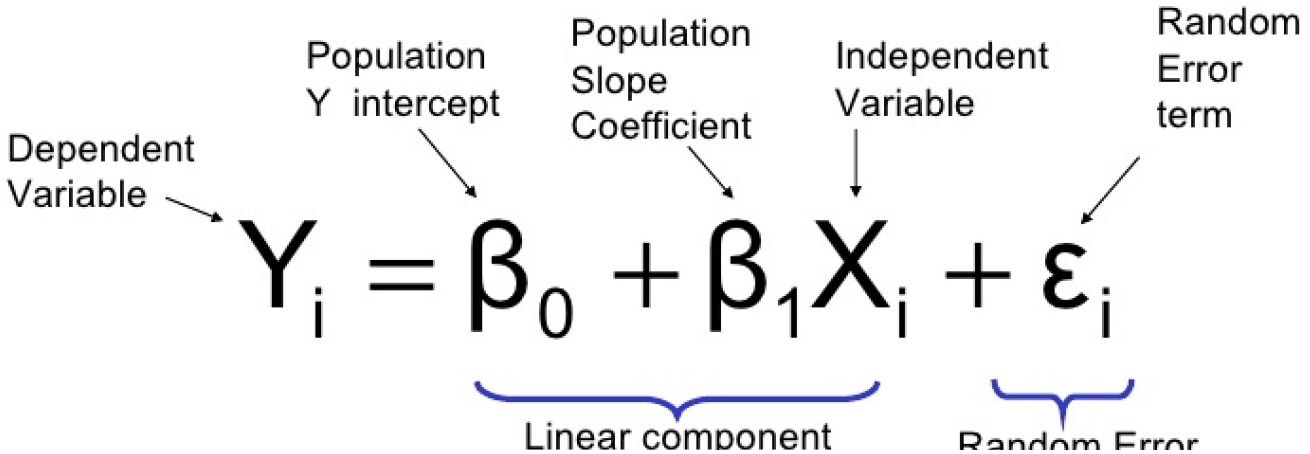
Linear Regression in Python

Dannar Mawardi Follow

Aug 21, 2017 · 4 min read

In linear regression, you are attempting to build a model that allows you to predict the value of new data, given the training data used to train your model. This will become clear as we work through this post.



Courtesy of Department of Statistics, ITS Surabaya

Above, we can see the simple linear regression equation. The y-variable is considered our response or dependent variable. This is what we intend to predict, for example, *Sales* is a popular choice.

The B0 is the y-intercept, i.e) Where X=0 and the line meets the y-axis. The B1X is essentially, our B1 (The amount of impact our X has on our y), and our X, which is our feature/independent variable. Unlike our y-variable, multiple X's can be used with a corresponding beta (coefficient for each). This allows us to create a model with many feature (X) variables to predict values in y. The random error component is irreducible error.

First Step: Visualization

Using visualisation, you should be able to judge which variables have a linear relationship with y. Start by using Seaborn's pairplot.

In this case, we have used "Sales" as our response/y. Substitute the variables list of beta's with your anticipated feature list

```
sns.pairplot(data, x_vars=[b1, b2, b3], y_vars='Sales', kind='reg')
```

Seaborn Pairplot

Additional parameters to use:

size= : Allows you to manipulate the size of the rendered pairplot

kind= 'reg' : Will attempt to add line of best fit and a 95% confidence band. Will aim to minimize sum of squared error.

Second Step: SK Learn — Setting variables

Scikit-Learn expects X to be a feature matrix (Pandas Dataframe) and y to be a response vector (Pandas Series). Let's begin by separating our variables as below.

Handling your features (X):

In this example, we are using the columns TV, Radio, and Social as predictor variables.

```
# Assign feature columns as list: feature_cols
feature_cols = ['TV', 'Radio', 'Social']

# Assign to X a subset of the data including only feature columns
X = data[feature_cols]
```

Handling your response (y):

```
# Assign to y the response variable as Pandas series
y = data['Sales']
```

If you are wondering why a capital X is used for features, and lowercase y for response, it is mainly due to convention.

Third Step: SK Learn — Splitting our data

Splitting X & y into training and testing sets:

By passing our X and y variables into the train_test_split method, we are able to capture the splits in data by assigning 4 variables to the result.

```
# Import SK Learn train test split
```

```
from sklearn.cross_validation import train_test_split

# Assign variables to capture train test split output
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Fourth step: SK Learn — Training our model

Firstly, importing of sklearn.linear_model is required for us to access LinearRegression. It then needs to be instantiated and model fit to our training data. This is seen below.

```
# Import SK Learn linear regression
from sklearn.linear_model import LinearRegression

# Instantiate
linreg = LinearRegression()

# Fit model to training data i.e) Learn coefficients
linreg.fit(X_train, y_train)
```

Instantiate and fitting model to training data

Fifth step: Interpreting Coefficients

The coefficients will allow us to model our equation with values for our beta's. The linreg variable (assigned to a LinearRegression object), is able to have the intercept and coefficients extracted, using the code below.

```
# Prints y-intercept
print(linreg.intercept_)

# Prints the beta coefficients in same order as passed
print(linreg.coef_)

# Zip can pair feature names and coefficients together
zip(feature_cols, linreg.coef_)
```

Extracting data from model

The intercept will be your B0 value; and each coefficient will be the corresponding Beta for the X's passed (in their respective order).

Sixth step: Making predictions based on your model

Making predictions based on your model is as simple as using the code below: passing the predict method your test data. This will return predicted values of y given the new test X data.

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

Returns results of y predictions given X data in X_test

Seventh Step: Model Evaluation

There are three primary metrics used to evaluate linear models. These are: Mean absolute error (MAE), Mean squared error (MSE), or Root mean squared error (RMSE).

MAE: The easiest to understand. Represents average error

MSE: Similar to MAE but noise is exaggerated and larger errors are "punished". It is harder to interpret than MAE as it's not in base units, however, it is generally more popular.

RMSE: Most popular metric, similar to MSE, however, the result is square rooted to make it more interpretable as it's in base units. It is recommended that RMSE be used as the primary metric to interpret your model.

Below, you can see how to calculate each metric. All of them require two lists as parameters, with one being your predicted values and the other

being the true values

```
# Import metrics library
from sklearn import metrics

# Print result of MAE
print(metrics.mean_absolute_error(y_test, y_pred))

# Print result of MSE
print(metrics.mean_squared_error(y_true, y_pred))

# Print result of RMSE
print(np.sqrt(metrics.mean_squared_error(y_true, y_pred)))
```

Eighth Step: Feature Selection

Once you have obtained your error metric/s, take note of which X's have minimal impacts on y. Removing some of these features may result in an increased accuracy of your model.

So, We begin a process of trial and error, where the process is started over again, until a satisfactory model is produced. The steps below may be useful for this particular part.

1. Replace feature_cols & X
2. Train_test_split your data
3. Fit the model to linreg again using linreg.fit
4. Make predictions using (y_pred = linreg.predict(X_test))
5. Compute RMSE
6. Repeat until RMSE satisfactory

Machine Learning Linear Regression Python Pandas Sklearn