CS405 Machine Learning

Lab #4 Linear Regression

(100 points)

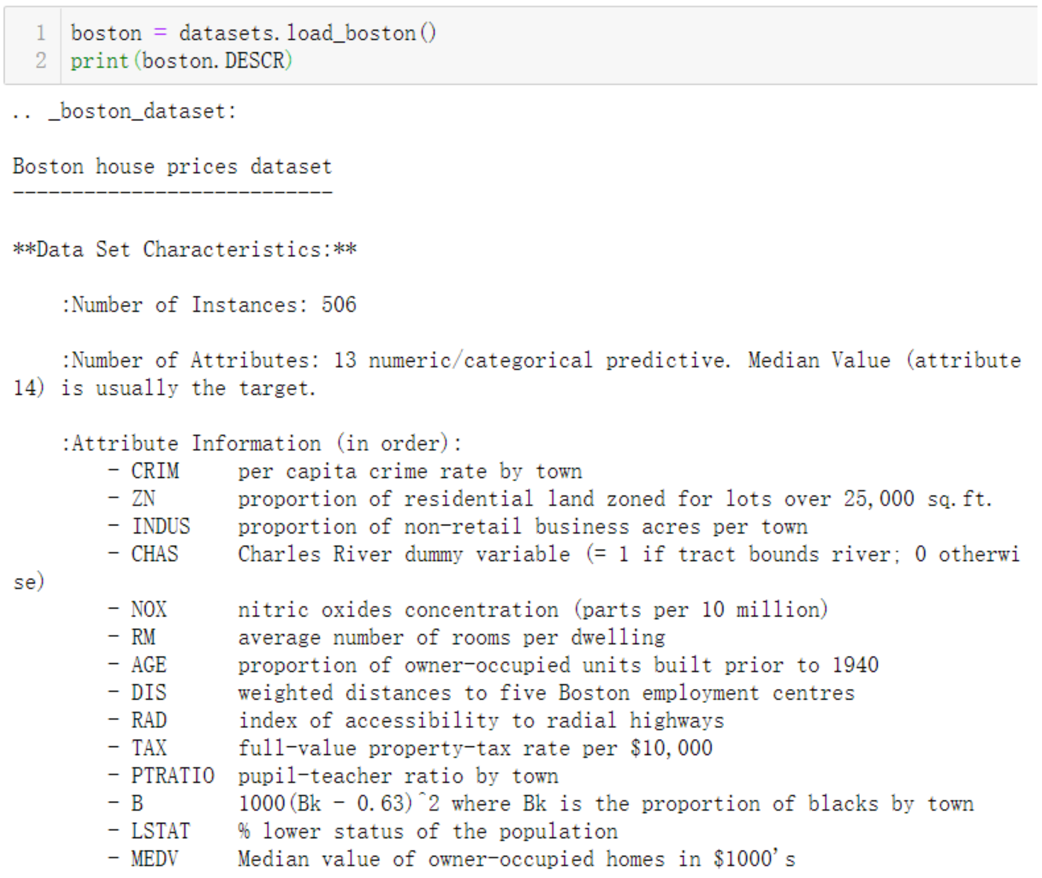
1. Introduction

You have learnt *linear regression* in lecture, now it is time to implement this machine learning technique in practice. In this lab, you will use linear regression to fit a house price model. You will leave some data as test set to evaluate your model. The scikit learn package of Python provides many modules for easily using machine learning algorithms.

2. Scikit learn package

Implement a linear regression by scratch is not hard, but there are many platform providing this algorithm. Here we take Python scikit learn as an example. This package contains many classical machine learning algorithms modules and easy to use.

Datasets: scikit learn provides some datasets which can be directly loaded by function. There are some small datasets called *toy datasets* and some large ones with thousands of samples called *real world datasets*. Firstly we load one as example.



```
boston = datasets.load_boston()
print(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 otherwi
se)
        - 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 blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

Fig. 1 toy dataset boston house price

See https://scikit-learn.org/stable/datasets/index.html for details. To do this you have to import right packages and modules. This is a small dataset containing 506 samples and 13 attributes. Use visualization method to have an intuitive understanding. We choose the sixth attribute and draw a scattering plot to see the distribution of each sample. This step uses *matplotlib*.

```
# Use one feature for visualization
x = boston.data[:, 5]

# Get the target vector
y = boston.target
```

```
plt.scatter(x, y)
plt.show()
```

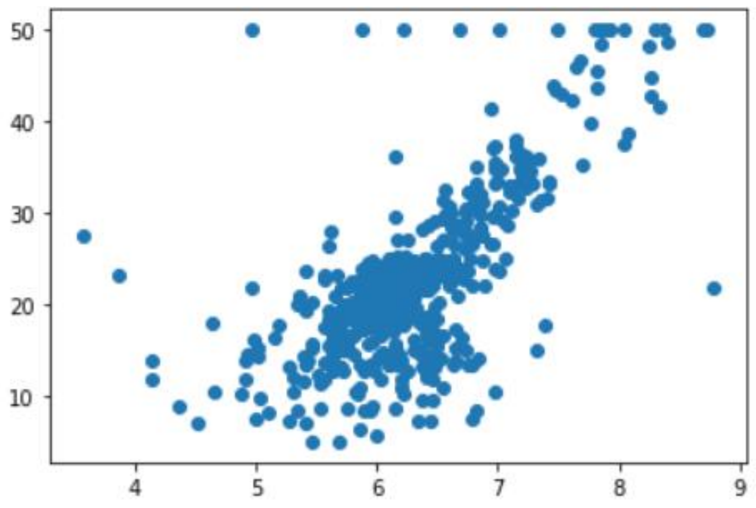


Fig. 2 scattering plot of price vs. room number

We found that the data values have some exceptional distributions at the top of the plot. That indicates they may be outliers owing to the practical consideration (take any price larger than 50 as 50). However, they are harmful to our model training. So remove them.

```
x = x[y < 50.0]
y = y[y < 50.0]

plt.scatter(x, y)
plt.show()
```

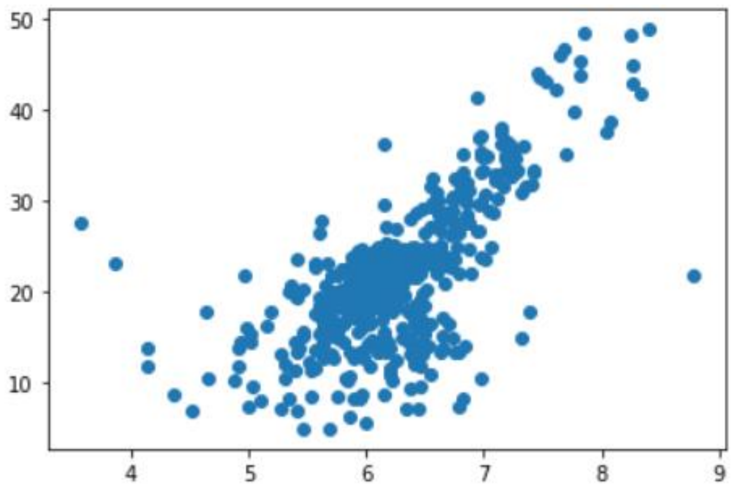


Fig. 3 remove the outliers

From this picture we can see that the data is nearly linear, although just from one dimension. Now we use X to denote all attributes

```
# Use full features
X = boston.data
y = boston.target
X = X[y < 50.0]
y = y[y < 50.0]
```

```
X.shape
```

```
(490, 13)
```

Split data: now we divide the whole dataset into training set and test set using scikit learn *model_selection* module.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
y_train.shape
```

```
(367,)
```

Usually we also split another set called validation set. We should use test for evaluation, which means the model will not change after the evaluation. However, we want to see how good is our model than change the parameters according to some results. The solution is splitting a validation set from training set for adjusting our model. When we think it is good enough, we test our model on test set.

A more rigorous and costly way is cross validation. In that method, training set is divided into several same size pieces and take every pieces as validation set in turn.

Linear regression: now we try to implement simple linear regression because the dataset seems linear.

```
from sklearn.linear_model import LinearRegression
```

```
lin_reg = LinearRegression()
```

```
lin_reg.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

The model has been trained just by few lines of codes. Now make a prediction for testing.

```
# Make a prediction
y_0_hat = lin_reg.predict(X_test[0].reshape(1,-1))
y_0_hat
```

```
array([23.84926326])
```

```
# the real target value is
y_test[0]
```

```
21.9
```

Notice that in scikit learn, standard interface for machine learning is 1) instantiate a learner with several super parameters or not; 2) use *fit()* method and feed the learner training data; 3) use *predict()* for prediction. Moreover, the data preprocessing algorithms also have the same interface, they just use *tramsform()* instead of *predict()*.

See the trained parameters.

```
lin_reg.coef_
```

```
array([-1.17653933e-01,  3.33290753e-02, -5.58580359e-02,  1.01820272e+00,
       -1.30134325e+01,  3.58643578e+00, -2.60252207e-02, -1.27009079e+00,
        2.71037017e-01, -1.37721935e-02, -8.83461872e-01,  7.78099860e-03,
       -3.40673604e-01])
```

```
lin_reg.intercept_
```

```
34.867798938662176
```

Use evaluation method to see if it is a good model. The *score()* method uses R-square.

```
lin_reg.score(X_test, y_test)
```

```
0.8102817015888306
```

3. Polynomial regression

If you know simple linear regression clearly, you can easily implement polynomial regression. Just a little bit more you should know:

1) Extend the attributes to polynomial attributes. we can achieve that easily by using scikit learn. See *PolynomialFeatures* in module *preprocessing*.
2) As the data extending to polynomial features, the value would be extremely large or small because of the power operation. That will influence the use of gradient descent which runs in background when we call *fit()*. So a normalization or standardization is necessary. See *StandardScaler* in *preprocessing*.
3) Pipeline can help us assembling several preprocessing functions and one last learning process together. It uses like

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures

poly_reg = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('std_scaler', StandardScaler()),
    ('lin_reg', LinearRegression())
])
```

   Than poly_reg has the same interface with other learners.

4) Regularization in scikit learn is *RidgeRegression*. See it in *linear_model*. Use it if you want to add regularization in your model.

4. Lab requirement

Please use the real world dataset, **California housing price**, in this lab and training a model to fit it and evaluate the performance. You can use simple linear regression, polynomial regression or more complicated base functions like Gaussian function or use regularization method. Make sure at least **20% data for testing** and choose one evaluation method you think good. **Please do not just training your model and say that is good enough, you need to analysis the bias and variance.** For that, validation set or cross validation is needed. Compare the score in training set and validation set. If they are both good enough, than use it on test set. **Your test set can only be used as final evaluation!**

5. References

https://scikit-learn.org/stable/