EECS 16B Designing Information Devices and Systems II
Fall 2021

Note 13A: Outlier Detection

Overview

We have just seen how to use least squares in order to identify the parameters of an unknown system. With enough measurements, least squares works well at identifying these parameters even in the presence of random noise that is small relative to the observations, but can fail when the noise is large relative to the observations.

Of course, when *all* our observations are influenced by large random noise, then we cannot reasonably expect to accurately recover the original system parameters. However, what happens if a *few* of our observations are corrupted by some large noise, but the rest are only *slightly* perturbed? This is not unrealistic - in system ID, for instance, it might be the case that a sensor was temporarily obstructed, and so produced entirely incorrect observations for a few time steps, known as *outliers*. In this note, we will develop techniques to perform system ID in such a scenario.

1 Outliers

Recall that system ID produces an overdetermined system of linear equations

$$A\vec{x} \approx \vec{b},$$

where $\vec{x}$ is the vector of parameters we wish to identify. Assume we have m equations and n unknowns, where $m > n$. To solve this system using least squares, we try to compute an $\vec{x}$ that minimizes the squared magnitude of the residual

$$(\vec{b} - A\vec{x})^2,$$

which we know from 16A has the closed form solution:

$$\vec{x} = (A^T A)^{-1} A^T \vec{b}.$$

Let's see how this approach performs when our observations are noisy, so no exact solution $\vec{x}$ where $A\vec{x} = \vec{b}$ can be found. For simplicity, we will focus on using least squares to fit a line $y = ax + b$ through a set of points

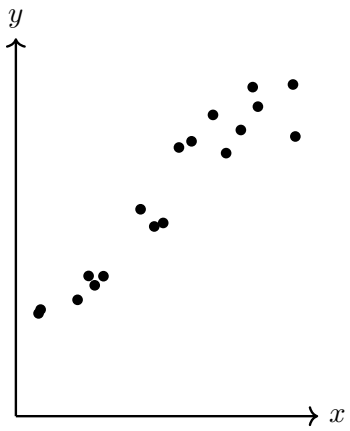
$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m),$$

so our linear system will look like

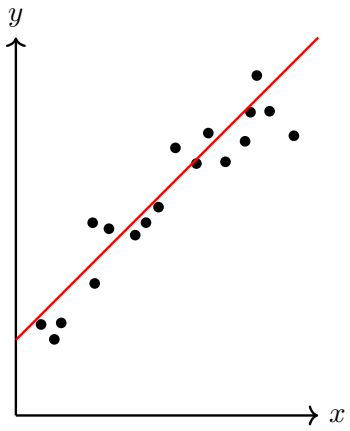
$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

with $n = 2$ unknowns to solve for.

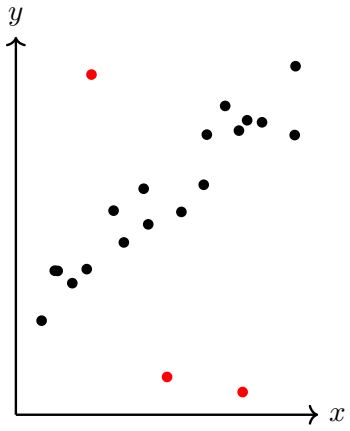
First, imagine that our points are only *slightly* off from a perfect fit, and so might look a little like this:



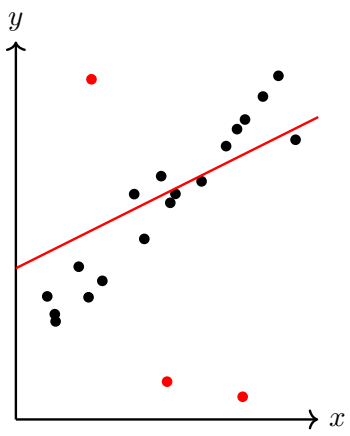
Then, we have previously seen how least squares will do a fairly good job of fitting a line to the data, as we show below:



However, what happens if a few of our data points are perturbed significantly, and so lie very far away from the linear model? This might look a little like the below figure:



Notice that the colored points deviate significantly from the rest of the dataset. If we try to perform least squares again on the whole dataset, we will see that these few outliers affect the results of our linear fit significantly, as shown:



Observe that the linear fit is being "skewed" to try and get closer to the outlier points. But since the values of the outlier points are essentially a consequence of errors in our observation process, and *not* reflective of our underlying system, in trying to better model the outliers our linear fit is in fact doing a worse job of estimating the system parameters!

2 Outlier Identification

We have seen that the presence of even a few outlier observations can affect the quality of a linear fit significantly. Ideally, what we'd like to do is *identify* these outliers, delete them from our dataset, and perform least squares on the remaining points.

To do so, let's expand out our previous matrix equation into a system of scalar linear equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\approx b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\approx b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\approx b_m. \end{aligned}$$

Right now, we can come up with an approximate solution to the above system of equations using least squares, without taking outliers into consideration. What if we were told that the first observation (i.e. the first equation) is an outlier, and so we should try to disregard it when solving for $\vec{x}$?

One way would be to delete the first equation entirely. Alternatively, imagine rewriting it as the following:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + f_1 \approx b_1,$$

where f_1 is not used in any other equation. Recall that least squares aims to minimize the sum of squares of the residuals across all equations in the system - in the case of the above equation, the residual would be

$$r_1 = b_1 - a_{11}x_1 - a_{12}x_2 - \cdots - a_{1n}x_n - f_1.$$

Notice that, regardless of what the $\{x_i\}$ are, least squares can choose f_1 to make $r_1 = 0$, without hurting the residuals of any other equation!

Thus, it is clear that including equation 1 with the f_1 term is equivalent to not including it at all, since least squares can always make its residual $r_1 = 0$ regardless of what the $\{x_i\}$ are without affecting the other residuals. In a similar manner, we can see[1] that adding f_i terms to all the equations that we wish to treat as outliers will effectively delete them from our least squares problem, since again their residuals can be driven to zero regardless of the $\{x_i\}$. Basically, whenever we add a set of columns that are targeted at only a few observations (in that they have zeros everywhere else), if they can span everything within the targeted observations, then these targeted observations have essentially been decoupled from the rest as far as least-squares is concerned.

So now, rather than asking "which of our equations should be deleted", we can ask "which of our equations should have an f_i term added?" What happens if we add f_i terms to *all* our equations? Then our system becomes as follows:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + f_1 &\approx b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + f_2 &\approx b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + f_m &\approx b_m. \end{aligned}$$

Rewriting it in block matrix form, we obtain

$$\begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} \vec{x} \\ \vec{f} \end{bmatrix} = \vec{b},$$

where

$$\vec{f} = \begin{bmatrix} f_1 & f_2 & \cdots & f_m \end{bmatrix}^T.$$

This linear system still has m equations, like before, but now has $m+n$ unknowns, since we have added the m new unknowns $\{f_i\}$ to the previous n unknowns $\{x_i\}$. Furthermore, it is clear that no matter what $\{x_i\}$ we choose, we can choose $\{f_i\}$ to satisfy *all* the equations exactly. So whereas before we had an overdetermined system of equations that we were trying to solve approximately, now we have an underdetermined system of equations with an infinite number of solutions!

What went wrong? Adding f_i terms to every single one of our equations and then solving doesn't seem to work. We probably should have expected this - after all, treating *all* our equations as outliers means that we have no non-outlier data left, so how were we expecting to solve for our system's parameters? Ideally, we'd like to treat *as few* equations as possible as outliers, and set the rest of the f_i to zero, meaning that those observations with $f_i = 0$ can still contribute towards estimating our system parameters.

Another way of looking at this is that we'd like to include as many zeros as possible in $\vec{f}$ while still obtaining a reasonably good solution for the system

$$\begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} \vec{x} \\ \vec{f} \end{bmatrix} \approx \vec{b}.$$

[1] Be aware, however, that these arguments are not yet fully rigorous. You will prove the correctness of this process rigorously in your homework.

Do we know any techniques for solving this problem?

3 Orthogonal Matching Pursuit

Yes, we do! From 16A, we saw that the technique of Orthogonal Matching Pursuit (OMP) does exactly what we want - given a system of equations, it finds a *sparse* candidate solution (i.e. a solution with few nonzero elements) that approximately solves the system. So what we can do is, given the system of equations

$$A\vec{x} \approx \vec{b},$$

write down the new system

$$\begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} \vec{x} \\ \vec{f} \end{bmatrix} \approx \vec{b},$$

and then solve for a sparse solution for $\begin{bmatrix} \vec{x} \\ \vec{f} \end{bmatrix}$ using OMP.

There's one subtlety to consider here. Observe that we really wanted a sparse solution for $\vec{f}$, but OMP will find a sparse solution for $\begin{bmatrix} \vec{x} \\ \vec{f} \end{bmatrix}$. Will this be a problem? We have a couple of choices. We could choose to initialize OMP with the columns corresponding to the original A already selected. In that case, the sparsity will only be sought within the $\vec{f}$ variables as desired.

As it turns out, even if we initialize with an empty set of selected columns, OMP will almost always include all of the first n columns in the final solution, so all the $\vec{x}$ parameters will be made nonzero. Thus, finding a sparse solution for $\begin{bmatrix} \vec{x} \\ \vec{f} \end{bmatrix}$ ends up being the same thing as finding a sparse solution for $\vec{f}$. If OMP does set some of the parameters in $\vec{x}$ to exactly zero, then we have to consider whether this choice is reasonable. Certainly if the actual parameter here was very small and had only a very tiny effect on the predictions for y, then indeed it is perfectly reasonable to have a very small number get snapped to zero in the estimated solution. The alternative case is that the parameter here is not very small and has a significant effect on predicting y. For this to happen, the input data has to be badly normalized. In general, we always want to pick the units for our model so that learned parameters come out to be moderately sized values. If the parameter values come out to be huge, it means that the corresponding column in A had to have been filled with tiny numbers — no wonder it couldn't win the OMP competition to be selected. Avoiding tiny feature columns is an aspect of real-world modeling and data cleaning that is important to do.

With properly normalized features, even vanilla OMP works pretty well in removing outliers while being able to learn true parameters.

4 Stopping Conditions

There's another detail to consider - when performing OMP, we need to know when to stop! Otherwise, OMP will just keep adding nonzero elements back into our vector of unknowns, and we'll end up back where we started, with an underdetermined system of equations. To figure out when to stop, there are a couple of things that we can look at.

First, we might be told approximately how many outliers there are in our data, so we know roughly how

many nonzero f_i terms there are. Thus, after we encounter that many terms, we know to stop OMP and return our candidate solution.

Alternatively, we can look at the residual generated by OMP at each step, and wait for that to drop below some predetermined value. Or we can even simply just make a guess as to when to stop OMP, and inspect the candidate solutions manually!

But what happens if we get it wrong? We've seen that if we have even a few outliers that the least squares solution ends up estimating our system parameters rather poorly. So we certainly don't want to stop OMP too early and leave outliers in our data.

What if we stop OMP a bit too late? After it finishes deleting outliers, it will start deleting valid points from our dataset, so if we stop OMP later than we should have then our least squares problem will have fewer equations that it should have. Will this be an issue? Not really. After all, even if we have slightly fewer equations than we should have, the overall trend in our dataset is still present and will be picked up by least squares, since at least all the outliers will be gone. The random noise will have a slightly greater impact since there are fewer observations remaining, but this tends not to be a big deal.

Thus, it's OK if we run OMP until we're *absolutely sure* that all the problematic outliers have been deleted, even if we end up losing a few valid data points as well along the way. This approach goes to show that automated approaches to outlier rejection generally rely on having plentiful good data in the mix.

Contributors:

- Rahul Arya.
- Anant Sahai.

