Inference, Models and Simulation for Complex Systems CSCI 7000-001
Lecture 4 6 September 2011
Prof. Aaron Clauset

1 Confronting models with data: a few comments

There are several distinct and rich literatures on the general topics of model plausibility and model comparison. As a result, there is no universally accepted prescription for how to test or compare models, especially models of complex systems. Instead, different fields tend to favor different approaches. Sometimes this is for principled reasons, although often it's best understood as simply a cultural tradition within that community. For instance, Bayesian approaches are widely used in machine learning and computational biology, while classic frequentist methods dominate the natural sciences. More recently, computer science has introduced approaches based on compression and minimum description lengths (MDL). Each of these major approaches to confronting mathematical models with empirical data are based on parametric approaches (that is, models with a finite number of parameters) that use a likelihood function to capture the agreement between model and data.

Not all approaches to confronting models with data demand a likelihood function. Non-likelihood-based techniques can be useful when, for instance, a model is too complicated to translate into a likelihood function[1] or if it is theoretically unreasonable to make strong iid assumptions about your empirical data. One of these alternatives uses goodness-of-fit statistics to make comparisons (like the KS statistic we encountered in Lecture 3); another uses model-free (non-parametric) operations like shuffling or sorting the data. In some cases, these methods, by making weaker assumptions about the generating processes, are less reliable than likelihood-based methods. As a result, likelihood-approaches are typically preferred, when available.

The goal of all such approaches is to answer two straightforward questions:

1. *Model plausibility*: Given some empirical data $\{x_i\}$ and a model M, is M a plausible explanation for $\{x_i\}$?
2. *Model comparison*: Given some empirical data $\{x_i\}$ and a set of models $\{M_j\}$, which model $M_k \in \{M_j\}$ is the best explanation for $\{x_i\}$?

To be precise in our answers to these questions, we require that candidate models be cast in precise terms; this typically means mathematical formula but it can also mean a computational proce-

[1]Remember that likelihood functions rely on assumptions of (potentially conditional) independence, and not all models can be cast in such terms. Additionally, some worthy hypotheses are difficult to formalization mathematically, which makes it difficult to cast them in terms of a likelihood function. A failure to be cast in mathematical terms does not lessen their value, but it does make it more difficult to know precisely what it is that the hypothesis states and thus whether or not the empirical data supports it.

dure. (The important thing is to be explicit and clear about all of your assumptions.) But, we must also specify what exactly we mean by the phrases *a plausible explanation* and *the best explanation* for some data. It is on answering these questions that frequentist, Bayesian, compression and goodness-of-fit approaches diverge; that is, they have different definitions of a *good* model.[2] And yet, all such approaches can be viewed as efforts to quantitatively wrestle with questions of *statistical power* (the probability of reaching a false conclusion), model simplicity (Occam's razor), and generalizability (out-of-sample prediction). The goal is to accurately identify and discard *bad* models while favoring models that generalize beyond the observations they were based on.

In some ways, the mathematical nature of many of these approaches has encouraged their misuse. That is, their quantitative nature has facilitated the erroneous conclusion that the hard problems of epistemology in science (How can we be confident that we've learned something true about the world from our data?) have been solved and that any question can be answered conclusively by mechanically applying the favored scientific-method algorithm. The problem is that each of these approaches come with assumptions about empirical data or models that are invariably violated by the real world. For instance, real data from complex systems are rarely iid or the *null hypothesis* or the alternative models may have been chosen poorly. When this happens, our tools may yield spurious or misleading results and, worse, they may not alert us of their failure! The best methods fail gracefully and conservatively when their assumptions aren't met, that is, they err on the side of saying "I don't know; need more data".

In modeling complex systems, where we must often rely on messy observational data rather than on clean, controlled experimental data,[3] statistical methods are indispensable for detecting important patterns. But, methods can only support, not replace, careful thinking, careful tests and better experiments. In this lecture, we'll investigate a few specific approaches to answering questions of model plausibility and comparison, and provide pointers into the literature on other approaches.

2 Model plausibility, p-values and statistical power

In testing a model of a complex system, the first question to answer is whether the model is a plausible explanation of the data. A useful approach to answering this question focuses on testing whether the observed differences between the data and the model can be explained as mere statistical fluctuations.

[2]Unfortunately, there has been too little work on trying to understand or resolve the differences between the different approaches, and too much effort spent bashing largely legitimate alternative statistical approaches.

[3]Most of the frequentist methods developed in statistics and widely used in the natural sciences were developed for analyzing experimental data where the goal of the experimenter was to design the experiment so that the assumptions of the statistical method were met, e.g., iid random variables.

This can be done using a classic statistical *hypothesis test*, but in a backwards way. In either direction, we choose a particular *null model* that we hypothesize generates our data. The forwards version aims to detect deviations so large that they are probably not attributable to noise (fluctuations) and thereby indicate the presence of "non random" effects. This approach underlies tests of normality, t-tests and their ilk, which are commonly used in controlled experiments. The backwards version aims to test whether the size of the observed deviations between the model and the data can plausibly be attributed to mere statistical fluctuations. That is, the goal is to determine if the observed pattern could *plausibly* be explained by the null model, except possible for random effects.

In either case, the test relies on a statistical measure of "closeness" $D = f(\{x_i\}, A)$ between the data $\{x_i\}$ and the model A. The Kolmogorov-Smirnov goodness-of-fit statistic is one such measure (see Lecture 3). To begin, we fit the model to the data by estimating any free parameters.[4] Then, we measure the closeness D for the empirical data (denoted D^*) relative to the fitted model, and ask what fraction of comparable synthetic data sets (that is, the same sample size) generated by A would yield a *larger* deviation, i.e., we want to compute $\Pr(D \geq D^*)$. Figure 1a illustrates this idea.

Here is pseudo code for doing the test. Let x be the data set, n be the number of observations it contains, N be a very large number and A be our model.

```
% -- get some data
{x_i}       = empirical data

% -- fit the model and measure its goodness-of-fit D
theta*      = maximum likelihood parameter estimate of A fitted to {x_i}
compute D*  = f({x_i},theta*)

% -- simulate distribution Pr(D) under null hypothesis
for i=1:N
    {y_i}        = n synthetic observations, drawn from A
    theta-s[i]   = maximum likelihood parameter estimate of A fitted to {y_i}
    compute D[i] = f({y_i},theta-s[i])
end

% -- compute p-value
p = sum(D>D*)/N
```

[4]Classic hypothesis tests typically skip this step because the model has fixed parameters, e.g., observations are drawn iid from a normal distribution with zero mean and unit variance. In this case, the Monte Carlo procedure is not necessary because the distribution of deviations can be derived analytically when the null hypothesis is true. This is the origin of the formulas you sometimes encounter in statistics textbooks, into which you plug the observed data and out pops a p-value.

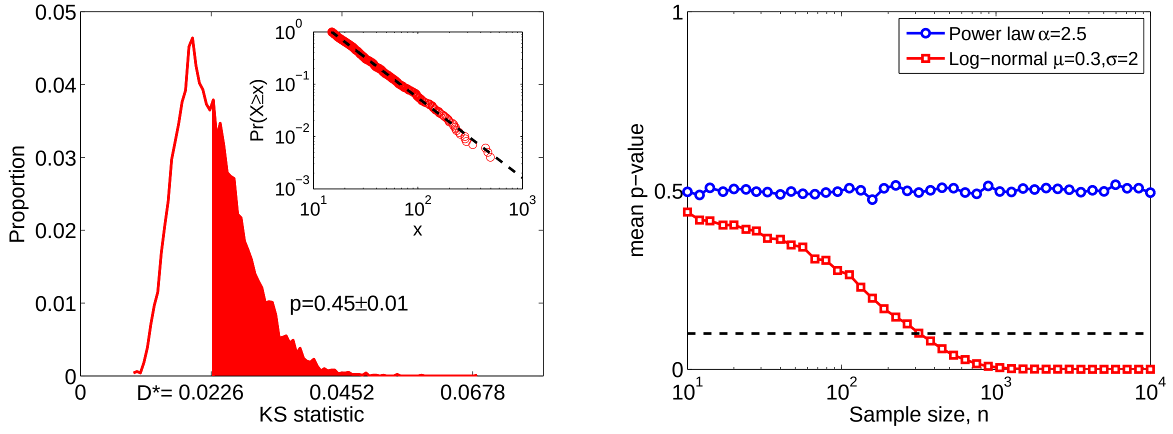

Figure 1: (a) The distribution of deviations $\Pr(D)$ for $n = 1000$ observations drawn from a power-law distribution with $x_{\min} = 15$ and $\alpha = 2.5$; the inset shows a single synthetic data set with its maximum likelihood model ($\hat{\alpha} = 2.530 \pm 0.048$), and D^* marks the corresponding deviation. (b) The behavior of the average p-value for data drawn from a power-law distribution and from a log-normal distribution, when both are fitted with a power-law form.

The inner loop runs a Monte Carlo simulation to numerically estimate $\Pr(D)$, the distribution of deviations between the null model fitted to data generated by the same null model. It does this by repeating many many times the same three steps we performed on our empirical data, (i) data acquisition (generation), (ii) model fitting or parameter estimation, and (iii) deviation measurement. If the model parameters are known *a priori*, e.g., the model has no parameters or their values are known independently of our data, then we can skip the estimation step.[5]

2.1 False conclusions and statistical power

If our data $\{x_i\}$ are truly generated by our null model A, then the distribution of p-values $\Pr(p)$ can be proved to be uniform on the unit interval $(0, 1)$ (do you see why this makes sense?). By convention, we "reject" the hypothesis A if $p < \alpha$, where α is an arbitrary value. Depending on the community, choices of $\alpha = 0.1$, 0.05 and 0.01 are typical. The choice of α simply sets the *false conclusion* rate. For instance, when $\alpha = 0.1$, we can expect to falsely reject A as the generating process, when A is in fact the true generating process, about 10% of the time.

To demonstrate this more concretely, let's again use the power-law distribution (denoted model A)

[5]If we use θ^* in our calculation of the deviation of the data from the model, then we bias the deviations high and thus make D^* look smaller than it should be.

along with the log-normal distribution (denoted model B), which is defined as

$$\Pr(x) \propto \frac{1}{x}\mathrm{e}^{-\frac{(\ln x-\mu)^2}{\sqrt{2\sigma^2}}} \ , \tag{1}$$

where μ and σ are parameters. For large σ, log-normals have very heavy tails, and thus have broad regions that are approximately power-law-like. This effect is more pronounced if n is relatively small. We can see this if we take the logarithm of both sides of Eq. (1):

$$\ln \Pr(x) \propto -\ln x - \frac{1}{\sqrt{2\sigma^2}}(\ln x - \mu)^2 \ ,$$

where the quadratic term effectively controls the amount of curvature observed in the ccdf. When σ is large, this term is small and $\ln \Pr(x) \approx -\ln x$, which is a power-law form. The larger σ, the greater an n you need before this curvature becomes apparent.

2.1.1 A simple experiment

Suppose we do the following two experiments. First, let's generate some data using A and then fit them using A. That is, we fit to the data the same model that generated the data. In this case, it can be proved that p will be a uniformly distributed random variable on the $(0, 1)$ interval. Thus, we'll observe non-trivial p-values, i.e., $p > 0.1$, 90% of the time.

Now, let's generate our data using B (for instance, the log-normal) and then fit them using A (the power law). In this case, we would expect $p \to 0$ as $n \to \infty$. That is, as we consider larger and larger samples, the deviations between the observed data and the fitted model will become consistently larger than we expect, leading us to reject A as the generative process.

Figure 1b shows precisely this experiment, plotting the average p-value observed as a function of sample size n (recall that $\langle p \rangle = 1/2$ when $p \sim U(0, 1)$).[6] Although the experimental results confirm our intuition, it also shows that the p-value doesn't accurately detect the "wrong model" case for small sample sizes. That is, when the data set is small, the p-value remains high and we cannot reject the power-law hypothesis. This is a reasonable thing to do—when n is small, the fluctuations are large and we cannot tell the two hypotheses apart—but underlines the point that a high p-value does not conclusively validate a model, merely indicate that it is statistically plausible.

This behavior illustrates the idea of *statistical power*, which is the ability of a test to make accurate conclusions about the underlying generating process. In this case, as $n \to \infty$ the statistical power increases, i.e., the test better distinguishes power-law and log-normal data, but for small sample sizes, the statistical power is low and we cannot distinguish A from B. (Note: there's also some

[6]See Section 3 below for Matlab code that produces Fig. 1b.

dependency on the particular choices of θ_A and θ_B.)

A few more cautionary comments: In this backwards version of the hypothesis test, getting a non-trivial p-value does not mean that the power-law model is correct and that our data do in fact follow a power-law distribution. Rather, it simply means that we could not falsify that notion.

To demonstrate more conclusive support for the hypothesis requires (i) coming up with a mechanistic (causal) model that produces power-law distributed data, (ii) showing that the model assumptions hold up empirically, and (iii) showing that it produces data that are statistically indistinguishable from the empirical data. In principle, once we have a candidate mechanistic model M, we can replace A with M in the procedure above and compute its p-value. In practice, as the model being tested becomes more complicated, some aspects of this procedure become trickier. For instance, (i) the parameter estimation can be more difficult, especially if the likelihood function becomes rugged, (ii) the model's computational complexity can become problematic, and (iii) it can become more ambiguous how define a useful "deviation" between model and data as the model makes more and different kinds of predictions.

2.1.2 How to lie with hypothesis tests

In the forward version, which is far more common than the backwards version, a small p-value, which would lead you to reject the candidate model A as the generating process, may not in fact be particularly helpful. That is, a small p-value is only useful if the model A is scientifically meaningful.

This mean that we can always rig the test in favor of rejection (in favor of finding "statistical significance") by choosing a ridiculously unrealistic candidate model that is sure to fail for any reasonably complex empirical process. Tiny p-values like $p < 10^{-5}$ or even $p < 10^{-243}$ (which I saw in a paper just the other day) should actually be a warning sign that the candidate model was completely and utterly unrealistic to begin with because it only generates observations like the empirical data once in 100,000 or more trials.

3 Matlab code

3.1 Figure 1a

Here's Matlab code that produces a version of Fig. 1a for data drawn from a log-normal distribution but fitted using a power-law distribution, which illustrates the calculation of the *p*-value for a case where the null hypothesis is the wrong generating process for the data.

```
% parameter values for data generation
xmin     = 15;
[mu sig] = deal(0.3,2);
n        = 1000;

% (A) generate synthetic data
% bounded log-normal generator (via rejection sampling)
y = exp(mu+sig.*randn(n,1)); y(y<xmin) = [];
while length(y)<n
    yy = exp(mu+sig.*randn(n,1));
    yy(yy<xmin) = [];
    y = [y; yy];
    if length(y)>n, y = y(1:n); end;
end;
x = y;

% (B) fit the PL model to each synthetic data set
thetx = 1+n./sum(log(x./xmin));            % power law cdf

% (C) calculate KS statistics for these fits
cn    = (0:n-1)'./n;
cx    = 1-(xmin./sort(x)).^(thetx-1);   % power law cdf
gofx  = max( abs(cn - cx) );            %

% ----- figure showing the KS calculation
c1 = [sort(x) (0:1:n-1)'./n];
c2 = [sort(x) 1-(xmin./sort(x)).^(thetx-1)];
dd = abs(c1(:,2)-c2(:,2));
D  = max(dd); xs = find(dd==D);
rx = sort(x);

tt = [rx'; rx'];
ts = [(0:1:n-1)./n; (0:1:n-1)./n];
```

```
cx = [tt(:) ts(:)];
cx(1:end-1,2) = cx(2:end,2);
cx(end,2) = 1;

figure(4); clf;
%plot(sort(x),(0:1:n-1)./n,'r-','LineWidth',2); hold on;
semilogx(cx(:,1),cx(:,2),'r-','LineWidth',2); hold on;
semilogx(sort(x),1-(xmin./sort(x)).^(thetx-1),'k--','LineWidth',2);
semilogx([rx(xs) rx(xs)],[(xs-1)/n 1-(xmin./rx(xs)).^(thetx-1)],'ko-','LineWidth',2); hold off
set(gca,'FontSize',16,'YTick',(0:0.1:1));
xlabel('x','FontSize',16);
ylabel('Pr(X<x)','FontSize',16);
h=text(23,0.65,strcat('D=',num2str(D))); set(h,'FontSize',16);
legend('data','fitted model',4);
% -----

% (D) monte carlo calculation of Pr(D)
Ds = zeros(5000,3);
for k=1:size(Ds,1)
    % (i)   bounded power law
    r       = rand(n,1);
    qx      = xmin*(1-r).^(-1/(thetx-1));      % (i)   power law
    thexx   = 1+n./sum(log(qx./xmin));         %
    c       = 1-(xmin./sort(qx)).^(thexx-1);   %
    Ds(k,1) = max( abs((0:n-1)'./n - c) );     %
end;
p   = sum(Ds(:,1)>=gofx)./size(Ds,1);
bns = linspace(min(Ds(:,1)),max(Ds(:,1)),100);
h   = hist(Ds(:,1),bns)./size(Ds,1);
ds  = find(bns>=gofx,1,'first');
if isempty(ds), ds = length(bns); end;

% --- make a pretty figure showing the results
figure(6); clf;
plot(bns,h,'r-','LineWidth',2); hold on;
fill([bns(ds:end) bns(end:-1:ds)],[h(ds:end) 0.*h(end:-1:ds)],[1 0 0]);
plot([bns(ds) bns(ds)],[0 h(ds)],'r-','LineWidth',2);
plot(bns,h,'r-','LineWidth',2); hold off;
xlabel('KS statistic','FontSize',16);
```

```
ylabel('Proportion','FontSize',16);
set(gca,'FontSize',16,'XTick',gofx.*(0:4));
h2=text(1.6*gofx,0.01,strcat('p=',num2str(p,'%4.2f'),'\pm0.01')); set(h2,'FontSize',16);
h2=text(0.43*gofx,-0.00229,'D*='); set(h2,'FontSize',16);

% make the inset, showing the empirical data and the fitted model
h1=axes('position',[0.52 0.53 0.35 0.35]);
loglog(h1,sort(x),(n:-1:1)./n,'ro'); hold on;
loglog(h1,sort(x),(sort(x)./xmin).^(-thetx+1),'k--','LineWidth',2); hold off;
set(gca,'FontSize',14,'XTick',10.^(1:3),'YTick',10.^(-3:0));
set(gca,'XLim',10.^[1 3],'YLim',10.^[-3 0]);
ylabel('Pr(X\geqx)','FontSize',14);
xlabel('x','FontSize',14);
```

3.2 Figure 1b

And, here's Matlab code that produces Fig. 1b. The `xxxxxx` bits correspond to the random deviate generator for the power-law distribution that you derive as part of the first problem set; if you replace it with the correct expression, the code should run just fine. It will take a while (about 3 hours on my laptop) to finish as-is, but after each value of n, it will draw the current results in a figure so you can see how things evolve.

```
[xmin alpha] = deal(15,2.5);
[mu   sigma] = deal(0.3,2);

reps   = 1000;  % number of trials over which we average the p-value
N      = 100;   % number samples in Monte Carlo simulation of Pr(D)
nr     = unique(round(logspace(1,4,41)));                % sample sizes n

pvalx  = zeros(length(nr),reps);
pvaly  = zeros(length(nr),reps);

tic;
for i=1:length(nr)
    n = nr(i);                                           % set sample size
    for j=1:reps

        % (1) generate synthetic data for (i) PL (via transformation
        %     method) and (ii) LN (via rejection sampling)
        x = xxxxxxxxxxxxxxxxxxxxx;                          % PL
        y = exp(mu+sigma.*randn(n,1)); y(y<xmin) = [];    % LN
        while length(y)<n                                   %
            yy = exp(mu+sig.*randn(n,1));                   % cheat
            y = [y; yy(yy>=xmin)];                          % drop values <xmin
            if length(y)>n, y = y(1:n); end;                % truncate down to n obs
        end;


        % (2) fit the PL model to each synthetic data set

        thetx = 1+n./sum(log(x./xmin));                     % PL
        thety = 1+n./sum(log(y./xmin));                     % LN

        % (3) calculate KS statistics for the fitted models

        cn    = (0:n-1)'./n;                                % EDF
        cx    = 1-(xmin./sort(x)).^(thetx-1);               % PL
        gofx  = max( abs(cn - cx) );                        %
        cy    = 1-(xmin./sort(y)).^(thety-1);               % LN
        gofy  = max( abs(cn - cy) );                        %

        % (4) Monte Carlo simulation

        Ds = zeros(N,3);                                    % stores simulated Pr(D)
        for k=1:size(Ds,1)
            qx      = xxxxxxxxxxxxxxxxxxxxx;                % PL
            thexx   = 1+n./sum(log(qx./xmin));              %
            cx      = 1-(xmin./sort(qx)).^(thexx-1);        %
            Ds(k,1) = max( abs(cn - cx) );                  %

            qy      = xxxxxxxxxxxxxxxxxxxxx;                % LN
            theyy   = 1+n./sum(log(qy./xmin));              %
            cy      = 1-(xmin./sort(qy)).^(theyy-1);        %
            Ds(k,2) = max( abs(cn - cy) );                  %
        end;
        pvalx(i,j) = sum(Ds(:,1)>=gofx)./size(Ds,1);        % compute and store PL p value
        pvaly(i,j) = sum(Ds(:,2)>=gofy)./size(Ds,1);        % compute and store LN p value

    end;
```

```
    fprintf('n[%i]\t[%4.2fm]\n',i,toc/60);                 % write out our progress
    figure(1); clf;                                         % draw progress in a figure
    semilogx(nr(1:i),mean(pvalx(1:i,:),2),'bo-','LineWidth',2,'MarkerFaceColor',[1 1 1]);
    hold on;
    semilogx(nr(1:i),mean(pvaly(1:i,:),2),'rs-','LineWidth',2,'MarkerFaceColor',[1 1 1]);
    semilogx([nr(1) nr(end)],[0.1 0.1],'k--','LineWidth',2); hold off;
    set(gca,'YLim',[0 1],'YTick',[0 0.5 1],'FontSize',16);
    xlabel('Sample size, n','FontSize',16); ylabel('mean p-value','FontSize',16);
    h=legend('Power law \alpha=2.5','Log-normal \mu=0.3,\sigma=2',1);
    set(h,'FontSize',14);
    drawnow;

end;
```