Homework 4

Group 45

Prachi Patel

Jignasuben Vekariya

424-394-6096 (Tel. of Prachi)

857-262-0803 (Tel of Jignasu)

patel.prachi2@northeastern.edu

vekariya.j@northeastern.edu

Percentage of Effort Contributed by Student 1: 50%

Percentage of Effort Contributed by Student 2: 50%

Signature of Student 1: Prachi Patel

Signature of Student 2: J.R.Vekariya

Submission Date: 03/25/2022

Assignment_4

March 25, 2022

1 Homework 4

Before you start: Read Chapter 6 Linear Regression and Chapter 7 K-Nearst-Neighbors in the textbook.

Note: Please enter the code along with your comments in the **TODO** section.

Alternative solutions are welcomed.

1.1 Part 1: Linear Regression

1.1.1 Problem 1

In this problem, you are expected to build a model to predict the Boston housing price.

[142]:
```
# # Please remove # and run the following code if you have an error while importing the dataset
# !pip install --upgrade openpyxl
```

[143]:
```
from sklearn import datasets
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
np.set_printoptions(suppress=True)
import sklearn
```

[144]:
```
print(sklearn.datasets.load_boston().DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.
```

```
    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people
by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
…', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function load_boston is deprecated; `load_boston` is deprecated
in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

[145]:

```
#Load boston housing dataset
boston_housing = datasets.load_boston()
X = pd.DataFrame(boston_housing['data'], columns =␣
 ↪boston_housing['feature_names'])

#"target" is the response variable
# which represents the median value of owner-occupied homes in $1000
```

```python
y = boston_housing['target']

#X.head()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function load_boston is deprecated; `load_boston` is deprecated
in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

[146]:
```python
data = X
data['target'] = y
```

[147]:
```python
data.head()
```

[147]:
```
       CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  target
0     15.3  396.90   4.98    24.0
1     17.8  396.90   9.14    21.6
2     17.8  392.83   4.03    34.7
3     18.7  394.63   2.94    33.4
4     18.7  396.90   5.33    36.2
```

TODO 1

Prevent collinearity by removing linearly dependent variables.

For example, if 2 variables A and B have a correlation coefficient larger than 0.9, eliminate one to avoid redundancy.

[148]:
```python
corr_mat = X.corr().abs()       # Creating a heatmap to find the correlation␣
 ↪between the given vlaues based pn color and value.
plt.figure(figsize=(25,10))
ax=sns.heatmap(X.corr(),square=True,annot=True,vmax=1,vmin=-1)
upper_tri = corr_mat.where(np.triu(np.ones(corr_mat.shape), k=1).astype(np.
 ↪bool))
red_col = [column for column in upper_tri.columns if any(upper_tri[column] > 0.
 ↪90)]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4:
DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To
silence this warning, use `bool` by itself. Doing this will not modify any
behavior and is safe. If you specifically wanted the numpy scalar type, use
`np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  after removing the cwd from sys.path.
```

[149]:
```
X.drop(red_col, axis=1, inplace=True)    # Removing the TAX column from the heatmap
X.head()
```

[149]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	PTRATIO	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	15.3	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	17.8	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	17.8	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	18.7	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	18.7	

	B	LSTAT	target
0	396.90	4.98	24.0
1	396.90	9.14	21.6
2	392.83	4.03	34.7
3	394.63	2.94	33.4
4	396.90	5.33	36.2

As seen the correlation between RAD and TAX is 0.91 which exceeded the 0.9 cutoff that was set and hence the TAX column was eliminated in order to avoid redundancy.

TODO 2

Partition the data into 75% training and 25% validation set.

```
[150]: from sklearn.model_selection import train_test_split
       X_train,X_val,y_train,y_val=train_test_split((X.iloc[:,0:12]),y,train_size=0.
        ↪75,random_state=21)  # Splitting the data into 75% training and 25%␣
        ↪validation data
       print(X_train.shape,y_train.shape,X_val.shape,y_val.shape)
```

```
(379, 12) (379,) (127, 12) (127,)
```

```
[151]: print(X.iloc[:,0:12])
```

```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  PTRATIO  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0     15.3
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0     17.8
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0     17.8
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0     18.7
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0     18.7
..       …     …      …     …      …      …     …     …    …        …
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0     21.0
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0     21.0
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0     21.0
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0     21.0
505  0.04741   0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0     21.0

          B  LSTAT
0    396.90   4.98
1    396.90   9.14
2    392.83   4.03
3    394.63   2.94
4    396.90   5.33
..      …      …
501  391.99   9.67
502  396.90   9.08
503  396.90   5.64
504  393.45   6.48
505  396.90   7.88

[506 rows x 12 columns]
```

TODO 3

If we fit a linear regression model on the training set, what will be the feature weights?

Calculate the feature weights using the matrix form (do not use any built-in packages such as sklearn or stat models).

[152]:
```
t1 = np.ones((len(X_train), 1))
t2 = np.ones((len(X_val), 1))
```

[153]:
```
X_p = np.append(t1, np.array(X_train), axis=1)
X_p.shape
```

[153]:
```
(379, 13)
```

[154]:
```
y_p = np.array(y_train).reshape((len(y_train),1))
y_p
```

[154]:
```
array([[27.9],
       [18.9],
       [50. ],
       [23.3],
       [33.2],
       [35.2],
       [12.7],
       [22.8],
       [22.6],
       [28.7],
       [18. ],
       [16.7],
       [23.1],
       [22.5],
       [19.5],
       [ 7.2],
       [50. ],
       [44.8],
       [37.2],
       [19. ],
       [46. ],
       [16.6],
       [24.6],
       [14. ],
       [15.6],
       [22.3],
       [21.4],
       [15.4],
       [13.1],
       [18.2],
       [23.9],
       [10.5],
       [30.1],
       [25.3],
       [18.1],
       [25. ],
```

```
[23.1],
[26.6],
[18.5],
[22.9],
[23.1],
[23.2],
[23.6],
[19.5],
[33.1],
[22.9],
[19.2],
[20.8],
[30.7],
[14.1],
[15.6],
[15.3],
[50. ],
[ 8.5],
[50. ],
[25. ],
[50. ],
[11.7],
[18.3],
[15.6],
[19.8],
[37.9],
[19.7],
[23.3],
[15.6],
[36. ],
[19.9],
[13.1],
[11.9],
[22.2],
[32. ],
[18.2],
[17.8],
[22.4],
[21.8],
[20.8],
[19.6],
[23.8],
[28.4],
[21. ],
[29.6],
[20.6],
[14.5],
```

```
[21.2],
[34.9],
[23.4],
[18.6],
[20. ],
[16.7],
[30.1],
[38.7],
[22.5],
[20. ],
[24.8],
[20.8],
[12.7],
[13.3],
[32.7],
[34.6],
[23.1],
[13.8],
[29.4],
[17.2],
[13.1],
[10.2],
[21.4],
[18.7],
[20.6],
[23. ],
[18.8],
[25. ],
[10.2],
[35.1],
[23.8],
[29.6],
[29. ],
[21.9],
[21.7],
[20.5],
[14.6],
[13.8],
[31.7],
[44. ],
[ 8.3],
[19.4],
[23.9],
[14.3],
[19.9],
[13.6],
[25. ],
```

```
[16.4],
[12.3],
[21.9],
[13.8],
[ 7.4],
[26.6],
[48.5],
[26.5],
[22.6],
[24.7],
[18.9],
[22.2],
[23.4],
[19.2],
[19.3],
[19.9],
[24.1],
[19.6],
[23.7],
[24.8],
[22.7],
[17.5],
[36.4],
[21.5],
[14.5],
[14.2],
[13.3],
[50. ],
[22.8],
[27.1],
[21.2],
[23.7],
[22.9],
[32.2],
[21.4],
[ 7.2],
[34.7],
[20.3],
[20.7],
[24.4],
[13.4],
[19. ],
[22.8],
[30.3],
[19.5],
[31.2],
[17.8],
```

```
[23.8],
[14.1],
[29.1],
[18.7],
[27.5],
[28.4],
[25. ],
[24.5],
[16.1],
[14.9],
[17. ],
[19.6],
[12.8],
[21.1],
[18.5],
[28.7],
[21.8],
[11.5],
[22.6],
[21.7],
[37.6],
[19.5],
[19.1],
[29. ],
[19.6],
[24.1],
[16.8],
[21.1],
[14.3],
[31. ],
[30.1],
[ 8.1],
[ 9.6],
[10.8],
[20. ],
[25. ],
[20.7],
[10.9],
[24.3],
[13.3],
[24. ],
[26.6],
[13.5],
[ 5. ],
[16.1],
[17.4],
[22.6],
```

```
[15. ],
[17.5],
[21.2],
[31.5],
[34.9],
[20.2],
[ 7. ],
[50. ],
[20.3],
[31.1],
[24.5],
[24.7],
[20.6],
[33.4],
[20. ],
[22. ],
[23.9],
[21.4],
[23.2],
[32.5],
[27.9],
[23.2],
[22.6],
[12. ],
[50. ],
[11.7],
[19.4],
[ 9.7],
[ 9.5],
[23.5],
[12.7],
[22. ],
[18.5],
[17.5],
[21.7],
[19.8],
[16.5],
[29.1],
[23.2],
[13.2],
[21.2],
[50. ],
[14.8],
[12.5],
[28.2],
[21.6],
[31.6],
```

```
[11.8],
[50. ],
[21.6],
[ 5. ],
[20.1],
[50. ],
[23. ],
[18.6],
[21.9],
[18.5],
[28.6],
[20.2],
[19.1],
[28.7],
[33.1],
[24.4],
[26.7],
[19.4],
[33. ],
[20.4],
[15.7],
[35.4],
[45.4],
[19.3],
[23.3],
[16.5],
[13.1],
[21.7],
[20.3],
[20.4],
[36.2],
[22. ],
[ 8.4],
[25. ],
[18.8],
[27. ],
[20.6],
[21.5],
[24.4],
[ 7.5],
[27.5],
[24.3],
[11.8],
[20.1],
[19.4],
[10.2],
[20.4],
```

```
[17.1],
[24. ],
[24.6],
[29.8],
[ 8.8],
[27.5],
[17.1],
[18.4],
[17.2],
[15.1],
[25.2],
[20.6],
[22. ],
[43.5],
[10.4],
[24.5],
[16.8],
[37.3],
[28. ],
[43.1],
[20.3],
[22.4],
[25.1],
[10.5],
[19.7],
[14.9],
[32.4],
[31.5],
[18.4],
[22.3],
[26.4],
[50. ],
[15. ],
[17.8],
[16.2],
[23.9],
[17.8],
[23.7],
[36.1],
[20.5],
[24.3],
[15.2],
[48.3],
[42.8],
[19.8],
[11. ],
[12.6],
```

```
        [21.7],
        [19.3],
        [42.3],
        [20.1],
        [43.8],
        [29.8],
        [ 8.4],
        [22. ],
        [50. ],
        [14.4],
        [33.8],
        [19.4],
        [22.5],
        [13.5]])
```

[155]:
```
Beta = np.dot((np.linalg.inv(np.dot(X_p.T, X_p))), np.dot(X_p.T,y_p))
Beta
```

[155]:
```
array([[ 39.79755023],
       [ -0.10213657],
       [  0.03956367],
       [ -0.06830324],
       [  3.22419083],
       [-20.04876015],
       [  3.38218024],
       [  0.00310189],
       [ -1.51000183],
       [  0.16685191],
       [ -1.08614041],
       [  0.01138894],
       [ -0.54703226]])
```

TODO 4

Now only consider two input variables: Age and RM.

Fit a linear regression model on the training set with a package at your choice.

Present the model summary. We call this model **Model 1**.

[156]:
```
import sklearn.metrics as metrics
X1=X[['AGE','RM']]                                    #
X1_val=X_val[['AGE','RM']]
from sklearn.linear_model import LinearRegression
m1 = LinearRegression().fit(X1, y)
y_pred1 = m1.predict(X1_val)
print('Coeffecient: ',m1.coef_)
pd.DataFrame({'y-validation':y_val,'y_prediction':y_pred1})
```

```
Coeffecient:  [-0.07277679  8.40158122]
```

[156]:

```
     y-validation  y_prediction
0            14.1     23.247726
1            13.4     12.838666
2            22.1     22.314467
3            41.7     39.432045
4            28.5     30.335377
..            …           …
122          19.6     21.083709
123          20.9     21.764273
124          16.3     10.755074
125          17.3     16.862912
126          20.1     18.912400

[127 rows x 2 columns]
```

TODO 5

Evaluate the prediction performance of Model 1 on the validation set with RMSE and MAE as performance matrics.

[157]:

```
def regression_results(y_val, y_pred):
  # Regression metrics
  explained_variance=metrics.explained_variance_score(y_val, y_pred)
  mean_absolute_error=metrics.mean_absolute_error(y_val, y_pred)
  mse=metrics.mean_squared_error(y_val, y_pred)
  median_absolute_error=metrics.median_absolute_error(y_val, y_pred)
  r2=metrics.r2_score(y_val, y_pred)
  print('explained_variance: ', round(explained_variance,4))
  print('R-square: ', round(r2,4))
  print('MAE: ', round(mean_absolute_error,4))
  print('RMSE: ', round(np.sqrt(mse),4))
regression_results(y_val,y_pred1)
```

```
explained_variance:  0.5963
R-square:  0.5894
MAE:  3.8606
RMSE:  5.9024
```

TODO 6

Now consider all the features (after removing linearly dependent variables).

Fit a linear regression model on the training set with a package at your choice. Present the model summary.

We call this model **Model 2**.

[158]:

```
from sklearn.linear_model import LinearRegression
#X1=X[['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','PTRATIO','B','LSTAT']]
#X_val=X_val[['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','PTRATIO','B',␣
↪'LSTAT']]
```

```
m2 = LinearRegression().fit(X_train, y_train)                                    ␣
 ↪                            # Fitting the model according to all the features␣
 ↪on the training dataset
y_pred2 = m2.predict(X_val)
print('Coeffecients: ',m2.coef_)
print(pd.DataFrame({'y-val':y_val,'y_pred':y_pred2}))
```

```
Coeffecients:  [ -0.10213657   0.03956367  -0.06830324   3.22419083 -20.04876015
   3.38218024   0.00310189  -1.51000183   0.16685191  -1.08614041
   0.01138894  -0.54703226]
     y-val     y_pred
0     14.1  15.166721
1     13.4  15.188045
2     22.1  26.948165
3     41.7  37.259115
4     28.5  32.458547
..     …         …
122   19.6  20.483352
123   20.9  21.710176
124   16.3  12.493405
125   17.3  13.458538
126   20.1  19.043769

[127 rows x 2 columns]
```

TODO 7

Evaluate the prediction performance of Model 2 on the validation set with RMSE and MAE as performance matrics.

[159]:
```
def regression_results(y_val, y_pred):
 # Regression metrics
 explained_variance=metrics.explained_variance_score(y_val, y_pred)
 mean_absolute_error=metrics.mean_absolute_error(y_val, y_pred)
 mse=metrics.mean_squared_error(y_val, y_pred)
 median_absolute_error=metrics.median_absolute_error(y_val, y_pred)
 r2=metrics.r2_score(y_val, y_pred)
 print('explained_variance: ', round(explained_variance,4))
 print('R-square: ', round(r2,4))
 print('MAE: ', round(mean_absolute_error,4))
 print('MSE: ', round(mse,4))
 print('RMSE: ', round(np.sqrt(mse),4))
regression_results(y_val,y_pred2)
```

```
explained_variance:  0.7167
R-square:  0.7143
MAE:  3.5073
MSE:  24.2409
RMSE:  4.9235
```

The R-Squared values for this model is higher than for the previous model, showing a more positive correlation The MAE value is low the model has good performance The RMSE value is pretty low, so again, less variance in distribution of predicted values compared to actual values

TODO 8

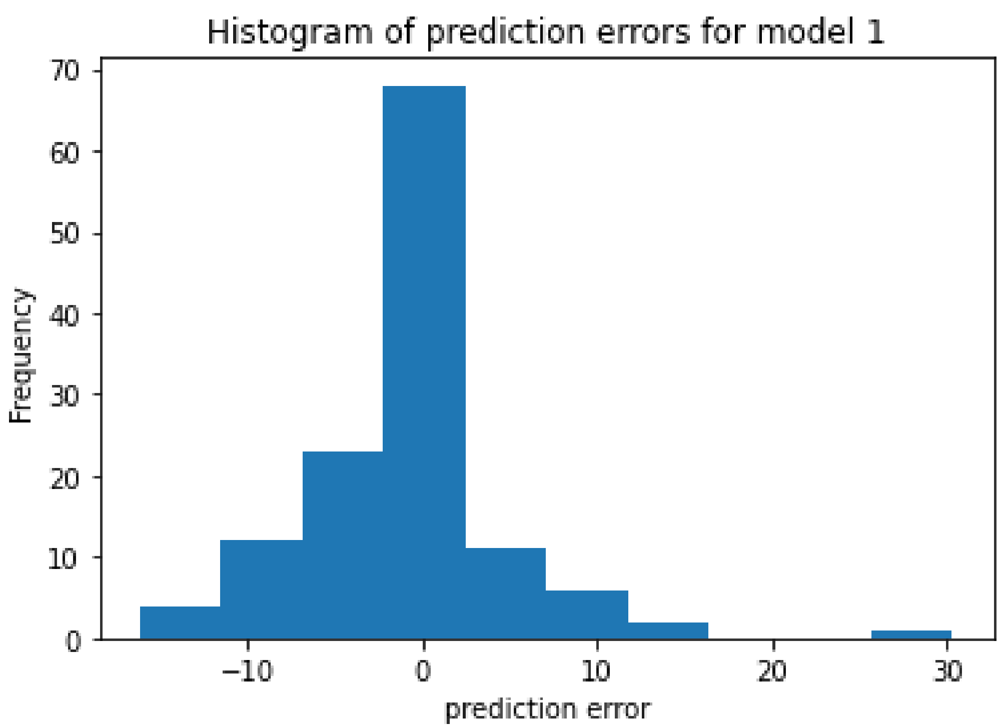
Compare the performance of Model 1 and Model 2.

Visualize the prediction error of both models using histogram.

Comment on the model fitting.

[160]:
```python
#Model-1
diff = y_val - y_pred1
plt.hist(diff)
plt.title('Histogram of prediction errors for model 1')
plt.xlabel('prediction error')
plt.ylabel('Frequency')
```

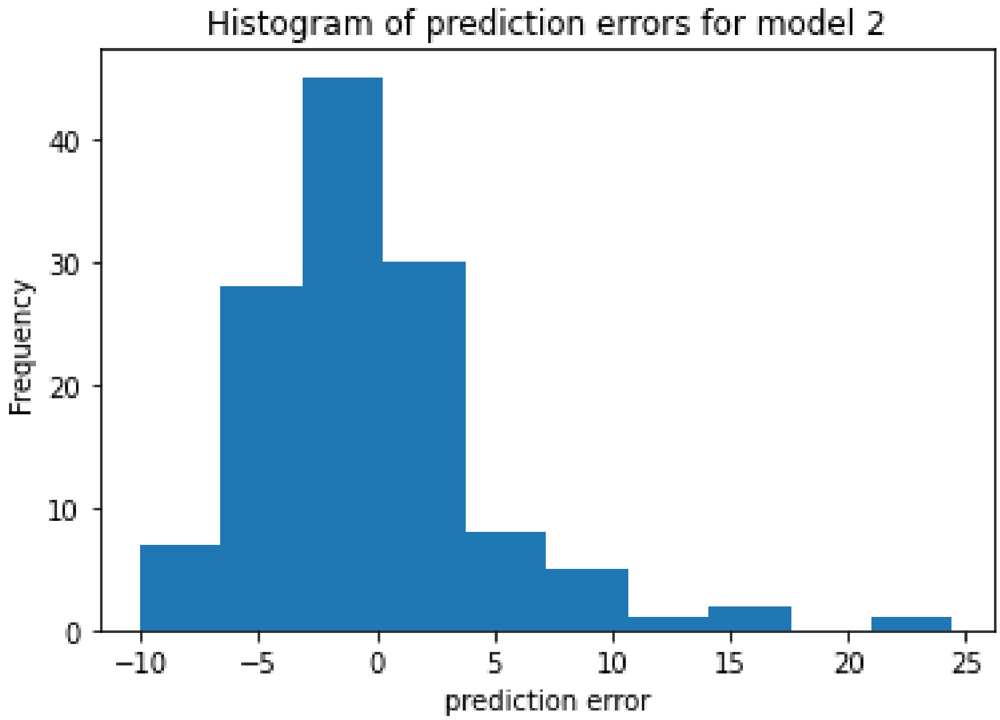
[160]: Text(0, 0.5, 'Frequency')



[161]:
```python
#Model-2
diff = y_val - y_pred2
plt.hist(diff)
plt.title('Histogram of prediction errors for model 2')
plt.xlabel('prediction error')
```

```
plt.ylabel('Frequency')
```

[161]: Text(0, 0.5, 'Frequency')



We can see the predicted and actual value difference in the histograms above with the error on the x axis and frequency on the y.

TODO 9

Now consider all the features (after removing linearly dependent variables).

The goal is to fit a LASSO linear regression model on the training set with a package at your choice.

Compare the model performance of lambda in the range of [0,1] with the step of 0.01.

Plot RMSE versus log(lambda).

Pick the appropriate lambda value according to the plot.

Present the model summary with the selected lambda. We call this model **Model 3**.

[162]:
```
from sklearn.linear_model import Lasso
m3 = Lasso(alpha=.0001,normalize=True, max_iter=1e5)
m3.fit(X_train,y_train)
y_pred2 = m3.predict(X_val)
print(pd.DataFrame({'y-val':y_val,'y_pred':y_pred2}))
```

```
     y-val     y_pred
0     14.1  15.170698
1     13.4  15.203462
2     22.1  26.950354
3     41.7  37.256258
4     28.5  32.447802
..     …         …
122   19.6  20.485979
123   20.9  21.717167
124   16.3  12.492869
125   17.3  13.466572
126   20.1  19.057832

[127 rows x 2 columns]
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_base.py:145:
FutureWarning: 'normalize' was deprecated in version 1.0 and will be removed in
1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Lasso())

If you wish to pass a sample_weight parameter, you need to pass it as a fit
parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
  FutureWarning,
```

TODO 10

Evaluate the prediction performance of Model 3 on the validation set with RMSE and MAE as performance matrics.

[163]:
```
explained_variance=metrics.explained_variance_score(y_val, y_pred2)
mean_absolute_error=metrics.mean_absolute_error(y_val, y_pred2)
mse=metrics.mean_squared_error(y_val, y_pred2)
median_absolute_error=metrics.median_absolute_error(y_val, y_pred2)
r2=metrics.r2_score(y_val, y_pred2)
print('explained_variance: ', round(explained_variance,4))
print('R-square: ', round(r2,4))
print('MAE: ', round(mean_absolute_error,4))
print('MSE: ', round(mse,4))
print('RMSE: ', round(np.sqrt(mse),4))
```

```
print('Coeffecients: ',m3.coef_)
```

```
explained_variance:  0.7169
R-square:  0.7145
MAE:  3.5047
MSE:  24.2292
RMSE:  4.9223
Coeffecients:  [ -0.10109983   0.0393375   -0.06761281   3.22022508 -19.9382264
   3.38658887   0.00282729  -1.50483226   0.16535568  -1.08407638
   0.01137203  -0.54684879]
```

TODO 11

Among Model 1, 2, and 3, which one would be your pick for future implementation? State your reasons.

Among models 1,2 and 3 the best model to choose would be model 3 as we can see that the RMSE and MAE values for model 3 is the least compared to all the three models and thereby having a higher accuracy

1.2 Part 2: K-Nearst-Neighbors

1.2.1 Problem 2

The wine dataset is the result of a chemical analysis of wines produced in the same region in Italy but derived from three different cultivars. (For illustration simplicity purpose, only 2 classes, 0 and 1, will be included for the classification task.) The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The objective is to classify the wines into class 0 or 1 using the 13 given attributes and k-NN classifier.

[15]:
```
# load the wine dataset
wine = datasets.load_wine()
print(wine.DESCR)
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
                - Alcohol
                - Malic acid
                - Ash
                - Alcalinity of ash
                - Magnesium
```

```
            - Total phenols
            - Flavanoids
            - Nonflavanoid phenols
            - Proanthocyanins
            - Color intensity
            - Hue
            - OD280/OD315 of diluted wines
            - Proline

    - class:
            - class_0
            - class_1
            - class_2

    :Summary Statistics:

    ============================= ==== ===== ======= =====
                                   Min   Max   Mean     SD
    ============================= ==== ===== ======= =====
    Alcohol:                      11.0  14.8    13.0   0.8
    Malic Acid:                   0.74  5.80    2.34  1.12
    Ash:                          1.36  3.23    2.36  0.27
    Alcalinity of Ash:            10.6  30.0    19.5   3.3
    Magnesium:                    70.0 162.0    99.7  14.3
    Total Phenols:                0.98  3.88    2.29  0.63
    Flavanoids:                   0.34  5.08    2.03  1.00
    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
    Proanthocyanins:              0.41  3.58    1.59  0.57
    Colour Intensity:              1.3  13.0     5.1   2.3
    Hue:                          0.48  1.71    0.96  0.23
    OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
    Proline:                       278  1680     746   315
    ============================= ==== ===== ======= =====

    :Missing Attribute Values: None
    :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen different
measurements taken for different constituents found in the three types of
wine.
```

```
Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
  has achieved 100% correct classification.
  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
  (All results using the leave-one-out technique)

  (2) S. Aeberhard, D. Coomans and O. de Vel,
  "THE CLASSIFICATION PERFORMANCE OF RDA"
  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Journal of Chemometrics).
```

[164]:
```python
# convert the data into dataframe format
X = pd.DataFrame(wine['data'], columns = wine['feature_names'])
y = wine['target']

# only consider wine class 0 and 1
X = X.loc[0:129, :]
y = y[0:130]

X.head()
```

[164]:
```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
```

```
2    13.16           2.36  2.67                 18.6       101.0            2.80
3    14.37           1.95  2.50                 16.8       113.0            3.85
4    13.24           2.59  2.87                 21.0       118.0            2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline
0                          3.92   1065.0
1                          3.40   1050.0
2                          3.17   1185.0
3                          3.45   1480.0
4                          2.93    735.0
```

[165]:
```python
data = X
data['target'] = y
```

TODO 1

Considering the fundamental idea of k-NN, would you recommend data rescaling before model building? Why?

If so, partition the data into 75% training and 25% validation set, then standardize them.

The basic concept of kNN is to determine the k nearest neighbours,and one of the ways this is possible is by using the euclidean distance.The distance calculation would increase in accuracy if the variables were normalized to the same scale and hence the data needs to be rescaled.

[166]:
```python
from sklearn.model_selection import train_test_split     # Partitioning or␣
 ↪splitting the data into 75% training data and 25% remaining is the␣
 ↪validation
X_train,X_val,y_train,y_val=train_test_split(data,y,train_size=0.75)
X_train_std=(X_train-X_train.mean())/X_train.std()
X_val_std=(X_val-X_val.mean())/X_val.std()
```

TODO 2

Choose the best k from 1-10 based on the classification accuracy of different k values on the validation set.

[23]:
```python
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
results = []
for k in range(1, 11):                          # To train the dataset for␣
 ↪different values of k
```

```
    #k=k+1
    knn = KNeighborsClassifier(n_neighbors=k).fit(X_train_std, y_train)
    results.append({
    'k': k,
    'accuracy': accuracy_score(y_val, knn.predict(X_val_std))
})
# Convert results to a pandas data frame
results = pd.DataFrame(results)
print(results)
```

```
    k  accuracy
0   1  0.969697
1   2  0.939394
2   3  0.969697
3   4  0.939394
4   5  0.969697
5   6  0.969697
6   7  0.969697
7   8  0.969697
8   9  0.969697
9  10  0.969697
```

The best value of k would be 5 as the accuracy of 0.96967 is the the most consistent value and remains constant for most values of k post k=5

TODO 3

Classify the new record given below using the chosen k.

Considering the size of the wine dataset, would you recommend data partition before scoring the new record? Why?

[159]:
```
# New record
new_wine = pd.DataFrame(columns = wine['feature_names'])
new_wine.loc[0,:] = np.array([14.12, 1.88, 2.31, 18.5, 125, 2.50, 3.12, 0.26, 2.
↪12, 4.87, 1.02, 3.23, 955])
new_wine
```

[159]:
```
  alcohol malic_acid   ash alcalinity_of_ash magnesium total_phenols  \
0   14.12       1.88  2.31              18.5     125.0           2.5

  flavanoids nonflavanoid_phenols proanthocyanins color_intensity   hue  \
0       3.12                 0.26            2.12            4.87  1.02

  od280/od315_of_diluted_wines proline
0                         3.23   955.0
```

[160]:
```
#normal_df.shape[0]
normalized_record=(new_wine-X.mean())/X.std()   # Normalizing the dataset
normalized_record
```

[160]:
```
  alcalinity_of_ash   alcohol       ash color_intensity flavanoids       hue  \
0         -0.083925  1.323091 -0.102508         0.415327    0.85393 -0.230555

  magnesium malic_acid nonflavanoid_phenols od280/od315_of_diluted_wines  \
0  1.631843  -0.100091            -0.643806                      0.58004

  proanthocyanins   proline  target total_phenols
0         0.68131  0.467806     NaN     -0.041607
```

The new data needs to be split post normalization to determine the optimal k value .The new data needs to be on the same scale as the training data before scoring it ,hence a split is reccomended.

1.2.2 Problem 3

The data concerns city-cycle fuel consumption in miles per gallon (mpg). The objective is to use k-NN classifier to predict the mpg with the given attributes.

[24]:
```python
# import the dataset "auto_mpg.csv"
from google.colab import files
file = files.upload()  #upload file into google colab session
df = pd.read_csv("auto_mpg.csv")
```

```
<IPython.core.display.HTML object>
```

```
Saving auto_mpg.csv to auto_mpg.csv
```

[25]:
```python
raw_dataset=df.iloc[1:,:]
car_df = raw_dataset.copy()
car_df.head()
```

[25]:
```
    mpg  cylinders  displacement horsepower  weight  acceleration  model year  \
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70
3  16.0          8         304.0        150    3433          12.0          70
4  17.0          8         302.0        140    3449          10.5          70
5  15.0          8         429.0        198    4341          10.0          70

   origin              car name
1       1   buick skylark 320
2       1  plymouth satellite
3       1       amc rebel sst
4       1         ford torino
5       1    ford galaxie 500
```

TODO 1

Check the unique value of the variable "car name".

Would you recommend keeping "car name" for prediction? Why?

If not, eliminate the variable "car name".

[26]:
```
car_df = car_df.rename(columns = {"car name":"car_name"})
```

[27]:
```
print(car_df.car_name.value_counts())
car_df=car_df.iloc[:,:-1]
```

```
ford pinto             5
amc matador            5
toyota corolla         5
ford maverick          5
peugeot 504            4
                      ..
buick skyhawk          1
chevrolet monza 2+2    1
ford mustang ii        1
pontiac astro          1
chevy s-10             1
Name: car_name, Length: 301, dtype: int64
```

The car name is not required for classification for mpg as its not categorical in nature and will have very less correlation with the target variable.

[29]:
```
car_df.head()
```

[29]:
```
    mpg  cylinders  displacement horsepower  weight  acceleration  model year  \
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70
3  16.0          8         304.0        150    3433          12.0          70
4  17.0          8         302.0        140    3449          10.5          70
5  15.0          8         429.0        198    4341          10.0          70

   origin
1       1
2       1
3       1
4       1
5       1
```

TODO 2

Convert the variable "origin" to dummy variables before modeling

[30]:
```
car_df = pd.concat([car_df, pd.get_dummies(car_df['origin'])], axis = 1);
car_df.head()
```

[30]:
```
    mpg  cylinders  displacement horsepower  weight  acceleration  model year  \
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70
3  16.0          8         304.0        150    3433          12.0          70
4  17.0          8         302.0        140    3449          10.5          70
```

```
5  15.0          8         429.0         198     4341           10.0           70

   origin  1  2  3
1       1  1  0  0
2       1  1  0  0
3       1  1  0  0
4       1  1  0  0
5       1  1  0  0
```

[36]:
```
car_df=car_df.replace('?',np.nan)
car_df['horsepower'].fillna(method='ffill',inplace=True)
car_df.isnull().sum()
```

[36]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
1               0
2               0
3               0
dtype: int64
```

[33]:
```
car_df.dtypes
```

[33]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
model year        int64
origin            int64
1                 uint8
2                 uint8
3                 uint8
dtype: object
```

TODO 3

Rescale the numeric data. Note that dummy variables should not be rescaled.

[40]:
```
car_df['cylinders'] = car_df['cylinders'].astype(str).astype(int)
car_df["displacement"] = car_df["displacement"].astype(str).astype(float)
car_df["horsepower"] = car_df["horsepower"].astype(str).astype(float)
```

```
car_df["weight"] = car_df["weight"].astype(str).astype(int)
car_df["acceleration"] = car_df["acceleration"].astype(str).astype(float)
car_df["model year"] = car_df["model year"].astype(str).astype(int)
car_df["mpg"]= car_df["mpg"].astype(str).astype(float)
n_data1=(car_df.iloc[:,0:6]-car_df.iloc[:,0:6].mean())/car_df.iloc[:,0:6].std()
d1=car_df.iloc[0:398,6:11]
frames=[n_data1,d1]
final_data=pd.concat(frames,axis=1)
final_data.head()
```

[40]:

```
        mpg  cylinders  displacement  horsepower    weight  acceleration  \
1 -1.083635   1.489056      1.491649    1.575673  0.844547     -1.472068
2 -0.699075   1.489056      1.185397    1.185783  0.541838     -1.653622
3 -0.955448   1.489056      1.051412    1.185783  0.538305     -1.290515
4 -0.827262   1.489056      1.032271    0.925856  0.557150     -1.835176
5 -1.083635   1.489056      2.247710    2.433432  1.607799     -2.016730

   model year  origin  1  2  3
1          70       1  1  0  0
2          70       1  1  0  0
3          70       1  1  0  0
4          70       1  1  0  0
5          70       1  1  0  0
```

TODO 4

Partition the data into 75% training and 25% validation set.

[122]:

```
y=final_data['mpg']
```

[123]:

```
from sklearn.model_selection import train_test_split
X_train,X_val,y_train,y_val=train_test_split(final_data,y,random_state=42,train_size=0.
↪75)
```

[47]:

```
y_train
```

[47]:

```
266    0.518698
17    -0.314515
67    -1.340008
159   -0.827262
8     -1.211822
         …
72    -1.340008
107   -0.442702
271    0.044408
349    0.826346
103   -1.468195
Name: mpg, Length: 294, dtype: float64
```

TODO 5

Choose the best k from 1-10 based on the MSE of different k values on the validation set. Explain the reason for your choice.

[48]:
```
from sklearn.neighbors import NearestNeighbors, KNeighborsRegressor
from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics
# Train a classifier for different values of k
results = []
for k in range(1, 10):
 knn = KNeighborsRegressor(n_neighbors=k).fit(X_train, y_train)
 results.append({
 'k': k,
 'MSE': metrics.mean_squared_error(y_val, knn.predict(X_val))
})
# Convert results to a pandas data frame
results = pd.DataFrame(results)
print(results)
#Best MSE value for k=7
```

```
   k       MSE
0  1  0.094211
1  2  0.074346
2  3  0.072085
3  4  0.064226
4  5  0.065178
5  6  0.063329
6  7  0.061442
7  8  0.062706
8  9  0.062483
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
```

```
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
```

```
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
```

The best MSE value is for k=7 which means that the accuracy is highest for that vale and hence its the best choice.

TODO 6

Score the validation set with the best k. Comment on the model performance.

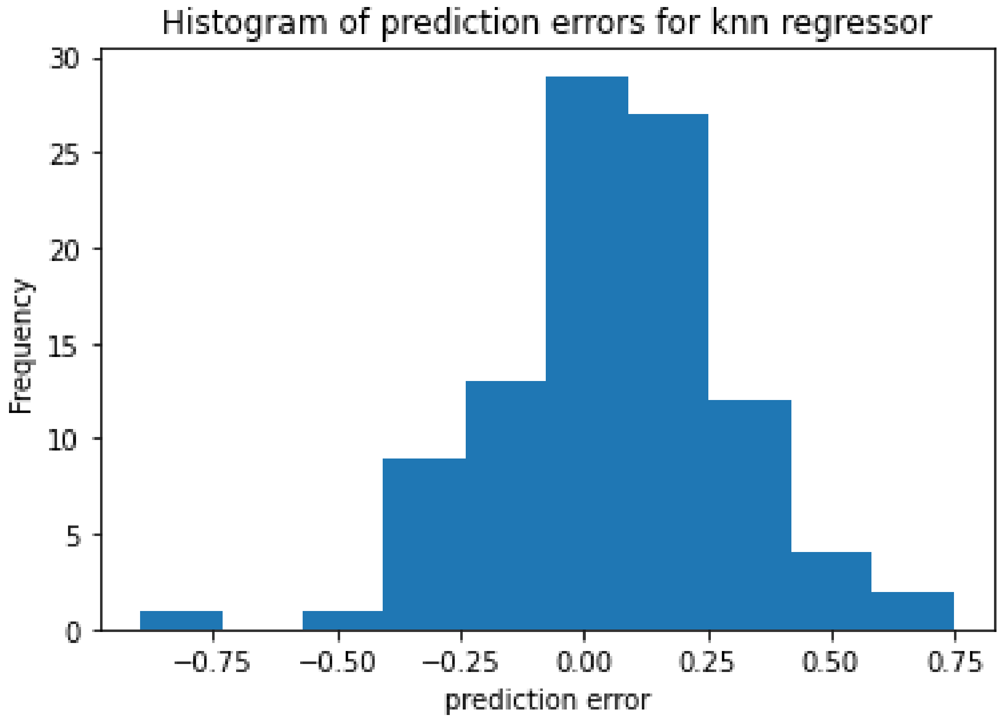
[50]:
```
knn = KNeighborsRegressor(n_neighbors=7).fit(X_train, y_train)
y_pred3=knn.predict(X_val)
print(pd.DataFrame({'y-val':y_val,'y_pred':y_pred3,'Residual':(y_val-y_pred3)}))
plt.hist(y_pred3-y_val)
plt.title('Histogram of prediction errors for knn regressor')
plt.xlabel('prediction error')
plt.ylabel('Frequency')
```

```
        y-val    y_pred  Residual
79  -0.186328  0.015108 -0.201436
275 -0.237603 -0.065467 -0.172136
247  1.621103  1.253025  0.368079
56   0.070045 -0.003205  0.073250
388  0.454605  0.414318  0.040287
..        …         …         …
366  0.967351  0.784228  0.183124
251 -0.545251 -0.440871 -0.104381
210 -0.891355 -0.543420 -0.347935
76  -0.186328  0.106670 -0.292998
105 -1.468195 -1.376633 -0.091562

[98 rows x 3 columns]
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692:
FutureWarning: Feature names only support names that are all strings. Got
feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
```

[50]:
```
Text(0, 0.5, 'Frequency')
```



We can see that the MSE for k=7 is low and hence it has a good performance