Robust statistics: A brief overview

Vishwanath, Saragadam Raja Venkata

May 7, 2015

Abstract

We look at the need for robust statistics as a powerful tool to deal with outliers in experimental setup. In particular, we emphasize on the need for robust statistics, methods to quantify robustness and some methods of obtaining robust statistics.

1 Introduction

Modeling of physical processes is at most a good approximation of the actual underlying process. Hence, any uncertainties in the model may lead to gross errors in estimating parameters of the process. For example, consider the sample mean, $\mu = \frac{\sum_{i=1}^{n} x_i}{n}$. Even if just one of the value is very large, the estimate of μ is completely wrong.

Ideally, we need statistics that would overlook the effect of points which don't seem to fit the model, which we call the outliers. The goal of robust statistics is to answer the issue of outliers in experimental data. Robust statistics as a tool helps in identifying the right parameter estimate which describes majority of the data. Further, it also helps to identify the effect of individual points, and hence the outliers.

To identify a statistic as robust, we want the following properties:

1. The statistic has to be efficient. With sufficiently large data, the statistic should converge to true parameter value.

2. The statistic has to be stable. Small deviations from the model assumption shouldn't give wrong estimates.

3. The statistic has to have large breakdown point. A large deviation shouldn't create a catastrophe.

The report is organized as follows. We first discuss the theory which will aide the understanding of robust statistic. Following that, we will look at some ad-hoc robust statistics. We will then formalize a robust statistic using the M-estimator method. Then we discuss about robust statistical modeling using student-t distribution. We finally conclude by showing toy examples to illustrate various robust estimators.

2 Robustness: Theory

We look at some theory which will aide in understanding the following sections.

2.1 Sensitivity curve and influence function

Let $T_n(\{x_i\}_{i=1}^n)$ be a statistic. Then the sensitivity curve is defined as,

$$SC(x,T) = \lim_{n\to\infty} n[T_n(x_1,\ldots,x_{n-1},x) - T_{n-1}(x_1,\ldots,x_{n-1})] \tag{1}$$

. The sensitivity curve quantifies the effect of an individual point on the estimate. The population version of the sensitivity curve is the influence function, which is defined as follows. Let F be a distribution and $F_\epsilon = (1-\epsilon)F + \epsilon\delta_x$ be the contaminated distribution. Then,

$$IF(x,T,F) = \lim_{\epsilon\to 0} \frac{T(F_\epsilon) - T(F)}{\epsilon} = \frac{\partial}{\partial\epsilon} T(F_\epsilon)\bigg|_{\epsilon=0} \tag{2}$$

, where T is a statistic on F. For example, the influence function for sample mean is $IF(x, mean, F) = x - \bar{\mu}$. Hence, if $x \to \infty$, then $\mu \to \infty$. Robust statistic needs that we have a bounded influence function.

2.2 Breakdown point

Let us define the bias of an estimator T_n as,

$$bias(m, T_n, X) = \sup_{x'} \|T_n(X') - T_n(X)\| \tag{3}$$

, where X' is X with m points replaced by corrupted points.Then, the breakdown point,

$$\epsilon_n^*(T_n, X) = \min\{\frac{m}{n} : bias(m, T_n, X) = \infty\} \tag{4}$$

. For example, for sample mean, $\epsilon_n^* = \frac{1}{n}$, since one rogue sample is enough to take the estimate to ∞.

The theory discussed in section 2, we would want a statistic with bounded influence function, which would ensure that outliers won't affect the statistic by a large margin, and also a high break down point, meaning that we need a large number of corrupted points to give a bad estimate.

3 Some robust estimators

3.1 Heuristic methods

If we assume that data is indeed from a normal process, and that outliers are due to some experimental error, then we can drop the tail end of the data and recalculate the mean. If we drop α fraction of the tail, it is known as α-trimmed mean. Similarly, we can instead replace the α fraction of the tail end with the next nearest distribution instead, which gives the α-winsorized mean.

For both the above mentioned methods, the break down point, $\epsilon_n^* = \alpha$. However, if there are more than α fraction of bad points, then the mean can go to ∞, hence the influence function is not bounded.

3.2 M-estimators

Given data $\{x_i\}_{i=1}^n$ and statistic T_n. Assume that we wish to minimize the following function:

$$\min_{T_n} \sum_{i=1}^{n} \rho(x_i; T_n) \tag{5}$$

This is called an M-estimator, from Maximum likelihood type estimator. Differentiating w.r.t T_n,

$$\sum_{i=1}^{n} \psi(x_i; T_n) = 0 \tag{6}$$

Eg, if $\rho(x_i; T_n) = \frac{1}{2}(x_i - T_n)^2, \psi(x_i; T_n) = (x_i - T_n)$, which gives sample mean. If $\rho(x_i; T_n) = |x_i - T_n|, \psi(x_i; T_n) = sign(x_i - T_n)$, then the statistic gives median.

For a reliable estimate, we want a function that heavily penalizes points which are close to the actual statistic, but relaxes on the points that are very far away. One such popular function is the Huber loss function, given by,

$$h_k(x) = \begin{cases} \frac{1}{2}x^2 & |x| < k \\ k(|x| - \frac{1}{2}k) & |x| > k \end{cases} \tag{7}$$

. For points near to center, the ψ function is proportional to the difference, but for far away points, it's constant, which removes the effect of outliers. For appropriately chosen k, an M-estimator can have a very high breakdown point of $\epsilon_n^* = 0.5$

3.3 Robust statistics from distribution

Though the M-estimator is a popular robust estimator, it is parametrized, and not an outcome of a PDF. Presence of outliers means that there is high probability for values at the tail end, which requires a heavy tail distribution. The first distribution that comes to the mind is a cauchy distribution. However, if there are hardly any outliers, then cauchy distribution fails.

A Student-t distribution[2], on the other hand gives control over the heavy tailed-ness of the distribution(Figure 1). The student-t distribution with ν degrees of freedom is given by

$$f_X(x) = \frac{(1 + \frac{u}{\nu})^{-\frac{\nu+1}{2}}}{\sqrt{\nu} B(\frac{1}{2}, \frac{\nu}{2})} \tag{8}$$

Given the data $\{x_i\}_{i=1}^n$, with true center c and true variance s, we can write the log-likelihood function as

$$L(c, s) = -\frac{\nu + 1}{2} \sum_{i=1}^{n} \log(1 + \frac{u_i^2}{\nu}) - n \log(s\sqrt{\nu} B(\frac{1}{2}, \frac{\nu}{2})) \tag{9}$$

, where, $u_i = \frac{x_i - c}{s}$. We can now maximize $L(c, s)$ to obtain an MLE estimate of c and s. While the function is non-convex, an alternating maximization step is known to give a good estimate. The number of degrees of freedom, ν, is chosen to give maximum likelihood.

Observe, that differentiating eq. 9 w.r.t c, we get,

$$\frac{\partial L}{\partial c} = 0 \implies \frac{\nu + 1}{s} \sum_{i=1}^{n} \frac{u_i}{\nu + u_i^2} = 0 \tag{10}$$

$$\equiv \sum_{i=1}^{n} \psi(x_i; \theta) = 0$$

The student-t distribution is free of parameters, is similar to an M-estimator, and is a more intuitive way of approaching a robust estimator. Figures 2, 3, 4, 5 give an idea of the robustness of various estimators.

4 Conclusion

We looked at an introduction to the robust statistics and gave a motivation from the sample mean. The theory section helped quantifying robustness. We also looked at some types of robust estimators and showed some simulation results.

5 References

References

[1] *Robust statistics: a brief introduction and overview.*

[2] D R Divgi, *Robust estimation using student's t distribution*, (1990).

[3] Peter J Huber, *Robust statistics*, Springer, 2011.

[4] Elvezio Ronchetti, *Introduction to robust statistics.*

[5] David E Tyler, *A short course on robust statistics.*

6 Appendix A: Figures

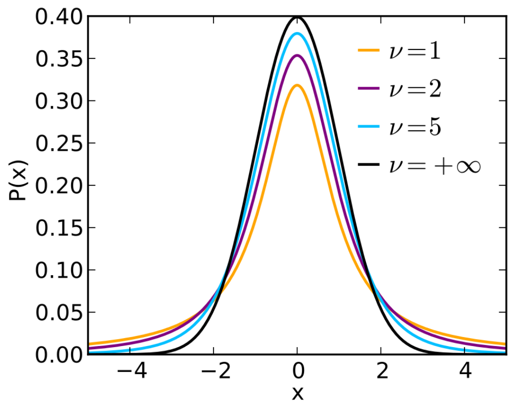

Figure 1: Image courtesy: Wikipedia.

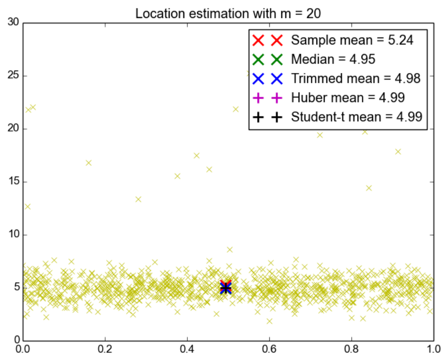

Figure 2: Location estimation with 20 outliers out of 1000

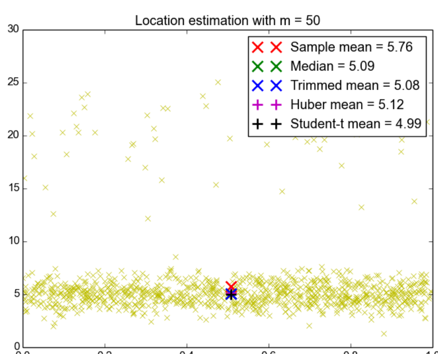

Figure 3: Location estimation with 50 outliers out of 1000

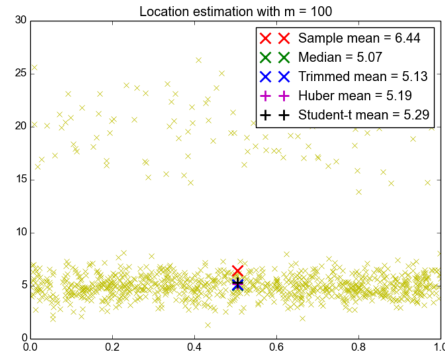

Figure 4: Location estimation with 100 outliers out of 1000

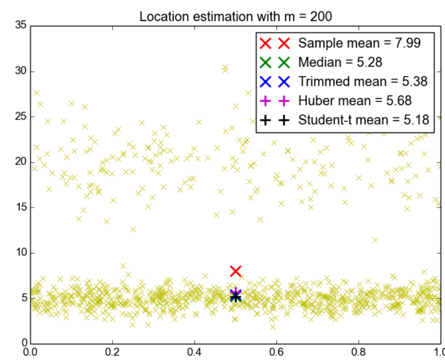

Figure 5: Location estimation with 200 outliers out of 1000

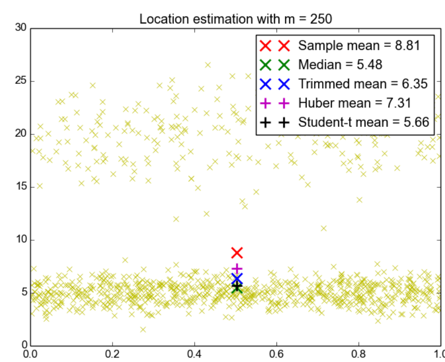

Figure 6: Location estimation with 250 outliers out of 1000

7 Appendix B: Code for simulation

```
#!/usr/bin/env python

from numpy import *
import scipy.stats as st
import statsmodels.api as sm
import scipy.special as sp
from matplotlib.pyplot import *

# Global constants
n = 1000;
#m_array = range(0, n, 10);
m_array = [20, 50, 100, 150, 200, 250];

'''
We want to execute the following estimation algorithms:
    1. Linear least squares.
    2. Median.
    3. \alpha-trimmed mean.
    4. M-estimator with Huber loss function.
    5. Student-t distribution.
'''

class student_t(object):
    '''
        Minimizer class for robust location and scale estimation using
        student-t distribution
    '''
    def __init__(self, data):
        self.data = data
        niters = 10000

        # Initialization constants
        v_array = arange(1, 10)

        c_array = zeros(len(v_array))
        s_array = zeros(len(v_array))
        ll_array = zeros(len(v_array))

        for idx in range(len(v_array)):
            c = median(data)
            s = 1.0
            v = v_array[idx]

            t = 0.01

            # Start gradient ascent
            f_new = self.fval(v, c, s)
            f_old = -float('inf')
            iters = 0
            while abs(f_new - f_old) > 1e-6 and iters < niters:
                del_c, del_s = self.grad(v, c, s)
                c = c + t*del_c
```

```python
                s = s + t*del_s

                f_old = f_new
                f_new = self.fval(v, c, s)
                iters += 1

            c_array[idx] = c
            s_array[idx] = s
            ll_array[idx] = self.fval(v, c, s)

        # Find the one with greatest log-likelihood
        ll_max = ll_array.max()
        c = c_array[where(ll_array == ll_max)]
        s = s_array[where(ll_array == ll_max)]

        # Debugging
        self.c_array = c_array
        self.s_array = s_array
        self.ll_array = ll_array

        self.c = c
        self.s = s
        self.v = v

    def fval(self, v, c, s):
        '''Function handle for the log likelihood function'''
        u = (self.data - c)/s
        ll = -0.5*(v+1)*sum(log(1+ pow(u, 2)/v)) -(
                n*log(s*sqrt(v)*sp.beta(0.5, float(v)/2)))
        return ll

    def grad(self, v, c, s):
        '''Gradient handle for the log likelihood function'''
        u = (self.data - c)/s
        n = len(self.data)
        grad_c = ((v+1)/s)*sum(u/(v + pow(u, 2)))
        grad_s = -n/s + ((v+1)/s)*sum(2*pow(u,2)/(v + pow(u,2)))

        return grad_c, grad_s

def ll_t(data, c, s, v):
    '''Function to return log-likelihood of the data with student's
    t distribution.
    '''
    u = (x - c)/v
    ll = -0.5*(v+1)*sum(log(1+ pow(u, 2)/v)) -n*log(s*sqrt(v)*sp.beta(0.5, v/2))
    return ll

# Generated data constants.
mu0 = 5
noise_sigma = 1.0
outlier_sigma = 3.0
outlier_mean = 15

# Place holders.
```

```
estim_m = zeros((1, len(m_array)))
estim_s = zeros((1, len(m_array)))

# Now start the estimation.
for idx in range(len(m_array)):
    m = m_array[idx];
    noise = random.randn(n-m)*noise_sigma;
    outliers = random.randn(m)*outlier_sigma + outlier_mean;
    data = np.hstack((noise, outliers)) + mu0;

    # Use this for spreading data.
    rand_data = np.random.rand(n)
    rand_data_center = mean(rand_data)

    # OLS estimation.
    ols_mu = mean(data);
    # Median estimate
    #median_mu = sm.robust.scale.mad(data);
    median_mu = median(data);
    # \alpha-trimmed mean
    atrim_mu = st.trim_mean(data, 0.2);
    # M-estimator with huber loss.
    huber_mu, huber_scale = sm.robust.scale.Huber(maxiter=1000)(data);
    # Student-t distribution.
    st_obj = student_t(data);

    # Print the estimates.
    print 'OLS: ', ols_mu
    print 'Median: ', median_mu
    print 'Trimmed mean: ', atrim_mu
    print 'Huber mean: ', huber_mu
    print 'Student-t mean: ', st_obj.c

    # Plot the estimate.
    figure()
    plot(rand_data, data, 'yx')
    plot(rand_data_center, ols_mu, 'rx',
            label='Sample mean = %.2f'%ols_mu, markersize=12, mew=2.0)
    plot(rand_data_center, median_mu, 'gx',
            label='Median = %.2f'%median_mu, markersize=12, mew=2.0)
    plot(rand_data_center, atrim_mu, 'bx',
            label='Trimmed mean = %.2f'%atrim_mu, markersize=12, mew=2.0)
    plot(rand_data_center, huber_mu, 'm+',
            label='Huber mean = %.2f'%huber_mu, markersize=12, mew=2.0)
    plot(rand_data_center, st_obj.c, 'k+',
            label = 'Student-t mean = %.2f'%st_obj.c, markersize=12, mew=2.0)
    title('Location estimation with m = %d'%m)
    legend()
    savefig('plot_m_%d.png'%m)
```