Comp Neuro

Introduction to Modeling

Model Types

▼ Intro

In any research, we typically start with descriptive (“what”) models; you will see examples of those during the model fitting, GLM, dimensionality reduction, and deep learning days. Next, we often ask about the mechanisms and build “how” models to generate or test hypotheses of underlying mechanisms; examples of those will be in linear systems, real neurons, dynamic networks, and decision making days. Ultimately, we are usually interested in the underlying reason of why the phenomenon exists in the first place; examples of those are in Bayes, optimal Control, and reinforcement learning days.

（1）Cosine Tuning of neuronal activity

（2）Hodgkin & Huxley single-neuron spike generation model

（3）Reinforcement Learning

Why models are great?

- Knowledge synthesis (CT, HH, RL)
- Identifying hidden assumptions, hypotheses, unknowns (HH)
- Mechanistic insights (HH)
- Retrieve latent information (HH, RL)
- A testbench for medical interventions (HH)
- Guidance in designing useful experiments → quantitative predictions (HH, RL)
- Inspire new technologies / applications (CT, RL)

▼ “What” models

Objective:

- Load a dataset with spiking activity from hundreds of neurons and understand how it is organized
- Make plots to visualize characteristics of the spiking activity across the population
- Compute the distribution of “inter-spike intervals” (ISIs) for a single neuron
- Consider several formal models of this distribution’s shape and fit them to the data “by hand”

Practice in Jupyter Notebook

▼ “How” models

Objective:

To understand the mechanisms that give rise to the neural data we save in Tutorial 1, we will build simple neuronal models and compare their spiking response to real data. We will:

- Write code to simulate a simple “leaky integrate-and-fire” neuron model
- Make the model more complicated — but also more realistic — by adding more physiologically-inspired details

Section 1: The Linear Integrate-and-Fire Neuron

How does a neuron spike?

A neuron charges and discharges an electric field across its cell membrane. The state of this electric field can be described by the *membrane potential*. The membrane potential rises due to excitation of the neuron, and when it reaches a threshold a spike occurs. The potential resets, and must rise to a threshold again before the next spike occurs.

One of the simplest models of spiking neuron behavior is the linear integrate-and-fire model neuron. In this model, the neuron increases its membrane potential V_m over time in response to excitatory input currents I scaled by some factor α:

$$dV_m = \alpha I \tag{125}$$

Once V_m reaches a threshold value a spike is produced, V_m is reset to a starting value, and the process continues.

Here, we will take the starting and threshold potentials as 0 and 1, respectively. So, for example, if $\alpha I = 0.1$ is constant—that is, the input current is constant—then $dV_m = 0.1$, and at each timestep the membrane potential V_m increases by 0.1 until after $(1 - 0)/0.1 = 10$ timesteps it reaches the threshold and resets to $V_m = 0$, and so on.

Note that we define the membrane potential V_m as a scalar: a single real (or floating point) number. However, a biological neuron's membrane potential will not be exactly constant at all points on its cell membrane at a given time. We could capture this variation with a more complex model (e.g. with more numbers). Do we need to?

Spike Inputs

Unlike in the simple example above, where $\alpha I = 0.1$, the input current is generally not constant. Physical inputs tend to vary with time. We can describe this variation with a distribution.

We'll assume the input current I over a timestep is due to equal contributions from a non-negative (≥ 0) integer number of input spikes arriving in that timestep. Our model neuron might integrate currents from 3 input spikes in one timestep, and 7 spikes in the next timestep. We should see similar behavior when sampling from our distribution.

Given no other information about the input neurons, we will also assume that the distribution has a mean (i.e. mean rate, or number of spikes received per timestep), and that the spiking events of the input neuron(s) are independent in time. Are these reasonable assumptions in the context of real neurons?

A suitable distribution given these assumptions is the Poisson distribution, which we'll use to model I:

$$I \sim \text{Poisson}(\lambda) \tag{126}$$

where λ is the mean of the distribution: the average rate of spikes received per timestep.

Practice in Jupyter Notebook

Section 2: Inhibitory signals

Our linear integrate-and-fire neuron from the previous section was indeed able to produce spikes. However, our ISI histogram doesn't look much like empirical ISI histograms seen in Tutorial 1, which had an exponential-like shape. What is our model neuron missing, given that it doesn't behave like a real neuron?

In the previous model we only considered excitatory behavior – the only way the membrane potential could decrease was upon a spike event. We know, however, that there are other factors that can drive V_m down. First is the natural tendency of the neuron to return to some steady state or resting potential. We can update our previous model as follows:

$$dV_m = -\beta V_m + \alpha I \tag{127}$$

where V_m is the current membrane potential and β is some leakage factor. This is a basic form of the popular Leaky Integrate-and-Fire model neuron (for a more detailed discussion of the LIF Neuron, see Bonus Section 2 and the Biological Neuron Models day later in this course).

We also know that in addition to excitatory presynaptic neurons, we can have inhibitory presynaptic neurons as well. We can model these inhibitory neurons with another Poisson random variable:

$$\begin{aligned} I &= I_{\text{exc}} - I_{\text{inh}} \\ I_{\text{exc}} &\sim \text{Poisson}(\lambda_{\text{exc}}) \\ I_{\text{inh}} &\sim \text{Poisson}(\lambda_{\text{inh}}) \end{aligned} \tag{128}$$

where λ_{exc} and λ_{inh} are the average spike rates (per timestep) of the excitatory and inhibitory presynaptic neurons, respectively.

Practice in Jupyter Notebook

▼ "Why" models

Objectives:

To understand why different spiking behaviors may be beneficial, we will learn about the concept of entropy. Specifically, we will:

- Write code to compute formula for entropy, a measure of information
- Compute the entropy of a number of toy distributions
- Compute the entropy of spiking activity from the Steinmetz dataset

Section 1: Optimization and Information

> What is the optimal way for a neuron to fire in order to maximize its ability to communicate information?

In order to explore this question, we first need to have a quantifiable measure for information. Shannon introduced the concept of entropy to do just that, and defined it as

$$H_b(X) = -\sum_{x \in X} p(x) \log_b p(x) \tag{131}$$

where H is entropy measured in units of base b and $p(x)$ is the probability of observing the event x from the set of all possible events in X. See the Bonus Section 1 for a more detailed look at how this equation was derived.

The most common base of measuring entropy is $b = 2$, so we often talk about *bits* of information, though other bases are used as well (e.g. when $b = e$ we call the units *nats*).

- discrete probability distributions - probability mass functions (PMF)

  p(xi) equals the i-th value in an array, and mass refers to how much of the distribution is contained at that value.

  uniform distribution

  If we split the probability mass among even more points, the entropy continues to increase. Let's derive the general form for N points of equal mass, where $p_i = p = 1/N$:

$$\begin{aligned} -\sum_i p_i \log_b p_i &= -\sum_i \frac{1}{N} \log_b \frac{1}{N} \\ &= -\log_b \frac{1}{N} \\ &= \log_b N \end{aligned} \tag{132}$$

  If we have N discrete points, the *uniform distribution* (where all points have equal mass) is the distribution with the highest entropy: $\log_b N$. This upper bound on entropy is useful when considering binning strategies, as any estimate of entropy over N discrete points (or bins) must be in the interval $[0, \log_b N]$.

Section 2: Information, neurons, and spikes

Entropy of different distributions

If our neuron has a fixed budget, what ISI distribution should it express (all else being equal) to maximize the information content of its outputs?

Bonus: The foundations for Entropy

In his foundational 1948 paper on information theory, Claude Shannon began with three criteria for a function H defining the entropy of a discrete distribution of probability masses $p_i \in p(X)$ over the points $x_i \in X$:

1. H should be continuous in the p_i.

- That is, H should change smoothly in response to smooth changes to the mass p_i on each point x_i.

2. If all the points have equal shares of the probability mass, $p_i = 1/N$, H should be a non-decreasing function of N.

- That is, if X_N is the support with N discrete points and $p(x \in X_N)$ assigns constant mass to each point, then $H(X_1) < H(X_2) < H(X_3) < \dots$

3. H should be preserved by (invariant to) the equivalent (de)composition of distributions.

- For example (from Shannon's paper) if we have a discrete distribution over three points with masses $(\frac{1}{2}, \frac{1}{3}, \frac{1}{6})$, then their entropy can be represented in terms of a direct choice between the three and calculated $H(\frac{1}{2}, \frac{1}{3}, \frac{1}{6})$. However, it could also be represented in terms of a series of two choices:
  1. either we sample the point with mass $1/2$ or not (*not* is the other $1/2$, whose subdivisions are not given in the first choice),
  2. if (with probability $1/2$) we *don't* sample the first point, we sample one of the two remaining points, masses $1/3$ and $1/6$.

  Thus in this case we require that $H(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}) = H(\frac{1}{2}, \frac{1}{2}) + \frac{1}{2} H(\frac{1}{3}, \frac{1}{6})$

There is a unique function (up to a linear scaling factor) which satisfies these 3 requirements:

$$H_b(X) = -\sum_{x \in X} p(x) \log_b p(x) \tag{135}$$

Where the base of the logarithm $b > 1$ controls the units of entropy. The two most common cases are $b = 2$ for units of *bits*, and $b = e$ for *nats*.

We can view this function as the expectation of the self-information over a distribution:

$$\begin{aligned} H_b(X) &= \mathbb{E}_{x \in X} \left[I_b(x) \right] \\ I_b(x) &= -\log_b p(x) \end{aligned} \tag{136}$$

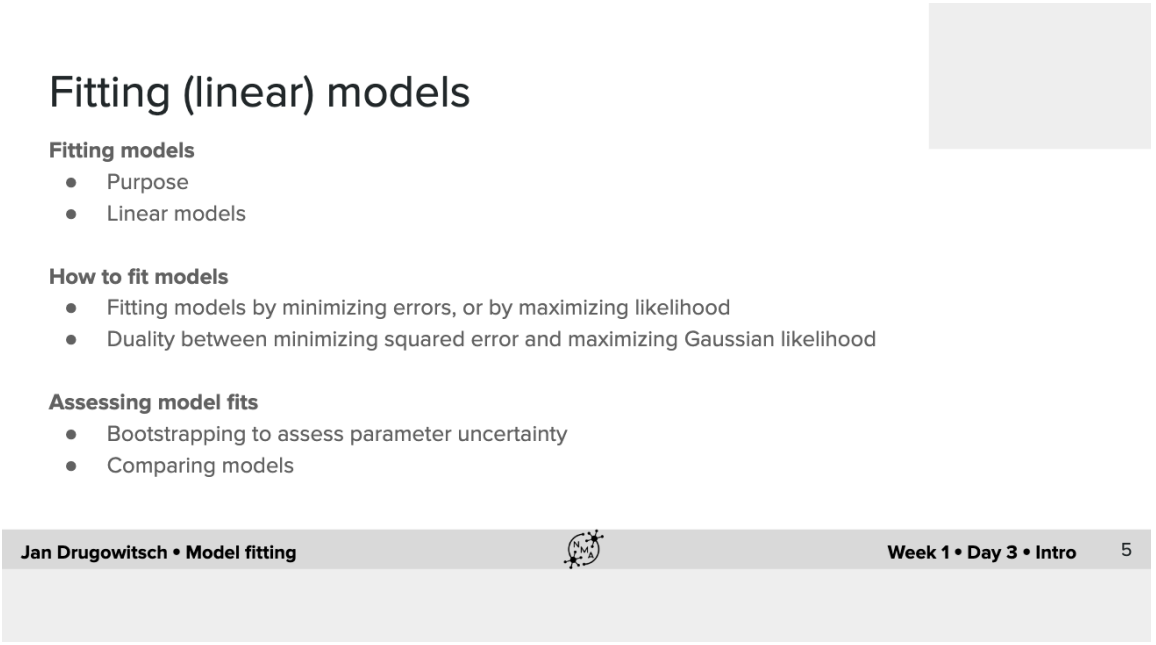
Self-information is just the negative logarithm of probability, and is a measure of how surprising an event sampled from the distribution would be. Events with $p(x) = 1$ are certain to occur, and their self-information is zero (as is the entropy of the distribution they compose) meaning they are totally unsurprising. The smaller the probability of an event, the higher its self-information, and the more surprising the event would be to observe.

Machine Learning

Model Fitting

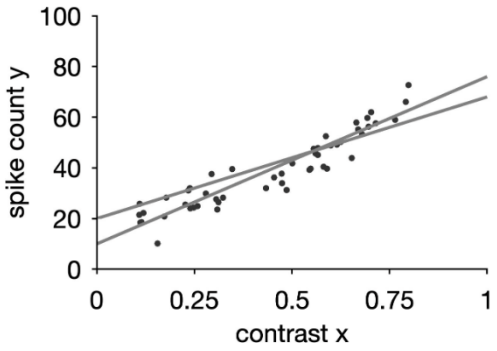
▼ Intro

- [ ] Does my model capture the behavior of my participant or its neural activity?
- [ ] Does the data support my model against alternatives?
- [ ] Which component in the model is needed?
- [ ] Do parameters in the model vary systematically between two subject populations?



> Why we fit models & linear models(**parameters are linear rather than contrasts**)

A simple linear model



Simple model

spike count ~ increases linearly with contrast

$$y \approx \theta_0 + \theta_1 x$$

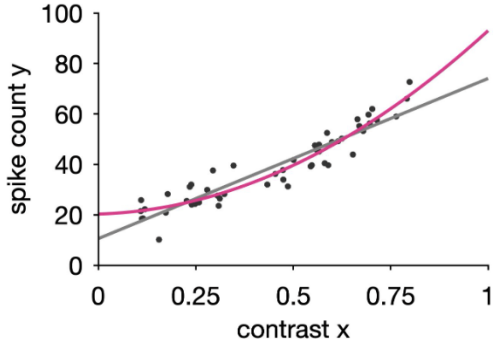
intercept slope

What is the best set of parameters?

How do we measure goodness-of-fit?

How do we find the best-fitting parameters?

Purpose of model fitting



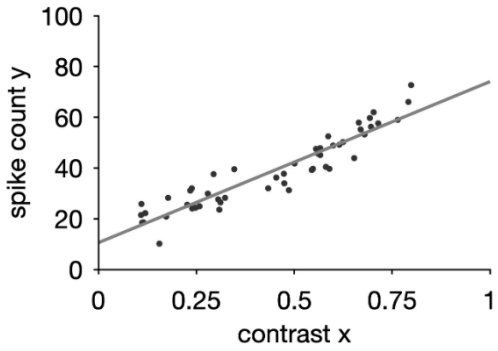
Validation: generate new data
check on held-out data

Prediction: behavior outside of data

Interpret: e.g., spike count ~ contrast? ($\theta_0 \neq 0$?)
(simple models only)

Compare: fits across different models

Linear model can be more complex



spike count ~ increases linearly with contrast

$$y \approx \theta_0 + \theta_1 x$$

intercept slope

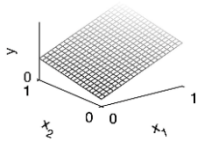
Linear models in general

Assume multiple inputs, one for each stimulus feature (e.g., orientation, contrast, etc.)

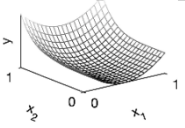
$$\boldsymbol{x} = (x_1, x_2, \dots)^T$$

(Simple) linear model
defines (hyper)plane in $\boldsymbol{x}$

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$



Can be non-linear in inputs
e.g., $y = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^4 + \dots$



More generally,

$$y = \sum_i \theta_i \phi_i(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x})$$

linear in parameters $\boldsymbol{\theta}$, *not* (necessarily) inputs $\boldsymbol{x}$

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} 1 \\ \phi_1(\boldsymbol{x}) \\ \phi_2(\boldsymbol{x}) \\ \vdots \end{pmatrix}$$

How to fit models

Two philosophies for fitting models

Models as functions (e.g., Day 2) $y = f(x; \theta)$

Aim: find model with small errors

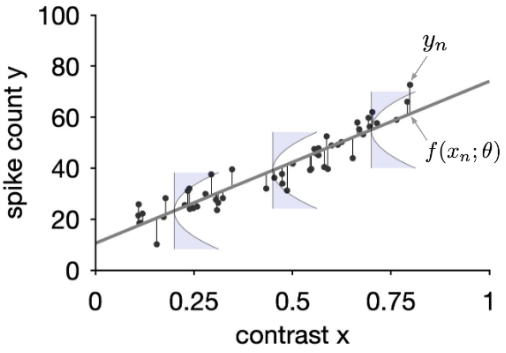
Models as generators $y_{\text{measured}} = f(x; \theta) + \eta$

noise from some distribution

Aim: find model that assigns high probability to the data

Supports richer set of statements about models!

Fitting models by minimizing squared errors



Mean squared error (MSE)

Average squared difference between data and model prediction

$$\text{MSE}(\theta) = \frac{1}{N} \sum_{n=1}^{N} (y_n - f(x_n; \theta))^2$$

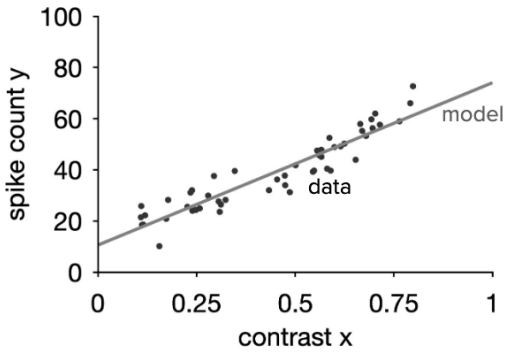
measured

model prediction

Best-fitting parameters

$$\hat{\theta}_{\text{MSE}} = \underset{\theta}{\text{argmin}}\, \text{MSE}(\theta)$$

Generative perspective on model fitting



Generative perspective
Model assumed to "generate" observed data

data ~ model prediction + noise

what we can't control
(e.g., measurement noise)

what we don't care about
(e.g., deviation from mean firing rate)

Likelihood function

$$p\left(\text{data}|\text{parameters } \theta\right) = \mathcal{L}\left(\theta|\text{data}\right)$$

"How likely is data for given parameters?"

Fitting models by maximum likelihood

Aim of maximum likelihood (ML) fits
Find parameters that make data most likely

$$\hat{\theta}_{\text{ML}} = \underset{\theta}{\text{argmax}}\, \mathcal{L}\left(\theta|\text{data}\right) = \underset{\theta}{\text{argmax}} \log \mathcal{L}\left(\theta|\text{data}\right)$$

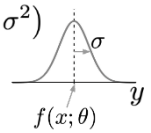
ML for independent trials
If trials are independent, then $\mathcal{L}\left(\theta|\text{data}\right) = \prod_n \mathcal{L}\left(\theta|\text{data}_n\right)$
As a result,

$$\hat{\theta}_{\text{ML}} = \underset{\theta}{\text{argmax}} \prod_n \mathcal{L}\left(\theta|\text{data}_n\right) = \underset{\theta}{\text{argmax}} \sum_n \log \mathcal{L}\left(\theta|\text{data}_n\right)$$

Maximum likelihood with Gaussian noise

Gaussian noise with variance σ^2

$$y = f(x;\theta) + \eta \quad \Leftrightarrow \quad p(y|x,\theta) = \mathcal{L}(\theta|x,y) = \mathcal{N}\left(y|f(x;\theta), \sigma^2\right)$$



trials are independent

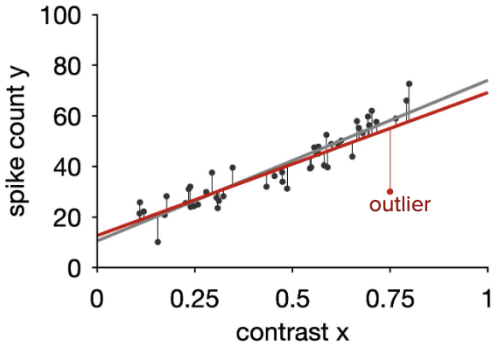
$$\log \mathcal{L}(\theta|X,Y) = \sum_n \log \mathcal{L}(\theta|x_n, y_n)$$

$$= -\frac{N}{2\sigma^2}\frac{1}{N}\sum_n \left(y_n - f(x_n;\theta)\right)^2 + \text{const.} \; = -\frac{N}{2\sigma^2}\text{MSE}(\theta) + \text{const.}$$

linear model with Gaussian noise — independent of θ

maximizing likelihood with Gaussian noise = minimizing mean squared error

Gaussian noise: sensitivity to outliers



Gaussian noise: quadratic error function

- Larger errors weigh more strongly
- Fits sensitive to outliers

Fitting linear models

Linear model

$$y = f(\boldsymbol{x}; \boldsymbol{\theta}) + \eta = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}) + \eta$$

Log-likelihood with Gaussian noise

$$\log \mathcal{L}(\boldsymbol{\theta} | \boldsymbol{X}, \boldsymbol{y}) = -\frac{N}{2\sigma^2} \frac{1}{N} \sum_n \left(y_n - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\right)^2 + \text{const.}$$

Properties

- Single most important statistical model
- Likelihood quadratic in $\boldsymbol{\theta}$ (concave function) → easy to find best-fitting parameters
- Analytic expression for ML estimate (see tutorial)

What we have learned

Two philosophies for fitting models

Minimizing error
Maximizing likelihood

Minimizing mean squared error = maximizing likelihood with Gaussian noise

Squared error makes fit sensitive to outliers

Applied to linear model

Easy to find best-fitting parameters,
computable by analytical expression

Assessing model fits

Parameter uncertainty

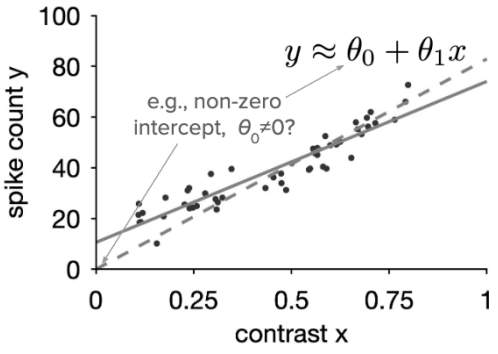
Limited data → multiple parameter values $\boldsymbol{\theta}$ might explain the data about equally well. Reflects *inherent uncertainty* about best-fitting parameters.

Example uses

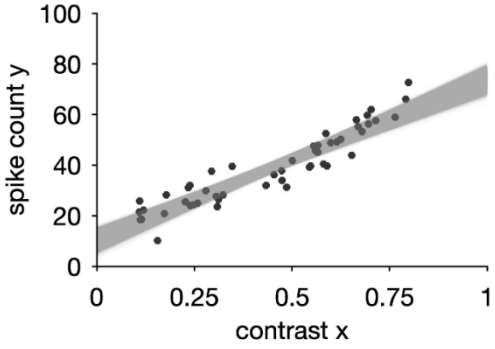
- How well does data constrain parameters?
- Are parameters significantly non-zero (i.e., relevant)?

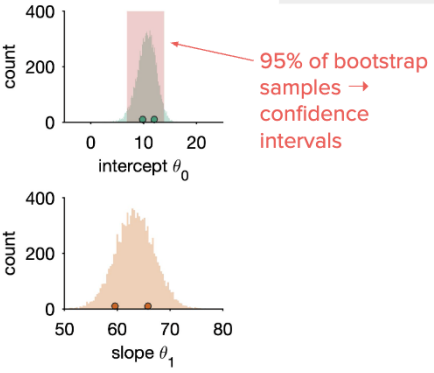
Linear models can assess uncertainty through standard statistics (not discussed further).

Generally assess parameter uncertainty through *bootstrapping*.

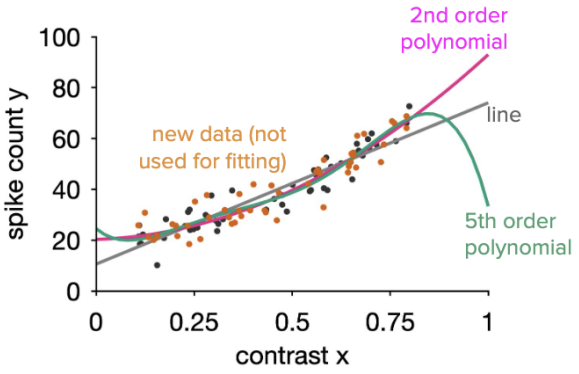


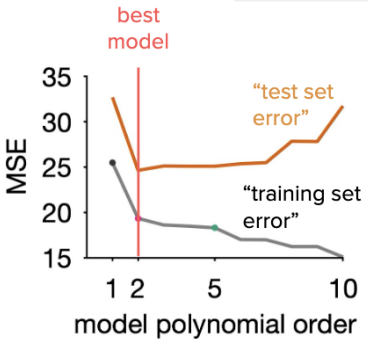
Assessing uncertainty by bootstrap



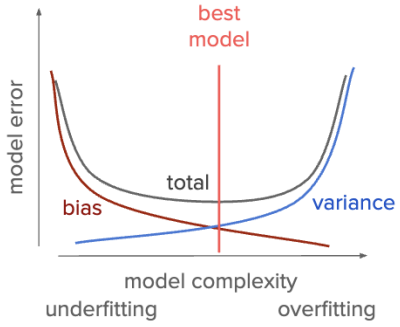


Fitting & comparing multiple models





Bias-variance trade-off



Bias
Low model complexity: systematic deviation from structure underlying data (underfitting)

Variance
High model complexity: capturing variability beyond the structure underlying data (i.e., noise; overfitting)

Total error = bias + variance

Best model: balances bias / variance

Two philosophies for comparing models

Goodness of fit
(popular in statistics)

Compute likelihood of fitted model, and correct for number of parameters, compare goodness of fits.
Good models use few parameters to produce good fits (e.g, Day 2)

Cross validation
(popular in machine learning)

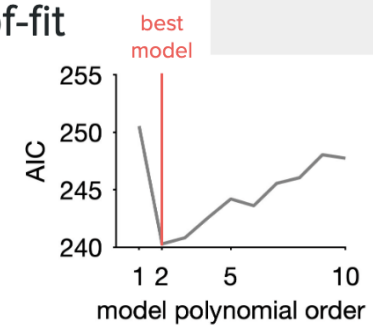
Fit model to some data (training set), then check how well it predicts new data (test set).

Model comparison by goodness-of-fit

Example: Akaike Information Criterion (AIC)
(lower is better)

$$\mathrm{AIC} = 2k - 2\log\mathcal{L}\left(\hat{\theta}_{\mathrm{ML}}|X,Y\right)$$

k: number of parameters



Pros Easy to compute

Cons Strong assumptions about the model's structure

Alternatives

Other information criteria: BIC / DIC / ..., differ in how they measure model complexity

Bayesian model comparison: implicit complexity penalty by averaging over model parameters

...

Model comparison by cross-validation

Compare models by prediction error on held-out data

Pros Minimal assumptions about data
Widely applicable

Cons Requires lots of data
Computationally expensive
Little sensitivity to small model differences

More details: today's tutorial

▼ Linear regression with MSE

Objectives:

- Learn how to calculate the mean-squared error (MSE)
- Explore how model parameters (slope) influence the MSE
- Learn how to find the optimal model parameter using least-squares optimization

While visually exploring several estimates can be instructive, it's not the most efficient for finding the best estimate to fit our data. Another technique we can use is to choose a reasonable range of parameter values and compute the MSE at several values in that interval. This allows us to plot the error against the parameter value (this is also called an **error landscape**, especially when we deal with more than one parameter). We can select the final $\hat{\theta}$ ($\hat{\theta}_{\mathrm{MSE}}$) as the one which results in the lowest error.

We can do this by minimizing the cost function. Mean squared error is a convex objective function, therefore we can compute its minimum using calculus. Please see video or Bonus Section 1 for this derivation! After computing the minimum, we find that:

$$\hat{\theta} = \frac{\mathbf{x}^\top \mathbf{y}}{\mathbf{x}^\top \mathbf{x}} \tag{141}$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors of data points.

This is known as solving the normal equations. For different ways of obtaining the solution, see the notes on Least Squares Optimization by Eero Simoncelli.

Bonus Section 1: Least Squares Optimization Derivation

We will outline here the derivation of the least squares solution.

We first set the derivative of the error expression with respect to θ equal to zero,

$$\frac{d}{d\theta}\frac{1}{N}\sum_{i=1}^{N}(y_i - \theta x_i)^2 = 0$$
$$\frac{1}{N}\sum_{i=1}^{N} -2x_i(y_i - \theta x_i) = 0 \tag{143}$$

where we used the chain rule. Now solving for θ, we obtain an optimal value of:

$$\hat{\theta} = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2} \tag{144}$$

Which we can write in vector notation as:

$$\hat{\theta} = \frac{\mathbf{x}^\top \mathbf{y}}{\mathbf{x}^\top \mathbf{x}} \tag{145}$$

This is known as solving the *normal equations*. For different ways of obtaining the solution, see the notes on Least Squares Optimization by Eero Simoncelli.

▼ Linear regression with MLE

Maximum Likelihood Estimation (MLE)

Gaussian/Normal distribution

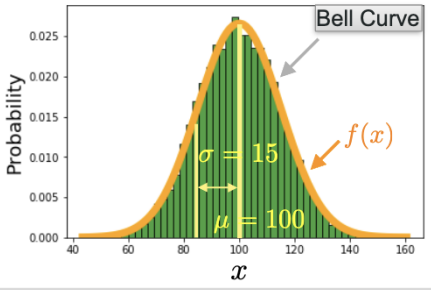
Data can be "distributed" (spread out) in different ways. But there are many cases where the data tends to be around a central value with no bias left or right, and it gets close to a "Gaussian Distribution" like this:

A Gaussian distribution function is defined by

mean parameter μ standard deviation $\sigma > 0$

variance σ^2

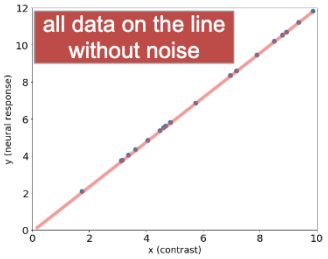
probability density function $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$



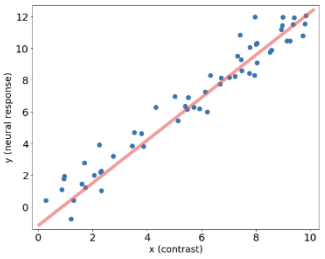
Maximum Likelihood Estimation (MLE)

If we generate y from x using

$$y = \theta \times x$$



However this is the data we observe, with noise.



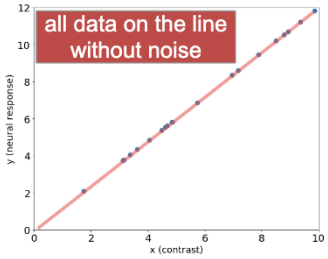
what we don't care about
(e.g., deviation from mean firing rate)

what we can't control
(e.g., measurement noise)

Maximum Likelihood Estimation (MLE)

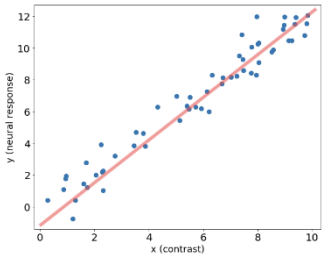
If we generate y from x using

$$y = \theta \times x$$



Model data with noise.

$$y = \theta \times x + \eta$$ ← Noise



what we don't care about
(e.g., deviation from mean firing rate)

what we can't control
(e.g., measurement noise)

$$\eta \sim \mathcal{N}(0, \sigma^2)$$

$$\downarrow$$

$$y \sim \mathcal{N}(\theta x, \sigma^2)$$

Maximum Likelihood Estimation (MLE)

We consider the linear regression model with a Gaussian distributed noise η.

$$y = \theta \times x + \eta$$

- y: neural response
- θ: linear weight
- x: contrast
- η: Gaussian noise, $\eta \sim \mathcal{N}(0, \sigma^2)$

the larger σ^2 is, the noisier the data is

encoding model

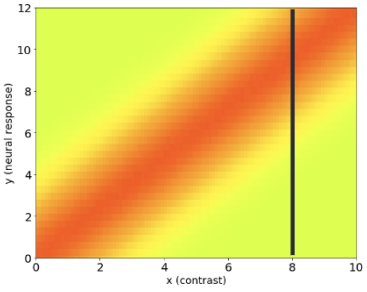
$$p(y|x,\theta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(y-\theta x)^2}$$

$$\text{mean}(y) = \theta x$$

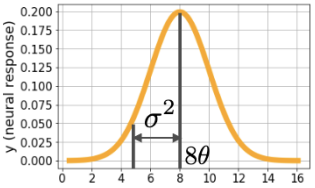
$$\text{var}(y) = \sigma^2$$

Maximum Likelihood Estimation (MLE)

$p(y|x,\theta)$



$p(y|x=8,\theta)$



encoding model

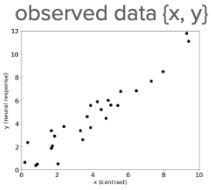
$$p(y|x=8,\theta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(y-8\theta)^2}$$

$$\text{mean}(y) = 8\theta$$

$$\text{var}(y) = \sigma^2$$

Maximum Likelihood Estimation (MLE)

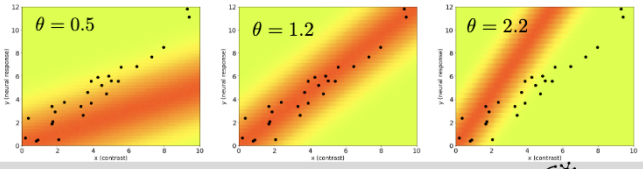
observed data {x, y}



Likelihood: which θ leads to the Gaussian distribution that most likely matches the data points $\{x, y\}$?

$$\mathcal{L}(\theta|x,y) = p(y|x,\theta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(y-\theta x)^2}$$

$p(y|x,\theta=0.5)$ $p(y|x,\theta=1.2)$ $p(y|x,\theta=2.2)$



$$\mathcal{L}(\theta=1.2) > \mathcal{L}(\theta=0.5) > \mathcal{L}(\theta=2.2)$$

Therefore, we could calculate the optimal θ by maximizing the likelihood.

Maximum Likelihood Estimation (MLE)

Empirically, we maximize the log likelihood

$$\begin{aligned}
\log \mathcal{L}(\theta|x,y) &= \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(y_i-\theta x_i)^2} \\
&= \sum_{i=1}^{N} \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(y_i-\theta x_i)^2} \\
&= \sum_{i=1}^{N} \log \frac{1}{\sigma\sqrt{2\pi}} + \sum_{i=1}^{N} \log e^{-\frac{1}{2\sigma^2}(y_i-\theta x_i)^2} \\
&= -\frac{N}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \theta x_i)^2
\end{aligned}$$

Maximum Likelihood Estimation (MLE)

Empirically, we maximize the log likelihood

$$\log \mathcal{L}(\theta|x,y) = -\frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \theta x_i)^2$$

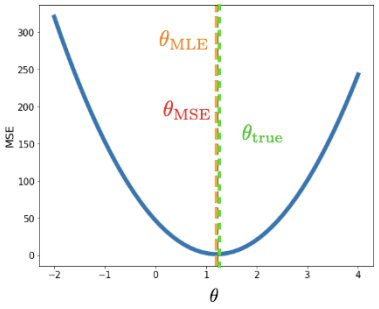
Nx MSE $= \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

Differentiate and set to zero

$$\frac{\partial \log \mathcal{L}(\theta|x,y)}{\partial \theta} = \frac{1}{\sigma^2}\sum_{i=1}^{N}(y_i - \theta x_i)x_i = 0$$

$$\Rightarrow \theta_{\text{MLE}} = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2}$$

(the same as θ_{MSE} when $p(y|x,\theta)$ is Gaussian)



Anqi Wu • Model Fitting

Week 1 • Day 3 • Tutorial 2 22

Practice in Jupyter Notebook

▼ Confidence intervals and bootstrapping

how to gauge how good our estimated model parameters are

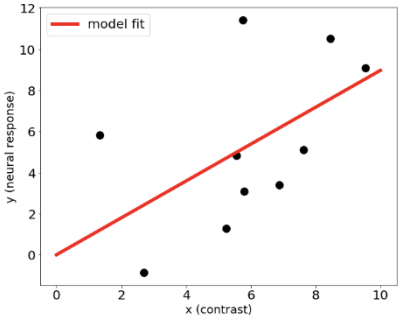
- Learn how to use bootstrapping to generate new sample datasets
- Estimate our model parameter on these new sample datasets
- Quantify the variance of our estimate using confidence intervals

Bootstrap

Bootstrapping is a widely applicable method to assess confidence/uncertainty about estimated parameters, it was originally proposed by Bradley Efron. The idea is to generate many new synthetic datasets from the initial true dataset by randomly sampling from it, then finding estimators for each one of these new datasets, and finally looking at the distribution of all these estimators to quantify our confidence.

Note that each new resampled datasets will be the same size as our original one, with the new data points sampled with replacement i.e. we can repeat the same data point multiple times. Also note that in practice we need a lot of resampled datasets, here we use 2000.

Uncertainty of Parameters



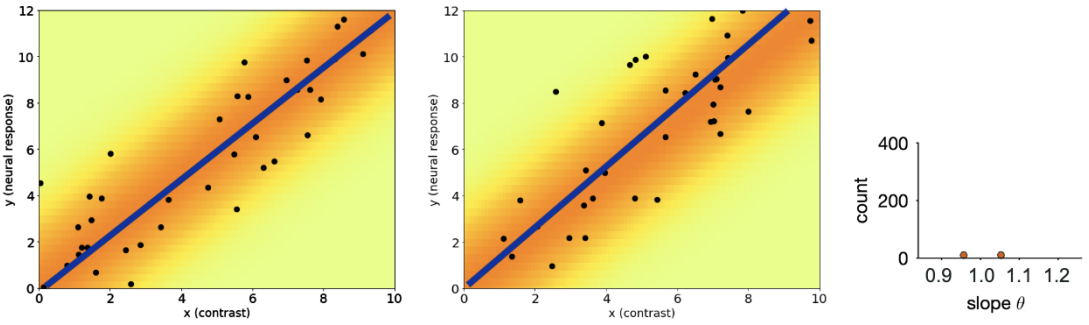
MSE = 0.38

Question: How confident we are about the model fit?

Use resampling to estimate uncertainty thus confidence

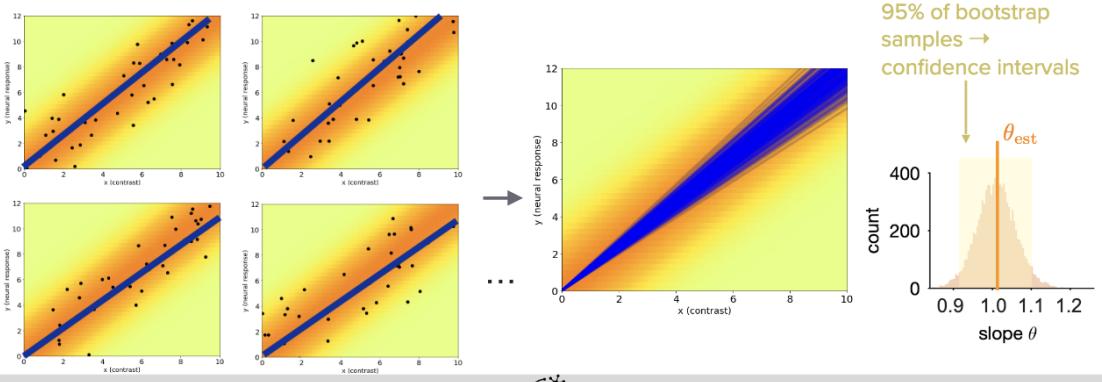
Resampling to Estimate Uncertainty

Knowing the true distribution



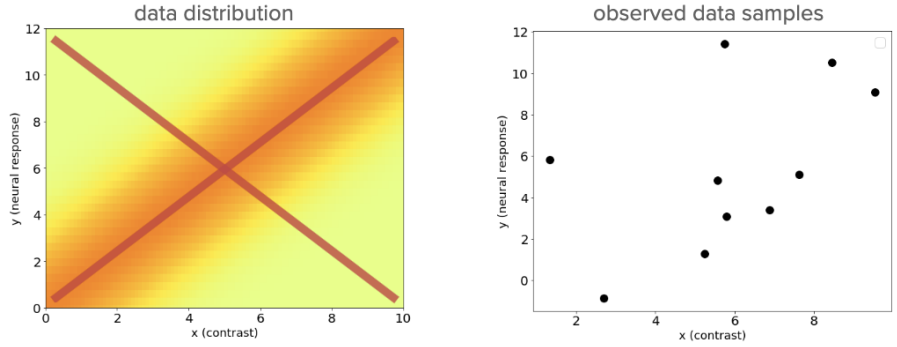
Resampling to Estimate Uncertainty

Knowing the underlying true distribution



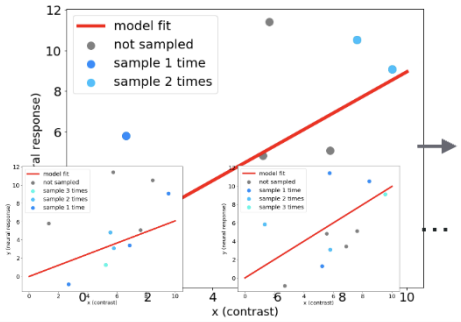
Assessing Uncertainty by Bootstrap

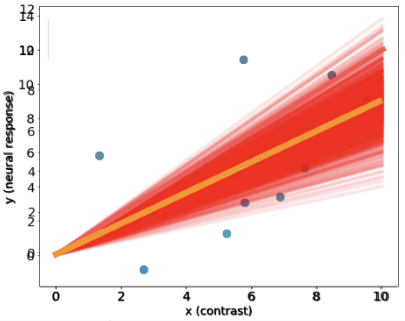
Of course we never know the true population; we only have data samples.



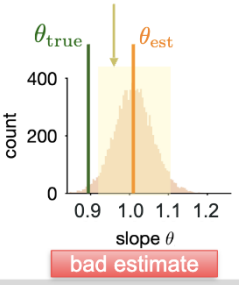
Assessing Uncertainty by Bootstrap

1. Resampling from the observed dataset with replacement; the number of the resampled data points is the same as the number of observed data points.
2. Collect all estimates into a distribution, and analyze the confidence intervals.





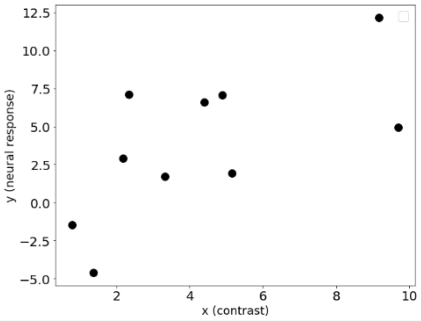
95% of bootstrap samples → confidence intervals

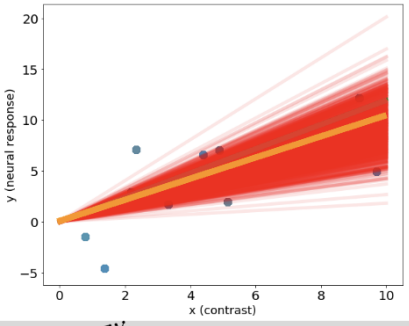


bad estimate

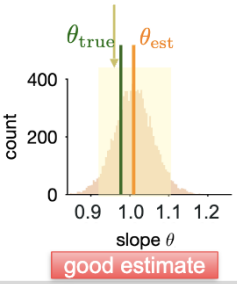
Assessing Uncertainty by Bootstrap

1. Resampling from the observed dataset.
2. Collect all estimates into a distribution, and analyze the confidence intervals.



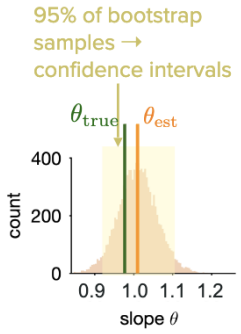


95% of bootstrap samples → confidence intervals



good estimate

Assessing Uncertainty by Bootstrap



- In most of real-world applications, we don't know the ground truth.
- But it's still beneficial to have a distribution rather than a point estimate.
- With distribution and uncertainty, you have more information to make decision.

▼ Multiple Linear Regression and Polynomial Regression

In this tutorial, we will generalize the regression model to incorporate multiple features.

- Learn how to structure inputs for regression using the 'Design Matrix'
- Generalize the MSE for multiple features using the ordinary least squares estimator
- Visualize data and model fit in multiple dimensions
- Fit polynomial regression models of different complexity
- Plot and evaluate the polynomial regression fits

[Simple] Linear model

$$y = \theta_1 \times x + \theta_0$$

neural response: y; linear weight: θ_1; contrast: x; intercept: θ_0

Multiple linear model

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_d x_d$$

neural response: y

Assume multiple stimulus features (e.g., orientation, contrast, etc.)

More generally, in vector form, for one data point i

$$y_i = \boldsymbol{\theta}^\top \boldsymbol{x}_i \qquad \forall i = 1, ..., N$$

$\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, ...]^\top$ $\quad \boldsymbol{x}_i = [1, x_{i,1}, x_{i,2}, ...]^\top$

Build up to matrix version

$$\boldsymbol{y} = X\boldsymbol{\theta}$$

Index i

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} — \ \boldsymbol{x}_0 \ — \\ — \ \boldsymbol{x}_1 \ — \\ — \ \boldsymbol{x}_2 \ — \\ \vdots \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \end{bmatrix}$$

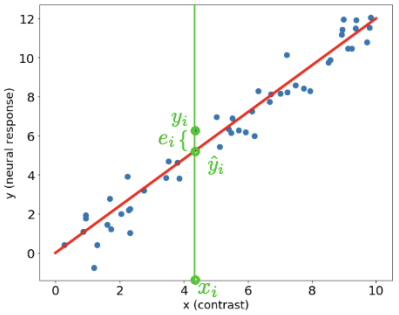
design matrix

Geometric Interpretation of MSE

MSE computes the average error between the model prediction $\hat{y}$ and the true y.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i=1}^{N} e_i^2$$

model prediction; residual; total number of data points; true neural response; index of data points i=1,...,N



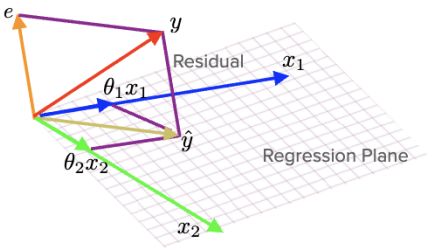
Geometric Interpretation of MSE

2D case without intercept

model predict $\hat{y} = \theta_1 x_1 + \theta_2 x_2$

residual $e = y - \hat{y} = y - (\theta_1 x_1 + \theta_2 x_2)$

minimize $\text{MSE} = (\theta_1 x_1 + \theta_2 x_2 - y)^2 = e^2$



source: https://www.datasciencecentral.com/profiles/blogs/linear-regression-geometry

MSE solution:

the number of data points

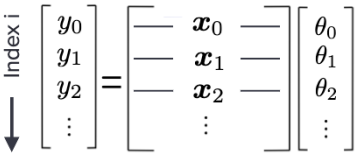
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} ||X\boldsymbol{\theta} - \boldsymbol{y}||_2^2 = \sum_{i=1}^{N} (y_i - \boldsymbol{\theta}^\top \boldsymbol{x}_i)^2$$

Differentiate and set to zero

$$\frac{\partial ||X\boldsymbol{\theta} - \boldsymbol{y}||_2^2}{\partial \boldsymbol{\theta}} = 2X^\top (X\boldsymbol{\theta} - \boldsymbol{y}) = 0$$

$$\Rightarrow \boldsymbol{\theta}_{\text{MSE}} = (X^\top X)^{-1} X^\top \boldsymbol{y}$$

$$\boldsymbol{y} = X\boldsymbol{\theta}$$



MLE solution:

$$\log \mathcal{L}(\boldsymbol{\theta}|X, \boldsymbol{y}) = \log p(\boldsymbol{y}|X, \boldsymbol{\theta}) = \log \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} \log \frac{1}{(\sigma\sqrt{2\pi})^d} e^{-\frac{1}{2\sigma^2}(y_i - \boldsymbol{\theta}^\top \boldsymbol{x}_i)^2}$$

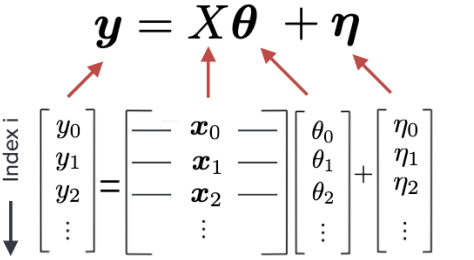
the number of input features

$$= \sum_{i=1}^{N} \log \frac{1}{(\sigma\sqrt{2\pi})^d} + \sum_{i=1}^{N} \log e^{-\frac{1}{2\sigma^2}(y_i - \boldsymbol{\theta}^\top \boldsymbol{x}_i)^2}$$

$$= -\frac{Nd}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \boldsymbol{\theta}^\top \boldsymbol{x}_i)^2$$

$$= -\frac{Nd}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} (\boldsymbol{y} - X\boldsymbol{\theta})^\top (\boldsymbol{y} - X\boldsymbol{\theta})$$

$$\boldsymbol{y} = X\boldsymbol{\theta} + \boldsymbol{\eta}$$



MLE solution:

$$\log \mathcal{L}(\boldsymbol{\theta}|X,\boldsymbol{y}) = \log p(\boldsymbol{y}|X,\boldsymbol{\theta}) = -\frac{Nd}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}(\boldsymbol{y}-X\boldsymbol{\theta})^\top(\boldsymbol{y}-X\boldsymbol{\theta})$$

Differentiate and set to zero

$$\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|X,\boldsymbol{y})}{\partial \boldsymbol{\theta}} = -\frac{1}{\sigma^2}X^\top(X\boldsymbol{\theta}-\boldsymbol{y}) = 0$$

$$\Rightarrow \boldsymbol{\theta}_{\text{MLE}} = (X^\top X)^{-1}X^\top \boldsymbol{y}$$

(same as MSE when the noise is Gaussian)

$$\boldsymbol{y} = X\boldsymbol{\theta} + \boldsymbol{\eta}$$

Index i

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} — & \boldsymbol{x}_0 & — \\ — & \boldsymbol{x}_1 & — \\ — & \boldsymbol{x}_2 & — \\ & \vdots & \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \end{bmatrix} + \begin{bmatrix} \eta_0 \\ \eta_1 \\ \eta_2 \\ \vdots \end{bmatrix}$$

design matrix

$\eta \sim \mathcal{N}(0, \sigma^2)$

Anqi Wu • Model Fitting — Week 1 • Day 3 • Tutorial 4 — 41

Polynomial Regression

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_P x^P = \sum_{p=0}^{P} \theta_p x^p$$

More generally, in vector form, for one data point i

$$y_i = \boldsymbol{\theta}^\top \boldsymbol{x}_i \qquad \forall i = 1, ..., N$$

$\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, ...]^\top$

The only difference

$\boldsymbol{x}_i = [1, x_i, x_i^2, ...]^\top$

$\boldsymbol{x}_i = [1, x_{i,1}, x_{i,2}, ...]^\top$ linear

Build up to matrix version

$$\boldsymbol{y} = X\boldsymbol{\theta}$$

Index i

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} — & \boldsymbol{x}_0 & — \\ — & \boldsymbol{x}_1 & — \\ — & \boldsymbol{x}_2 & — \\ & \vdots & \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \end{bmatrix}$$

design matrix

Anqi Wu • Model Fitting — Week 1 • Day 3 • Tutorial 4 — 42

▼ Model Selection: Bias-variance trade-off

In this tutorial, we will learn about the bias-variance tradeoff and see it in action using polynomial regression models.

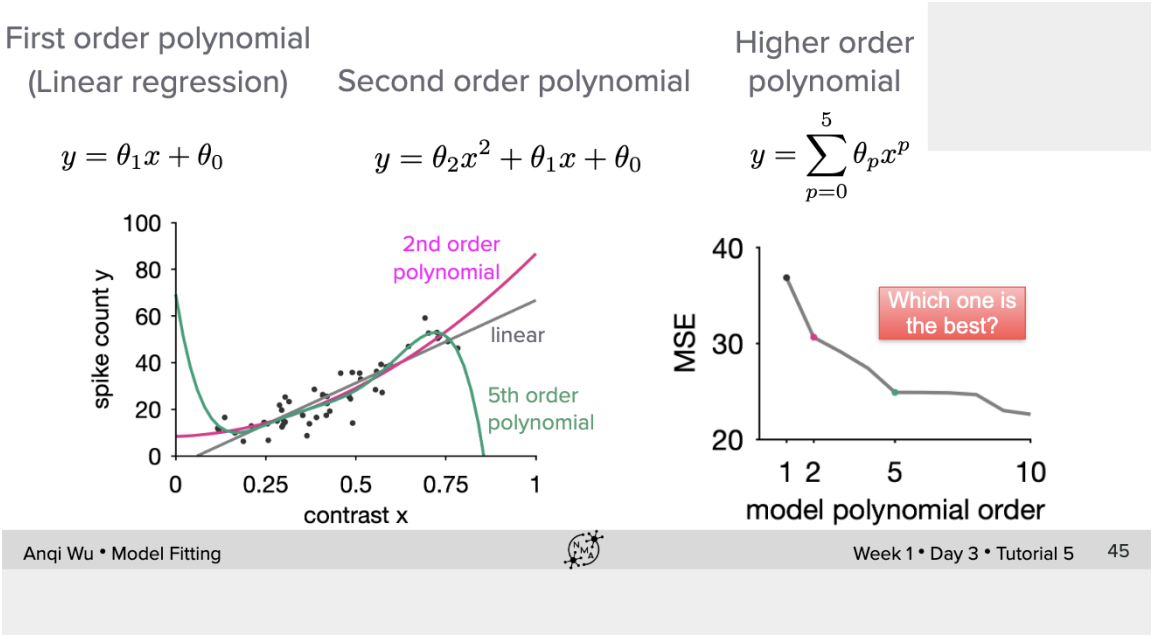
Tutorial objectives:

- Understand difference between test and train data
- Compare train and test error for models of varying complexity

- Understand how bias-variance tradeoff relates to what model we choose

Train vs test data

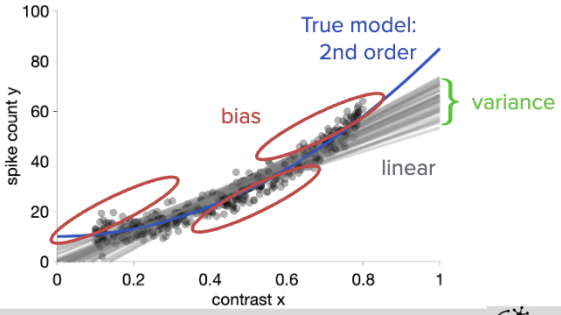
The data used for the fitting procedure for a given model is the **training data**. In tutorial 4, we computed MSE on the training data of our polynomial regression models and compared training MSE across models. An additional important type of data is **test data**. This is held-out data that is not used (in any way) during the fitting procedure. When fitting models, we often want to consider both the train error (the quality of prediction on the training data) and the test error (the quality of prediction on the test data) as we will see in the next section.



Bias and Variance

Bias: systematic deviation from structure underlying data, low model complexity.

Variance: capturing variability in predictions from models trained on different datasets, high model complexity.



Low model complexity

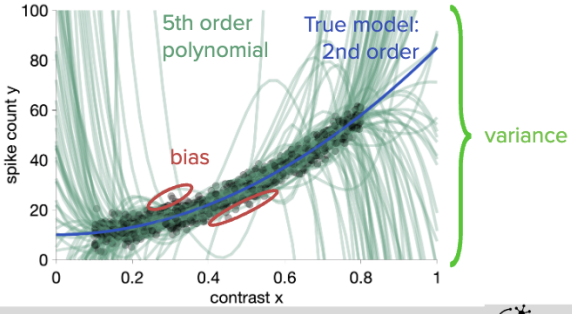
Bias is large: the linear slope always misses many data points

Variance is small: linear slopes are close to each other across samples

Bias and Variance

Bias: systematic deviation from structure underlying data, low model complexity.

Variance: capturing variability in predictions from models trained on different datasets, high model complexity.



High model complexity

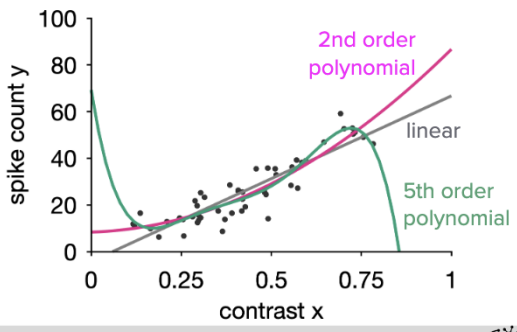
Bias is small: the 5th order polynomials go through almost all data points

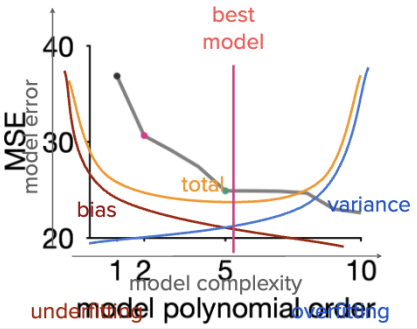
Variance is large: the 5th order polynomials are much different from each other

Bias-variance Trade-off

Total error = **bias** + **variance**

Best model: balances bias / variance



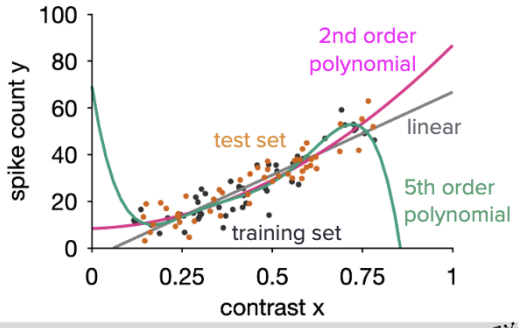


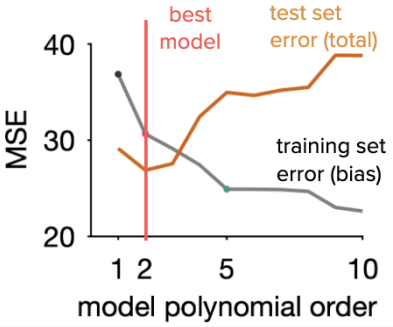
Bias-variance Trade-off

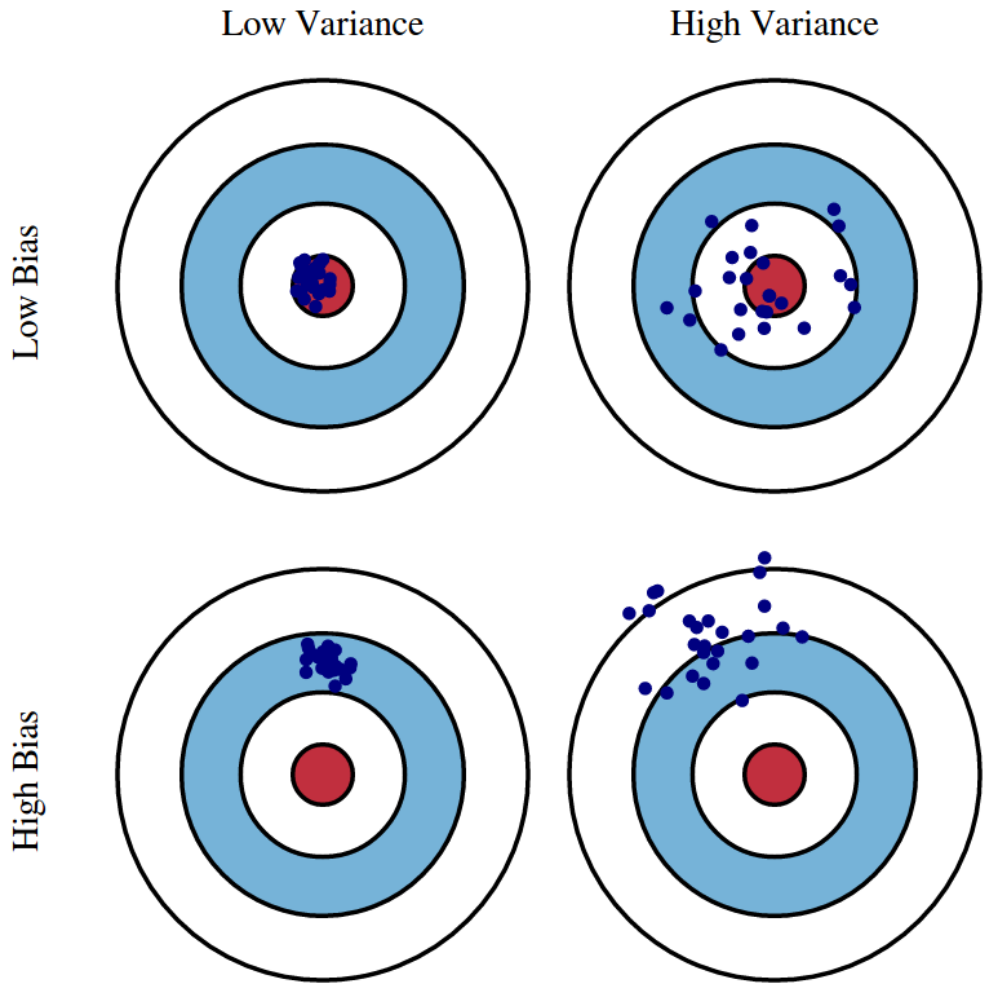
Model evaluation aims at estimating the *generalization error* of the selected model, i.e., how well the selected model performs on unseen data.

Training set: data used for fitting

Test set: unseen data (not used for fitting)







Bonus Exercise: Proof of bias-variance decomposition

Prove the bias-variance decomposition for MSE

$$\mathbb{E}_x\left[\left(y-\hat{y}(x;\theta)\right)^2\right]=\left(\text{Bias}_x[\hat{y}(x;\theta)]\right)^2+\text{Var}_x[\hat{y}(x;\theta)]+\sigma^2 \tag{175}$$

where

$$\text{Bias}_x[\hat{y}(x;\theta)]=\mathbb{E}_x[\hat{y}(x;\theta)]-y \tag{176}$$

and

$$\text{Var}_x[\hat{y}(x;\theta)]=\mathbb{E}_x\left[\hat{y}(x;\theta)^2\right]-\mathrm{E}_x[\hat{y}(x;\theta)]^2 \tag{177}$$

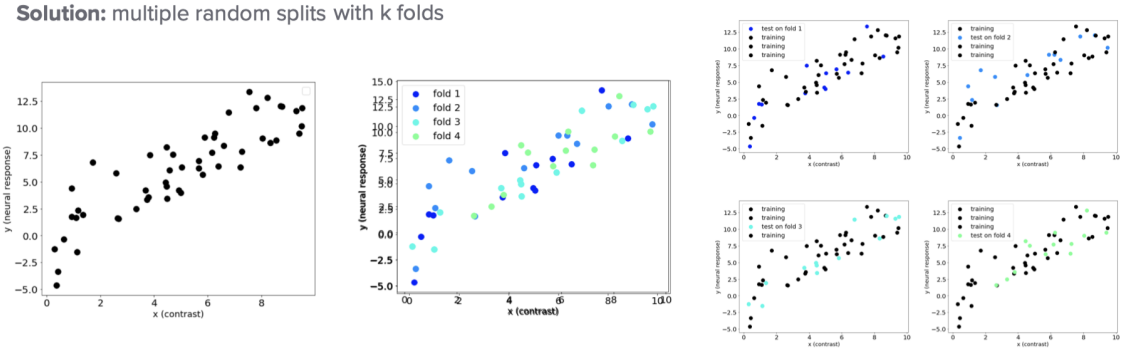
▼ Model Selection: Cross-validation

• Implement cross-validation and use it to compare polynomial regression model

K-fold Cross Validation

Motivation: one random split could result in a biased training-test distribution

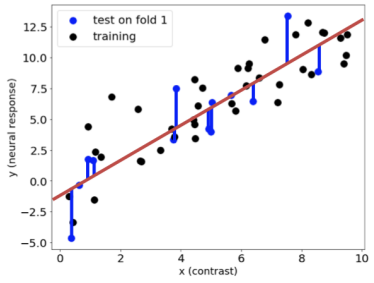
Solution: multiple random splits with k folds



K-fold Cross Validation

Motivation: one random split could result in a biased training-test distribution

Solution: multiple random splits with k folds.



Steps:

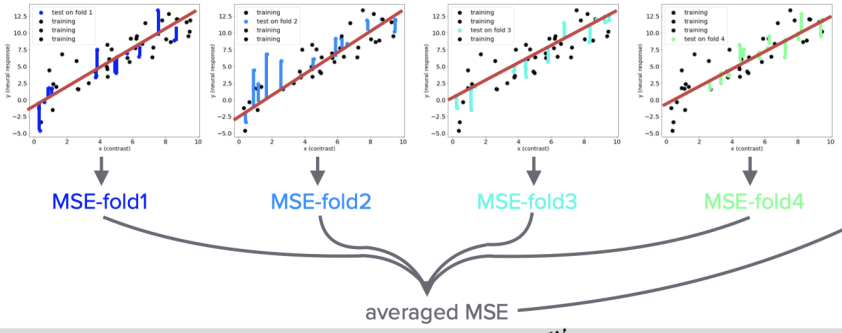
0. Define a model to evaluate, e.g. $y = \sum_{p=0}^{P} \theta_p x^p$ when P=1
1. Take one fold as the test set, the rest as the training set.
2. Fit the model with the training set, and get the optimal θ_p.
3. Estimate MSE on the test set using the above θ_p,

$$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{N_{\mathrm{test}}} \sum_{i=1}^{N_{\mathrm{test}}} \left(\sum_{p=0}^{P} \theta_p x_{\mathrm{test}}^p - y_{\mathrm{test}} \right)^2$$

where $\{x_{\mathrm{test}}, y_{\mathrm{test}}\}_{i=1}^{N_{\mathrm{test}}}$ denotes the test set.

K-fold Cross Validation

Motivation: one random split could result in a biased training-test distribution
Solution: multiple random splits with k folds.



MSE-fold1 MSE-fold2 MSE-fold3 MSE-fold4

averaged MSE

model

$$y = \sum_{p=0}^{P} \theta_p x^p$$

when P=1

K-fold Cross Validation

Motivation: one random split could result in a biased training-test distribution
Solution: multiple random splits with k folds.

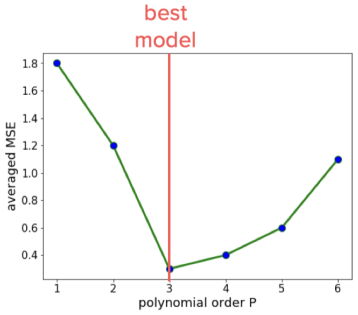
model

$$y = \sum_{p=0}^{P} \theta_p x^p$$

when P=1

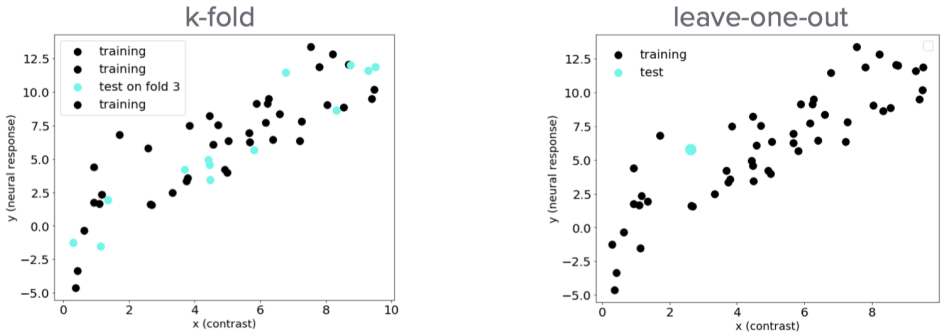
averaged MSE

Repeat the same 4-fold cross validation on other Ps, i.e. P = {2, 3, 4…}, get averaged test MSE



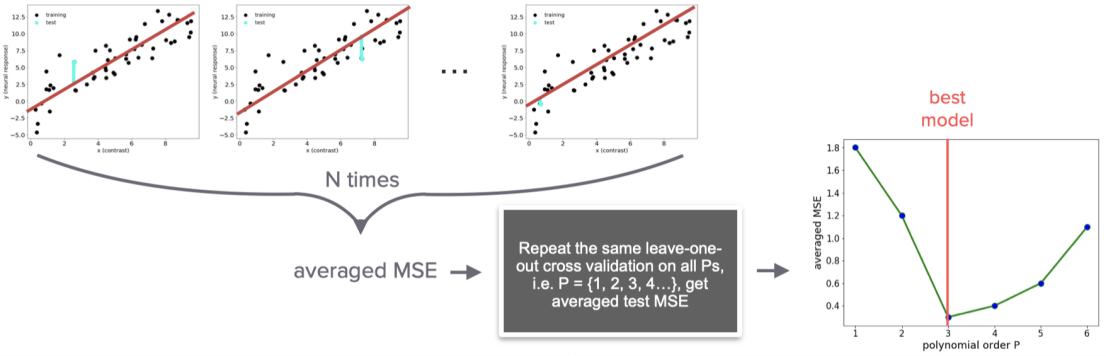
Leave-one-out Cross Validation

Instead of leaving a fold out for test, leaving one data point out.



Leave-one-out Cross Validation

Instead of leaving a fold out for test, leaving one data point out.



We often have a limited amount of data though (especially in neuroscience), so we do not want to further reduce our potential training data by reassigning some as validation. Luckily, we can use **k-fold cross-validation**! In k-fold cross validation, we divide up the training data into k subsets (that are called *folds*, see diagram below), train our model on the first k-1 folds, and then compute error on the last held-out fold. We can then repeat this process k times, once on each k-1 folds of the data. Each of these k instances (which are called *splits*, see diagram below) excludes a different fold from fitting. We then average the error of each of the k trained models on its held-out subset - this is the final measure of performance which we can use to do model selection.

To make this explicit, let's say we have 1000 samples of training data and choose 4-fold cross-validation. Samples 0 - 250 would be subset 1, samples 250 - 500 subset 2, samples 500 - 750 subset 3, and samples 750-1000 subset 4. First, we train an order 3 polynomial regression on subsets 1, 2, 3 and evaluate on subset 4. Next, we train an order 3 polynomial model on subsets 1, 2, 4 and evaluate on subset 3. We continue until we have 4 instances of a trained order 3 polynomial regression model, each with a different subset as held-out data, and average the held-out error from each instance.

We can now compare the error of different models to pick a model that generalizes well to held-out data. We can choose the measure of prediction quality to report error on the held-out subsets to suit our purposes. We will use MSE here but we could also use log likelihood of the data and so on.

As a final step, it is common to retrain this model on all of the training data (without subset divisions) to get our final model that we will evaluate on test data. This approach allows us to evaluate the quality of predictions on new data without sacrificing any of our precious training data.

An important consideration in the choice of model selection method are the relevant biases. If we just fit using MSE on training data, we will generally find that fits get better as we add more parameters because the model will overfit the data, as we saw in Tutorial 5. When using cross-validation, the bias is the other way around. Models with more parameters are more affected by variance so cross-validation will generally prefer models with fewer parameters.

Akaike's Information Criterion (AIC)

In order to choose the best model for a given problem, we can ask how likely the data is under a given model. We want to choose a model that assigns high probability to the data. A commonly used method for model selection that uses this approach is **Akaike's Information Criterion (AIC)**.

Essentially, AIC estimates how much information would be lost if the model predictions were used instead of the true data (the relative information value of the model). We compute the AIC for each model and choose the model with the lowest AIC. Note that AIC only tells us relative qualities, not absolute - we do not know from AIC how good our model is independent of others.

AIC strives for a good tradeoff between overfitting and underfitting by taking into account the complexity of the model and the information lost. AIC is calculated as:

$$\mathrm{AIC} = 2K - 2\log(\mathcal{L}) \tag{179}$$

where K is the number of parameters in your model and $\mathcal{L}$ is the likelihood that the model could have produced the output data.

Now we know what AIC is, we want to use it to pick between our polynomial regression models. We haven't been thinking in terms of likelihoods though - so how will we calculate $\mathcal{L}$?

As we saw in Tutorial 2, there is a link between mean squared error and the likelihood estimates for linear regression models that we can take advantage of.

Derivation time!

We start with our formula for AIC from above:

$$\mathrm{AIC} = 2k - 2\log\mathcal{L} \tag{180}$$

For a model with normal errors, we can use the log likelihood of the normal distribution:

$$\log\mathcal{L} = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \sum_{i}^{N}\frac{1}{2\sigma^2}(y_i - \tilde{y}_i)^2 \tag{181}$$

We can drop the first as it is a constant and we're only assessing relative information with AIC. The last term is actually also a constant: we don't know σ^2 in advance so we use the empirical estimate from the residual ($\hat{\sigma}^2 = 1/N\sum_i^N(y_i - \tilde{y}_i)^2$). Once we plug this in, the two $\sum[(y-\tilde{y})^2]$ terms (in the numerator and denominator, respectively) cancel out and we are left with the last term as $\frac{N}{2}$.

Once we drop the constant terms and incorporate into the AIC formula we get:

$$\mathrm{AIC} = 2k + n\log\left(\sigma^2\right) \tag{182}$$

We can replace σ^2 with the computation for variance (the sum of squared errors divided by number of samples). Thus, we end up with the following formula for AIC for linear and polynomial regression:

$$\mathrm{AIC} = 2K + n\log\left(\frac{\mathrm{SSE}}{n}\right) \tag{183}$$ #

where k is the number of parameters, n is the number of samples, and SSE is the summed squared error.

▼ Outro

Limitations of MLE

The competing, Bayesian framework:

$$P(parameter|data) \propto P(data|parameter)P(parameter)$$

Posterior *Likelihood* *Prior*

Bayesian:

- Use Likelihood + Prior to estimate parameters, i.e., use sample data to update prior belief of the parameter.
- CI is related to the probability (or uncertainty) of the posterior.

Limitations of the Frequentist approach

- Completely rely on the data, thus subject to biases of a particular random sample.
- Obtain the most likely parameters without considering prior knowledge, sometimes computationally expensive.

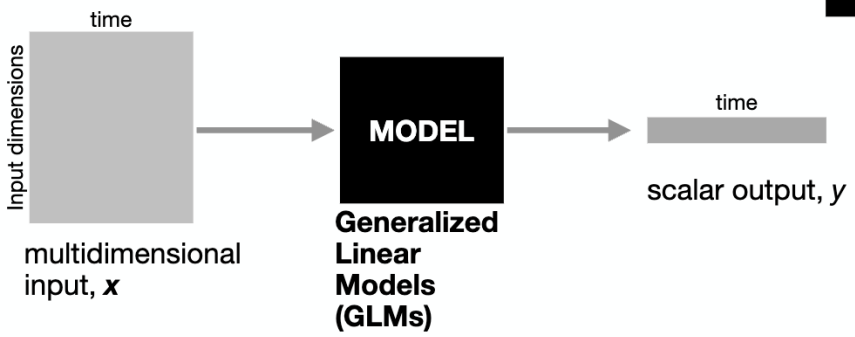
Crossvalidation VS. Bootstrapping

	Crossvalidation	Bootstrapping
Common	Both are resampling methods, computationally expensive (CPU hungry)	
Purpose	Good for estimating the model prediction errors	Good for estimating the confidence interval of model parameters.
Approach	Split the data into multiple sets, thus no overlapping between datasets.	Clone the data to create more sets, thus overlapping datasets.
Sample size	Needs a large sample size	Fine with small samples

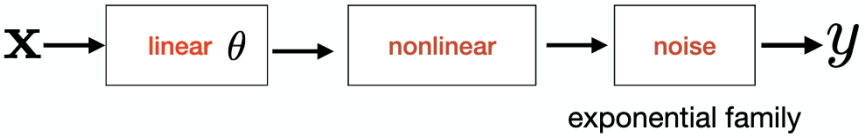
Generalized Linear Models

▼ Intro

A common thread



Generalized Linear Models



exponential family

Different GLMs can solve different problems

Output type	Likelihood	Nonlinearity		
real values	Gaussian $\mathrm{P}(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(y-\mu)^2}{2\sigma^2}$	identity $\mu = \boldsymbol{\theta}^\top \mathbf{x}$		Linear regression
discrete counts 0,1,2,3…	Poisson $\mathrm{P}(y) = \frac{\lambda^y \exp(-\lambda)}{y!}$	exponential $\lambda = \exp\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$		Poisson GLM
binary 0,1	Bernoulli $\mathrm{P}(y) = p^y (1-p)^{1-y}$	logistic $p = f\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$		Logistic regression

Parameter learning: MLE

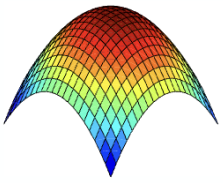
Find parameters that maximize the (log) likelihood:

$$\boldsymbol{\theta}^* = \mathrm{argmax}_\theta \log \mathcal{L}(\boldsymbol{\theta})$$

Nice math: the log-likelihood is concave

-> a single global optimum

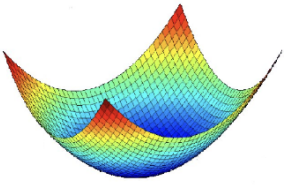
Concave



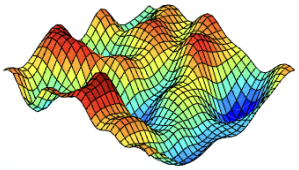
MLE numerical optimization

In practice, we minimize convex negative log likelihood
Usually no closed form solution for optimum: gradient descent

convex



non-convex
local optima



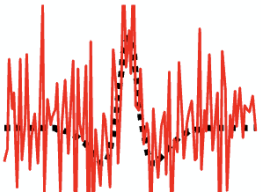
Practical considerations

Q: MLE gives us best fit on training data, but is does it give the best model?

A: NOT NECESSARILY

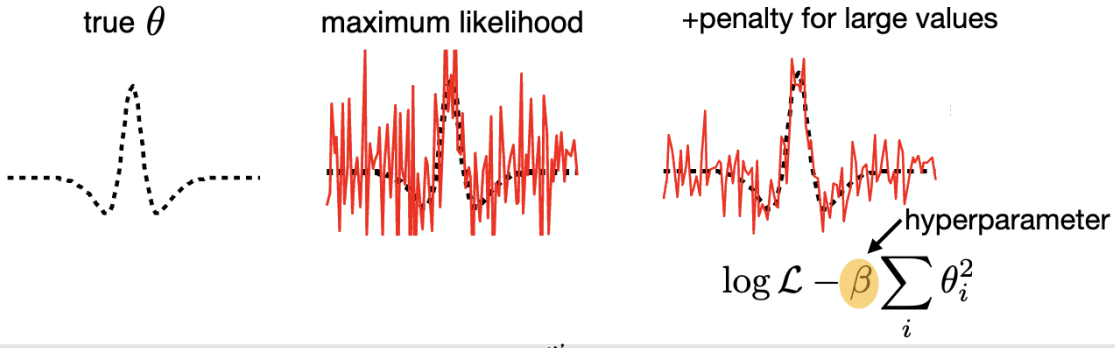
true θ

maximum likelihood



"overfitting"

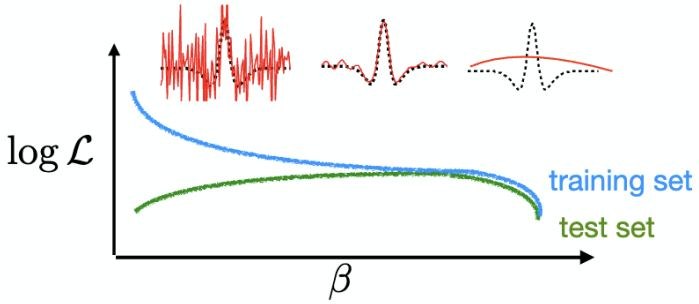
Regularization



Setting hyper-parameters $\log \mathcal{L} - \beta \sum_i \theta_i^2$

Cross-validation

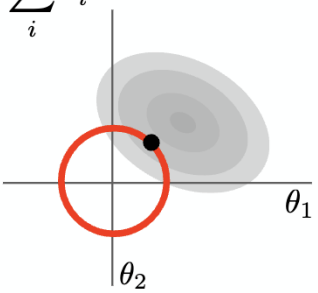
Use a separate validation set



Different choices of regularizers

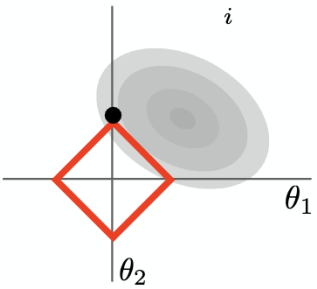
L2: keep weights small

$-\beta \sum_i \theta_i^2$



L1: encourage sparsity

$-\beta \sum_i |\theta_i|$



The Bayesian perspective

MLE+regularization = **MAP** estimate, with regularizer as prior

$$\log \mathcal{L}' = \log \mathcal{L} - \beta \sum_i \theta_i^2 \quad \leftarrow \quad \log \mathcal{N}\left(\boldsymbol{\theta}; \mathbf{0}, \frac{1}{2\beta}\mathbf{I}\right)$$

log(Posterior)= log(Likelihood × Prior)

$$\boldsymbol{\theta}^* = \text{argmax}_\theta \text{P}(\boldsymbol{\theta}|\mathbf{x}, \mathbf{y})$$

Different GLMs can solve different problems

[redacted]

Output type	Likelihood	Nonlinearity	Regularizer:
real values	Gaussian $\mathrm{P}(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(y-\mu)^2}{2\sigma^2}$	identity $\mu = \boldsymbol{\theta}^\top \mathbf{x}$	L2 $-\beta \sum_i \theta_i^2$
discrete counts 0,1,2,3…	Poisson $\mathrm{P}(y) = \frac{\lambda^y \exp(-\lambda)}{y!}$	exponential $\lambda = \exp\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$	L1 $-\beta \sum_i \lvert\theta_i\rvert$
binary 0,1	Bernoulli $\mathrm{P}(y) = p^y (1-p)^{1-y}$	logistic $p = f\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$	

×

▼ GLMs for Encoding

On day 3, we talked about the multiple linear model :

Vector form

neural response

$$y_i = \boldsymbol{\theta}^\top \boldsymbol{x}_i \qquad \forall i = 1, ..., N$$

$\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, ...]^\top$ linear weights

$\boldsymbol{x}_i = [1, x_{i,1}, x_{i,2}, ...]^\top$ multiple stimulus features (e.g., orientation, contrast, etc.)

number of data points

Matrix form

$$\boldsymbol{y} = X\boldsymbol{\theta}$$

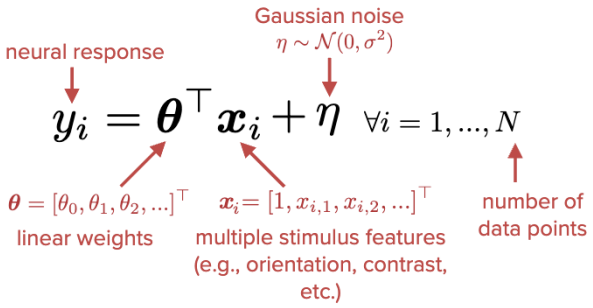
Index i

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \end{bmatrix} \quad \begin{bmatrix} — & \boldsymbol{x}_0 & — \\ — & \boldsymbol{x}_1 & — \\ — & \boldsymbol{x}_2 & — \\ & \vdots & \end{bmatrix} \quad \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \end{bmatrix}$$

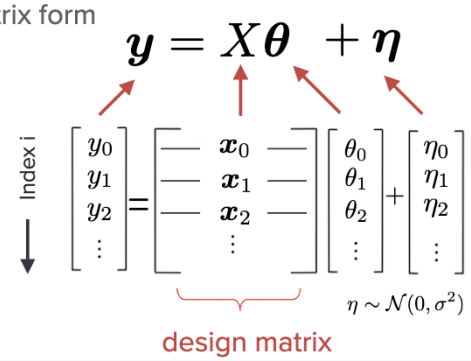
design matrix

If we assume the observation noise to be Gaussian, then

Vector form

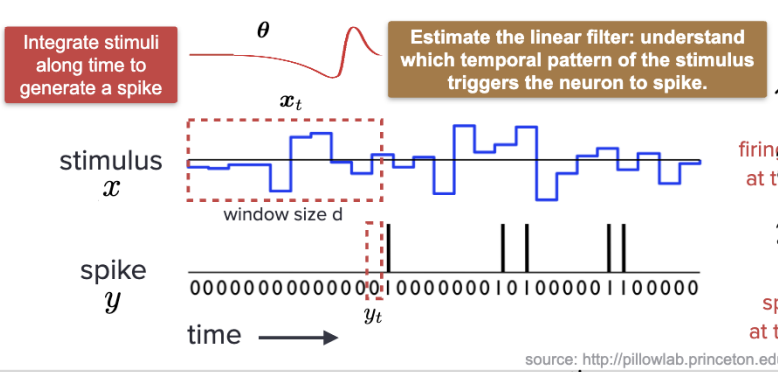


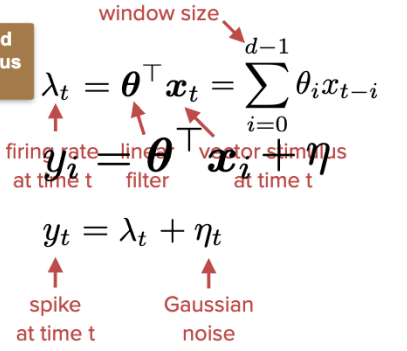
Matrix form



Temporal Filtering Model

Task: predict neural spikes from stimuli at all time points

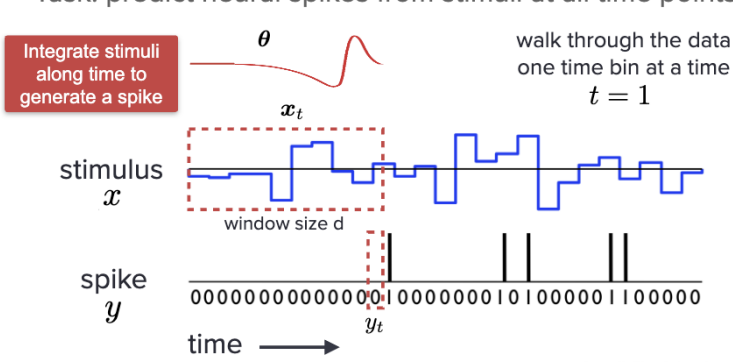




source: http://pillowlab.princeton.edu/teaching/statneuro2018/slides/lec09_GLMs1.pdf

Temporal Filtering Model

Task: predict neural spikes from stimuli at all time points



window size

$$\lambda_t = \boldsymbol{\theta}^\top \boldsymbol{x}_t = \sum_{i=0}^{d-1} \theta_i x_{t-i}$$

firing rate at time t | linear filter | vector stimulus at time t

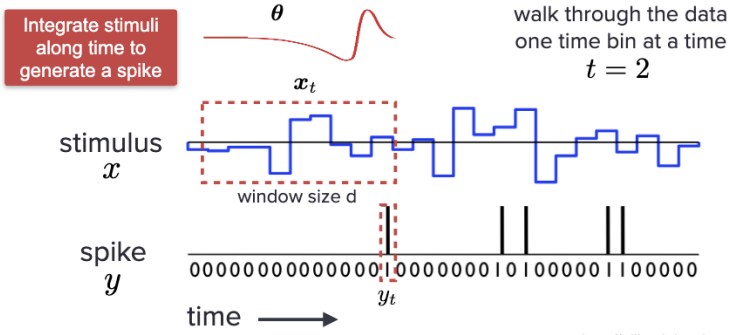
$$y_t = \lambda_t + \eta_t$$

spike at time t | Gaussian noise

source: http://pillowlab.princeton.edu/teaching/statneuro2018/slides/lec09_GLMs1.pdf

Temporal Filtering Model

Task: predict neural spikes from stimuli at all time points



window size

$$\lambda_t = \boldsymbol{\theta}^\top \boldsymbol{x}_t = \sum_{i=0}^{d-1} \theta_i x_{t-i}$$

firing rate at time t | linear filter | vector stimulus at time t

$$y_t = \lambda_t + \eta_t$$

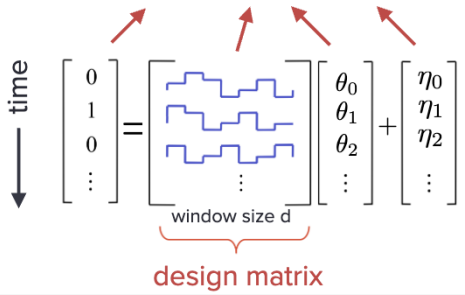
spike at time t | Gaussian noise

source: http://pillowlab.princeton.edu/teaching/statneuro2018/slides/lec09_GLMs1.pdf

Temporal Filtering Model

Build up to the following matrix version along time axis:

$$\boldsymbol{y} = X\boldsymbol{\theta} + \boldsymbol{\eta}$$



MSE solution, ignoring the noise $\boldsymbol{\eta}$ (revisit D3):

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \, ||X\boldsymbol{\theta} - \boldsymbol{y}||_2^2 = \sum_{t=1}^{T} (y_t - \boldsymbol{\theta}^\top \boldsymbol{x}_t)^2$$

Differentiate and set to zero

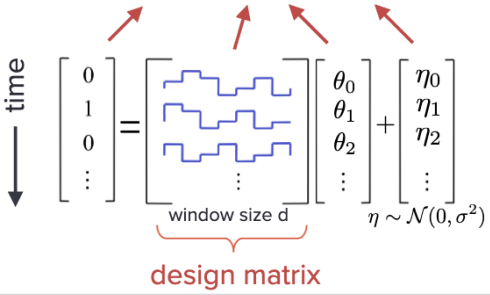
$$\frac{\partial \, ||X\boldsymbol{\theta} - \boldsymbol{y}||_2^2}{\partial \, \boldsymbol{\theta}} = 2X^\top (X\boldsymbol{\theta} - \boldsymbol{y}) = 0$$

$$\Rightarrow \boldsymbol{\theta}_{\text{MSE}} = \underbrace{(X^\top X)^{-1}}_{\text{stimulus covariance}} \underbrace{X^\top \boldsymbol{y}}_{\text{spike-triggered avg (STA)}}$$

Temporal Filtering Model

Build up to the following matrix version along time axis:

$$\boldsymbol{y} = X\boldsymbol{\theta} + \boldsymbol{\eta}$$



MLE solution (revisit D3): the number of time points, window size

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \; \log \mathcal{L}(\boldsymbol{\theta}|X, \boldsymbol{y}) = -\frac{Td}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} (\boldsymbol{y} - X\boldsymbol{\theta})^\top (\boldsymbol{y} - X\boldsymbol{\theta})$$

Differentiate and set to zero

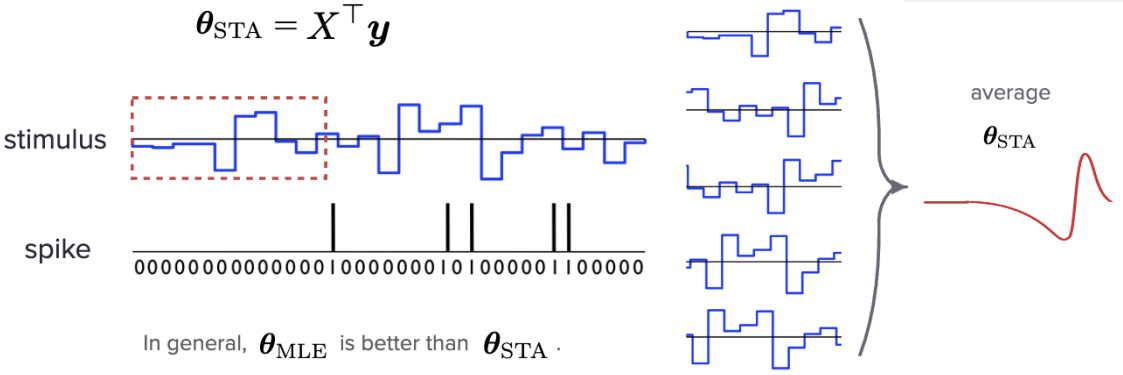
$$\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|X, \boldsymbol{y})}{\partial \, \boldsymbol{\theta}} = -\frac{1}{\sigma^2} X^\top (X\boldsymbol{\theta} - \boldsymbol{y}) = 0$$

$$\Rightarrow \boldsymbol{\theta}_{\text{MLE}} = \underbrace{(X^\top X)^{-1}}_{\text{stimulus covariance}} \underbrace{X^\top \boldsymbol{y}}_{\text{spike-triggered avg (STA)}}$$

(same as MSE when the noise is Gaussian)

Spike-triggered Average (STA)

$$\boldsymbol{\theta}_{\text{STA}} = X^\top \boldsymbol{y}$$



In general, $\boldsymbol{\theta}_{\text{MLE}}$ is better than $\boldsymbol{\theta}_{\text{STA}}$.

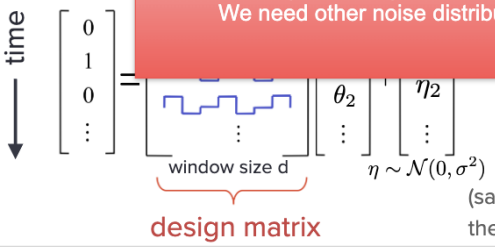
Temporal Filtering Model

Build up to following matrix version along time axis:

The noise is not Gaussian!
The neural response is spike count which is non-negative and discrete!
We need other noise distribution and nonlinear function.

$$\ldots - \frac{1}{2\sigma^2}(\boldsymbol{y} - X\boldsymbol{\theta})^\top(\boldsymbol{y} - X\boldsymbol{\theta})$$

$\ldots$ts and d is the window

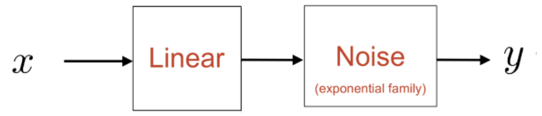


$$\frac{\partial \log \mathcal{L}(\boldsymbol{\theta}|X, \boldsymbol{y})}{\partial \boldsymbol{\theta}} = -\frac{1}{\sigma^2} X^\top (X\boldsymbol{\theta} - \boldsymbol{y}) = 0$$

$$\Rightarrow \boldsymbol{\theta}_{\text{MLE}} = \underbrace{(X^\top X)^{-1}}_{\text{stimulus covariance}} \underbrace{X^\top \boldsymbol{y}}_{\text{spike-triggered avg (STA)}}$$

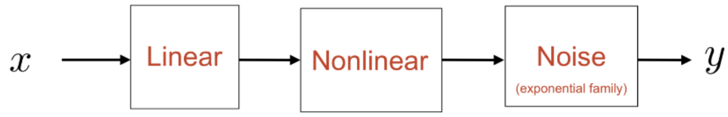
(same as MSE when the noise is Gaussian)

Linear model



Example: linear Gaussian model $y = \theta x + \eta$ where $\eta \sim \mathcal{N}(0, \sigma^2)$

Generalized linear model



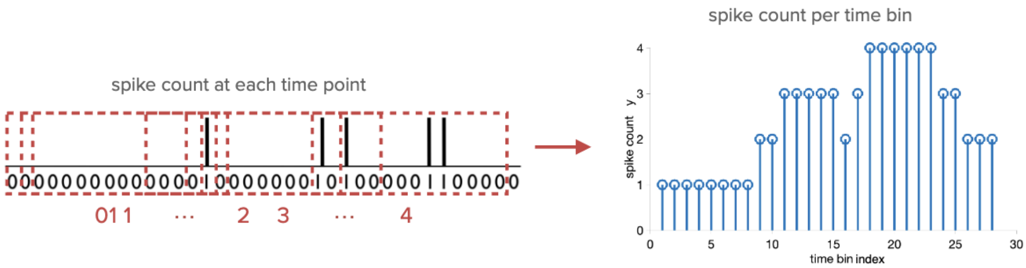
Example: nonlinear Gaussian model $y = f(\theta x) + \eta$ where $\eta \sim \mathcal{N}(0, \sigma^2)$

nonlinear
f^{-1}: link function

Poisson GLM $y \sim \text{Poisson}\left(f(\theta x)\right)$ for spike train encoding

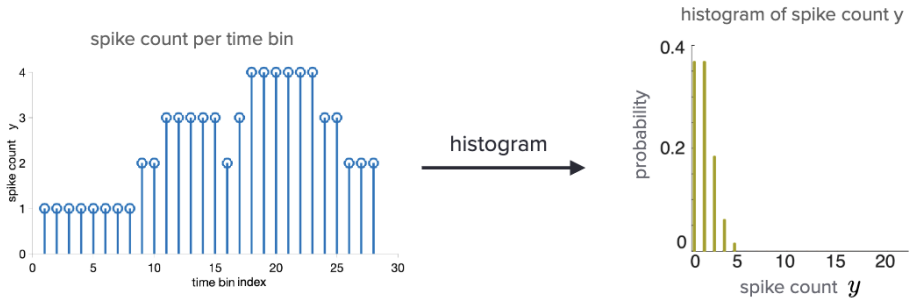
Poisson GLM

Poisson distribution is used to model the number of events (spikes) occurring within a given time interval.



Poisson GLM

Poisson distribution is used to model the number of events (spikes) occurring within a given time interval.



Poisson GLM

Poisson distribution is used to model the number of events (spikes) occurring within a given time interval.

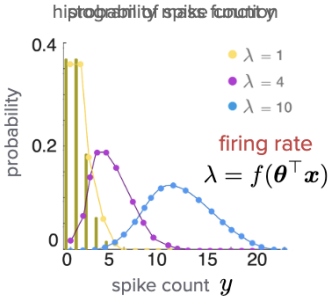
The probability mass function (pmf) of Y is given by

$$P(Y=y)=\frac{\lambda^y e^{-\lambda}}{y!}$$

unknown parameter, >0

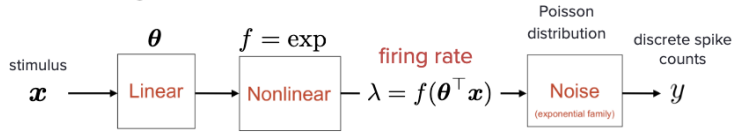
observed discrete spike count y=0,1,2,3,...

not contain λ



Poisson GLM

Temporal filtering model



Given the probability mass function for Poisson distribution $P(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!}$

We can now assume the encoding distribution to be

for time t $p(y_t|\boldsymbol{\theta}, \boldsymbol{x}_t) = P(Y = y_t|\lambda_t = f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)) = \frac{[f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)]^{y_t}}{y_t!} e^{-f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)}$

Poisson GLM

Temporal filtering model

for time t $p(y_t|\boldsymbol{\theta}, \boldsymbol{x}_t) = P(Y = y_t|\lambda_t = f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)) = \frac{[f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)]^{y_t}}{y_t!} e^{-f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)}$

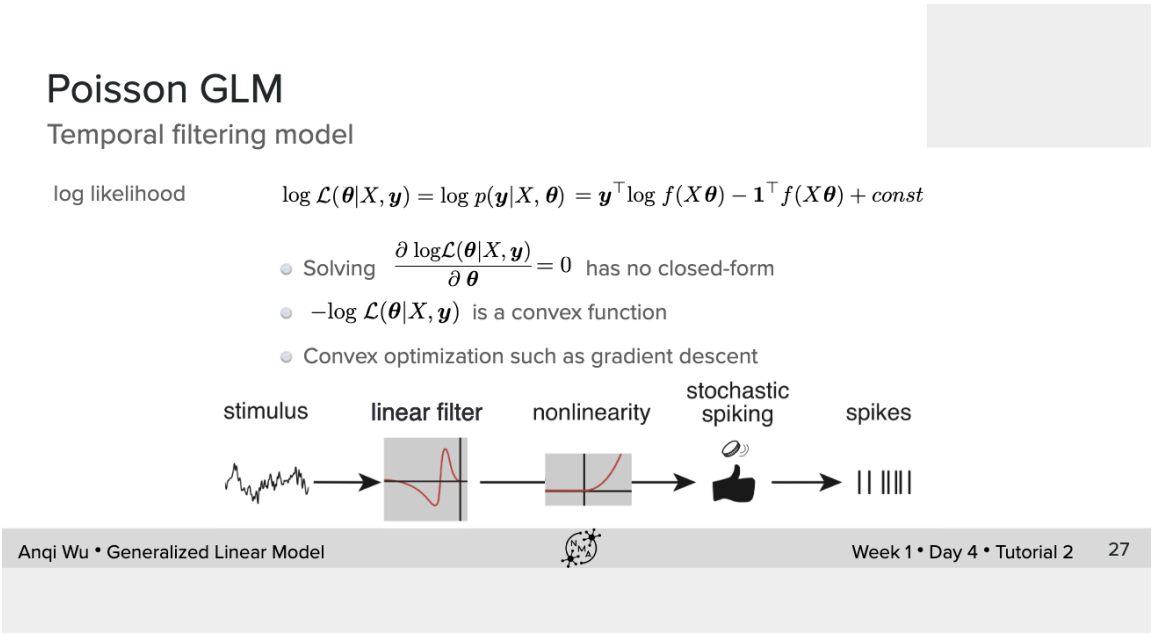
log likelihood

$$\log \mathcal{L}(\boldsymbol{\theta}|X, \boldsymbol{y}) = \log p(\boldsymbol{y}|X, \boldsymbol{\theta}) = \log \prod_{t=1}^{T} p(y_t|\boldsymbol{\theta}, \boldsymbol{x}_t) = \log \prod_{t=1}^{T} \frac{[f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)]^{y_t}}{y_t!} e^{-f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)}$$

$$= \sum_{t=1}^{T} \log \frac{[f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)]^{y_t}}{y_t!} + \sum_{t=1}^{T} \log e^{-f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)}$$

$$= \sum_{t=1}^{T} y_t \log f(\boldsymbol{\theta}^\top \boldsymbol{x}_t) - \sum_{t=1}^{T} \log y_t! - \sum_{t=1}^{T} f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)$$

$$= \sum_{t=1}^{T} \left(y_t \log f(\boldsymbol{\theta}^\top \boldsymbol{x}_t) - f(\boldsymbol{\theta}^\top \boldsymbol{x}_t)\right) + const = \boldsymbol{y}^\top \log f(X\boldsymbol{\theta}) - \mathbf{1}^\top f(X\boldsymbol{\theta}) + const$$



importance of convexity in optimization

- We have seen previously that in the Linear-Gaussian case, maximum likelihood parameter estimate can be computed analytically. That is great because it only takes us a single line of code!
- Unfortunately in general there is no analytical solution to our statistical estimation problems of interest. Instead, we need to apply a nonlinear optimization algorithm to find the parameter values that minimize some *objective function*. This can be extremely tedious because there is no general way to check whether we have found *the optimal solution* or if we are just stuck in some local minimum.
- Somewhere in between these two extremes, the special case of convex objective function is of great practical importance. Indeed, such optimization problems can be solved very reliably (and usually quite rapidly too!) using some standard software.

Here we will use the `scipy.optimize` module, it contains a function called `minimize` that provides a generic interface to a large number of optimization algorithms. This function expects as argument an objective function and an "initial guess" for the parameter values. It then returns a dictionary that includes the minimum function value, the parameters that give this minimum, and other information.

Let's see how this works with a simple example. We want to minimize the function $f(x) = x^2$:

```
f = np.square
res = minimize(f, x0=2)
print(f"Minimum value: {res['fun']:.4g} at x = {res['x'].item():.5e}")
```

```
Minimum value: 3.566e-16 at x = -1.88846e-08
```

When minimizing a $f(x) = x^2$, we get a minimum value of $f(x) \approx 0$ when $x \approx 0$. The algorithm doesn't return exactly 0, because it stops when it gets "close enough" to a minimum. You can change the `tol` parameter to control how it defines "close enough".

Fitting the Poisson GLM

What should the objective function look like? We want it to return the negative log likelihood: $-\log P(y \mid \mathbf{X}, \theta)$.

In the Poisson GLM,

$$\log P(\mathbf{y} \mid \mathbf{X}, \theta) = \sum_t \log P(y_t \mid \mathbf{x_t}, \theta), \tag{185}$$

where

$$P(y_t \mid \mathbf{x_t}, \theta) = \frac{\lambda_t^{y_t} \exp(-\lambda_t)}{y_t!}, \text{ with rate } \lambda_t = \exp(\mathbf{x_t}^\top \theta). \tag{186}$$

Now, taking the log likelihood for all the data we obtain:

$$\log P(\mathbf{y} \mid X, \theta) = \sum_t (y_t \log(\lambda_t) - \lambda_t - \log(y_t!)). \tag{187}$$

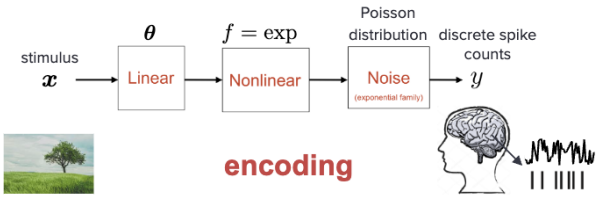
Because we are going to minimize the negative log likelihood with respect to the parameters θ, we can ignore the last term that does not depend on θ. For faster implementation, let us rewrite this in matrix notation:

$$\mathbf{y}^\top \log(\lambda) - \mathbf{1}^\top \lambda, \text{ with rate } \lambda = \exp(\mathbf{X}\theta) \tag{188}$$

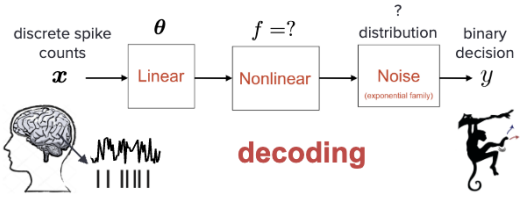
Finally, don't forget to add the minus sign for your function to return the negative log likelihood.

▼ Classifiers and regularizers

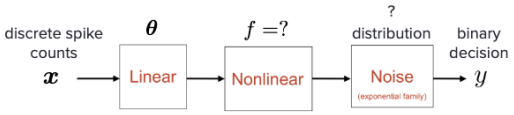
Previously we introduce how to use Poisson GLM to **encode** neural spike trains from stimuli.



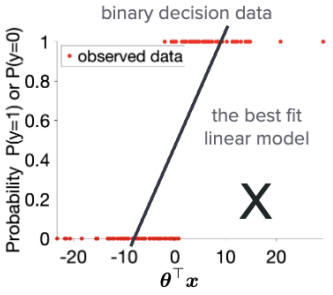
Here we will talk about **decoding** a binary decision of an animal from spike train data using a specific instantiation of GLM.

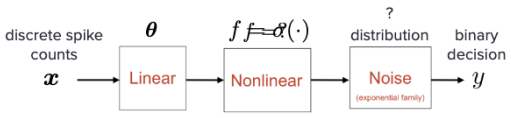


Anqi Wu • Generalized Linear Model — Week 1 • Day 4 • Tutorial 3 — 30

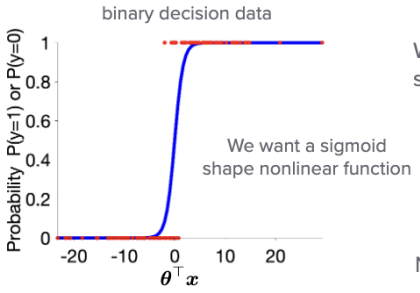


Link function $f = ?$



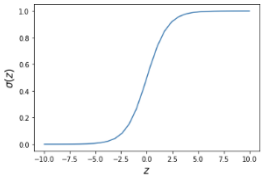


Link function $f = ?$

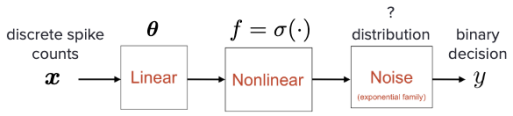


We define $f = \sigma(\cdot)$ which is a "squashing" function called the sigmoid function.

$$\sigma(z) = \frac{1}{\exp(-z) + 1}$$



Notice $0 \leq f(\boldsymbol{\theta}^\top \boldsymbol{x}) = \frac{1}{\exp(-\boldsymbol{\theta}^\top \boldsymbol{x}) + 1} \leq 1$



Distribution of the observation noise

$f(\boldsymbol{\theta}^\top \boldsymbol{x}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}) = \dfrac{1}{\exp(-\boldsymbol{\theta}^\top \boldsymbol{x}) + 1}$ only gives us a probability-like value, not a binary decision.

Bernoulli distribution: generate a binary value with some input probability value.

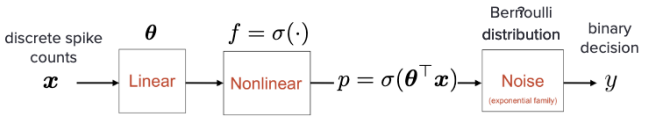
single coin flip	outcome y:	head	tail
	probability:	p	1-p

The probability mass function for y is

$$P(y|p) = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{if } y = 0 \end{cases}$$

Alternatively,

$$P(y|p) = p^y(1-p)^{1-y}$$



Distribution of the observation noise

Given the probability mass function $P(y|p) = p^y(1-p)^{1-y}$

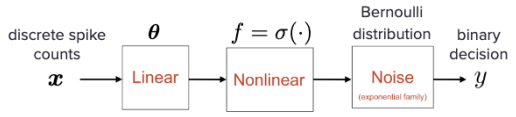
We can write down the decoding distribution as

$$p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) = P(y_i|p_i = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i)) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i)^{y_i}(1 - \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i))^{1-y_i}$$

for all data points

$$p(\boldsymbol{y}|X, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) = \prod_{i=1}^{N} \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i)^{y_i}(1 - \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i))^{1-y_i}$$

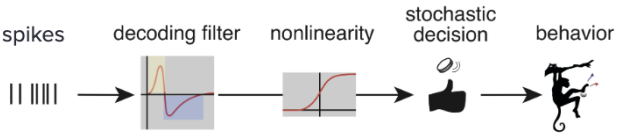
Bernoulli GLM *(Logistic Regression)*



Decoding distribution $p(\boldsymbol{y}|X,\boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i,\boldsymbol{\theta}) = \prod_{i=1}^{N} \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i)^{y_i}(1-\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i))^{1-y_i}$

log likelihood $\log \mathcal{L}(\boldsymbol{\theta}|X,\boldsymbol{y}) = \log p(\boldsymbol{y}|X,\boldsymbol{\theta}) = \sum_{i=1}^{N} y_i \log \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i) + (1-y_i)\log\,(1-\sigma(\boldsymbol{\theta}^\top \boldsymbol{x}_i))$

- no closed form solution, need optimization
- **logistic regression**: closely related to linear regression, but for discrete outcomes.

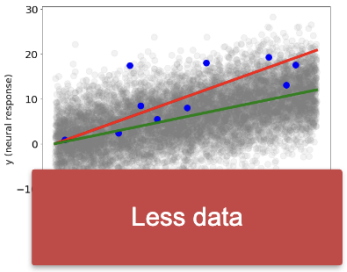


Different GLMs can solve different problems

Output type	Likelihood	Nonlinearity	
real values	Gaussian $\mathrm{P}(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(y-\mu)^2}{2\sigma^2}$	identity $\mu = \boldsymbol{\theta}^\top \mathbf{x}$	Linear regression
discrete counts 0,1,2,3...	Poisson $\mathrm{P}(y) = \frac{\lambda^y \exp(-\lambda)}{y!}$	exponential $\lambda = \exp\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$	Poisson GLM
binary 0,1	Bernoulli $\mathrm{P}(y) = p^y(1-p)^{1-y}$	logistic $p = \sigma\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$	Logistic regression (Bernoulli GLM)

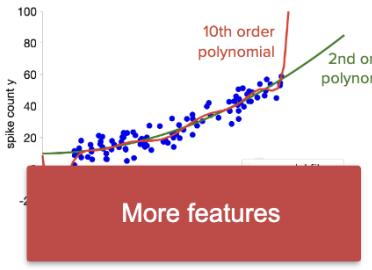
Revisit Overfitting

Linear regression overfits when **data sample is small and different from true data distribution.**



Polynomial regression overfits when the polynomial order is high (**high model complexity**)

$$y = \sum_{p=0}^{P} \theta_p x^p$$



Regularization

Regularization shrinks the parameters $\boldsymbol{\theta}$ in GLMs towards 0 — this can reduce overfitting.

Example: linear Gaussian model $\boldsymbol{y} = X\boldsymbol{\theta} + \boldsymbol{\eta}$

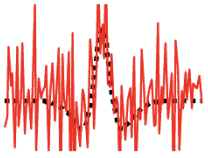
L2 (a.k.a. "ridge") regularization:

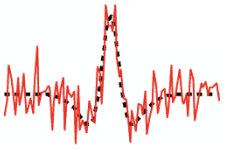
- penalize squared parameters
- suppress all features

hyperparameter

$$\log \mathcal{L}' = \log \mathcal{L} - \frac{\beta}{2} \sum_i \theta_i^2$$

$\boldsymbol{\theta}_{\text{MLE}} = (X^\top X)^{-1} X^\top \boldsymbol{y}$



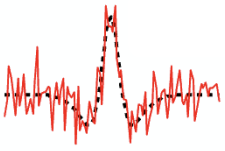


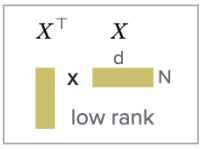
Regularization for Linear Gaussian Model

L2 (a.k.a. "ridge") regularization:
- penalize squared parameters
- suppress all features

hyperparameter

$$\log \mathcal{L}' = \log \mathcal{L} - \frac{\beta}{2}\sum_i \theta_i^2$$





Formally, if $\log \mathcal{L}$ is the log likelihood of linear Gaussian

$$\log \mathcal{L}'(\boldsymbol{\theta}|X,\boldsymbol{y}) = \log \mathcal{L}(\boldsymbol{\theta}|X,\boldsymbol{y}) - \frac{\beta}{2}\sum_i \theta_i^2$$

new quadratic term

$$= -\frac{Nd}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}(\boldsymbol{y}-X\boldsymbol{\theta})^\top(\boldsymbol{y}-X\boldsymbol{\theta}) - \frac{\beta}{2}\boldsymbol{\theta}^\top\boldsymbol{\theta}$$

linear Gaussian on D3

Differentiate and set to zero

$$\frac{\partial \log \mathcal{L}'(\boldsymbol{\theta}|X,\boldsymbol{y})}{\partial \boldsymbol{\theta}} = -\frac{1}{\sigma^2}X^\top(X\boldsymbol{\theta}-\boldsymbol{y}) - \beta\boldsymbol{\theta} = 0$$

$$\Rightarrow \boldsymbol{\theta}_{\text{Ridge}} = (X^\top X + \sigma^2\beta I)^{-1}X^\top\boldsymbol{y}$$

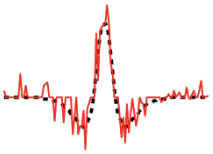
$$\boldsymbol{\theta}_{\text{MLE}} = (X^\top X)^{-1}X^\top\boldsymbol{y}$$

Regularization for Linear Gaussian Model

L1 (a.k.a, "lasso") regularization:
- penalize the absolute value of parameters
- encourages sparse parameters
- selects informative features

hyperparameter

$$\log \mathcal{L}' = \log \mathcal{L} - \frac{\beta}{2}\sum_i |\theta_i|$$



Formally, if $\log \mathcal{L}$ is the log likelihood of linear Gaussian model

$$\log \mathcal{L}'(\boldsymbol{\theta}|X,\boldsymbol{y}) = \log \mathcal{L}(\boldsymbol{\theta}|X,\boldsymbol{y}) - \frac{\beta}{2}\sum_i |\theta_i|$$

$$= -\frac{Nd}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}(\boldsymbol{y}-X\boldsymbol{\theta})^\top(\boldsymbol{y}-X\boldsymbol{\theta}) - \frac{\beta}{2}\sum_i |\theta_i|$$

no closed form solution, even for linear Gaussian need optimization

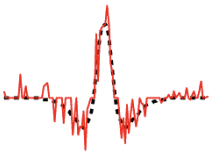
Regularization for Linear Gaussian Model

L1 (a.k.a, “lasso”) regularization:

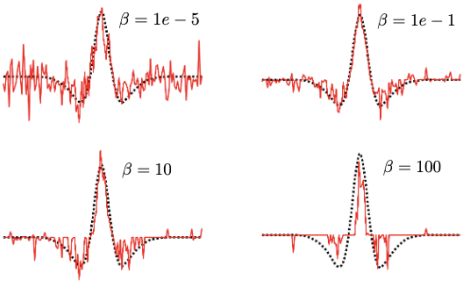
- penalize the absolute value of parameters
- encourages sparse parameters
- selects informative features

hyperparameter

$$\log \mathcal{L}' = \log \mathcal{L} - \frac{\beta}{2} \sum_i |\theta_i|$$



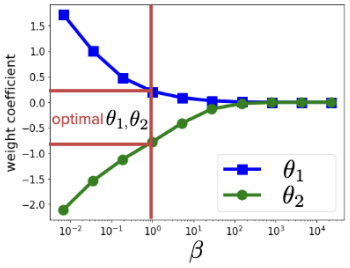
Different β would lead to different sparsity levels



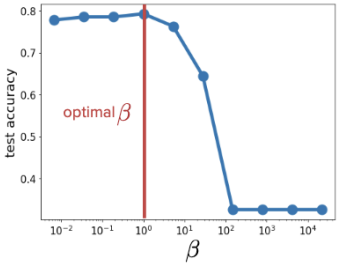
Regularization for Logistic Regression

Similarly, we could add L2 (ridge) or L1 (lasso) penalty to the log likelihood of logistic regression.

logistic regression + **L2** regularization



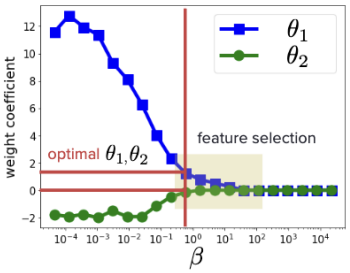
accuracy performance on test data



Regularization for Logistic Regression

Similarly, we could add L2 (ridge) or L1 (lasso) penalty to the log likelihood of logistic regression.

logistic regression + **L1** regularization



accuracy performance on test data