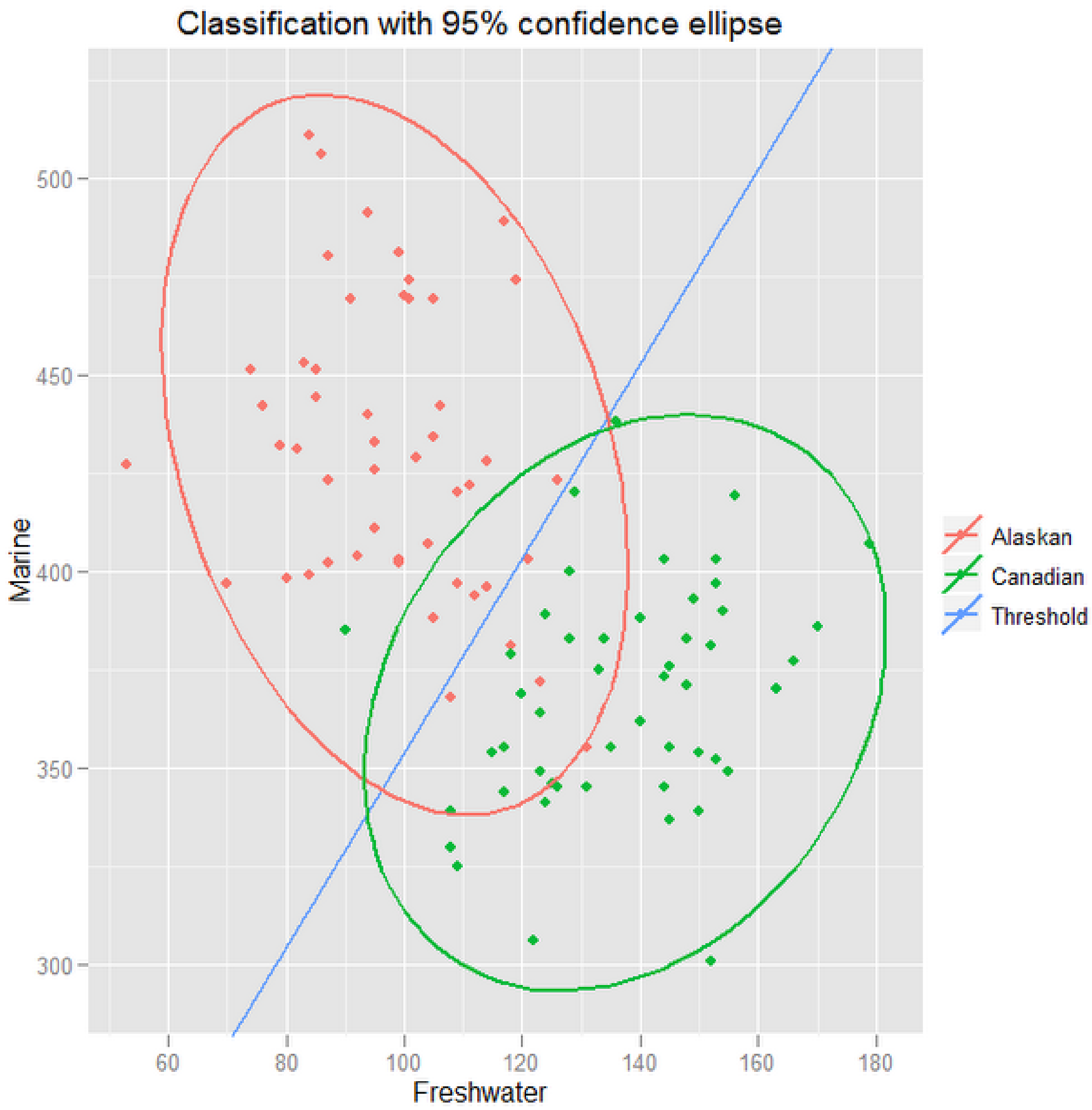
multidimensional confidence intervals

I have numerous tuples (par1,par2), i.e. points in a 2 dimensional parameter space obtained from repeating an experiment multiple times.

I'm looking for a possibility to calculate and visualize confidence ellipses (not sure if thats the correct term for this). Here an example plot that I found in the web to show what I mean:



source: blogspot.ch/2011/07/classification-and-discrimination-with.html

So in principle one has to fit a multivariate normal distribution to a 2D histogram of data points I guess. Can somebody help me with this?

python matplotlib scipy

asked Sep 6 '12 at 13:20

Raphael Roth
3,532 2 13 27

1 What's the input data? Is it an array of 2d points? Do you know in advance that there are 2 clusters? – Daniel Velkov Sep 6 '12 at 19:21

yes I know the number of clusters. I don't yet know what the format of the input data is, I guess a nx2 array where n is the number of points. – Raphael Roth Sep 7 '12 at 5:06

In that case you should cluster them first, then fit a gaussian to each cluster and finally plot the confidence intervals. Look at sklearn.cluster – Daniel Velkov Sep 7 '12 at 17:01

4 Answers

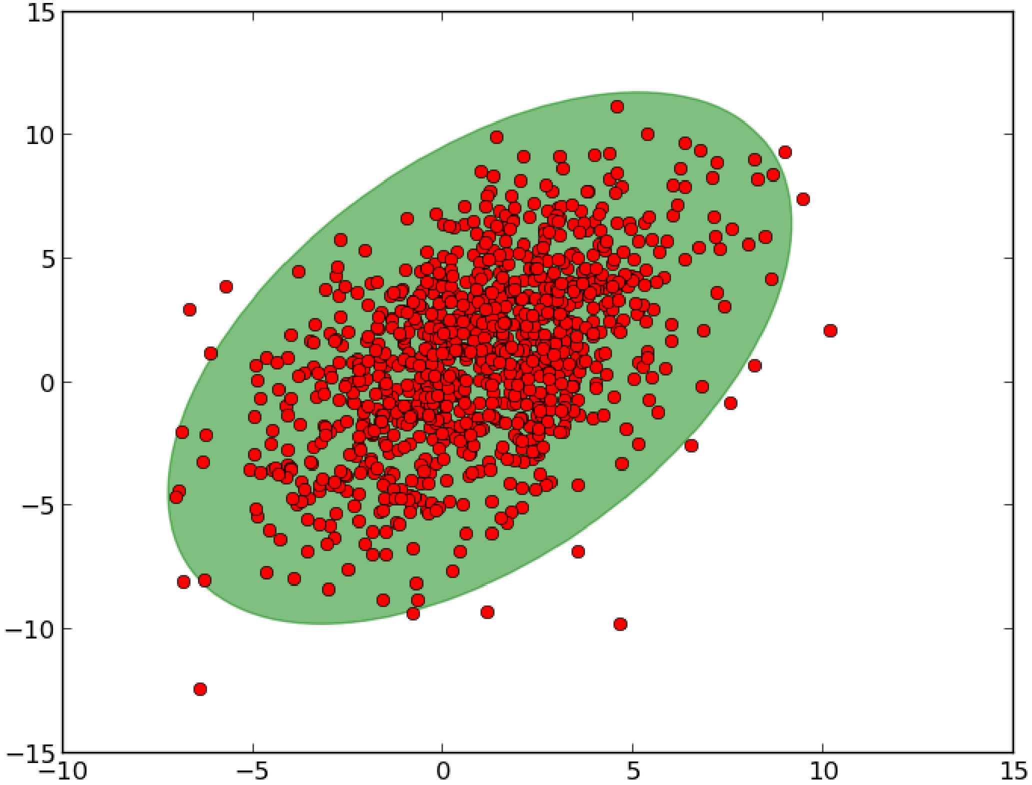
It sounds like you just want the 2-sigma ellipse of the scatter of points?

If so, consider something like this (From some code for a paper here: https://github.com/joferkington/oost_paper_code/blob/master/error_ellipse.py):

```
import numpy as np

import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse

def plot_point_cov(points, nstd=2, ax=None, **kwargs):
    """
    Plots an `nstd` sigma ellipse based on the mean and covariance of a point
    "cloud" (points, an Nx2 array).

    Parameters
    ----------
        points : An Nx2 array of the data points.
        nstd : The radius of the ellipse in numbers of standard deviations.
            Defaults to 2 standard deviations.
        ax : The axis that the ellipse will be plotted on. Defaults to the
            current axis.
```

```
            Additional keyword arguments are pass on to the ellipse patch.

        Returns
        -------
            A matplotlib ellipse artist
        """
        pos = points.mean(axis=0)
        cov = np.cov(points, rowvar=False)
        return plot_cov_ellipse(cov, pos, nstd, ax, **kwargs)

    def plot_cov_ellipse(cov, pos, nstd=2, ax=None, **kwargs):
        """
        Plots an `nstd` sigma error ellipse based on the specified covariance
        matrix (`cov`). Additional keyword arguments are passed on to the
        ellipse patch artist.

        Parameters
        ----------
            cov : The 2x2 covariance matrix to base the ellipse on
            pos : The location of the center of the ellipse. Expects a 2-element
                sequence of [x0, y0].
            nstd : The radius of the ellipse in numbers of standard deviations.
                Defaults to 2 standard deviations.
            ax : The axis that the ellipse will be plotted on. Defaults to the
                current axis.
            Additional keyword arguments are pass on to the ellipse patch.

        Returns
        -------
            A matplotlib ellipse artist
        """
        def eigsorted(cov):
            vals, vecs = np.linalg.eigh(cov)
            order = vals.argsort()[::-1]
            return vals[order], vecs[:,order]

        if ax is None:
            ax = plt.gca()

        vals, vecs = eigsorted(cov)
        theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))

        # Width and height are "full" widths, not radius
        width, height = 2 * nstd * np.sqrt(vals)
        ellip = Ellipse(xy=pos, width=width, height=height, angle=theta, **kwargs)

        ax.add_artist(ellip)
        return ellip
```

```
if __name__ == '__main__':
    #-- Example usage -----------------------
    # Generate some random, correlated data
    points = np.random.multivariate_normal(
            mean=(1,1), cov=[[0.4, 9],[9, 10]], size=1000
            )
    # Plot the raw points...
    x, y = points.T
    plt.plot(x, y, 'ro')

    # Plot a transparent 3 standard deviation covariance ellipse
    plot_point_cov(points, nstd=3, alpha=0.5, color='green')

    plt.show()
```



answered Sep 7 '12 at 15:39

Joe Kington
128k 19 299 307

nice, thanks for the answer. I hope I got this right: Assuming a multivariate normal distribution, one can simply take the eigenvalues and the eigenvectors to calculate the ellipses. – Raphael Roth Sep 10 '12 at 13:15

unfortunately, matplotlib patches cannot be drawn with logarithmic axes (or at least not correctly) as I need to .... why is life so complicated? – Raphael Roth Sep 10 '12 at 13:51

Yeah, I never thought to test it on logarithmic axes. One work-around would be to use a PathPatch, which will draw correctly on logarithmic axes. You'd have to generate points along the ellipse manually, but that's not too hard. – Joe Kington Sep 11 '12 at 0:15

@RaphaelRoth Another possibility to use logarithmic scale would be to fake it by transforming the datapoints and using a tick formatter for the axes (doesn't sound easy, but could be a way) – heltonbiker Oct 31 '12 at 13:02

2 @ThePredator - `arctan2` returns the full angle (can be in any of the 4 quadrants). `arctan` restricts the output to quadrants 1 and 4 (between -pi/2 and pi/2). You may notice that `arctan` takes a single parameter. Therefore, it can't distinguish between angles in quadrants 1 and 4 and a similar angle in quadrants 2 and 3. This is a convention that's shared by many other programming languages, in no small part because C defines them that way. – Joe Kington Jul 6 '15 at 11:44

|

I slightly modified one of the examples above that plots the error or confidence region contours. Now I think it gives the right contours.

It was giving the wrong contours because it was applying the scoreatpercentile method to the joint dataset (blue + red points) when it should be applied separately to each dataset.

The modified code can be found below:

```
import numpy
import scipy
import scipy.stats
import matplotlib.pyplot as plt

# generate two normally distributed 2d arrays
x1=numpy.random.multivariate_normal((100,420),[[120,80],[80,80]],400)
x2=numpy.random.multivariate_normal((140,340),[[90,-70],[-70,80]],400)
```

```
# fit a KDE to the data
pdf1=scipy.stats.kde.gaussian_kde(x1.T)
pdf2=scipy.stats.kde.gaussian_kde(x2.T)

# create a grid over which we can evaluate pdf
q,w=numpy.meshgrid(range(50,200,10), range(300,500,10))
r1=pdf1([q.flatten(),w.flatten()])
r2=pdf2([q.flatten(),w.flatten()])

# sample the pdf and find the value at the 95th percentile
s1=scipy.stats.scoreatpercentile(pdf1(pdf1.resample(1000)), 5)
s2=scipy.stats.scoreatpercentile(pdf2(pdf2.resample(1000)), 5)

# reshape back to 2d
r1.shape=(20,15)
r2.shape=(20,15)

# plot the contour at the 95th percentile
plt.contour(range(50,200,10), range(300,500,10), r1, [s1],colors='b')
plt.contour(range(50,200,10), range(300,500,10), r2, [s2],colors='r')

# scatter plot the two normal distributions
plt.scatter(x1[:,0],x1[:,1],alpha=0.3)
plt.scatter(x2[:,0],x2[:,1],c='r',alpha=0.3)
```

answered Jul 1 '13 at 16:29

Rodrigo
21 3

Refer the post How to draw a covariance error ellipse.

Here's the python realization:

```
import numpy as np
from scipy.stats import norm, chi2

def cov_ellipse(cov, q=None, nsig=None, **kwargs):
    """
    Parameters
    ----------
    cov : (2, 2) array
        Covariance matrix.
    q : float, optional
```

```
        Confidence level, should be in (0, 1)
    nsig : int, optional
        Confidence level in unit of standard deviations.
        E.g. 1 stands for 68.3% and 2 stands for 95.4%.

    Returns
    -------
    width, height, rotation :
         The lengths of two axises and the rotation angle in degree
    for the ellipse.
    """

    if q is not None:
        q = np.asarray(q)
    elif nsig is not None:
        q = 2 * norm.cdf(nsig) - 1
    else:
        raise ValueError('One of `q` and `nsig` should be specified.')
    r2 = chi2.ppf(q, 2)

    val, vec = np.linalg.eigh(cov)
    width, height = 2 * sqrt(val[:, None] * r2)
    rotation = np.degrees(arctan2(*vec[::-1, 0]))

    return width, height, rotation
```

The meaning of *standard deviation* is **wrong** in the answer of Joe Kington. Usually we use 1, 2 sigma for 68%, 95% confidence levels, but the 2 sigma ellipse in his answer does not contain 95% probability of the total distribution. The correct way is using a chi square distribution to esimate the ellipse size as shown in the post.

edited Sep 28 '16 at 14:09

answered Sep 28 '16 at 13:40

Syrtis Major
991 6 24

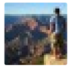
I guess what you are looking for is to compute the Confidence Regions.

I don't know much how about it, but as a starting point, I would check the sherpa application for python. At least, in their Scipy 2011 talk, authors mention that you can determine and obtain confidence regions with it (you may need to have a model for your data though).

See the video and corresponding slides of the Sherpa talk.

HTH

answered Sep 6 '12 at 23:25



gcalmettes
4,106 16 24

I also came along the sherpa-documentation, but I have actually no idea what this is :) – Raphael Roth Sep 7 '12 at 5:01