Colab_Lab1_nuevo

January 25, 2019

1 SVM and NN

1.1 Eloy Alfageme, Inés Heras, Jorge García, Manuel Pasieka

In [0]:
```
import numpy as np
import pandas as pd
import math
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error, mean_absolute_error, m
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import io
import requests
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from  matplotlib import pyplot
```

1.2 Data preparation

In [2]:
```
#Read Data from source
url="http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
s=requests.get(url).content

#Define target y features variables
target = ['mpg']
features = [ 'cylinders',
'displacement',
'horsepower',
'weight',
'acceleration',
'model year',
'origin',
'car name']
input_columns = target + features
```

```
        #Load the data as a pandas dataframe
        data = pd.read_csv(io.StringIO(s.decode('utf-8')),delim_whitespace=True,names=input_col
        data.head()
```

Out[2]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

```
In [3]: print(f'Number of features {data.shape[1]-1}\nNumber of Data Points {data.shape[0]}')
        missing_data_points =(data == '?').astype(int).sum()
        print(f'\nThe number of missing data points per feature\n\n{missing_data_points}')
```

```
Number of features 8
Number of Data Points 398

The number of missing data points per feature

mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

```
In [4]: #Missing data and string treatment
        data['car name'] = LabelEncoder().fit_transform(data['car name'])
        data=data.replace('?', np.nan).ffill()

        #Split the data in training, validation and tet
        train_x, rest_x, train_y, rest_y = train_test_split(data[features], data[target],test_s
        test_x, validation_x, test_y, validation_y = train_test_split(rest_x, rest_y,test_size

        # Convert targets to 1d array in order to remove warnings
        train_y = train_y.values.ravel()
```

```
test_y = test_y.values.ravel()
validation_y = validation_y.values.ravel()

print(f'Using {train_x.shape[0]} Data points for training, {test_x.shape[0]} for testi
```

```
Using 298 Data points for training, 50 for testing and 50 for validation ...
```

1.3 Support vector machine

```
In [5]: #To find the most optimal configuration manually,
        # We defined two possible types of kernel. A radial and a linear kernel
        kernels=['rbf','linear']
        #Posible values of C, that determine the size of the softmargin
        cs=[0.001, 0.01, 0.1, 1, 10]
        #Posible gamma values, which determine the radius of influence of each point
        gammas=[0.001, 0.01, 0.1, 1]

        #Initialize the variables that will save the best values found
        maxscore=-100
        maxkernel=''
        maxcs=0
        maxgamma=0
        mse = 0
        mse = 0
        mae = 0
        prediction=0
        best_SVM = None

        #Going through all the possible configurations of kernel, C and gamma
        for i in kernels:
            for j in cs:
                for k in gammas:
                    #New model
                    model=svm.SVR(kernel=i,C=j, gamma=k)
                    #Fit the model
                    model.fit(train_x, train_y)
                    #We obtain the values of prediction, score, MSE and MAE
                    pred   = model.predict(test_x)
                    scor   = model.score(test_x, test_y)
                    mse_   = mean_squared_error(test_y,pred)
                    mae_   = mean_absolute_error(test_y,pred)

                    #If the score is greater than the maximum score obtained,
                    # we save this model as the best, storing all its values
                    #of configuration and metrics.
                    if(scor>maxscore):
                        maxscore=scor
```

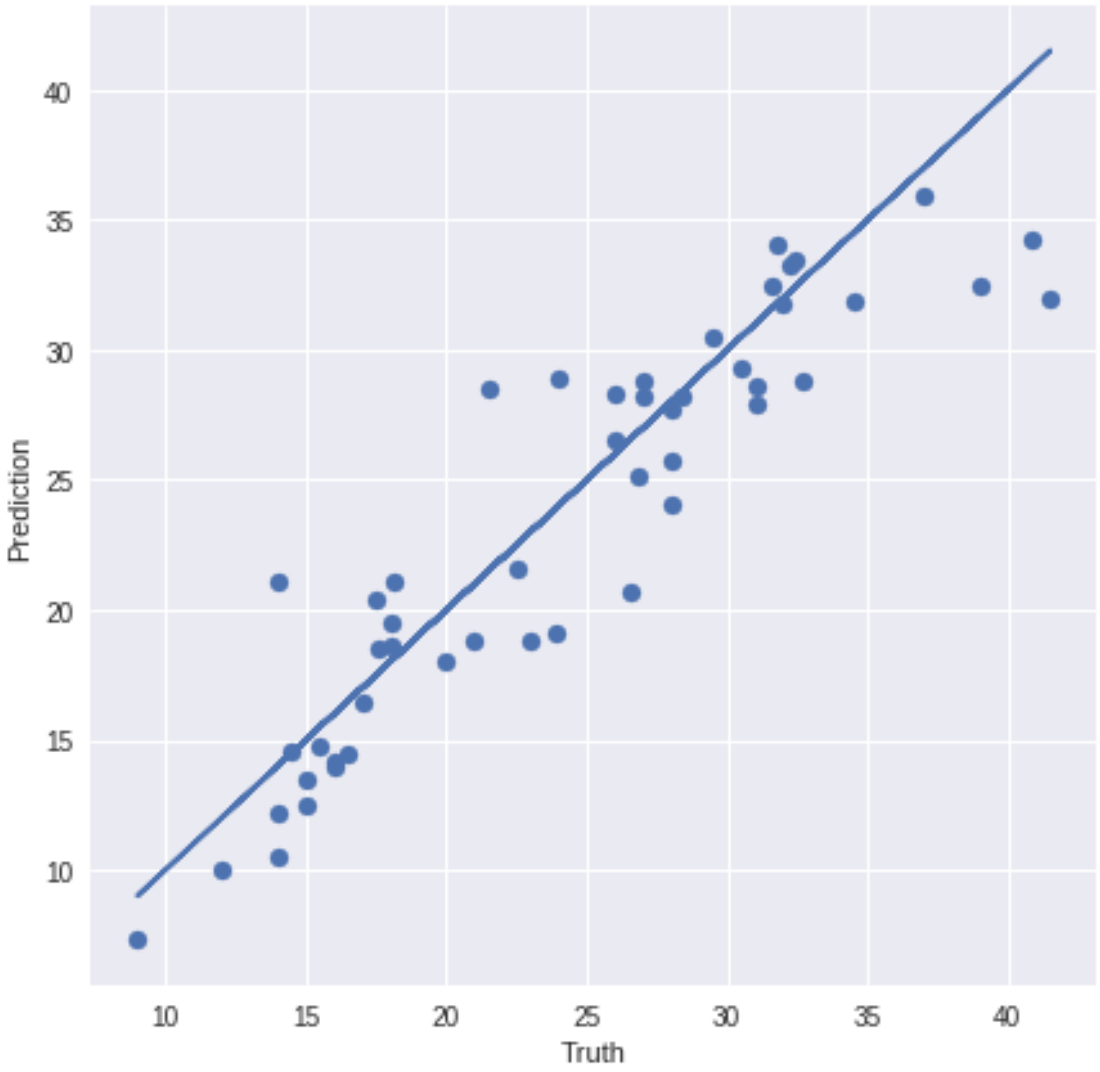
```
                    maxkernel=i
                    maxcs=j
                    maxgamma=k
                    mse    = mse_
                    mae    = mae_
                    prediction = pred
                    best_SVM = model



        print("Score "+
              str(maxscore)+" using params: C: "+str(maxcs)+", Kernel: "+maxkernel+", Gamma: "+
        print("Mean squared error (MSE): "+str(mse))
        print("Mean Absolute Error (MAE): "+str(mae))
        print("Root mean squared error (RMSE): "+str(math.sqrt(mse)))
```

```
Score 0.8345560117732478 using params: C: 1, Kernel: linear, Gamma: 0.001
Mean squared error (MSE): 10.638848530107234
Mean Absolute Error (MAE): 2.5145364278111817
Root mean squared error (RMSE): 3.261724778412065
```

```
In [6]: pyplot.figure(figsize=(7,7))
        pyplot.scatter(test_y,prediction)
        pyplot.plot(test_y, test_y)
        pyplot.xlabel('Truth')
        pyplot.ylabel('Prediction')
        pyplot.show()
```



1.4 Neural network

```
In [7]: # Prepare the data, standardizing features
        scaler = StandardScaler()
        scaler.fit(train_x)
        scaled_train_x = scaler.transform(train_x)
        scaled_test_x = scaler.transform(test_x)
        scaled_validation_x = scaler.transform(validation_x)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/data.py:625: DataConversionWarning
  return self.partial_fit(X, y)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: Data wi
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: DataConversionWarning: Data wi
```

```
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversionWarning: Data wi
  """
```

In [0]:
```
# Define some helper functions
def train_NN(training_features, training_targets, model, model_kwargs, seed):
    # Build a model of given type and arguments
    modelNN = model(**model_kwargs, random_state=seed)

    # Train it
    modelNN.fit(training_features, training_targets)

    return modelNN

def NN_testbed(training_features, training_targets, test_features, test_targets, model
    # Train a NN on given training data using the given model arguments
    # Return its accuracy and mse on the test data

    modelNN = train_NN(training_features, training_targets, model, model_kwargs, seed)

    # Calculate its score on the test data
    pred = modelNN.predict(test_features)

    score = modelNN.score(test_features, test_targets)
    mse = mean_squared_error(test_targets, pred)

    return score, mse

def generate_all_configurations(config):
    # Use given dictionary to generate all possible permutations of configs
    complete_configs = []
    template = {}

    # Start out with one template config
    complete_configs.append({})

    # Add a new config for each possible parameter and value tuple
    for param, values in config.items():
        current_configs = complete_configs.copy()
        complete_configs = [] # Make sure to discard the old ones, not containing a la

        for new_config in current_configs.copy():
                for value in values:
                    new_config.update({param: value})
                    complete_configs.append(new_config.copy())

    return complete_configs
```

In [9]:
```
# Generate a set of hyper paremeters to test
configurations = {'hidden_layer_sizes': [(10, ), (50,), (100,), (10, 10,), (50, 50, )]
                  'activation': ['relu', 'identity'],
                  'alpha': [0.0001, 0.01, 0.10],
                  'batch_size': ['auto'],
                  'learning_rate_init': [0.1, 0.01, 0.001],
                  'max_iter': [1000],
                 }

test_configs = generate_all_configurations(configurations)
print(f'Generated {len(test_configs)} different configurations to test ...')
```

```
Generated 90 different configurations to test ...
```

In [10]:
```
# Find the parameter setting that produces the highest r2 score

best_score = -1000
best_mse = 0
best_config = {}

for i, config in enumerate(test_configs):
    score, mse = NN_testbed(scaled_train_x, train_y, scaled_test_x, test_y, MLPRegress
    print(f'config {i} gives a score of {score}')

    if score > best_score:
        best_score = score
        best_mse = mse
        best_config = config

print(f'\n\nFound the best config to be:\n{best_config}\n with a score of {best_score
```

```
config 0 gives a score of 0.8137527562300947


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 1 gives a score of 0.8123243681693234


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 2 gives a score of 0.5872333739258879
config 3 gives a score of 0.8262249408789459
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 4 gives a score of 0.8123324165452341


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 5 gives a score of 0.5872427298946787
config 6 gives a score of 0.8471750967820075


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 7 gives a score of 0.813226232161675


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 8 gives a score of 0.5871981702942568
config 9 gives a score of 0.8587261700929848
config 10 gives a score of 0.8527589283897955


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 11 gives a score of 0.851765893683966
config 12 gives a score of 0.8587220276526873
config 13 gives a score of 0.8527595090726481


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 14 gives a score of 0.85176609717906
config 15 gives a score of 0.8586829542968911
config 16 gives a score of 0.8527647492479538
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 17 gives a score of 0.8517679146459164
config 18 gives a score of 0.8529688327336493
config 19 gives a score of 0.8310750423048794


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 20 gives a score of 0.8544144805268252
config 21 gives a score of 0.8533615925856457
config 22 gives a score of 0.8261238330443197


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 23 gives a score of 0.8544124204060934
config 24 gives a score of 0.8398791035964077
config 25 gives a score of 0.838856968342808


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 26 gives a score of 0.8544339245905472
config 27 gives a score of 0.8650988275413813
config 28 gives a score of 0.8764621420187609
config 29 gives a score of 0.8562384062846476
config 30 gives a score of 0.8651023077041236
config 31 gives a score of 0.8764623830673844
config 32 gives a score of 0.8562384855375903
config 33 gives a score of 0.8651338600155759
config 34 gives a score of 0.8764645601031199
config 35 gives a score of 0.8562391724993944
config 36 gives a score of 0.8244577256604105
config 37 gives a score of 0.833885718410091


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)
```

```
config 38 gives a score of 0.8555959080492923
config 39 gives a score of 0.8297128216246992
config 40 gives a score of 0.8155780329515655


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 41 gives a score of 0.8555510615980088
config 42 gives a score of 0.8393450872160295
config 43 gives a score of 0.8313648063506861


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 44 gives a score of 0.8542728050661058
config 45 gives a score of 0.8613309905639641
config 46 gives a score of 0.8492636910150275
config 47 gives a score of 0.8584112340668033
config 48 gives a score of 0.8613419164403375
config 49 gives a score of 0.8492631950596605
config 50 gives a score of 0.8584102134017519
config 51 gives a score of 0.861436204421642
config 52 gives a score of 0.8492586805597858
config 53 gives a score of 0.8584009214937937
config 54 gives a score of 0.8567064495279737
config 55 gives a score of 0.8481294749316877


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 56 gives a score of 0.8716392336912739
config 57 gives a score of 0.8573706639060704
config 58 gives a score of 0.8280170644029933


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 59 gives a score of 0.8716480664171049
config 60 gives a score of 0.8566605090044843
config 61 gives a score of 0.8307633671988872
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 62 gives a score of 0.8713668065102603
config 63 gives a score of 0.8443896813390092
config 64 gives a score of 0.8508545803337851
config 65 gives a score of 0.8576318481045039
config 66 gives a score of 0.8443854530643516
config 67 gives a score of 0.8508550269581724
config 68 gives a score of 0.8576311017115983
config 69 gives a score of 0.8443458028044836
config 70 gives a score of 0.8508591097554558
config 71 gives a score of 0.8576243139361626
config 72 gives a score of 0.738116346726654
config 73 gives a score of 0.8217295327473524


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 74 gives a score of 0.855657989501762
config 75 gives a score of 0.8587859721905029
config 76 gives a score of 0.8158233759013301


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 77 gives a score of 0.8525975684223893
config 78 gives a score of 0.8520319283319893
config 79 gives a score of 0.8304504907410343


/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)


config 80 gives a score of 0.8679544132558834
config 81 gives a score of 0.8518842152040479
config 82 gives a score of 0.8593650147323093
config 83 gives a score of 0.8671147612576592
config 84 gives a score of 0.8518668364063774
config 85 gives a score of 0.8593633923990038
config 86 gives a score of 0.8671141028595062
config 87 gives a score of 0.8564457703738773
config 88 gives a score of 0.8593485908627606
```

```
config 89 gives a score of 0.8671081070081244


Found the best config to be:
{'hidden_layer_sizes': (50,), 'activation': 'identity', 'alpha': 0.1, 'batch_size': 'auto', 'le
 with a score of 0.8764645601031199, and mse 7.943926202756725
```

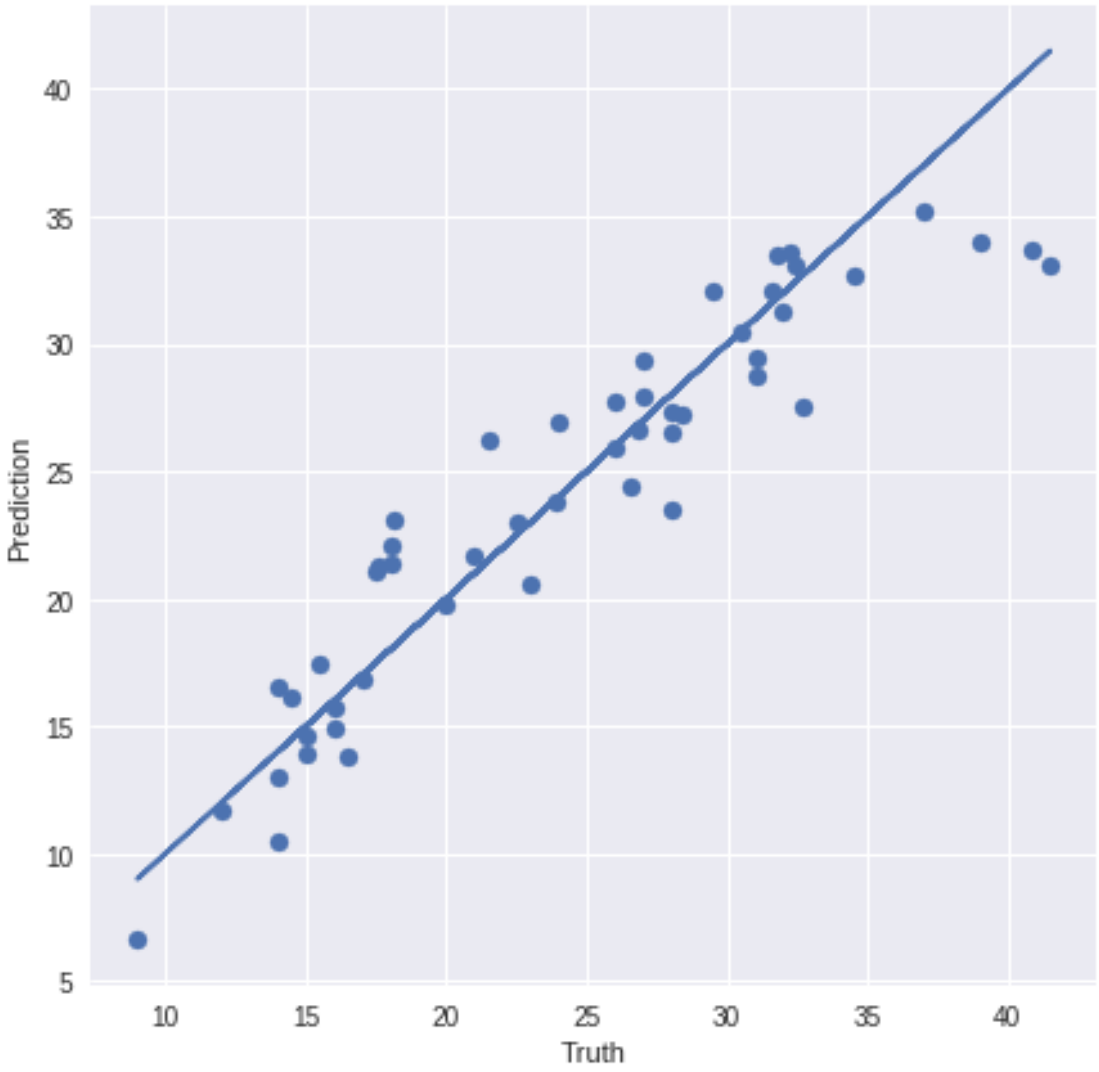
```
In [11]: best_NN = train_NN(scaled_train_x, train_y, MLPRegressor, best_config, 42)

         prediction = best_NN.predict(scaled_test_x)
         score = best_NN.score(scaled_test_x, test_y)
         mse = mean_squared_error(test_y, prediction)

         print(f'Best NN has a score of {score} with a mse {mse}')

         pyplot.figure(figsize=(7,7))
         pyplot.scatter(test_y, prediction)
         pyplot.plot(test_y, test_y)
         pyplot.xlabel('Truth')
         pyplot.ylabel('Prediction')
         pyplot.show()
```

```
Best NN has a score of 0.8764645601031199 with a mse 7.943926202756725
```



1.5 Compare the models

```
In [12]: # Compare the two models using the validation data
         svm_prediction   = best_SVM.predict(validation_x)
         svm_score   = best_SVM.score(validation_x, validation_y)
         svm_mse   = mean_squared_error(validation_y, svm_prediction)

         NN_prediction   = best_NN.predict(scaled_validation_x)
         NN_score   = best_NN.score(scaled_validation_x, validation_y)
         NN_mse   = mean_squared_error(validation_y, NN_prediction)

         print(f'SVM {svm_score} with a mse {svm_mse}')
         print(f'NN {NN_score} with a mse {NN_mse}')
```

```
        pyplot.figure(figsize=(7,7))
        pyplot.scatter(validation_y, NN_prediction, label='NN')
        pyplot.scatter(validation_y, svm_prediction, label='SVM')
        pyplot.plot(validation_y, validation_y)
        pyplot.xlabel('Truth')
        pyplot.ylabel('Prediction')
        pyplot.legend()
        pyplot.show()
```

```
SVM 0.8787425432847087 with a mse 7.4140223645710215
NN 0.8750287565816383 with a mse 7.641093741620101
```