Boston Housing Dataset

Predicting Median value of owner-occupied homes

The aim of this assignment is to learn the application of machine learning algorithms to data sets. This involves learning what data means, how to handle data, training, cross validation, prediction, testing your model, etc. This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass. It was obtained from the StatLib archive, and has been used extensively throughout the literature to benchmark algorithms. The data was originally published by Harrison, D. and Rubinfeld, D.L. Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. The dataset is small in size with only 506 cases. It can be used to predict the median value of a home, which is done here. There are 14 attributes in each case of the dataset. They are:

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

Aim

- To implement a linear regression with regularization via gradient descent.
- to implement gradient descent with Lp norm, for 3 different values of p in (1,2]
- To contrast the difference between performance of linear regression Lp norm and L2 norm for these 3 different values.
- Tally that the gradient descent for L2 gives same result as matrix inversion based solution.

All the code is written in a single python file. The python program accepts the data directory path as input where the train and test csv files reside. Note that the data directory will contain two files train.csv used to train your model and test.csv for which the output predictions are to be made. The output predictions get written to a file named output.csv. The output.csv file should have two comma separated columns [ID,Output].

Working of Code

- NumPy library would be required, so code begins by importing it
- Import phi and phi_test from train and test datasets using NumPy's loadtxt function
- Import y from train dataset using the loadtxt function
- Concatenate coloumn of 1s to right of phi and phi_test
- Apply min max scaling on each coloumn of phi and phi_test

- Apply log scaling on y
- Define a function to calculate change in error function based on phi, w and p norm
- Make a dictionary containing filenames as keys and p as values
- For each item in this dictionary
  - Set the w to all 0s
  - Set an appropriate value for lambda and step size
  - Calculate new value of w
  - Repeat steps until error between consecutive ws is less than threshold
  - Load values of id from test data file
  - Calculate y for test data using phi test and applying inverse log
  - Save the ids and y according to filename from dictionary

Feature Engineering

- Columns of phi are not in same range, this is because their units are different i.e phi is ill conditioned
- So, min max scaling for each column is applied to bring them in range 0-1
- Same scaling would be required on columns of phi test
- Log scaling was used on y. This was determined by trial and error

Comparison of performance

(p1=1.75, p2=1.5, p3=1.3)

- As p decreases error in y decreases
- As p decreases norm of w increases but this can be taken care by increasing lambda
- As p decreases number of iterations required decreases

Tuning of Hyperparameter

- If p is fixed and lambda is increased error decreases up to a certain lambda, then it starts rising
- So, lambda was tuned by trial and error.
- Starting with 0, lambda was increased in small steps until a minimum error was achieved.

Comparison of L2 gradient descent and closed form

- Error from L2 Gradient descent were 4.43268 and that from closed form solution was 4.52624.
- Errors are comparable so, the L2 gradient descent performs closely with closed form solution.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

In [3]:
```
from sklearn.datasets import load_boston
boston_dataset = load_boston()
```

```
C:\Users\Karthi\PycharmProjects\pythonProject\venv\lib\site-packages\sklearn\utils\deprecation.p
y: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)
```

In [4]:

```
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston.head()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [5]:

```
boston['MEDV'] = boston_dataset.target
```

In [6]:

```
boston.isnull().sum()
```

Out[6]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

Exploratory Data Analysis

In [7]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(boston['MEDV'], bins=30)
plt.show()
```

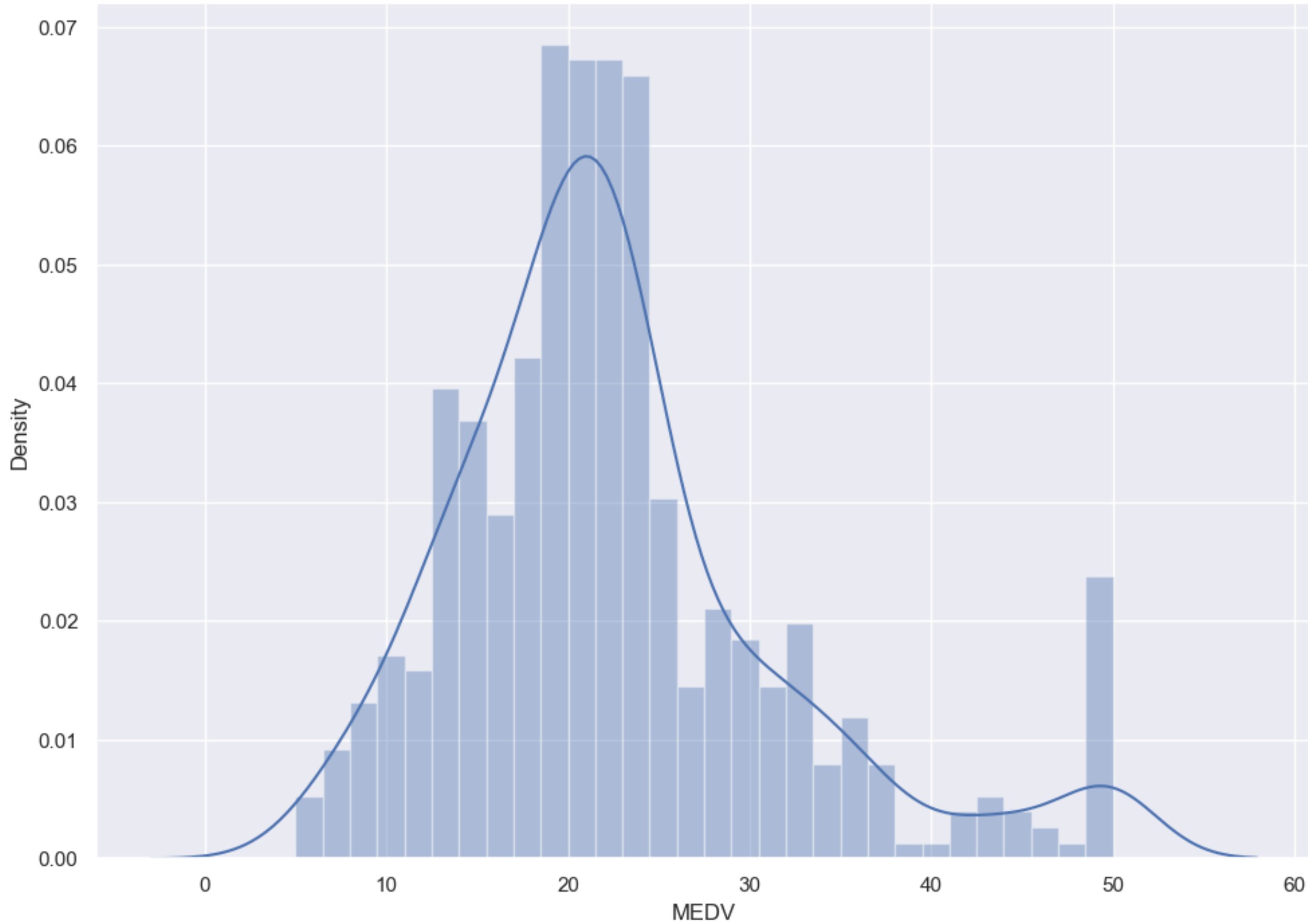
```
C:\Users\Karthi\AppData\Local\Temp\ipykernel_4500\1962179664.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(boston['MEDV'], bins=30)
```



In [8]:
```
correlation_matrix = boston.corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

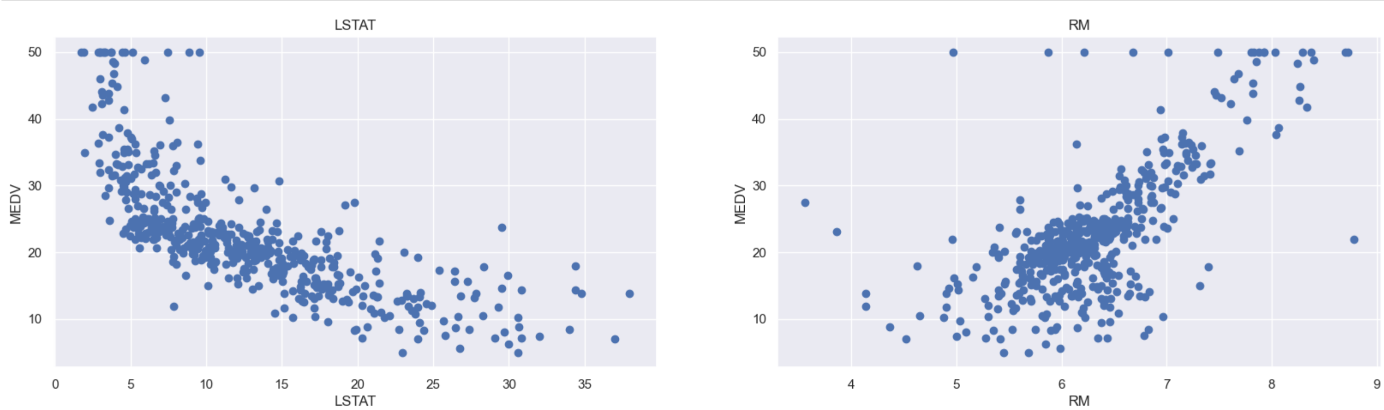
Out[8]: <AxesSubplot: >

In [9]:

```python
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = boston['MEDV']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = boston[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```



Preparing the data for training the model

In [10]:

```python
X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])
Y = boston['MEDV']
```

Splitting the data into training and testing sets

In [11]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 2)
(102, 2)
(404,)
(102,)
```

Training and testing the model

In [12]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

Out[12]:
```
▾ LinearRegression
LinearRegression()
```

Model evaluation

In [15]:
```
from sklearn.metrics import r2_score
```

In [16]:
```
# model evaluation for training set
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 5.637129335071195
R2 score is 0.6300745149331701


The model performance for testing set
--------------------------------------
RMSE is 5.137400784702911
R2 score is 0.6628996975186952
```