Computational Cognitive Science

Lecture 3: Parameter Estimation

Chris Lucas

School of Informatics

University of Edinburgh

September 24, 2019

Reading

Reading: Chapter 3 of F&L

(don't worry about bootstrapping or exact details of simplex algorithm)

Introduction

Most models have free parameters.

Example:

- Decision threshold or criterion in the random-walk decision model

Introduction

Example:

- Coefficients in a linear regression model

E.g., b_0 and b_1 in

$$y = b_0 + b_1 x$$

Introduction

Example:

- ϵ in "ϵ-greedy" models: The probability that someone chooses a random option over the best so far.

To model trade-offs between exploration and exploration, like choosing between restaurants or foraging spots.

Introduction

Example:

- Weights in a neural network

GPT2 (a language generation model) has > 8 billion parameters.

Also, "hyperparameters", e.g., learning rate, architectural decisions.

Why estimate parameters?

Usually necessary for precise and accurate predictions.

- A model with free parameters $\rightarrow$ a family or space of models.
- Sometimes we want to choose between model families – we can estimate parameters to pick a best representative of that family.
- Sometimes we want to identify better models **within** a family, e.g.,

$$b_0 + b_2 x^2$$

in

$$b_0 + b_1 x + b_2 x^2 + b_3 x^3$$

Why estimate parameters?

Some parameters are *interpretable*: They describe or explain behavior in a way that people can understand

- Random-walk model: Threshold, drift rate, standard deviation
- ϵ in ϵ-greedy: How conservative are people?

(Estimated parameters might tell us about people in general, individuals, or both)

Parameter Estimation

The aim of parameter estimation is to find the parameter values that *minimize loss* (or error), according to some discrepancy function.

- If our goal is to have an accurate predictive model, we want to minimize predictive error.
- If our goal is to interpret parameters, we want the most likely or plausible values.

These goals often go hand in hand. Today we'll focus on the first.

Quantifying error

A popular loss[1] function function is *root mean squared deviation*:

$$RMSD = \sqrt{\frac{\sum_{j=1}^{J}(d_j - p_j)^2}{J}}$$

where

- J is the number of data points
- p_j (or $\hat{y}_j$) is the j^{th} prediction
- d_j (or y_j) is the j^{th} data point

Minimizing RMSD is equivalent to minimizing the sum squared error or the mean squared error; RMSD is easier to interpret.

[1] or "discrepancy" or "error". Can also call it RMSE.

Quantifying error

RMSD has some appealing/intuitive features:

- Larger errors are worse than small errors
- Often easy to minimize
- Min RMSD $\rightarrow$ max likelihood in some cases

Quantifying error

Alternatives to RMSD:

- Mean absolute error: $\frac{1}{J}\sum_{j=1}^{J}|d_j - p_j|$
- 0/1 error: Can be useful in specific situations (e.g., roulette)
- R^2 (and adjusted R^2): Not scale-dependent (good) but blind to systematic bias (bad?)

Sometimes goal-appropriate loss functions are important, e.g., - It's better to deploy a parachute too early than too late. - Medical diagnosis: Low-cost, low-precision tests can be useful.

Quantifying error: Discrete variables

Most loss functions for continuous variables aren't applicable to discrete variables (e.g., forced-choice judgments).

In psychological research, one popular discrepancy function for discrete variables is χ^2:

$$\chi^2 = \sum_{j=1}^{J} \frac{(O_j - Np_j)^2}{Np_j}$$

where

- J is the number of response categories
- N is the total number of responses
- O_j is the observed number of responses in category j
- p_j is the predicted probability for response j

Quantifying error: Discrete variables

Another common choice is G^2:

$$G^2 = 2 \sum_{j=1}^{J} O_j \log(\frac{O_j}{N p_j})$$

These are popular in part because they are used in null hypothesis significance tests – we can quantify "badness of fit" of a model.

In both cases, zero indicates a perfect fit.

Quantifying error: Discrete variables

Issues:

- Not that easy to interpret
- Require not just predictions, but probabilities – what happens if a predicted probability is zero?

Also:

- Accuracy – proportion of correct predictions – is an simple, intuitive, and complementary measure.

Quantifying error: Binary variables

Many discrete loss functions focus on binary tasks, e.g., "$y = 1$ iff x is in the target category, else $y = 0$". We have:

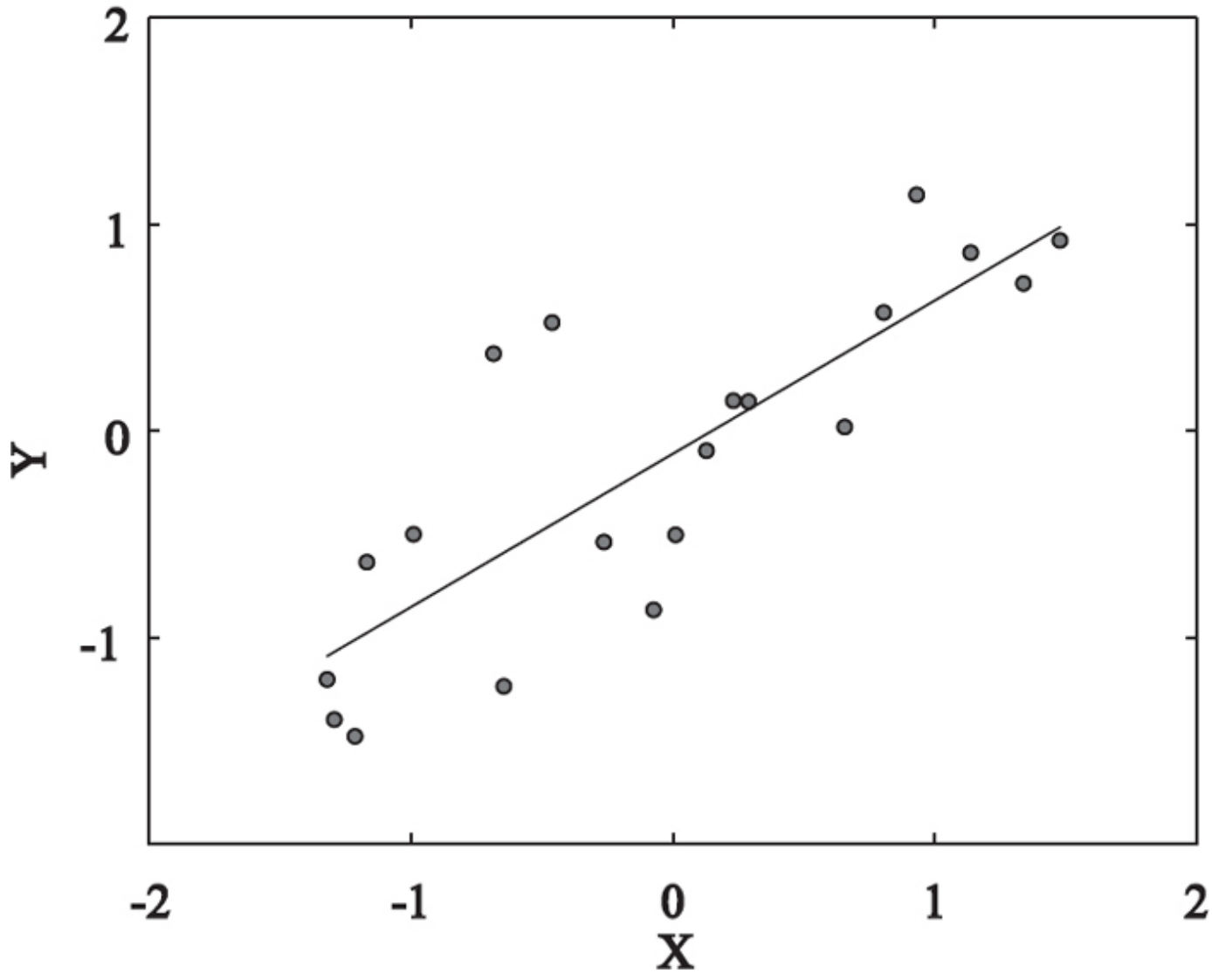
- true positives ($\hat{y}_i = 1$, $y_i = 1$)
- false positives ($\hat{y}_i = 1$, $y_i = 0$)
- true negatives ($\hat{y}_i = 0$, $y_i = 0$)
- false negatives ($\hat{y}_i = 0$, $y_i = 1$)

Quantifying error: Binary variables

We can use these to express several measures of goodness of fit:

- precision: $TP/(TP + FP)$ – useful, but easy to game
- recall: $TP/(TP + FN)$ – useful, but easy to game
- F1-score: $F_1 = 2\frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$
  - the harmonic mean of precision and recall; requires good performance in both

Example: Linear Regression



Linear Regression

Let's assume a simple model that describes a set of data points (x_i, y_i) as follows:

$$y_i = b_0 + b_1 x_i + e_i$$

i.e.,

$$\mathbf{y} = \mathbf{Xb} + \mathbf{e}$$

This equation describes *linear regression.* Here, b_0 and b_1 (intercept and slope) are the *parameters* of the model, and e_i is an error term – often assumed to be Gaussian.

How do we estimate these parameters?

How to minimize a loss function?

Suppose we want to minimize predictive RMSD.

A proxy: Minimize RMSD for the data we have.

Some approaches:

- Analytic solution
  - $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$
- "try them all" – look at a dense grid of possible parameters
- search selectively for an optimum

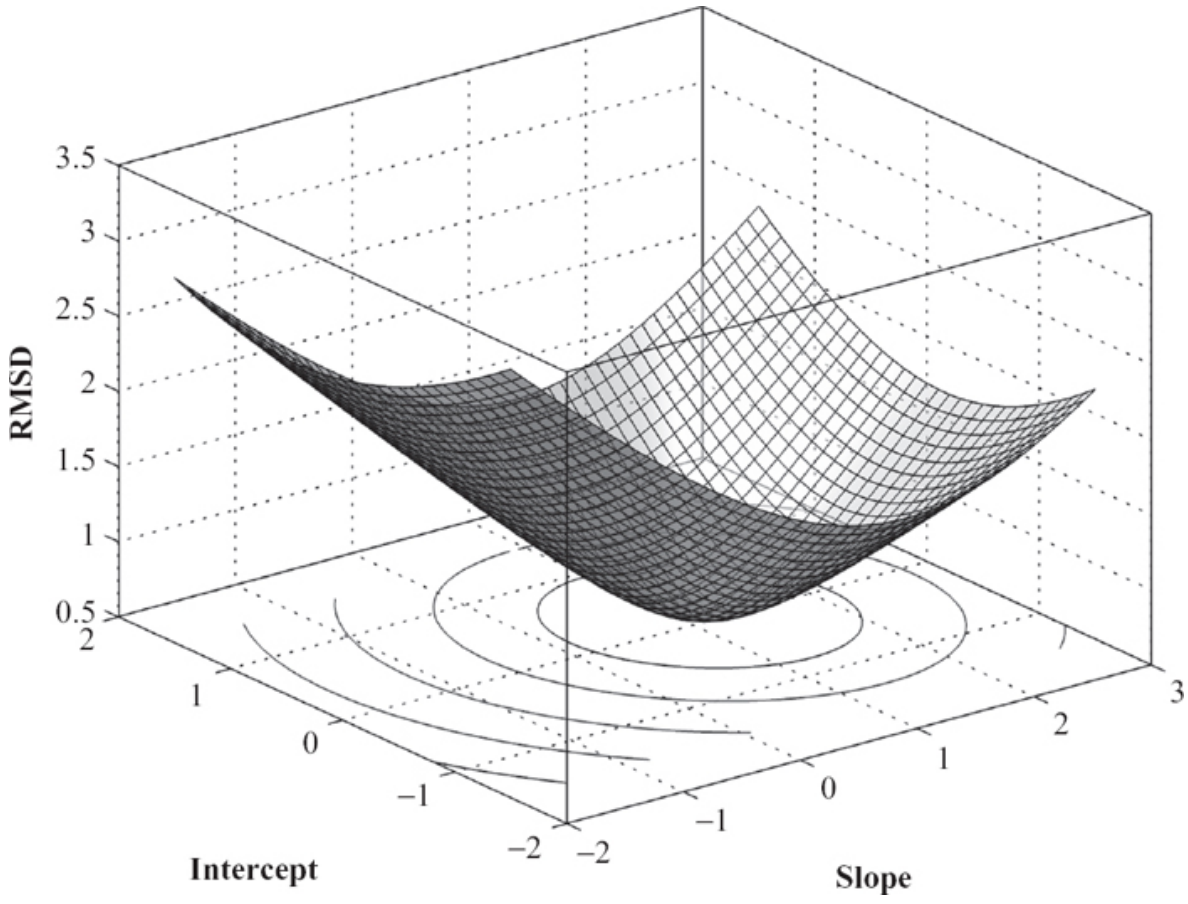
Optimization is covered in depth in other courses, e.g., MLP[2].

[2] For a relevant reading, see http://www.deeplearningbook.org/contents/optimization.html

Parameter Estimation: Analytic solutions

- *Often* the best approach, where available.
- Usually not available.

Parameter Estimation: Gridded search



Parameter Estimation: Gridded search

Pros:

- Easy to see how error changes as a function of 1-2 parameters.
- Easy to implement.
- No issues with local optima.

Issues with G^2:

- Intractable for large parameter spaces or computationally expensive problems.
  - 5 parameters, 20 values per, 0.1 seconds to compute: 89 hours.
- Doesn't find actual optimum.
- Bounds of grid aren't always known in advance.

If tractable, worth doing; complementary to other approaches.

Parameter Estimation: Sequential search / gradient descent

1. Determine a starting value for the parameters (randomly or through an educated guess)
2. Propose an adjustment to the parameters; use this adjustment if it reduces error
3. Iterate until no further error reduction is possible

Having gradients (i.e., how error changes locally as a function of parameters) can make this very efficient.

Parameter Estimation: Sequential search / gradient descent

See text for an example: the Nelder-Mead optimization algorithm.

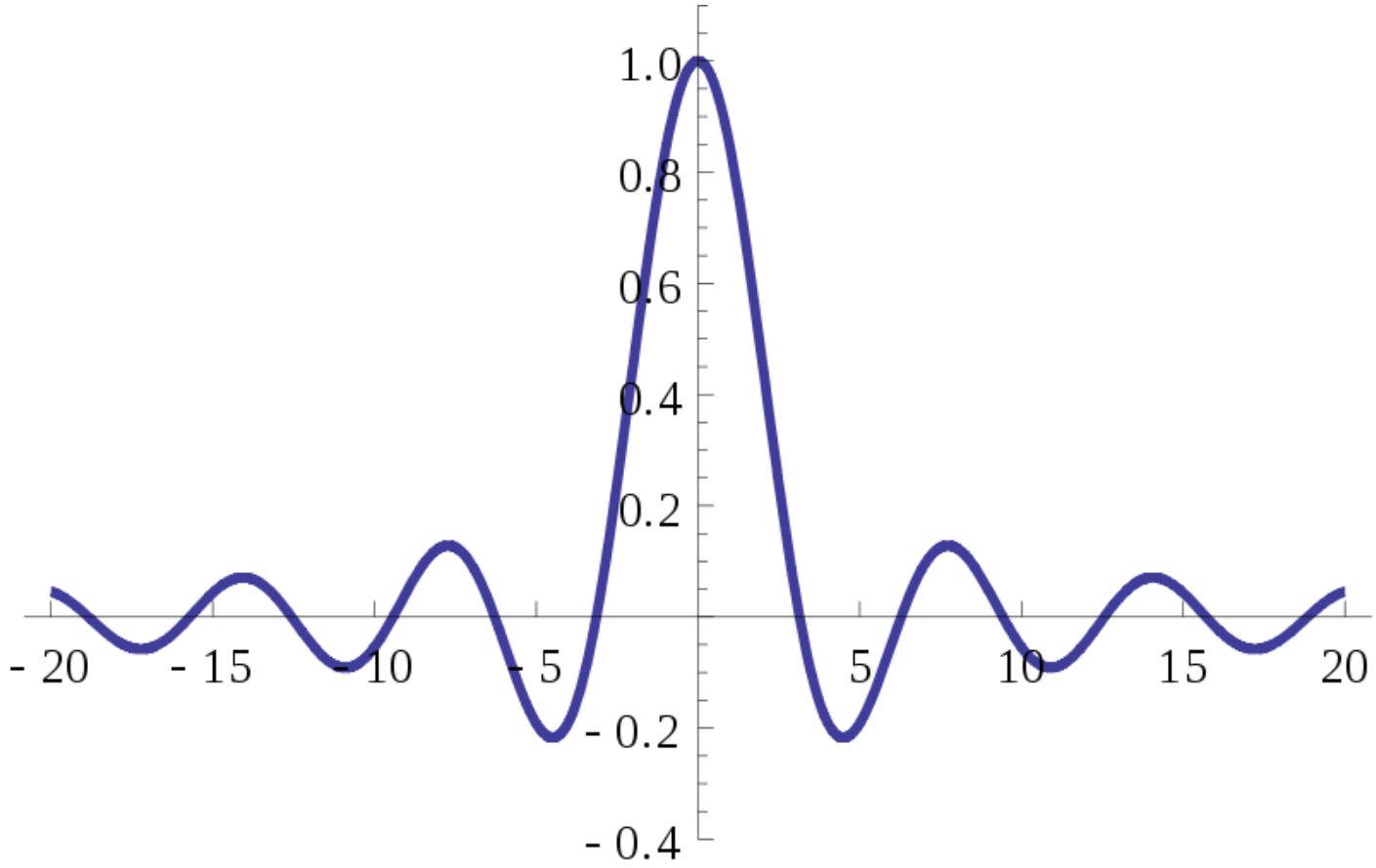
This algorithm is simple and popular, but has some disadvantages:

- Deals poorly with high-dimensional problems
- Deals poorly with stochastic loss functions (e.g., simulation-based models)
- Deals poorly with constrains on variables
- Can't deal with non-continuous variables
- Not very good at dealing with local optima

Pitfall: Local optima

Most efficient methods for optimizing non-trivial functions only guarantee finding *local* optima.

These may not be the same as the *global* optimum.



Pitfall: Local optima

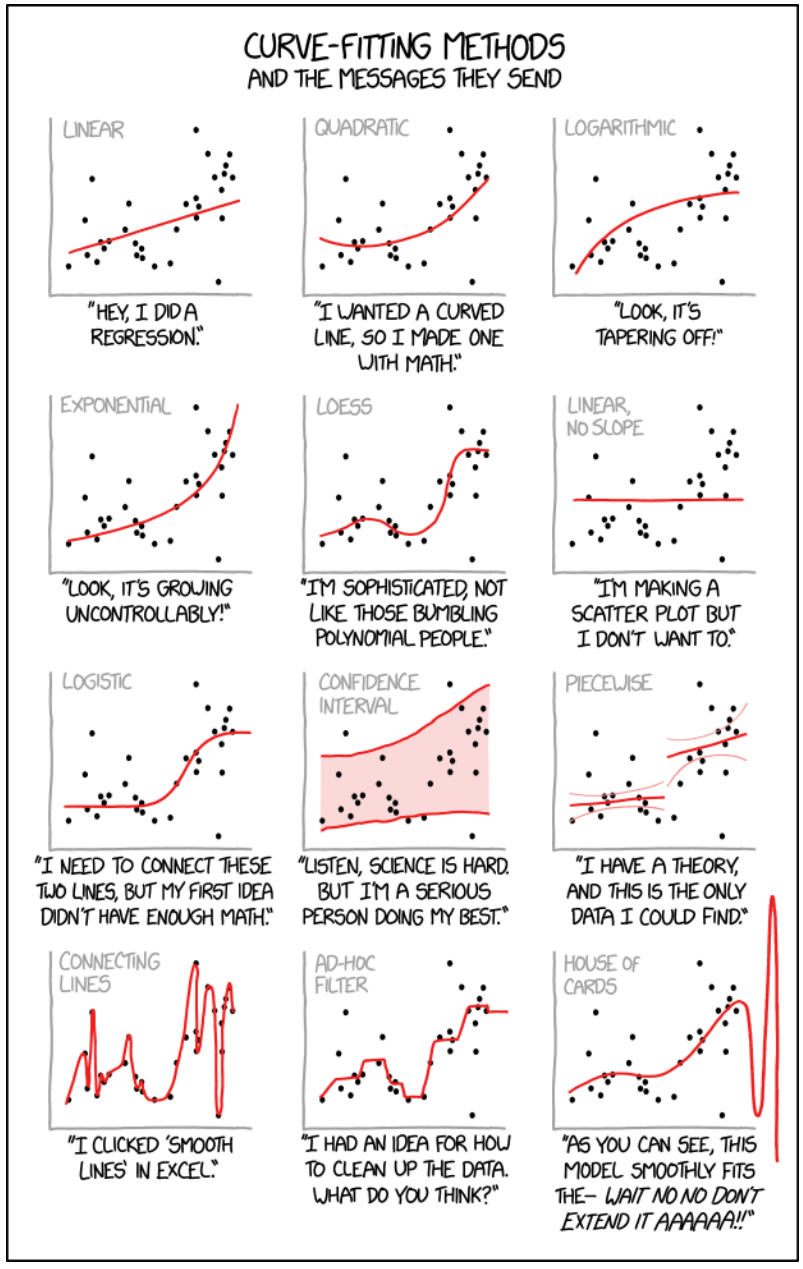
There is no magic solution to the problem of local optima.

Many methods exist to mitigate it. See the text for a discussion of one of these: *simulated annealing*.

Pitfall: Overfitting and predictive accuracy

We care about *predictive* error, not post-hoc error on our old data. The latter is not always a good proxy for the former.

Pitfall: Overfitting and predictive accuracy



Pitfall: Overfitting and predictive accuracy

How do we find parameters that maximize predictive accuracy?

- If data set is extremely large and representative, fits can be a reasonable proxy.
  - This is rarely true for experimental data.
- We can use prior knowledge, e.g., in regression:
  - parameters in a regression model are likely to be close to zero.
  - most features are likely to be irrelevant.
- "regularized" or "penalized" models, e.g., minimize $\mathrm{RMSD} + f(\mathbf{b})$.
- Other tricks: "drop-out", early stopping, ...

Pitfall: Overfitting and predictive accuracy

How do we assess the predictive accuracy of a model?

- Predict! Set aside representative data, one test once model is final.

Failing that, there are some approximations, e.g., penalties for model complexity.

Doing this well is imporant for model selection – discussed in a future lecture.

Discrepancy functions and fitting: What are the data?

We've been assuming we know what our data are. Do we?

Consider the reaction-time experiment:

- Are all participants the same, e.g., same threshold for evidence?
- Is it realistic to predict the accuracy and reaction time for every judgment?
- What if we take averages per condition as our data?
  - This is common in psychological research.
  - Is is safe to do? Not always.
  - Later: Data aggregation

Summary

- Cognitive models often have parameters that need to be estimated
- These are optimized relative to a discrepancy or loss function
  - RMSD is a popular choice, but loss function should be carefully considered
- Many optimization methods exist
  - Analytic
  - Exhaustive/gridded
  - Sequential; see text for discussion of the Nelder-Mead algorithm
- Beware local optima. Simulated annealing is one way escape them
- Post-hoc error not always a good indicator of predictive error