with unique correlation score) in each category in the dataset. From the result in Table 4 Result comparision (before and after terms removal, we can clearly see that there are improvements in the validation and testing RMSE when we remove the mentioned predictors in the dataset. Also, we successfully reduced the computing cost to our machine since less terms are fetched into the model so the gain here is much more significant.

The same phenomenon with a decrease in R_square and an increase in RMSEs is observed when moving from Linear Regression to Lasso / Ridge Regression model. In fact, this is similar to what Lasso / Ridge Regression model do to their predictors. They reduce or remove completely the weights of unimportant / unrelated terms from the model and in return, receive higher accuracy. When the model is complex, some metrics may behave unpredictably so it can be safe to say that R_square should only be considered as a suggestion metric.

Linear Regression Results	Before removal	After removal
RMSE_train	0.104200	0.105886
RMSE_val	0.154912	0.152282
RMSE_test	0.153292	0.150934
R_squared (from training result)	0.759085	0.751226

TABLE 4 RESULT COMPARISION (BEFORE AND AFTER TERMS REMOVAL)

Finally, the prediction inaccuracy may also be caused by many more external, inconsistent factors that are related to the final result. For examples, flaw in collecting data, living condition improvement, goverment support services development, etc. All of these examples could certainly affect the number of crimes in an area and this dataset does not cover them. Therefore, it is acceptable to exhibit a small level of error in this model like what we see in test RMSE.

Problem 2: Classification

1 Data

1.1 Data Exploration

Data exploration shows that the original data value range varies on different ranges (See Appendixes/ 3) so standardisation is implemented for equal consideration among dimensions when fitting a model. To ensure the practical trait in performance, validation and test datasets are standardized using mean and standard deviation calculated from train data. No null value is found in the datasets so no further data pre-processing is needed.

The final prediction goal ('class' column) is classifying land use into 4 types: 's' for Sugi forest, 'h' for Hinoki forest, 'd' for mixed deciduous forest, and 'o' for others. 'class' data is converted to Categorical type so that the model is able to understand and extracts information.

However, many terms in this dataset are highly correlated with each other (See Appendixes/ 4), which means some of them may be redundant. Imbalance between classes as well as difference in each class's data points between 3 datasets (See Appendixes/ 5) are also observed and thus, classifiers should have adequate measures. Despite that, the final prediction goal is considered as uncritical in this scenario (e.g. unrelate to human lives) so it is acceptable if the models choose to maximize the accuracy in the majority classes which were initially anticipated to be 'd' and 's' (with 1st and 2nd highest numbers of datapoints in train dataset).

Training, validation, testing have shapes of (198, 28), (164, 28), and (161, 28) respectively. Similar to Problem 1, this is not ideal split ratio due to the same reasons discussed in **1.1 Data Exploration**.

1.2 Data Pre-processing

To standardize data, for each dimension in the train data (except for the goal dimension), we get the mean and standard deviation. Then, data is substracted by the mean and divided by the standard deviation. This process is repeated with validation and test data but with using mean and the standard deviation we had from train data.

2 Models

2.1 Hyper-parameter selection

Throughout Problem 2, all hyper-parameter selection processes utilized **GridSearchCV** to exhaustively considers all parameter combinations we provide with the param_grid parameter and **PredefinedSplit** to configure GridSearchCV to only use the train data for fitting and validation data for performance evaluation. The best hyper-parameter set is selected based on the highest validation accuracy (Scikit-learn.org, 2022).

2.1.1 K-Nearest Neighbours Classifier (KNN)

The param_grid is described in the code block below:

```
param_grid = [{
        'n_neighbors': list(range(1,101)),
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan', 'chebyshev']
}]
```

The number of n_neighbors is capped at 100 which is an already large enough value and already covered the most recommended value which is 14 - the square root of sample number in our train dataset (Jordan, 2022). PredefinedSplit is utilized so the GridSearchCV evaluates each combinations based on the validation data.

The best set of hyper-parameter found is: {'metric': 'euclidean', 'n_neighbors': 16, 'weights': 'uniform'}. As expected, the n_neighbors value is pretty closed to our initial conjecture – 14.

2.1.2 Random Forest

As there is imbalance of numbers of samples between classes, we also tried 'balanced_subsample' as a possible hyper-parameter for class_weight. max_depth and n_estimators are attempted at a wide range and with medium interval of 5 for n_estimators.

```
param_grid = [{
        'n_estimators': list(range(1,302,5)),
        'max_depth': list(range(1,11)),
        'class_weight': ['balanced_subsample', None]
    }]
```

The best hyper-parameter set found in this first round is {'class_weight': 'balanced_subsample', 'max_depth': 2, 'n_estimators': 186}. Finer search interval of 1 in n_estimators from 180 to 192 inclusive delivers the final best n_estimators of 184.

Although the hyper-parameter has been searched in more smaller interval, the gain in validation performance is not significant or even negative in some runs (because the randomness trait of Random Forest). Later findings also show that the test performance behaves in the opposite direction with the validation performance due to difference in class samples, which will be discussed in details in **2.2 Comparision & Evaluation**.

2.1.3 Support Vector Machines: One-vs-One (1v1)

As this is a multiclass class classification problem, the sklearn.svm.SVC class will automatically handle according to one-vs-one scheme. In Support Vector Machine (SVM), 3 types of kernel (linear, rbf, poly) are available so 3 according sets of grid_param is used. The 'degree' of poly kernel is capped at 7 since greater degree is more likely to overfit the model. 'balanced' class weight is also attempted as it helps adjust weights inversely proportional to class frequencies in the input data and reduce the negative impact of imbalance sample numbers (Scikit-learn.org, 2022). The remaining hyper-parameters are used as below:

```
param_grid = [
  {'C': [0.01,0.1,1,10,100,1000], 'kernel': ['linear'], 'class_weight': ['balanced',None]},
  {'C': [0.01,0.1,1,10,100,1000], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf'], 'class_weight': ['balanced',None]},
  {'C': [0.01,0.1,1,10,100,1000], 'degree': [2,3,4,5,6,7], 'kernel': ['poly'], 'class_weight': ['balanced',None]},
]
```

The best set found is {'C': 0.1, 'class_weight': None, 'kernel': 'linear'}. The second and third search round focus on finer 'C' value and the results are shown in Table 5 below:

	Searched C values	Best C
2nd	0.05-0.15 with 0.01 interval	0.01
3rd	0.09-0.11 with 0.001 interval	0.0949

TABLE 5 SVM FINER SEARCH: ROUND 2 AND 3

So our final hyper-parameter set is {'C': 0.0949, 'class_weight': None, 'kernel': 'linear'}

2.2 Comparision & Evaluation

Overall, the accuracy in all models decreases from train, validation to test data (See score details in Appendixes/6). SVM 1v1 gives the highest test score. The test accuracy ranges from 78-83% and also varies in multiple runs with different random states (if applicable).

Furthermore, the overall performance (espressed in F1-score) in each individual class is consistent throughout 3 methods and is heavily affected by the number of instances. For example, 's' and 'd' classes maintain the highest numbers of instances in all datasets. Therefore, they usually perform better than 'h' and 'o' (expressed in F1-score).

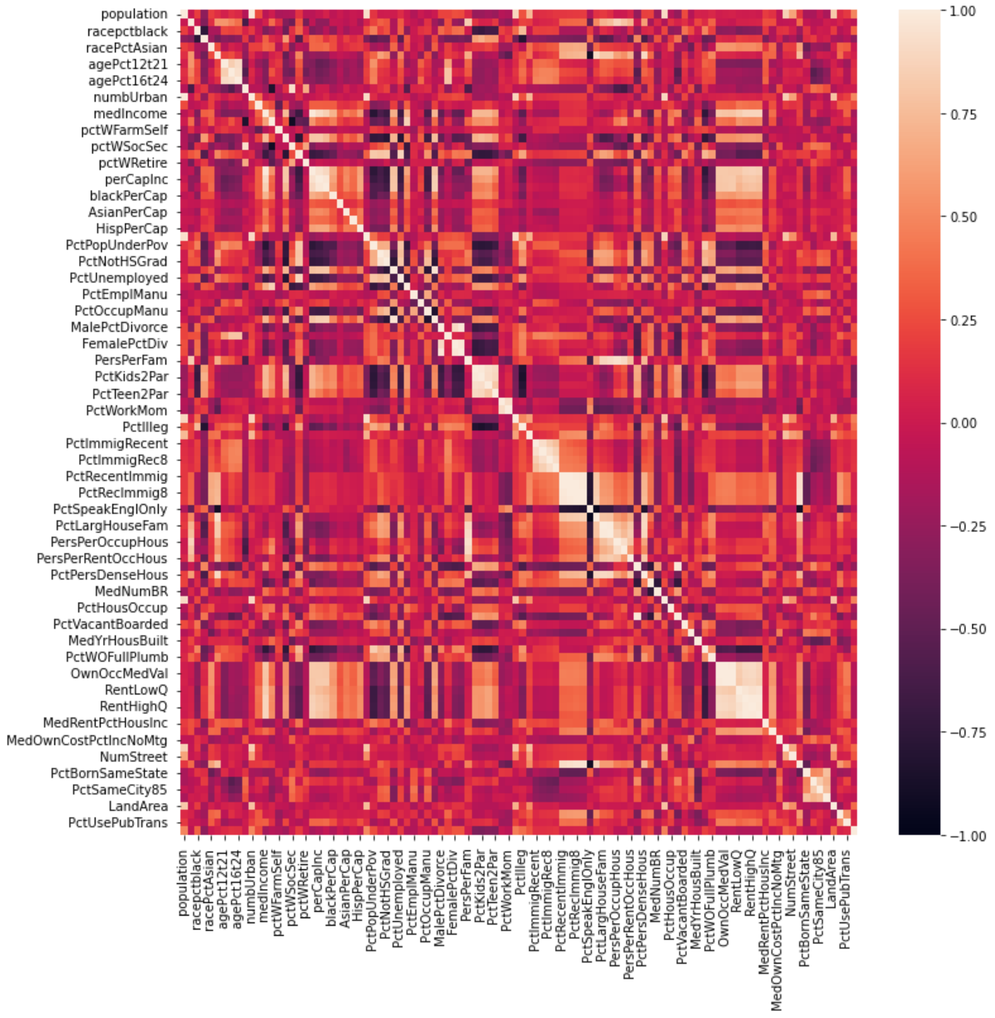
Interestingly, precission and recall scores of a class can also tell more stories about the dataset. Take 'o' class for instance: There are few 'o' datapoints in training so when the model tries to optimize accuracy, it actually tries to classify fewer instances as 'o'. Hence, fewer false positive leads to higher Precission. And since there are more 'o' in testing, more false negative leads to lower Recall.

The reversed scenario can be observed in 'h' class.

Furthermore, because the model learns from training data, any change to the test data, class distribution for instance, will impact the performance. Therefore, it can be speculated that experimenting model with class_weight=None may perform more accurately due to different class distribution in test data or to express more generally, due to uncertain test data (See result comparison in Appendixes/7).

Appendixes

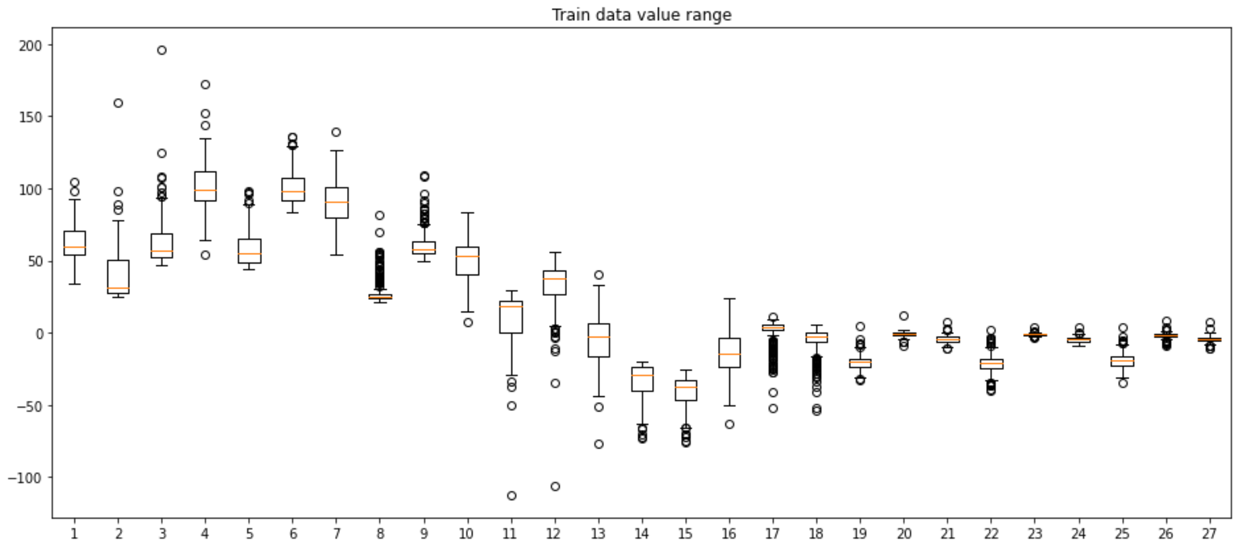
1. Correlation heatmap between data features



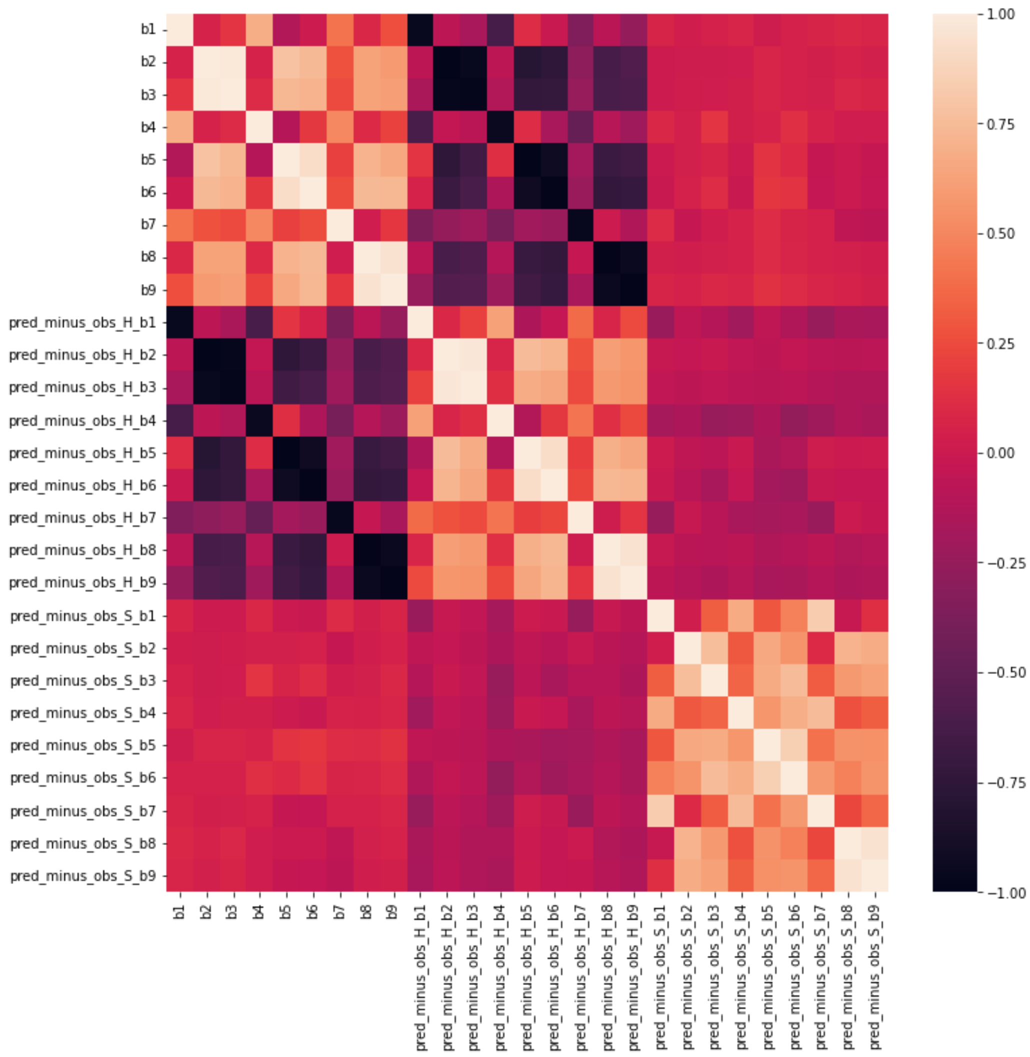
2. From the Root-mean-square deviation (RMSE) and Coefficient of determination (R-Squared) scores below, the model with fit_intercept=True performs slightly better in the training and validation dataset.

fit_intercept	True	False
RMSE_train	0.104200	0.104350
RMSE_val	0.154912	0.155598
R_squared (from training result)	0.759085	0.758391

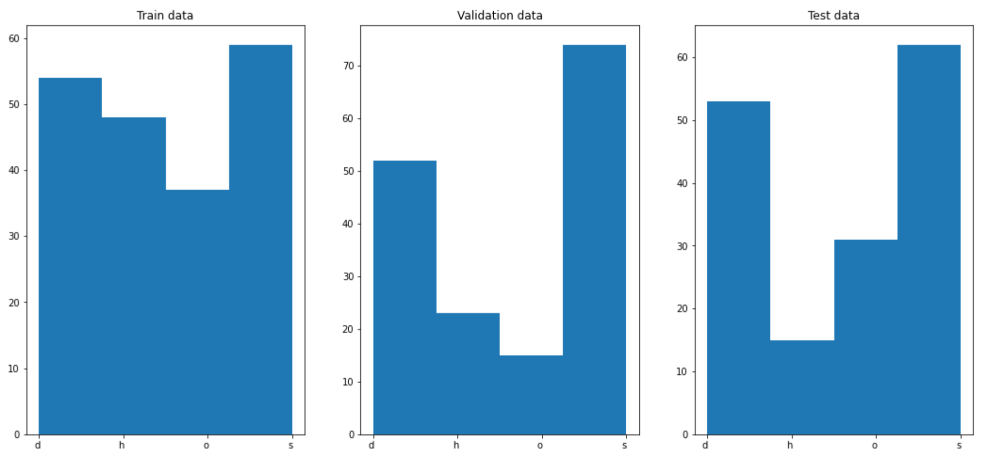
3. Problem 2: Train data value range



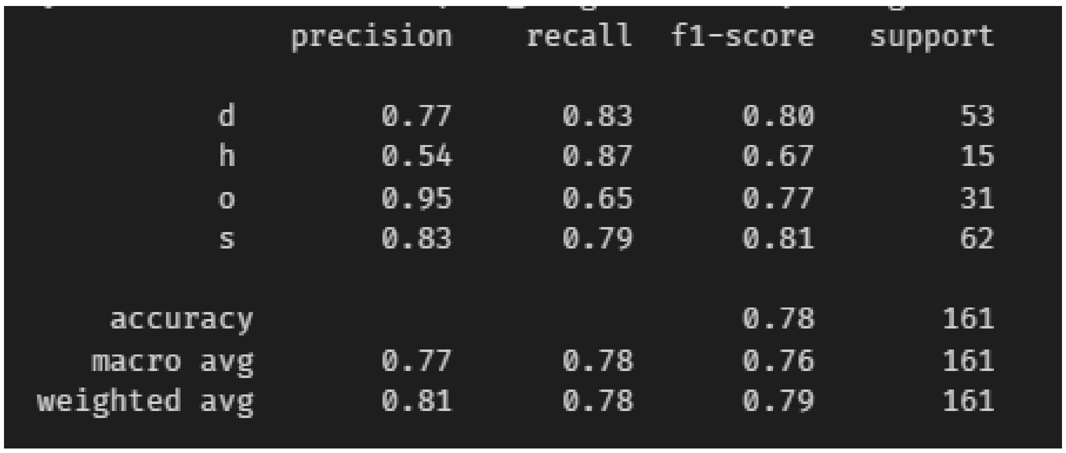
4. Train data correlation matrix



5. Classes distribution in train, val, test datasets



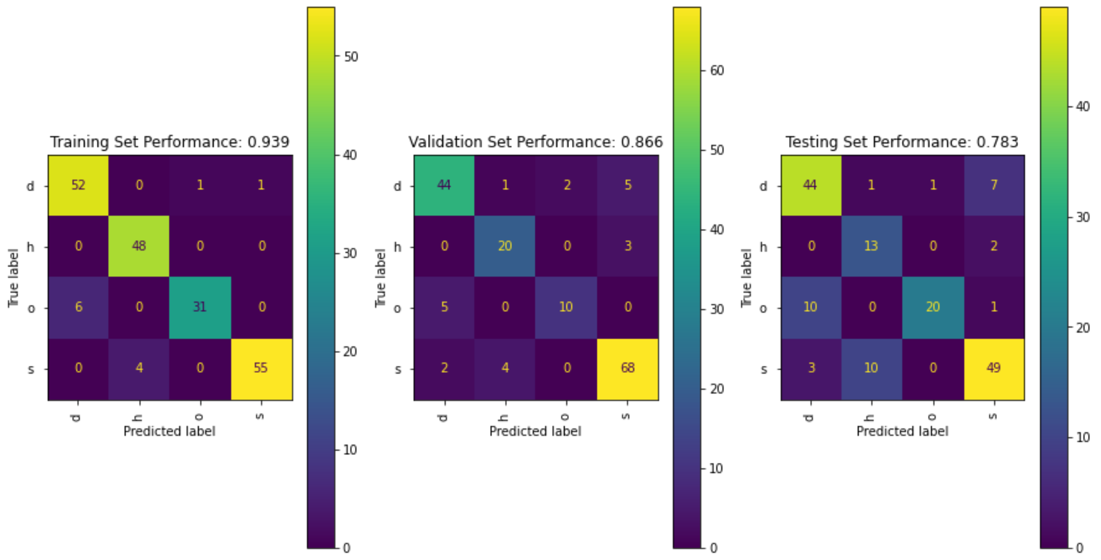
6. Multi-class classification test prediction results and other metrics
   - KNN:



```
              precision    recall  f1-score   support

           d       0.77      0.83      0.80        53
           h       0.54      0.87      0.67        15
           o       0.95      0.65      0.77        31
           s       0.83      0.79      0.81        62

    accuracy                           0.78       161
   macro avg       0.77      0.78      0.76       161
weighted avg       0.81      0.78      0.79       161
```



Training set:

	d	h	o	s
d	52	0	1	1
h	0	48	0	0
o	6	0	31	0
s	0	4	0	55

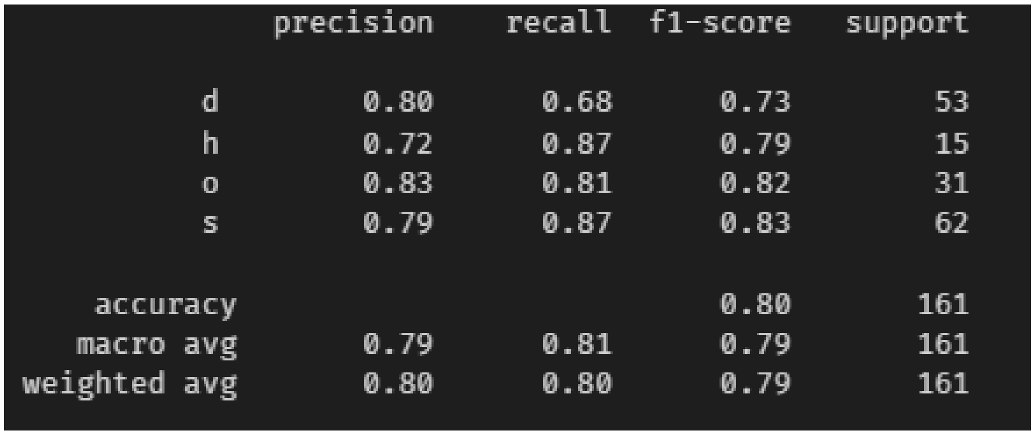
Validation set:

	d	h	o	s
d	44	1	2	5
h	0	20	0	3
o	5	0	10	0
s	2	4	0	68

Testing set:

	d	h	o	s
d	44	1	1	7
h	0	13	0	2
o	10	0	20	1
s	3	10	0	49

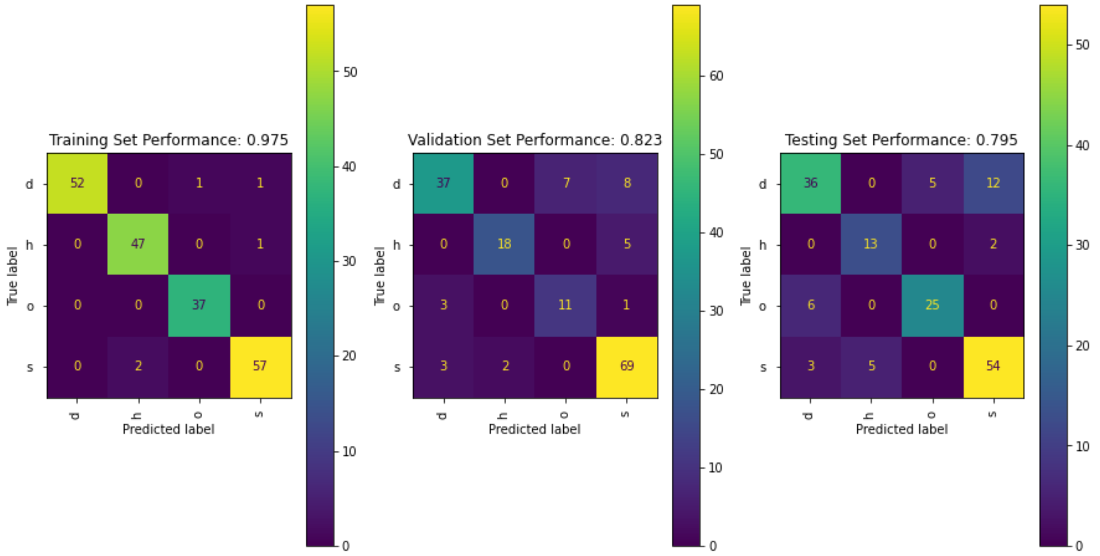
   - Random forest:



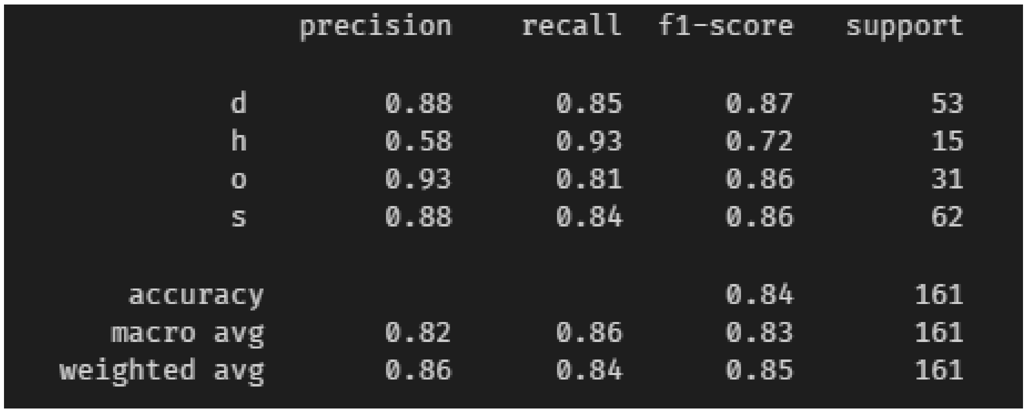
```
              precision    recall  f1-score   support

           d       0.80      0.68      0.73        53
           h       0.72      0.87      0.79        15
           o       0.83      0.81      0.82        31
           s       0.79      0.87      0.83        62

    accuracy                           0.80       161
   macro avg       0.79      0.81      0.79       161
weighted avg       0.80      0.80      0.79       161
```



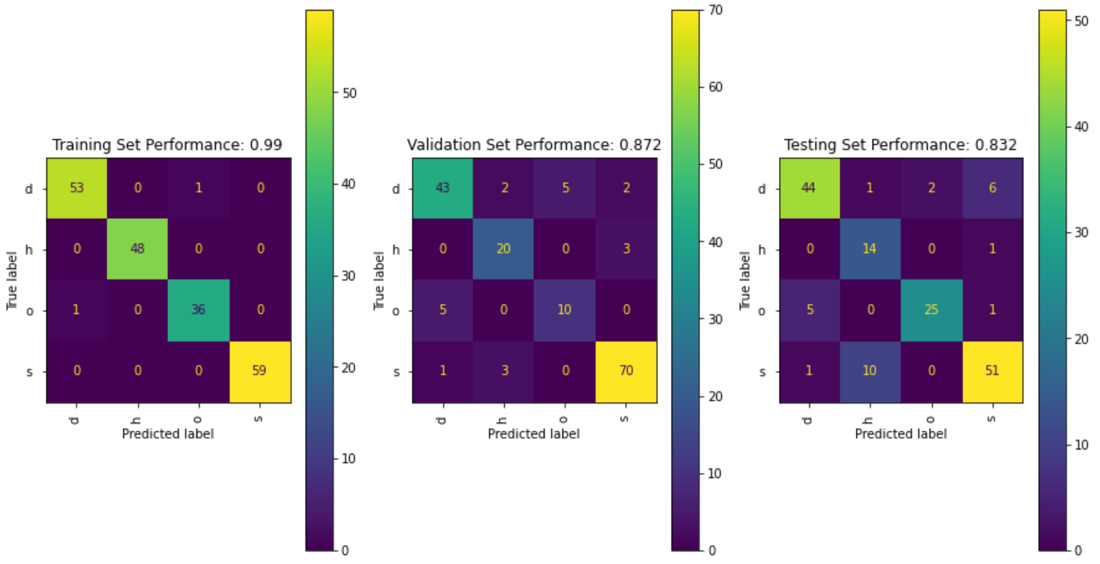
- SVM 1v1:



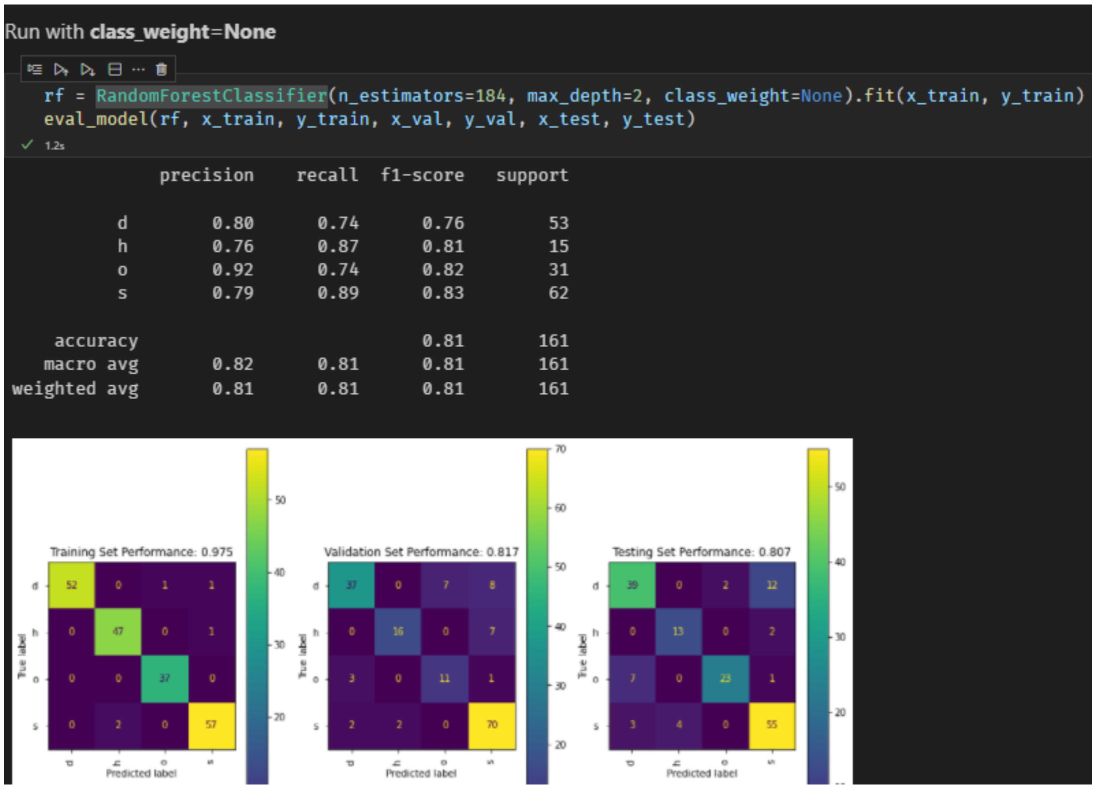
```
              precision    recall  f1-score   support

           d       0.88      0.85      0.87        53
           h       0.58      0.93      0.72        15
           o       0.93      0.81      0.86        31
           s       0.88      0.84      0.86        62

    accuracy                           0.84       161
   macro avg       0.82      0.86      0.83       161
weighted avg       0.86      0.84      0.85       161
```



7. Prediction performance of Random Forest and SVM 1v1 with class_weight=None are approximately equal or higher than ones with class_weight='balanced'/'balanced_subclass' (compared with results in Appendixes/6).
    - Random Forest



Run with **class_weight=None**

```
rf = RandomForestClassifier(n_estimators=184, max_depth=2, class_weight=None).fit(x_train, y_train)
eval_model(rf, x_train, y_train, x_val, y_val, x_test, y_test)
```

```
              precision    recall  f1-score   support

           d       0.80      0.74      0.76        53
           h       0.76      0.87      0.81        15
           o       0.92      0.74      0.82        31
           s       0.79      0.89      0.83        62

    accuracy                           0.81       161
   macro avg       0.82      0.81      0.81       161
weighted avg       0.81      0.81      0.81       161
```

1.2s
Training Set Performance: 0.975
Validation Set Performance: 0.817
Testing Set Performance: 0.807
True label
Predicted label

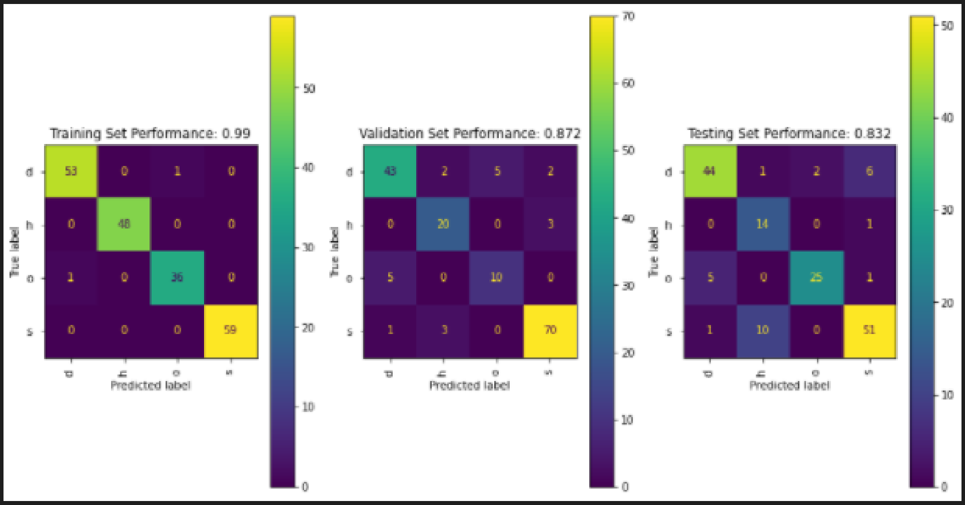
- SVM 1v1

```
svm = SVC(C=0.0949, kernel='linear', class_weight=None)
svm.fit(x_train_std, y_train)
eval_model(svm, x_train_std, y_train, x_val_std, y_val, x_test_std, y_test)
```

```
✓ 1.1s
              precision    recall  f1-score   support

           d       0.88      0.83      0.85        53
           h       0.56      0.93      0.70        15
           o       0.93      0.81      0.86        31
           s       0.86      0.82      0.84        62

    accuracy                           0.83       161
   macro avg       0.81      0.85      0.81       161
weighted avg       0.85      0.83      0.84       161
```



References

Jordan, J. (2022). *K-nearest neighbors*. Retrieved 25 April 2022, from https://www.jeremyjordan.me/k-nearest-neighbors/.

Scikit-learn.org. (2022). Retrieved 25 April 2022, from https://scikit-learn.org.