3_Short_list_promising_models

April 11, 2021

Completed by:

Name : Adarsh Ghimire

Student ID : 100058927

MSc in Electrical and Computer Engineering

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```python
try:
  import os
  os.chdir('drive/MyDrive/COSC 606 Machine learning/Project/notebooks/')
except:
  print("Already in working directory")
```

[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

[ ]:
```python
# Training and test data were made in second notebook, and were saved to make␣
↪future use easier
training_dataset_path = "../final_dataset/train.csv"
test_dataset_path = "../final_dataset/test.csv"
train_df = pd.read_csv(training_dataset_path)
test_df = pd.read_csv(test_dataset_path)
train_df.head()
```

```
[ ]:   Report Date  Route      Time  … Direction Min Delay Min Gap
     0  2014-01-02    505  06:31:00  …       E/B       4.0     8.0
     1  2014-01-02    504  12:43:00  …       E/B      20.0    22.0
     2  2014-01-02    501  14:01:00  …       W/B      13.0    19.0
     3  2014-01-02    504  14:22:00  …       W/B       7.0    11.0
     4  2014-01-02    504  16:42:00  …       E/B       3.0     6.0

     [5 rows x 8 columns]
```

0.1 Data Manipulations functions

```
[ ]: # Since our target is Min Delay so the function below cleans the Min Delay rows␣
      ↪if they are nan values
     def clean_delay(df):
       df = df[df['Min Delay'].notna()]
       return df
```

```
[ ]: # This function checks for valid routes only
     def check_route(x):
       # load the valid list of TTC Streetcar routes
       valid_routes = [501, 502, 503, 504, 505, 506, 509, 510, 511, 512, 301, 304,␣
      ↪306, 310]

       if x in valid_routes:
         return x
       else:
         return "bad route"

     # This function cleans the data based on valid routes
     def clean_route(df):
       # This function takes dataframe as input
       # cleans the route column based on the validity of the route of street car
       # returns the cleaned dataframe
       df['Route'] = df['Route'].apply(lambda x:check_route(x))
       df = df[df.Route != "bad route"]
       df['Route'] = df['Route'].astype('int64')
       return df
```

```
[ ]: # This function drops the Location column
     def drop_location(df):
       df = df.drop(["Location"], axis=1)
       return df
```

```
[ ]: def create_date_time_column(df):
         # This function takes dataframe, then merges the date and time
         # Then convert that column into datetime datatype
```

```
    # Such that it can be further used in time series easily
  try:
    new = pd.to_datetime(df["Report Date"] + " "+ df["Time"], utc=True)
    df["Date Time"] = new
    df = df.drop(["Report Date", "Time"], axis=1)
    return df
  except:
    return df
```

[ ]:
```
# This function divides a day into different period
def day_divider(hour):
  if hour > 5 and hour < 12:
    return "morning"
  elif hour >= 12 and hour < 17:
    return "afternoon"
  elif hour >= 17 and hour < 21:
    return "evening"
  else:
    return "night"
```

[ ]:
```
def clean_gap(df):
  # This function will help to clean the Min Gap column feature with training␣
↪data Min Gap mean value
  df["Min Gap"] = df["Min Gap"].fillna(train_df['Min Gap'].mean())
  return df
```

[ ]:
```
# These function help to filter the Direction values and clean them
valid_directions = ['eb','wb','nb','sb','bw']
def check_direction (x):
    if x in valid_directions:
        return(x)
    else:
        return("bad direction")

def direction_cleanup(df):
    df['Direction'] = df['Direction'].str.lower()
    df['Direction'] = df['Direction'].str.replace('/','')
    df['Direction'] = df['Direction'].replace({'eastbound':'eb','westbound':
↪'wb','southbound':'sb','northbound':'nb'})
    df['Direction'] = df['Direction'].apply(lambda x:check_direction(x))
    return(df)
```

[ ]:
```
def complete_cleaner(df):
  df = clean_delay(df) # drops the nan Min delay rows
  df = clean_route(df) # cleans the unwanted route from the dataset
  df = drop_location(df) # drops the location column from the dataset
  df = create_date_time_column(df) # creates Date Time column in the dataset
```

```
  df["Part of Day"] = df.apply(lambda x: day_divider(x["Date Time"].
↪hour),axis=1) # Creates Part of Day column in the dataset
  df = clean_gap(df) # cleans gap based on the mean of gap values
  df = direction_cleanup(df) # cleans the direction column to 5␣
↪directions(eb,wb,nb,sb,bw)
  df = df[df["Direction"] != "bad direction"]
  df.reset_index(inplace=True, drop=True)
  df.drop(['Date Time'], axis=1, inplace=True)
  return df
```

[ ]:
```
# Creating a copy of training data to work on
df = train_df.copy(deep=True)
```

[ ]:
```
df = complete_cleaner(df)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[ ]:
```
df.sample(10)
```

[ ]:

	Route	Day	Direction	Min Delay	Min Gap	Part of Day
11615	512	Monday	eb	4.0	8.0	morning
25712	509	Saturday	wb	18.0	36.0	night
61661	506	Tuesday	wb	6.0	12.0	afternoon
57689	506	Thursday	wb	5.0	10.0	evening
17206	501	Monday	eb	10.0	20.0	afternoon
57370	504	Wednesday	eb	11.0	21.0	night
27191	504	Monday	eb	4.0	8.0	morning
51570	501	Friday	eb	60.0	67.0	night
64756	509	Saturday	eb	13.0	26.0	evening
22315	504	Wednesday	wb	7.0	11.0	evening

1 Now doing one hot encoding for the categorical data

1. Route is categorical
2. Day is categorical
3. Direction is categorical
4. Part of Day is categorical

[ ]:
```
# One hot encoder object made using training data, so that it can be used for␣
↪testing later on.
from sklearn.preprocessing import OneHotEncoder
import numpy as np
day_enc = OneHotEncoder()
route_enc = OneHotEncoder()
dir_enc = OneHotEncoder()
part_enc = OneHotEncoder()

day_enc.fit(np.array(df["Day"]).reshape(-1,1))
route_enc.fit(np.array(df["Route"]).reshape(-1,1))
dir_enc.fit(np.array(df["Direction"]).reshape(-1,1))
part_enc.fit(np.array(df["Part of Day"]).reshape(-1,1))
```

[ ]:
```
OneHotEncoder(categories='auto', drop=None, dtype=<class 'numpy.float64'>,
              handle_unknown='error', sparse=True)
```

[ ]:
```
def one_hot_encoder(df):

  df[day_enc.categories_[0]] = day_enc.transform(np.array(df["Day"]).
↪reshape(-1,1)).toarray()
  df[route_enc.categories_[0]] = route_enc.transform(np.array(df["Route"]).
↪reshape(-1,1)).toarray()
  df[dir_enc.categories_[0]] = dir_enc.transform(np.array(df["Direction"]).
↪reshape(-1,1)).toarray()
  df[part_enc.categories_[0]] = part_enc.transform(np.array(df["Part of Day"]).
↪reshape(-1,1)).toarray()

  df.drop(['Day', 'Route', 'Direction', 'Part of Day'], axis=1, inplace=True)
  return df
```

[ ]:
```
df = one_hot_encoder(df)
```

[ ]:
```
df
```

[ ]:
```
       Min Delay  Min Gap  Friday  Monday  …  afternoon  evening  morning
night
0            4.0      8.0     0.0     0.0  …        0.0      0.0      1.0
0.0
1           20.0     22.0     0.0     0.0  …        1.0      0.0      0.0
```

```
0.0
2            13.0     19.0     0.0     0.0  …         1.0      0.0      0.0
0.0
3             7.0     11.0     0.0     0.0  …         1.0      0.0      0.0
0.0
4             3.0      6.0     0.0     0.0  …         1.0      0.0      0.0
0.0
…          …      …      …      …  …         …      …      …
…
71507         8.0     16.0     0.0     0.0  …         0.0      0.0      1.0
0.0
71508         7.0     14.0     0.0     0.0  …         0.0      0.0      1.0
0.0
71509         5.0     10.0     0.0     0.0  …         0.0      0.0      1.0
0.0
71510         5.0     10.0     0.0     0.0  …         0.0      0.0      1.0
0.0
71511         9.0     16.0     0.0     0.0  …         0.0      0.0      1.0
0.0

[71512 rows x 32 columns]
```

1.1 Need to do feature scaling for Min Gap column feature

[ ]:
```python
# Feature Standardization, which means changing between -1 and 1, such that␣
↪algorithm can coverge swiftly
min_gap_train_mean = np.mean(df["Min Gap"])
min_gap_train_std = np.std(df["Min Gap"])
print(min_gap_train_mean)
print(min_gap_train_std)

def min_gap_scaler(df):
  df["Min Gap"] = (df["Min Gap"]-min_gap_train_mean)/min_gap_train_std
  return df
```

```
18.13560073354417
33.68768666213863
```

[ ]:
```python
df = min_gap_scaler(df)
```

[ ]:
```python
df.head()
```

[ ]:
```
   Min Delay   Min Gap  Friday  Monday  …  afternoon  evening  morning  night
0        4.0 -0.300870     0.0     0.0  …        0.0      0.0      1.0    0.0
1       20.0  0.114713     0.0     0.0  …        1.0      0.0      0.0    0.0
2       13.0  0.025659     0.0     0.0  …        1.0      0.0      0.0    0.0
3        7.0 -0.211816     0.0     0.0  …        1.0      0.0      0.0    0.0
```

```
4         3.0 -0.360238      0.0      0.0  …         1.0        0.0        0.0     0.0

[5 rows x 32 columns]
```

2 The data is ready for training

[ ]:
```python
def complete_data_preprocessing(df):
  # This function is for test set
  df = complete_cleaner(df)
  df = one_hot_encoder(df) # one hot encoder based on training samples
  df = min_gap_scaler(df) # feature scaler based on training sample
  print("------------------------")
  if df.isnull().values.any():
    print("There are some null values")
  else:
    print("Data is ready for testing")
  return df
```

[ ]:
```python
processed_test_data = complete_data_preprocessing(test_df)
```

```
------------------------
Data is ready for testing

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

2.0.1 The regression results function computes the different metrics like r2 score, MAE, and RMSE.

[ ]:
```python
import sklearn.metrics as metrics
def regression_results(y_true, y_pred):
    # Regression metrics
    mean_absolute_error=metrics.mean_absolute_error(y_true, y_pred)
    mse=metrics.mean_squared_error(y_true, y_pred)
```

```
    r2 = metrics.r2_score(y_true, y_pred)
    print("R2:", round(r2, 4))
    print('MAE: ', round(mean_absolute_error,4))
    print('RMSE: ', round(np.sqrt(mse),4))
```

[ ]:
```
# Selected features for model training
selected_features = list(df.columns)
selected_features.remove("Min Delay")
target = "Min Delay"
```

Training and validation dataset will be used throughout the model selection and fine tuning process

Final test set has been kept seperate, it will be used only after the model is finalized.

[ ]:
```
X = df[selected_features]
y = df[target]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1,␣
 ↪random_state=42)
```

[ ]:
```
def evaluate_model(model):
  print("Training Regression results:")
  regression_results(y_train, model.predict(X_train))
  print("\n")
  print("Validation Regression results")
  regression_results(y_val, model.predict(X_val))
```

Model 1 - Simple linear regression

[ ]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
print("Linear regression results")
evaluate_model(lin_reg)
```

```
Linear regression results
Training Regression results:
R2: 0.6239
MAE:  3.9846
RMSE:  18.5552


Validation Regression results
R2: 0.8012
MAE:  3.799
RMSE:  11.0773
```

Model 2 - Polynomial regression of degree 2

```
[ ]: poly_2 = PolynomialFeatures(degree = 2)
X_poly_2 = poly_2.fit_transform(X)
poly_2.fit(X_poly_2, y)

X_poly_2_train, X_poly_2_val, y_train_poly_2, y_val_poly_2 = train_test_split(
    X_poly_2,
    y,
    test_size=0.1,
    random_state=42)

reg_2 = LinearRegression()
reg_2.fit(X_poly_2_train, y_train_poly_2)

print("Training polynomial Regression of degree 2 results:")
regression_results(y_train_poly_2, reg_2.predict(X_poly_2_train))
print("\n")
print("Validation polynomial Regression of degree 2 results")
regression_results(y_val_poly_2, reg_2.predict(X_poly_2_val))
```

```
Training polynomial Regression of degree 2 results:
R2: 0.8298
MAE:  2.9791
RMSE:  12.4826


Validation polynomial Regression of degree 2 results
R2: 0.852
MAE:  2.9469
RMSE:  9.558
```

Model 3 - Bayesian regression

```
[ ]: from sklearn import linear_model
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1,␣
 ↪random_state=42)

BayReg = linear_model.BayesianRidge()
BayReg.fit(X_train, y_train)
print("Bayesian regression results")
evaluate_model(BayReg)
```

```
Bayesian regression results
Training Regression results:
R2: 0.6239
MAE:  3.9854
RMSE:  18.5552
```

```
Validation Regression results
R2: 0.8011
MAE:  3.7998
RMSE:  11.0804
```

Model 4 - SVM Regressor

```
[ ]: # It takes forever to train, waiter for more then 20 minutes, but still
     # no results were given so, terminated

     # from sklearn.svm import SVR
     # X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1,␣
      ↪random_state=42)
     # svr = SVR(kernel='poly', C=1.0, epsilon=0.2)
     # svr.fit(X_train, y_train)
     # print("SVM regression results")
     # evaluate_model(svr)

     # SVM regressor took forever to train, so terminated it in the middle of␣
      ↪training.
     # SVM regressor not suitable for this task
```

Model 5 - Random Forest

```
[ ]: from sklearn.ensemble import RandomForestRegressor
     X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1,␣
      ↪random_state=42)
     rf = RandomForestRegressor(random_state=42)
     rf.fit(X_train, y_train)
     print("Random Forest regression results")
     evaluate_model(rf)
```

```
Random Forest regression results
Training Regression results:
R2: 0.9382
MAE:  1.3267
RMSE:  7.523


Validation Regression results
R2: 0.7878
MAE:  2.3279
RMSE:  11.4465
```

Model 6 - Neural Network

```
[ ]: from sklearn.neural_network import MLPRegressor
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1,␣
 ↪random_state=42)
nn = MLPRegressor(random_state=1, max_iter=500).fit(X_train, y_train)
print("Neural network regression results")
evaluate_model(nn)
```

```
Neural network regression results
Training Regression results:
R2: 0.8424
MAE:  2.7425
RMSE:  12.0125


Validation Regression results
R2: 0.8636
MAE:  2.6741
RMSE:  9.177
```

Model 7 - XGBoost

```
[ ]: from sklearn.ensemble import GradientBoostingRegressor
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1,␣
 ↪random_state=42)
boost = GradientBoostingRegressor().fit(X_train, y_train)
print("Gradient Boosting network regression results")
evaluate_model(boost)
```

```
Gradient Boosting network regression results
Training Regression results:
R2: 0.876
MAE:  2.1704
RMSE:  10.6554


Validation Regression results
R2: 0.8709
MAE:  2.1503
RMSE:  8.9268
```

3 K-Fold Cross validation

3.1 Cross validation function for model training

```
[ ]: def cross_validation(model, X, y, n=10): # default 10 splits
       from sklearn.model_selection import KFold
       kf = KFold(n_splits=n)
       kf.get_n_splits(X)

       train_scores = {}
       val_scores = {}
       train_scores["R2"], train_scores["MAE"] ,train_scores["RMSE"] = [], [], []
       val_scores["R2"], val_scores["MAE"], val_scores["RMSE"] = [], [], []
       model_count = 1
       for train_index, val_index in kf.split(X):

         X_train, X_val = X.loc[train_index], X.loc[val_index]
         y_train, y_val = y[train_index], y[val_index]
         model.fit(X_train, y_train)
         print("**********************************************")
         print("Model {} trained".format(model_count))
         print("Training Scores")
         print("R2 Score : {} \t MAE : {} \t RMSE : {}".format(round(metrics.
      ↪r2_score(y_train, model.predict(X_train)),4),
                                                               round(metrics.
      ↪mean_absolute_error(y_train, model.predict(X_train)),4),
                                                               round(np.sqrt(metrics.
      ↪mean_squared_error(y_train, model.predict(X_train))),4)
                                                               ))
         print("Validation Scores")
         print("R2 Score : {} \t MAE : {} \t RMSE : {}".format(round(metrics.
      ↪r2_score(y_val, model.predict(X_val)),4),
                                                               round(metrics.
      ↪mean_absolute_error(y_val, model.predict(X_val)),4),
                                                               round(np.sqrt(metrics.
      ↪mean_squared_error(y_val, model.predict(X_val))),4)
                                                               ))
         train_scores["R2"].append(round(metrics.r2_score(y_train, model.
      ↪predict(X_train)),4))
         train_scores["MAE"].append(round(metrics.mean_absolute_error(y_train, model.
      ↪predict(X_train)),4))
         train_scores["RMSE"].append(round(np.sqrt(metrics.
      ↪mean_squared_error(y_train, model.predict(X_train))),4))

         val_scores["R2"].append(round(metrics.r2_score(y_val, model.
      ↪predict(X_val)),4))
```

```
    val_scores["MAE"].append(round(metrics.mean_absolute_error(y_val, model.
↪predict(X_val)),4))
    val_scores["RMSE"].append(round(np.sqrt(metrics.mean_squared_error(y_val,␣
↪model.predict(X_val))),4))


    model_count += 1

  return train_scores, val_scores
```

[ ]:
```
def print_results(train_scores, val_scores):
    """
    train_scores, and val_scores should be a dictionary containing R2, MAE and␣
↪RMSE
    """
    print("\n")
    print("Mean and standard deviation of {}-fold cross validation results".
↪format(len(train_scores["R2"])))
    print("-----------------------------------------------------------------")
    print("Train Mean R2 Score: \t {} \t\t Validation Mean R2 Score: \t {}".
↪format(round(np.mean(train_scores["R2"]),4), round(np.
↪mean(val_scores["R2"]),4)))
    print("Train Std R2 Score: \t {} \t\t Validation Std R2 Score: \t {}".
↪format(round(np.std(train_scores["R2"]),4), round(np.
↪std(val_scores["R2"]),4)))
    print("\n")
    print("Train Mean MAE Score: \t {} \t\t Validation Mean MAE Score: \t {}".
↪format(round(np.mean(train_scores["MAE"]),4), round(np.
↪mean(val_scores["MAE"]),4)))
    print("Train Std MAE Score: \t {} \t\t Validation Std MAE Score: \t {}".
↪format(round(np.std(train_scores["MAE"]),4), round(np.
↪std(val_scores["MAE"]),4)))
    print("\n")
    print("Train Mean RMSE Score: \t {} \t\t Validation Mean RMSE Score: \t {}".
↪format(round(np.mean(train_scores["RMSE"]),4), round(np.
↪mean(val_scores["RMSE"]),4)))
    print("Train Std RMSE Score: \t {} \t\t Validation Std RMSE Score: \t {}".
↪format(round(np.std(train_scores["RMSE"]),4), round(np.
↪std(val_scores["RMSE"]),4)))
```

3.2 Linear regression

[ ]:
```
train_scores, val_scores = cross_validation(LinearRegression(), X, y, 10)
```

```
**********************************************
```

```
Model 1 trained
Training Scores
R2 Score : 0.6216       MAE : 3.9786    RMSE : 18.4698
Validation Scores
R2 Score : 0.7948       MAE : 3.8018    RMSE : 12.3318
********************************************
Model 2 trained
Training Scores
R2 Score : 0.6201       MAE : 3.9552    RMSE : 18.1851
Validation Scores
R2 Score : 0.756        MAE : 4.2323    RMSE : 15.7741
********************************************
Model 3 trained
Training Scores
R2 Score : 0.6266       MAE : 3.8001    RMSE : 17.9763
Validation Scores
R2 Score : 0.7073       MAE : 4.2025    RMSE : 17.6362
********************************************
Model 4 trained
Training Scores
R2 Score : 0.6221       MAE : 3.9856    RMSE : 18.3186
Validation Scores
R2 Score : 0.7646       MAE : 4.0355    RMSE : 14.2505
********************************************
Model 5 trained
Training Scores
R2 Score : 0.8285       MAE : 2.6946    RMSE : 12.3923
Validation Scores
R2 Score : -1.8484      MAE : 3.417     RMSE : 47.584
********************************************
Model 6 trained
Training Scores
R2 Score : 0.6222       MAE : 4.0218    RMSE : 17.8355
Validation Scores
R2 Score : 0.7146       MAE : 4.407     RMSE : 19.0997
********************************************
Model 7 trained
Training Scores
R2 Score : 0.6267       MAE : 4.03      RMSE : 18.7283
Validation Scores
R2 Score : 0.8398       MAE : 3.563     RMSE : 7.9691
********************************************
Model 8 trained
Training Scores
R2 Score : 0.6207       MAE : 4.0756    RMSE : 18.3995
Validation Scores
R2 Score : 0.7859       MAE : 3.4817    RMSE : 13.2421
********************************************
```

```
Model 9 trained
Training Scores
R2 Score : 0.6175        MAE : 4.0498    RMSE : 18.5795
Validation Scores
R2 Score : 0.8417        MAE : 3.6346    RMSE : 10.7612
**********************************************
Model 10 trained
Training Scores
R2 Score : 0.5989        MAE : 4.0737    RMSE : 18.5448
Validation Scores
R2 Score : 0.8809        MAE : 4.2483    RMSE : 11.6323
```

[ ]:
```
print("Linear Regression results : ")
print_results(train_scores, val_scores)
```

```
Linear Regression results :


Mean and standard deviation of 10-fold cross validation results
-----------------------------------------------------------------
Train Mean R2 Score:     0.6405                  Validation Mean R2 Score:
0.5237
Train Std R2 Score:      0.0631                  Validation Std R2 Score:
0.7925


Train Mean MAE Score:    3.8665                  Validation Mean MAE Score:
3.9024
Train Std MAE Score:     0.3979                  Validation Std MAE Score:
0.3465


Train Mean RMSE Score:   17.743                  Validation Mean RMSE Score:
17.0281
Train Std RMSE Score:    1.8026                  Validation Std RMSE Score:
10.6511
```

3.3 Bayesian regression

[ ]:
```
train_scores, val_scores = cross_validation(linear_model.BayesianRidge(), X, y,
 ↪10)
```

```
**********************************************
Model 1 trained
Training Scores
R2 Score : 0.6216        MAE : 3.9792    RMSE : 18.4699
Validation Scores
R2 Score : 0.7947        MAE : 3.8023    RMSE : 12.3347
```

```
**********************************************
Model 2 trained
Training Scores
R2 Score : 0.6201        MAE : 3.9595    RMSE : 18.1852
Validation Scores
R2 Score : 0.756         MAE : 4.2339    RMSE : 15.7742
**********************************************
Model 3 trained
Training Scores
R2 Score : 0.6266        MAE : 3.801     RMSE : 17.9763
Validation Scores
R2 Score : 0.7073        MAE : 4.2038    RMSE : 17.6362
**********************************************
Model 4 trained
Training Scores
R2 Score : 0.6221        MAE : 3.9865    RMSE : 18.3186
Validation Scores
R2 Score : 0.7645        MAE : 4.0355    RMSE : 14.2527
**********************************************
Model 5 trained
Training Scores
R2 Score : 0.8285        MAE : 2.6946    RMSE : 12.3923
Validation Scores
R2 Score : -1.8479       MAE : 3.4173    RMSE : 47.5796
**********************************************
Model 6 trained
Training Scores
R2 Score : 0.6222        MAE : 4.0227    RMSE : 17.8356
Validation Scores
R2 Score : 0.7145        MAE : 4.4076    RMSE : 19.1025
**********************************************
Model 7 trained
Training Scores
R2 Score : 0.6267        MAE : 4.0309    RMSE : 18.7284
Validation Scores
R2 Score : 0.8397        MAE : 3.5633    RMSE : 7.9706
**********************************************
Model 8 trained
Training Scores
R2 Score : 0.6207        MAE : 4.0764    RMSE : 18.3996
Validation Scores
R2 Score : 0.7859        MAE : 3.4828    RMSE : 13.2443
**********************************************
Model 9 trained
Training Scores
R2 Score : 0.6175        MAE : 4.0507    RMSE : 18.5795
Validation Scores
R2 Score : 0.8417        MAE : 3.6356    RMSE : 10.7636
```

```
**********************************************
Model 10 trained
Training Scores
R2 Score : 0.5989        MAE : 4.0748    RMSE : 18.5448
Validation Scores
R2 Score : 0.8808        MAE : 4.2496    RMSE : 11.635
```

[ ]:
```
print("Bayesian Regression results : ")
print_results(train_scores, val_scores)
```

```
Bayesian Regression results :


Mean and standard deviation of 10-fold cross validation results
-----------------------------------------------------------------
Train Mean R2 Score:     0.6405                 Validation Mean R2 Score:
0.5237
Train Std R2 Score:      0.0631                 Validation Std R2 Score:
0.7923


Train Mean MAE Score:    3.8676                 Validation Mean MAE Score:
3.9032
Train Std MAE Score:     0.3983                 Validation Std MAE Score:
0.3467


Train Mean RMSE Score:   17.743                 Validation Mean RMSE Score:
17.0293
Train Std RMSE Score:    1.8026                 Validation Std RMSE Score:
10.6492
```

3.4 Random Forest regression

3.4.1 With default parameters

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
train_scores, val_scores =␣
↪cross_validation(RandomForestRegressor(random_state=42), X, y, 10)
```

```
**********************************************
Model 1 trained
Training Scores
R2 Score : 0.936         MAE : 1.3269    RMSE : 7.5963
Validation Scores
R2 Score : 0.8438        MAE : 2.5018    RMSE : 10.758
**********************************************
Model 2 trained
```

```
Training Scores
R2 Score : 0.9392       MAE : 1.3003    RMSE : 7.2726
Validation Scores
R2 Score : 0.7908       MAE : 2.7948    RMSE : 14.6076
**********************************************
Model 3 trained
Training Scores
R2 Score : 0.9313       MAE : 1.3428    RMSE : 7.7094
Validation Scores
R2 Score : 0.8554       MAE : 2.4851    RMSE : 12.3964
**********************************************
Model 4 trained
Training Scores
R2 Score : 0.942        MAE : 1.3253    RMSE : 7.1799
Validation Scores
R2 Score : 0.8146       MAE : 2.1795    RMSE : 12.6465
**********************************************
Model 5 trained
Training Scores
R2 Score : 0.9398       MAE : 1.301     RMSE : 7.3453
Validation Scores
R2 Score : 0.5226       MAE : 2.8284    RMSE : 19.481
**********************************************
Model 6 trained
Training Scores
R2 Score : 0.9434       MAE : 1.2755    RMSE : 6.9034
Validation Scores
R2 Score : 0.736        MAE : 2.9011    RMSE : 18.3674
**********************************************
Model 7 trained
Training Scores
R2 Score : 0.934        MAE : 1.3915    RMSE : 7.8732
Validation Scores
R2 Score : 0.8035       MAE : 1.9532    RMSE : 8.8249
**********************************************
Model 8 trained
Training Scores
R2 Score : 0.9353       MAE : 1.3866    RMSE : 7.5979
Validation Scores
R2 Score : 0.8439       MAE : 1.6439    RMSE : 11.306
**********************************************
Model 9 trained
Training Scores
R2 Score : 0.9309       MAE : 1.3724    RMSE : 7.8973
Validation Scores
R2 Score : 0.8997       MAE : 2.0552    RMSE : 8.5652
**********************************************
Model 10 trained
```

```
Training Scores
R2 Score : 0.9261        MAE : 1.3753    RMSE : 7.9614
Validation Scores
R2 Score : 0.9374        MAE : 2.2813    RMSE : 8.4341
```

[ ]:
```python
print("Random Forest regression results")
print_results(train_scores, val_scores)
```

```
Random Forest regression results


Mean and standard deviation of 10-fold cross validation results
-----------------------------------------------------------------
Train Mean R2 Score:     0.9358                  Validation Mean R2 Score:
0.8048
Train Std R2 Score:      0.0052                  Validation Std R2 Score:
0.1081


Train Mean MAE Score:    1.3398                  Validation Mean MAE Score:
2.3624
Train Std MAE Score:     0.0384                  Validation Std MAE Score:
0.3933


Train Mean RMSE Score:   7.5337                  Validation Mean RMSE Score:
12.5387
Train Std RMSE Score:    0.331          Validation Std RMSE Score:      3.7095
```

3.5 Gradient Boosting regression

3.5.1 With default parameters

[ ]:
```python
from sklearn.ensemble import GradientBoostingRegressor
train_scores, val_scores = cross_validation(GradientBoostingRegressor(), X, y,␣
↪10)
```

```
**********************************************
Model 1 trained
Training Scores
R2 Score : 0.8717        MAE : 2.1268    RMSE : 10.755
Validation Scores
R2 Score : 0.8678        MAE : 2.4526    RMSE : 9.8983
**********************************************
Model 2 trained
Training Scores
R2 Score : 0.8795        MAE : 2.0974    RMSE : 10.2405
Validation Scores
```

```
R2 Score : 0.827        MAE : 2.6866    RMSE : 13.2818
**********************************************
Model 3 trained
Training Scores
R2 Score : 0.8629       MAE : 2.159     RMSE : 10.8944
Validation Scores
R2 Score : 0.8872       MAE : 2.5061    RMSE : 10.9496
**********************************************
Model 4 trained
Training Scores
R2 Score : 0.8764       MAE : 2.1553    RMSE : 10.4759
Validation Scores
R2 Score : 0.8327       MAE : 2.2145    RMSE : 12.0133
**********************************************
Model 5 trained
Training Scores
R2 Score : 0.8876       MAE : 2.1058    RMSE : 10.0339
Validation Scores
R2 Score : 0.6278       MAE : 2.6106    RMSE : 17.2006
**********************************************
Model 6 trained
Training Scores
R2 Score : 0.8899       MAE : 2.0705    RMSE : 9.6293
Validation Scores
R2 Score : 0.7867       MAE : 2.6321    RMSE : 16.5122
**********************************************
Model 7 trained
Training Scores
R2 Score : 0.8716       MAE : 2.2348    RMSE : 10.9837
Validation Scores
R2 Score : 0.8654       MAE : 1.8158    RMSE : 7.3045
**********************************************
Model 8 trained
Training Scores
R2 Score : 0.8734       MAE : 2.2182    RMSE : 10.629
Validation Scores
R2 Score : 0.8581       MAE : 1.6427    RMSE : 10.7821
**********************************************
Model 9 trained
Training Scores
R2 Score : 0.8628       MAE : 2.2012    RMSE : 11.1295
Validation Scores
R2 Score : 0.9203       MAE : 1.9805    RMSE : 7.6348
**********************************************
Model 10 trained
Training Scores
R2 Score : 0.8565       MAE : 2.1976    RMSE : 11.0939
Validation Scores
```

```
R2 Score : 0.9683        MAE : 2.1003    RMSE : 6.0022
```

[ ]:
```python
print("XGBoost regression results")
print_results(train_scores, val_scores)
```

```
XGBoost regression results


Mean and standard deviation of 10-fold cross validation results
-----------------------------------------------------------------
Train Mean R2 Score:     0.8732                 Validation Mean R2 Score:
0.8441
Train Std R2 Score:      0.0102                 Validation Std R2 Score:
0.0865


Train Mean MAE Score:    2.1567                 Validation Mean MAE Score:
2.2642
Train Std MAE Score:     0.0529                 Validation Std MAE Score:
0.35


Train Mean RMSE Score:   10.5865                Validation Mean RMSE Score:
11.1579
Train Std RMSE Score:    0.4678                 Validation Std RMSE Score:
3.5502
```

3.6 Neural Network regression

3.6.1 Default is 1 hidden layer with 100 neurons

[ ]:
```python
from sklearn.neural_network import MLPRegressor
train_scores, val_scores = cross_validation(MLPRegressor(random_state=1,␣
↪max_iter=500), X, y, 10)
```

```
**********************************************
Model 1 trained
Training Scores
R2 Score : 0.8442        MAE : 2.6132    RMSE : 11.8528
Validation Scores
R2 Score : 0.844         MAE : 2.8365    RMSE : 10.7524
**********************************************
Model 2 trained
Training Scores
R2 Score : 0.8553        MAE : 2.0893    RMSE : 11.2233
Validation Scores
R2 Score : 0.8328        MAE : 2.4967    RMSE : 13.0572
**********************************************
```

```
Model 3 trained
Training Scores
R2 Score : 0.8337        MAE : 2.6943    RMSE : 11.9967
Validation Scores
R2 Score : 0.9152        MAE : 2.7922    RMSE : 9.4942
**********************************************
Model 4 trained
Training Scores
R2 Score : 0.8565        MAE : 2.3367    RMSE : 11.2896
Validation Scores
R2 Score : 0.8268        MAE : 2.4525    RMSE : 12.2233
```

```
/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
**********************************************
Model 5 trained
Training Scores
R2 Score : 0.8811        MAE : 2.6773    RMSE : 10.3188
Validation Scores
R2 Score : 0.6288        MAE : 3.0712    RMSE : 17.177
**********************************************
Model 6 trained
Training Scores
R2 Score : 0.8545        MAE : 2.7583    RMSE : 11.0693
Validation Scores
R2 Score : 0.7661        MAE : 3.2038    RMSE : 17.2891
**********************************************
Model 7 trained
Training Scores
R2 Score : 0.8517        MAE : 2.6029    RMSE : 11.8045
Validation Scores
R2 Score : 0.8762        MAE : 2.3416    RMSE : 7.004
**********************************************
Model 8 trained
Training Scores
R2 Score : 0.8421        MAE : 3.4655    RMSE : 11.8719
Validation Scores
R2 Score : 0.8178        MAE : 3.0049    RMSE : 12.2158
**********************************************
Model 9 trained
Training Scores
R2 Score : 0.839         MAE : 2.7145    RMSE : 12.0548
Validation Scores
R2 Score : 0.9034        MAE : 2.6728    RMSE : 8.4049
**********************************************
```

```
Model 10 trained
Training Scores
R2 Score : 0.8261        MAE : 2.698     RMSE : 12.2111
Validation Scores
R2 Score : 0.971         MAE : 2.6819    RMSE : 5.741
```

[ ]:
```python
print("Neural network regression results")
print_results(train_scores, val_scores)
```

```
Neural network regression results


Mean and standard deviation of 10-fold cross validation results
-----------------------------------------------------------------
Train Mean R2 Score:     0.8484                 Validation Mean R2 Score:
0.8382
Train Std R2 Score:      0.0145                 Validation Std R2 Score:
0.0888


Train Mean MAE Score:    2.665          Validation Mean MAE Score:      2.7554
Train Std MAE Score:     0.3316                 Validation Std MAE Score:
0.267


Train Mean RMSE Score:   11.5693                Validation Mean RMSE Score:
11.3359
Train Std RMSE Score:    0.554          Validation Std RMSE Score:      3.6943
```

3.7 Neural Network regression

3.7.1 2 hidden layer with 64 neurons and 32 neurons

[70]:
```python
from sklearn.neural_network import MLPRegressor
train_scores, val_scores =␣
↪cross_validation(MLPRegressor(hidden_layer_sizes=(64, 64), random_state=1,␣
↪max_iter=500), X, y, 10)
```

```
**********************************************
Model 1 trained
Training Scores
R2 Score : 0.8537        MAE : 2.4604    RMSE : 11.4837
Validation Scores
R2 Score : 0.8638        MAE : 2.7862    RMSE : 10.0478
**********************************************
Model 2 trained
Training Scores
R2 Score : 0.8425        MAE : 3.1197    RMSE : 11.7073
```

```
Validation Scores
R2 Score : 0.8267        MAE : 3.4537    RMSE : 13.2953
*********************************************
Model 3 trained
Training Scores
R2 Score : 0.8329        MAE : 2.6483    RMSE : 12.0261
Validation Scores
R2 Score : 0.9025        MAE : 2.8511    RMSE : 10.1812
*********************************************
Model 4 trained
Training Scores
R2 Score : 0.8446        MAE : 3.3345    RMSE : 11.7464
Validation Scores
R2 Score : 0.8172        MAE : 3.3833    RMSE : 12.5576
*********************************************
Model 5 trained
Training Scores
R2 Score : 0.8849        MAE : 2.3674    RMSE : 10.1522
Validation Scores
R2 Score : 0.592         MAE : 2.8878    RMSE : 18.0079
*********************************************
Model 6 trained
Training Scores
R2 Score : 0.8635        MAE : 2.456     RMSE : 10.7184
Validation Scores
R2 Score : 0.7768        MAE : 3.0608    RMSE : 16.889
*********************************************
Model 7 trained
Training Scores
R2 Score : 0.8505        MAE : 2.8334    RMSE : 11.8505
Validation Scores
R2 Score : 0.8768        MAE : 2.6092    RMSE : 6.9869
*********************************************
Model 8 trained
Training Scores
R2 Score : 0.8533        MAE : 2.7682    RMSE : 11.4416
Validation Scores
R2 Score : 0.8308        MAE : 2.4281    RMSE : 11.7737
*********************************************
Model 9 trained
Training Scores
R2 Score : 0.8372        MAE : 3.972     RMSE : 12.1236
Validation Scores
R2 Score : 0.8973        MAE : 3.7011    RMSE : 8.6678
*********************************************
Model 10 trained
Training Scores
R2 Score : 0.833         MAE : 3.017     RMSE : 11.9659
```

```
Validation Scores
R2 Score : 0.9709        MAE : 2.8536    RMSE : 5.7455
```

[71]:
```python
print("Neural network regression results")
print_results(train_scores, val_scores)
```

```
Neural network regression results


Mean and standard deviation of 10-fold cross validation results
-----------------------------------------------------------------
Train Mean R2 Score:     0.8496                 Validation Mean R2 Score:
0.8355
Train Std R2 Score:      0.015          Validation Std R2 Score:        0.0961


Train Mean MAE Score:    2.8977                 Validation Mean MAE Score:
3.0015
Train Std MAE Score:     0.4647                 Validation Std MAE Score:
0.3781


Train Mean RMSE Score:   11.5216                Validation Mean RMSE Score:
11.4153
Train Std RMSE Score:    0.5946                 Validation Std RMSE Score:
3.7562
```