1 Estimate coefficient and Variance of Coefficient in Linear Regression

1.1 Best Estimate of Coefficient

Simple derviation for the case of linear regression:

y=a+bx+u is the true relationship, also known as population regression line. Here, u is random error.

Then,

$y_i = \hat{a} + \hat{b}x_i$, where $\hat{a}, \hat{b}$ are the estimated coefficients.

and $\bar{y} = \hat{a} + \hat{b}\bar{x}$, where $\bar{y}, \bar{x}$ are the mean of y and x, respectively.

$y_i - \bar{y} = \hat{b}(x_i - \bar{x})$

or, $(x_i - \bar{x})(y_i - \bar{y}) = \hat{b}(x_i - \bar{x})^2$ (I don't know why I should multiply the equation by $(x_i - \bar{x})$, but the standard formula has it this way).

Since we will be dealing with a dataset with many events, it is more meaningful to sum over all data points.

$\sum_i (x_i - \bar{x})(y_i - \bar{y}) = \hat{b} \sum_i (x_i - \bar{x})^2$

or, $\hat{b} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$ is the best estimate of the coefficient b.

1.2 Variance of Coefficient

$\text{var}(\hat{b}) = \text{var}(\frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2})$

Variance of constants = 0. So, we will only keep the non-constant terms. Expand y_i and $\bar{y}$ in terms of x_i and $\bar{x}$.

$\text{Var}(\hat{b}) = (\frac{1}{\sum_i (x_i - \bar{x})^2})^2 \, Var(\sum_i (x_i - \bar{x}) u_i)$ Online websites say that only u is random variable, and we know that $Var(kX) = k^2 \, Var(X)$, so, we get:

$\text{Var}(\hat{b}) = \frac{\sum_i (x_i - \bar{x})^2}{(\sum_i (x_i - \bar{x}))^2} \, Var(u_i)$

$= \frac{\sigma^2}{\sum_i (x_i - \bar{x})^2}$

2 Estimate coefficient and Variance using Matrix Approach

Root and other software use matrices to estimate best values for coefficients and their corresponding variances.

See the following link for linear regression:

https : //goo.gl/yTDs7Z

Basically, $\hat{b} = \frac{X^T Y}{(X^T X)}$

and $\text{Var}(\hat{b}) = \frac{\sigma^2}{X^T X}$

Where X is $(n \, X \, 1)$ matrix with n datapoints, Y is $1 \, X \, n$ matrix with n labels. See Minuit manual to understand how root uses Hessian and covariance matrices to determine errors in the estimation of the coefficients.

3 Cross-Validation

In cross-validation, you divide the training dataset into n subsets (in sklearn, you specify the value of n via the parameter 'cv'). Then, you fit the machine learning classifier on $(n-1)$ subsets, and evaluate the fit on the nth subset. There are n possible ways of selecting $(n-1)$ datasets, so this validation is done n times, each time with a different set of $(n-1)$ datasets. The evaluation is done by computing MSE (mean square error) between predicted and actual labels of the nth dataset.

For each validation, you get an MSE. So, at the end of cross validation, you get n MSE values. You can compute the mean and standard deviation of these MSEs in sklearn. The smaller the mean and std deviation, the better is the model fit to your data.

4 Bootstrapping

Bootstrapping example: assume we are interested in the average (or mean) height of people worldwide. We cannot measure all the people in the global population, so instead we sample only a tiny part of it, and measure that. Assume the sample is of size N; that is, we measure the heights of N individuals. From that single sample, only one estimate of the mean can be obtained. In order to reason about the population, we need some sense of the variability of the mean that we have computed. The simplest bootstrap method involves taking the original data set of N heights, and, using a computer, sampling from it to form a new sample (called a 'resample' or bootstrap sample) that is also of size N. The bootstrap sample is taken from the original by using sampling with replacement (e.g. we might 'resample' 5 times from [1,2,3,4,5] and get [2,5,4,4,1]), so, assuming N is sufficiently large, for all practical purposes there is virtually zero probability that it will be identical to the original "real" sample. This process is repeated a large number of times (typically 1,000 or 10,000 times), and for each of these bootstrap samples we compute its mean (each of these are called bootstrap estimates). We now have a histogram of bootstrap means. This provides an estimate of the shape of the distribution of the mean from which we can answer questions about how much the mean varies. (The method here, described for the mean, can be applied to almost any other statistic or estimator.)

In Bootstrapping, the main assumption is that the sample of N data points is an accurate representation of the population. One must understand that Bootstrapping does not give any new information other than what is contained in the sample itself. The power of this method is that resampling over that sample on a large enough scale reveals the sampling distribution of the statistic at issue, for example the mean.

5 Numpy: Argpartition and Partition

These are two very useful features to partially sort arrays in Python. First, let us understand some basics of Numpy array arrays:

Suppose, $x = [1, 9, 3, 7, 5, 6, 2, 8]$. Here $D = 8$ is the shape of the array, that is, it has 8 elements.

The following operations are very frequently used in Data science:

- Select first n elements of the array using $x[: n]$.
  For example, $x[: 5] = [1, 9, 3, 7, 5]$
- Select last n elements of the array using $x[-n :]$. This is same as $x[D - n + 1 : D]$, where D is the shape of the 1-dimensional array.
  For example, $x[-5 :] = x[8 - 5 + 1 : 8] = [7, 5, 6, 2, 8]$
  A common application is in sorted arrays which are sorted in increasing order by default, and you want the 5 biggest numbers.
- $x[5 :] = x[5 : D] = [5, 6, 2, 8]$
- $x[: -5] = x[1 : D - 5] = x[1 : 3] = [1, 9, 3]$

5.1 Partition

Syntax: numpy.partition(array, k). It finds the $k + 1^{th}$ smallest element of the array and partitions the array to the left and right of that element. All elements to the left are smaller, and all right elements are larger than the $k+1^{th}$ element. NOTE: the left and right partitions are not sorted.

Partition is very useful because it saves a lot of computation time in many applications. For example, if you have an array of 20 scores of attributes, and you want only the 3 smallest scores. Then, you do not need to sort the entire array, instead use Partition. Once you find the 3 smallest scores, you don't need to sort the remaining 17 scores.

Example:$x = [1, 9, 3, 7, 5, 6, 2, 8]$

- np.partition$(x, 5)$ means partition about the 6th smallest element.
  np.partition$(x, 5) = [6, 1, 3, 2, 5, 7, 9, 8]$. Here, the $6th$ smallest element is 7.
- np.partition$(x, -5)$ means partition about the 5th (**not 6th**) biggest element. np.partition$(x, -5) = [2, 3, 1, 5, 6, 7, 9, 8]$. Here, the $5th$ biggest element is 5.

5.2 Argpartition

Syntax: numpy.argpartition(array, k). It performs the job of partition but stores the indices of the elements and not the values.

ArgPartition is very useful when you want to refer to the index of the array element. For example, you have an array of 20 scores of attributes, and you want to select the attributes with 3 highest scores. In this case, you don't want the scores themselves, but you want the indices of the attributes corresponding to the 3 highest scores. This is used extensively in Data science where you want to select only the important attributes and drop the rest.

Example: $x = [1, 9, 3, 7, 5, 6, 2, 8]$
Indices_of_x $= [0, 1, 2, 3, 4, 5, 6, 7]$

- np.partition$(x, 5) = [6, 1, 3, 2, 5, 7, 9, 8]$.
  Then, np.argpartition$(x, 5) = [5, 0, 2, 6, 4, 3, 1, 7]$, where index of 6 is 5, index of 1 is 0, index of 3 is 2 and so on.
- np.partition$(x, -5) = [2, 3, 1, 5, 6, 7, 9, 8]$
  Then, np.argpartition$(x, -5) = [6, 2, 0, 4, 5, 3, 1, 7]$
- np.partition$(x, -5)[-5:] = [5, 6, 7, 9, 8]$
  Then, np.argpartition$(x, -5)[-5:] = [4, 5, 3, 1, 7]$. This gives the INDICES of the 5 biggest elements of x.

6 Lists and Tuples

There are fundamental differences between lists and tuples:
Lists:

- are ordered
- can contain arbitrary objects
- can be accessed by index
- can be nested, i.e. a list within a list
- are mutable i.e. dimensions can be changed easily. E.g. append
- are dynamic

Tuples:

- are ordered
- cannot be changed

Tuples are used when you don't want data to be modified. E.g., one of the components of our dictionary needs to be of immutable type.

A very good discussion with many examples can be found at:
https://realpython.com/python-lists-tuples/

7 np.c_

This command attaches two 1-D arrays of same dimension together to make a matrix. Eg.:

$x = [1, 2, 3]$

$y = [5, 6, 7]$

$z = \text{np.c\_}[x, y]$

z

Output:array

[1, 5],
[2, 6],
[3, 7]