Find p-value (significance) in scikit-learn LinearRegression

Asked 6 years, 7 months ago Active 5 months ago Viewed 252k times

192

81

Highly active question. You have enough reputation to answer or unprotect this question.

How can I find the p-value (significance) of each coefficient?

```
lm = sklearn.linear_model.LinearRegression()
lm.fit(x,y)
```

python numpy statistics scikit-learn regression Edit tags

Share Edit Follow Close Flag Unprotect

asked Jan 13 '15 at 17:46

elplatt
2,649 3 15 20

2 Not your answer, but maybe an answer to others: scipy provides pvalues in linregression: docs.scipy.org/doc/scipy-0.14.0/reference/generated/... – DaveRGP Apr 9 '19 at 14:05

1 it only works for one dimension vs one dimension. – music_piano Jan 4 '20 at 15:54

14 Answers

Active Oldest Votes

16

*EDIT: **This is not the right way to do it**, see comments below. Others might make the same mistake, which is why I think it is good to keep this answer*

You could use sklearn.feature_selection.f_regression.

click here for the scikit-learn page

Share Edit Follow Flag

edited Mar 22 at 15:37

answered Jan 24 '16 at 23:44

Pinna_be
4,188 1 14 30

1 So those are F-tests? I thought the p-values for linear regression was typically for each individual regressor, and it was a test vs the null of the coefficient being 0? More explanation of the function would be necessary for a good answer. – wordsforthewise Oct 3 '17 at 4:46

@wordsforthewise documentation page says that the returned value is an array of p_values. So it is indeed a value for each individual regressor. – piedpiper Mar 20 '18 at 17:59

4 Don't use this method as it is not correct! It performs univariate regressions, but you probably want a single multivariate regression – user357269 Oct 17 '19 at 15:47

5 No, don't use f_regression. The actual p-value of each coefficient should come from the t test for each coefficient after fitting the data. f_regression in sklearn comes from the univariate regressions. It didn't build the mode, just calcuate the f score for each variable. Same as chi2 function in sklearn This is correct: import statsmodels.api as sm mod = sm.OLS(Y,X) – music_piano Jan 4 '20 at 16:29

@RichardLiang, use sm.OLS() is the correct way to calculate p-value (multivariate) for any algorithm? (like decision tree, svm, k-means, logistic regression, etc)? I would like a generic method to get p-value. Thanks – Gilian Jul 17 '20 at 18:26

226

This is kind of overkill but let's give it a go. First lets use statsmodel to find out what the p-values should be

```
import pandas as pd
import numpy as np
from sklearn import datasets, linear_model
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from scipy import stats

diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
print(est2.summary())
```

and we get

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.518
Model:                            OLS   Adj. R-squared:                  0.507
Method:                 Least Squares   F-statistic:                     46.27
Date:                Wed, 08 Mar 2017   Prob (F-statistic):           3.83e-62
Time:                        10:08:24   Log-Likelihood:                -2386.0
No. Observations:                 442   AIC:                             4794.
Df Residuals:                     431   BIC:                             4839.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        152.1335      2.576     59.061      0.000     147.071     157.196
x1           -10.0122     59.749     -0.168      0.867    -127.448     107.424
x2          -239.8191     61.222     -3.917      0.000    -360.151    -119.488
x3           519.8398     66.534      7.813      0.000     389.069     650.610
x4           324.3904     65.422      4.958      0.000     195.805     452.976
x5          -792.1842    416.684     -1.901      0.058   -1611.169      26.801
x6           476.7458    339.035      1.406      0.160    -189.621    1143.113
x7           101.0446    212.533      0.475      0.635    -316.685     518.774
x8           177.0642    161.476      1.097      0.273    -140.313     494.442
x9           751.2793    171.902      4.370      0.000     413.409    1089.150
x10           67.6254     65.984      1.025      0.306     -62.065     197.316
==============================================================================
Omnibus:                        1.506   Durbin-Watson:                   2.029
Prob(Omnibus):                  0.471   Jarque-Bera (JB):                1.404
Skew:                           0.017   Prob(JB):                        0.496
Kurtosis:                       2.726   Cond. No.                         227.
==============================================================================
```

Ok, let's reproduce this. It is kind of overkill as we are almost reproducing a linear regression analysis using Matrix Algebra. But what the heck.

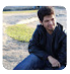
```
lm = LinearRegression()
lm.fit(X,y)
params = np.append(lm.intercept_,lm.coef_)
predictions = lm.predict(X)

newX = pd.DataFrame({"Constant":np.ones(len(X))}).join(pd.DataFrame(X))
MSE = (sum((y-predictions)**2))/(len(newX)-len(newX.columns))

# Note if you don't want to use a DataFrame replace the two lines above with
# newX = np.append(np.ones((len(X),1)), X, axis=1)
# MSE = (sum((y-predictions)**2))/(len(newX)-len(newX[0]))

var_b = MSE*(np.linalg.inv(np.dot(newX.T,newX)).diagonal())
sd_b = np.sqrt(var_b)
ts_b = params/ sd_b

p_values =[2*(1-stats.t.cdf(np.abs(i),(len(newX)-len(newX[0])))) for i in ts_b]

sd_b = np.round(sd_b,3)
ts_b = np.round(ts_b,3)
p_values = np.round(p_values,3)
params = np.round(params,4)

myDF3 = pd.DataFrame()
myDF3["Coefficients"],myDF3["Standard Errors"],myDF3["t values"],myDF3["Probabilities"]
= [params,sd_b,ts_b,p_values]
print(myDF3)
```

And this gives us.

```
    Coefficients  Standard Errors  t values  Probabilities
0       152.1335            2.576    59.061          0.000
1       -10.0122           59.749    -0.168          0.867
2      -239.8191           61.222    -3.917          0.000
3       519.8398           66.534     7.813          0.000
4       324.3904           65.422     4.958          0.000
5      -792.1842          416.684    -1.901          0.058
6       476.7458          339.035     1.406          0.160
7       101.0446          212.533     0.475          0.635
8       177.0642          161.476     1.097          0.273
9       751.2793          171.902     4.370          0.000
10       67.6254           65.984     1.025          0.306
```

So we can reproduce the values from statsmodel.

Share Edit Follow Flag

edited Jul 9 '20 at 3:07

Sambit Paul
23 1 4

answered Mar 8 '17 at 17:17
JARH
2,276 1 7 4

2 what does it meant that my var_b are all Nans? Is there any underlying reason why the linear algebra part fails? – famargar Mar 10 '17 at 14:23

1 It looks like `code` np.linalg.inv can sometimes return a result even when the matrix is non-invertable. That might be the issue. – JARH Mar 10 '17 at 17:11

8 @famargar I also had the problem of all `nan` s. For me it was because my `X` 's were a sample of my

data so the index was off. This causes errors when calling `pd.DataFrame.join()` . I made this one line change and it seems to work now: `newX = pd.DataFrame({"Constant":np.ones(len(X))}).join(pd.DataFrame(X.reset_index(drop=True)))` – pault Dec 1 '17 at 18:46

1 @mLstudent33 The "probabilities" column. – skeller88 Apr 26 '20 at 2:39

1 For me, `p_values =[2*(1-stats.t.cdf(np.abs(i),(len(newX)-len(newX[0])))) for i in ts_b]` returns all Nan, and I rewrite the degree of freedom with `len(newX) - 1` and get the same p_value as the `statsmodels` given. But I am not sure whether the df is correct for the case or not. If wrong, a correction to my concept is appreciated. Tks. – Denny Chen Jul 29 at 7:52

|

4

For a one-liner you can use the [pingouin.linear_regression](#) function (*disclaimer: I am the creator of Pingouin*), which works with uni/multi-variate regression using NumPy arrays or Pandas DataFrame, e.g:

```
import pingouin as pg
# Using a Pandas DataFrame `df`:
lm = pg.linear_regression(df[['x', 'z']], df['y'])
# Using a NumPy array:
lm = pg.linear_regression(X, y)
```

The output is a dataframe with the beta coefficients, standard errors, T-values, p-values and confidence intervals for each predictor, as well as the R^2 and adjusted R^2 of the fit.

Share Edit Follow Flag

answered Apr 28 '20 at 23:05
Raphael
381 1 6

10

An easy way to pull of the p-values is to use statsmodels regression:

```
import statsmodels.api as sm
mod = sm.OLS(Y,X)
fii = mod.fit()
p_values = fii.summary2().tables[1]['P>|t|']
```

You get a series of p-values that you can manipulate (for example choose the order you want to keep by evaluating each p-value):

```
Out[538]: x1    1.103093e-21
          x2    1.170528e-07
          Name: P>|t|, dtype: float64
```

Share Edit Follow Flag

edited Aug 16 '19 at 20:41
G. Sliepen
5,637 1 12 25

answered Apr 8 '19 at 9:29
benaou mouad
302 3 10

Use sm.OLS() is the correct way to calculate p-value (multivariate) for any algorithm? (like decision

tree, svm, k-means, logistic regression, etc)? I would like a generic method to get p-value. Thanks
– Gilian Jul 17 '20 at 18:27

0

This post is hidden. It was deleted 2 years ago by the post author.

Modified elyase's answer (Python3). Solved: "super","init signature (no varargs)" and "sse dimension" problems:

```
from sklearn import linear_model
from scipy import stats
import numpy as np

class LinearRegressionWithP(linear_model.LinearRegression):
    """
    LinearRegression class after sklearn's, but calculate t-statistics
    and p-values for model coefficients (betas).
    Additional attributes available after .fit()
    are `t` and `p` which are of the shape (y.shape[1], X.shape[1])
    which is (n_features, n_coefs)
    This class sets the intercept to 0 by default, since usually we include it
    in X.
    """

    def __init__(self, fit_intercept=True, normalize=False, copy_X=True,
                 n_jobs=1):
        self.fit_intercept = fit_intercept
        self.normalize = normalize
        self.copy_X = copy_X
        self.n_jobs = n_jobs

    def fit(self, X, y):
        self = super().fit(X, y)
        sse = np.sum((self.predict(X) - y) ** 2, axis=0) / float(X.shape[0] -
X.shape[1])
        se = np.array([np.sqrt(np.diagonal(sse * np.linalg.inv(np.dot(X.T, X))))])
        with np.errstate(divide='ignore'):
            self.t = self.coef_ / se
        self.p = 2 * (1 - stats.t.cdf(np.abs(self.t), y.shape[0] - X.shape[1]))
        return self
```

Share Edit Follow Flag

answered Mar 26 '19 at 6:45
VovaM
152 1 6

Comments disabled on deleted / locked posts / reviews

0

This answer is hidden. This answer was deleted via review 2 years ago by Baum mit Augen ♦, Dave, rassar, Mihai Chelaru.

I think that there is a mistake in the top answer (I do not have enough reputation to comment). The student distribution should have n - p degrees of freedom with n the number of observations and p the number of coefficient (including the intercept if there is an intercept), not n-1.

Share Edit Follow Flag

answered Dec 7 '18 at 18:40

jean
1

1 What student distribution? And where does degrees of freedom appear? – lxop Dec 7 '18 at 19:20

Please don't misuse the answers field to post comments, that could lead to you being blocked from posting answers at all. That said, this has the elements of a proper answer but needs a bit more of explanation. – brasofilo Dec 7 '18 at 19:45

This does not provide an answer to the question. Once you have sufficient reputation you will be able to comment on any post; instead, provide answers that don't require clarification from the asker. - From Review – rassar Dec 7 '18 at 22:06

Comments disabled on deleted / locked posts / reviews

0

This answer is hidden. This answer was deleted via review 2 years ago by Mark Rotteveel, Ormoz, n2o, Daniel Beck.

What about finding p-value in scikit-learn Logistic Regression?

Share Edit Follow Flag

answered Sep 27 '18 at 14:29

felipef
1

This does not provide an answer to the question. You can search for similar questions, or refer to the related and linked questions on the right-hand side of the page to find an answer. If you have a related but different question, ask a new question, and include a link to this one to help provide context. See: Ask questions, get answers, no distractions – Natty Sep 27 '18 at 14:30

This does not provide an answer to the question. Once you have sufficient reputation you will be able to comment on any post; instead, provide answers that don't require clarification from the asker. - From Review – Ormoz Sep 27 '18 at 15:09

Comments disabled on deleted / locked posts / reviews

10

There could be a mistake in @JARH's answer in the case of a multivariable regression. (I do not have enough reputation to comment.)

In the following line:

```
p_values =[2*(1-stats.t.cdf(np.abs(i),(len(newX)-1))) for i in ts_b] ,
```

the t-values follows a chi-squared distribution of degree `len(newX)-1` instead of following a chi-squared distribution of degree `len(newX)-len(newX.columns)-1` .

So this should be:

```
p_values =[2*(1-stats.t.cdf(np.abs(i),(len(newX)-len(newX.columns)-1))) for i in ts_b]
```

(See [t-values for OLS regression](#) for more details)

Share Edit Follow Flag

answered Sep 17 '18 at 14:50

Jules K

101 1 3

13

The code in elyase's answer https://stackoverflow.com/a/27928411/4240413 does not actually work. Notice that sse is a scalar, and then it tries to iterate through it. The following code is a modified version. Not amazingly clean, but I think it works more or less.

```
class LinearRegression(linear_model.LinearRegression):

    def __init__(self,*args,**kwargs):
        # *args is the list of arguments that might go into the LinearRegression object
        # that we don't know about and don't want to have to deal with. Similarly,
**kwargs
        # is a dictionary of key words and values that might also need to go into the
orginal
        # LinearRegression object. We put *args and **kwargs so that we don't have to
look
        # these up and write them down explicitly here. Nice and easy.

        if not "fit_intercept" in kwargs:
            kwargs['fit_intercept'] = False

        super(LinearRegression,self).__init__(*args,**kwargs)

    # Adding in t-statistics for the coefficients.
    def fit(self,x,y):
        # This takes in numpy arrays (not matrices). Also assumes you are leaving out
the column
        # of constants.

        # Not totally sure what 'super' does here and why you redefine self...
        self = super(LinearRegression, self).fit(x,y)
        n, k = x.shape
        yHat = np.matrix(self.predict(x)).T

        # Change X and Y into numpy matricies. x also has a column of ones added to it.
        x = np.hstack((np.ones((n,1)),np.matrix(x)))
        y = np.matrix(y).T

        # Degrees of freedom.
        df = float(n-k-1)

        # Sample variance.
        sse = np.sum(np.square(yHat - y),axis=0)
        self.sampleVariance = sse/df

        # Sample variance for x.
        self.sampleVarianceX = x.T*x

        # Covariance Matrix = [(s^2)(X'X)^-1]^0.5. (sqrtm = matrix square root.  ugly)
        self.covarianceMatrix =
sc.linalg.sqrtm(self.sampleVariance[0,0]*self.sampleVarianceX.I)

        # Standard erros for the difference coefficients: the diagonal elements of the
covariance matrix.
        self.se = self.covarianceMatrix.diagonal()[1:]

        # T statistic for each beta.
        self.betasTStat = np.zeros(len(self.se))
        for i in xrange(len(self.se)):
```

```
            self.betasTStat[i] = self.coef_[0,i]/self.se[i]

        # P-value for each beta. This is a two sided t-test, since the betas can be
        # positive or negative.
        self.betasPValue = 1 - t.cdf(abs(self.betasTStat),df)
```

Share Edit Follow Flag

edited Jul 19 '18 at 19:24
Davide Fiocco
3,851 1 26 56

answered Jan 16 '15 at 0:46
Alex
139 2

6

p_value is among f statistics. if you want to get the value, simply use this few lines of code:

```
import statsmodels.api as sm
from scipy import stats

diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
print(est.fit().f_pvalue)
```

Share Edit Follow Flag

answered Apr 28 '18 at 14:15
Afshin Amiri
2,940 1 15 20

6 This doesn't answer the question since you are using a different library than the one mentioned. – gented Jan 1 '19 at 3:53

1 @gented What are the scenarios where one method of calculation would be better than the other? – Don Quixote Jun 1 '19 at 18:40

0

This post is hidden. It was deleted 3 years ago by Jeremy.

Alex how do I invoke your code? Sorry a newbie here.

Share Edit Follow Flag

answered Mar 6 '18 at 16:35
MWeiss
1 2

3 This does not provide an answer to the question. You can search for similar questions, or refer to the related and linked questions on the right-hand side of the page to find an answer. If you have a related but different question, ask a new question, and include a link to this one to help provide context. See: Ask questions, get answers, no distractions – Natty Mar 6 '18 at 16:36

1 This does not provide an answer to the question. Once you have sufficient reputation you will be able to comment on any post; instead, provide answers that don't require clarification from the asker. - From Review – Mohammad Usman Mar 6 '18 at 17:05

Comments disabled on deleted / locked posts / reviews

6

You can use **scipy** for p-value. This code is from scipy documentation.

```
>>> from scipy import stats
>>> import numpy as np
>>> x = np.random.random(10)
>>> y = np.random.random(10)
>>> slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
```

Share Edit Follow Flag

answered Oct 24 '17 at 13:57
Ali Mirzaei
1,248 1 13 22

3 I don't think this applies for multiple vectors being used during fit – O.rka Aug 3 '18 at 2:05

-1

This post is hidden. It was deleted 5 years ago by Brad Larson ♦.

@Alex - This is great code, Alex! As a follow up question: is there a process to systematically remove non-significant coefficients (and appropriate columns in the matrix (representing calculated variable instance for that coefficient) , of course, and keep recalculating the regression until all coefficients are statistically significant?

Share Edit Follow Flag

answered Sep 15 '15 at 16:57
Toly
1,917 7 21 30

This does not provide an answer to the question. To critique or request clarification from an author, leave a comment below their post - you can always comment on your own posts, and once you have sufficient reputation you will be able to comment on any post. – hiro protagonist Sep 15 '15 at 17:22

1 If you have a new question, please ask it by clicking the Ask Question button. Include a link to this question if it helps provide context. – All Workers Are Essential Sep 15 '15 at 17:48

Comments disabled on deleted / locked posts / reviews

57

scikit-learn's LinearRegression doesn't calculate this information but you can easily extend the class to do it:

```
from sklearn import linear_model
from scipy import stats
import numpy as np


class LinearRegression(linear_model.LinearRegression):
    """
    LinearRegression class after sklearn's, but calculate t-statistics
```

```
    and p-values for model coefficients (betas).
    Additional attributes available after .fit()
    are `t` and `p` which are of the shape (y.shape[1], X.shape[1])
    which is (n_features, n_coefs)
    This class sets the intercept to 0 by default, since usually we include it
    in X.
    """

    def __init__(self, *args, **kwargs):
        if not "fit_intercept" in kwargs:
            kwargs['fit_intercept'] = False
        super(LinearRegression, self)\
                .__init__(*args, **kwargs)

    def fit(self, X, y, n_jobs=1):
        self = super(LinearRegression, self).fit(X, y, n_jobs)

        sse = np.sum((self.predict(X) - y) ** 2, axis=0) / float(X.shape[0] -
X.shape[1])
        se = np.array([
            np.sqrt(np.diagonal(sse[i] * np.linalg.inv(np.dot(X.T, X))))
                                                    for i in range(sse.shape[0])
                    ])

        self.t = self.coef_ / se
        self.p = 2 * (1 - stats.t.cdf(np.abs(self.t), y.shape[0] - X.shape[1]))
        return self
```

Stolen from here.

You should take a look at statsmodels for this kind of statistical analysis in Python.

Share Edit Follow Flag

edited Jan 13 '15 at 18:02

answered Jan 13 '15 at 17:54

elyase

34.9k 10 96 108

3 Well. This does not seep to work because sse is a scalar so sse.shape does not really mean anything.
– piedpiper Mar 20 '18 at 17:52