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 $a,\hat{b}$ are the estimated coefficients.

and $\bar{y} = \hat{a} + \hat{b}\bar{x}$, where $y,\bar{x}$ 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

var$(\hat{b})$ = 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 $\bar{x}$.

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:

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 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 Adaboost or Boosted Decision Trees

1)Unlike neural networks and SVMs, the AdaBoost (or boosted decision trees) training process selects only those features known to improve the predictive power of the model, reducing dimensionality and potentially improving execution time as irrelevant features do not need to be computed.

2) To understand the basics of AdaBoost: `https://www.youtube.com/watch?v=ix6IvwbVpw0`YoutubeLink

6 Random Forest

Best explanation (in my opinion) is here: `http://blog.echen.me/2011/03/14/laymans-introduction-to-random-forests/`
Suppose you?re very indecisive, so whenever you want to watch a movie, you ask your friend Willow if she thinks you?ll like it. In order to answer, Willow first needs to figure out what movies you like, so you give her a bunch of movies and tell her whether you liked each one or not (i.e., you give her a labeled training set). Then, when you ask her if she thinks you?ll like movie X or not, she plays a 20 questions-like game with IMDB, asking questions like ?Is X a romantic movie??, ?Does Johnny Depp star in X??, and so on. She asks more informative questions first (i.e., she maximizes the information gain of each question), and gives you a yes/no answer at the end.

Thus, Willow is a decision tree for your movie preferences.

But Willow is only human, so she doesn?t always generalize your preferences very well (i.e., she overfits). In order to get more accurate recommendations, you?d like to ask a bunch of your friends, and watch movie X if most of them say they think you?ll like it. That is, instead of asking only Willow, you want to ask Woody, Apple, and Cartman as well, and they vote on whether you?ll like a movie (i.e., you build an ensemble classifier, aka a forest in this case).

Now you don?t want each of your friends to do the same thing and give you the same answer, so you first give each of them slightly different data. After all, you?re not absolutely sure of your preferences yourself ? you told Willow you loved Titanic, but maybe you were just happy that day because it was your birthday, so maybe some of your friends shouldn?t use the fact that you liked Titanic in making their recommendations. Or maybe you told her you loved Cinderella, but actually you really really loved it, so some of your friends should give Cinderella more weight. So instead of giving your friends the same data you gave Willow, you give them slightly perturbed versions. You don?t change your love/hate decisions, you just say you love/hate some movies a little more or less (formally, you give each of your friends a bootstrapped version of your original training data). For example, whereas you told Willow that you liked Black Swan and Harry Potter and disliked Avatar, you tell Woody that you liked Black Swan so much you watched it twice, you disliked Avatar, and don?t mention Harry Potter at all.

By using this ensemble, you hope that while each of your friends gives somewhat idiosyncratic recommendations (Willow thinks you like vampire movies more than you do, Woody thinks you like Pixar movies, and Cartman thinks you just hate everything), the errors get canceled out in the majority. Thus, your friends now form a bagged (bootstrap aggregated) forest of your movie preferences.

There?s still one problem with your data, however. While you loved both Titanic and Inception, it wasn?t because you like movies that star Leonardio DiCaprio. Maybe you liked both movies for other reasons. Thus, you don?t want your friends to all base their recommendations on whether Leo is in a movie or not. So when each friend asks IMDB a question, only a random subset of the possible questions is allowed (i.e., when you?re building a decision tree, at each node you use some randomness in selecting the attribute to split on, say by randomly selecting an attribute or by selecting an attribute from a random subset). This means your friends aren?t allowed to ask whether Leonardo DiCaprio is in the movie whenever they want. So whereas previously you injected randomness at the data level, by perturbing your movie preferences slightly, now you?re injecting randomness at the model level, by making your friends ask different questions at different times.

And so your friends now form a random forest.

7 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.
- $x[5 :] = x[5 : D] = [5, 6, 2, 8]$
- $x[: -5] = x[1 : D - 5] = x[1 : 3] = [1, 9, 3]$

7.1 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.

7.2 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.

8 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/`

9 np.c

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

$x = [1, 2, 3]$
$y = [5, 6, 7]$
$z =$ np.c_$[x, y]$
z
Output:array
$[1, 5]$,
$[2, 6]$,
$[3, 7]$

10 Standard Error, Root Mean Square Error, Standard Deviation

Standard deviation is the square root of the sample variance. It has nothing to do with fits.
$\sigma = \sqrt{\frac{\sum_i (y_i - y_{mean})^2}{n}}$
where y_mean is the mean of the response in the sample.

Standard error is the standard deviation divided by $\sqrt{n}$, where n is the population in the sample. It gives the standard error of the mean, i.e. how close is the estimated mean to the true mean.
Standard error = $\frac{\sigma}{\sqrt{n}}$

RMSE is square root of Mean Square Error and it gives an estimate of how good a fit is. It is used to measure distance between some values and prediction for those values from a fit.
$RMSE = \frac{\sum_i (y_i - y_{estimated})^2}{n}$

11 Identification of Outliers using Studentized Residuals

Residuals, ϵ, for each data point are estimated as follows:
$\epsilon = y_{actual} - y_{estimated}$

Plotting residuals versus the predictor gives an idea of whether the fit to the data did or did not fully capture the relationship between the predictor and response.

A studentized residual is used to identify outliers in the data. A studentized residual is
$\frac{residual}{\epsilon}$
where ϵ is the standard error.

12 Correlation and Covariance

A very nice explanation can be found at link

Correlation = $cov(x, y)/\sigma_x\,\sigma_y = = \sigma_(x, y)/\sigma_x\,\sigma_y$
Correlation ranges from -1 to 1 whereas covariance ranges from -infinity to +infinity.

13 A/B Testing and Null Hypothesis

A very nice explanation can be found at link

A/B testing is used to find that feature with property A gives a better response than the same feature with property B, in a specified time. For example, the feature can be Amazon website design A versus webdesign B. Note here that we are not comparing two different features, instead we are comparing two types of the same feature. The response can be 'website visitors that became customers'. For the A/B testing, we define a null hypothesis and an alternative hypothesis:

Null hypothesis (H0): Amazon.com visitors that receive Layout B will not have higher end-of-visit conversion rates compares to visitors that receive Layout A.
Alternative hypothesis (H1): Amazon.com visitors that receive Layout B will have higher end-of-visit conversion rates compared to visitors that receive layout A

To perform the test, it is very important to create two random test samples within minimized bias. See the link for further details.

14 Lagrange Multiplier

This method is used when you want to maximize a function f(x,y) given a constraint g(x,y). A nice example is shown in a video from Khan's academy (link).

Example:
Goal: Maximize revenue, f, given a specific monthly budget, g, for a steel company.

Revenue, $f(x, y) = 100 * x^{2/3} * y^{1/3}$ where,
x is hours of labor and y tons of steel used in product manufacture.

Budget, $g = 20 * x + 2000 * y = 20,000$ is the constraint (that is, you have $20,000$ dollars monthly budget).

Maximizing revenue while satisfying the constraint means that the contour of f(x,y) for some revenue will be just touch g(x,y) and the point of tangent will be the desired values (x_0, y_0). Which means that at (x_0, y_0) the gradients of f(x,y) and g(x,y) match. That is,

$\nabla f = \lambda \nabla g$. Solving this will give us (x_0, y_0) and λ.
Here, λ is the Lagrangian multiplier.

15 Confidence Interval and Margin of Error

A good explanation from Khan's academy is here.

Suppose we want to estimate the percentage of votes candidate A will win at an upcoming election. We perform a survey on a sample population of 100 and estimate it to be about 50%. Repeating this survey over n samples (with each sample consisting of 100 people) will give us a normal distribution with an estimated mean of say about 0.54 and a standard deviation of 0.05.

Note that a 2σ interval corresponds to a 95% confidence interval. What it means is that there is a 95% chance that the true percentage of votes the candidate A will win is within $0.54 \pm 2\sigma$. 2σ is call the margin of error.
So, another way of saying this is that "with 95% confidence interval, between 0.44 to 0.64 voters will vote for A. The margin of error is 0.1".