Week 9 Worksheet - Data Analysis

Barry Rowlingson

December 5, 2021

For this workshop we'll be doing some more work with the raised incidence model introduced in the last workshop. Make sure you're familiar with that before proceeding.

Work in a Jupyter Lab notebook but import from the supplied `raised_incidence.py` file. Run the following Jupyter Lab "magic" commands in an initial code chunk so that imported modules get automatically reloaded when the are used after a change in the file.

```
%load_ext autoreload
%autoreload 2
```

Edit or add functions to the `raised_incidence.py` file and call them from your notebook. If you get stuck, model answer functions are in `ri_answers.py`. Either copy into your `raised_incidence.py` file or do `import ri_answers as ri` at the top of your notebook to switch to that file.

1 Model

In summary, the probability of one of the data points at distance d from the putative source of raised incidence being a case instead of a control is given by

$$p_1(d; \alpha, \beta, \rho) = \rho f/(1 + \rho f)$$

where

$$f(d, \alpha, \beta) = 1 + \alpha \exp(-(d/\beta)^2)$$

and α, β, and γ and parameters.

2 Probability

Check you can import the case probability function and replicate these results. For brevity, import the raised incidence file as `ri`.

```
import raised_incidence as ri
```

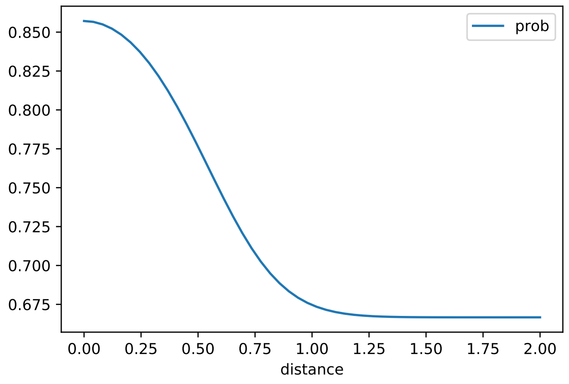
Now check these probabilities agree with yours:

```
import numpy as np
d = np.linspace(0, 2, 50)
print(ri.p(d, alpha=2, beta=0.5, rho=2))
```

```
[0.85714286 0.85659861 0.85496287 0.85222735 0.84838057 0.84341142
 0.83731441 0.83009636 0.82178463 0.81243613 0.80214627 0.7910561
 0.77935583 0.76728252 0.75511044 0.74313387 0.73164411 0.72090419
 0.7111259  0.70245376 0.69495852 0.68864055 0.6834413  0.67925936
 0.67596769 0.6734292  0.67150902 0.67008293 0.66904211 0.66829506
 0.66776744 0.66740057 0.66714932 0.66697981 0.66686711 0.66679325
 0.66674555 0.66671517 0.66669609 0.66668428 0.66667707 0.66667273
 0.66667016 0.66666865 0.66666778 0.66666728 0.666667   0.66666685
 0.66666676 0.66666672]
```

Check that this plot of probability against distance looks the same for you too.

```
import pandas as pd
# define a distance axis from 0 to 2:
d = np.linspace(0, 2, 50)
# make a data frame with the distance and probabilty
dist_prob = pd.DataFrame(dict(distance=d, prob=ri.p(d, alpha=2, beta=0.5, rho=2)))
# plot it
p = dist_prob.plot(x='distance',y='prob')
```



3 Data

The file `incidence.csv` contains the case and control data from the paper. Let's read this into a pandas data frame:

```
ll_study = pd.read_csv("casecontrol.csv")
ll_study
```

```
          x       y  cc
0    359014  416976   0
1    352909  426935   0
2    353848  422172   0
3    359202  417326   0
4    357795  415825   0
..      ...     ...  ..
969  356340  413295   1
970  355903  413619   1
971  355563  414116   1
972  355398  414390   1
973  355350  414031   1

[974 rows x 3 columns]
```

The columns here are the coordinates of the points and a case-control flag, which is one for a case of interest - here larynx cancer - and zero for a control point, here a case of lung cancer selected randomly from all lung cancer cases in the area. For the justification of this selection see the paper.

Now try some basic Pandas functionality:

```
ll_study.shape # rows, cols
```

```
(974, 3)
```

```
ll_study[:5] # first 5 rows
```

```
        x       y  cc
0  359014  416976   0
1  352909  426935   0
2  353848  422172   0
3  359202  417326   0
4  357795  415825   0
```

```
ll_study["cc"].describe() # simple summary
```

```
count    974.000000
mean       0.058522
std        0.234848
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: cc, dtype: float64
```

```
ll_study.groupby("cc").cc.count() # tabulate controls/cases
```

```
cc
0    917
1     57
Name: cc, dtype: int64
```

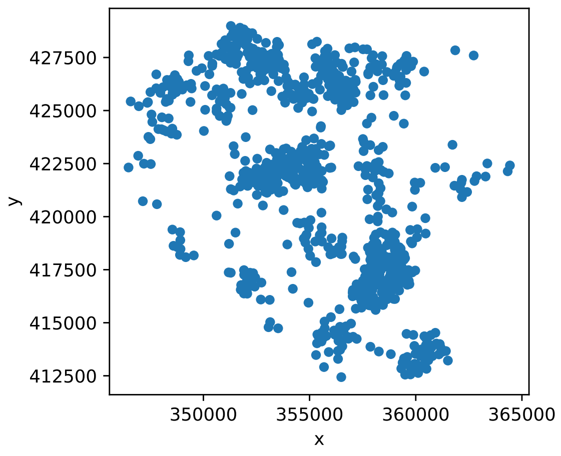
```
ll_study.groupby("cc").mean() # mean X and Y coord for cases and controls
```

```
                x              y
cc
0   355566.651036  421514.501636
1   354869.245614  421578.245614
```

4 Map

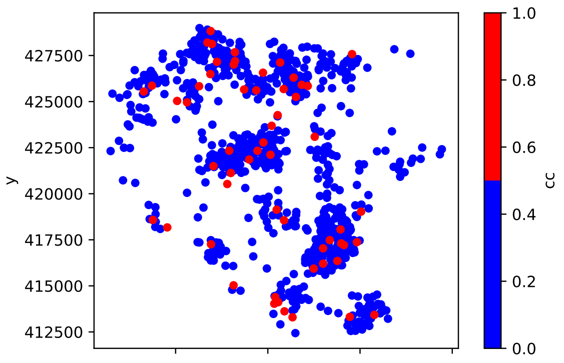
We can plot these points as a map. The coordinates here are in metres on a regular grid (not latitude and longitude) so we have to make sure `matplotlib` doesn't stretch the plot to fit its graphics area. We do this by setting the *aspect ratio* of the plot to 1.

```
p = ll_study.plot.scatter(x="x",y="y")
p.set_aspect(1.0)
```



We can also plot the cases and controls in different colours using a custom colour map.

```
import matplotlib
cmap = matplotlib.colors.ListedColormap(["blue", "red"])
p = ll_study.plot.scatter(x="x",y="y",c="cc", colormap=cmap)
p.set_aspect(1.0)
```



Put this into a function so you can call it on any data set. Think about what other arguments you might want to include in order to customise the plotting. A blank function is included in the Python file.

```
ri.map(d)
```

```
Make a map here!
```

5 Incinerator Location

The locations of three possible sources of raised incidence are included in a YAML file. Install the `pyyaml` package using `pip` or your Python package managment system. Then we can read in the data.

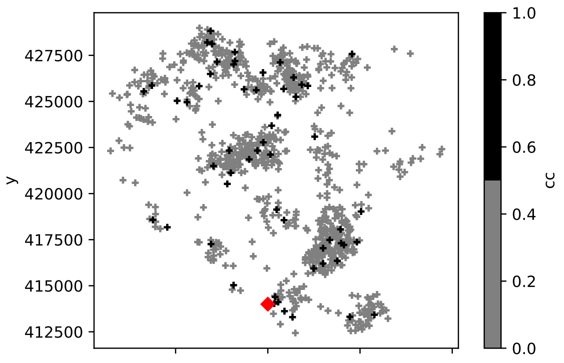
```
import yaml
incinerators = yaml.safe_load(open("incinerator.yml"))
incinerators
```

```
{'incinerators': {'a1': {'x': 355000, 'y': 414000},
  'a2': {'x': 349632, 'y': 424577},
  'a3': {'x': 355641, 'y': 422522}}}
```

There are three points in this YAML file (the first one is the actual incinerator, the other two are test points).

Modify your map function to also include the location of an incinerator. You need to pass the incinerator location and use the plot method to mark it on the map. You can use the `marker='x'` option to change the plotting symbol. You might want to change the plotting symbols and colours for the cases and controls too.

```
incA1 = incinerators['incinerators']['a1']
ri.map(ll_study, incA1)
```



6 Data Manipulation

To fit the model we need to work out the log-likelihood for the data given a set of parameters. To do that we need the distance from each point to the incinerator.

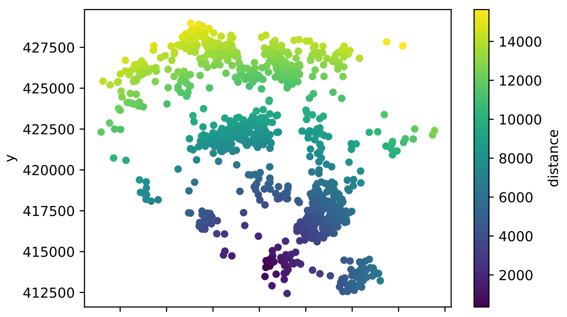
Write a function called `dist` (in your Python `raised_incidence.py` file), that takes the data frame with the case-control data and returns the distance to the incinerator. Do this by Pythagoras' theorem. Compute the difference between the X coordinate of the points and the X coordinate of the incinerator, and square it. Do the same for the Y coordinates. Add the squares and take the square root using `np.sqrt`. This should create a Pandas "Series" object which can be added to a Pandas DataFrame.

Add the distance to the data frame as a new column:

```
ll_study['distance'] = ri.dist(ll_study, incA1)
```

As a check you can now make a simple scatter plot coloured by distance.

```
dplot = ll_study.plot.scatter(x="x",y="y", c="distance", colormap="viridis")
```



7 Log-Likelihood for the Model

Now we have the distance and the case-control 1/0 value for our data we can compute the case probability for each event using our `p(.)` function. Recall that this is the probability of a *case* at a distance, and we have cases *and controls*. To compute the likelihood of the data given the parameters we have to multiply $p(d_i; \theta)$ for all cases and $1 - p(d_i; \theta)$ for all controls, since each point is either a case or control, hence the probability of a case is one minus the probability of a control. The likelihood is the product of all these probabilities, but for computational reasons statisticians prefer the log-likelihood, computing the log of the product as the sum of the log of the probabilities.

So our log-likelihood function should have distance and case-control parameters as well as parameters for alpha, beta, and rho. It should then compute the case probability using the `p(.)` function. Then it should, for the controls, subtract that `p(.)` from 1. Then it should take the log of those values and sum them.

As a test, the log-likelihood should evaluate at (an arbitrary set) of $\alpha = 3$, $\beta = 300$, $\rho = 1$ as:

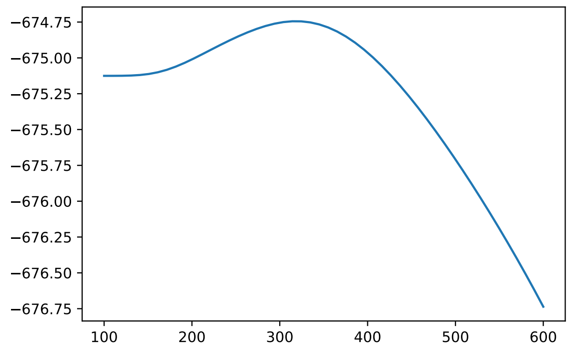
```
ri.loglik(ll_study.distance, ll_study.cc, alpha=3, beta=300, rho=1)
```

```
-674.872281563321
```

Note that in the formula for $p(.)$ the β parameter is a distance scale (it divides the distance to get dimensionless units for the exponential). Hence the 300 value above corresponds to a 300 metre distance.

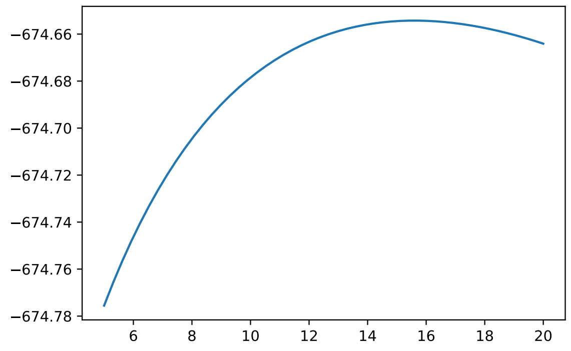
To fit the model requires finding the values of the three parameters that maximises the log-likelihood, and that gives us estimates and standard errors on the estimates, giving us the ability to make inference about the parameters. For now however, we'll just evaluate the log-likelihood for fixed `alpha` and `rho` and vary beta between 100 and 600. This code should then get this plot:

```
import matplotlib.pyplot as plt
betas = np.linspace(100,600,50)
LLbeta = [ri.loglik(ll_study.distance, ll_study.cc, alpha=6, beta=beta, rho=1) for beta in betas]
p = plt.plot(betas, LLbeta)
```



This shows a maximum at around $\beta = 310$. We could then vary α and find a maximum with β fixed at 310:

```
beta = 310
alphas=np.linspace(5, 20, 50)
rho=1
LLalpha = [ri.loglik(ll_study.distance, ll_study.cc, alpha=alpha, beta=beta, rho=1) for alpha in alphas]
p = plt.plot(alphas, LLalpha)
```



This shows the initial guess of $\alpha = 6$ was way off, and maybe $\alpha = 15$ is better, but its likely that the optimum β is no longer 310. And we've not yet considered the ρ parameter. So this isn't a practical method for finding the parameters that maximise the likelihood. Luckily there are optimisation algorithms that can do a good job of this, and that's for later!

8 Further Reading

1. NumPy User Guide
2. Pandas: Getting Started
3. Pandas For R Users
4. Tom Augsberger's series on Modern Pandas