Fitting empirical distribution to theoretical ones with Scipy (Python)?

INTRODUCTION: I have a list of more than 30 000 values ranging from 0 to 47 e.g.[0,0,0,0,..,1,1,1,1,...,2,2,2,2,..., 47 etc.] which is the continuous distribution.

PROBLEM: Based on my distribution I would like to calculate p-value (the probability of seeing greater values) for any given value. For example, as you can see p-value for 0 would be approaching 1 and p-value for higher numbers would be tending to 0.

I don't know if I am right, but to determine probabilities I think I need to fit my data to a theoretical distribution that is the most suitable to describe my data. I assume that some kind of goodness of fit test is needed to determine the best model.

Is there a way to implement such an analysis in Python (Scipy or Numpy)? Could you present any examples?

Thank you!

python numpy statistics scipy distribution

edited Mar 15 '15 at 10:26
Saullo Castro
24.3k 7 61 119

asked Jul 8 '11 at 6:00
s_sherly
301 1 8 14

1 You have only discrete empirical values but want a continuous distribution? Do I understand that correctly? – Michael J. Barber Jul 8 '11 at 10:25

Exactly, Michael! – s_sherly Jul 8 '11 at 10:37

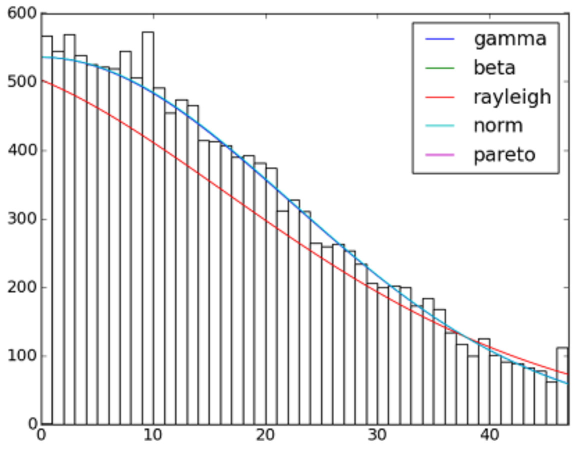
1 It seems nonsensical. What do the numbers represent? Measurements with limited precision? – Michael J. Barber Jul 8 '11 at 10:44

1 Michael, I explained what the numbers represent in my previous question: stackoverflow.com/questions/6615489/… – s_sherly Jul 8 '11 at 11:01

2 That's count data. It's not a continuous distribution. – Michael J. Barber Jul 8 '11 at 11:08

6 Answers

There are 82 implemented distribution functions in SciPy 0.12.0. You can test how some of them fit to your data using their `fit()` method. Check the code below for more details:



```
import matplotlib.pyplot as plt
import scipy
import scipy.stats
size = 30000
x = scipy.arange(size)
y = scipy.int_(scipy.round_(scipy.stats.vonmises.rvs(5,size=size)*47))
h = plt.hist(y, bins=range(48), color='w')

dist_names = ['gamma', 'beta', 'rayleigh', 'norm', 'pareto']

for dist_name in dist_names:
    dist = getattr(scipy.stats, dist_name)
    param = dist.fit(y)
    pdf_fitted = dist.pdf(x, *param[:-2], loc=param[-2], scale=param[-1]) * size
    plt.plot(pdf_fitted, label=dist_name)
    plt.xlim(0,47)
plt.legend(loc='upper right')
plt.show()
```

References:

- Fitting distributions, goodness of fit, p-value. Is it possible to do this with Scipy (Python)?

- Distribution fitting with Scipy

And here a list with the names of all distribution functions available in Scipy 0.12.0 (VI):

```
dist_names = [ 'alpha', 'anglit', 'arcsine', 'beta', 'betaprime', 'bradford',
'burr', 'cauchy', 'chi', 'chi2', 'cosine', 'dgamma', 'dweibull', 'erlang', 'expon',
'exponweib', 'exponpow', 'f', 'fatiguelife', 'fisk', 'foldcauchy', 'foldnorm',
'frechet_r', 'frechet_l', 'genlogistic', 'genpareto', 'genexpon', 'genextreme',
'gausshyper', 'gamma', 'gengamma', 'genhalflogistic', 'gilbrat', 'gompertz',
'gumbel_r', 'gumbel_l', 'halfcauchy', 'halflogistic', 'halfnorm', 'hypsecant',
'invgamma', 'invgauss', 'invweibull', 'johnsonsb', 'johnsonsu', 'ksone',
'kstwobign', 'laplace', 'logistic', 'loggamma', 'loglaplace', 'lognorm', 'lomax',
'maxwell', 'mielke', 'nakagami', 'ncx2', 'ncf', 'nct', 'norm', 'pareto',
'pearson3', 'powerlaw', 'powerlognorm', 'powernorm', 'rdist', 'reciprocal',
'rayleigh', 'rice', 'recipinvgauss', 'semicircular', 't', 'triang', 'truncexpon',
'truncnorm', 'tukeylambda', 'uniform', 'vonmises', 'wald', 'weibull_min',
'weibull_max', 'wrapcauchy']
```

Saullo Castro
24.3k 7 61 119
São José dos Campos - State of São Paulo, Brazil
sites.google.com/site/saullogpcastro/home

I am a PhD on the field of Solid Mechanics, currently working with finite element simulation and implementation of structural analysis methods and processes.
Developed the DESICOS package while working on the European project DESICOS (www.desicos.eu).

the CompMech Google Sites NumPy's gold badge

edited Oct 17 '15 at 0:54

answered May 20 '13 at 14:40
Saullo Castro
24.3k 7 61 119

4 What if `normed = True` in plotting the histogram? You wouldn't multiply `pdf_fitted` by the `size`, right? – aloha Mar 23 '15 at 21:00

@po6 I haven't tested but it seems you are right... Thanks for the suggestion – Saullo Castro Mar 23 '15 at 21:10

1 See this answer if you would like to see what all the distributions look like or for an idea of how to access all of them. – tmthydvnprt Jun 1 at 12:54

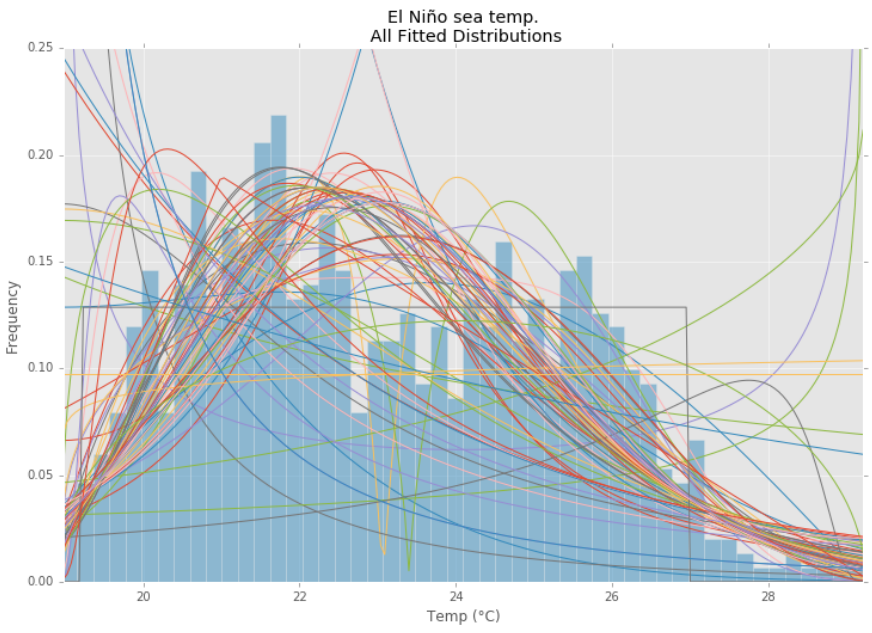
Distribution Fitting with Sum of Square Error (SSE)

This is an update and modification to suallo's answer, that uses the full list of the current `scipy.stats` distributions and returns the distribution with the least SSE between the distribution's histogram and the data's histogram.

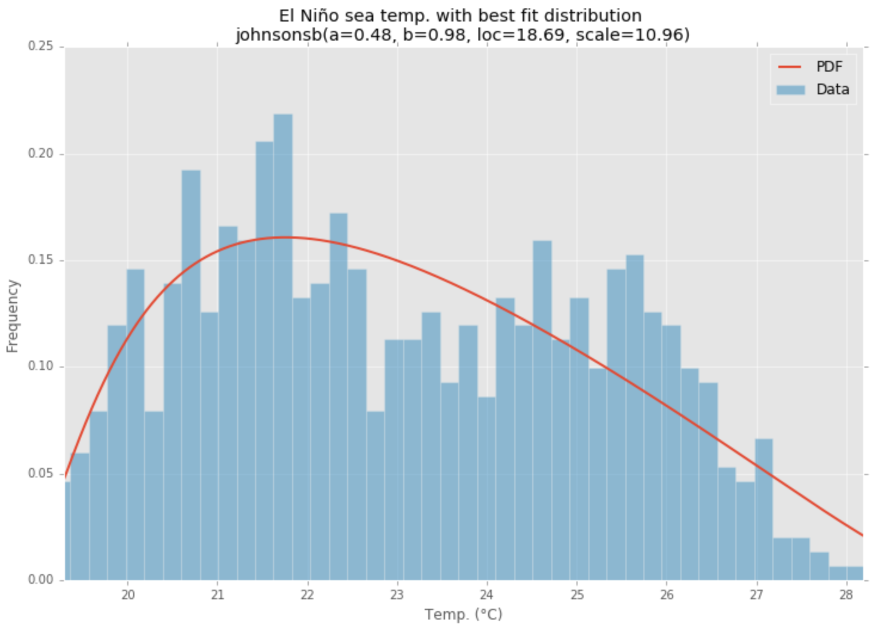
Example Fitting

Using the El Niño dataset from `statsmodels`, the distributions are fit and error is determined. The distribution with the least error is returned.

All Distributions



Best Fit Distribution



Example Code

```
%matplotlib inline

import warnings
import numpy as np
import pandas as pd
import scipy.stats as st
import statsmodels as sm
import matplotlib
import matplotlib.pyplot as plt

matplotlib.rcParams['figure.figsize'] = (16.0, 12.0)
matplotlib.style.use('ggplot')

# Create models from data
def best_fit_distribution(data, bins=200, ax=None):
    """Model data by finding best fit distribution to data"""
    # Get histogram of original data
    y, x = np.histogram(data, bins=bins, normed=True)
    x = (x + np.roll(x, -1))[:-1] / 2.0
    # Distributions to check
    DISTRIBUTIONS = [

st.alpha,st.anglit,st.arcsine,st.beta,st.betaprime,st.bradford,st.burr,st.cauchy,st.


st.dgamma,st.dweibull,st.erlang,st.expon,st.exponnorm,st.exponweib,st.exponpow,st.f,


st.foldcauchy,st.foldnorm,st.frechet_r,st.frechet_l,st.genlogistic,st.genpareto,st.g


st.genextreme,st.gausshyper,st.gamma,st.gengamma,st.genhalflogistic,st.gilbrat,st.go


st.gumbel_l,st.halfcauchy,st.halflogistic,st.halfnorm,st.halfgennorm,st.hypsecant,st


st.invweibull,st.johnsonsb,st.johnsonsu,st.ksone,st.kstwobign,st.laplace,st.levy,st.


st.logistic,st.loggamma,st.loglaplace,st.lognorm,st.lomax,st.maxwell,st.mielke,st.na


st.nct,st.norm,st.pareto,st.pearson3,st.powerlaw,st.powerlognorm,st.powernorm,st.rdi


st.rayleigh,st.rice,st.recipinvgauss,st.semicircular,st.t,st.triang,st.truncexpon,st


st.uniform,st.vonmises,st.vonmises_line,st.wald,st.weibull_min,st.weibull_max,st.wra

    ]

    # Best holders
    best_distribution = st.norm
    best_params = (0.0, 1.0)
    best_sse = np.inf

    # Estimate distribution parameters from data
    for distribution in DISTRIBUTIONS:

        # Try to fit the distribution
        try:
            # Ignore warnings from data that can't be fit
            with warnings.catch_warnings():
                warnings.filterwarnings('ignore')

                # fit dist to data
                params = distribution.fit(data)

                # Separate parts of parameters
                arg = params[:-2]
                loc = params[-2]
                scale = params[-1]

                # Calculate fitted PDF and error with fit in distribution
                pdf = distribution.pdf(x, loc=loc, scale=scale, *arg)
                sse = np.sum(np.power(y - pdf, 2.0))

                # if axis pass in add to plot
                try:
                    if ax:
                        pd.Series(pdf, x).plot(ax=ax)
                    end
                except Exception:
                    pass

                # identify if this distribution is better
                if best_sse > sse > 0:
                    best_distribution = distribution
                    best_params = params
                    best_sse = sse

        except Exception:
            pass

    return (best_distribution.name, best_params)

def make_pdf(dist, params, size=10000):
    """Generate distributions's Propbability Distribution Function """

    # Separate parts of parameters
    arg = params[:-2]
    loc = params[-2]
    scale = params[-1]

    # Get sane start and end points of distribution
    start = dist.ppf(0.01, *arg, loc=loc, scale=scale) if arg else dist.ppf(0.01,
loc=loc, scale=scale)
    end = dist.ppf(0.99, *arg, loc=loc, scale=scale) if arg else dist.ppf(0.99,
loc=loc, scale=scale)
```

```
    # Build PDF and turn into pandas Series
    x = np.linspace(start, end, size)
    y = dist.pdf(x, loc=loc, scale=scale, *arg)
    pdf = pd.Series(y, x)

    return pdf

# Load data from statsmodels datasets
data =
pd.Series(sm.datasets.elnino.load_pandas().data.set_index('YEAR').values.ravel())

# Plot for comparison
plt.figure(figsize=(12,8))
ax = data.plot(kind='hist', bins=50, normed=True, alpha=0.5,
color=plt.rcParams['axes.color_cycle'][1])
# Save plot limits
dataYLim = ax.get_ylim()

# Find best fit distribution
best_fit_name, best_fir_paramms = best_fit_distribution(data, 200, ax)
best_dist = getattr(st, best_fit_name)

# Update plots
ax.set_ylim(dataYLim)
ax.set_title(u'El Niño sea temp.\n All Fitted Distributions')
ax.set_xlabel(u'Temp (°C)')
ax.set_ylabel('Frequency')

# Make PDF
pdf = make_pdf(best_dist, best_fir_paramms)

# Display
plt.figure(figsize=(12,8))
ax = pdf.plot(lw=2, label='PDF', legend=True)
data.plot(kind='hist', bins=50, normed=True, alpha=0.5, label='Data', legend=True,
ax=ax)

param_names = (best_dist.shapes + ', loc, scale').split(', ') if best_dist.shapes
else ['loc', 'scale']
param_str = ', '.join(['{}={:0.2f}'.format(k,v) for k,v in zip(param_names,
best_fir_paramms)])
dist_str = '{}({})'.format(best_fit_name, param_str)

ax.set_title(u'El Niño sea temp. with best fit distribution \n' + dist_str)
ax.set_xlabel(u'Temp. (°C)')
ax.set_ylabel('Frequency')
```

edited Jun 6 at 13:22 answered Jun 3 at 14:26
tmthydvnprt
2,397 2 15 37

1 Please don't add the same answer to multiple questions. Answer the best one and flag the rest as duplicates. See Is it acceptable to add a duplicate answer to several questions? – Bhargav Rao Jun 3 at 14:34

1 @BhargavRao Thank you for the suggestion. I updated that answer to reference this answer and flagged the question. – tmthydvnprt Jun 5 at 18:06

AFAICU, your distribution is discrete (and nothing but discrete). Therefore just counting the frequencies of different values and normalizing them should be enough for your purposes. So, an example to demonstrate this:

```
In []: values= [0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4]
In []: counts= asarray(bincount(values), dtype= float)
In []: cdf= counts.cumsum()/ counts.sum()
```

Thus, probability of seeing values higher than `1` is simply (according to the complementary cumulative distribution function (ccdf):

```
In []: 1- cdf[1]
Out[]: 0.40000000000000002
```

Please note that ccdf is closely related to survival function (sf), but it's also defined with discrete distributions, whereas sf is defined only for contiguous distributions.

edited Jul 8 '11 at 18:24 answered Jul 8 '11 at 10:27
eat
5,615 7 25

`fit()` method mentioned by @Saullo Castro provides maximum likelihood estimates (MLE). The best distribution for your data is the one give you the highest can be determined by several different ways: such as

1, the one that gives you the highest log likelihood.

2, the one that gives you the smallest AIC, BIC or BICc values (see wiki: http://en.wikipedia.org/wiki/Akaike_information_criterion, basically can be viewed as log likelihood adjusted for number of parameters, as distribution with more parameters are expected to fit better)

3, the one that maximize the Bayesian posterior probability. (see wiki: http://en.wikipedia.org/wiki/Posterior_probability)

Of course, if you already have a distribution that should describe you data (based on the theories in your particular field) and want to stick to that, you will skip the step of identifying the best fit distribution.

`scipy` does not come with a function to calculate log likelihood (although MLE method is provided), but hard code one is easy: see Is the build-in probability density functions of `scipy.stat.distributions` slower than a user provided one?

answered Sep 12 '13 at 1:18
CT Zhu
20.9k 3 31 54

It sounds like probability density estimation problem to me.

```
from scipy.stats import gaussian_kde
occurences = [0,0,0,0,..,1,1,1,1,...,2,2,2,2,...,47]
values = range(0,48)
kde = gaussian_kde(map(float, occurences))
p = kde(values)
p = p/sum(p)
print "P(x>=1) = %f" % sum(p[1:])
```

Also see http://jpktd.blogspot.com/2009/03/using-gaussian-kernel-density.html.

edited Sep 21 '11 at 9:34
answered Sep 20 '11 at 17:02
emre
69 5

For future readers: this solution (or at least the idea) provides the simplest answer to the OPs questions ('what is the p-value') - it would be interesting to know how this compares to some of the more involved methods that fit a known distribution. – Greg Oct 10 at 20:58

Forgive me if I don't understand your need but what about storing your data in a dictionary where keys would be the numbers between 0 and 47 and values the number of occurrences of their related keys in your original list?
Thus your likelihood p(x) will be the sum of all the values for keys greater than x divided by 30000.

answered Jul 8 '11 at 6:09
PierrOz
3,902 4 33 50

In this case the p(x) will be the same (equals 0) for any value greater than 47. I need a continuous probability distribution. – s_sherly Jul 8 '11 at 6:15

1 @s_sherly - It would be probably a good thing if you could edit and clarify your question better, as indeed the *"the probability of seeing greater values"* - as you put it - **IS** zero for values that are above the highest value in the pool. – mac Jul 8 '11 at 7:00

---

protected by Saullo Castro Nov 25 '15 at 1:02

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?