Hyperiondev

TASK

Multiple Linear Regression

Visit our website

Introduction

WELCOME TO THE MULTIPLE LINEAR REGRESSION TASK!

In the previous tasks, we explored the relationship between one explanatory variable and the response or dependent variable. In many cases, you will be interested in more than a single input variable. Thus, we need to learn about Multiple Linear Regression, a special case of simple linear regression that adds more terms to the regression model.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.
Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

MULTIPLE LINEAR REGRESSION

In the last task, we saw how simple linear regression can shed light on the relationship between two variables. We looked only at the impact of a single independent variable on a single outcome variable. Our example case, however, had two independent variables: TV and social media. Multiple Linear Regression allows us to model the impact of those two variables combined on the outcome variable.

Modelling multiple variables is quite useful. A simple linear regression approach may over- or underestimate the impact of a variable on the outcome because it does not have any information about the impact of other variables. Imagine, for example, that our social media budget and billboards budget were increased simultaneously and sales went up shortly thereafter. Our simple regression models would attribute the entire increase in sales to the single variable they were modelling, which would be incorrect. A multiple linear regression model could make a less biased prediction of how much each type of advertisement contributes to sales.

The extension of simple linear regression is fairly straightforward. Instead of a function for **Y** with only one coefficient, the function has coefficients for each variable. In the case of two variables, the formula takes the form:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Note that the coefficients (β_i) will not just be the same values as independent simple linear regression models would return. This extended model will adjust each value according to the relative contribution of each variable.

A note from the
Hyperion Team

Over time, there have been several myths surrounding Machine Learning such as "Machine Learning is just summarising data" or "Learning algorithms just discover correlations between events", but probably one of the most prominent is that "Machine Learning can't predict unseen events". This line of thought goes thus: If something has never happened before, its predicted probability must be zero. Right?

On the contrary, machine learning is the art of predicting rare events with high accuracy. If A is one of the causes of B, and B is one of the causes of C, A can lead to C, even if we've never seen it happen before. A practical example of the predictive powers of machine learning is the spam filter in your mail inbox. Every day, spam filters correctly flag freshly concocted spam emails, based on mails you marked as spam earlier and other such predictor data.

APPLICATION

Both simple and multiple linear regression can be performed very simply using scikit-learn. Provided that your dataset is properly pre-processed, sklearn infers on its own whether your input has one or multiple features. Fitting looks like this:

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X, y)
```

Let's apply this to some data on the impact of TV, radio and newspaper advertising on sales. This dataset **Advertising.csv** is also in your lesson folder if you want to try it for yourself.

Multiple linear regression applied to this dataset returns the coefficients 0.045, 0.189 and 0.001.

To see what these coefficients say about the variables, it helps to see them in the context of the formula for **Y**:

sales = 2.94 + 0.045(TV) + 0.189(radio) - 0.001(newspaper)

One thing this shows is that TV and radio advertising has a positive impact on sales, but newspaper advertising has an impact that is close to zero, and even a bit below it. Negative coefficients mean that the variables have an opposite relationship: as one increases, the other one decreases. When comparing the coefficients for TV and radio, we see that the coefficient for radio is larger than that for TV. This means that radio contributes more to total sales than TV does.

Based on these findings, we may recommend a focus on radio advertising. We might also recommend that the newspaper advertising budget be scrapped.

TRAINING AND TEST DATA IN MACHINE LEARNING

When you are teaching a student arithmetic summation, you want to start by teaching them the pattern of summation, and then test whether they can apply that pattern. You will show them that 1+1=2, 4+5=9, 2+6=8, and so forth. If the student memorises all the examples, they could answer the question 'what is 2+6?' without understanding summation. To test whether they have recovered not just the facts but the pattern behind the facts, you need to test them on numbers that you have not exposed them to directly. For example, you might ask them 'what is 4+9?'

This intuition forms the motivation behind training and testing data in machine learning. We create a model (e.g. regression) based on some data that we have, but we do not use all of our data: we hide some data samples from the model and then test our model on the hidden data samples to confirm that the model is making valid predictions as opposed to reiterating what was given to it in the training dataset.

A **training set** is the actual dataset that we use to train the model. The model sees and learns from this data. A **test set** is a set of held-out examples used only to assess the performance of a model. Although the best ratio depends on how much data is available, a common split is a ratio of 80:20. For example, 8000 training examples and 2000 test cases. Sklearn allows us to do this quite easily:

```
# import from sklearn the train_test_split functionality
from sklearn.model_selection import train_test_split

# pass your x and y variables in the function and get all four at the same
time
# the test size parameter is used to tell how to split, 0.25 means 25% to be
test samples

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25)
```

One thing to watch out for when dividing the data is that the test set and training set should not be systematically different. If the total data set is ordered in some way, for example by alphabet or by the time of collection, the test set might contain different kinds of instances from the training set. If that happens, the model will perform badly, not because it did not learn from the data well, but because it is tested on a different kind of task from the one it learned to do. For this purpose, it is customary to investigate the distribution of labels in your data and make sure they are similar across your training and test data, to check that

samples with different labels are distributed suitably. In the example below, notice that there are no *0* samples in the test set.

```
from sklearn.model_selection import train_test_split

X = [1,2,3,4,2,6,7,8,6,7]
y = [0,0,0,0,0,1,1,1,1,1]

x_train,x_test,y_train,y_test= train_test_split(X,y,test_size =
0.2,shuffle=False)
print("y_train {}".format(y_train))

print("y_test {}".format(y_test))

#which prints: (notice there are no 0 labels in the test set)
>> y_train [0, 0, 0, 0, 0, 1, 1, 1]
>>y_test [1, 1]
```

This can largely be solved using **`shuffle=True`** as a parameter in **`train_test_split`**. This means that the labels would be distributed randomly between the training and testing set, so it would be less likely that the training and testing data would be systematically different.

```
#notice there is now a 0 label in the test set
>>y_train [1, 0, 0, 1, 0, 1, 0, 1]
>>y_test [0, 1]
```

However, there is still the possibility that most samples of one type end up in the train set, without a representative proportion in the test set. This could also result in lower performance than expected. Consider the case where we set **`test_set=0.4`**, we get the following:

```
#notice the labels are not distributed proportionally between test and train
>>y_train [0, 1, 1, 1, 0, 1]
>>y_test [0, 0, 0, 1]
```

A further way to ensure that data is represented equally in both the training and testing data is to make use of the *stratify* parameter. This ensures that labels are represented in as close to the same proportions as possible in both the training and testing data.

```
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size= 0.5,
shuffle=True, stratify=y)
#note that the labels are now distributed in the same proportion across the
train and test sets
>>y_train [0, 0, 1, 1, 1, 0]
>>y_test [1, 0, 0, 1]
```

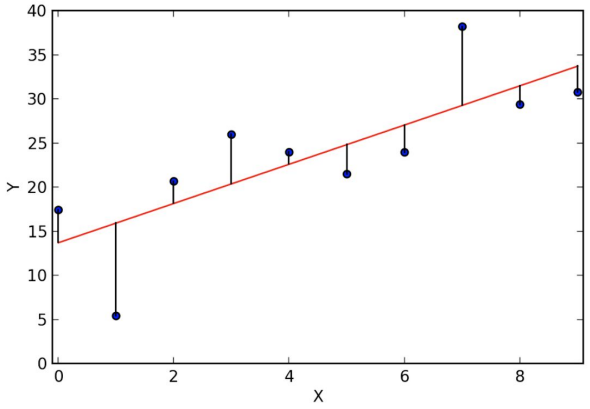
Note: you need to be using sklearn version 0.17 or higher to make use of the stratify parameter.

TRAINING AND TEST ERROR

We've discussed before how a regression model is only an approximation of the data. The model predicts values that are close to the observed training values, in hopes of making good predictions on unseen data. The difference between the actual values and the predictions is called the *error*.

The training dataset error and the test dataset error can be obtained by comparing the predictions to the known, true labels (the gold standard). Of the training error and test error, the test error value is more important as it is the more reliable assessment of the value of the model. After all, we want to use our model on unknown data points, as opposed to applying it to cases for which we already have the actual outcome. In most cases, the training error value will also be lower than the test error value.

There are many different ways to measure the error. As said, the error is the difference between observed and predicted values, as in this plot:



Here we have observed data (**Y**) is in blue and prediction of a regression model (**y_pred**) along the red line. The error is depicted by vertical black lines. There are a

number of different ways one can aggregate the error to get a final score for the model. Two common ones are the Root Mean Squared Error (RMSE) and R squared (R^2).

Instructions

- First, open the accompanying notebook (Multiple Regression.ipynb) to see a code example of multiple regression in Python.
- Read the following post on **the R^2 metric** to learn more about how to evaluate regression models.

Compulsory Task 1

Follow these steps:

- Read **Iris.csv** into a Jupyter notebook.
- Differentiate between the independent variables and the dependent variable and assign them to variables x and y.
- Convert the classes from strings (like 'Iris-setosa') to numbers using `np.unique(y, return_inverse=True)[1]`
- Generate training and test sets comprising 80% and 20% of the data respectively.
- Generate a multiple linear regression model using the training set. Use all of the 4 independent variables.
- Print out the intercept and coefficients of the trained model.
- Generate predictions for the test set.
- Generate an error plot for your predictions. Hint: make sure you use an appropriate range for the 'x' values.
- Compute the Root Mean Squared Error of your model on the test set.
- Ensure your notebook is well commented with topics and comments on what your code is accomplishing.

Compulsory Task 2

Follow these steps:

- Read in the **hourlywagedata.csv**. This dataset contains information about the hourly wage paid to employees who work in a hospital, along with their position, age and number of years worked, see 'More about the data' below.
- Clean up the data: drop any rows which contain a missing value in any one of the 4 variables (ie. containing a string containing only a space " ") and convert the hourly wage values to floats.
- Generate three plots showing the average hourly wage against the three categorical independent variables.

  As we did in the advertising example, we are going to use multiple linear regression to train a model to predict the hourly wage.
- Generate a training and test set, again with an 80:20 split.
- Investigate whether the distribution of the variables in your test set is representative of the distribution in the training set. For example, find out if the number of nurses who work in the hospital and the office found in the test set are similar proportion wise to those found in the training set.
- Generate a multiple linear regression model using the training set. Use all the independent variables.
- Generate predictions for the test set.
- Generate an error plot for your predictions.
- Compute the Root Mean Squared Error of your model on the test set.
- Print the coefficients and try to interpret them as we did for the advertisement model.

More about the data:

Position: 0 = "Hospital"
1 = "Office"

Agerange : 1 = "18-30"
2 = "31-45"

3 = "46-65"

Yrsscale: 1 = "5 or less"
2 = "6-10"
3 = "11-15"
4 = "16-20"
5 = "21-35"
6 = "36 or more"

Completed the task(s)?

Ask your mentor to review your work!

Review work

Things to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **Python or Notepad++** may not be installed correctly.
2. If you are not using Windows, please ask your mentor for alternative instructions.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.