challenger

October 14, 2018

1 Risk Analysis of the Space Shuttle: Pre-Challenger Prediction of Failure

In this document we reperform some of the analysis provided in *Risk Analysis of the Space Shuttle: Pre-Challenger Prediction of Failure* by *Siddhartha R. Dalal, Edward B. Fowlkes, Bruce Hoadley* published in *Journal of the American Statistical Association*, Vol. 84, No. 408 (Dec., 1989), pp. 945-957 and available at http://www.jstor.org/stable/2290069.

On the fourth page of this article, they indicate that the maximum likelihood estimates of the logistic regression using only temperature are: $\hat{\alpha}$ = **5.085** and $\hat{\beta}$ = **-0.1156** and their asymptotic standard errors are $s_{\hat{\alpha}}$ = **3.052** and $s_{\hat{\beta}}$ = **0.047**. The Goodness of fit indicated for this model was G^2 = **18.086** with **21** degrees of freedom. Our goal is to reproduce the computation behind these values and the Figure 4 of this article, possibly in a nicer looking way.

1.1 Technical information on the computer on which the analysis is run

We will be using the python3 language using the pandas, statsmodels, and numpy library.

```
In [8]: def print_imported_modules():
            import sys
            for name, val in sorted(sys.modules.items()):
                if(hasattr(val, '__version__')):
                    print(val.__name__, val.__version__)
        #        else:
        #            print(val.__name__, "(unknown version)")
        def print_sys_info():
            import sys
            import platform
            print(sys.version)
            print(platform.uname())

        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import statsmodels.api as sm
```

```
        print_sys_info()
        print_imported_modules()
```

```
3.6.5rc1 (default, Mar 14 2018, 06:54:23)
[GCC 7.3.0]
uname_result(system='Linux', node='icarus', release='4.15.0-2-amd64', version='#1 SMP Debian 4

IPython 5.5.0
IPython.core.release 5.5.0
PIL 4.3.0
PIL.version 4.3.0
_csv 1.0
_ctypes 1.1.0
_curses b'2.2'
decimal 1.70
argparse 1.1
csv 1.0
ctypes 1.1.0
cvxopt 1.1.9
cycler 0.10.0
dateutil 2.7.3
decimal 1.70
decorator 4.3.0
distutils 3.6.5rc1
ipaddress 1.0
ipykernel 4.8.2
ipykernel._version 4.8.2
ipython_genutils 0.2.0
ipython_genutils._version 0.2.0
ipywidgets 6.0.0
ipywidgets._version 6.0.0
joblib 0.11
json 2.0.9
jupyter_client 5.2.3
jupyter_client._version 5.2.3
jupyter_core 4.4.0
jupyter_core.version 4.4.0
logging 0.5.1.2
matplotlib 2.1.1
matplotlib.backends.backend_agg 2.1.1
matplotlib.pylab 1.14.5
numexpr 2.6.5
numpy 1.14.5
numpy.core 1.14.5
numpy.core.multiarray 3.1
numpy.core.umath b'0.4.0'
numpy.lib 1.14.5
numpy.linalg._umath_linalg b'0.1.5'
```

```
numpy.matlib 1.14.5
optparse 1.5.3
pandas 0.22.0
_libjson 1.33
patsy 0.5.0
patsy.version 0.5.0
pexpect 4.2.1
pickleshare 0.7.4
pkg_resources._vendor.packaging.__about__ 16.8
pkg_resources._vendor.six 1.10.0
pkg_resources._vendor.appdirs 1.4.0
pkg_resources._vendor.packaging 16.8
pkg_resources._vendor.pyparsing 2.1.10
pkg_resources._vendor.six 1.10.0
platform 1.0.8
prompt_toolkit 1.0.15
ptyprocess 0.5.2
py 1.5.3
py._vendored_packages.apipkg 1.4
pytest 3.3.2
pygments 2.2.0
pyparsing 2.2.0
pytz 2018.5
re 2.2.1
scipy 0.19.1
scipy._lib.decorator 4.3.0
scipy._lib.six 1.2.0
scipy.fftpack 0.4.3
scipy.fftpack._fftpack b'$Revision: $'
scipy.fftpack.convolve b'$Revision: $'
scipy.integrate._dop b'$Revision: $'
scipy.integrate._ode $Id$
scipy.integrate._odepack  1.9
scipy.integrate._quadpack  1.13
scipy.integrate.lsoda b'$Revision: $'
scipy.integrate.vode b'$Revision: $'
scipy.interpolate._fitpack  1.7
scipy.interpolate.dfitpack b'$Revision: $'
scipy.linalg 0.4.9
scipy.linalg._fblas b'$Revision: $'
scipy.linalg._flapack b'$Revision: $'
scipy.linalg._flinalg b'$Revision: $'
scipy.ndimage 2.0
scipy.optimize._cobyla b'$Revision: $'
scipy.optimize._lbfgsb b'$Revision: $'
scipy.optimize._minpack  1.10
scipy.optimize._nnls b'$Revision: $'
scipy.optimize._slsqp b'$Revision: $'
```

```
scipy.optimize.minpack2 b'$Revision: $'
scipy.signal.spline 0.2
scipy.sparse.linalg.eigen.arpack._arpack b'$Revision: $'
scipy.sparse.linalg.isolve._iterative b'$Revision: $'
scipy.special.specfun b'$Revision: $'
scipy.stats.mvn b'$Revision: $'
scipy.stats.statlib b'$Revision: $'
simplejson 3.15.0
six 1.11.0
statsmodels 0.9.0
statsmodels.__init__ 0.9.0
traitlets 4.3.2
traitlets._version 4.3.2
urllib.request 3.6
zlib 1.0
zmq 17.0.0
zmq.sugar 17.0.0
zmq.sugar.version 17.0.0
```

1.2 Loading and inspecting data

Let's start by reading data.

```
In [2]: data = pd.read_csv("https://app-learninglab.inria.fr/gitlab/moocrr-session1/moocrr-rep
        data
```

Out[2]:

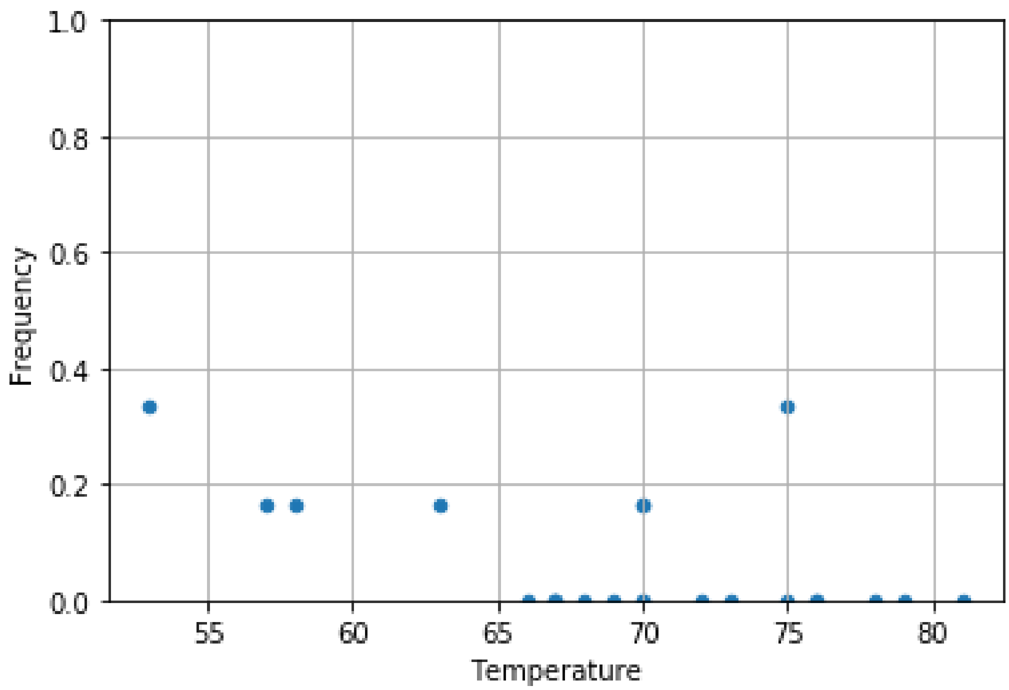
	Date	Count	Temperature	Pressure	Malfunction
0	4/12/81	6	66	50	0
1	11/12/81	6	70	50	1
2	3/22/82	6	69	50	0
3	11/11/82	6	68	50	0
4	4/04/83	6	67	50	0
5	6/18/82	6	72	50	0
6	8/30/83	6	73	100	0
7	11/28/83	6	70	100	0
8	2/03/84	6	57	200	1
9	4/06/84	6	63	200	1
10	8/30/84	6	70	200	1
11	10/05/84	6	78	200	0
12	11/08/84	6	67	200	0
13	1/24/85	6	53	200	2
14	4/12/85	6	67	200	0
15	4/29/85	6	75	200	0
16	6/17/85	6	70	200	0
17	7/2903/85	6	81	200	0
18	8/27/85	6	76	200	0
19	10/03/85	6	79	200	0

20	10/30/85	6	75	200	2
21	11/26/85	6	76	200	0
22	1/12/86	6	58	200	1

We know from our previous experience on this data set that filtering data is a really bad idea. We will therefore process it as such.

```
In [3]: %matplotlib inline
        pd.set_option('mode.chained_assignment',None) # this removes a useless warning from pa
        import matplotlib.pyplot as plt

        data["Frequency"]=data.Malfunction/data.Count
        data.plot(x="Temperature",y="Frequency",kind="scatter",ylim=[0,1])
        plt.grid(True)
```



1.3 Logistic regression

Let's assume O-rings independently fail with the same probability which solely depends on temperature. A logistic regression should allow us to estimate the influence of temperature.

```
In [4]: import statsmodels.api as sm

        data["Success"]=data.Count-data.Malfunction
        data["Intercept"]=1
```

```
logmodel=sm.GLM(data['Frequency'], data[['Intercept','Temperature']], family=sm.famili

logmodel.summary()
```

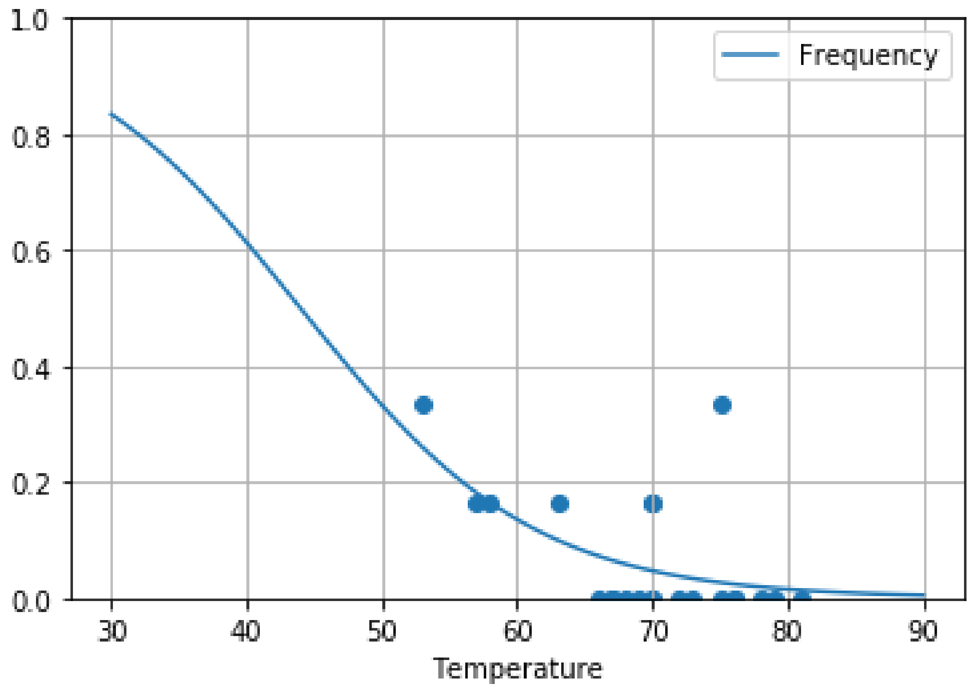
```
Out[4]: <class 'statsmodels.iolib.summary.Summary'>
"""
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              Frequency   No. Observations:                   23
Model:                            GLM   Df Residuals:                       21
Model Family:                Binomial   Df Model:                            1
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -3.9210
Date:                Sun, 23 Sep 2018   Deviance:                       3.0144
Time:                        22:50:48   Pearson chi2:                     5.00
No. Iterations:                     6   Covariance Type:             nonrobust
===============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept       5.0850      7.477      0.680      0.496      -9.570      19.740
Temperature    -0.1156      0.115     -1.004      0.316      -0.341       0.110
===============================================================================
"""
```

The maximum likelyhood estimator of the intercept and of Temperature are thus $\hat{\alpha}$ = **5.0849** and $\hat{\beta}$ = **-0.1156**. This **corresponds** to the values from the article of Dalal *et al.* The standard errors are $s_{\hat{\alpha}}$ = **7.477** and $s_{\hat{\beta}}$ = **0.115**, which is **different** from the ***3.052*** and ***0.04702*** reported by Dallal *et al.* The deviance is **3.01444** with **21** degrees of freedom. I cannot find any value similar to the Goodness of fit (G^2 = ***18.086***) reported by Dalal *et al.* **I have therefore managed to partially replicate the results of the Dalal *et al.* article**.

1.4 Predicting failure probability

The temperature when launching the shuttle was 31řF. Let's try to estimate the failure probability for such temperature using our model:

```
In [5]: %matplotlib inline
        data_pred = pd.DataFrame({'Temperature': np.linspace(start=30, stop=90, num=121), 'Inte
        data_pred['Frequency'] = logmodel.predict(data_pred)
        data_pred.plot(x="Temperature",y="Frequency",kind="line",ylim=[0,1])
        plt.scatter(x=data["Temperature"],y=data["Frequency"])
        plt.grid(True)
```



This figure is very similar to the Figure 4 of Dalal *et al.* **I have managed to replicate the Figure 4 of the Dalal *et al.* article.**

No confidence region was given in the original article. I have tried to compute and draw the confidence region in python but I haven't found how to do so. **I have failed so far to obtain the confidence region**. Here are my attempts

```
In [6]: # Inspiring from http://blog.yhat.com/posts/logistic-regression-and-python.html
        # odds ratios and 95% CI
        params = logmodel.params
        conf = logmodel.conf_int()
        conf['OR'] = params
        conf.columns = ['low', 'up', 'OR']

        #conf.low.Temperature = conf.OR.Temperature-2*0.047 ## I know my previous estimates of
        #conf.up.Temperature = conf.OR.Temperature+2*0.047
        #conf.low.Intercept = conf.OR.Intercept-2*3.052
        #conf.up.Intercept = conf.OR.Intercept+2*3.052

        print(conf)
        def logit_inv(x):
            return(np.exp(x)/(np.exp(x)+1))

        data_pred['Prob']=logit_inv(data_pred['Temperature'] * conf.OR.Temperature + conf.OR.In

        # mean_temp = np.mean(data.Temperature)
```

```
# mean_prob_logit = mean_temp * conf.OR.Temperature + conf.OR.Intercept
# #               (np.power((data_pred.Temperature-mean_temp),2) /
# #                ((np.sum(np.power(data_pred.Temperature,2))) - n*(np.power(mean_temp,

data_pred['Prob_low']=logit_inv(np.minimum(
    data_pred['Temperature'] * conf.low.Temperature + conf.low.Intercept/2,
    data_pred['Temperature'] * conf.up.Temperature + conf.low.Intercept/2))
data_pred['Prob_up']=logit_inv(np.maximum(
    data_pred['Temperature'] * conf.low.Temperature + conf.up.Intercept/2,
    data_pred['Temperature'] * conf.up.Temperature + conf.up.Intercept/2))
```

```
                  low         up        OR
Intercept   -9.569730  19.739685  5.084977
Temperature -0.341358   0.110156 -0.115601
```

In [7]:
```
%matplotlib inline
## http://markthegraph.blogspot.com/2015/05/using-python-statsmodels-for-ols-linear.htm
data_pred.plot(x="Temperature",y="Prob",kind="line",ylim=[0,1])
plt.fill_between(data_pred.Temperature,data_pred.Prob_low,data_pred.Prob_up,color='#888
plt.scatter(x=data["Temperature"],y=data["Frequency"])
plt.grid(True)
```