Week 2, Lecture 2 - Fitting And Regression

Aaron Meyer

Outline

- Administrative Issues
- Project Proposals
- Fitting
  - Ordinary Least Squares
  - Non-Linear Least Squares
  - Generalized Linear Model
- Some Examples

Project Proposals

In this project, you have two options for the general route you can take:

1. Reimplement analysis from the literature.
2. New, exploratory analysis of existing data.

More details at:

https://aarmey.github.io/ml-for-bioe/final-project/

Project Proposals

The proposal should be less than two pages and describe the following items:

- Why the topic you chose is interesting
- Demonstrate that your project fits the criteria above
- What overall approach do you plan to take for the project and why
- Demonstrate that your project can be finished within a month
- Estimate the difficulty of your project

We are available to discuss your ideas whenever you are ready, and you should discuss your idea with us prior to submitting your proposal.

Recommend an early start—the earlier you finalize a proposal the sooner you can begin the project.

Goal Of Fitting

- Fitting is the process of comparing a model to a compendium of data
- After fitting, we will have a model that explains existing data and can predict new data

Process of Fitting

The *process* of fitting is nothing more than finding the maximum likelihood distribution of models for a set of points.

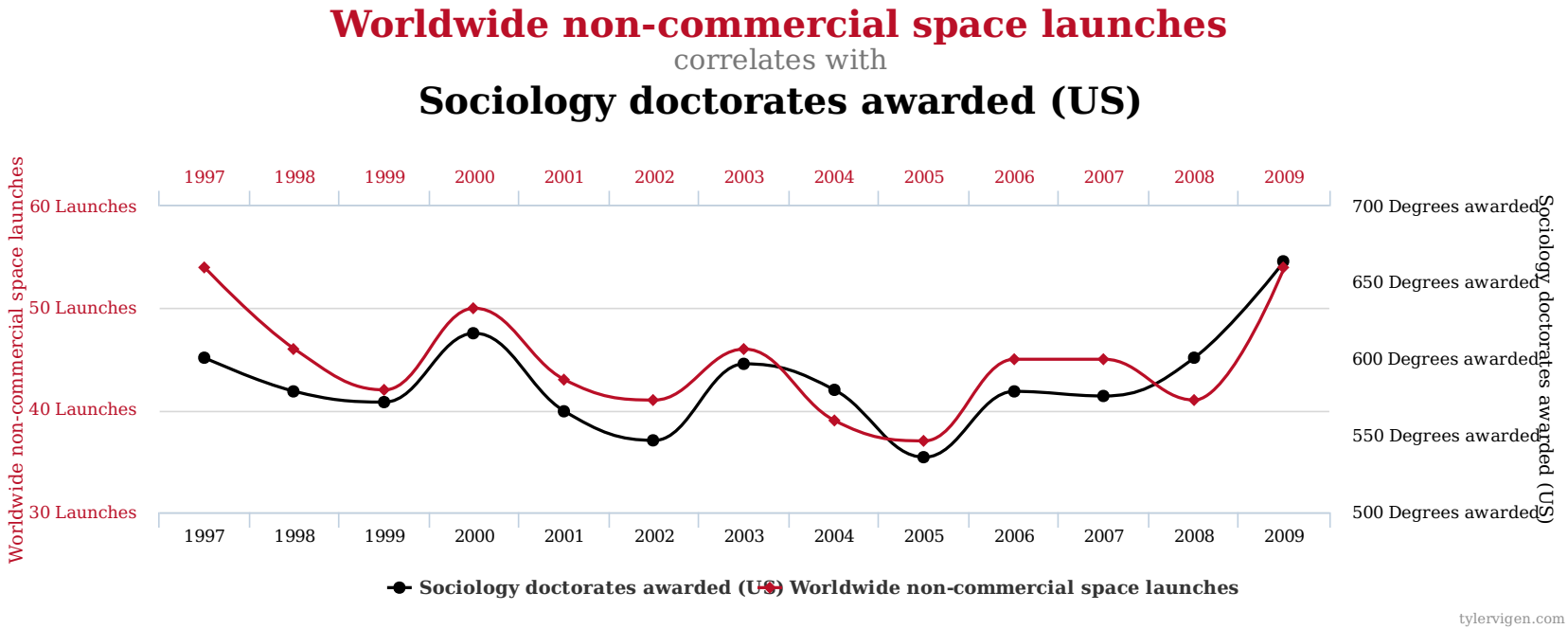
The key factor is how one defines the problem—i.e. how the distribution is described.

Caveats

- Any fitting result is highly dependent upon the correctness of the model
- Successful fitting requires concordance between the model and data
  - Too little data and a model is underdetermined
  - Unaccounted for variables can lead to systematic error

Since all models are wrong the scientist cannot obtain a "correct" one by excessive elaboration. On the contrary following William of Occam he should seek an economical description of natural phenomena. Just as the ability to devise simple but evocative models is the signature of the great scientist so overelaboration and overparameterization is often the mark of mediocrity. ~George Box, *J American Stat Assoc*, 1976

Any Fitting Is Dependent On The Correctness Of The Model



Fitting does not happen in a vacuum!

Ordinary Least Squares

- Probably the most widely used estimation technique.
- Based on extending the maximum likelihood estimate of a distribution.
- Model assumes output quantity is linear combination of inputs.

Ordinary Least Squares

If we have a vector of n observations $\mathbf{y}$, our predictions are going to follow the form:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

Here $\mathbf{X}$ is a $n \times p$ structure matrix, β is a p-dimensional vector with the parameters of our model, and $\epsilon = (\epsilon_1, \epsilon_2, ...\epsilon_n)'$ is the noise present in the model.

Ordinary Least Squares

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

ϵ is usually handled to be uncorrelated random components with constant variance σ^2:

$$\epsilon \sim \left(\mathbf{0}, \sigma^2\mathbf{I}\right)$$

Ordinary Least Squares

Single variable case

The structure matrix is little more than the data, sometimes transformed, usually with an offset. So, another way to write:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

would be:

$$\mathbf{y} = m_1\mathbf{x_1} + m_2\mathbf{x_2} \ldots + b + \epsilon$$

Ordinary Least Squares

Single variable case

$$\mathbf{y} = m\mathbf{x} + b + \epsilon$$

The values of m and b that minimize the distance from y are optimal, and they don't depend on ϵ.

Ordinary Least Squares

Gauss and Markov in the early 1800s identified that the least squares estimate of β, $\hat{\beta}$, is:

$$\hat{\beta} = \arg\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2$$

Ordinary Least Squares

$$\hat{\beta} = \arg\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2$$

can be directly calculated by:

$$\hat{\beta} = \mathbf{S}^{-1}\mathbf{X}'\mathbf{y}$$

where

$$\mathbf{S} = \mathbf{X}'\mathbf{X}$$

Ordinary Least Squares

$\hat{\beta}$ is the maximum likelihood estimate of β, and has covariance matrix $\sigma^2\mathbf{S}^{-1}$:

$$\hat{\beta} \sim \left(\beta, \sigma^2\mathbf{S}^{-1}\right)$$

In the normal case (when our assumptions hold), $\hat{\beta}$ is an *unbiased estimator* of β. Making these calculations tractable for larger data sets used to be a challenge but is now trivial.

Ordinary Least Squares

Likelihood of Model

$$\frac{-n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - x_i\beta)^2$$

therefore, only considering β (the only factor that influences predictions), we need to maximize:

$$\sum_{i=1}^{n}(y_i - x_i\beta)^2$$

Exactly how we calculate β!

Ordinary Least Squares

What might be some advantages of a method such as this?

Ordinary Least Squares

What are some of the assumptions?

What are the implications of these assumptions not holding?

What are some downsides?

Application: Paternal *de novo* mutations

Questions:

- Where do *de novo* mutations arise?
- Are there factors that influence the rate of *de novo* mutations from one generation to another?

Application: Paternal *de novo* mutations

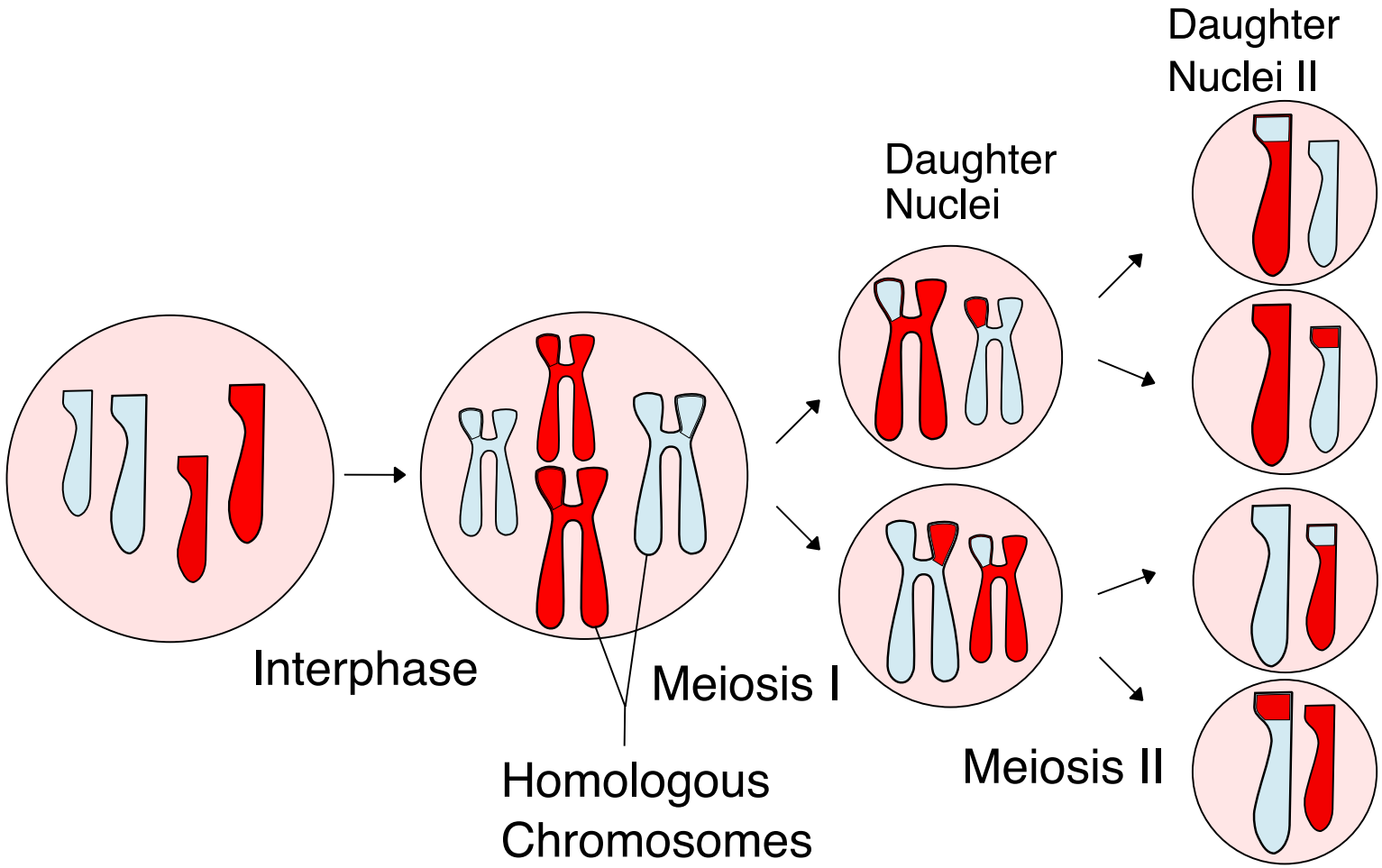


Figure: By Rdbickel - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=49599354

Application: Paternal *de novo* mutations

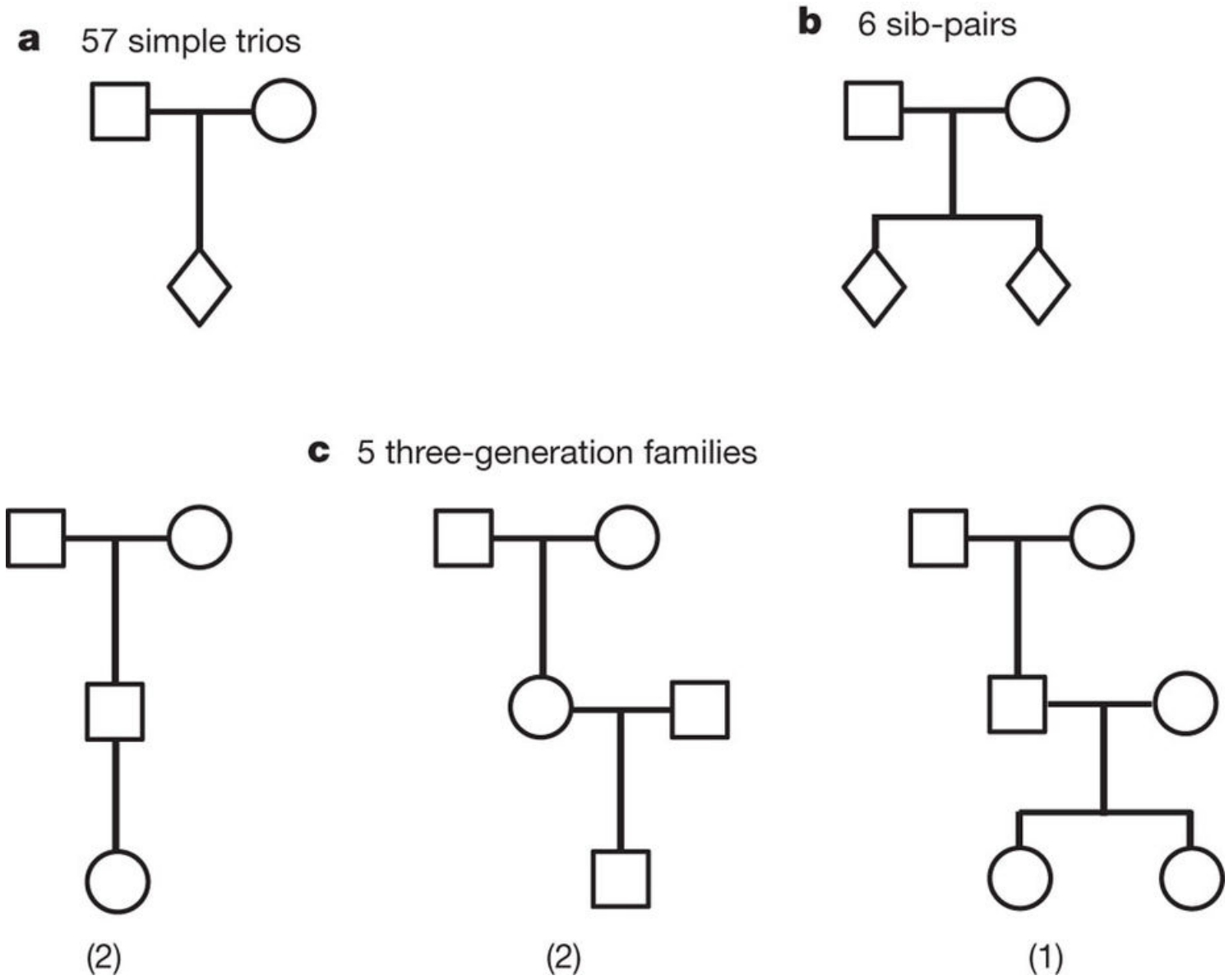


Figure: Kong *et al*, *Nature*, 2012

Application: Paternal *de novo* mutations

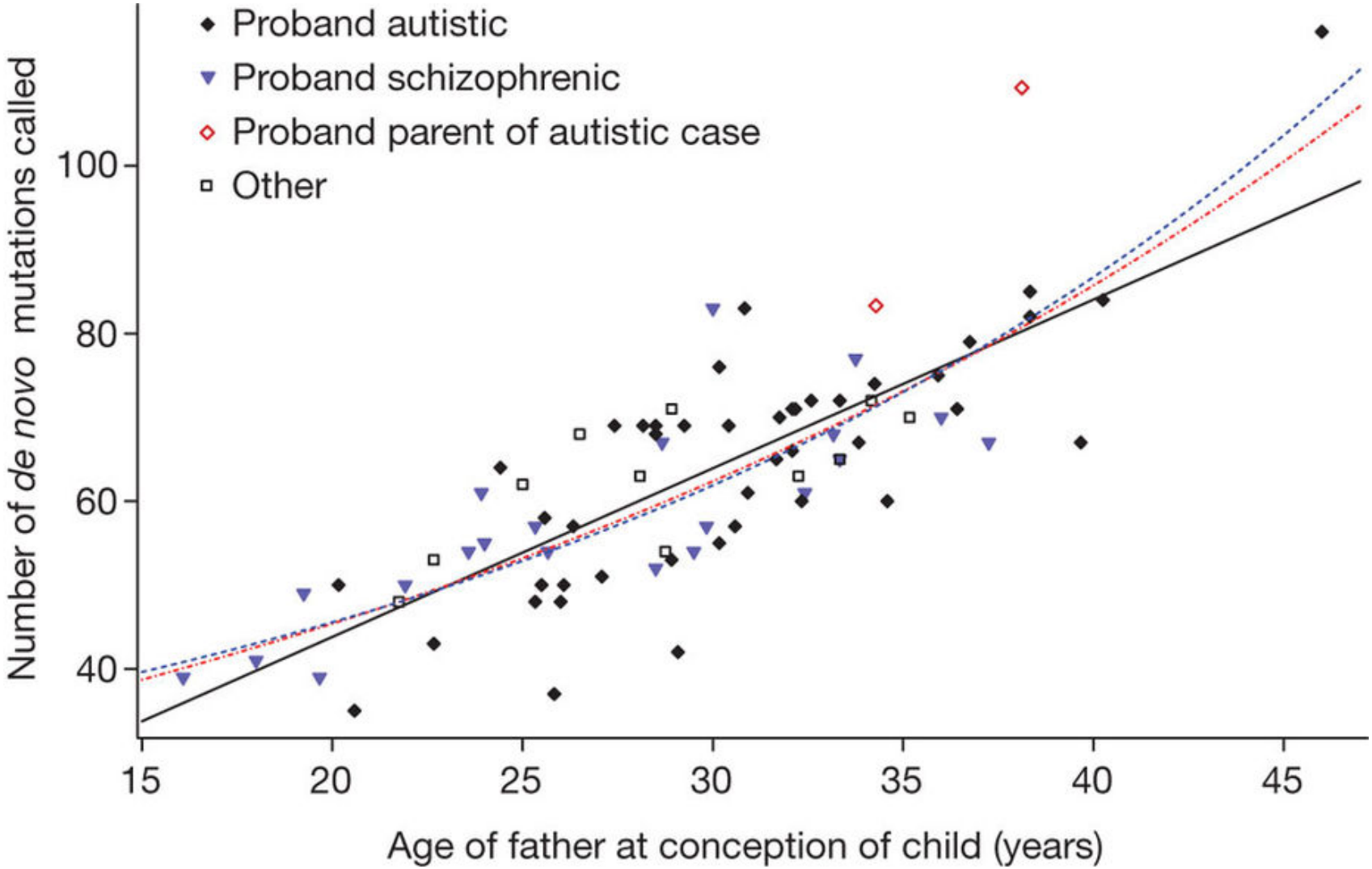


Figure: Kong *et al*, *Nature*, 2012

Implementation

sklearn provides a very basic function for ordinary least squares.

- `sklearn.linear_model.LinearRegression`
  - `fit_intercept`: Should an intercept value be fit?
  - `normalize`: Should the input variables be mean and variance scaled?
- No tests for significance/model performance included.
- We'll discuss evaluating the model in depth later.

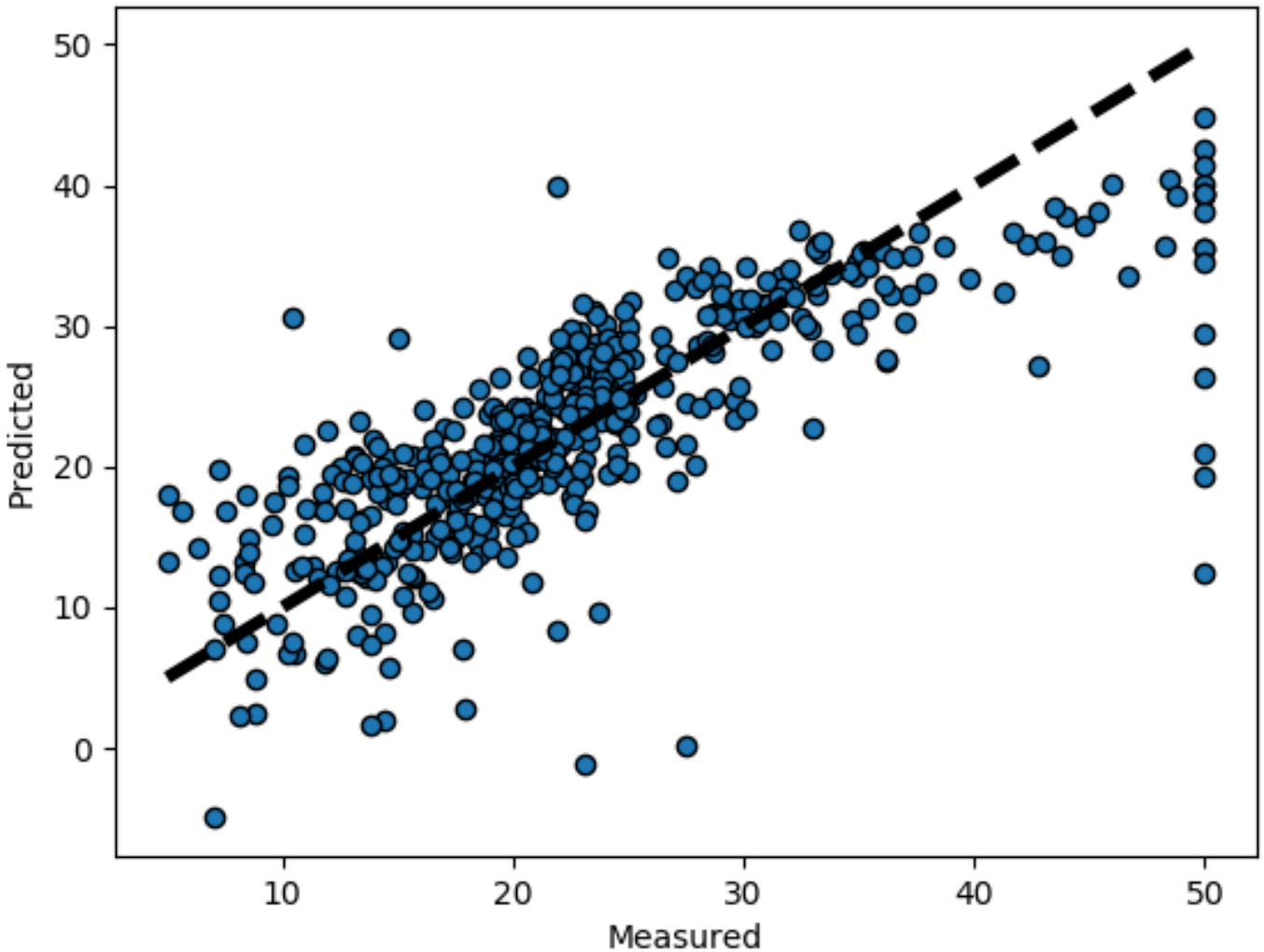
Or there's an even more bare function in numpy `numpy.linalg.lstsq`.

- Takes input variables `a` and `b`.
- Solves the equation $ax = b$ by computing a vector x that minimizes the Euclidean 2-norm $\|b - ax\|^2$.

Implementation

```
import sklearn as sk, matplotlib.pyplot as plt

lr = sk.linear_model.LinearRegression()
boston = sk.datasets.load_boston()
y = boston.target

lr.fit(boston.data, y)

predicted = lr.predict(boston.data)

fig, ax = plt.subplots()
ax.scatter(y, predicted, edgecolors=(0, 0, 0))
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
plt.show()
```

Implementation



Non-Linear Least Squares

Non-Linear Least Squares makes similar assumptions to ordinary least squares, but for arbitrary functions. Thus, instead of following the form:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

Our input-output relationship is:

$$\mathbf{y} = f(\mathbf{X}, \beta) + \epsilon$$

for the same construction of ϵ.

Transformation

NNLSQ used to be mostly performed by transforming one's data into a linear model.

E.g. taking the ratio of variables, or log-transforming them.

This is now considered **bad practice**.

Why?

Non-Linear Least Squares

Algorithms

We again need to solve for β to minimize the sum of squared error:

- There are many methods to solve these problems, and finding the true minimum is not a trivial task.
- We're not going to cover how these algorithms work in depth.

Non-Linear Least Squares

Algorithms

One property we can take advantage of is that the gradient of the SSE w.r.t. β at the minimum is zero (r_i is the residual of the ith point):

$$\frac{\partial S}{\partial \beta_j} = 2 \sum_i r_i \frac{\partial r_i}{\partial \beta_j} = 0$$

- $\frac{\partial r_i}{\partial \beta_j}$ is a function of both the nonlinear function and the data.
- This can be expanded out through a first-order Taylor approximation.
- Doing so essentially performs ordinary least squares around the current point, for the linearized function.

Non-Linear Least Squares

Algorithms

$$\frac{\partial S}{\partial \beta_j} = 2 \sum_i r_i \frac{\partial r_i}{\partial \beta_j} = 0$$

- $\frac{\partial r_i}{\partial \beta_j}$ is a function of both the nonlinear function and the data.
- This can be expanded out through a first-order Taylor approximation.
- Doing so essentially performs ordinary least squares around the current point, for the linearized function.
  - $\frac{\partial r_i}{\partial \beta_j} = -J_{ij}$, where J is the Jacobian of the function.
  - Many NNLSQ solvers require J for this reason: can be approximated by finite differences.
  - Probably the most common method, Gauss-Newton, uses this property with Newton's method.

Non-Linear Least Squares

Algorithms - Key Takaways

- Unlike ordinary least squares, no guarantee about finding the optimal solution.
- Depending upon the data and model, there may be many local minima.
- Exactly equivalent to shifting normal distributions up and down around one's data.

Implementation

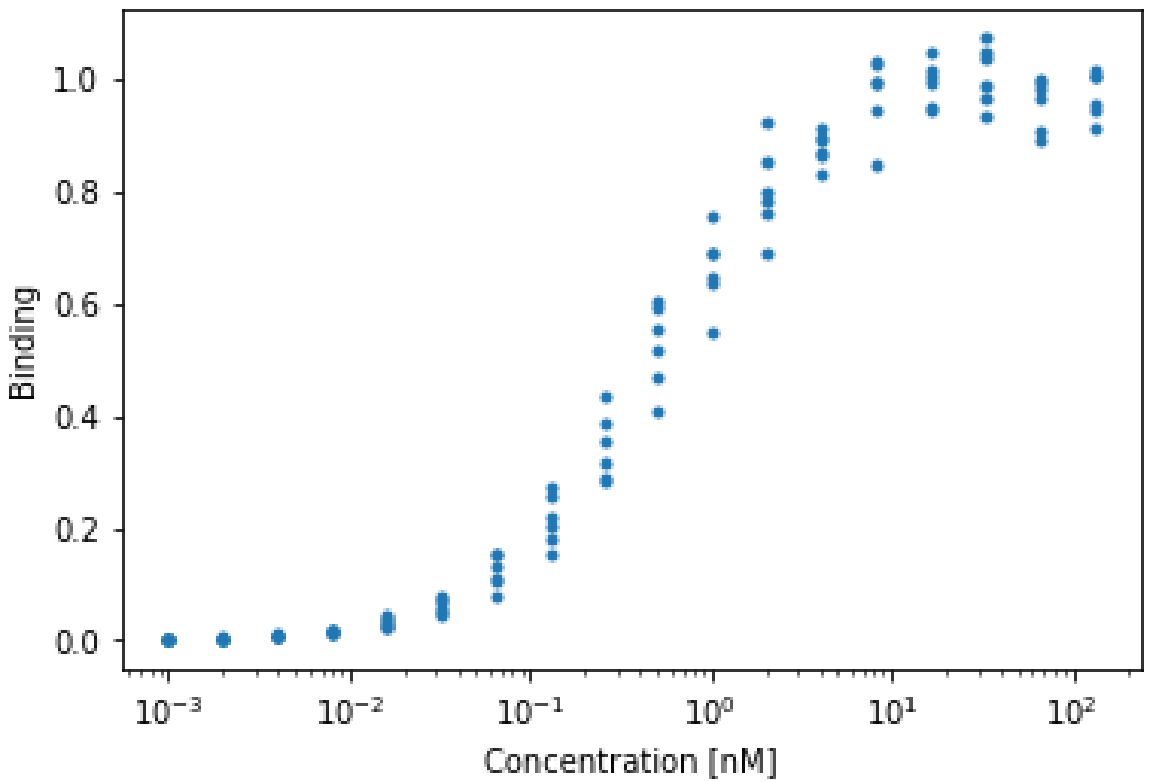
SciPy's `scipy.optimize.least_squares` is a very capable implementation.

- The main necessary parameters are:
  - `fun`, the function
  - `x0`, an initial guess for the parameter values
- Note that `fun` should return a vector of the residuals
  - So it should handle all the data itself

NNLSQ Example - Binding Data

Let's say we're looking at a protein-protein interaction such as this:

```
plt.semilogx(X, Y,'.');
plt.xlabel('Concentration [nM]')
plt.ylabel('Binding')
```



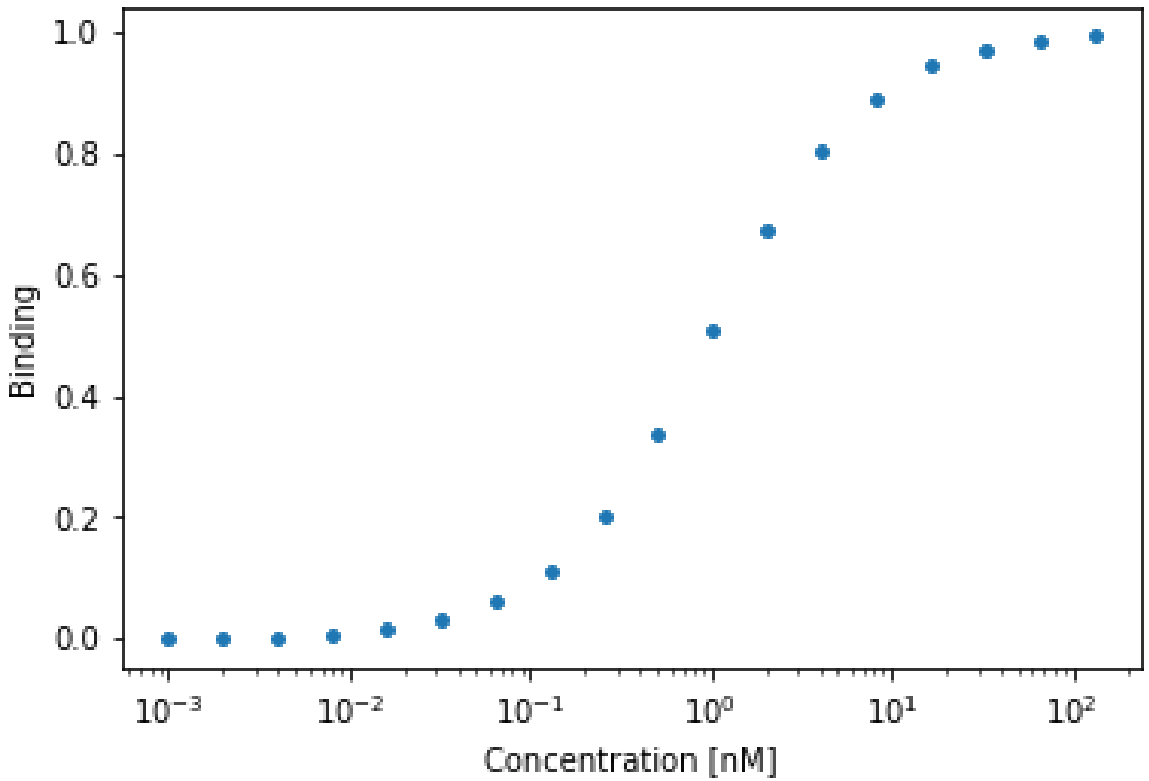
NNLSQ Example - Binding Data

We can predict the amount of binding we'd observe from a single-site binding model:

```
def klotz1(k1, lig):
    return (k1*lig)/(1 + k1*lig)
```

NNLSQ Example - Binding Data

```
plt.semilogx(X,klotz1(1.,X),'.')
plt.xlabel('Concentration [nM]')
plt.ylabel('Binding')
```



NNLSQ Example - Binding Data

SciPy asks for the residuals at each fitting point, so we need to convert a prediction to that:

```
def ls_obj_k1(k1, ligs, data):
    return(data - klotz1(k1,ligs))
```

NNLSQ Example - Binding Data

```
sp.optimize.least_squares(ls_obj_k1, 1., args=(X,Y))
# --------
 active_mask: array([ 0.])
        cost: 0.008677649670891657 3
         fun: array([  4.79e-05,   9.00e-05,  -1.09e-04,
         8.04e-04,  -9.67e-04,   3.85e-03,
         4.61e-03,   2.34e-03,   2.36e-02,
         9.64e-03,  -2.48e-02,   1.93e-02,
        -4.93e-02,   5.54e-02,  -3.66e-02,
         2.97e-03,   3.39e-02,  -8.74e-02])
        grad: array([ -9.57228474e-09])
         jac: array([[-0.00099809],
       [-0.00199235],
       [-0.0039695 ],
# ...
       [-0.03119024],
       [-0.01608763],
       [-0.00817133]])
     message: '`gtol` termination condition is satisfied.'
        nfev: 4
        njev: 4
```

Generalized Linear Model

What if the error term isn't Gaussian?

- In many cases linear regression can be inappropriate
  - E.g. A measurement that is Poisson distributed

Questions

- Would you expect the confidence interval of your model to be larger or smaller than the confidence interval of prediction?
- Given the binding data presented here, do you think a least squares model is most appropriate?
- How might you test whether your data fits the model you've specified?

Further Reading

- Computer Age Statistical Inference, Chapter 8
- sklearn: Linear Models