George Mason University

Learning From Data Fall 2024

Computer Exercise #4

Assigned: September 18 2024
Due Date: September 26, 2024 (Midnight)

Overview: The focus of this computer exercise is on **regression**. As with previous computer exercises, your report should be submitted as a PDF file. Your grade will depend, in part, on the competeness of your report, its conciseness, and the amount of extra work and experiments you report on. The due date for this is Thursday, September 26 at midnight. No submissions after this time will be accepted.

The purpose of this lab is to give you some experience with linear and nonlinear regression, and multivariate regression. This is your opportunity to experiment and, along the way (if you put in the effort), become move familiar with regression techniques.

Unlike classification, where the goal is to learn a function $f(\boldsymbol{x})$ that maps vectors into classes, with regression the goal is to learn a function that maps vectors $\boldsymbol{x}$ into a real variable y. Whereas with classification the error to be minimized may be the classification error, with regression the error is often a least squares error. For example over a training set $(\boldsymbol{x}_n, y_n)$ of N data samples, a least squares error is

$$\mathcal{E}_{LS} = \sum_{n=1}^{N} (y - g(\boldsymbol{x}))^2$$

In this computer exercise you wlll look at the problem of trying to predict the gasoline mileage of a car based on one or more features, such as weight, horsepower, and number of cylinders.

Computer Exercise 4.1 (Linear Regression):

1. Getting Started

The first step in this exercise is to load a number of classes, all of which you have worked with before.

```
#Common imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

Others for various tasks related to regression will be loaded as they are needed.

2. The Data Set

The data set used in this exercise is a set of 9-dimensional vectors containing features of 398 different cars over different model years. The features in the data set include miles per gallon, weight, and engine displacement. The data is stored in a .csv file and may be loaded and put into a `dataframe` as follows (assuming the file `cars_data.csv` is stored in the current directory),

```
input_file = "datasets/Cars_data.csv"
df = pd.read_csv(input_file, header = 0)
```

The first thing that you should do is explore the data set to see what the features are,

```
df.head()
```

and to get a feeling of what the feature values are. One of these features, `mpg` (miles per gallon), will be the target value for our regression analysis. In other words, we will want to predict the gas efficiency of a car based on one or more of its features.

In order to use scikit-*learn* classes, you will want to convert the dataframe `df` into a numpy array,

```
X_cars = df.to_numpy()
```

Since `mpg` (feature 0) will be the target variable, remove it from `X_cars` and put it into the numpy array `y`,

```
y = X_cars[:,0].reshape(-1,1)
X = np.delete(X_cars,0,1)
```

We now have a data set `(X,y)` consisting of 398 eight-dimensional data samples `X` along with 398 target values, `y`. You should verify that you have done everything correctly and have the correct values in `X` and `y`.

3. Visualization

In order to get a feel for the data, and to see what features might be useful in the prediction of gas mileage, let's start with some data visualizations.

Considering, first, a simple linear predictor based on one variable, what would be a good feature to use? In linear regression, it is important for the target variable to be correlated with the predictor. If there is no correlation between the input feature and the target value, then a linear predictor will not be very effective. As we know, correlation is a measure of the linear dependence between two random variables, and if the correlation between two features is high (a value that is close to ± 1), then there is an approximate linear relationship between the two variables.[1] Therefore, a good starting point might be to look at the correlation between `mpg` and each of the features in the data samples $\boldsymbol{x}$.

The method `corr`, which is part of the `seaborn` library, finds the correlations between each pair of features in a data frame,

```
corr = df.corr()
print(corr)
```

and from this you may find some candidate features to consider. You may also create a heat map as an aid in visualization,

```
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns)
```

Those cells that are the whitest have the largest positive correlation while those that are the darkest have the largest negative correlation (assuming that you use the standard colormap).

Another visualization tool that may be useful is `pairplot`, which generates an array of scatter plots between pairs of features, with plots along the diagonal showing the estimated density function of a feature. This visualization may be created as follows:

```
sns.pairplot(df)
plt.show()
```

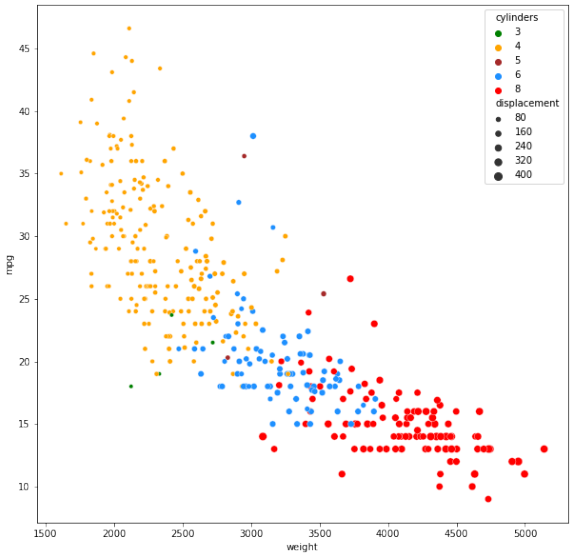
[1] In multivariate linear prediction, where two or more features are used to predict the target value, the situation is a little different.

If you would like to add a hue to these plots to color code the samples according to the values of a particular feature, such as `cylinders`, then consider using[2]

```
sns.pairplot(df,hue='cylinders').
```

Instead of an array of scatter plots, here is an example of a single scatter plot of `acceleration` versus `mpg` with a color and circle size of each sample coded according to the number of cylinders,[3]

```
plt.figure(figsize=(10,10))
sns.scatterplot(data=df,x='acceleration',y='mpg',hue='cylinders',size='cylinders')
```



Questions:

(a) Based on what you have been able to learn about the data set, what features seem to be the best for predicting gas mileage?

(b) Are there any features that seem to be irrelevant or not useful in predicting gas mileage? Which ones are they, and why would they not be useful?

4. Linear Regression

Having become familiar with the data set, we now focus on designing a linear regressor to predict the gas mileage of a car from a single feature.

Based on your visualizations in the previous part, pick one variable to use as a predictor of gas mileage. Assuming that the feature is `acceleration` (this is selected only as an example and is not necessarily the best one), put this feature into the numpy array `X`:

```
X = X_cars[:,4].reshape(-1,1)  #Put feature 4 into the numpy array X
```

With the target array `y` created previously, you should now have a data set `(X,y)` with `X` having a shape `(398,9)` and `y` having a shape `(398,1)`. You should confirm that you have the right features in `X` and `y`.

[2]Note that with a color code based on the number of cylinders, you will get a warning because some of the features will not have data samples in all possible numbers of cylinders, so the density functions in these cases cannot be generated.

[3]You can change the palette of colors be setting `palette='name'` where `name` is the name of the color palette.

Question: Is it important or necessary to scale the date before performing regression?

Create Training and Test Sets

Now we are ready to perform linear regression on the data set. First, however, split the data set `(X,y)` into a training set, `(X_train,y_train)` and a test set `(X_test,y_test)` with 20% of the samples going into the test set.

Learn a Linear Regression Function

To design a linear predictor to minimize the squared error, use the class `LinearRegression` in the `linear_model` library of scikit-*learn*. The `fit` method is used to find the regressor from a data set. The steps are:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,y_train)
```

Once the regressor is designed, you may print the coefficients of the linear regressor,

$$y = ax + b$$

where `lr.coef_[0]` is the regression coefficient a and `lr.intercept_` is the intercept b.

Evaluating the Performance

To evaluate the performance, there are several things that may be done. The first is to find the R^2 score, which is defined by

$$R^2 = 1 - \frac{\sum_{n=1}^{N}(y - h(x_n))^2}{\sum_{n=1}^{N}(y - m_y)^2}$$

where $h(x_n)$ is the linear prediction of y using the sample x_n, and m_y is the mean of the samples in the data set. The term in the numerator is the sum of the squares of the residuals, and the term in the denominator is the sample variance of the data set. The best possible value is 1.0. A constant model that always predicts the expected value of `y`, ignoring the input features, would get a score of 0.0. Negative values for R^2 are possible, and indicate a very poor model, one that is worse than a horizontal line. The R^2 score may be found using the `score` method, and here is an example of how to use the score method on the test set:

```
r2 = lr.score(X_test, y_test)
print('r2 =', r2)
```

Another performance measure is the **mean-square error**, the sum of the squares of the residuals. This may be found using the `mean_squared_error` method in the `metrics` library,

```
from sklearn.metrics import mean_squared_error as mse
print('Mean-square error =:', mse(y_predicted, y_test)
```

Exercises:

(a) Design a linear predictor using the training set `(X_train,y_train)`.

(b) Evaluate the performance of your linear regressor on the test set `(X_test,y_test)` and discuss how well your predictor performs.

(c) Create a scatter plot of the residuals to see if there is any bias or trends that were missed by your predictor. Note that in order to do this, you will need to use your linear regressor to predict the value of `mpg` from the feature you selected. This may be done using the `predict` method. For the training set, this would be done as follows"

```
y_pred = lr.predict(X_test)
```

(d) Discuss your results. What conclusions can you draw from this experiment?

It is possible that your linear regressor did not work very well for your selected variable. There are more things that one can do, such as looking at other features.

Exercises:

(a) Repeat the previous exercise using a different feature for the linear regression.

(b) Evaluate the performance and compare it to your previous regressor.

Another option to improve the linear regressor is to use more than one feature. This is known as **multivariate linear regression**. The first thing that needs to be done is to create a new data set `(X,y)` where `X` contains two or more features. This may be done easily as follows. If you want to use features 3 and 6, then a data matrix using only these features may be created as follows:

```
X2_train = X_train[:,[3,6]]
```

If you wanted to use more than two features, you would just expand the list of features in the list from `[3,5]` to something else.

Exercises:

(a) Pick two features to use in a multvariate linear regressor, and form the appropriate data matrix `X`. Split your data set into a training set and a test set.

(b) Design a two-variable linear regressor, evaluate its performance, and compare it to previous regressors. You may plot the regressor along with a scatter plot of the data as follows:

```
xmin = X1_test.min()
xmax = X1_test.max()
xx = np.linspace(xmin, xmax, 50).reshape(-1,1)
y_pred = lr.predict(xx)
fig, ax = plt.subplots(1,1, figsize=(10,6))
ax.plot(xx,y_pred)
ax.plot(X1_test, y_test, 'o')
```

(c) Now design a regressor that uses **all of the features** except the model year and origin. Does it perform better or worse than your previous regressors? Comment and explain what you observe.

Computer Exercise 4.2 (Kernel Ridge Regression):

When a linear regressor is not performing well and, in particular when the residuals have a pattern or are not uniformly distributed over the range of values of the predictor, then a nonlinear regressor may be necessary. With only one feature, a quadratic regressor would find the best linear regressor on the feature vector $\boldsymbol{z}$ that is transformed from $\boldsymbol{x}$ as follows,

$$\boldsymbol{x}^T = [1,\ x] \Longrightarrow \boldsymbol{z}^T = [1, x, x^2]$$

With two features, the transformed feature vector would be

$$\boldsymbol{x}^T = [1,\ x_1,\ x_2] \Longrightarrow \boldsymbol{z}^T = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2]$$

and similar transformations with more than two features. Nonlinear regression is done easily using the Scikit-*Learn* class for Kernel Ridge Regression (KRR) where the transformations are done with kernels, such as

$$k(\boldsymbol{x}, \boldsymbol{x}') = (r + \gamma\langle \boldsymbol{x}, \boldsymbol{x}' \rangle)^p$$

The KRR class is `KernelRidge` in the `kernel_ridge` library,

```
from sklearn.kernel_ridge import KernelRidge
```

This class makes it easy to experiment with different nonlinear regressors with L_2 regularization. Read the documention to become familiar with how to use this class.

Experiment:

1. Decide what feature you believe would be best for a nonlinear regressor, and design a regressor using a polynomial kernel with $p = 2$ to predict the number of miles per gallon of a car.
2. Evaluate the performance of your classifier and compare it to previous designs. Is there any evidence of overfitting? Experiment with the amount of regularization that is controlled by `alpha`.
3. Repeat (a) and (b) using two features and with $p = 3$.