ASSIGNMENT 1A

CAB420, Machine Learning

Ash Phillips, N10477659
Date: 21/04/2022

Problem 1: Regression

Discussion of Data Characteristics and Pre-Processing

Characteristics and Issues Within the Data

The data set was multivariate with real attributes. It was assumed this data was accurate, reliable, and relevant to the prediction of the number of violent crimes per capita. Some values were missing within the data, potentially affecting the predictions. In future exploration, the accuracy of the predictions may be improved by filtering out this data.

Pre-Processing

The pre-processing of data before modelling would normally include filtering data containing NaNs (missing data), encoding categorical data (e.g., reformatting dates), and then splitting the data into training, testing, and validation data sets. As the data was provided pre-split, none of this processing was required.

Before the modelling process, constants were added to the data to be used in the model to learn the intercept. Each data set was then split into *x* and *y* values.

No standardisation was performed on the data for linear regression as it was not required to create an accurate model.

Once the linear model had been finalised, the data was standardised so variables with larger ranges could not distort the results, skewing the accuracy of the succeeding models. This gave the data a mean of 0 and standard deviation of 1 so the ridge and LASSO regression processes would consider all data values equally; these regularisation process would not attempt to minimise all model weights equally to maximise accuracy, causing further issue.

Model Development and Hyper-Parameter Selection

Linear Model

This model was created via the Statsmodel's Linear Regression method on the test data set, using OLS (Ordinary Least Squares) to find R^2. As linear regression does not use hyper-parameters nothing further was done.

Ridge Model

Ridge regression follows the same process as linear regression with an included L2 penalty term (the sum of squared coefficient values), which is used to regularise the model. The hyper-parameter lambda (*λ*) controls the regularisation weighting. This makes the value weights small for regularsation; as it gets bigger it becomes more important than data fitting.

For the selection of lambda, a range of values were used to find which value would work best for an accurate resulting model.

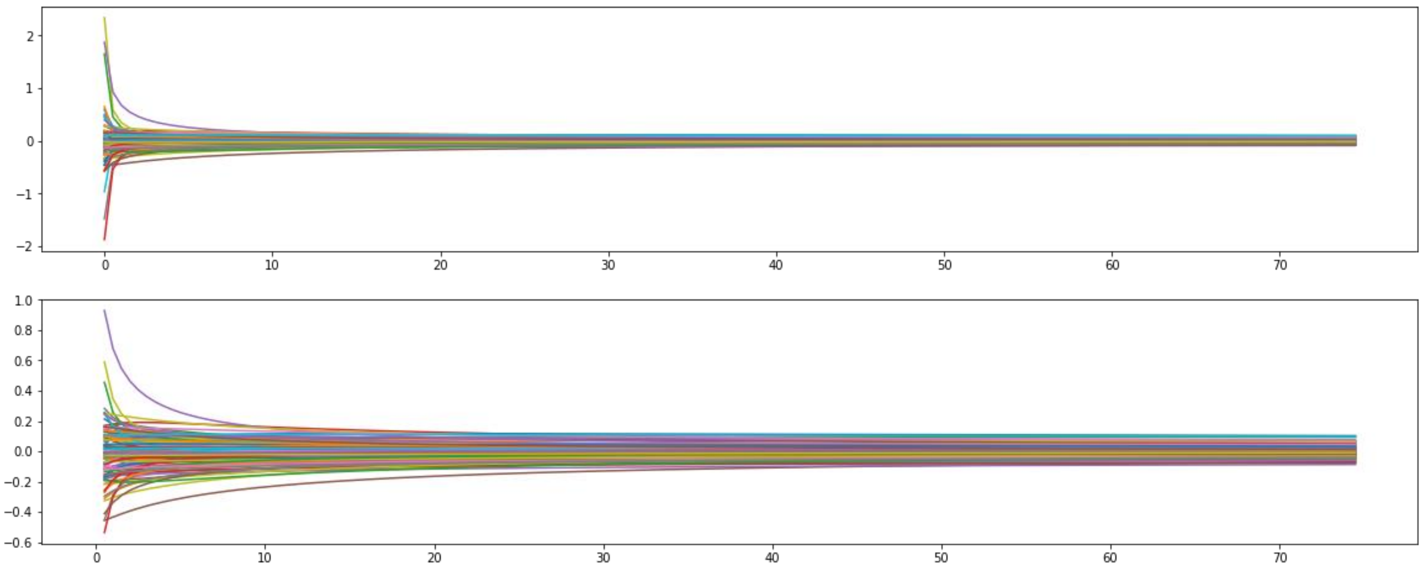


Figure 1: The model of the selection method to find the best value for lambda for Ridge Regression.

As seen in *Figure 1* above, as lambda changes, values are suppressed but are never eliminated (i.e., get smaller but never become zero), with some values taking longer than others. As shown, when lambda is approaching 40, all values are fully suppressed, therefore highlighting the best possible lambda to create an accurate model for ridge regression; the best lambda was determined to be 40.5 through caluation of model results.

When developing the model with this selected lambda, the standardised data was converted back to the original range for comparison with the linear model.

LASSO Model

LASSO regression also uses the same process as linear regression, albeit including a L1 penalty term (the sum of the absolute value of the coefficients). Again, lambda is used to control the regularisation weighting, requiring the repetition of the evaluation process.

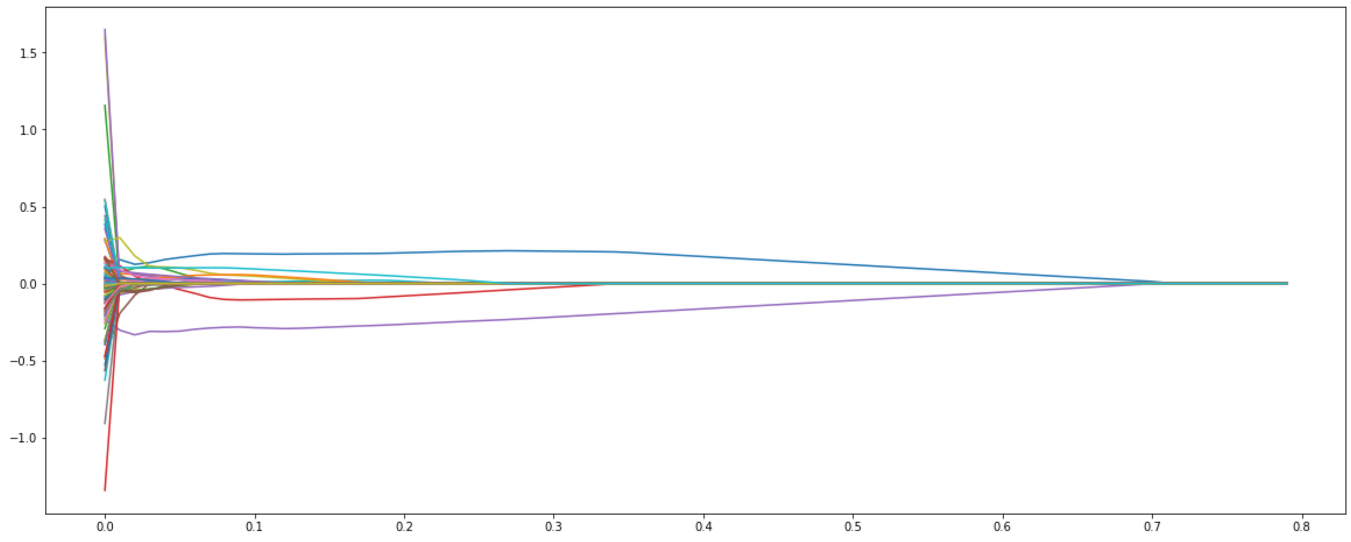


Figure 2: The model of the selection method to find the best value for lambda for LASSO Regression.

As seen in *Figure 2* above, unlike what is seen previously in *Figure 1*, as lambda changes, value weights do go to zero, therefore eliminating values. Once the end of the plot is reached, all the values will become zero. The best value for lambda was selected from the point where most values become zero, which was determined through calculation of model results to be 0.02.

When developing the LASSO model with this selected lambda, the standardised data was again converted back to the original range for comparison with the linear model.

Model Evaluation and Analysis

The accuracy of each regression model depends on the R^2 variable. The closer R^2 is to one, the more accurate it is. To calculate this R^2 variable for linear regression, OLS regression was performed. As can be seen in *Table 1* below, the R^2 value was exactly 1, resulting in the highest accuracy. Later, this R^2 was also computed for the ridge and LASSO models by looking at the final models' scores. These were computed to be approximately 0.67 for ridge and 0.70 for LASSO. These values for R^2 are very similar to each other, though not as accurate as the value for linear regression. From these R^2 values, the expected models should be perfectly accurate for linear regression and have a very close relationship between the actual and predicted data for both the ridge and LASSO models.

```
                              OLS Regression Results
==================================================================================
Dep. Variable:      ViolentCrimesPerPop    R-squared:                      1.000
Model:                              OLS    Adj. R-squared:                 1.000
Method:                   Least Squares    F-statistic:                2.994e+27
Date:                  Sun, 24 Apr 2022    Prob (F-statistic):              0.00
Time:                          02:19:20    Log-Likelihood:                9374.6
No. Observations:                   299    AIC:                       -1.855e+04
Df Residuals:                       197    BIC:                       -1.817e+04
Df Model:                           101
Covariance Type:              nonrobust
============================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
```

Table 1: Ordinary Least Squares regression results to calculate R^2 for linear regression.

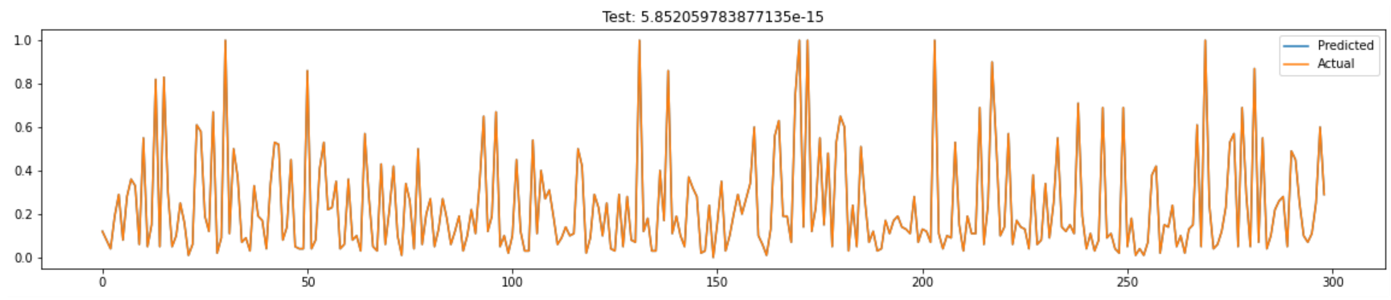


Figure 3: The overlapping actual and predicted values model for linear regression, and the RMSE (Root Mean Squared Error) of the model – evaluated on the test set.

In *Figure 3* above, as predicted, the linear regression prediction data accuracy is almost perfect as the two lines (predicted and actual) overlap, with a RMSE (root mean squared error) of approximately 5.85×10^{-15}, which is an incredibly small amount of error.

In *Figure 4* below, for ridge regression, it can be seen the predicted data does not match the actual data as close as the linear regression model, as was expected. It does not appear to be overfitting and only has a small amount of error; the RMSE for the testing set data is approximately 0.13. While this model is not as perfect as the linear regression model, due to the low error, it still accurately portrays the data and is still a good fit.

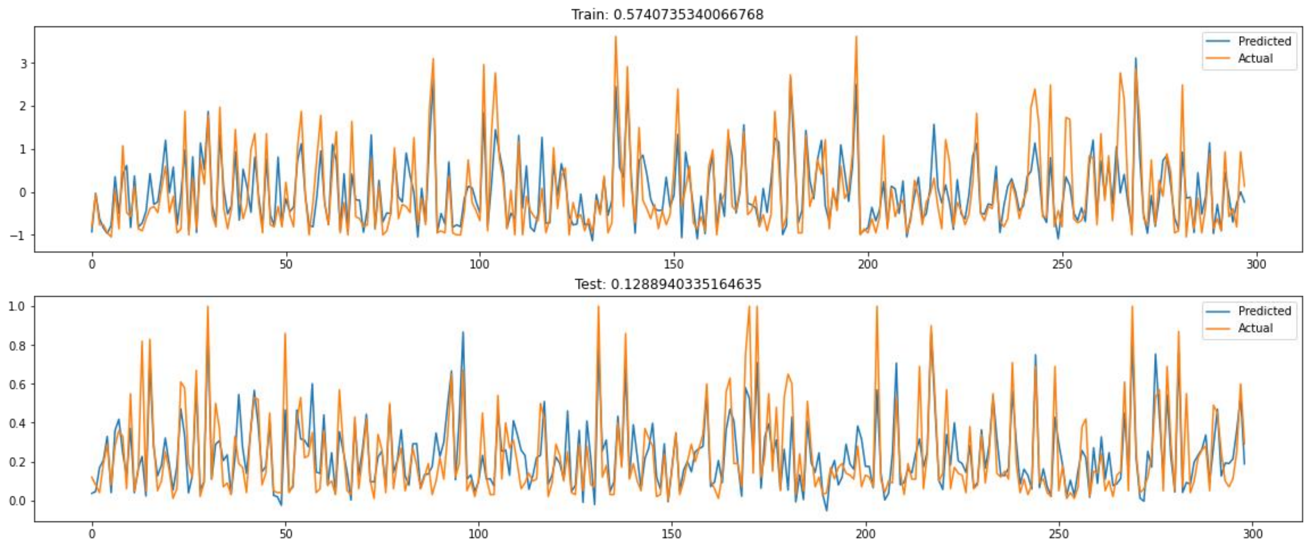


Figure 4: The actual and predicted values models for ridge regression training and testing sets, and the RMSE of each model.

As expected, the LASSO model acted almost the same as the ridge regression model. In *Figure 5* below, it can be seen the predicted values do not fit the data exactly, but again only with a small amount of error; the RMSE for the testing set data is approximately 0.13.

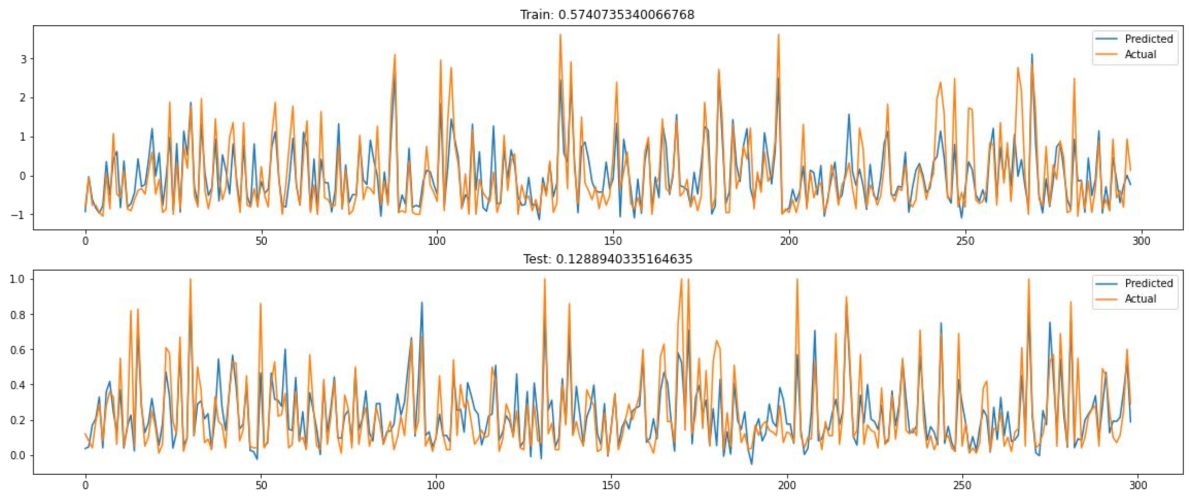


Figure 5: The actual and predicted values models for LASSO regression training and testing sets, and the RMSE of each model.

When comparing these three models, for this data, the linear regression model is the most accurate, with the ridge and LASSO models acting very similar to each other. Looking at the final three figure below, we can see this judgement is correct. These figures show all three regression models fit the relationship between the actual and predicted data close to a straight line.

In *Figure 7* and *Figure 8*, the similarity between the results is highlighted, with the ridge model looking to be slightly more fitted than LASSO and therefore more accurate. In *Figure 6*, it can be seen the data fits almost perfectly on the line, with little error, as was seen throughout the linear models. Therefore, it can be concluded, the linear model was the highest accurate model for predicting this data set.

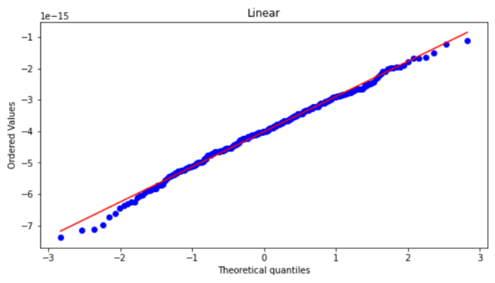


Figure 6: The relationship between the sample and theoretical values for linear regression.

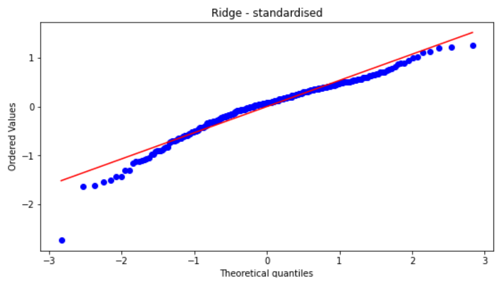


Figure 7: The relationship between the ordered and theoretical values for ridge regression.

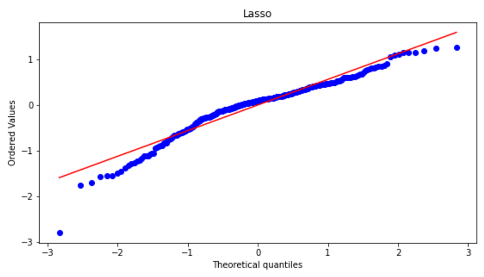


Figure 8: The relationship between the ordered and theoretical values for LASSO regression.

Problem 2: Classification

Discussion and Justification

Characteristics and Issues Within the Data

It was assumed the provided data was accurate, reliable, and relevant for the classification of land types. The data set was multivariate with real-world attributes. There were no values missing from the data; it was complete.

Pre-Processing

As the provided data was already split into the testing, training, and validation data sets, no further editing was performed (e.g., filtering, encoding, etc.).

Each data set was standardised to aid the performance of certain classification processes when modelling the data. This was necessary as, by scaling the data this way, the model algorithms could judge all data values equally, enabling the potential for better analysis.

This standardisation effected comparison of the data in the K-Nearest Neighbours (KNN) and Support Vector Machines (SVM) model processes, as these algorithms are distance based; directly affected by the distance of the variables. If the data was not standardised, the resulting models may have been less accurate as higher distances could have caused higher allocated weights, impacting performance and accuracy.

This standardisation would not affect the Random Forest algorithm as it is tree-based, and therefore is not affected by the distance of variables. Though standardised data was still used to provide simpler comparison of models.

Model Development and Hyper-Parameter Selection

The selection method used to find the best possible hyper-parameters for each model was grid search. By using the grid search method, all possible combinations of parameters were able to be searched over to select which would be best for each model.

For the KNN classifier, the parameters number of neighbours (*k*) and the weight type were needed to be found. This was done by creating a list ranging from 1-131 for *k* (chosen as 131 is the number of samples), and a weight of either ‘uniform’ or ‘distance’. After the grid search, the best values were found to be:

k = 10, weight = ‘distance’

With the Random Forest classifier, the parameters maximum depth, minimum samples split, and number of estimators were required. These were chosen following different multiples. Depth was chosen via a multiple of 2 (i.e., 2, 4, etc.), sample splits used a multiple of 5 (i.e., 5, 10, etc.), and finally estimators used a set of 25, 50, and 100. Using this parameter list, the best values found were:

maximum depth = None, minimum samples split = 5, and number of estimators = 50

Finally, for the SVM classifier, the parameters C and kernel were required, including different kernel parameters used for ‘rbf’ and ‘poly’ kernels. Using a parameter list of unique values for C (following orders of magnitude from 0.1 to 1000), kernel types (linear, rbf, poly), and kernel parameters (gamma and degree), the best values found were:

C = 1000, kernel = ‘rbf’, and ‘gamma’ = 0.0001

Model Evaluation and Analysis

Each of the final models are confusion matrices that show the true and predicted classes on the vertical and horizontal axes respectively. For these models to show the ideal performance, the results should show values of 1 along the top-left diagonal and 0 for each other value.

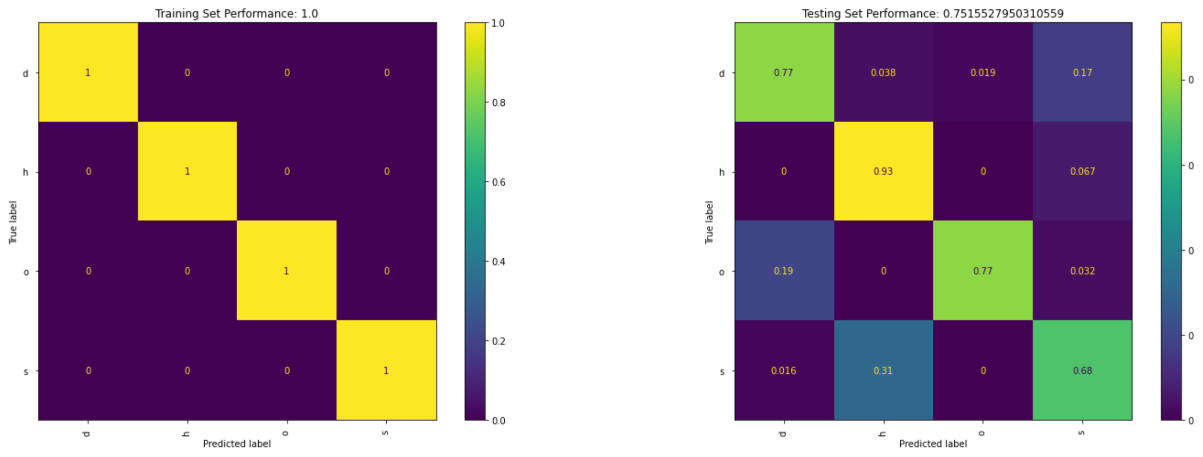


Figure 9: The final confusion matrices displaying the performance of the K-Nearest Neighbours (KNN) classifier on the training and testing sets.

As seen in *Figure 9* above, the performance of the KNN classifier when evaluating the testing set was approximately 75.16% accurate. The accuracy when predicting the true classes of forests were 77% for class *d* (Mixed deciduous forest), 93% for class *h* (Hinoki forest), 77% for class *o* (Other non-forest land), and 68% for class *s* (Sugi forest). The highest error percentage of 31% can be seen when the true class is *s* and the predicted class is *h*. This is quite a high level of error, likely caused by too small a parameter set for the grid search. In future evaluation, a larger set may want to be used. Furthermore, the model does not seem to overfit the data, as no value is higher than that of the training set model.

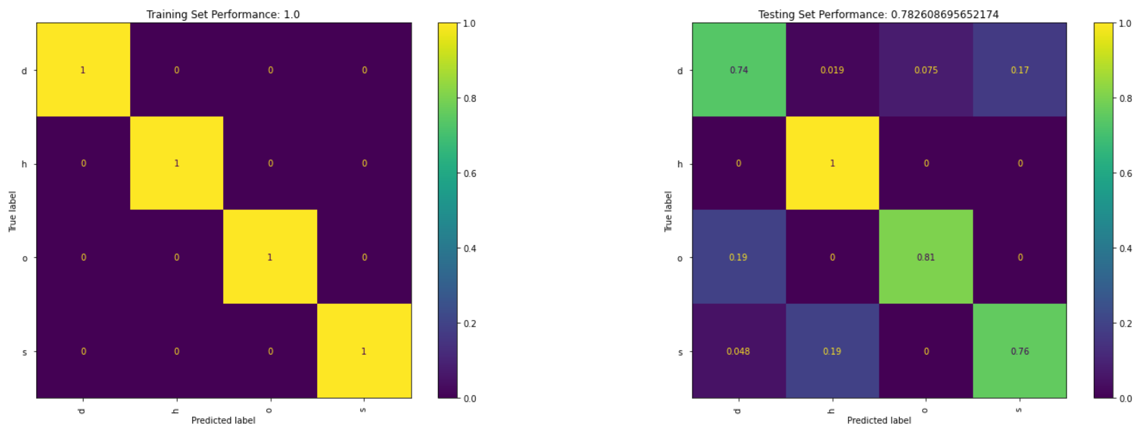
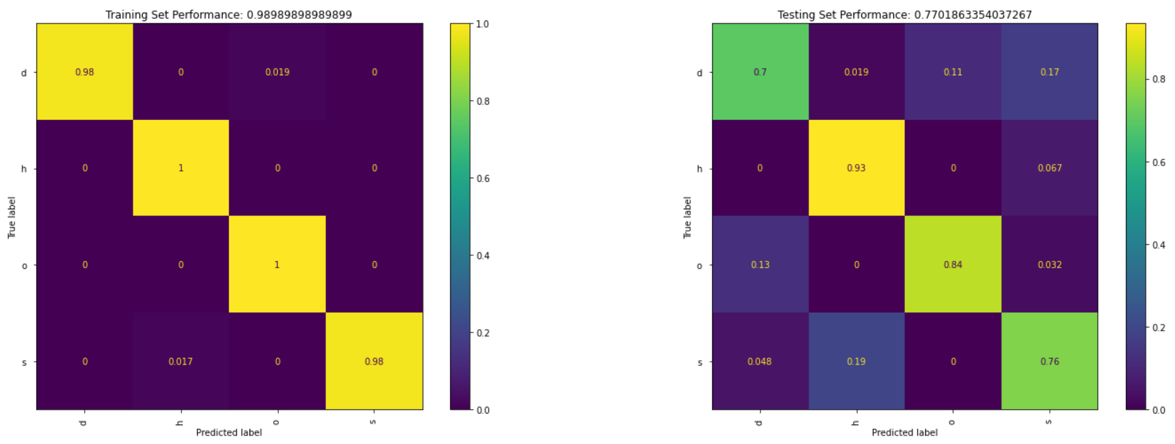


Figure 10: The final confusion matrices displaying the performance of the Random Forest classifier on the training and testing sets.

The performance of the Random Forests classifier on the testing set, seen in *Figure 10* above, was approximately 78.26% accurate, slightly higher than the KNN classifier. The accuracy when predicting the true classes of forests can be seen to be 74% for class *d*, 100% for class *h*, 81% for class *o*, and 76% for class *s*. Similarly to KNN, the performance of the training set evaluation was also 100% accurate. The highest error percentage of 19% is present in two predictions (true class of *o* and *s* with predicted class of *d* and *h* respectively). This error is much smaller than that of the KNN model, showcasing this classifier can predict the land type with higher accuracy. The model does not seem to overfit the data when compared to the train set model.

As the Random Forest classifier does have an element of randomness to it, this final model took multiple iterations to create. In *Figure 11* below, another model with slightly less accuracy can be seen. While the discrepancies between both models are slight and either can be considered accurate, the validity of models created via the Random Forests classifier may be impacted if the randomness creates larger error.



`Seed: 'max_depth': None, 'min_samples_split': 10, 'n_estimators': 50`

Figure 11: Different seed for Random Forest confusion matrices.

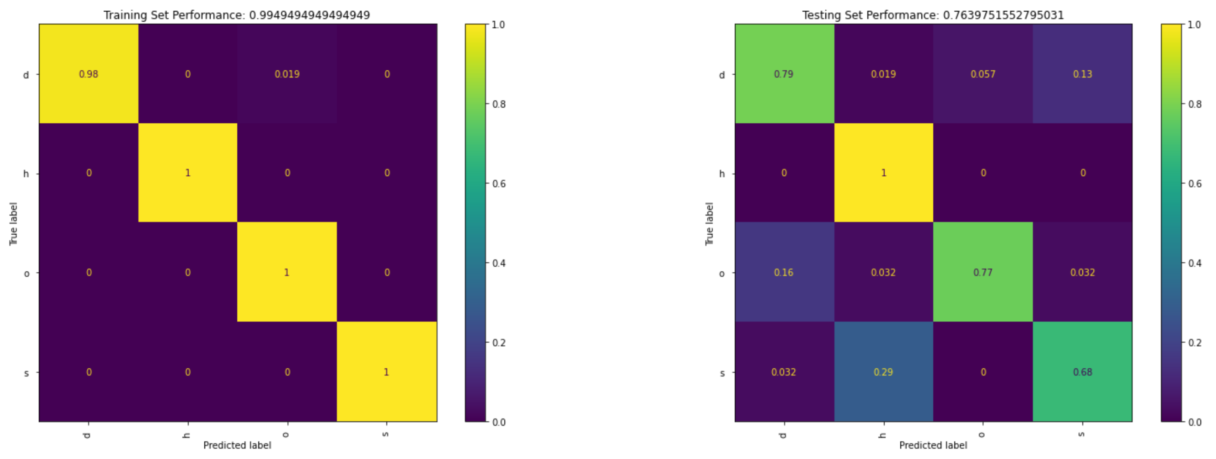


Figure 12: The final confusion matrices displaying the performance of the Support Vector Machine (SVM) classifier on the training and testing sets.

Finally, as seen in *Figure 12* above, the performance of the SVM classifier when evaluating the testing set was approximately 76.40% accurate, a higher actuary than the KNN classifier though still lower than Random Forests. The accuracy when predicting the true classes were 79% for class *d*, 100% for class *h*, 77% for class *o*, and 68% for class *s*. Unlike the other models, the performance of the training set evaluation was approximately 99.49% accurate. The highest error here was 29% with the true class of *s* and prediction of *h*. This error fits between the previous models, showing the SVM classifier accuracy is higher than KNN, but less than Random Forests. This model also does not overfit the data when compared to the train set model.

Overall, the random forest class had the highest accuracy at 78.26% compared to SVM (76.40%) and KNN (75.16%), concluding that this model works best for the given data and grid search parameters. However, as the accuracy for each model is very close, being within approximately 3%, all would work relatively well with the data.

Appendix

All code used was built upon using code from Lecture examples and Tutorial solutions.

n10477659 Final Assignment_1A

April 24, 2022

0.1 Assignment 1A

1 Import Functions

[4]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# numpy handles pretty much anything that is a number/vector/matrix/array
import numpy as np
# pandas handles dataframes
import pandas as pd
# matplotlib emulates Matlabs plotting functionality
import matplotlib.pyplot as plt
# seaborn is another good plotting library. In particular, I like it for␣
↪heatmaps (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
import seaborn as sns;
# stats models is a package that is going to perform the regression analysis
from statsmodels import api as sm
from scipy import stats
from sklearn.metrics import mean_squared_error, r2_score
# os allows us to manipulate variables on out local machine, such as paths and␣
↪environment variables
import os
# self explainatory, dates and times
from datetime import datetime, date
# a helper package to help us iterate over objects
import itertools


## Q1
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler

## Q2
from gensim.models import Word2Vec
import re
import string
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.svm import SVC, NuSVC
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV,␣
 ↪HalvingGridSearchCV
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from scipy.stats import norm
from sklearn import tree

# To export as pdf with better quality plots
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf', 'svg')
```

```
Mounted at /content/drive
```

---

2 Q1: Regression

- Train a linear regression model to predict the number of violent crimes per captia from the socio-economic data.
- Train a Ridge regression model to predict the number of violent crimes per captia from the socio-economic data.
- Train a LASSO regression model to predict the number of violent crimes per captia from the socio-economic data.

do not performing any filtering or selection to remove columns and/or rows

[ ]:
```
# Load the data
##info = pd.read_csv('/content/drive/My Drive/Colab Notebooks/
 ↪CAB420_Assessment_1A_Data/Communities_Info.txt')
test = pd.read_csv('/content/drive/My Drive/Colab Notebooks/
 ↪CAB420_Assessment_1A_Data/Q1/communities_test.csv')
train = pd.read_csv('/content/drive/My Drive/Colab Notebooks/
 ↪CAB420_Assessment_1A_Data/Q1/communities_train.csv')
val = pd.read_csv('/content/drive/My Drive/Colab Notebooks/
 ↪CAB420_Assessment_1A_Data/Q1/communities_val.csv')
```

[ ]:
```
# Plot data
# fig = plt.figure(figsize=[10, 6])
# ax = fig.add_subplot()
# ax.plot(test[' ViolentCrimesPerPop '])
# ax.set_title('Test Data')
# fig = plt.figure(figsize=[10, 6])
# ax = fig.add_subplot()
# ax.plot(train[' ViolentCrimesPerPop '])
# ax.set_title('Train Data')

# fig = plt.figure(figsize=[10, 6])
# ax = fig.add_subplot()
# ax.plot(val[' ViolentCrimesPerPop '])
# ax.set_title('Validate Data')
```

2.1 Linear Regression

[ ]:
```
## Add Constants ##
train = sm.add_constant(train)
val = sm.add_constant(val)
test = sm.add_constant(test)

## Data Splitting ##
X_train = train
Y_train = train.iloc[:,-1]
X_val = val
Y_val = val.iloc[:,-1]
X_test = test
Y_test = test.iloc[:,-1]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:117:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

[ ]:
```
## LINEAR MODEL ##
model = sm.OLS(Y_test, X_test)
trained_model = model.fit()
print(trained_model.summary())
```

```
                              OLS Regression Results
================================================================================
=
Dep. Variable:      ViolentCrimesPerPop    R-squared:
1.000
Model:                              OLS    Adj. R-squared:
1.000
Method:                   Least Squares    F-statistic:
2.994e+27
```

```
Date:                    Sun, 24 Apr 2022   Prob (F-statistic):
0.00
Time:                            14:17:19   Log-Likelihood:
9374.6
No. Observations:                     299   AIC:
-1.855e+04
Df Residuals:                         197   BIC:
-1.817e+04
Df Model:                             101
Covariance Type:                nonrobust
================================================================================
===========
                              coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
const                   -1.332e-15      4e-14     -0.033      0.973   -8.02e-14
7.76e-14
 population              1.998e-15   8.46e-14      0.024      0.981   -1.65e-13
1.69e-13
 householdsize           1.145e-16   1.88e-14      0.006      0.995   -3.69e-14
3.71e-14
 racepctblack            3.279e-16   9.53e-15      0.034      0.973   -1.85e-14
1.91e-14
 racePctWhite           -9.992e-16   1.16e-14     -0.086      0.932    -2.4e-14
2.2e-14
 racePctAsian           -4.198e-16   5.82e-15     -0.072      0.943   -1.19e-14
1.11e-14
 racePctHisp            -3.678e-16   1.02e-14     -0.036      0.971   -2.04e-14
1.97e-14
 agePct12t21              2.29e-15   1.88e-14      0.122      0.903   -3.49e-14
3.94e-14
 agePct12t29             1.804e-15   2.55e-14      0.071      0.944   -4.85e-14
5.22e-14
 agePct16t24            -2.755e-15    2.8e-14     -0.098      0.922    -5.8e-14
5.25e-14
 agePct65up              9.992e-16   1.76e-14      0.057      0.955   -3.36e-14
3.56e-14
 numbUrban              -1.776e-15   7.85e-14     -0.023      0.982   -1.57e-13
1.53e-13
 pctUrban               -2.498e-16   2.84e-15     -0.088      0.930   -5.84e-15
5.34e-15
 medIncome               2.481e-15   3.29e-14      0.075      0.940   -6.25e-14
6.74e-14
 pctWWage                6.384e-16   1.53e-14      0.042      0.967   -2.95e-14
3.08e-14
 pctWFarmSelf           -2.394e-16    3.4e-15     -0.070      0.944   -6.94e-15
6.47e-15
```

```
 pctWInvInc          -9.853e-16   1.19e-14      -0.083        0.934   -2.44e-14
2.25e-14
 pctWSocSec          -3.712e-16   1.79e-14      -0.021        0.984   -3.57e-14
3.5e-14
 pctWPubAsst          9.021e-17   9.59e-15       0.009        0.993   -1.88e-14
1.9e-14
 pctWRetire          -2.272e-16      7e-15      -0.032        0.974    -1.4e-14
1.36e-14
 medFamInc           -8.812e-16   3.24e-14      -0.027        0.978   -6.48e-14
6.3e-14
 perCapInc            9.992e-16   3.54e-14       0.028        0.978   -6.89e-14
7.09e-14
 whitePerCap          -1.11e-15   3.08e-14      -0.036        0.971   -6.18e-14
5.96e-14
 blackPerCap         -9.125e-16   4.25e-15      -0.215        0.830    -9.3e-15
7.47e-15
 indianPerCap        -1.648e-16   3.22e-15      -0.051        0.959   -6.51e-15
6.18e-15
 AsianPerCap         -1.154e-16   3.02e-15      -0.038        0.970   -6.07e-15
5.84e-15
 OtherPerCap         -3.123e-17   2.55e-15      -0.012        0.990   -5.06e-15
5e-15
 HispPerCap          -5.621e-16   4.21e-15      -0.133        0.894   -8.87e-15
7.75e-15
 NumUnderPov          1.221e-15   2.73e-14       0.045        0.964   -5.27e-14
5.51e-14
 PctPopUnderPov      -4.996e-16   1.19e-14      -0.042        0.967   -2.39e-14
2.29e-14
 PctLess9thGrade     -1.804e-16   1.12e-14      -0.016        0.987   -2.22e-14
2.18e-14
 PctNotHSGrad          2.22e-16   1.45e-14       0.015        0.988   -2.83e-14
2.87e-14
 PctBSorMore          6.939e-17   1.33e-14       0.005        0.996   -2.61e-14
2.63e-14
 PctUnemployed       -2.914e-16   7.98e-15      -0.037        0.971    -1.6e-14
1.54e-14
 PctEmploy           -4.059e-16   1.57e-14      -0.026        0.979   -3.13e-14
3.05e-14
 PctEmplManu         -1.388e-17   5.13e-15      -0.003        0.998   -1.01e-14
1.01e-14
 PctEmplProfServ      1.492e-16   7.29e-15       0.020        0.984   -1.42e-14
1.45e-14
 PctOccupManu         -2.22e-16   8.22e-15      -0.027        0.978   -1.64e-14
1.6e-14
 PctOccupMgmtProf    -8.743e-16    1.6e-14      -0.055        0.957   -3.25e-14
3.07e-14
 MalePctDivorce       2.637e-16   4.92e-14       0.005        0.996   -9.67e-14
9.72e-14
```

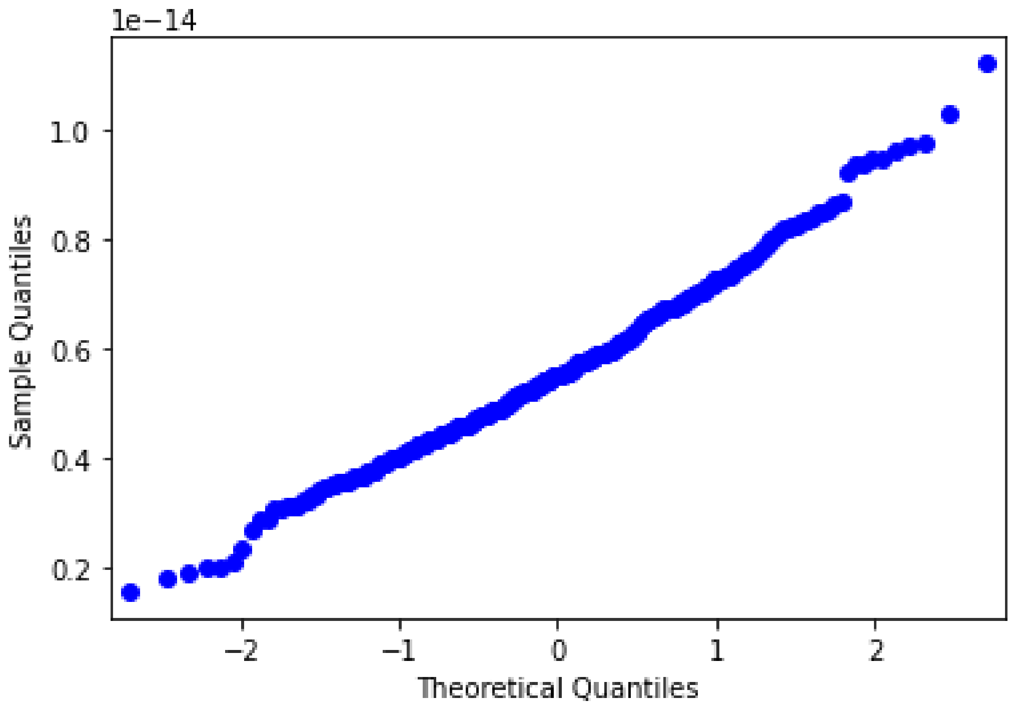
```
 MalePctNevMarr           4.996e-16    1.19e-14      0.042       0.967     -2.3e-14
2.4e-14
 FemalePctDiv            -8.327e-16    6.57e-14     -0.013       0.990     -1.3e-13
1.29e-13
 TotalPctDiv              1.707e-15    1.08e-13      0.016       0.987    -2.12e-13
2.16e-13
 PersPerFam               4.857e-16    2.79e-14      0.017       0.986    -5.44e-14
5.54e-14
 PctFam2Par              -4.441e-16    3.01e-14     -0.015       0.988    -5.97e-14
5.88e-14
 PctKids2Par              8.327e-16    2.86e-14      0.029       0.977    -5.55e-14
5.72e-14
 PctYoungKids2Par        -2.129e-16    9.07e-15     -0.023       0.981    -1.81e-14
1.77e-14
 PctTeen2Par             -3.313e-16    7.63e-15     -0.043       0.965    -1.54e-14
1.47e-14
 PctWorkMomYoungKids     -5.447e-16    8.23e-15     -0.066       0.947    -1.68e-14
1.57e-14
 PctWorkMom               -4.51e-17    8.97e-15     -0.005       0.996    -1.77e-14
1.76e-14
 NumIlleg                -6.939e-18    2.08e-14     -0.000       1.000     -4.1e-14
4.1e-14
 PctIlleg                 4.857e-17    9.67e-15      0.005       0.996     -1.9e-14
1.91e-14
 NumImmig                 3.608e-16    1.77e-14      0.020       0.984    -3.46e-14
3.53e-14
 PctImmigRecent           2.411e-16    5.66e-15      0.043       0.966    -1.09e-14
1.14e-14
 PctImmigRec5             1.388e-16    1.05e-14      0.013       0.989    -2.05e-14
2.08e-14
 PctImmigRec8            -1.471e-15    1.48e-14     -0.100       0.921    -3.06e-14
2.76e-14
 PctImmigRec10            1.596e-16    1.18e-14      0.014       0.989    -2.32e-14
2.35e-14
 PctRecentImmig          -2.134e-15    2.36e-14     -0.090       0.928    -4.87e-14
4.45e-14
 PctRecImmig5             4.163e-15    4.66e-14      0.089       0.929    -8.78e-14
9.61e-14
 PctRecImmig8            -2.554e-15    6.15e-14     -0.042       0.967    -1.24e-13
1.19e-13
 PctRecImmig10                    0     5.3e-14          0       1.000    -1.05e-13
1.05e-13
 PctSpeakEnglOnly                 0    1.24e-14          0       1.000    -2.45e-14
2.45e-14
 PctNotSpeakEnglWell     -3.053e-16    1.52e-14     -0.020       0.984    -3.03e-14
2.97e-14
 PctLargHouseFam          2.776e-16    4.14e-14      0.007       0.995    -8.14e-14
8.2e-14
```

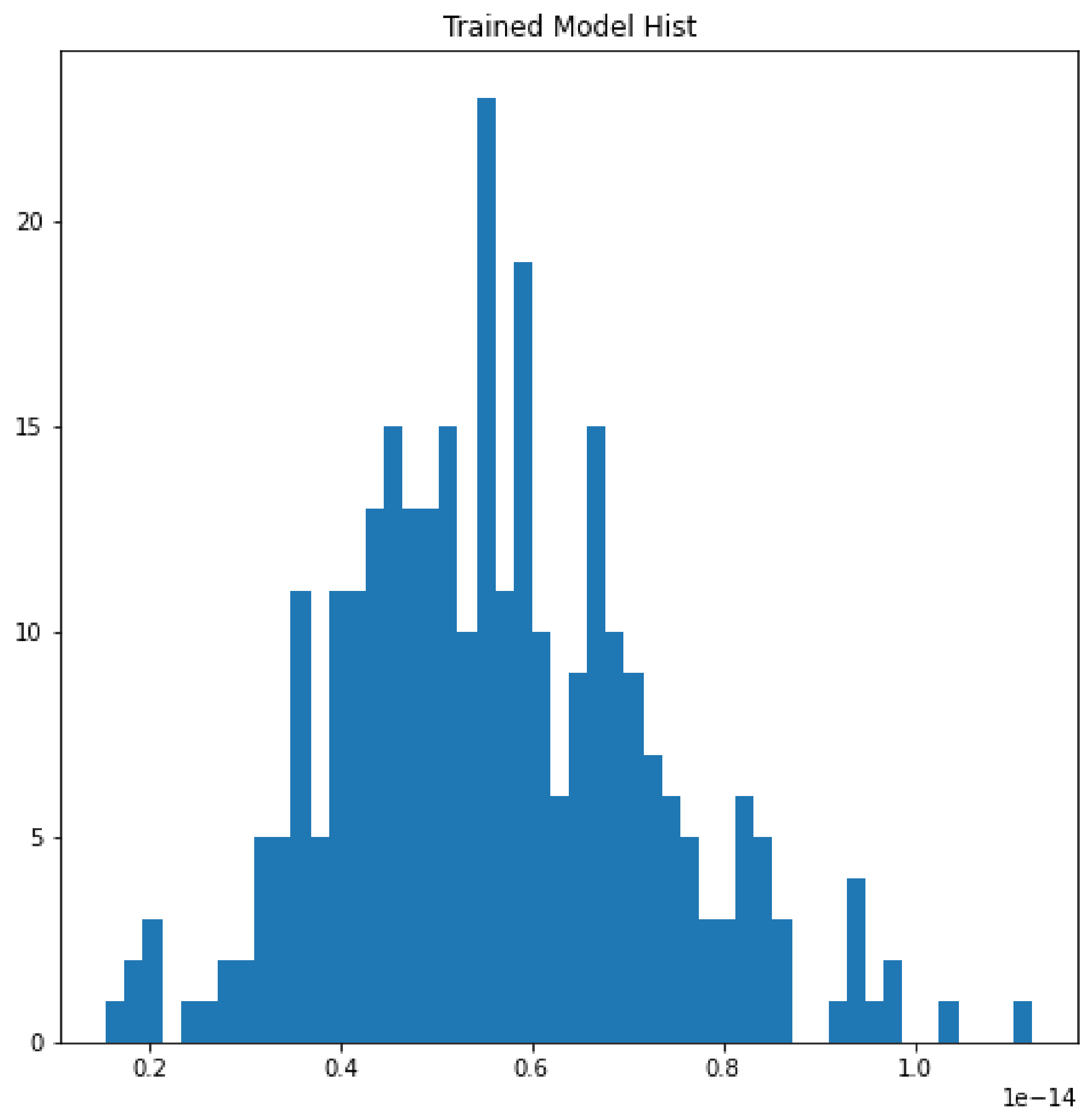
```
 PctLargHouseOccup      -3.053e-16   4.41e-14      -0.007       0.994   -8.73e-14
8.67e-14
 PersPerOccupHous       -2.824e-15   4.38e-14      -0.065       0.949   -8.91e-14
8.35e-14
 PersPerOwnOccHous       6.661e-16   2.77e-14       0.024       0.981   -5.39e-14
5.52e-14
 PersPerRentOccHous     -2.082e-16   1.38e-14      -0.015       0.988   -2.75e-14
2.71e-14
 PctPersOwnOccup        -5.135e-15   6.38e-14      -0.081       0.936   -1.31e-13
1.21e-13
 PctPersDenseHous        6.661e-16    1.6e-14       0.042       0.967    -3.1e-14
3.23e-14
 PctHousLess3BR          2.845e-16   1.02e-14       0.028       0.978   -1.98e-14
2.04e-14
 MedNumBR                2.359e-16   3.22e-15       0.073       0.942   -6.11e-15
6.58e-15
 HousVacant             -1.627e-15   1.33e-14      -0.122       0.903   -2.79e-14
2.47e-14
 PctHousOccup           -3.105e-16   5.28e-15      -0.059       0.953   -1.07e-14
1.01e-14
 PctHousOwnOcc            2.13e-15   6.61e-14       0.032       0.974   -1.28e-13
1.33e-13
 PctVacantBoarded       -2.776e-17   3.63e-15      -0.008       0.994   -7.18e-15
7.12e-15
 PctVacMore6Mos          4.059e-16   5.05e-15       0.080       0.936   -9.56e-15
1.04e-14
 MedYrHousBuilt          -4.51e-16    5.4e-15      -0.084       0.933   -1.11e-14
1.02e-14
 PctHousNoPhone         -2.533e-16   6.24e-15      -0.041       0.968   -1.26e-14
1.2e-14
 PctWOFullPlumb         -3.678e-16   3.54e-15      -0.104       0.917   -7.34e-15
6.61e-15
 OwnOccLowQuart          9.992e-16   4.48e-14       0.022       0.982   -8.73e-14
8.93e-14
 OwnOccMedVal           -3.775e-15   6.77e-14      -0.056       0.956   -1.37e-13
1.3e-13
 OwnOccHiQuart          -1.249e-16   3.24e-14      -0.004       0.997   -6.41e-14
6.38e-14
 RentLowQ               -4.163e-16    1.2e-14      -0.035       0.972   -2.42e-14
2.33e-14
 RentMedian              2.109e-15   3.03e-14       0.070       0.945   -5.77e-14
6.19e-14
 RentHighQ              -1.055e-15    1.5e-14      -0.070       0.944   -3.07e-14
2.86e-14
 MedRent                -2.498e-16   2.19e-14      -0.011       0.991   -4.34e-14
4.29e-14
 MedRentPctHousInc       7.286e-17   5.23e-15       0.014       0.989   -1.02e-14
1.04e-14
```

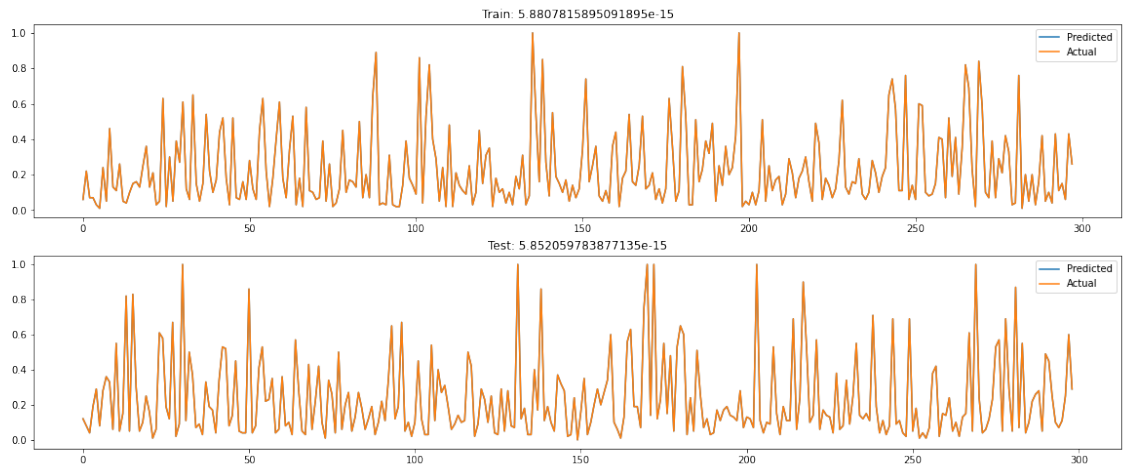
```
 MedOwnCostPctInc       -4.163e-17       6e-15     -0.007      0.994   -1.19e-14
1.18e-14
 MedOwnCostPctIncNoMtg  -4.163e-17     4.3e-15     -0.010      0.992   -8.52e-15
8.43e-15
 NumInShelters           4.788e-16    1.03e-14      0.047      0.963   -1.97e-14
2.07e-14
 NumStreet               1.041e-16    6.73e-15      0.015      0.988   -1.32e-14
1.34e-14
 PctForeignBorn          4.163e-17    1.77e-14      0.002      0.998   -3.49e-14
3.5e-14
 PctBornSameState       -3.539e-16    6.94e-15     -0.051      0.959    -1.4e-14
1.33e-14
 PctSameHouse85         -4.198e-16    1.08e-14     -0.039      0.969   -2.17e-14
2.08e-14
 PctSameCity85          -7.633e-17    6.81e-15     -0.011      0.991   -1.35e-14
1.34e-14
 PctSameState85          5.898e-16    7.67e-15      0.077      0.939   -1.45e-14
1.57e-14
 LandArea               -2.498e-16    1.22e-14     -0.020      0.984   -2.43e-14
2.38e-14
 PopDens                -2.203e-16    5.28e-15     -0.042      0.967   -1.06e-14
1.02e-14
 PctUsePubTrans         -9.021e-17    4.17e-15     -0.022      0.983   -8.31e-15
8.13e-15
 LemasPctOfficDrugUn    -1.752e-16    3.16e-15     -0.055      0.956    -6.4e-15
6.05e-15
 ViolentCrimesPerPop        1.0000    3.92e-15   2.55e+14      0.000       1.000
1.000
==============================================================================
Omnibus:                        7.425   Durbin-Watson:                   0.167
Prob(Omnibus):                  0.024   Jarque-Bera (JB):                7.232
Skew:                           0.362   Prob(JB):                       0.0269
Kurtosis:                       3.238   Cond. No.                     1.37e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.37e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[ ]:
```
## Looking at residuals
f = sm.qqplot(trained_model.resid)
fig = plt.figure(figsize=[8, 8])
ax = fig.add_subplot(1, 1, 1)
ax.hist(trained_model.resid, 50)
ax.set_title("Trained Model Hist")
```

```
Y_train_pred = trained_model.predict(X_train)
Y_test_pred = trained_model.predict(X_test)
rmse_train = np.sqrt(np.mean((Y_train_pred - Y_train)**2))
rmse_test = np.sqrt(np.mean((Y_test_pred - Y_test)**2))

fig = plt.figure(figsize=[20, 8])
ax = fig.add_subplot(2, 1, 1)
ax.plot(np.arange(len(Y_train_pred)), Y_train_pred, label='Predicted')
ax.plot(np.arange(len(Y_train_pred)), Y_train, label='Actual')
ax.set_title("Train: "+ str(rmse_train))
ax.legend()
ax = fig.add_subplot(2, 1, 2)
ax.plot(np.arange(len(Y_test_pred)), Y_test_pred, label='Predicted')
ax.plot(np.arange(len(Y_test_pred)), Y_test, label='Actual')
ax.set_title("Test: "+ str(rmse_test))
ax.legend();
```




Trained Model Hist
0
5
10
15
20
0.2
0.4
0.6
0.8
1.0
1e−14



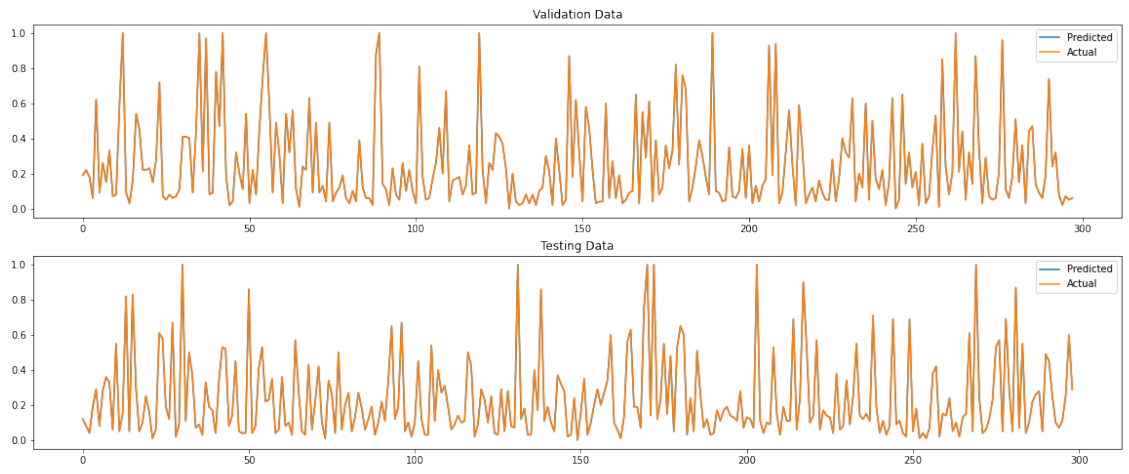
2.2 SKLearn's Regression Function

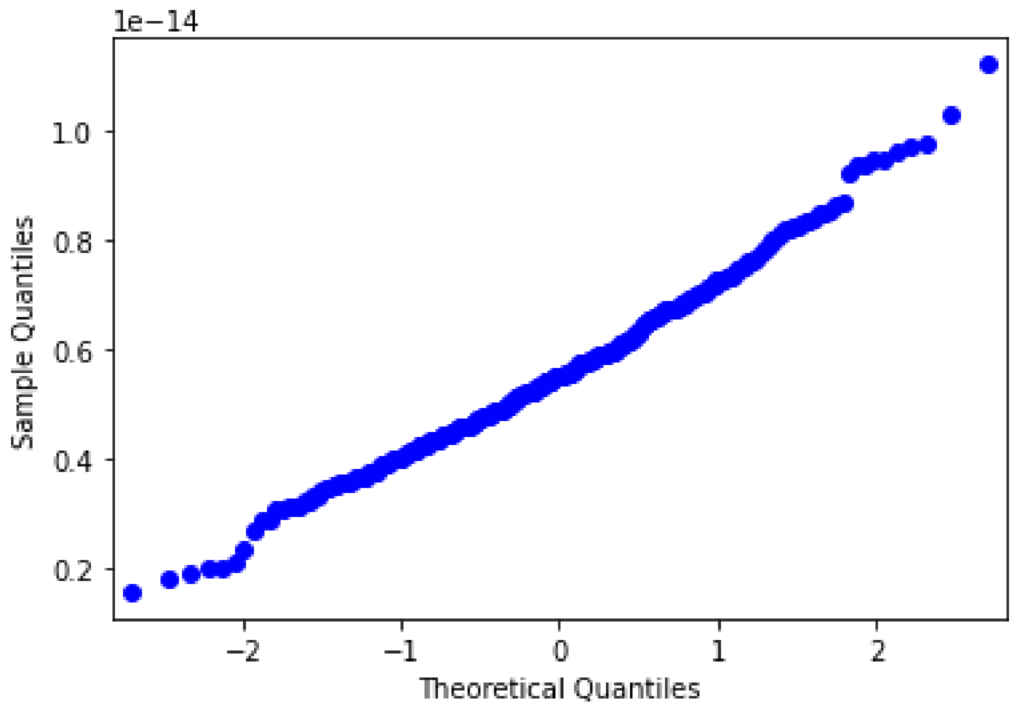
```
[ ]: ## SKLearn's Regression Funct
model_sk_linear = LinearRegression(fit_intercept = False).fit(X = X_test.
 ↪to_numpy(), y = Y_test.to_numpy())
print(model_sk_linear.coef_)
print(model_sk_linear.score(X_test.to_numpy(), Y_test.to_numpy()))

fig = plt.figure(figsize=[20, 8])
ax = fig.add_subplot(2, 1, 1)
ax.plot(model_sk_linear.predict(X_val.values), label='Predicted')
ax.plot(Y_val.to_numpy(), label='Actual')
ax.legend()
ax.set_title('Validation Data')
ax = fig.add_subplot(2, 1, 2)
ax.plot(model_sk_linear.predict(X_test.values), label='Predicted')
ax.plot(Y_test.to_numpy(), label='Actual')
ax.legend()
ax.set_title('Testing Data')

f = sm.qqplot(trained_model.resid)
fig = plt.figure(figsize=[20, 16])

residuals = model_sk_linear.predict(X_train) - Y_train
ax = fig.add_subplot(3, 2, 1)
stats.probplot(residuals, dist="norm", plot=ax)
ax.set_title("Linear")
ax = fig.add_subplot(3, 2, 2)
```

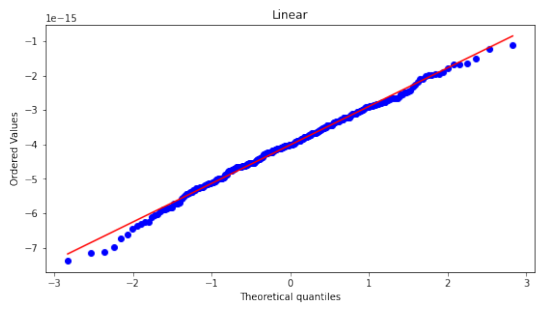
```python
ax.hist(residuals, 50)
ax.set_title("Linear")

print('Linear R**2 = ' + str(model_sk_linear.score(X_train, Y_train)))
```

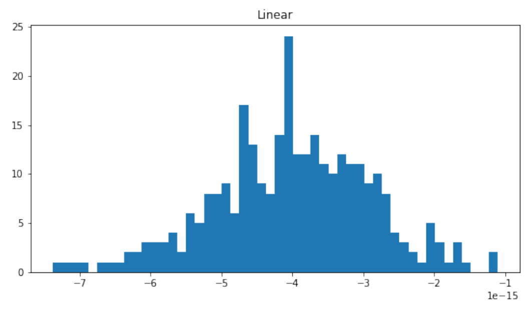
```
[-3.45948572e-15 -1.09287579e-14 -1.58206781e-15  2.52575738e-15
  1.88737914e-15  5.37764278e-16  6.38378239e-16 -8.88178420e-16
  1.11022302e-15 -1.33226763e-15  4.01068068e-15  1.20806143e-14
 -6.34908792e-16  3.99680289e-15  1.10328413e-15  7.94503352e-16
  1.11022302e-15 -4.71844785e-16  2.98372438e-16 -1.36696210e-15
 -1.47104551e-15  2.05391260e-15 -1.27675648e-15 -7.49400542e-16
 -3.95516953e-16 -5.55111512e-16 -4.16333634e-17 -3.69496100e-16
  4.96130914e-16  1.35308431e-16 -2.88657986e-15  1.05471187e-15
  1.41553436e-15  1.04083409e-15 -2.91433544e-16  4.71844785e-16
  1.20736754e-15  6.62664368e-16 -4.30558367e-15  2.94209102e-15
  3.13638004e-15  6.63358257e-15 -9.95037386e-15 -5.20417043e-16
 -2.51881849e-15  4.82253126e-16  1.42594270e-15  2.69229083e-15
 -9.99200722e-16 -8.88178420e-16 -1.03042574e-15 -5.72458747e-16
 -4.99600361e-16  1.29063427e-15  2.06085149e-15 -4.51028104e-15
  8.11850587e-16  4.45476989e-15 -9.47159018e-15  8.49320614e-15
 -3.59087760e-15 -1.64104841e-15 -1.48492330e-15  5.88418203e-15
 -5.45397061e-15  1.30104261e-15 -4.16333634e-17 -1.27675648e-15
 -6.34214903e-15  2.77555756e-16 -1.05471187e-15 -6.93889390e-17
 -1.69309011e-15 -1.01307851e-15  3.84414722e-15  4.51028104e-16
 -1.54043445e-15  1.79023463e-15 -8.46545056e-16  4.16333634e-16
  1.85615412e-15 -9.57567359e-16 -2.99066327e-15 -3.85108612e-16
 -2.40779618e-15 -2.50494070e-15  3.68802211e-15 -2.32452946e-16
  3.67761377e-16  4.26741975e-16  2.06779038e-15  1.31838984e-16
 -9.36750677e-16 -7.73686670e-16  6.90853624e-16  1.48492330e-15
  5.55111512e-17 -1.77809156e-16  7.07767178e-16 -1.59594560e-15
 -2.39391840e-16  1.00000000e+00]
1.0
Linear R**2 = 1.0
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has
feature names, but LinearRegression was fitted without feature names
  f"X has feature names, but {self.__class__.__name__} was fitted without"
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has
feature names, but LinearRegression was fitted without feature names
  f"X has feature names, but {self.__class__.__name__} was fitted without"
```


Validation Data
Predicted
Actual
Testing Data
Predicted
Actual


1e−14
Sample Quantiles
Theoretical Quantiles

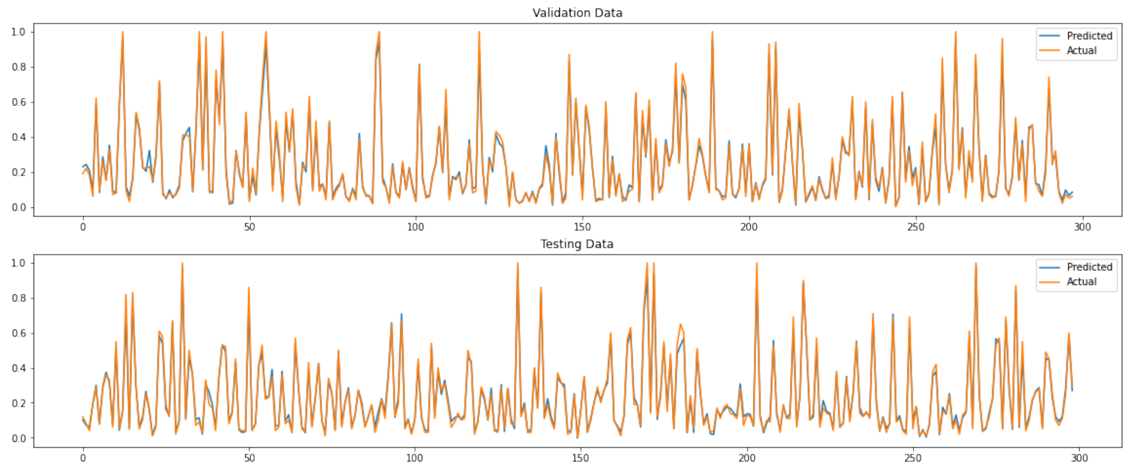

Linear
1e−15
Ordered Values
Theoretical quantiles

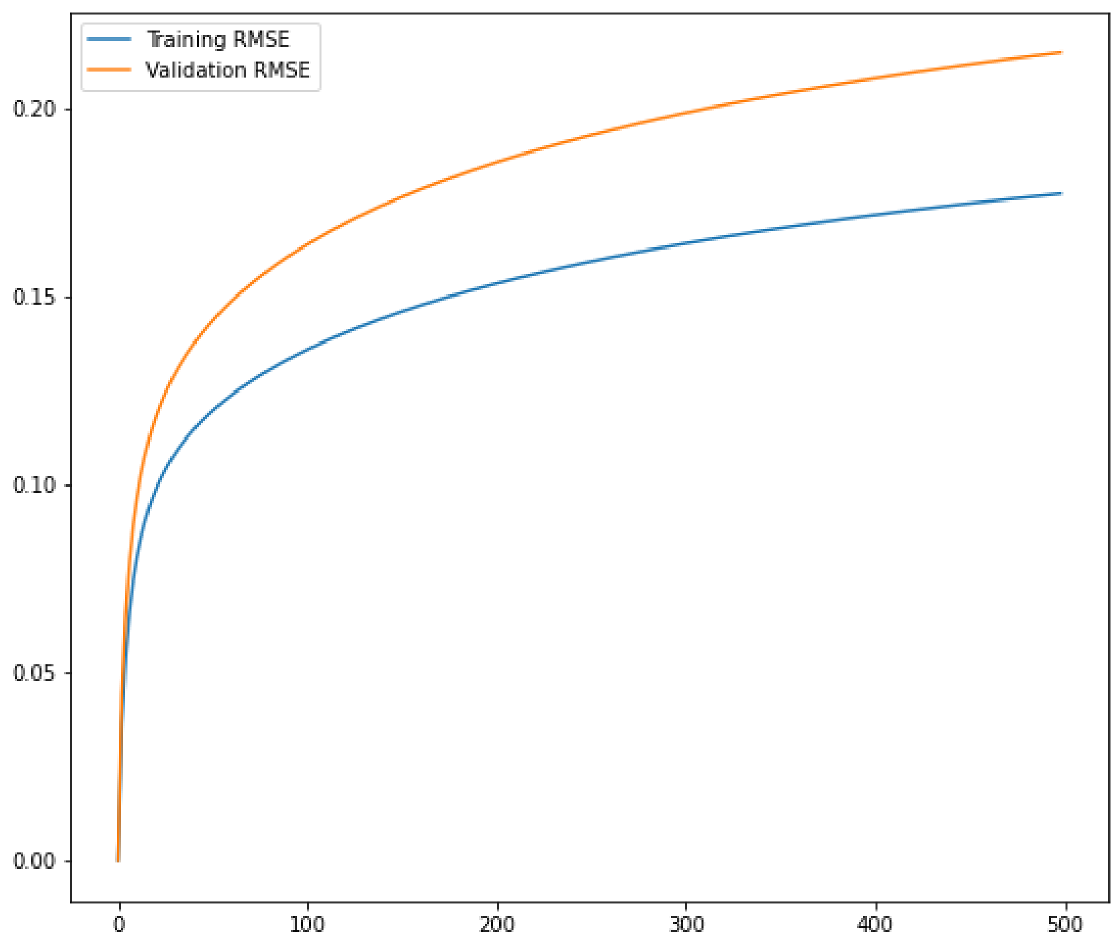

Linear
1e−15

2.3 Ridge Regression

```
[ ]: ## RIDGE REGRESSION ##

model_sk_ridge = Ridge(fit_intercept = False).fit(X = X_train.to_numpy(), y =␣
 ↪Y_train.to_numpy())

print(model_sk_ridge.coef_)
print(model_sk_ridge.score(X_train.to_numpy(), Y_train.to_numpy()))

fig = plt.figure(figsize=[20, 8])
ax = fig.add_subplot(2, 1, 1)
ax.plot(model_sk_ridge.predict(X_val.values), label='Predicted')
ax.plot(Y_val.to_numpy(), label='Actual')
ax.legend()
ax.set_title('Validation Data')
ax = fig.add_subplot(2, 1, 2)
ax.plot(model_sk_ridge.predict(X_test.values), label='Predicted')
ax.plot(Y_test.to_numpy(), label='Actual')
ax.legend()
ax.set_title('Testing Data')


lambdas = np.arange(0, 500, 2)
rmse_train = []
rmse_validation = []
for l in lambdas:
    model_sk_ridge = Ridge(fit_intercept=False, alpha=l).fit(X = X_train.
 ↪to_numpy(), y = Y_train.to_numpy())
    rmse_train.append(np.sqrt(np.mean((model_sk_ridge.predict(X_train.
 ↪to_numpy()) - Y_train.to_numpy())**2)))
    rmse_validation.append(np.sqrt(np.mean((model_sk_ridge.predict(X_val.
 ↪to_numpy()) - Y_val.to_numpy())**2)))

fig = plt.figure(figsize=[20, 8])
ax = fig.add_subplot(1, 2, 1)
ax.plot(lambdas, rmse_train, label='Training RMSE')
ax.plot(lambdas, rmse_validation, label='Validation RMSE')
ax.legend();
```

```
[ 1.33867815e-02  1.73786447e-03 -1.56660803e-02  1.81700213e-02
 -2.42015002e-03  1.55896186e-03 -2.54652110e-02 -5.38336018e-03
  1.09156108e-02  1.09901875e-02  1.64499712e-02  1.23250570e-03
  7.77049772e-03  7.31514365e-03 -4.56800898e-03  3.79664242e-03
 -1.42599584e-02  1.58015262e-02  1.32355289e-02 -1.43886065e-02
  3.41238665e-03  2.59890972e-03  4.70531900e-03 -1.60152661e-03
 -7.83645970e-03  1.67954166e-02  2.56255830e-03  1.06967314e-02
 -2.17449714e-03 -1.28077893e-02 -7.14587908e-03  7.09269167e-03
  7.40679155e-03 -1.41698174e-02  2.76361935e-02 -1.55838514e-02
 -1.59821310e-03  4.16584748e-03 -4.21711937e-03  4.96973236e-04
  2.56865452e-03  3.44226711e-03  1.70247318e-03  1.09012684e-02
 -1.43216578e-02 -1.72228739e-02 -1.40695694e-02 -3.80014553e-03
  1.15512768e-02 -1.49896945e-02  1.02676800e-02  2.66225740e-02
 -9.40259759e-03 -7.78924251e-03 -6.70600403e-03 -3.34991155e-03
  1.69977259e-02 -7.74206138e-03 -5.41958763e-03  6.56849455e-03
  7.21850743e-03  1.02208432e-02  6.02738617e-03  9.96233977e-03
  3.57698813e-03  1.48326150e-02 -9.67077573e-03  1.49465142e-02
 -1.49777742e-02  2.34958516e-02  1.00686290e-02 -9.48059022e-04
  1.38589361e-02 -9.20119223e-03 -2.51682366e-03  1.84857153e-02
 -9.51750001e-03  4.29380901e-03 -4.56909080e-04  8.88041855e-03
 -2.84890391e-03 -6.51071406e-03 -8.12929051e-03 -1.98749361e-02
 -3.20567503e-04  1.18530074e-02  4.61821558e-03  2.46891833e-02
  1.20558714e-03 -8.98865552e-03  2.70959938e-03  1.02092144e-02
  6.04930182e-03 -4.64536287e-03 -2.37866487e-03  2.93147263e-03
  8.43480771e-03 -1.20415276e-02 -1.13445206e-02 -2.02989277e-03
  1.82272842e-02  8.08934936e-01]
0.9894176945475675
```





2.4 Standardisation

[ ]:
```
## Poly test
poly_transform = PolynomialFeatures(4)

X_test_poly = poly_transform.fit_transform(X_test.iloc[:,1:-1])

# fig = plt.figure(figsize=[20, 8])
# ax = fig.add_subplot(1, 1, 1)
# ax.boxplot(X_test_poly[:,1:100]);
```

[ ]:
```
## Standardisation test

mu = np.mean(X_test.iloc[:, 1:-1])
sigma = np.std(X_test.iloc[:, 1:-1])
```

```
X_test_poly = (X_test.iloc[:, 1:-1] - mu) / sigma

# fig = plt.figure(figsize=[20, 8])
# ax = fig.add_subplot(1, 1, 1)
# ax.boxplot(X_test_poly);
```

[ ]:
```
poly_transform = PolynomialFeatures(4)

X_train_poly = poly_transform.fit_transform(X_train.iloc[:,1:-1])

# fig = plt.figure(figsize=[20, 8])
# ax = fig.add_subplot(1, 1, 1)
# ax.boxplot(X_train_poly[:,1:100]);
```

[ ]:
```
## Standardisation train

mu = np.mean(X_train.iloc[:, 1:-1])
sigma = np.std(X_train.iloc[:, 1:-1])
X_train_poly = (X_train.iloc[:, 1:-1] - mu) / sigma

# fig = plt.figure(figsize=[20, 8])
# ax = fig.add_subplot(1, 1, 1)
# ax.boxplot(X_train_poly);
```

[ ]:
```
poly_transform = PolynomialFeatures(4)

X_val_poly = poly_transform.fit_transform(X_val.iloc[:,1:-1])

# fig = plt.figure(figsize=[20, 8])
# ax = fig.add_subplot(1, 1, 1)
# ax.boxplot(X_val_poly[:,1:100]);
```

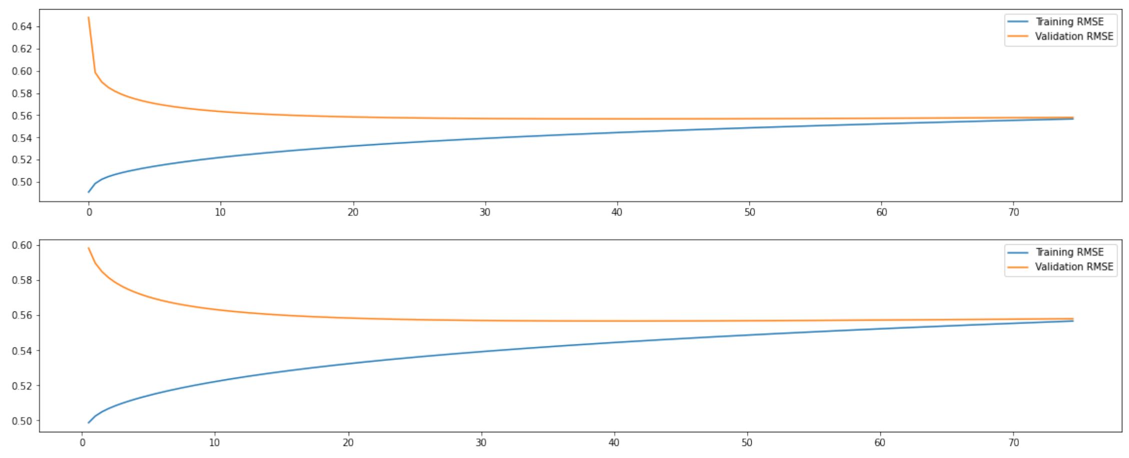
[ ]:
```
## Standardisation val

mu = np.mean(X_val.iloc[:, 1:-1])
sigma = np.std(X_val.iloc[:, 1:-1])
X_val_poly = (X_val.iloc[:, 1:-1] - mu) / sigma

# fig = plt.figure(figsize=[20, 8])
# ax = fig.add_subplot(1, 1, 1)
# ax.boxplot(X_val_poly);
```

[ ]:
```
## Standardise Y values
## Training Samples ##
num_samples = test.shape[0]
training_samples = train.shape[0]
#int(num_samples*0.7)
```

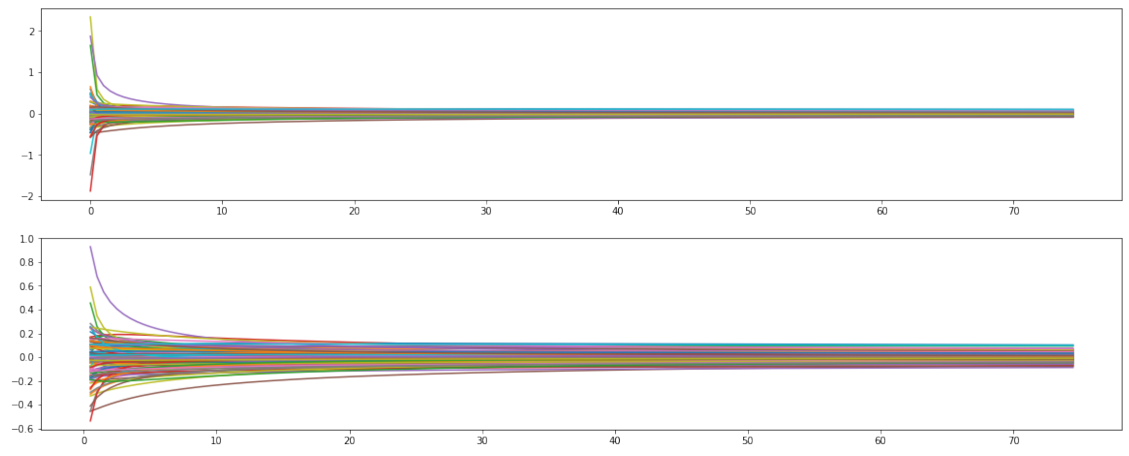
```
validation_samples = val.shape[0]
#int(num_samples*0.15)

## Test
Y_mu_test = np.mean(Y_test)
Y_sigma_test = np.std(Y_test)
Y_test_poly = (Y_test - Y_mu_test) / Y_sigma_test

## Train
Y_mu = np.mean(Y_train)
Y_sigma = np.std(Y_train)
Y_train_poly = (Y_train - Y_mu) / Y_sigma

## Val
Y_mu = np.mean(Y_val)
Y_sigma = np.std(Y_val)
Y_val_poly = (Y_val - Y_mu) / Y_sigma
```

2.5 Glenn Ridge With Standardised Data

[ ]:
```
lambdas = np.arange(0, 75, 0.5)
rmse_train = []
rmse_validation = []
coeffs = []
for l in lambdas:
    trained_model_poly_ridge = Ridge(fit_intercept=False, alpha=l).
 ↪fit(X_train_poly, Y_train_poly)
    coeffs.append(trained_model_poly_ridge.coef_)
    rmse_train.append(np.sqrt(np.mean((trained_model_poly_ridge.
 ↪predict(X_train_poly) - Y_train_poly)**2)))
    rmse_validation.append(np.sqrt(np.mean((trained_model_poly_ridge.
 ↪predict(X_val_poly) - Y_val_poly)**2)))

fig = plt.figure(figsize=[20, 8])
ax = fig.add_subplot(2, 1, 1)
ax.plot(lambdas, rmse_train, label='Training RMSE')
ax.plot(lambdas, rmse_validation, label='Validation RMSE')
ax.legend();
ax = fig.add_subplot(2, 1, 2)
ax.plot(lambdas[1:], rmse_train[1:], label='Training RMSE')
ax.plot(lambdas[1:], rmse_validation[1:], label='Validation RMSE')
ax.legend();
```



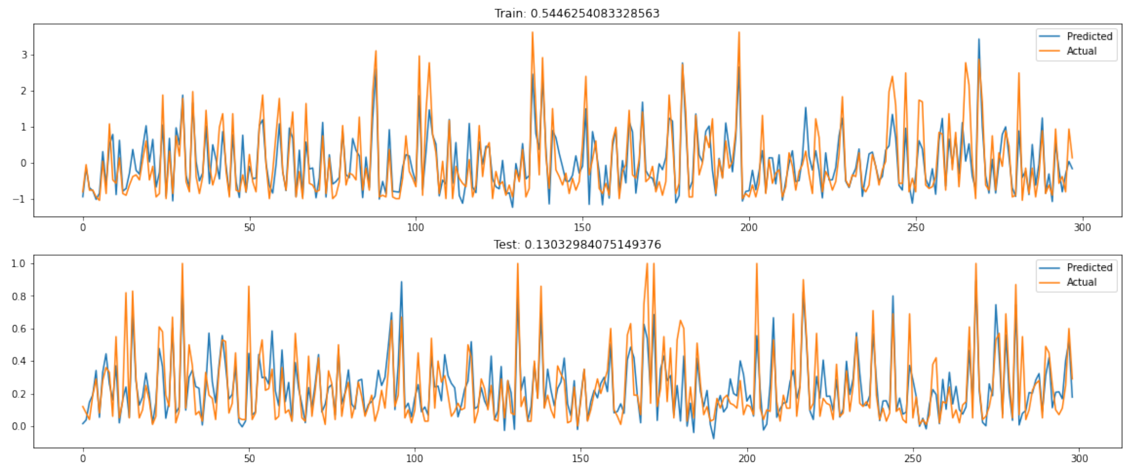
```
[ ]: fig = plt.figure(figsize=[20, 8])
     ax = fig.add_subplot(2, 1, 1)
     ax.plot(lambdas, coeffs);
     coeffs = np.array(coeffs)
     ax = fig.add_subplot(2, 1, 2)
     ax.plot(lambdas[1:400], coeffs[1:400,:]);
```



```
[ ]: ## Best fit model
     best_lambda = lambdas[np.argmin(rmse_validation)]
     print(best_lambda)
     trained_model_poly_ridge = Ridge(fit_intercept=False, alpha=best_lambda).
      ↪fit(X_train_poly, Y_train_poly)

     Y_train_poly_pred = trained_model_poly_ridge.predict(X_train_poly)
     Y_test_poly_pred = trained_model_poly_ridge.predict(X_test_poly)
     rmse_train = np.sqrt(np.mean((Y_train_poly_pred - Y_train_poly)**2))
```

```
rmse_test = np.sqrt(np.mean(((Y_test_poly_pred*Y_sigma_test + Y_mu_test) -␣
↪(Y_test_poly*Y_sigma_test + Y_mu_test))**2))

fig = plt.figure(figsize=[20, 8])
ax = fig.add_subplot(2, 1, 1)
ax.plot(np.arange(len(Y_train_pred)), Y_train_poly_pred, label='Predicted')
ax.plot(np.arange(len(Y_train_pred)), Y_train_poly, label='Actual')
ax.set_title("Train: "+ str(rmse_train))
ax.legend()
ax = fig.add_subplot(2, 1, 2)
ax.plot(np.arange(len(Y_test_pred)), Y_test_poly_pred*Y_sigma_test + Y_mu_test,␣
↪label='Predicted')
ax.plot(np.arange(len(Y_test_pred)), Y_test_poly*Y_sigma_test + Y_mu_test,␣
↪label='Actual')
ax.set_title("Test: "+ str(rmse_test))
ax.legend();
```

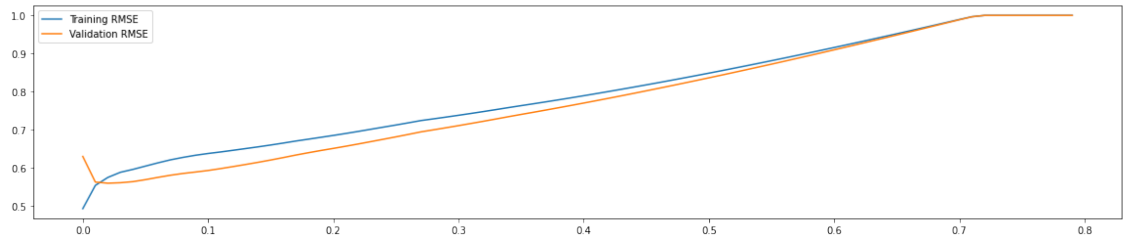
```
40.5
```



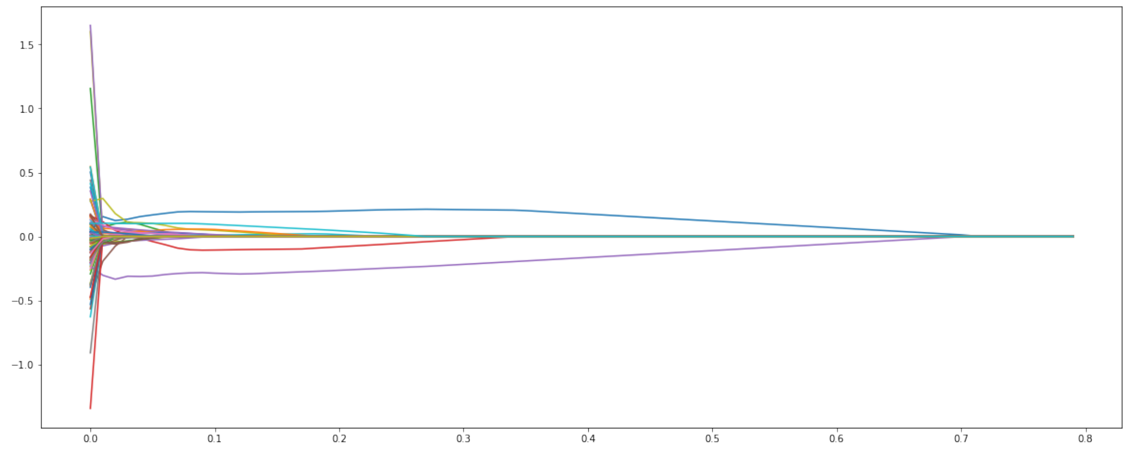
2.6 LASSO Regression

```
[ ]: ## LASSO REGRESSION ##

lambdas = np.arange(0.0, 0.8, 0.01)
rmse_train = []
rmse_validation = []
coeffs = []
for l in lambdas:
    trained_model_poly_lasso = Lasso(fit_intercept=False, alpha=l).
↪fit(X_train_poly, Y_train_poly)
    coeffs.append(trained_model_poly_lasso.coef_)
    rmse_train.append(np.sqrt(np.mean((trained_model_poly_lasso.
↪predict(X_train_poly) - Y_train_poly)**2)))
    rmse_validation.append(np.sqrt(np.mean((trained_model_poly_lasso.
↪predict(X_val_poly) - Y_val_poly)**2)))

fig = plt.figure(figsize=[20, 4])
ax = fig.add_subplot(1, 1, 1)
ax.plot(lambdas, rmse_train, label='Training RMSE')
ax.plot(lambdas, rmse_validation, label='Validation RMSE')
ax.legend();
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
With alpha=0, this algorithm does not converge well. You are advised to use the
LinearRegression estimator

/usr/local/lib/python3.7/dist-
packages/sklearn/linear_model/_coordinate_descent.py:648: UserWarning:
Coordinate descent with no regularization may lead to unexpected results and is
discouraged.
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/usr/local/lib/python3.7/dist-
packages/sklearn/linear_model/_coordinate_descent.py:648: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 3.606e+01, tolerance: 2.980e-02 Linear regression models with null weight
for the l1 regularization term are more efficiently fitted using one of the
solvers implemented in sklearn.linear_model.Ridge/RidgeCV instead.
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
```



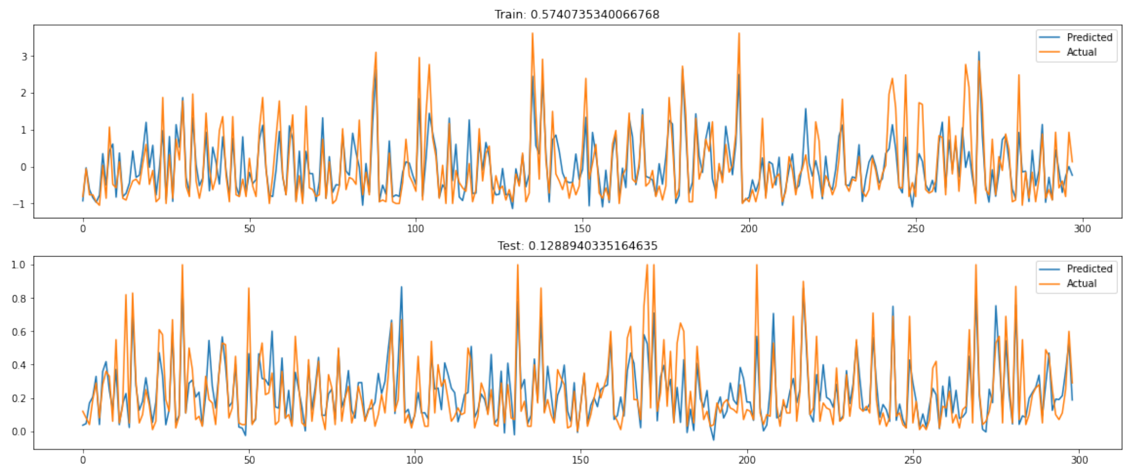
```
[ ]: fig = plt.figure(figsize=[20, 8])
     ax = fig.add_subplot(1, 1, 1)
     ax.plot(lambdas, coeffs);
```



```
[ ]: best_lambda = lambdas[np.argmin(rmse_validation)]
     print(best_lambda)
```

```
0.02
```

```
[ ]: trained_model_poly_lasso = Lasso(fit_intercept=False, alpha=best_lambda).
      ↪fit(X_train_poly, Y_train_poly)

     Y_train_poly_pred = trained_model_poly_lasso.predict(X_train_poly)
     Y_test_poly_pred = trained_model_poly_lasso.predict(X_test_poly)
     rmse_train = np.sqrt(np.mean((Y_train_poly_pred - Y_train_poly)**2))
     rmse_test = np.sqrt(np.mean(((Y_test_poly_pred*Y_sigma_test + Y_mu_test) -␣
      ↪(Y_test_poly*Y_sigma_test + Y_mu_test))**2))

     fig = plt.figure(figsize=[20, 8])
     ax = fig.add_subplot(2, 1, 1)
     ax.plot(np.arange(len(Y_train_pred)), Y_train_poly_pred, label='Predicted')
     ax.plot(np.arange(len(Y_train_pred)), Y_train_poly, label='Actual')
     ax.set_title("Train: "+ str(rmse_train))
     ax.legend()
     ax = fig.add_subplot(2, 1, 2)
     ax.plot(np.arange(len(Y_test_pred)), Y_test_poly_pred*Y_sigma_test + Y_mu_test,␣
      ↪label='Predicted')
     ax.plot(np.arange(len(Y_test_pred)), Y_test_poly*Y_sigma_test + Y_mu_test,␣
      ↪label='Actual')
     ax.set_title("Test: "+ str(rmse_test))
     ax.legend();
```



```
[ ]: sum(trained_model_poly_lasso.coef_ != 0)
```

```
[ ]: 30
```

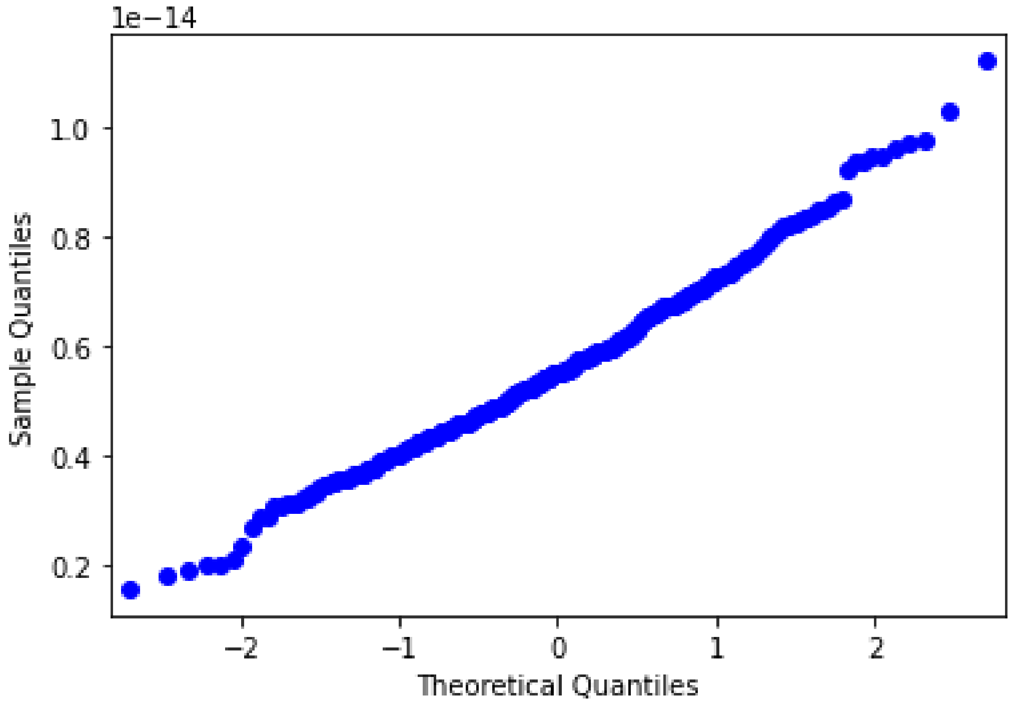
2.7 Compare Models

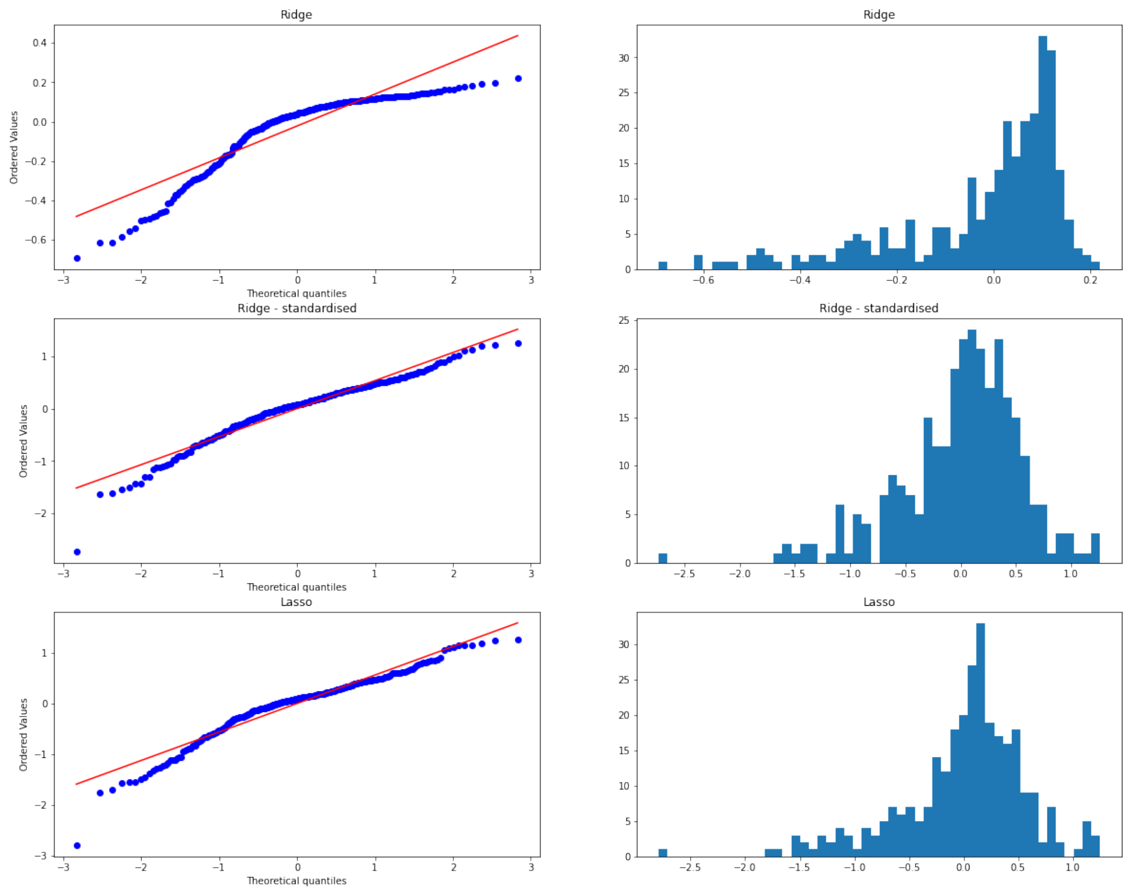
```
[ ]: f = sm.qqplot(trained_model.resid)
     fig = plt.figure(figsize=[20, 16])



     residuals = model_sk_ridge.predict(X_train) - Y_train
     ax = fig.add_subplot(3, 2, 1)
     stats.probplot(residuals, dist="norm", plot=ax)
     ax.set_title("Ridge")
     ax = fig.add_subplot(3, 2, 2)
     ax.hist(residuals, 50)
     ax.set_title("Ridge")

     residuals = trained_model_poly_ridge.predict(X_train_poly) - Y_train_poly
     ax = fig.add_subplot(3, 2, 3)
     stats.probplot(residuals, dist="norm", plot=ax)
     ax.set_title("Ridge - standardised")
     ax = fig.add_subplot(3, 2, 4)
     ax.hist(residuals, 50)
     ax.set_title("Ridge - standardised")

     residuals = trained_model_poly_lasso.predict(X_train_poly) - Y_train_poly
     ax = fig.add_subplot(3, 2, 5)
     stats.probplot(residuals, dist="norm", plot=ax)
     ax.set_title("Lasso")
```

```
ax = fig.add_subplot(3, 2, 6)
ax.hist(residuals, 50)
ax.set_title("Lasso")
print('Ridge R**2 = ' + str(trained_model_poly_lasso.score(X_train_poly,␣
 ↪Y_train_poly)))
print('Lasso R**2 = ' + str(trained_model_poly_ridge.score(X_train_poly,␣
 ↪Y_train_poly)))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has
feature names, but Ridge was fitted without feature names
  f"X has feature names, but {self.__class__.__name__} was fitted without"
```

```
Ridge R**2 = 0.670439577553085
Lasso R**2 = 0.7033831645982696
```





3 Q2: Classification

Using the provided data as-is, you are to train three multi-class classifiers to classify land type from the spectral data. These classifiers are to be: 1. A K-Nearest Neighbours Classifier; 2. A Random Forest; and 3. An ensemble of Support Vector Machines. Model hyper-parameters should be selected using a grid search operating over the validation set. The resultant models are to be evaluated on the testing set and compared.

```
[ ]: ## Load the data
     ##info = pd.read_csv('/content/drive/My Drive/Colab Notebooks/
      ↪CAB420_Assessment_1A_Data/Communities_Info.txt')
     test = pd.read_csv('/content/drive/My Drive/Colab Notebooks/
      ↪CAB420_Assessment_1A_Data/Q2/testing.csv')
     train = pd.read_csv('/content/drive/My Drive/Colab Notebooks/
      ↪CAB420_Assessment_1A_Data/Q2/training.csv')
```

```
val = pd.read_csv('/content/drive/My Drive/Colab Notebooks/
 ↪CAB420_Assessment_1A_Data/Q2/validation.csv')

test.head()
```

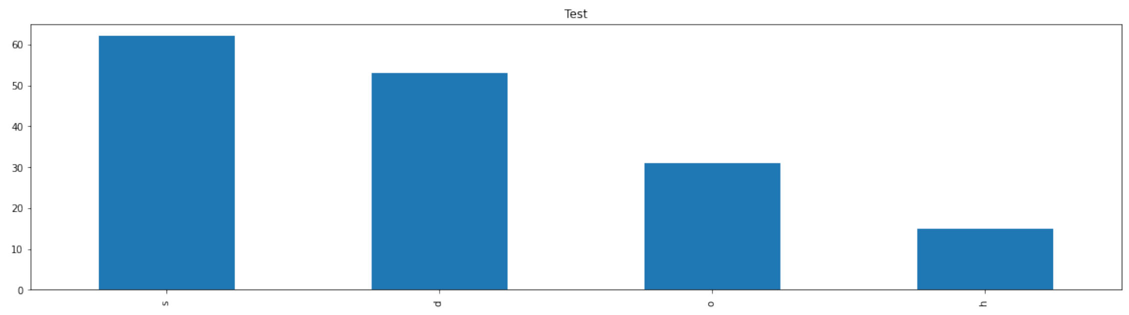
```
[ ]:   class  b1  b2  b3   b4  b5   b6   b7  b8  b9  …  pred_minus_obs_H_b9  \
     0     d  67  51  68  115  69  111  136  31  67  …                -9.17
     1     s  67  28  51   99  50   97   82  26  59  …                -2.25
     2     s  63  26  50   95  49   91   81  26  57  …                -0.44
     3     d  63  42  63   97  66  108  111  28  59  …                -2.34
     4     s  46  27  50   83  51   90   76  26  56  …                 1.25

        pred_minus_obs_S_b1  pred_minus_obs_S_b2  pred_minus_obs_S_b3  \
     0               -18.27                -1.80                -6.32
     1               -20.13                -2.11                -6.35
     2               -17.64                -1.81                -4.70
     3               -20.20                -1.89                -5.47
     4               -18.62                -2.17                -7.11

        pred_minus_obs_S_b4  pred_minus_obs_S_b5  pred_minus_obs_S_b6  \
     0               -20.88                -1.63                -6.13
     1               -21.94                -1.22                -6.13
     2               -19.39                -0.65                -5.01
     3               -21.65                -0.99                -5.71
     4               -21.12                -1.56                -6.35

        pred_minus_obs_S_b7  pred_minus_obs_S_b8  pred_minus_obs_S_b9
     0               -22.56                -5.53                -8.11
     1               -22.20                -3.41                -6.57
     2               -20.89                -3.96                -6.85
     3               -22.19                -3.41                -6.52
     4               -22.19                -4.45                -7.32

     [5 rows x 28 columns]
```

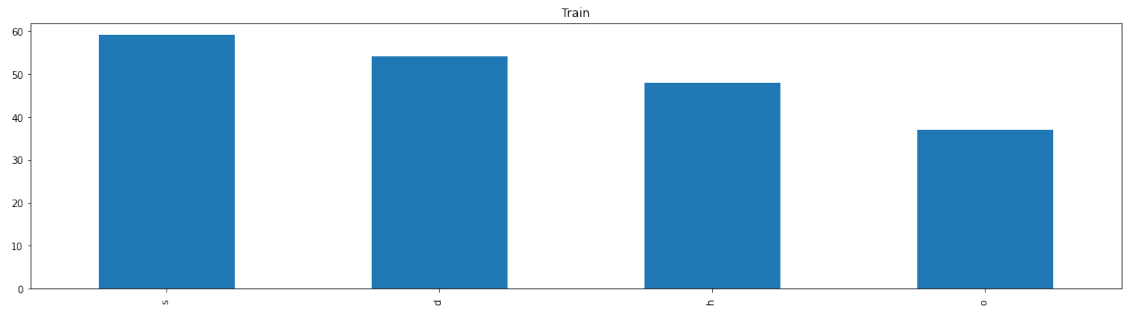
```
[ ]: # make class a categorical for each set
     test['class'] = pd.Categorical(test['class'])
     train['class'] = pd.Categorical(train['class'])
     val['class'] = pd.Categorical(val['class'])

     # s = ‘Sugi’ forest
     # h = ‘Hinoki’ forest
     # d = ‘Mixed deciduous’ forest
     # o = ‘Other’ non-forest land
     print("Where:\ns = ‘Sugi’ forest\nh = ‘Hinoki’ forest\nd = ‘Mixed deciduous’␣
      ↪forest\no = ‘Other’ non-forest land")
```

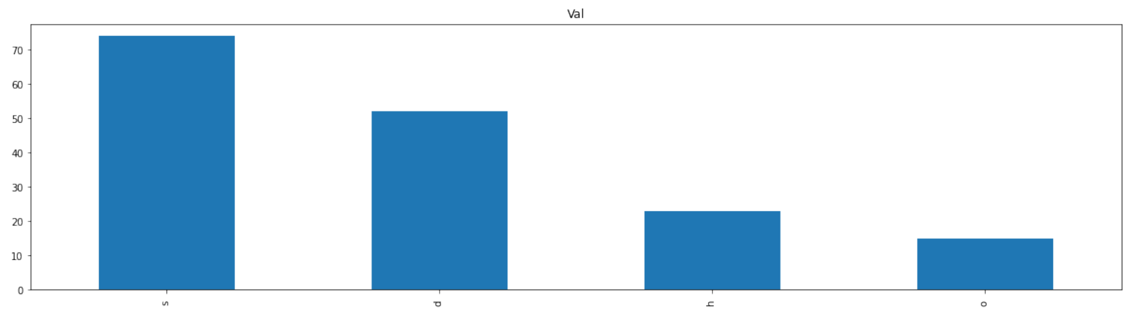
```python
# and draw a picture
counts = test['class'].value_counts()
fig = plt.figure(figsize=[20, 5])
ax = fig.add_subplot(1, 1, 1)
counts.plot(kind='bar')
ax.set_title("Test")

counts = train['class'].value_counts()
fig = plt.figure(figsize=[20, 5])
ax = fig.add_subplot(1, 1, 1)
counts.plot(kind='bar')
ax.set_title("Train")

counts = val['class'].value_counts()
fig = plt.figure(figsize=[20, 5])
ax = fig.add_subplot(1, 1, 1)
counts.plot(kind='bar')
ax.set_title("Val")
```

```
Where:
s = 'Sugi' forest
h = 'Hinoki' forest
d = 'Mixed deciduous' forest
o = 'Other' non-forest land
```

```
[ ]: Text(0.5, 1.0, 'Val')
```

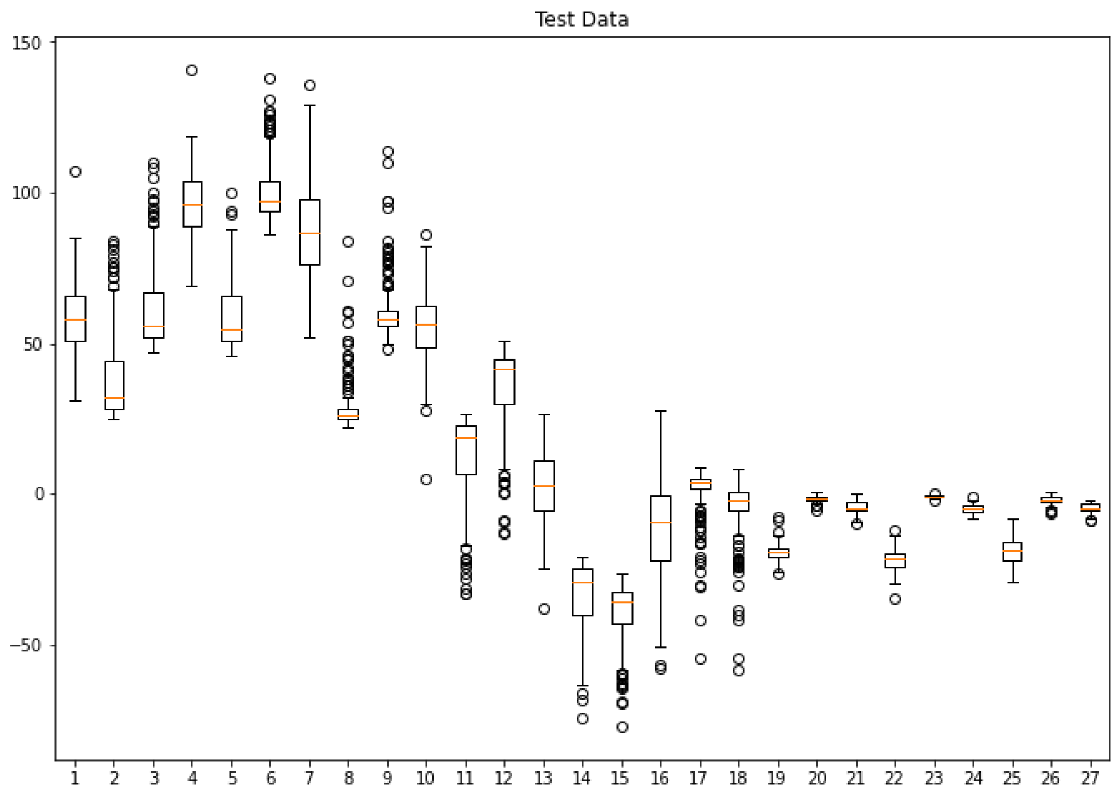


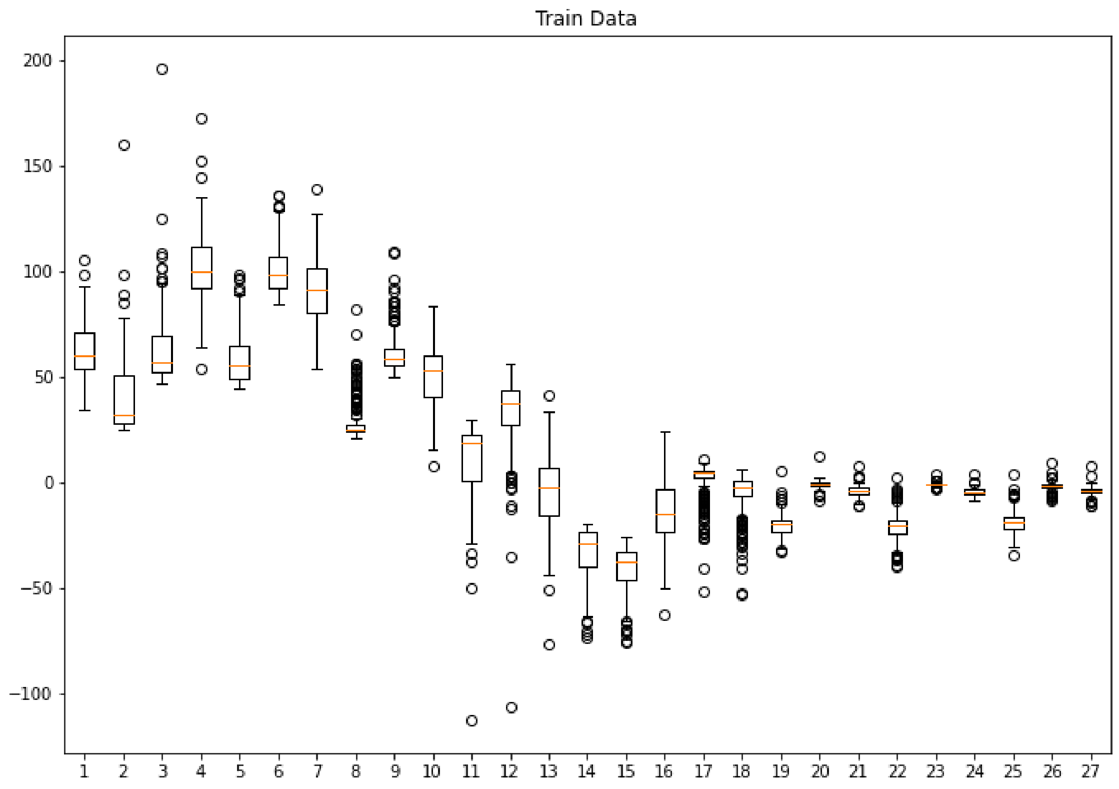


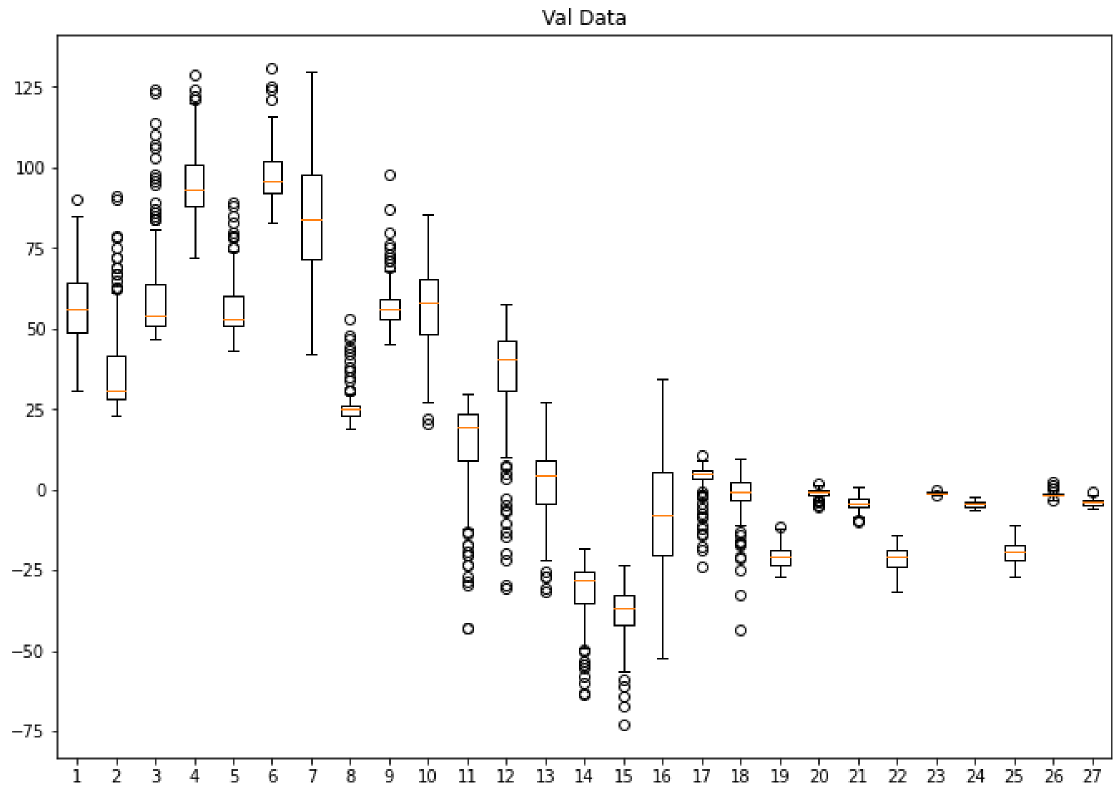


```
[ ]: ## Set tree categories in each set
     # test.iloc[:, 0] = test.iloc[:, 0].astype("category")
     # train.iloc[:, 0] = train.iloc[:, 0].astype("category")
     # val.iloc[:, 0] = val.iloc[:, 0].astype("category")

     ## Get variables for each set
     X_test = test.iloc[:, 1:].to_numpy()
     Y_test = test.iloc[:, 0]

     X_train = train.iloc[:, 1:].to_numpy()
     Y_train = train.iloc[:, 0]

     X_val = val.iloc[:, 1:].to_numpy()
     Y_val = val.iloc[:, 0]

     # plot box plot for the data
     fig = plt.figure(figsize=[25, 8])
     ax = fig.add_subplot(1, 2, 1)
     ax.boxplot(X_test)
     ax.set_title('Test Data')

     fig = plt.figure(figsize=[25, 8])
     ax = fig.add_subplot(1, 2, 1)
     ax.boxplot(X_train)
     ax.set_title('Train Data')

     fig = plt.figure(figsize=[25, 8])
     ax = fig.add_subplot(1, 2, 1)
     ax.boxplot(X_val)
     ax.set_title('Val Data')
```

```
[ ]: Text(0.5, 1.0, 'Val Data')
```

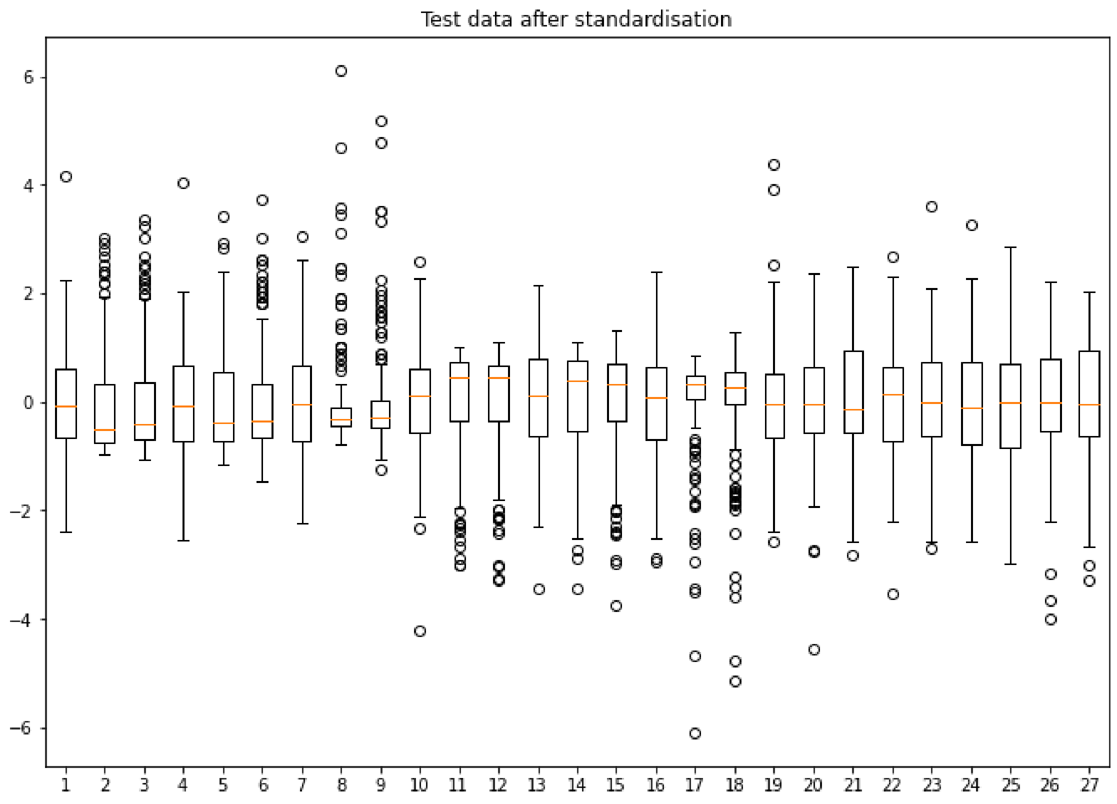

Test Data
150
100
50
0
−50
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

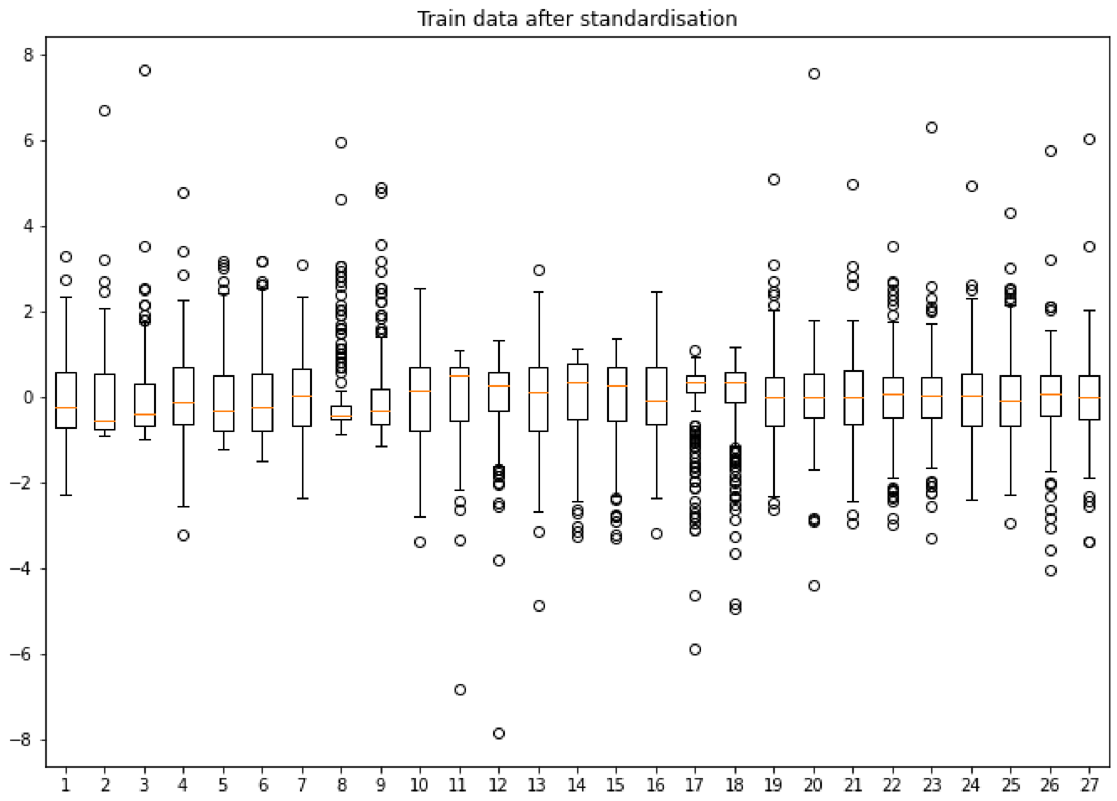

Train Data
200
150
100
50
0
−50
−100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

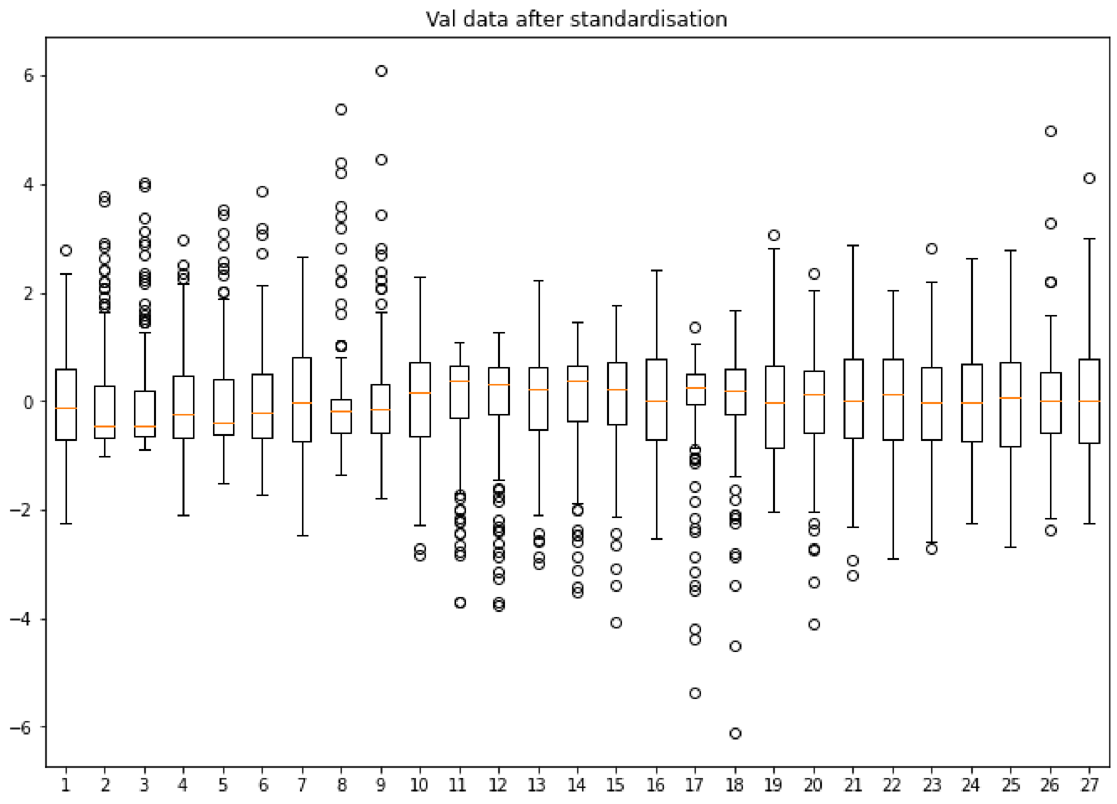


```
[ ]: ## Standardise the test data
     mu = np.mean(X_test, 0)
     sigma = np.std(X_test, 0)
     X_test = (X_test - mu) / sigma

     # box plot after standardisation
     fig = plt.figure(figsize=[25, 8])
     ax = fig.add_subplot(1, 2, 1)
     ax.boxplot(X_test)
     ax.set_title('Test data after standardisation');


     ## Standardise the train data
     mu = np.mean(X_train, 0)
     sigma = np.std(X_train, 0)
     X_train = (X_train - mu) / sigma

     # box plot after standardisation
     fig = plt.figure(figsize=[25, 8])
```

```python
ax = fig.add_subplot(1, 2, 1)
ax.boxplot(X_train)
ax.set_title('Train data after standardisation');


## Standardise the val data
mu = np.mean(X_val, 0)
sigma = np.std(X_val, 0)
X_val = (X_val - mu) / sigma

# box plot after standardisation
fig = plt.figure(figsize=[25, 8])
ax = fig.add_subplot(1, 2, 1)
ax.boxplot(X_val)
ax.set_title('Val data after standardisation');
```




Train data after standardisation
8
6
4
2
0
−2
−4
−6
−8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27


Val data after standardisation
6
4
2
0
−2
−4
−6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

3.1 Evaluation Function

[ ]:
```
# function to do our eval for us, this is quite simple and will
# - create a figure
# - draw a confusion matrix for the trainign data in a sub-fig on the left
# - draw a confusion matrix for the testing data in a sub-fig on the right
# - compute the overall classification accuracy on the testing data
# this has simply been created as we're going to do this for each test that we␣
↪run
def eval_model(model, X_train, Y_train, X_test, Y_test):
    fig = plt.figure(figsize=[25, 8])
    ax = fig.add_subplot(1, 2, 1)
    conf = ConfusionMatrixDisplay.from_estimator(model, X_train, Y_train,␣
↪normalize="true", xticks_rotation='vertical', ax=ax)
    pred = model.predict(X_train)

    conf.ax_.set_title('Training Set Performance: ' + str(sum(pred == Y_train)/
↪len(Y_train)));

    ax = fig.add_subplot(1, 2, 2)
    conf = ConfusionMatrixDisplay.from_estimator(model, X_test, Y_test,␣
↪normalize="true", xticks_rotation='vertical', ax=ax)
    pred = model.predict(X_test)
    conf.ax_.set_title('Testing Set Performance: ' + str(sum(pred == Y_test)/
↪len(Y_test)));
```

3.2 Support Vector Machines (SVMs)

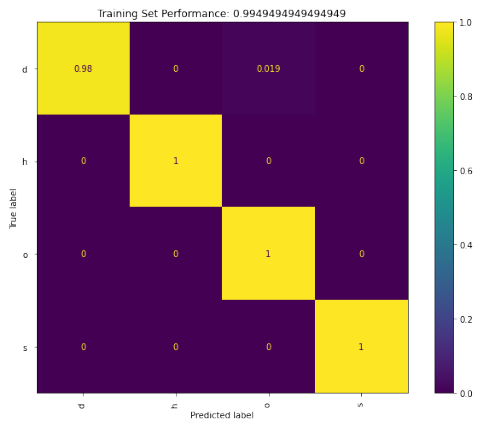
[ ]:
```
## Grid Search
param_grid = [
  {'C': [0.1, 1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [0.1, 1, 10, 100, 1000], 'gamma': [0.1, 0.01, 0.001, 0.0001], 'kernel':␣
↪['rbf']},
  {'C': [0.1, 1, 10, 100, 1000], 'degree': [3, 4, 5, 6], 'kernel': ['poly']},
 ]
svm = SVC()
grid_search = GridSearchCV(svm, param_grid)
grid_search.fit(X_val, Y_val)
grid_search.cv_results_
```

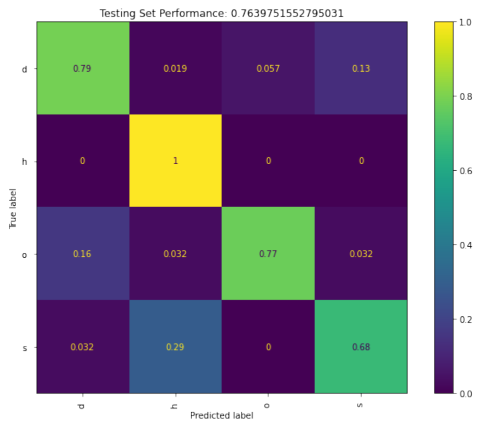
```
# Important:
# - rank test score gives us the order of which system is best
# - mean_test_score and std_test_score tell us the performance of the system␣
 ↪across multiple folds
# - params which give us the model 0.8594697
```

```
[ ]: {'mean_fit_time': array([0.00259194, 0.0024828 , 0.00428138, 0.01008492,
0.02343988,
        0.00279408, 0.00255837, 0.00262728, 0.00246625, 0.0034595 ,
        0.00219111, 0.00240893, 0.00251455, 0.00341659, 0.00204678,
        0.00207343, 0.00239062, 0.00326247, 0.00210996, 0.00192242,
        0.00202746, 0.0032414 , 0.00215216, 0.00264678, 0.00196285,
        0.00202665, 0.00227728, 0.00212889, 0.0023181 , 0.00208101,
        0.00245128, 0.00250654, 0.00231099, 0.00206823, 0.00291381,
        0.00210247, 0.00268817, 0.0021728 , 0.00249696, 0.00213032,
        0.00280085, 0.00221729, 0.00250626, 0.00208173, 0.00414715]),
 'mean_score_time': array([0.00068011, 0.00069404, 0.0007309 , 0.00091062,
0.00086126,
        0.00093279, 0.00097842, 0.00100179, 0.00096145, 0.00097613,
        0.00091958, 0.00091958, 0.00092154, 0.00097084, 0.0008111 ,
        0.00084314, 0.00090756, 0.00088458, 0.00074272, 0.00071907,
        0.00083127, 0.00090322, 0.00075278, 0.00076761, 0.00071545,
        0.00070891, 0.0007566 , 0.00073776, 0.00075197, 0.00069227,
        0.00080571, 0.00084324, 0.00070214, 0.00070767, 0.00071616,
        0.0006721 , 0.00082879, 0.00065136, 0.00069861, 0.00068769,
        0.00072861, 0.00071349, 0.00068746, 0.00063796, 0.00099678]),
 'mean_test_score': array([0.8592803 , 0.79185606, 0.76780303, 0.76174242,
0.72537879,
        0.45113636, 0.49393939, 0.45113636, 0.45113636, 0.79280303,
        0.8407197 , 0.56098485, 0.45113636, 0.83522727, 0.85909091,
        0.82878788, 0.58541667, 0.83522727, 0.8219697 , 0.8532197 ,
        0.83484848, 0.83522727, 0.80984848, 0.77935606, 0.8594697 ,
        0.54223485, 0.51799242, 0.52405303, 0.50587121, 0.71893939,
        0.59734848, 0.58522727, 0.55473485, 0.77367424, 0.65833333,
        0.68295455, 0.59128788, 0.79829545, 0.68882576, 0.71306818,
        0.60378788, 0.79829545, 0.66458333, 0.76799242, 0.60965909]),
 'param_C': masked_array(data=[0.1, 1, 10, 100, 1000, 0.1, 0.1, 0.1, 0.1, 1, 1,
1, 1,
                    10, 10, 10, 10, 100, 100, 100, 100, 1000, 1000, 1000,
                    1000, 0.1, 0.1, 0.1, 0.1, 1, 1, 1, 1, 10, 10, 10, 10,
                    100, 100, 100, 100, 1000, 1000, 1000, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_degree': masked_array(data=[--, --, --, --, --, --, --, --, --, --, --,
--, --, --,
                    --, --, --, --, --, --, --, --, --, --, --, 3, 4, 5, 6,
                    3, 4, 5, 6, 3, 4, 5, 6, 3, 4, 5, 6, 3, 4, 5, 6],
              mask=[ True,  True,  True,  True,  True,  True,  True,  True,
                     True,  True,  True,  True,  True,  True,  True,  True,
                     True,  True,  True,  True,  True,  True,  True,  True,
                     True, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_gamma': masked_array(data=[--, --, --, --, --, 0.1, 0.01, 0.001, 0.0001,
0.1,
                    0.01, 0.001, 0.0001, 0.1, 0.01, 0.001, 0.0001, 0.1,
                    0.01, 0.001, 0.0001, 0.1, 0.01, 0.001, 0.0001, --, --,
                    --, --, --, --, --, --, --, --, --, --, --, --, --, --,
                    --, --, --, --],
              mask=[ True,  True,  True,  True,  True, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False,  True,  True,  True,  True,  True,  True,  True,
                     True,  True,  True,  True,  True,  True,  True,  True,
                     True,  True,  True,  True,  True],
        fill_value='?',
             dtype=object),
 'param_kernel': masked_array(data=['linear', 'linear', 'linear', 'linear',
'linear',
                    'rbf', 'rbf', 'rbf', 'rbf', 'rbf', 'rbf', 'rbf', 'rbf',
                    'rbf', 'rbf', 'rbf', 'rbf', 'rbf', 'rbf', 'rbf', 'rbf',
                    'rbf', 'rbf', 'rbf', 'rbf', 'poly', 'poly', 'poly',
                    'poly', 'poly', 'poly', 'poly', 'poly', 'poly', 'poly',
                    'poly', 'poly', 'poly', 'poly', 'poly', 'poly', 'poly',
                    'poly', 'poly', 'poly'],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.1, 'kernel': 'linear'},
  {'C': 1, 'kernel': 'linear'},
  {'C': 10, 'kernel': 'linear'},
  {'C': 100, 'kernel': 'linear'},
  {'C': 1000, 'kernel': 'linear'},
  {'C': 0.1, 'gamma': 0.1, 'kernel': 'rbf'},
  {'C': 0.1, 'gamma': 0.01, 'kernel': 'rbf'},
  {'C': 0.1, 'gamma': 0.001, 'kernel': 'rbf'},
  {'C': 0.1, 'gamma': 0.0001, 'kernel': 'rbf'},
  {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'},
  {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'},
  {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'},
  {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'},
  {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'},
  {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'},
  {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'},
  {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'},
  {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'},
  {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'},
  {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'},
  {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'},
  {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'},
  {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'},
  {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'},
  {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'},
  {'C': 0.1, 'degree': 3, 'kernel': 'poly'},
  {'C': 0.1, 'degree': 4, 'kernel': 'poly'},
  {'C': 0.1, 'degree': 5, 'kernel': 'poly'},
  {'C': 0.1, 'degree': 6, 'kernel': 'poly'},
  {'C': 1, 'degree': 3, 'kernel': 'poly'},
  {'C': 1, 'degree': 4, 'kernel': 'poly'},
  {'C': 1, 'degree': 5, 'kernel': 'poly'},
  {'C': 1, 'degree': 6, 'kernel': 'poly'},
  {'C': 10, 'degree': 3, 'kernel': 'poly'},
  {'C': 10, 'degree': 4, 'kernel': 'poly'},
  {'C': 10, 'degree': 5, 'kernel': 'poly'},
  {'C': 10, 'degree': 6, 'kernel': 'poly'},
  {'C': 100, 'degree': 3, 'kernel': 'poly'},
  {'C': 100, 'degree': 4, 'kernel': 'poly'},
  {'C': 100, 'degree': 5, 'kernel': 'poly'},
  {'C': 100, 'degree': 6, 'kernel': 'poly'},
  {'C': 1000, 'degree': 3, 'kernel': 'poly'},
  {'C': 1000, 'degree': 4, 'kernel': 'poly'},
  {'C': 1000, 'degree': 5, 'kernel': 'poly'},
  {'C': 1000, 'degree': 6, 'kernel': 'poly'}],
 'rank_test_score': array([ 2, 16, 20, 21, 22, 42, 41, 42, 42, 15,  5, 35, 42,
6,  3, 10, 33,
        6, 11,  4,  9,  6, 12, 17,  1, 37, 39, 38, 40, 23, 31, 34, 36, 18,
       28, 26, 32, 13, 25, 24, 30, 13, 27, 19, 29], dtype=int32),
 'split0_test_score': array([0.81818182, 0.78787879, 0.72727273, 0.72727273,
0.72727273,
        0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.75757576,
        0.87878788, 0.54545455, 0.45454545, 0.78787879, 0.84848485,
        0.87878788, 0.60606061, 0.78787879, 0.84848485, 0.84848485,
        0.87878788, 0.78787879, 0.84848485, 0.75757576, 0.84848485,
        0.51515152, 0.51515152, 0.51515152, 0.48484848, 0.75757576,
        0.48484848, 0.51515152, 0.48484848, 0.84848485, 0.60606061,
        0.63636364, 0.51515152, 0.78787879, 0.6969697 , 0.75757576,
        0.57575758, 0.78787879, 0.66666667, 0.78787879, 0.60606061]),
 'split1_test_score': array([0.84848485, 0.81818182, 0.81818182, 0.81818182,
0.72727273,
        0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.78787879,
        0.90909091, 0.57575758, 0.45454545, 0.84848485, 0.87878788,
        0.84848485, 0.60606061, 0.84848485, 0.90909091, 0.84848485,
        0.84848485, 0.84848485, 0.93939394, 0.84848485, 0.84848485,
        0.57575758, 0.57575758, 0.57575758, 0.57575758, 0.78787879,
        0.6969697 , 0.60606061, 0.60606061, 0.87878788, 0.75757576,
        0.72727273, 0.66666667, 0.78787879, 0.6969697 , 0.78787879,
        0.57575758, 0.78787879, 0.66666667, 0.78787879, 0.60606061]),
 'split2_test_score': array([0.93939394, 0.81818182, 0.75757576, 0.75757576,
0.6969697 ,
        0.45454545, 0.51515152, 0.45454545, 0.45454545, 0.75757576,
        0.84848485, 0.51515152, 0.45454545, 0.81818182, 0.87878788,
        0.81818182, 0.51515152, 0.81818182, 0.87878788, 0.87878788,
        0.81818182, 0.81818182, 0.84848485, 0.84848485, 0.87878788,
        0.60606061, 0.54545455, 0.57575758, 0.51515152, 0.6969697 ,
        0.63636364, 0.63636364, 0.57575758, 0.66666667, 0.63636364,
        0.66666667, 0.60606061, 0.81818182, 0.6969697 , 0.63636364,
        0.60606061, 0.81818182, 0.66666667, 0.72727273, 0.57575758]),
 'split3_test_score': array([0.90909091, 0.87878788, 0.84848485, 0.81818182,
0.78787879,
        0.45454545, 0.54545455, 0.45454545, 0.45454545, 0.84848485,
        0.84848485, 0.60606061, 0.45454545, 0.90909091, 0.93939394,
        0.84848485, 0.60606061, 0.90909091, 0.84848485, 0.90909091,
        0.87878788, 0.90909091, 0.78787879, 0.84848485, 0.90909091,
        0.54545455, 0.48484848, 0.48484848, 0.48484848, 0.72727273,
        0.60606061, 0.60606061, 0.57575758, 0.81818182, 0.66666667,
        0.6969697 , 0.60606061, 0.87878788, 0.6969697 , 0.72727273,
        0.63636364, 0.87878788, 0.66666667, 0.81818182, 0.66666667]),
 'split4_test_score': array([0.78125, 0.65625, 0.6875 , 0.6875 , 0.6875 , 0.4375
, 0.5    ,
        0.4375 , 0.4375 , 0.8125 , 0.71875, 0.5625 , 0.4375 , 0.8125 ,
        0.75   , 0.75   , 0.59375, 0.8125 , 0.625  , 0.78125, 0.75   ,
        0.8125 , 0.625  , 0.59375, 0.8125 , 0.46875, 0.46875, 0.46875,
        0.46875, 0.625  , 0.5625 , 0.5625 , 0.53125, 0.65625, 0.625  ,
        0.6875 , 0.5625 , 0.71875, 0.65625, 0.65625, 0.625  , 0.71875,
        0.65625, 0.71875, 0.59375]),
 'std_fit_time': array([1.41102550e-03, 1.60003957e-04, 8.01821647e-04,
4.65258709e-03,
        1.93769643e-02, 7.18075281e-05, 1.80330748e-04, 1.18102813e-04,
        6.58873356e-05, 4.32870409e-04, 4.78886492e-05, 4.22393756e-05,
        8.68549950e-05, 1.45814230e-04, 5.63397581e-05, 9.47287152e-06,
        6.01463498e-05, 9.69541867e-05, 8.68167660e-05, 4.41324060e-05,
        7.75136018e-05, 7.02893555e-05, 4.30286818e-05, 2.57504587e-04,
        7.34656434e-05, 4.45967927e-05, 1.51449787e-04, 1.04634893e-04,
        6.52517111e-05, 9.31991125e-05, 2.76829027e-04, 4.48677070e-04,
        4.12515130e-05, 6.72604800e-05, 7.82093759e-04, 3.55301603e-05,
        4.88710494e-04, 2.35857878e-04, 1.12839360e-04, 2.92936400e-05,
        5.41583084e-04, 2.16896289e-04, 8.08934247e-05, 5.22184805e-05,
        9.76052090e-04]),
 'std_score_time': array([5.46521953e-05, 4.17762148e-05, 4.22496558e-05,
1.59150115e-04,
        1.11068425e-04, 6.76134241e-06, 4.78085790e-05, 2.88371781e-05,
        6.25443509e-05, 2.02132841e-05, 1.24689224e-04, 2.86257225e-05,
        2.00767131e-05, 4.79689649e-05, 3.36658515e-05, 1.72134912e-05,
        2.02195824e-05, 1.73842298e-05, 1.38277843e-05, 1.24389624e-05,
        3.10320259e-05, 1.33808697e-05, 3.35595133e-05, 4.19610197e-05,
        8.03495373e-06, 3.92702541e-05, 6.07630880e-05, 5.58932888e-05,
        4.50284274e-05, 2.10800858e-05, 1.66823277e-04, 1.78485658e-04,
        9.40107162e-06, 8.15825598e-05, 5.34928412e-05, 1.21148548e-05,
        1.74066686e-04, 1.61252763e-05, 2.45158302e-05, 1.74065811e-05,
        2.80872872e-05, 1.28422312e-04, 2.61546811e-05, 1.12713250e-05,
        2.26347643e-04]),
 'std_test_score': array([0.05795455, 0.07395682, 0.05872556, 0.05116301,
0.03508647,
        0.00681818, 0.0353391 , 0.00681818, 0.00681818, 0.03462544,
        0.06499386, 0.0303125 , 0.00681818, 0.04166667, 0.06202876,
        0.04380858, 0.03545463, 0.04166667, 0.10101641, 0.04242593,
        0.04800903, 0.04166667, 0.10432564, 0.0992576 , 0.03250521,
        0.04762646, 0.03904456, 0.04476914, 0.03803189, 0.05589657,
        0.07126449, 0.04221404, 0.04231757, 0.09366483, 0.0533755 ,
        0.03038814, 0.05048548, 0.05180539, 0.01628788, 0.05812265,
        0.02485033, 0.05180539, 0.00416667, 0.0384521 , 0.03058815])}
```

```
[ ]: ## Best system from data
     best_system = np.argmin(grid_search.cv_results_['rank_test_score'])
     params = grid_search.cv_results_['params'][best_system]
     print(params)
     svm = SVC().set_params(**params)
     svm.fit(X_train, Y_train)
     eval_model(svm, X_train, Y_train, X_test, Y_test)
```

```
{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
```





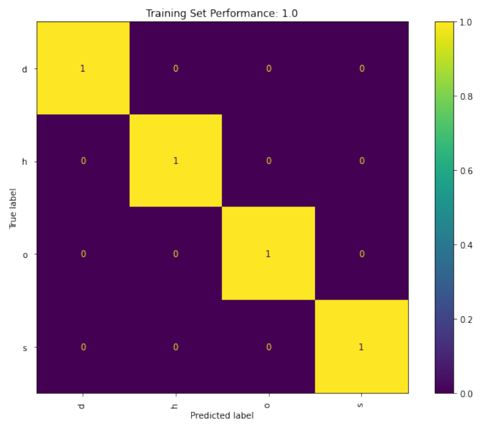
3.3 K Nearest Neighbours

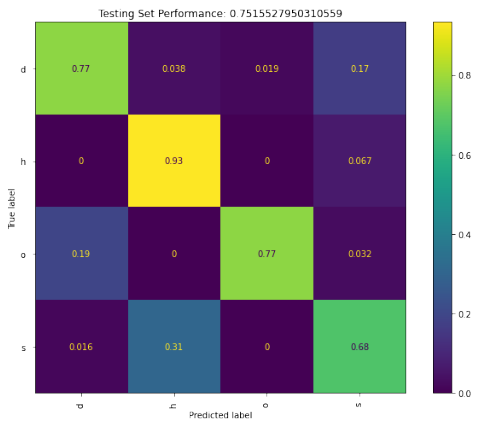
```
[ ]: ## Grid Search
     param_grid = {'n_neighbors' : list(range(1,131)), 'weights' : ['uniform',␣
      ↪'distance']}
     cknn = KNeighborsClassifier()
     grid_search = GridSearchCV(cknn, param_grid)
     grid_search.fit(X_val, Y_val)
     grid_search.cv_results_
```

```
[ ]: {'mean_fit_time': array([0.00161018, 0.00082264, 0.0008532 , 0.0013741 ,
     0.00107298,
             0.00108104, 0.00096321, 0.00091825, 0.00092912, 0.00084624,
             0.00090494, 0.00081439, 0.00083532, 0.0008286 , 0.00082483,
             0.00084109, 0.00118551, 0.00103765, 0.00083041, 0.00092812,
             0.00086789, 0.00089192, 0.00086875, 0.00083466, 0.00082164,
             0.00082159, 0.00082445, 0.00086794, 0.00093503, 0.00088754,
             0.00092187, 0.0009017 , 0.00102425, 0.00084639, 0.00085368,
             0.00081697, 0.00084553, 0.0009953 , 0.0008419 , 0.00081801,
             0.00080986, 0.00093656, 0.00081968, 0.00082417, 0.00081921,
             0.00081959, 0.00084581, 0.00080786, 0.00081244, 0.00081921,
             0.00084615, 0.00079718, 0.00081444, 0.00081911, 0.00085988,
             0.00087328, 0.00095339, 0.00082464, 0.00081906, 0.00081134,
             0.00082059, 0.00088034, 0.00080786, 0.00083389, 0.00083036,
             0.00084305, 0.00130391, 0.00082884, 0.00099568, 0.00086155,
             0.00084014, 0.00081482, 0.00084796, 0.00089936, 0.00082502,
             0.00081754, 0.00083961, 0.00082145, 0.00081768, 0.00079741,
             0.00082259, 0.00081196, 0.00080266, 0.00103378, 0.00104108,
             0.00114336, 0.00082226, 0.0008153 , 0.00086961, 0.00080771,
             0.00081964, 0.00079827, 0.00082154, 0.00082574, 0.00082316,
             0.00106783, 0.00082631, 0.00082102, 0.00081024, 0.00085254,
             0.0008224 , 0.00087385, 0.00083642, 0.0008625 , 0.00084009,
             0.00088935, 0.00086312, 0.00091572, 0.00087461, 0.00098391,
             0.0008719 , 0.00084848, 0.00086193, 0.00087256, 0.00085196,
             0.00094161, 0.00084028, 0.00083003, 0.00081782, 0.00082989,
             0.00085473, 0.00082064, 0.00081186, 0.00097365, 0.00080609,
             0.00082631, 0.00082841, 0.00082488, 0.0008606 , 0.00090785,
             0.00083246, 0.00085282, 0.00087538, 0.00082326, 0.00083671,
             0.00082264, 0.00087404, 0.00085783, 0.00086761, 0.0008616 ,
             0.00085454, 0.00085702, 0.00086026, 0.00082936, 0.00083094,
             0.00080652, 0.00081153, 0.00082536, 0.00122538, 0.00096655,
             0.0011867 , 0.00093384, 0.00093055, 0.00093112, 0.00093784,
             0.00083675, 0.00082393, 0.0008203 , 0.00083408, 0.00084658,
             0.000841  , 0.00083523, 0.0008338 , 0.00088873, 0.00082908,
             0.00085621, 0.00090551, 0.00087461, 0.00089741, 0.00085626,
             0.000875  , 0.00093045, 0.0009439 , 0.00086255, 0.0009316 ,
             0.00092864, 0.00102396, 0.00087504, 0.00085974, 0.00109959,
             0.00092525, 0.00098996, 0.00094976, 0.00092993, 0.0009716 ,
             0.00090914, 0.00095954, 0.00112104, 0.00090914, 0.00091114,
             0.0008635 , 0.0008842 , 0.00092521, 0.0008863 , 0.00085793,
             0.00087295, 0.00090446, 0.00090513, 0.00089064, 0.0008872 ,
             0.00085979, 0.00081511, 0.00093651, 0.00085249, 0.00097675,
             0.0008431 , 0.00085731, 0.00105119, 0.000948  , 0.00088573,
             0.00104127, 0.0012342 , 0.00091414, 0.00083237, 0.0008358 ,
             0.00082622, 0.0008409 , 0.0011219 , 0.00088801, 0.00086193,
             0.00084877, 0.00087299, 0.00098786, 0.00086856, 0.00085945,
             0.00090752, 0.00090518, 0.00090566, 0.00091844, 0.00096707,
             0.0008657 , 0.00087056, 0.00087924, 0.00100045, 0.00089788,
             0.00091248, 0.00086551, 0.00082231, 0.0008554 , 0.00082831,
             0.00089355, 0.0008822 , 0.00086694, 0.00086412, 0.00084925,
             0.0008306 , 0.00087981, 0.00085564, 0.000841  , 0.00104289,
             0.00084615, 0.0009058 , 0.00086131, 0.00093842, 0.00096869,
             0.00102429, 0.00091   , 0.0009409 , 0.0009635 , 0.00091228]),
      'mean_score_time': array([0.00223699, 0.00109243, 0.00203595, 0.00213485,
     0.00242772,
             0.00138054, 0.00209093, 0.00121436, 0.00202398, 0.00117197,
             0.00200577, 0.00116596, 0.00203362, 0.00123415, 0.00194755,
             0.00118923, 0.0031703 , 0.00140457, 0.00197625, 0.00126295,
             0.0020052 , 0.00126724, 0.00204782, 0.00128036, 0.00194964,
             0.00118876, 0.00195193, 0.00118928, 0.00258503, 0.00131216,
             0.00232334, 0.00136862, 0.00224504, 0.00123963, 0.00210557,
             0.00124846, 0.00199065, 0.00131879, 0.00209537, 0.00122662,
             0.00219355, 0.00134997, 0.00200429, 0.00132456, 0.00199308,
             0.00129576, 0.00201783, 0.00125346, 0.00199733, 0.00125194,
             0.00205817, 0.00130167, 0.00198717, 0.00127025, 0.00205436,
             0.0013298 , 0.00250955, 0.00128093, 0.00200601, 0.00126204,
             0.00198941, 0.00129299, 0.00200377, 0.00129924, 0.00240002,
             0.0012846 , 0.00209246, 0.00129662, 0.00253434, 0.00134249,
             0.00205765, 0.00134645, 0.00208354, 0.00132895, 0.00204611,
             0.00131469, 0.00202489, 0.0013351 , 0.00202022, 0.00130105,
             0.00205293, 0.00132523, 0.00202866, 0.00192113, 0.00291214,
             0.0017961 , 0.00210114, 0.00140171, 0.00216317, 0.00136447,
             0.00208483, 0.00133142, 0.00218477, 0.0013669 , 0.00208483,
             0.00164247, 0.00207515, 0.00136008, 0.00205708, 0.00138001,
             0.00208845, 0.00142932, 0.00209298, 0.0013823 , 0.00212173,
             0.00153184, 0.00209961, 0.00146623, 0.00222907, 0.00150623,
             0.00220284, 0.00143185, 0.00218797, 0.00144777, 0.00211244,
             0.00144353, 0.00209403, 0.00141263, 0.00210333, 0.00142212,
             0.00213432, 0.00143991, 0.00260825, 0.00179191, 0.00209241,
             0.00154147, 0.00212021, 0.00149074, 0.00221066, 0.00151234,
             0.00214581, 0.00147099, 0.00233793, 0.00145025, 0.00211949,
             0.00146561, 0.00214658, 0.00148191, 0.00219021, 0.00150166,
             0.00222654, 0.00150609, 0.00218673, 0.00148582, 0.00216222,
             0.00146718, 0.00212011, 0.00148201, 0.00322881, 0.00162315,
             0.00324192, 0.00156655, 0.00222106, 0.00152674, 0.00247984,
             0.00151873, 0.00215855, 0.00151415, 0.00216122, 0.00152779,
             0.0022851 , 0.00152988, 0.00218453, 0.00155864, 0.00218825,
             0.0015614 , 0.00228801, 0.0016099 , 0.00229316, 0.00159597,
             0.00240526, 0.00158448, 0.00245614, 0.0016242 , 0.00232983,
             0.00159631, 0.00252752, 0.00159621, 0.00303636, 0.00199175,
             0.00235524, 0.00195966, 0.00240436, 0.00166001, 0.00230041,
             0.00162578, 0.00303426, 0.0017911 , 0.00229774, 0.00165534,
             0.00234871, 0.0016356 , 0.00256681, 0.0016829 , 0.00224662,
             0.00160737, 0.00259562, 0.00165958, 0.00233698, 0.00166721,
             0.00224714, 0.00160694, 0.00230255, 0.00163884, 0.00255919,
             0.00164003, 0.00227561, 0.00240359, 0.00233655, 0.0018115 ,
             0.00288029, 0.00222964, 0.0023066 , 0.00168276, 0.00222545,
             0.00162945, 0.00226836, 0.0023715 , 0.00231705, 0.00171623,
             0.00229731, 0.00173836, 0.00243616, 0.00169621, 0.00233188,
             0.00177479, 0.00235405, 0.00174417, 0.00277553, 0.00178571,
             0.00230193, 0.00178237, 0.002387  , 0.0018538 , 0.00238853,
             0.00176926, 0.00231047, 0.00169997, 0.00226812, 0.00167284,
             0.00231423, 0.00172982, 0.00231667, 0.00172672, 0.00228019,
             0.00173159, 0.00233102, 0.00170364, 0.00235062, 0.00260653,
             0.00230041, 0.00192885, 0.00231247, 0.00204687, 0.00243073,
             0.00187039, 0.00243182, 0.00197992, 0.00249748, 0.00182676]),
      'mean_test_score': array([0.75606061, 0.75606061, 0.76875   , 0.75606061,
     0.79280303,
             0.78655303, 0.81685606, 0.81079545, 0.80473485, 0.79848485,
             0.79867424, 0.81041667, 0.79848485, 0.79223485, 0.81098485,
             0.79242424, 0.80454545, 0.80435606, 0.81685606, 0.82272727,
             0.80473485, 0.79848485, 0.78636364, 0.79223485, 0.78636364,
             0.78030303, 0.77424242, 0.79848485, 0.78011364, 0.78030303,
             0.75587121, 0.78617424, 0.76818182, 0.78636364, 0.76212121,
             0.79242424, 0.75606061, 0.78636364, 0.73162879, 0.76212121,
             0.71931818, 0.74981061, 0.70719697, 0.73162879, 0.70719697,
             0.73162879, 0.68882576, 0.71950758, 0.68882576, 0.72537879,
             0.69488636, 0.71931818, 0.68882576, 0.71325758, 0.69488636,
             0.71931818, 0.68882576, 0.71325758, 0.68276515, 0.71325758,
             0.67651515, 0.70719697, 0.66439394, 0.70719697, 0.67045455,
             0.71344697, 0.67045455, 0.71325758, 0.65833333, 0.69507576,
             0.66439394, 0.70113636, 0.65246212, 0.70113636, 0.64621212,
             0.69507576, 0.64015152, 0.69507576, 0.64640152, 0.69507576,
             0.64034091, 0.70113636, 0.64015152, 0.68295455, 0.60984848,
             0.68295455, 0.6280303 , 0.68901515, 0.6219697 , 0.66477273,
             0.6219697 , 0.66477273, 0.61590909, 0.65871212, 0.61590909,
             0.65265152, 0.60984848, 0.65265152, 0.61590909, 0.65871212,
             0.60378788, 0.65265152, 0.60984848, 0.64659091, 0.59753788,
             0.6344697 , 0.60984848, 0.6344697 , 0.59772727, 0.62840909,
             0.59147727, 0.62215909, 0.59147727, 0.62215909, 0.58541667,
             0.60378788, 0.59147727, 0.60378788, 0.59147727, 0.59753788,
             0.59147727, 0.60984848, 0.59147727, 0.59772727, 0.58541667,
             0.59772727, 0.58541667, 0.59166667, 0.57310606, 0.59166667,
             0.57916667, 0.59147727, 0.56704545, 0.59147727, 0.57935606,
             0.58541667, 0.57310606, 0.58541667, 0.56704545, 0.57935606,
             0.56704545, 0.59166667, 0.56704545, 0.57329545, 0.56098485,
             0.57935606, 0.56704545, 0.57329545, 0.56098485, 0.57310606,
             0.56098485, 0.56098485, 0.55492424, 0.56098485, 0.56098485,
             0.56098485, 0.55492424, 0.56098485, 0.55492424, 0.56098485,
             0.54886364, 0.56098485, 0.55492424, 0.56098485, 0.55492424,
             0.55492424, 0.54886364, 0.55492424, 0.54886364, 0.55492424,
             0.54886364, 0.56098485, 0.54261364, 0.56098485, 0.54261364,
             0.56098485, 0.53655303, 0.56098485, 0.53655303, 0.55492424,
             0.53655303, 0.55492424, 0.53655303, 0.55492424, 0.53049242,
             0.56098485, 0.53049242, 0.56098485, 0.51837121, 0.56098485,
             0.51837121, 0.56098485, 0.51837121, 0.56098485, 0.51837121,
             0.56098485, 0.51231061, 0.55492424, 0.50606061, 0.55492424,
             0.5       , 0.55492424, 0.49393939, 0.55492424, 0.48787879,
             0.55492424, 0.48181818, 0.54886364, 0.47556818, 0.54280303,
             0.47556818, 0.54280303, 0.46950758, 0.53655303, 0.47556818,
             0.53655303, 0.46950758, 0.53655303, 0.46950758, 0.53049242,
             0.46344697, 0.53049242, 0.46344697, 0.52424242, 0.45113636,
             0.52424242, 0.45113636, 0.52424242, 0.45113636, 0.52424242,
             0.45113636, 0.52424242, 0.45113636, 0.51818182, 0.45113636,
             0.52424242, 0.45113636, 0.50606061, 0.45113636, 0.5       ,
             0.45113636, 0.5       , 0.45113636, 0.5       , 0.45113636,
             0.49375   , 0.45113636, 0.49375   , 0.45113636, 0.48143939,
             0.45113636, 0.47537879, 0.45113636, 0.46325758, 0.45113636,
             0.46325758, 0.45113636, 0.46325758, 0.45113636, 0.46325758]),
      'param_n_neighbors': masked_array(data=[1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7,
     7, 8, 8, 9, 9,
                         10, 10, 11, 11, 12, 12, 13, 13, 14, 14, 15, 15, 16, 16,
                         17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 23,
                         24, 24, 25, 25, 26, 26, 27, 27, 28, 28, 29, 29, 30, 30,
                         31, 31, 32, 32, 33, 33, 34, 34, 35, 35, 36, 36, 37, 37,
                         38, 38, 39, 39, 40, 40, 41, 41, 42, 42, 43, 43, 44, 44,
                         45, 45, 46, 46, 47, 47, 48, 48, 49, 49, 50, 50, 51, 51,
                         52, 52, 53, 53, 54, 54, 55, 55, 56, 56, 57, 57, 58, 58,
                         59, 59, 60, 60, 61, 61, 62, 62, 63, 63, 64, 64, 65, 65,
                         66, 66, 67, 67, 68, 68, 69, 69, 70, 70, 71, 71, 72, 72,
                         73, 73, 74, 74, 75, 75, 76, 76, 77, 77, 78, 78, 79, 79,
                         80, 80, 81, 81, 82, 82, 83, 83, 84, 84, 85, 85, 86, 86,
                         87, 87, 88, 88, 89, 89, 90, 90, 91, 91, 92, 92, 93, 93,
                         94, 94, 95, 95, 96, 96, 97, 97, 98, 98, 99, 99, 100,
                         100, 101, 101, 102, 102, 103, 103, 104, 104, 105, 105,
                         106, 106, 107, 107, 108, 108, 109, 109, 110, 110, 111,
                         111, 112, 112, 113, 113, 114, 114, 115, 115, 116, 116,
                         117, 117, 118, 118, 119, 119, 120, 120, 121, 121, 122,
                         122, 123, 123, 124, 124, 125, 125, 126, 126, 127, 127,
                         128, 128, 129, 129, 130, 130],
                   mask=[False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False],
             fill_value='?',
                  dtype=object),
      'param_weights': masked_array(data=['uniform', 'distance', 'uniform',
     'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance',
                         'uniform', 'distance', 'uniform', 'distance'],
                   mask=[False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False, False, False, False, False,
                         False, False, False, False],
             fill_value='?',
                  dtype=object),
      'params': [{'n_neighbors': 1, 'weights': 'uniform'},
       {'n_neighbors': 1, 'weights': 'distance'},
       {'n_neighbors': 2, 'weights': 'uniform'},
       {'n_neighbors': 2, 'weights': 'distance'},
       {'n_neighbors': 3, 'weights': 'uniform'},
       {'n_neighbors': 3, 'weights': 'distance'},
       {'n_neighbors': 4, 'weights': 'uniform'},
       {'n_neighbors': 4, 'weights': 'distance'},
       {'n_neighbors': 5, 'weights': 'uniform'},
       {'n_neighbors': 5, 'weights': 'distance'},
       {'n_neighbors': 6, 'weights': 'uniform'},
       {'n_neighbors': 6, 'weights': 'distance'},
       {'n_neighbors': 7, 'weights': 'uniform'},
       {'n_neighbors': 7, 'weights': 'distance'},
       {'n_neighbors': 8, 'weights': 'uniform'},
       {'n_neighbors': 8, 'weights': 'distance'},
       {'n_neighbors': 9, 'weights': 'uniform'},
       {'n_neighbors': 9, 'weights': 'distance'},
       {'n_neighbors': 10, 'weights': 'uniform'},
       {'n_neighbors': 10, 'weights': 'distance'},
       {'n_neighbors': 11, 'weights': 'uniform'},
       {'n_neighbors': 11, 'weights': 'distance'},
       {'n_neighbors': 12, 'weights': 'uniform'},
       {'n_neighbors': 12, 'weights': 'distance'},
       {'n_neighbors': 13, 'weights': 'uniform'},
       {'n_neighbors': 13, 'weights': 'distance'},
       {'n_neighbors': 14, 'weights': 'uniform'},
       {'n_neighbors': 14, 'weights': 'distance'},
       {'n_neighbors': 15, 'weights': 'uniform'},
       {'n_neighbors': 15, 'weights': 'distance'},
       {'n_neighbors': 16, 'weights': 'uniform'},
       {'n_neighbors': 16, 'weights': 'distance'},
       {'n_neighbors': 17, 'weights': 'uniform'},
       {'n_neighbors': 17, 'weights': 'distance'},
       {'n_neighbors': 18, 'weights': 'uniform'},
       {'n_neighbors': 18, 'weights': 'distance'},
       {'n_neighbors': 19, 'weights': 'uniform'},
       {'n_neighbors': 19, 'weights': 'distance'},
       {'n_neighbors': 20, 'weights': 'uniform'},
       {'n_neighbors': 20, 'weights': 'distance'},
       {'n_neighbors': 21, 'weights': 'uniform'},
       {'n_neighbors': 21, 'weights': 'distance'},
       {'n_neighbors': 22, 'weights': 'uniform'},
       {'n_neighbors': 22, 'weights': 'distance'},
       {'n_neighbors': 23, 'weights': 'uniform'},
       {'n_neighbors': 23, 'weights': 'distance'},
       {'n_neighbors': 24, 'weights': 'uniform'},
       {'n_neighbors': 24, 'weights': 'distance'},
       {'n_neighbors': 25, 'weights': 'uniform'},
       {'n_neighbors': 25, 'weights': 'distance'},
       {'n_neighbors': 26, 'weights': 'uniform'},
       {'n_neighbors': 26, 'weights': 'distance'},
       {'n_neighbors': 27, 'weights': 'uniform'},
       {'n_neighbors': 27, 'weights': 'distance'},
       {'n_neighbors': 28, 'weights': 'uniform'},
       {'n_neighbors': 28, 'weights': 'distance'},
       {'n_neighbors': 29, 'weights': 'uniform'},
       {'n_neighbors': 29, 'weights': 'distance'},
       {'n_neighbors': 30, 'weights': 'uniform'},
       {'n_neighbors': 30, 'weights': 'distance'},
       {'n_neighbors': 31, 'weights': 'uniform'},
       {'n_neighbors': 31, 'weights': 'distance'},
       {'n_neighbors': 32, 'weights': 'uniform'},
       {'n_neighbors': 32, 'weights': 'distance'},
       {'n_neighbors': 33, 'weights': 'uniform'},
       {'n_neighbors': 33, 'weights': 'distance'},
       {'n_neighbors': 34, 'weights': 'uniform'},
       {'n_neighbors': 34, 'weights': 'distance'},
       {'n_neighbors': 35, 'weights': 'uniform'},
       {'n_neighbors': 35, 'weights': 'distance'},
       {'n_neighbors': 36, 'weights': 'uniform'},
       {'n_neighbors': 36, 'weights': 'distance'},
       {'n_neighbors': 37, 'weights': 'uniform'},
       {'n_neighbors': 37, 'weights': 'distance'},
       {'n_neighbors': 38, 'weights': 'uniform'},
       {'n_neighbors': 38, 'weights': 'distance'},
       {'n_neighbors': 39, 'weights': 'uniform'},
       {'n_neighbors': 39, 'weights': 'distance'},
       {'n_neighbors': 40, 'weights': 'uniform'},
       {'n_neighbors': 40, 'weights': 'distance'},
       {'n_neighbors': 41, 'weights': 'uniform'},
       {'n_neighbors': 41, 'weights': 'distance'},
       {'n_neighbors': 42, 'weights': 'uniform'},
       {'n_neighbors': 42, 'weights': 'distance'},
       {'n_neighbors': 43, 'weights': 'uniform'},
       {'n_neighbors': 43, 'weights': 'distance'},
       {'n_neighbors': 44, 'weights': 'uniform'},
       {'n_neighbors': 44, 'weights': 'distance'},
       {'n_neighbors': 45, 'weights': 'uniform'},
       {'n_neighbors': 45, 'weights': 'distance'},
       {'n_neighbors': 46, 'weights': 'uniform'},
       {'n_neighbors': 46, 'weights': 'distance'},
       {'n_neighbors': 47, 'weights': 'uniform'},
       {'n_neighbors': 47, 'weights': 'distance'},
       {'n_neighbors': 48, 'weights': 'uniform'},
       {'n_neighbors': 48, 'weights': 'distance'},
       {'n_neighbors': 49, 'weights': 'uniform'},
       {'n_neighbors': 49, 'weights': 'distance'},
       {'n_neighbors': 50, 'weights': 'uniform'},
       {'n_neighbors': 50, 'weights': 'distance'},
       {'n_neighbors': 51, 'weights': 'uniform'},
       {'n_neighbors': 51, 'weights': 'distance'},
       {'n_neighbors': 52, 'weights': 'uniform'},
       {'n_neighbors': 52, 'weights': 'distance'},
       {'n_neighbors': 53, 'weights': 'uniform'},
       {'n_neighbors': 53, 'weights': 'distance'},
       {'n_neighbors': 54, 'weights': 'uniform'},
       {'n_neighbors': 54, 'weights': 'distance'},
       {'n_neighbors': 55, 'weights': 'uniform'},
       {'n_neighbors': 55, 'weights': 'distance'},
       {'n_neighbors': 56, 'weights': 'uniform'},
       {'n_neighbors': 56, 'weights': 'distance'},
       {'n_neighbors': 57, 'weights': 'uniform'},
       {'n_neighbors': 57, 'weights': 'distance'},
       {'n_neighbors': 58, 'weights': 'uniform'},
       {'n_neighbors': 58, 'weights': 'distance'},
       {'n_neighbors': 59, 'weights': 'uniform'},
       {'n_neighbors': 59, 'weights': 'distance'},
       {'n_neighbors': 60, 'weights': 'uniform'},
       {'n_neighbors': 60, 'weights': 'distance'},
       {'n_neighbors': 61, 'weights': 'uniform'},
       {'n_neighbors': 61, 'weights': 'distance'},
       {'n_neighbors': 62, 'weights': 'uniform'},
       {'n_neighbors': 62, 'weights': 'distance'},
       {'n_neighbors': 63, 'weights': 'uniform'},
       {'n_neighbors': 63, 'weights': 'distance'},
       {'n_neighbors': 64, 'weights': 'uniform'},
       {'n_neighbors': 64, 'weights': 'distance'},
       {'n_neighbors': 65, 'weights': 'uniform'},
       {'n_neighbors': 65, 'weights': 'distance'},
       {'n_neighbors': 66, 'weights': 'uniform'},
       {'n_neighbors': 66, 'weights': 'distance'},
       {'n_neighbors': 67, 'weights': 'uniform'},
       {'n_neighbors': 67, 'weights': 'distance'},
       {'n_neighbors': 68, 'weights': 'uniform'},
       {'n_neighbors': 68, 'weights': 'distance'},
       {'n_neighbors': 69, 'weights': 'uniform'},
       {'n_neighbors': 69, 'weights': 'distance'},
       {'n_neighbors': 70, 'weights': 'uniform'},
       {'n_neighbors': 70, 'weights': 'distance'},
       {'n_neighbors': 71, 'weights': 'uniform'},
       {'n_neighbors': 71, 'weights': 'distance'},
       {'n_neighbors': 72, 'weights': 'uniform'},
       {'n_neighbors': 72, 'weights': 'distance'},
       {'n_neighbors': 73, 'weights': 'uniform'},
       {'n_neighbors': 73, 'weights': 'distance'},
       {'n_neighbors': 74, 'weights': 'uniform'},
       {'n_neighbors': 74, 'weights': 'distance'},
       {'n_neighbors': 75, 'weights': 'uniform'},
       {'n_neighbors': 75, 'weights': 'distance'},
       {'n_neighbors': 76, 'weights': 'uniform'},
       {'n_neighbors': 76, 'weights': 'distance'},
       {'n_neighbors': 77, 'weights': 'uniform'},
       {'n_neighbors': 77, 'weights': 'distance'},
       {'n_neighbors': 78, 'weights': 'uniform'},
       {'n_neighbors': 78, 'weights': 'distance'},
       {'n_neighbors': 79, 'weights': 'uniform'},
       {'n_neighbors': 79, 'weights': 'distance'},
       {'n_neighbors': 80, 'weights': 'uniform'},
       {'n_neighbors': 80, 'weights': 'distance'},
       {'n_neighbors': 81, 'weights': 'uniform'},
       {'n_neighbors': 81, 'weights': 'distance'},
       {'n_neighbors': 82, 'weights': 'uniform'},
       {'n_neighbors': 82, 'weights': 'distance'},
       {'n_neighbors': 83, 'weights': 'uniform'},
       {'n_neighbors': 83, 'weights': 'distance'},
       {'n_neighbors': 84, 'weights': 'uniform'},
       {'n_neighbors': 84, 'weights': 'distance'},
       {'n_neighbors': 85, 'weights': 'uniform'},
       {'n_neighbors': 85, 'weights': 'distance'},
       {'n_neighbors': 86, 'weights': 'uniform'},
       {'n_neighbors': 86, 'weights': 'distance'},
       {'n_neighbors': 87, 'weights': 'uniform'},
       {'n_neighbors': 87, 'weights': 'distance'},
       {'n_neighbors': 88, 'weights': 'uniform'},
       {'n_neighbors': 88, 'weights': 'distance'},
       {'n_neighbors': 89, 'weights': 'uniform'},
       {'n_neighbors': 89, 'weights': 'distance'},
       {'n_neighbors': 90, 'weights': 'uniform'},
       {'n_neighbors': 90, 'weights': 'distance'},
       {'n_neighbors': 91, 'weights': 'uniform'},
       {'n_neighbors': 91, 'weights': 'distance'},
       {'n_neighbors': 92, 'weights': 'uniform'},
       {'n_neighbors': 92, 'weights': 'distance'},
       {'n_neighbors': 93, 'weights': 'uniform'},
       {'n_neighbors': 93, 'weights': 'distance'},
       {'n_neighbors': 94, 'weights': 'uniform'},
       {'n_neighbors': 94, 'weights': 'distance'},
       {'n_neighbors': 95, 'weights': 'uniform'},
       {'n_neighbors': 95, 'weights': 'distance'},
       {'n_neighbors': 96, 'weights': 'uniform'},
       {'n_neighbors': 96, 'weights': 'distance'},
       {'n_neighbors': 97, 'weights': 'uniform'},
       {'n_neighbors': 97, 'weights': 'distance'},
       {'n_neighbors': 98, 'weights': 'uniform'},
       {'n_neighbors': 98, 'weights': 'distance'},
       {'n_neighbors': 99, 'weights': 'uniform'},
       {'n_neighbors': 99, 'weights': 'distance'},
       {'n_neighbors': 100, 'weights': 'uniform'},
       {'n_neighbors': 100, 'weights': 'distance'},
       {'n_neighbors': 101, 'weights': 'uniform'},
       {'n_neighbors': 101, 'weights': 'distance'},
       {'n_neighbors': 102, 'weights': 'uniform'},
       {'n_neighbors': 102, 'weights': 'distance'},
       {'n_neighbors': 103, 'weights': 'uniform'},
       {'n_neighbors': 103, 'weights': 'distance'},
       {'n_neighbors': 104, 'weights': 'uniform'},
       {'n_neighbors': 104, 'weights': 'distance'},
       {'n_neighbors': 105, 'weights': 'uniform'},
       {'n_neighbors': 105, 'weights': 'distance'},
       {'n_neighbors': 106, 'weights': 'uniform'},
       {'n_neighbors': 106, 'weights': 'distance'},
       {'n_neighbors': 107, 'weights': 'uniform'},
       {'n_neighbors': 107, 'weights': 'distance'},
       {'n_neighbors': 108, 'weights': 'uniform'},
       {'n_neighbors': 108, 'weights': 'distance'},
       {'n_neighbors': 109, 'weights': 'uniform'},
       {'n_neighbors': 109, 'weights': 'distance'},
       {'n_neighbors': 110, 'weights': 'uniform'},
       {'n_neighbors': 110, 'weights': 'distance'},
       {'n_neighbors': 111, 'weights': 'uniform'},
       {'n_neighbors': 111, 'weights': 'distance'},
       {'n_neighbors': 112, 'weights': 'uniform'},
       {'n_neighbors': 112, 'weights': 'distance'},
       {'n_neighbors': 113, 'weights': 'uniform'},
       {'n_neighbors': 113, 'weights': 'distance'},
       {'n_neighbors': 114, 'weights': 'uniform'},
       {'n_neighbors': 114, 'weights': 'distance'},
       {'n_neighbors': 115, 'weights': 'uniform'},
       {'n_neighbors': 115, 'weights': 'distance'},
       {'n_neighbors': 116, 'weights': 'uniform'},
       {'n_neighbors': 116, 'weights': 'distance'},
       {'n_neighbors': 117, 'weights': 'uniform'},
       {'n_neighbors': 117, 'weights': 'distance'},
       {'n_neighbors': 118, 'weights': 'uniform'},
       {'n_neighbors': 118, 'weights': 'distance'},
       {'n_neighbors': 119, 'weights': 'uniform'},
       {'n_neighbors': 119, 'weights': 'distance'},
       {'n_neighbors': 120, 'weights': 'uniform'},
       {'n_neighbors': 120, 'weights': 'distance'},
       {'n_neighbors': 121, 'weights': 'uniform'},
       {'n_neighbors': 121, 'weights': 'distance'},
       {'n_neighbors': 122, 'weights': 'uniform'},
       {'n_neighbors': 122, 'weights': 'distance'},
       {'n_neighbors': 123, 'weights': 'uniform'},
       {'n_neighbors': 123, 'weights': 'distance'},
       {'n_neighbors': 124, 'weights': 'uniform'},
       {'n_neighbors': 124, 'weights': 'distance'},
       {'n_neighbors': 125, 'weights': 'uniform'},
       {'n_neighbors': 125, 'weights': 'distance'},
       {'n_neighbors': 126, 'weights': 'uniform'},
       {'n_neighbors': 126, 'weights': 'distance'},
       {'n_neighbors': 127, 'weights': 'uniform'},
       {'n_neighbors': 127, 'weights': 'distance'},
       {'n_neighbors': 128, 'weights': 'uniform'},
       {'n_neighbors': 128, 'weights': 'distance'},
       {'n_neighbors': 129, 'weights': 'uniform'},
       {'n_neighbors': 129, 'weights': 'distance'},
       {'n_neighbors': 130, 'weights': 'uniform'},
       {'n_neighbors': 130, 'weights': 'distance'}],
      'rank_test_score': array([ 35,  35,  31,  35,  16,  21,   2,   5,   7,  13,
     11,   6,  12,
              19,   4,  18,   9,  10,   2,   1,   7,  13,  22,  19,  22,  27,
              30,  13,  29,  27,  39,  26,  32,  22,  33,  17,  35,  22,  41,
              33,  46,  40,  54,  41,  54,  41,  68,  45,  71,  44,  65,  46,
              68,  50,  65,  46,  68,  50,  74,  50,  75,  54,  80,  54,  76,
              49,  77,  50,  84,  61,  80,  58,  88,  58,  91,  61,  93,  61,
              90,  61,  92,  58,  93,  72, 106,  72,  98,  67, 101,  78, 101,
              78, 103,  82, 103,  85, 106,  85, 103,  82, 111,  85, 106,  89,
              117,  95, 106,  95, 114,  97, 122,  99, 122,  99, 130, 111, 122,
              111, 122, 117, 122, 106, 122, 114, 130, 114, 130, 119, 141, 119,
              138, 122, 144, 122, 135, 130, 141, 130, 144, 135, 144, 119, 144,
              139, 149, 135, 144, 139, 149, 141, 149, 149, 170, 149, 149, 149,
              170, 149, 170, 149, 186, 149, 170, 149, 170, 170, 186, 170, 186,
              170, 186, 149, 193, 149, 193, 149, 195, 149, 195, 170, 195, 170,
              195, 170, 202, 149, 202, 149, 212, 149, 212, 149, 212, 149, 212,
              149, 217, 170, 218, 170, 220, 170, 224, 170, 227, 170, 228, 186,
              230, 191, 230, 191, 234, 195, 230, 195, 234, 195, 234, 202, 237,
              202, 237, 206, 243, 206, 243, 206, 243, 206, 243, 206, 243, 216,
              243, 206, 243, 218, 243, 220, 243, 220, 243, 220, 243, 225, 243,
              225, 243, 229, 243, 233, 243, 239, 243, 239, 243, 239, 243, 239],
             dtype=int32),
      'split0_test_score': array([0.78787879, 0.78787879, 0.84848485, 0.78787879,
     0.81818182,
              0.81818182, 0.81818182, 0.81818182, 0.84848485, 0.81818182,
              0.87878788, 0.84848485, 0.90909091, 0.90909091, 0.87878788,
              0.87878788, 0.90909091, 0.90909091, 0.87878788, 0.90909091,
              0.87878788, 0.87878788, 0.84848485, 0.84848485, 0.84848485,
              0.84848485, 0.84848485, 0.87878788, 0.87878788, 0.87878788,
              0.84848485, 0.87878788, 0.87878788, 0.87878788, 0.81818182,
              0.87878788, 0.78787879, 0.87878788, 0.78787879, 0.81818182,
              0.78787879, 0.81818182, 0.78787879, 0.81818182, 0.78787879,
              0.78787879, 0.78787879, 0.78787879, 0.78787879, 0.81818182,
              0.78787879, 0.78787879, 0.75757576, 0.75757576, 0.78787879,
              0.78787879, 0.75757576, 0.75757576, 0.75757576, 0.75757576,
              0.75757576, 0.75757576, 0.75757576, 0.75757576, 0.72727273,
              0.78787879, 0.78787879, 0.78787879, 0.78787879, 0.78787879,
              0.78787879, 0.78787879, 0.75757576, 0.78787879, 0.75757576,
              0.75757576, 0.72727273, 0.75757576, 0.72727273, 0.75757576,
              0.6969697 , 0.75757576, 0.72727273, 0.75757576, 0.63636364,
              0.75757576, 0.6969697 , 0.81818182, 0.66666667, 0.72727273,
              0.66666667, 0.72727273, 0.66666667, 0.6969697 , 0.66666667,
              0.6969697 , 0.63636364, 0.6969697 , 0.63636364, 0.72727273,
              0.63636364, 0.66666667, 0.63636364, 0.66666667, 0.63636364,
              0.63636364, 0.63636364, 0.63636364, 0.60606061, 0.60606061,
              0.57575758, 0.60606061, 0.60606061, 0.60606061, 0.57575758,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.63636364, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.57575758, 0.57575758, 0.57575758,
              0.60606061, 0.60606061, 0.57575758, 0.60606061, 0.57575758,
              0.57575758, 0.57575758, 0.60606061, 0.57575758, 0.57575758,
              0.57575758, 0.60606061, 0.57575758, 0.57575758, 0.54545455,
              0.60606061, 0.57575758, 0.57575758, 0.54545455, 0.60606061,
              0.54545455, 0.54545455, 0.51515152, 0.54545455, 0.54545455,
              0.54545455, 0.51515152, 0.54545455, 0.51515152, 0.54545455,
              0.51515152, 0.54545455, 0.51515152, 0.54545455, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.54545455, 0.51515152, 0.54545455, 0.51515152,
              0.54545455, 0.51515152, 0.54545455, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.54545455, 0.51515152, 0.54545455, 0.48484848, 0.54545455,
              0.48484848, 0.54545455, 0.45454545, 0.54545455, 0.48484848,
              0.54545455, 0.45454545, 0.51515152, 0.45454545, 0.51515152,
              0.45454545, 0.51515152, 0.45454545, 0.51515152, 0.45454545,
              0.51515152, 0.45454545, 0.48484848, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545]),
       'split1_test_score': array([0.78787879, 0.78787879, 0.66666667, 0.78787879,
      0.78787879,
              0.78787879, 0.84848485, 0.84848485, 0.81818182, 0.81818182,
              0.81818182, 0.87878788, 0.84848485, 0.84848485, 0.81818182,
              0.84848485, 0.81818182, 0.81818182, 0.84848485, 0.84848485,
              0.81818182, 0.81818182, 0.81818182, 0.81818182, 0.78787879,
              0.81818182, 0.81818182, 0.81818182, 0.81818182, 0.81818182,
              0.78787879, 0.81818182, 0.81818182, 0.81818182, 0.81818182,
              0.84848485, 0.81818182, 0.84848485, 0.78787879, 0.81818182,
              0.78787879, 0.81818182, 0.75757576, 0.78787879, 0.75757576,
              0.78787879, 0.75757576, 0.75757576, 0.75757576, 0.75757576,
              0.75757576, 0.75757576, 0.75757576, 0.75757576, 0.75757576,
              0.75757576, 0.75757576, 0.75757576, 0.72727273, 0.75757576,
              0.72727273, 0.72727273, 0.72727273, 0.75757576, 0.72727273,
              0.72727273, 0.72727273, 0.75757576, 0.6969697 , 0.72727273,
              0.72727273, 0.72727273, 0.66666667, 0.72727273, 0.6969697 ,
              0.6969697 , 0.6969697 , 0.6969697 , 0.6969697 , 0.72727273,
              0.6969697 , 0.72727273, 0.6969697 , 0.72727273, 0.63636364,
              0.6969697 , 0.66666667, 0.6969697 , 0.66666667, 0.6969697 ,
              0.66666667, 0.6969697 , 0.66666667, 0.6969697 , 0.66666667,
              0.66666667, 0.66666667, 0.66666667, 0.66666667, 0.66666667,
              0.63636364, 0.66666667, 0.63636364, 0.66666667, 0.63636364,
              0.66666667, 0.63636364, 0.66666667, 0.63636364, 0.66666667,
              0.63636364, 0.66666667, 0.63636364, 0.66666667, 0.63636364,
              0.66666667, 0.63636364, 0.66666667, 0.63636364, 0.66666667,
              0.63636364, 0.66666667, 0.63636364, 0.63636364, 0.60606061,
              0.63636364, 0.60606061, 0.63636364, 0.60606061, 0.63636364,
              0.60606061, 0.63636364, 0.57575758, 0.63636364, 0.60606061,
              0.63636364, 0.60606061, 0.60606061, 0.57575758, 0.60606061,
              0.57575758, 0.60606061, 0.57575758, 0.57575758, 0.57575758,
              0.57575758, 0.57575758, 0.57575758, 0.57575758, 0.57575758,
              0.57575758, 0.57575758, 0.57575758, 0.57575758, 0.57575758,
              0.57575758, 0.57575758, 0.57575758, 0.57575758, 0.57575758,
              0.54545455, 0.57575758, 0.57575758, 0.57575758, 0.57575758,
              0.57575758, 0.54545455, 0.57575758, 0.54545455, 0.57575758,
              0.54545455, 0.57575758, 0.54545455, 0.57575758, 0.54545455,
              0.57575758, 0.51515152, 0.57575758, 0.51515152, 0.57575758,
              0.51515152, 0.57575758, 0.51515152, 0.57575758, 0.48484848,
              0.57575758, 0.48484848, 0.57575758, 0.48484848, 0.57575758,
              0.48484848, 0.57575758, 0.48484848, 0.57575758, 0.48484848,
              0.57575758, 0.48484848, 0.57575758, 0.48484848, 0.57575758,
              0.48484848, 0.57575758, 0.45454545, 0.57575758, 0.45454545,
              0.57575758, 0.45454545, 0.57575758, 0.45454545, 0.57575758,
              0.45454545, 0.57575758, 0.45454545, 0.57575758, 0.48484848,
              0.57575758, 0.45454545, 0.57575758, 0.45454545, 0.54545455,
              0.45454545, 0.54545455, 0.45454545, 0.54545455, 0.45454545,
              0.54545455, 0.45454545, 0.54545455, 0.45454545, 0.54545455,
              0.45454545, 0.54545455, 0.45454545, 0.51515152, 0.45454545,
              0.54545455, 0.45454545, 0.51515152, 0.45454545, 0.48484848,
              0.45454545, 0.48484848, 0.45454545, 0.48484848, 0.45454545,
              0.48484848, 0.45454545, 0.48484848, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545,
              0.45454545, 0.45454545, 0.45454545, 0.45454545, 0.45454545]),
       'split2_test_score': array([0.6969697 , 0.6969697 , 0.75757576, 0.6969697 ,
      0.75757576,
              0.75757576, 0.87878788, 0.81818182, 0.81818182, 0.81818182,
              0.78787879, 0.78787879, 0.66666667, 0.6969697 , 0.78787879,
              0.72727273, 0.78787879, 0.81818182, 0.81818182, 0.78787879,
              0.72727273, 0.75757576, 0.72727273, 0.75757576, 0.72727273,
              0.72727273, 0.72727273, 0.75757576, 0.66666667, 0.6969697 ,
              0.66666667, 0.72727273, 0.63636364, 0.6969697 , 0.63636364,
              0.66666667, 0.60606061, 0.66666667, 0.60606061, 0.63636364,
              0.60606061, 0.63636364, 0.60606061, 0.63636364, 0.60606061,
              0.63636364, 0.54545455, 0.63636364, 0.57575758, 0.63636364,
              0.57575758, 0.63636364, 0.57575758, 0.63636364, 0.57575758,
              0.63636364, 0.57575758, 0.63636364, 0.57575758, 0.63636364,
              0.57575758, 0.63636364, 0.54545455, 0.60606061, 0.57575758,
              0.60606061, 0.54545455, 0.60606061, 0.54545455, 0.57575758,
              0.51515152, 0.57575758, 0.51515152, 0.57575758, 0.54545455,
              0.63636364, 0.54545455, 0.63636364, 0.54545455, 0.63636364,
              0.54545455, 0.63636364, 0.54545455, 0.60606061, 0.54545455,
              0.60606061, 0.54545455, 0.60606061, 0.54545455, 0.57575758,
              0.54545455, 0.57575758, 0.51515152, 0.57575758, 0.51515152,
              0.54545455, 0.51515152, 0.54545455, 0.51515152, 0.54545455,
              0.51515152, 0.57575758, 0.54545455, 0.54545455, 0.51515152,
              0.54545455, 0.54545455, 0.54545455, 0.51515152, 0.54545455,
              0.54545455, 0.54545455, 0.51515152, 0.54545455, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.51515152, 0.51515152, 0.51515152, 0.51515152, 0.51515152,
              0.48484848, 0.51515152, 0.48484848, 0.51515152, 0.48484848,
              0.51515152, 0.48484848, 0.51515152, 0.48484848, 0.51515152,
              0.48484848, 0.51515152, 0.48484848, 0.51515152, 0.48484848,
              0.51515152, 0.48484848, 0.51515152, 0.48484848, 0.51515152,
              0.45454545, 0.51515152, 0.45454545, 0.51515152, 0.45454545,
              0.51515152, 0.45454545, 0.51515152, 0.45454545, 0.51515152,
              0.45454545, 0.51515152, 0.45454545, 0.51515152, 0.45454545,
              0.51515152, 0.45454545, 0.51515152, 0.45454545, 0.51515152,
              0.45454545, 0.51515152, 0.45454545, 0.51515152, 0.45454545,
              0.51515152, 0.45454545, 0.51515152, 0.45454545, 0.51515152,
              0.45454545, 0.48484848, 0.45454545, 0.48484848, 0.45454545,
              0.48484848, 0.45454545, 0.48484848, 0.45454545, 0.48484848]),
       'split3_test_score': array([0.75757576, 0.75757576, 0.72727273, 0.75757576,
      0.78787879,
              0.78787879, 0.75757576, 0.78787879, 0.75757576, 0.78787879,
              0.72727273, 0.81818182, 0.81818182, 0.78787879, 0.75757576,
              0.75757576, 0.75757576, 0.75757576, 0.75757576, 0.81818182,
              0.81818182, 0.78787879, 0.78787879, 0.81818182, 0.81818182,
              0.75757576, 0.72727273, 0.78787879, 0.81818182, 0.75757576,
              0.75757576, 0.78787879, 0.75757576, 0.78787879, 0.78787879,
              0.81818182, 0.81818182, 0.78787879, 0.75757576, 0.78787879,
              0.72727273, 0.75757576, 0.6969697 , 0.6969697 , 0.6969697 ,
              0.72727273, 0.6969697 , 0.6969697 , 0.66666667, 0.72727273,
              0.6969697 , 0.72727273, 0.6969697 , 0.72727273, 0.6969697 ,
              0.72727273, 0.6969697 , 0.72727273, 0.6969697 , 0.72727273,
              0.6969697 , 0.72727273, 0.66666667, 0.72727273, 0.6969697 ,
              0.72727273, 0.66666667, 0.72727273, 0.63636364, 0.6969697 ,
              0.66666667, 0.72727273, 0.66666667, 0.72727273, 0.60606061,
              0.6969697 , 0.60606061, 0.6969697 , 0.60606061, 0.66666667,
              0.60606061, 0.6969697 , 0.60606061, 0.63636364, 0.60606061,
              0.66666667, 0.60606061, 0.63636364, 0.60606061, 0.63636364,
              0.60606061, 0.63636364, 0.60606061, 0.63636364, 0.60606061,
              0.66666667, 0.60606061, 0.66666667, 0.63636364, 0.66666667,
              0.60606061, 0.66666667, 0.60606061, 0.66666667, 0.60606061,
              0.63636364, 0.60606061, 0.63636364, 0.60606061, 0.63636364,
              0.60606061, 0.63636364, 0.60606061, 0.63636364, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.60606061, 0.60606061,
              0.60606061, 0.60606061, 0.60606061, 0.57575758, 0.60606061,
              0.57575758, 0.60606061, 0.60606061, 0.60606061, 0.57575758,
              0.60606061, 0.57575758, 0.60606061, 0.57575758, 0.60606061,
              0.57575758, 0.60606061, 0.57575758, 0.60606061, 0.54545455,
              0.60606061, 0.51515152, 0.60606061, 0.51515152, 0.60606061,
              0.51515152, 0.60606061, 0.48484848, 0.60606061, 0.48484848,
              0.60606061, 0.48484848, 0.60606061, 0.48484848, 0.60606061,
              0.48484848, 0.60606061, 0.48484848, 0.60606061, 0.45454545,
              0.60606061, 0.45454545, 0.60606061, 0.45454545, 0.60606061,
              0.45454545, 0.60606061, 0.45454545, 0.60606061, 0.45454545,
              0.60606061, 0.45454545, 0.54545455, 0.45454545, 0.54545455,
              0.45454545, 0.54545455, 0.45454545, 0.54545455, 0.45454545,
              0.54545455, 0.45454545, 0.54545455, 0.45454545, 0.54545455,
              0.45454545, 0.54545455, 0.45454545, 0.48484848, 0.45454545,
              0.48484848, 0.45454545, 0.48484848, 0.45454545, 0.48484848]),
       'split4_test_score': array([0.75   , 0.75   , 0.84375, 0.75   , 0.8125 ,
      0.78125, 0.78125,
              0.78125, 0.78125, 0.75   , 0.78125, 0.71875, 0.75   , 0.71875,
              0.8125 , 0.75   , 0.75   , 0.71875, 0.78125, 0.75   , 0.78125,
              0.75   , 0.75   , 0.71875, 0.75   , 0.75   , 0.75   , 0.75   ,
              0.71875, 0.75   , 0.71875, 0.71875, 0.75   , 0.75   , 0.75   ,
              0.75   , 0.75   , 0.75   , 0.71875, 0.75   , 0.6875 , 0.71875,
              0.6875 , 0.71875, 0.6875 , 0.71875, 0.65625, 0.71875, 0.65625,
              0.6875 , 0.65625, 0.6875 , 0.65625, 0.6875 , 0.65625, 0.6875 ,
              0.65625, 0.6875 , 0.65625, 0.6875 , 0.625  , 0.6875 , 0.625  ,
              0.6875 , 0.625  , 0.71875, 0.625  , 0.6875 , 0.625  , 0.6875 ,
              0.625  , 0.6875 , 0.65625, 0.6875 , 0.625  , 0.6875 , 0.625  ,
              0.6875 , 0.65625, 0.6875 , 0.65625, 0.6875 , 0.625  , 0.6875 ,
              0.625  , 0.6875 , 0.625  , 0.6875 , 0.625  , 0.6875 , 0.625  ,
              0.6875 , 0.625  , 0.6875 , 0.625  , 0.6875 , 0.625  , 0.6875 ,
              0.625  , 0.6875 , 0.625  , 0.6875 , 0.625  , 0.6875 , 0.59375,
              0.6875 , 0.625  , 0.6875 , 0.625  , 0.6875 , 0.59375, 0.65625,
              0.59375, 0.65625, 0.59375, 0.625  , 0.59375, 0.625  , 0.59375,
              0.59375, 0.59375, 0.625  , 0.59375, 0.625  , 0.59375, 0.625  ,
              0.59375, 0.625  , 0.5625 , 0.625  , 0.5625 , 0.59375, 0.5625 ,
              0.59375, 0.59375, 0.59375, 0.5625 , 0.59375, 0.5625 , 0.59375,
              0.5625 , 0.625  , 0.5625 , 0.59375, 0.5625 , 0.59375, 0.5625 ,
              0.59375, 0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.5625 ,
              0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.5625 ,
              0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.5625 ,
              0.5625 , 0.5625 , 0.5625 , 0.5625 , 0.53125, 0.5625 , 0.53125,
              0.5625 , 0.53125, 0.5625 , 0.53125, 0.5625 , 0.53125, 0.5625 ,
              0.53125, 0.5625 , 0.53125, 0.5625 , 0.53125, 0.5625 , 0.53125,
              0.5625 , 0.53125, 0.5625 , 0.53125, 0.5625 , 0.53125, 0.5625 ,
              0.53125, 0.5625 , 0.5    , 0.5625 , 0.5    , 0.5625 , 0.5    ,
              0.5625 , 0.5    , 0.5625 , 0.5    , 0.5625 , 0.46875, 0.5625 ,
              0.46875, 0.5625 , 0.46875, 0.53125, 0.46875, 0.53125, 0.46875,
              0.53125, 0.46875, 0.53125, 0.46875, 0.53125, 0.46875, 0.5    ,
              0.4375 , 0.5    , 0.4375 , 0.5    , 0.4375 , 0.5    , 0.4375 ,
              0.5    , 0.4375 , 0.5    , 0.4375 , 0.5    , 0.4375 , 0.5    ,
              0.4375 , 0.5    , 0.4375 , 0.5    , 0.4375 , 0.5    , 0.4375 ,
              0.46875, 0.4375 , 0.46875, 0.4375 , 0.4375 , 0.4375 , 0.4375 ,
              0.4375 , 0.4375 , 0.4375 , 0.4375 , 0.4375 , 0.4375 , 0.4375 ,
              0.4375 ]),
       'std_fit_time': array([1.05529086e-03, 2.21750299e-05, 3.35500266e-05,
      3.66242402e-04,
              2.05947652e-04, 1.37023818e-04, 1.82288116e-04, 9.21405114e-05,
              6.37442291e-05, 3.03350893e-05, 1.58006135e-04, 8.24583676e-06,
              3.78470552e-05, 1.80270935e-05, 1.00153967e-05, 2.14737726e-05,
              2.79359175e-04, 2.33807773e-04, 1.21485906e-05, 1.59538298e-04,
              3.48979652e-05, 2.89825218e-05, 1.51791581e-05, 1.61022761e-05,
              2.03394385e-05, 2.49162815e-05, 1.73735015e-05, 1.15005926e-04,
              9.71920466e-05, 2.88703537e-05, 8.14407435e-05, 2.53159669e-05,
              2.33441279e-04, 2.21506130e-05, 2.19576133e-05, 1.08280889e-05,
              2.77052648e-05, 2.37840935e-04, 2.76686379e-05, 1.77649841e-05,
              1.14910830e-05, 1.91591449e-04, 2.76372283e-05, 2.00512151e-05,
              1.60376152e-05, 9.25923960e-06, 5.15342958e-05, 1.09252956e-05,
              1.15080872e-05, 1.60369063e-05, 4.17108517e-05, 8.56264652e-06,
              1.05217984e-05, 1.72273551e-05, 2.46601812e-05, 3.38921549e-05,
              1.81352612e-04, 4.09329082e-05, 1.34866543e-05, 2.70329242e-05,
              3.10418426e-05, 1.23047143e-04, 1.68753203e-05, 1.96567985e-05,
              2.54625685e-05, 4.80355160e-05, 9.71258334e-04, 2.79910317e-05,
              2.13566064e-04, 4.15200599e-05, 2.24414583e-05, 1.25196772e-05,
              1.41006803e-05, 1.47743929e-04, 1.38739129e-05, 1.54107691e-05,
              5.93467090e-05, 1.60281135e-05, 1.49339125e-05, 9.76364108e-06,
              1.18370923e-05, 2.43848014e-05, 6.02892658e-06, 2.92062506e-04,
              2.42787135e-04, 2.18272937e-04, 1.59590209e-05, 4.92459909e-06,
              4.07191014e-05, 1.29909906e-05, 1.48060817e-05, 1.63516910e-05,
              3.29795554e-05, 2.51707698e-05, 1.26191690e-05, 2.11687956e-04,
              2.71342550e-05, 2.84398889e-05, 1.84989466e-05, 7.73930946e-05,
              2.53552750e-05, 2.56578272e-05, 3.19201076e-05, 9.00679127e-05,
              2.61211900e-05, 1.73095172e-05, 3.83942458e-05, 1.02332676e-04,
              5.25338662e-05, 1.53248178e-04, 3.05056549e-05, 9.28963965e-06,
              1.99853370e-05, 1.86085052e-05, 1.66719892e-05, 1.50222747e-04,
              1.79575886e-05, 3.81295641e-05, 2.19087865e-05, 2.96688524e-05,
              1.32222396e-05, 2.51356964e-05, 1.09687058e-05, 3.10614426e-04,
              9.60681919e-06, 3.60916908e-05, 1.23560644e-05, 8.58492301e-06,
              2.12685409e-05, 1.41558762e-04, 3.92667800e-05, 7.32686323e-06,
              2.60029756e-05, 1.32093360e-05, 1.36308720e-05, 1.63990393e-05,
              9.94905619e-05, 1.34481605e-05, 2.54123339e-05, 1.98133729e-05,
              1.73999179e-05, 1.89644558e-05, 2.31820356e-05, 8.36247856e-06,
              8.92259036e-06, 1.36173539e-05, 1.13957098e-05, 1.13837319e-05,
              3.00967761e-04, 2.00797877e-04, 2.89560266e-04, 1.42863261e-04,
              1.45473084e-04, 1.37175418e-04, 1.45836309e-04, 1.65549731e-05,
              5.30854908e-06, 1.23781269e-05, 4.12279705e-05, 2.66147761e-05,
              1.13389029e-05, 1.03709615e-05, 1.31311292e-05, 1.49744072e-04,
              2.45867708e-05, 1.38179148e-05, 7.05419296e-05, 5.47741663e-05,
              4.09450158e-05, 1.97197389e-05, 4.90475174e-05, 1.28795703e-04,
              1.58977323e-04, 6.26416675e-05, 1.03109638e-04, 1.39139661e-04,
              1.98118202e-04, 3.29592107e-05, 1.46785318e-05, 2.39082604e-04,
              5.00250588e-05, 2.00823036e-04, 3.77556277e-05, 1.14874574e-04,
              2.12593525e-04, 8.88412251e-05, 1.25443667e-04, 1.93253400e-04,
              3.02937615e-05, 1.41200621e-04, 2.26326426e-05, 2.14298916e-05,
              9.93805286e-05, 4.73930560e-05, 1.47277083e-05, 3.97055633e-05,
              9.45016904e-05, 9.48364976e-05, 4.42880336e-05, 2.58775444e-05,
              1.52352270e-05, 1.43807320e-05, 1.78880171e-04, 1.86214526e-05,
              1.54733724e-04, 2.09132102e-05, 1.23623195e-05, 3.29287322e-04,
              1.39925295e-04, 3.11663335e-05, 2.11985444e-04, 2.20900113e-04,
              3.80784850e-05, 2.26923392e-05, 1.17688975e-05, 1.13358946e-05,
              1.98412394e-05, 3.25939723e-04, 7.19471271e-06, 6.62168706e-06,
              1.22473840e-05, 2.40330658e-05, 1.54498963e-04, 3.11112038e-05,
              1.30929794e-05, 2.91809537e-05, 2.26397743e-05, 5.56727752e-05,
              1.02627543e-04, 1.10626365e-04, 2.61691951e-05, 4.35382877e-05,
              3.13910995e-05, 1.34761166e-04, 1.09319533e-05, 2.34657181e-05,
              1.47414421e-05, 1.30151215e-05, 4.07837681e-05, 1.36723441e-05,
              4.91566614e-05, 1.93940875e-05, 1.51183701e-05, 9.47287152e-06,
              1.33964936e-05, 8.72677139e-06, 3.31631245e-05, 2.31506282e-05,
              1.13128046e-05, 2.19031888e-04, 1.38971652e-05, 2.68063075e-05,
              2.19840030e-05, 3.78327542e-05, 1.23178492e-04, 2.37915194e-04,
              2.57202268e-05, 8.03654675e-05, 3.14750091e-05, 7.28111136e-05]),
       'std_score_time': array([4.76724292e-04, 1.78263134e-05, 1.13465864e-04,
      2.88557383e-04,
              6.30889558e-04, 1.55879562e-04, 1.22378099e-04, 3.77276139e-05,
              8.23987027e-05, 7.96417984e-06, 1.18046083e-04, 9.00401898e-06,
              1.27848090e-04, 1.21268005e-04, 7.89536363e-06, 2.26821166e-05,
              9.76946425e-04, 2.71917342e-04, 4.41101434e-05, 1.02009099e-04,
              2.87563250e-05, 5.77444483e-05, 1.13001694e-04, 1.22582991e-04,
              2.41538614e-05, 1.50864525e-05, 2.49850841e-05, 1.72001449e-05,
              8.61999946e-04, 2.44904047e-05, 1.20698534e-04, 5.98103977e-05,
              1.44119691e-04, 2.29103259e-05, 1.23847251e-04, 2.59706021e-05,
              2.51070047e-05, 6.19087435e-05, 1.42598756e-04, 1.70080535e-05,
              4.47943948e-04, 2.08094566e-04, 2.79153847e-05, 7.98743660e-05,
              5.41401506e-05, 8.82184054e-05, 5.02898429e-05, 3.08497796e-05,
              2.10957199e-05, 1.35383130e-05, 4.25307204e-05, 9.22150053e-05,
              3.53636402e-05, 1.93723862e-05, 2.36568905e-05, 3.87312725e-05,
              6.27640528e-04, 2.69931952e-05, 2.71999551e-05, 2.11865999e-05,
              2.05191844e-05, 3.12222397e-05, 1.44156319e-05, 4.73412134e-05,
              7.41806980e-04, 9.93493395e-06, 1.65629275e-04, 1.78266960e-05,
              4.50543833e-04, 2.53345516e-05, 5.59989568e-05, 8.82226837e-05,
              6.74954534e-05, 2.83323141e-05, 2.47592844e-05, 1.44312385e-05,
              3.02800982e-05, 5.27678867e-05, 1.08301886e-05, 1.05336771e-05,
              2.43208454e-05, 1.35799003e-05, 1.83897244e-05, 7.40174361e-04,
              6.76184541e-04, 3.26353134e-04, 7.34647149e-05, 6.56049607e-05,
              1.55040066e-04, 3.47149241e-05, 5.83118622e-05, 2.19984780e-05,
              3.00689845e-04, 3.83912846e-05, 6.63746362e-06, 2.09194499e-04,
              5.22604389e-05, 1.99033681e-05, 2.89393410e-05, 2.65916996e-05,
              3.93662503e-05, 2.07273385e-05, 2.46956538e-05, 1.46981912e-05,
              3.26525190e-05, 1.63073568e-04, 2.57163368e-05, 7.07089215e-05,
              1.41401925e-04, 8.08299605e-05, 1.10959976e-04, 3.33917456e-05,
              5.85413505e-05, 1.24148105e-05, 1.00762811e-05, 2.55384474e-05,
              3.48100923e-05, 1.36969347e-05, 1.72704600e-05, 1.77793132e-05,
              2.64835718e-05, 4.00615867e-05, 9.96741819e-04, 7.04276851e-04,
              2.40429031e-05, 7.86442539e-05, 2.32011537e-05, 9.15734198e-05,
              1.25261809e-04, 7.33279431e-05, 6.08324245e-05, 1.60810813e-05,
              3.31238494e-04, 2.67867916e-05, 2.18745116e-05, 2.60352215e-05,
              1.74450729e-05, 2.07868188e-05, 4.97203298e-05, 2.73263889e-05,
              6.65201337e-05, 2.10479184e-05, 8.26282437e-05, 2.72674150e-05,
              5.81887883e-05, 1.95167998e-05, 2.44327749e-05, 9.63989736e-06,
              6.22987384e-04, 1.55438455e-04, 5.47134945e-04, 3.44178009e-05,
              5.59981041e-05, 2.73899680e-05, 3.83163429e-04, 2.37496500e-05,
              3.99902945e-05, 2.08822013e-05, 1.84267795e-05, 1.92426098e-05,
              1.92749683e-04, 2.69501174e-05, 4.26708249e-05, 1.09962061e-04,
              2.94164085e-05, 1.73931215e-05, 8.86978127e-05, 1.05338994e-04,
              6.92108069e-05, 5.83113553e-05, 2.06521283e-04, 2.71547772e-05,
              1.90501511e-04, 1.00402998e-04, 1.39451514e-04, 6.26504146e-05,
              2.45766025e-04, 2.39398848e-05, 1.57834804e-03, 4.70955263e-04,
              1.13281055e-04, 3.95502479e-04, 1.63276354e-04, 2.55660323e-05,
              6.27535817e-05, 2.22428051e-05, 7.32118862e-04, 1.02500426e-04,
              3.59614260e-05, 2.08868828e-05, 1.49632184e-04, 7.75707222e-06,
              4.53655414e-04, 4.19013717e-05, 3.65833868e-05, 2.27083653e-05,
              4.53338077e-04, 5.32130614e-05, 8.22946892e-05, 5.31810050e-05,
              2.85678391e-05, 2.52768674e-05, 1.44826726e-04, 2.42012592e-05,
              3.98705444e-04, 2.65094872e-05, 1.44195746e-05, 8.60751097e-04,
              1.06849876e-04, 1.66034534e-04, 5.20851793e-04, 3.37780464e-04,
              2.53207266e-05, 1.13369676e-04, 1.90510618e-05, 9.11195760e-06,
              3.36409882e-05, 7.78625239e-04, 7.29535296e-05, 7.05881036e-05,
              7.74616937e-05, 1.15614811e-04, 1.47406863e-04, 3.08387220e-05,
              2.64417276e-05, 8.41835712e-05, 5.14437686e-05, 6.93595011e-05,
              3.44143410e-04, 7.20190509e-05, 4.17993397e-05, 1.51306930e-04,
              1.16707139e-04, 1.08074235e-04, 1.08259406e-04, 5.14527843e-05,
              4.16143631e-05, 5.62179483e-05, 1.98752454e-05, 1.75496805e-05,
              2.84474830e-05, 2.26071107e-05, 4.21612502e-05, 2.05192952e-05,
              3.03911780e-05, 7.22200376e-05, 3.81699135e-05, 3.08379847e-05,
              4.38636797e-05, 1.23194431e-03, 2.76308927e-05, 2.38396138e-04,
              1.65939331e-05, 4.33148738e-04, 1.42428984e-04, 1.46928610e-04,
              1.45017963e-04, 1.28810707e-04, 1.49803415e-04, 8.86340107e-05]),
       'std_test_score': array([0.03333333, 0.03333333, 0.06964033, 0.03333333,
      0.02154767,
              0.01934787, 0.04386259, 0.02419799, 0.03178434, 0.02693392,
              0.04963278, 0.05494514, 0.08348471, 0.07913766, 0.0400943 ,
              0.05976692, 0.05757576, 0.06460416, 0.04386259, 0.05412294,
              0.0497944 , 0.04684735, 0.04401769, 0.04714197, 0.04401769,
              0.04545455, 0.04988508, 0.04684735, 0.07651234, 0.0624713 ,
              0.06154691, 0.05936344, 0.08068804, 0.06143374, 0.06769184,
              0.07599961, 0.07907872, 0.07490429, 0.06772681, 0.06769184,
              0.06827014, 0.06818287, 0.06287879, 0.06495853, 0.06287879,
              0.05583621, 0.08511272, 0.05204854, 0.07599301, 0.0616622 ,
              0.0751882 , 0.05314382, 0.06835941, 0.04626177, 0.0751882 ,
              0.05314382, 0.06835941, 0.04626177, 0.06313952, 0.04626177,
              0.06689012, 0.04183163, 0.07528641, 0.05673732, 0.0603213 ,
              0.05913096, 0.08333333, 0.06306107, 0.08075914, 0.06920528,
              0.09277133, 0.07042195, 0.07787293, 0.07042195, 0.07374602,
              0.03851734, 0.065081  , 0.03851734, 0.06482587, 0.04302203,
              0.0580436 , 0.04066284, 0.065081  , 0.05592223, 0.0340488 ,
              0.048915  , 0.05206921, 0.07266806, 0.04497219, 0.05327325,
              0.04497219, 0.05327325, 0.0556496 , 0.04719521, 0.0556496 ,
              0.05488635, 0.05126947, 0.05488635, 0.05224527, 0.06079988,
              0.04568125, 0.03928452, 0.0340488 , 0.05120786, 0.04447331,
              0.04853809, 0.0340488 , 0.04853809, 0.04288173, 0.04979872,
              0.03032433, 0.04356061, 0.04067166, 0.04356061, 0.0403031 ,
              0.04953874, 0.04067166, 0.04953874, 0.04067166, 0.04842711,
              0.04067166, 0.05126947, 0.04067166, 0.04288173, 0.03545463,
              0.04288173, 0.03545463, 0.04341378, 0.03361622, 0.04341378,
              0.03618168, 0.04067166, 0.02962305, 0.04067166, 0.03396652,
              0.0403031 , 0.03361622, 0.03545463, 0.02962305, 0.03396652,
              0.02962305, 0.03895442, 0.02962305, 0.03125631, 0.0303125 ,
              0.03396652, 0.02962305, 0.03125631, 0.0303125 , 0.03361622,
              0.0303125 , 0.0303125 , 0.03541211, 0.0303125 , 0.0303125 ,
              0.0303125 , 0.03541211, 0.0303125 , 0.03541211, 0.0303125 ,
              0.03388829, 0.0303125 , 0.03541211, 0.0303125 , 0.03541211,
              0.03541211, 0.03388829, 0.03541211, 0.03388829, 0.03541211,
              0.03388829, 0.0303125 , 0.03367806, 0.0303125 , 0.03367806,
              0.0303125 , 0.03530864, 0.0303125 , 0.03530864, 0.03541211,
              0.03530864, 0.03541211, 0.03530864, 0.03541211, 0.04065755,
              0.0303125 , 0.04065755, 0.0303125 , 0.03381411, 0.0303125 ,
              0.03381411, 0.0303125 , 0.05111391, 0.0303125 , 0.03381411,
              0.0303125 , 0.04119046, 0.03541211, 0.04020151, 0.03541211,
              0.04065578, 0.03541211, 0.04453618, 0.03541211, 0.03374403,
              0.03541211, 0.02424242, 0.04339395, 0.02273043, 0.05297075,
              0.02273043, 0.05297075, 0.01355722, 0.05211466, 0.01222141,
              0.05211466, 0.01355722, 0.05211466, 0.01355722, 0.04886364,
              0.01203211, 0.04886364, 0.01203211, 0.05034317, 0.00681818,
              0.05034317, 0.00681818, 0.05034317, 0.00681818, 0.05034317,
              0.00681818, 0.05034317, 0.00681818, 0.0492366 , 0.00681818,
              0.05034317, 0.00681818, 0.02969078, 0.00681818, 0.03030303,
              0.00681818, 0.03030303, 0.00681818, 0.03030303, 0.00681818,
              0.03277993, 0.00681818, 0.03277993, 0.00681818, 0.04150105,
              0.00681818, 0.03821818, 0.00681818, 0.0186954 , 0.00681818,
              0.0186954 , 0.00681818, 0.0186954 , 0.00681818, 0.0186954 ])}
```

[ ]:
```
## Select the best system from results
best_system = np.argmin(grid_search.cv_results_['rank_test_score'])
params = grid_search.cv_results_['params'][best_system]
print(params)
print(len(grid_search.cv_results_['params']))
print(best_system)
cknn = KNeighborsClassifier().set_params(**params)
cknn.fit(X_train, Y_train)
eval_model(cknn, X_train, Y_train, X_test, Y_test)
```

```
{'n_neighbors': 10, 'weights': 'distance'}
260
19
```



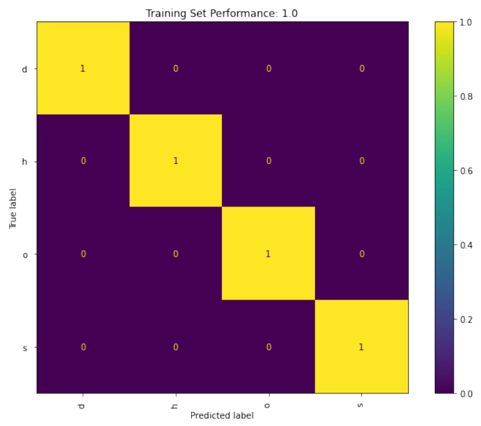


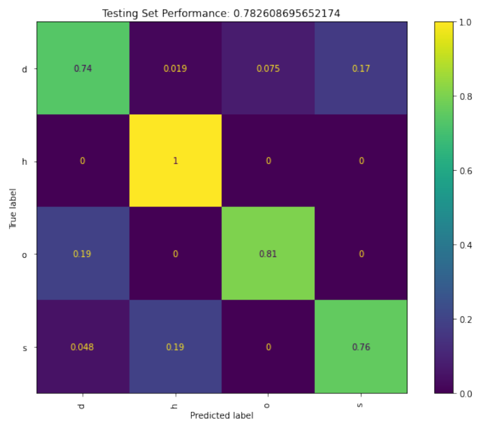
3.4 Random Forest Classifier

```
[ ]: ## Grid Search
     param_grid = {'max_depth': [2, 4, None], 'min_samples_split': [5, 10],␣
      ↪'n_estimators' : [25, 50, 100]}
     rf = RandomForestClassifier()
     grid_search = GridSearchCV(rf, param_grid)
     grid_search.fit(X_val, Y_val)
     grid_search.cv_results_
```

```
[ ]: ## Best system from data
     # rf = RandomForestClassifier().set_params(**grid_search.
      ↪cv_results_['params'][-1])
     # rf.fit(X_train, Y_train)
     # eval_model(rf, X_train, Y_train, X_test, Y_test)

     ## Select the best system from results
     best_system = np.argmin(grid_search.cv_results_['rank_test_score'])
     params = grid_search.cv_results_['params'][best_system]
     print(params)
     rf = RandomForestClassifier().set_params(**params)
     rf.fit(X_train, Y_train)
     eval_model(rf, X_train, Y_train, X_test, Y_test)

     #{'max_depth': 4, 'min_samples_split': 10, 'n_estimators': 50}
```

```
{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 50}
```





3.5 Export as PDF

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('n10477659 Final Assignment_1A.ipynb')
```

```
File ‘colab_pdf.py’ already there; not retrieving.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```