Welcome!
#pod-031

(Reviewed by: Deepak)

neuromatch academy

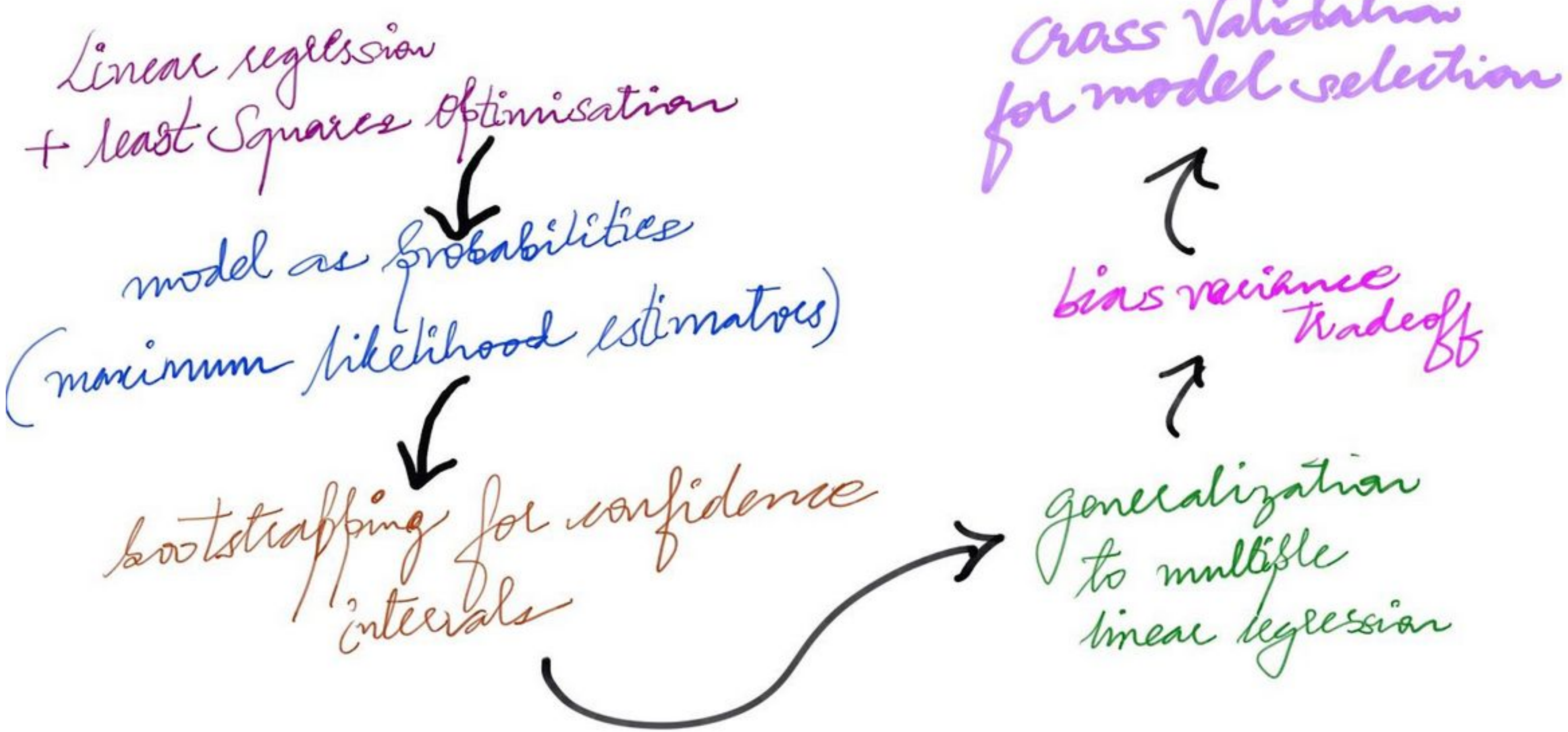
facebook Reality Labs

Agenda

- **Day-#3**
  - Tutorial part 1 (Least squares optimisation)
  - Tutorial part 2 (Maximum likelihood estimation)
  - Tutorial part 3 (Bootstrapping linear model parameters)
  - Tutorial part 4 (multiple linear regression and polynomial regression)
  - Tutorial part 5 (bias-variance trade-off)
  - Tutorial part 6 (Cross Validation for model selection)

W1D3_pod 031

W1D3_pod 031


Linear regression
+ least Squares Optimisation
model as probabilities
(maximum likelihood estimators)
bootstrapping for confidence intervals
generalization to multiple linear regression
bias variance tradeoff
Cross Validation for model selection

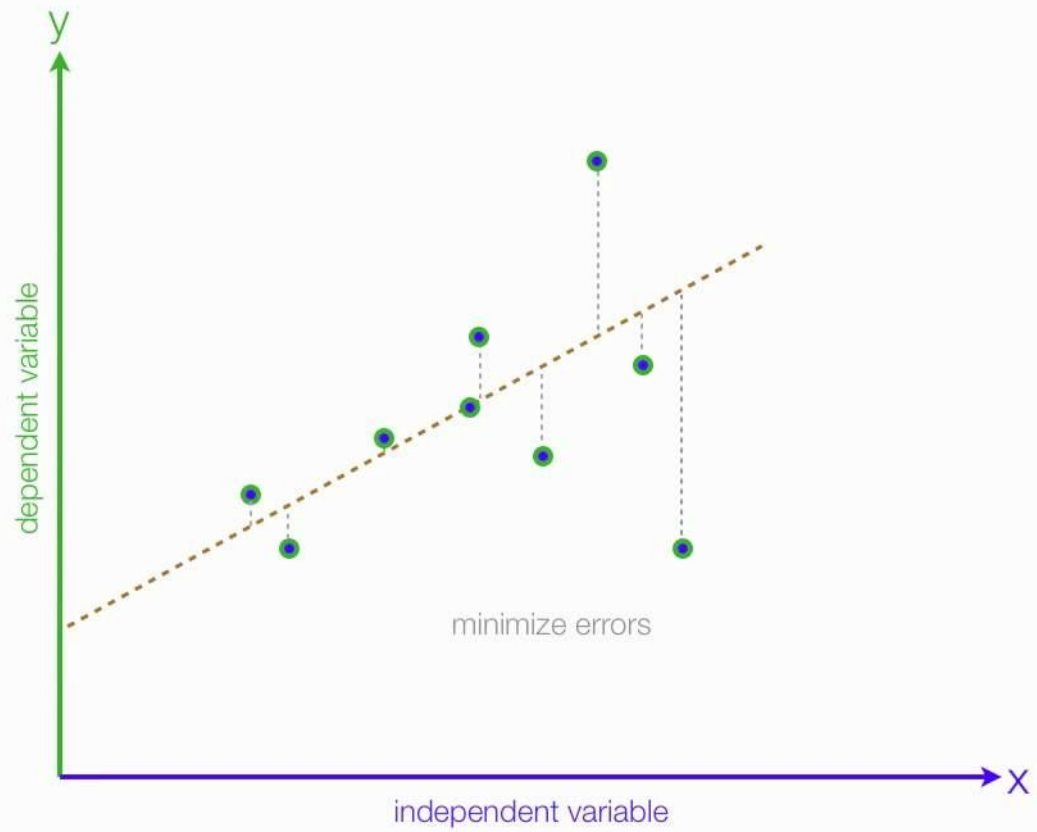
Tutorial #1 Explanations

Mean Squared Error

Assumption: Data is drawn from a linear relationship with noise added, and we have found an effective approach for estimating model parameters based on minimizing the mean squared error.

Least squares (LS) optimization problems are those in which the objective function is a quadratic function of the parameter(s) being optimized.

Suppose you have a set of measurements, y_n (the "dependent" variable) obtained for different input values, x_n (the "independent" or "explanatory" variable). Suppose we believe the measurements are proportional to the input values, but are corrupted by some (random) measurement errors,ϵ_n , that is: $yn = \vartheta xn + \epsilon n$ for some unknown slope parameter ϑ.


y
dependent variable
minimize errors
x
independent variable

Mean Squared error

The least squares regression problem uses **mean squared error (MSE)** as its objective function, and it aims to find the value of the parameter ϑ by minimizing the average of squared errors:

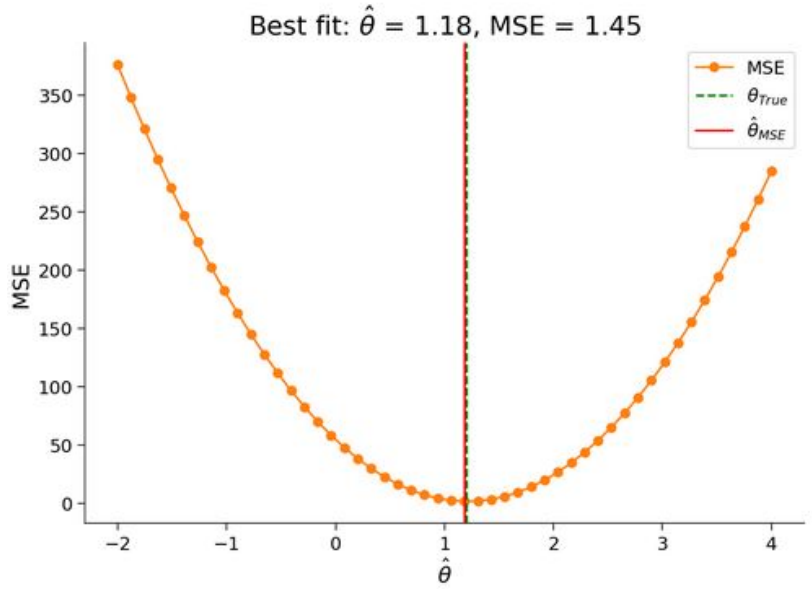
$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} (y_n - \theta x_n)^2$$

We use MSE to evaluate how successful a particular slope estimate ϑ^ is for explaining the data,

closer to 0 the MSE is, the better our estimate fits the data.

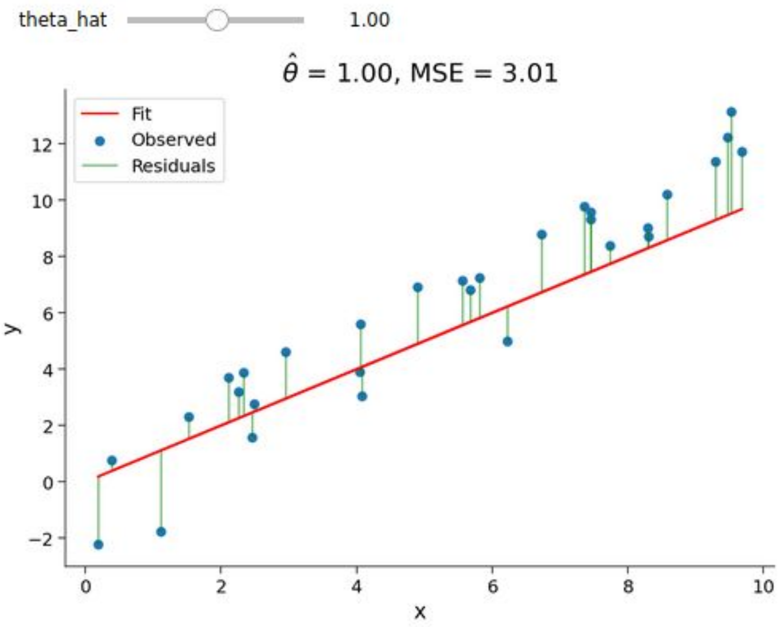
Best fit

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 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 (also called an **error landscape**, especially when we deal with more than one parameter). We can select the final ϑ^(ϑ^$_{MSE}$) as the one which results in the lowest error.



Effect of slope estimate on model fit.

We display the **residuals**, the differences between observed and predicted data, as line segments between the data point (observed response) and the corresponding predicted response on the model fit line.



Least Squares Optimisation

Analytical least squares solution: minimize the cost function.

Mean squared error is a convex objective function, therefore we can compute its minimum using calculus

$$\hat{\theta} = \frac{\vec{x}^\top \vec{y}}{\vec{x}^\top \vec{x}}$$

Why do we model landscapes as convex functions?

1. Because min of the function is always the global minimum
2. Big data helps convex functions to converge faster if your cost function is strictly convex (strictly convex means 2nd derivative of the cost is always greater than zero) and if cost function is convex in general.

Ref: https://michielstock.github.io/ConvexSummary/
https://www.robots.ox.ac.uk/seminars/Extra/2015_06_30_AnnaChoromanska.pdf

Derivation:

(Objective function)

Least Squares Regression $y_n = \theta x_n + \varepsilon_n$

Mean Squared Error : $\min\limits_{\theta} \frac{1}{N} \sum_{n=1}^{N} (y_n - \theta x_n)^2$

(avg of squared errors)

<u>derivative</u> : $\frac{d}{d\theta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \theta x_i)^2 = 0$

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

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

Chain Rule:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{d}{d\theta}(y_i^2)$$

$$+\frac{d}{d\theta}(\theta x_i)^2$$

$$-2\frac{d}{d\theta}(y_i)(\theta x_i) = 0$$

$$\frac{1}{N}\sum_{i=1}^{N} -2y_ix_i + 2\theta_{x_i} \cdot x_i - 2 = 0$$

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

Solving for Optimal value –

$$\hat{\theta} = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2} = \frac{\vec{x}^T\vec{y}}{\vec{x}^T\vec{x}}.$$

Tutorial #2
Explanations

Maximum Likelihood Estimation (MLE)

Recall our linear model: $y=\vartheta_x+\epsilon$.

Where the noise component ϵ is often modeled as a random variable drawn from a Gaussian/normal distribution

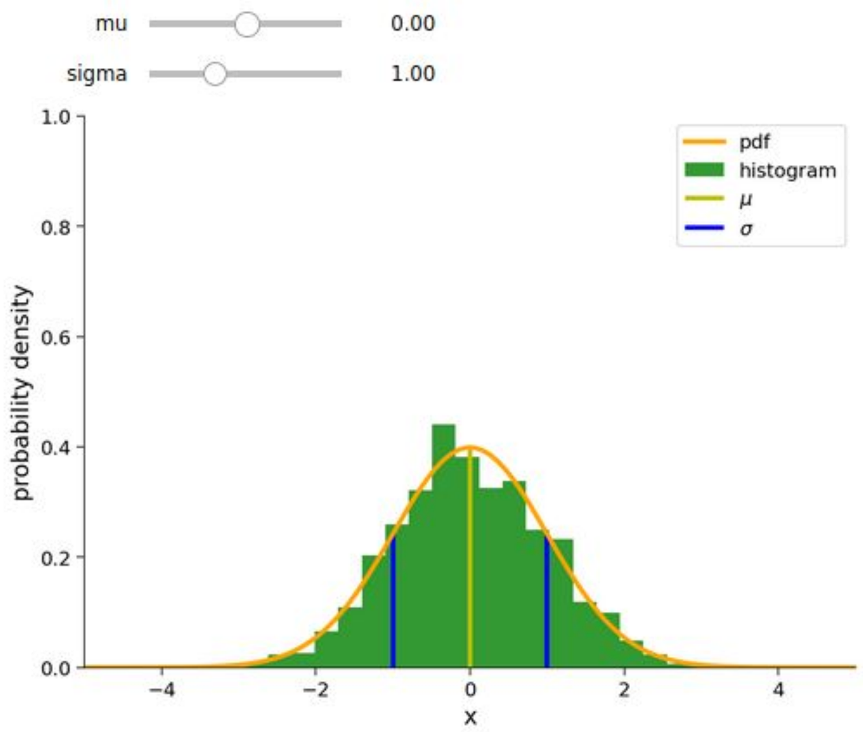
The Gaussian distribution is described by its probability density function (pdf)

$$\mathcal{N}(x;\mu,\sigma^2)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

and is dependent on two parameters: the mean μ and the variance σ^2. We often consider the noise signal to be Gaussian "white noise", with zero mean and unit variance:

$$\epsilon\sim\mathcal{N}(0,1).$$

Effect of mu and sigma on location/shape of samples



Probabilistic models

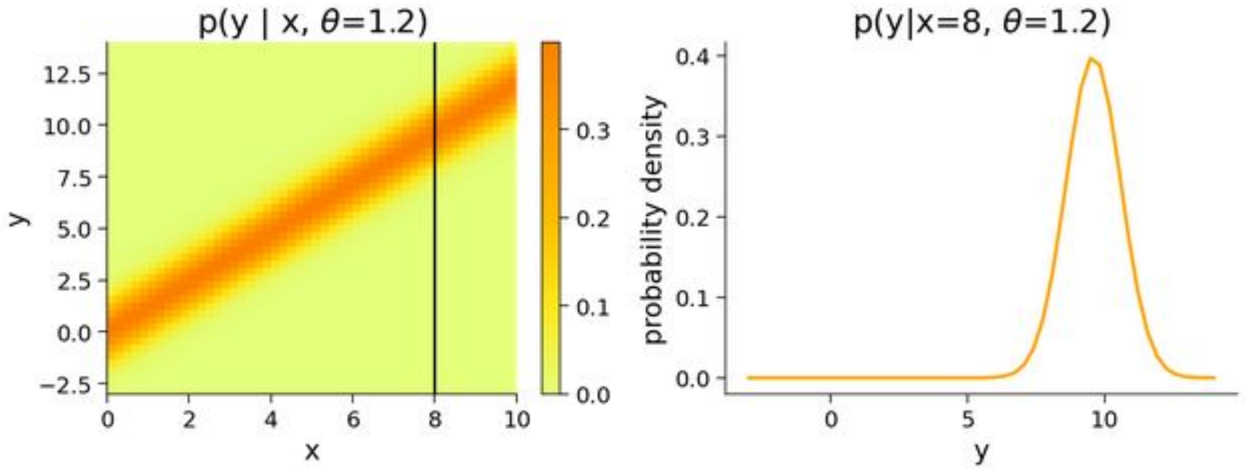
We can now also treat y as a random variable drawn from a Gaussian distribution where $\mu = \vartheta_x$ and $\sigma^2 = 1$

$$y \sim \mathcal{N}(0, 1).$$

which is to say that the probability of observing y given x and parameter ϑ is:

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

Visualization of p(y|x, theta=1.2)



We can also see $p(y|x,\vartheta)$ as a function of x.
This is the stimulus likelihood function, and it is useful in case we want to decode the input x from observed responses y.
This is relevant from the point of view of a neuron that does not have access to the outside world and tries to infer what's out there from the responses of other neurons!

Likelihood Estimation

When we model our data probabilistically, uncertainty becomes inherent. So, our estimates have likelihood ϑ^ that fits our data. The likelihood function L(ϑ) is equal to the probability density function parameterized by that ϑ:

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

When dealing with a set of data points, we are concerned with their joint probability - the likelihood that all data points are explained by our parameterization. Since we have assumed that the noise affects each output independently, we can factorize the likelihood, and write:

$$\mathcal{L}(\theta|X,Y) = \prod_{i=1}^{N} \mathcal{L}(\theta|x_i,y_i),$$

where we have N data points $X=\{x_1,\dots,x_N\}$ and $Y=\{y_1,\dots,y_N\}$.

Overcoming issues of Likelihood estimation

In practice, such a product can be numerically unstable. Indeed multiplying small values together can lead to underflow, the situation in which the digital representation of floating point number reaches its limit. This problem can be circumvented by taking the logarithm of the likelihood because the logarithm transforms products into sums:

$$\log \mathcal{L}(\theta|X,Y) = \sum_{i=1}^{N} \log \mathcal{L}(\theta|x_i, y_i)$$

We can take the sum of the log of the output of our likelihood method applied to the full dataset to get a better idea of how different ϑ^ compare. We can also plot the different distribution densities over our dataset and see how they line up qualitatively.

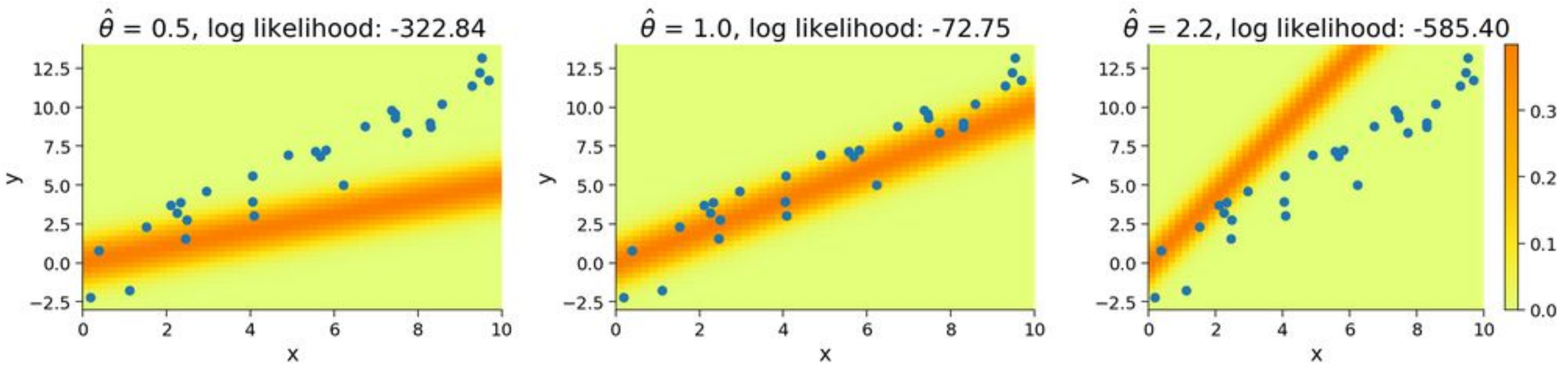
Why logs?

1. Helps with numerical stability (without need a change in the parameter values that maximizes the likelihood)
2. *Monotonically increasing*, which means that it preserves the order of its inputs.

Visualising different distribution densities

Finding the best estimator: estimator that maximizes the likelihood.

$$\hat{\theta}_{\mathrm{MLE}} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta | X, Y)$$



Solving for maximum likelihood estimator

So we have:

$$\hat{\theta}_{\text{MLE}} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{m} \log \mathcal{L}(\theta | x_i, y_i)$$

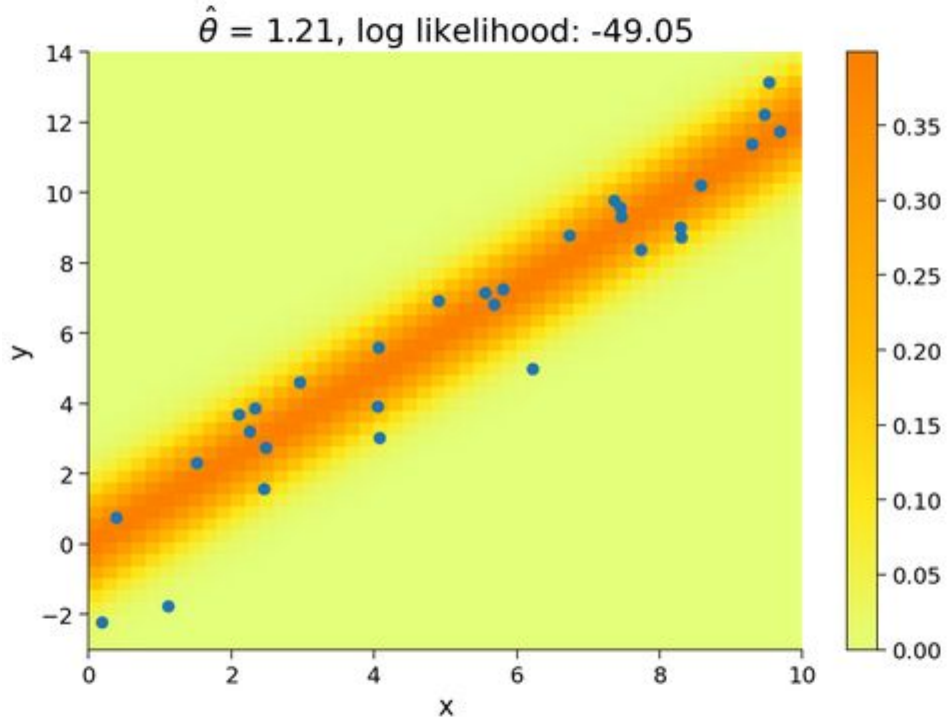
$$\hat{\theta}_{\text{MLE}} = \underset{\theta}{\text{argmax}} [-\frac{N}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \theta x_i)^2].$$

Maximizing the log likelihood is the same as minimizing the negative log likelihood (in practice optimization routines are developed to solve minimization not maximization problems). Because of the convexity of this objective function, we can take the derivative of our negative log likelihood, set it to 0, and solve - minimizing MSE.

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

$$\hat{\theta}_{\text{MLE}} = \hat{\theta}_{\text{MSE}} = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2}$$

Visualize density with theta_hat_mle



Tutorial #3
Explanations

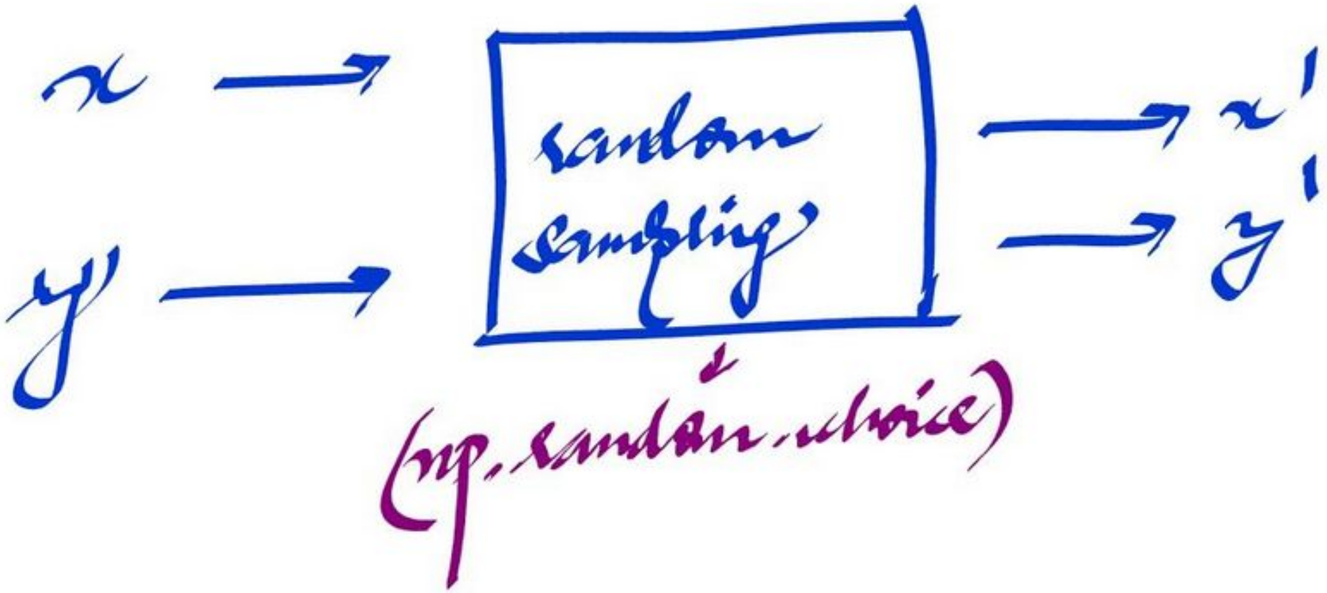
Why Bootstrap?

How good is our estimate really? How confident are we that it will generalize to describe new data we haven't seen yet?

One solution to this is to just collect more data and check the MSE on this new dataset with the previously estimated parameters. However this is not always feasible and still leaves open the question of how quantifiably confident we are about the accuracy of our model.

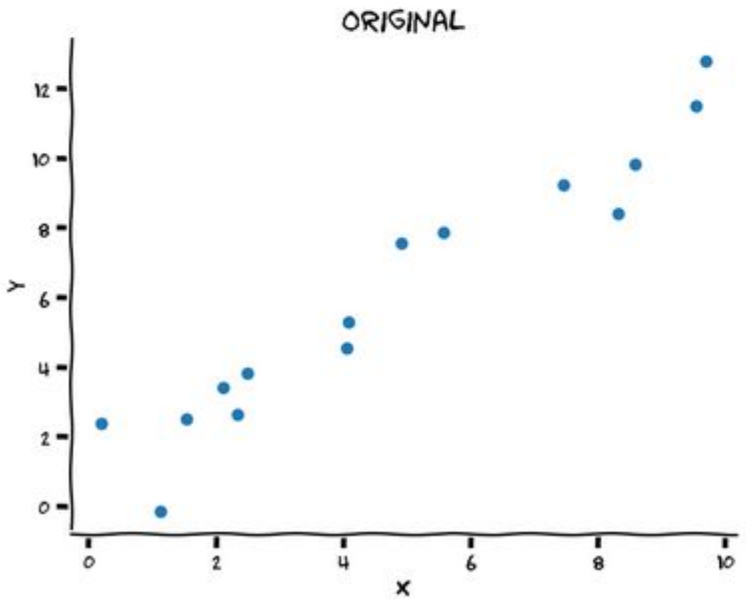
The idea is to generate many new synthetic datasets from the initial true dataset by randomly sampling from it, then finding estimators for each one of these new datasets, and finally looking at the distribution of all these estimators to quantify our confidence. Note that each new resampled datasets 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

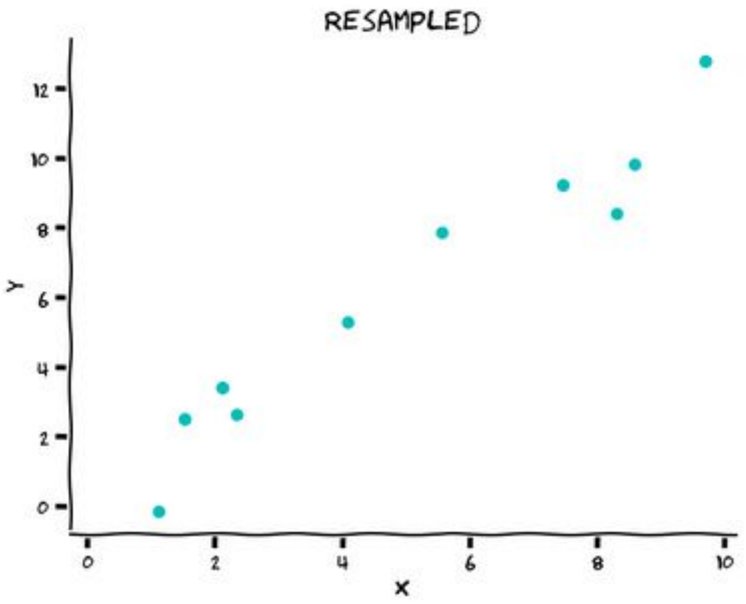
Resampling dataset with replacement



Resampling with replacement output

The actual number of points across the two sets is the same, but some have been repeated so they only display once.





Bootstrapping

Bootstrap process: generate a set of ϑ^ values from a dataset of x inputs and y measurements.

Algorithm:

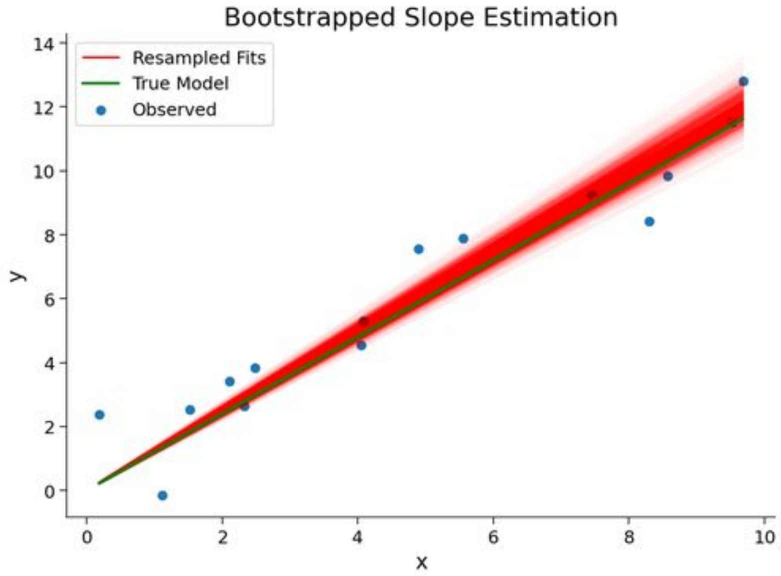
For a predetermined number of trials:

- Resample with replacement
- Solve the normal equation with the above values to obtain ϑ^[i]

OUTPUT: [1.27550888 1.17317819 1.18198819 1.25329255 1.20714664]

Visualise bootstrap estimates

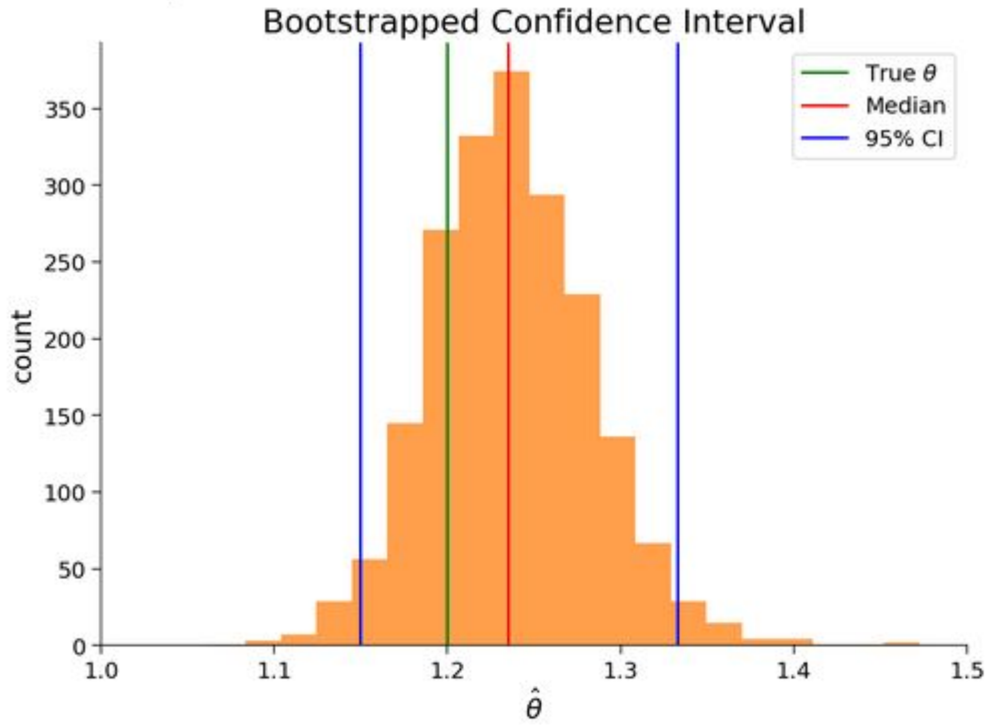
Now that we have our bootstrap estimates, we can visualize all the potential models (models computed with different resampling) together to see how distributed they are.



Bootstrapping confidence intervals

Looking at the distribution of bootstrapped ϑ^ values, we see that the true ϑ falls well within the 95% confidence interval, which is reassuring. We also see that the value ϑ=1 does not fall within the confidence interval. From this we would reject the hypothesis that the slope was 1.

```
mean = 1.24, std = 0.05
```



Observations

The bootstrapped estimates are spread around the true model.

Objective: Decipher ground truth value for ϑ from data therefore, assessing the quality of estimates based on finite data

Method: We quantify how uncertain our estimated slope is by computing confidence intervals (CIs) from our bootstrapped estimates. The most direct approach is to compute percentiles from the empirical distribution of bootstrapped estimates. (works for all data and data distributions since we are not assuming that this empirical distribution is Gaussian).

Readings

Computer Age Statistical Inference: Algorithms, Evidence and Data Science, by Bradley Efron and Trevor Hastie

Tutorial #4 Explanations

Multivariate case

Univariate case turned to the general linear regression case: more than one regressor, or feature, in our input.

Recall that our original univariate linear model was given as

$$y=\vartheta_x+\varepsilon$$

Where ϑ is the slope and ε some noise. We can easily extend this to the multivariate scenario by adding another parameter for each additional feature

$$y=\vartheta_0+\vartheta_1x_1+\vartheta_2x_2+\ldots+\vartheta_dx_d+\varepsilon$$

Where ϑ_0 is the intercept and d is the number of features (it is also the dimensionality of our input).

Multivariate case derivation

We can condense this succinctly using vector notation for a single data point

$$Yi = \vartheta^\top \mathbf{x}_i + \varepsilon$$

and fully in matrix form

$$\mathbf{y}=\mathbf{X}_\vartheta+\varepsilon$$

Where **y** is a vector of measurements, **X** is a matrix containing the feature values (columns) for each input sample (rows), and ***Θ*** is our parameter vector.

This matrix **X** is often referred to as the "design matrix".

Problem statement

Scientist records the spiking response of a retinal ganglion cell to patterns of light signals that vary in contrast and in orientation. Then contrast and orientation values can be used as features / regressors to predict the cells response.

In this case our model can be written as:

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

Matrix form:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} \\ 1 & x_{2,1} & x_{2,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{n,1} & x_{n,2} \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

Ordinary Least Squares

Recall our analytic solution to minimizing MSE for a single regressor:

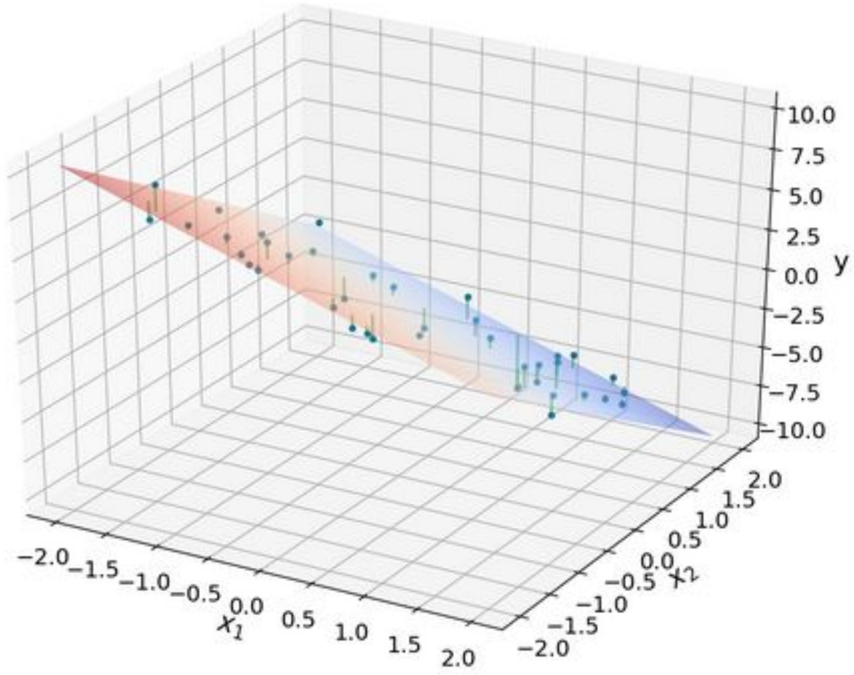
$$\hat{\theta} = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2}.$$

The same holds true for the multiple regressor case, only now expressed in matrix form

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

OUTPUT:
theta_cap = [0.13861386, -2.09395731, -3.16370742]

Geometric visualization of the data points (blue) and fitted plane.



Polynomial Regression

Linear regression can only capture a linear relationship between the inputs and outputs (predicted outputs are only a weighted sum of the inputs).

One model that is still very simple to fit and understand, but captures more complex relationships, is **polynomial regression**, an extension of linear regression.

We want to predict the dependent variable given the input values. The key change is the type of relationship between inputs and outputs that the model can capture.
Linear regression models predict the outputs as a weighted sum of the inputs:

$$y_n = \theta_0 + \theta x_n + \epsilon_n$$

With polynomial regression, we model the outputs as a polynomial equation based on the inputs. For example, we can model the outputs as:

$$y_n = \theta_0 + \theta_1 x_n + \theta_2 x_n^2 + \theta_3 x_n^3 + \epsilon_n$$

Design matrix for polynomial regression

The key difference between fitting a linear regression model and a polynomial regression model lies in how we structure the input variables.

For linear regression, we used X=x as the input data (X - design matrix). To add a constant bias (a y-intercept in a 2-D plot), we use X=[1, x], where 1 is a column of ones. When fitting, we learn a weight for each column of this matrix. So we learn a weight that multiples with column 1 - in this case that column is all ones so we gain the bias parameter (+ϑ_0). We also learn a weight for every column, or every feature of x.

Building Design matrices

We want to build our design matrix X for polynomial regression of order k as:

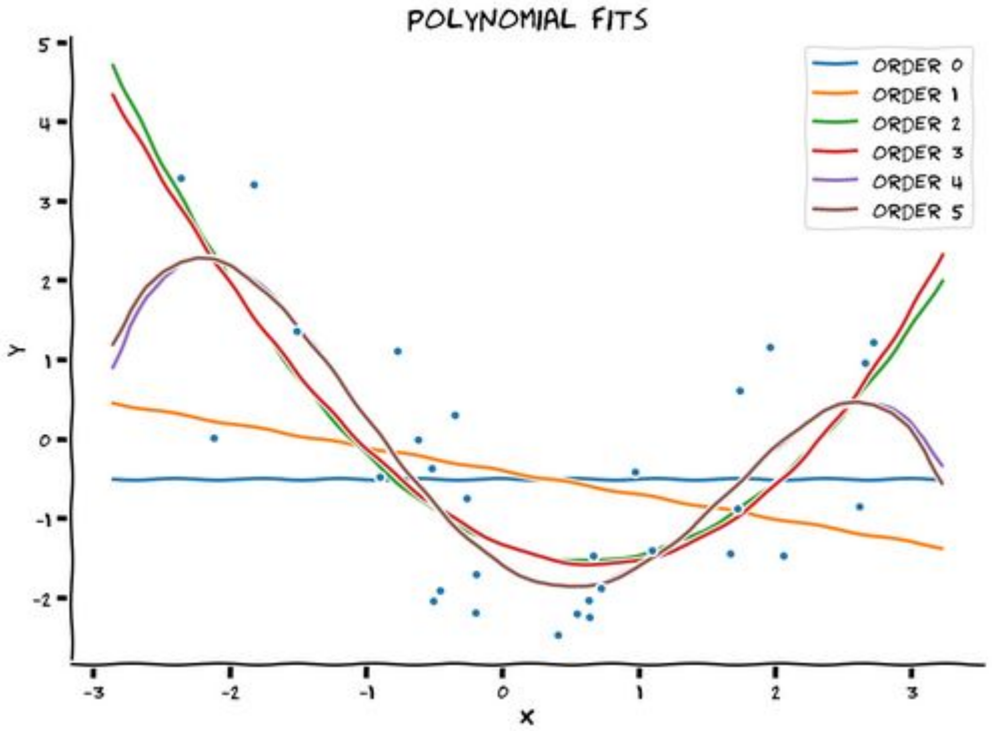
$$X = [\mathbf{1}, x^1, x^2, \ldots, x^k],$$

where 1 is the vector the same length as x consisting of all ones, and x^p is the vector or matrix x with all elements raised to the power p.
Note that $1=x^0$ and $x=x^1$

Fitting polynomial regression models

Here, we will fit polynomial regression models to find the regression coefficients (θ^0, θ^1, θ^2 , ...) by solving the least squares problem. We will then qualitatively inspect the quality of our fits for each order by plotting the fitted polynomials on top of the data. In order to see smooth curves, we evaluate the fitted polynomials on a grid of x values (ranging between the largest and smallest of the inputs present in the dataset).

Polynomial fit with different orders



Evaluating fit quality

As with linear regression, we can compute mean squared error (MSE) to get a sense of how well the model fits the data.

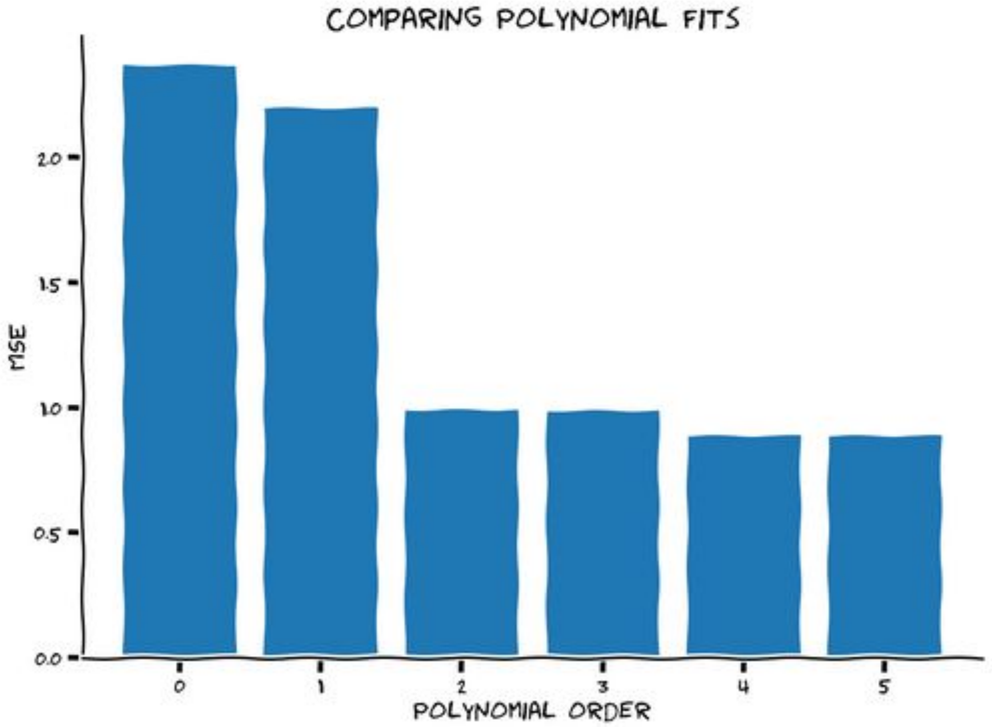
We compute MSE as:

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

where the predicted values for each model are given by

$$\hat{y} = X\hat{\theta}.$$

Comparing fits vs order



Tutorial #5 Explanations

Bias and Variance

The data used for the fitting procedure for a given model is the **training data**.

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).

Finding a good model can be difficult. One of the most important concepts to keep in mind when modeling is the **bias-variance tradeoff**.

Bias is the difference between the prediction of the model and the corresponding true output variables you are trying to predict. Models with high bias will not fit the training data well since the predictions are quite different from the true data. These high bias models are overly simplified - they do not have enough parameters and complexity to accurately capture the patterns in the data and are thus **underfitting**.

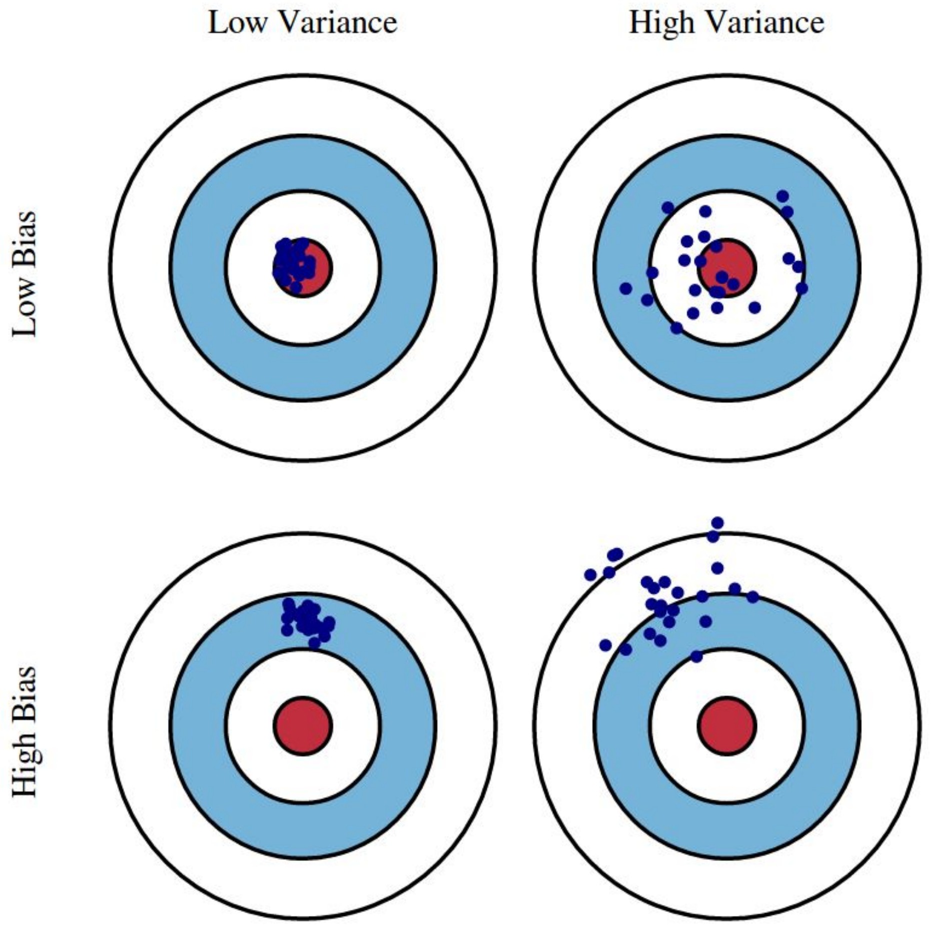
Variance refers to the variability of model predictions for a given input. Essentially, do the model predictions change a lot with changes in the exact training data used? Models with high variance are highly dependent on the exact training data used - they will not generalize well to test data. These high variance models are **overfitting** to the data.

Bias Variance Trade off

In essence:

- High bias, low variance models have high train and test error.
- Low bias, high variance models have low train error, high test error
- Low bias, low variance models have low train and test error

We ideally want low bias and low variance models! These goals can be in conflict though - models with enough complexity to have low bias also tend to overfit and depend on the training data more. We need to decide on the correct tradeoff.


Low Variance
High Variance
Low Bias
High Bias

Observations

More complex models (higher order polynomials) have lower MSE for training data. The overly simplified models (orders 0 and 1) have high MSE on the training data. As we add complexity to the model, we go from high bias to low bias.

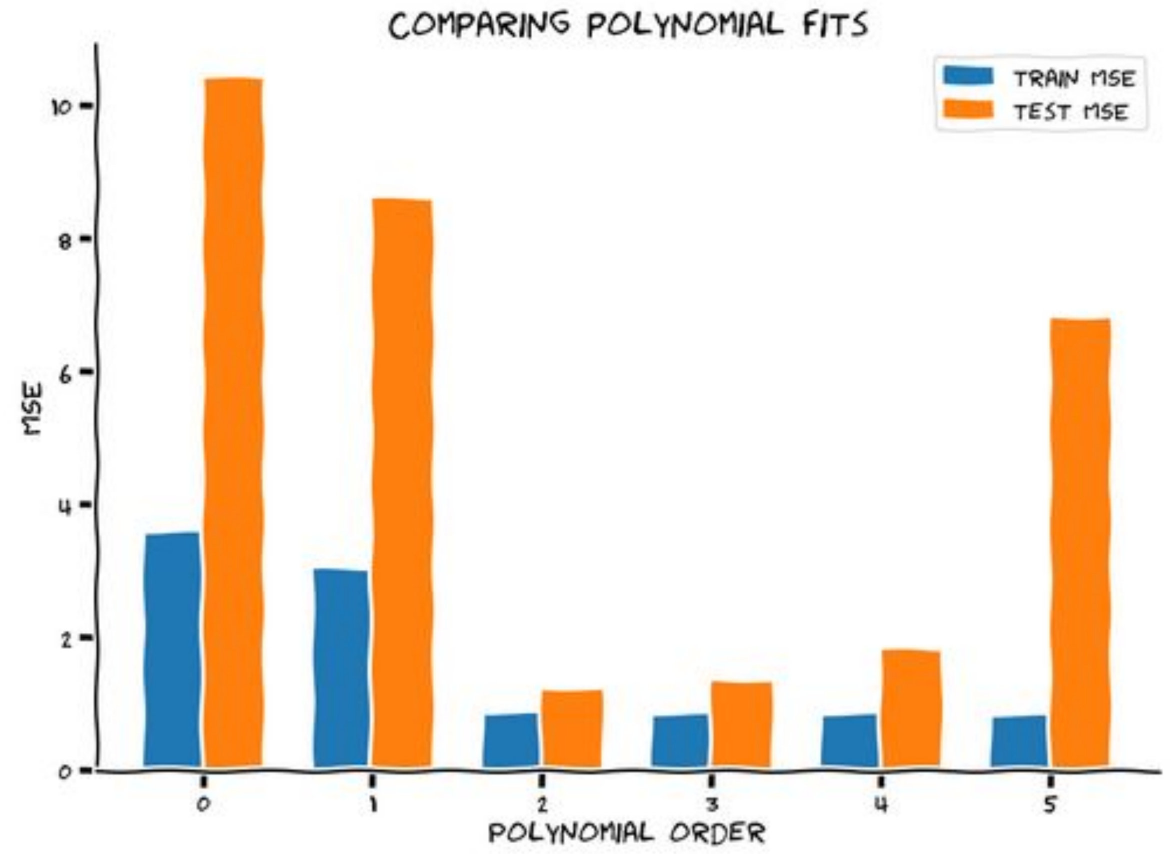
The MSE on test data follows a different pattern. The best test MSE is for an order 2 model - this makes sense as the data was generated with an order 2 model. Both simpler models and more complex models have higher test MSE.

So to recap:

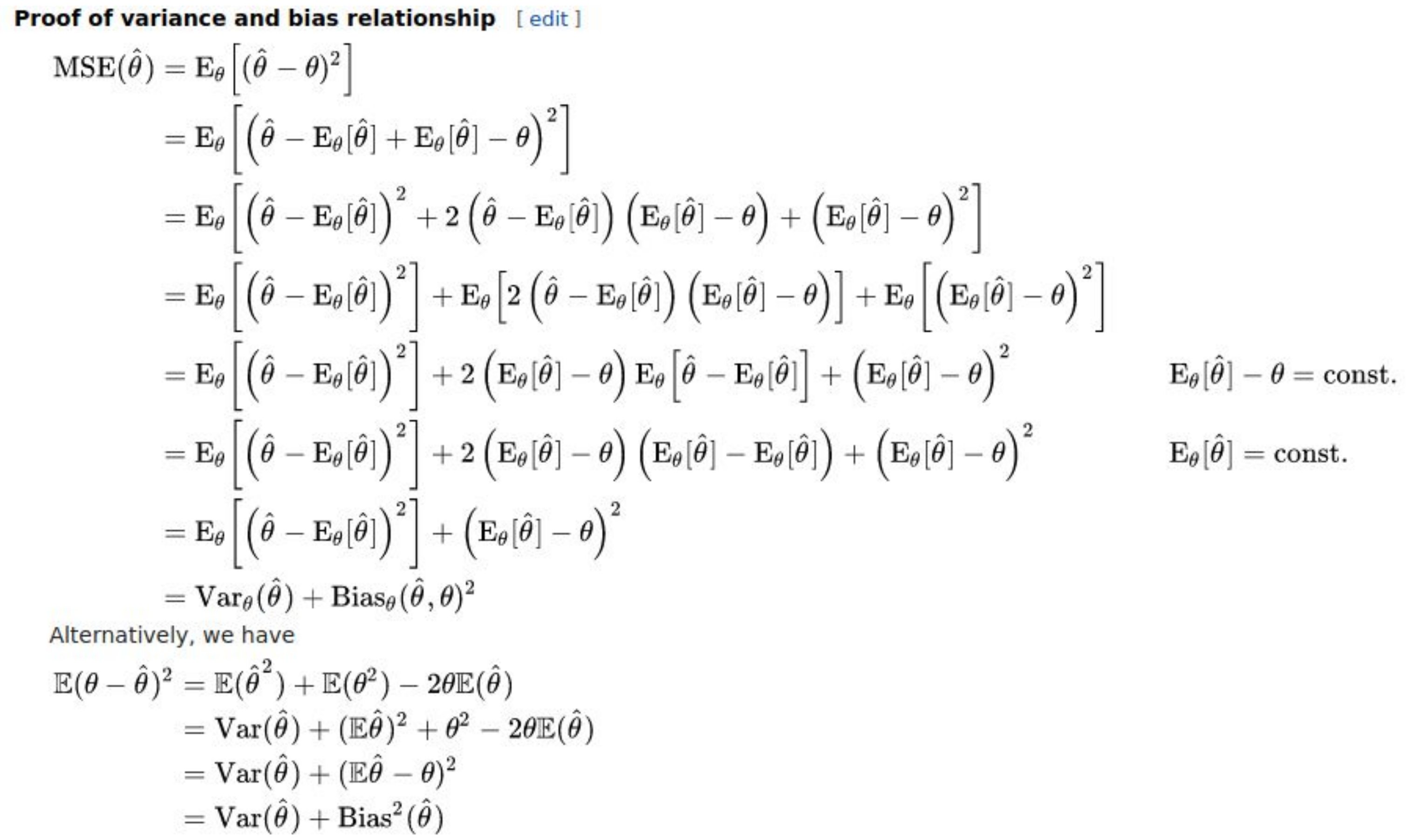
Order 0 model: High bias, low variance

Order 5 model: Low bias, high variance

Order 2 model: Just right, low bias, low variance


COMPARING POLYNOMIAL FITS
TRAIN MSE
TEST MSE
MSE
0
2
4
6
8
10
0
1
2
3
4
5
POLYNOMIAL ORDER

Bias-variance decomposition for MSE



Proof of variance and bias relationship [edit]

$$\begin{aligned}
\operatorname{MSE}(\hat{\theta}) &= \mathrm{E}_\theta\left[(\hat{\theta}-\theta)^2\right] \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]+\mathrm{E}_\theta[\hat{\theta}]-\theta\right)^2\right] \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]\right)^2+2\left(\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]\right)\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)+\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)^2\right] \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]\right)^2\right]+\mathrm{E}_\theta\left[2\left(\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]\right)\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)\right]+\mathrm{E}_\theta\left[\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)^2\right] \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]\right)^2\right]+2\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)\mathrm{E}_\theta\left[\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]\right]+\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)^2 && \mathrm{E}_\theta[\hat{\theta}]-\theta=\text{const.} \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]\right)^2\right]+2\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)\left(\mathrm{E}_\theta[\hat{\theta}]-\mathrm{E}_\theta[\hat{\theta}]\right)+\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)^2 && \mathrm{E}_\theta[\hat{\theta}]=\text{const.} \\
&= \mathrm{E}_\theta\left[\left(\hat{\theta}-\mathrm{E}_\theta[\hat{\theta}]\right)^2\right]+\left(\mathrm{E}_\theta[\hat{\theta}]-\theta\right)^2 \\
&= \operatorname{Var}_\theta(\hat{\theta})+\operatorname{Bias}_\theta(\hat{\theta},\theta)^2
\end{aligned}$$

Alternatively, we have

$$\begin{aligned}
\mathbb{E}(\theta-\hat{\theta})^2 &= \mathbb{E}(\hat{\theta}^2)+\mathbb{E}(\theta^2)-2\theta\mathbb{E}(\hat{\theta}) \\
&= \operatorname{Var}(\hat{\theta})+(\mathbb{E}\hat{\theta})^2+\theta^2-2\theta\mathbb{E}(\hat{\theta}) \\
&= \operatorname{Var}(\hat{\theta})+(\mathbb{E}\hat{\theta}-\theta)^2 \\
&= \operatorname{Var}(\hat{\theta})+\operatorname{Bias}^2(\hat{\theta})
\end{aligned}$$

Readings

The elements of statistical learning by Hastie, Tibshirani and Friedman

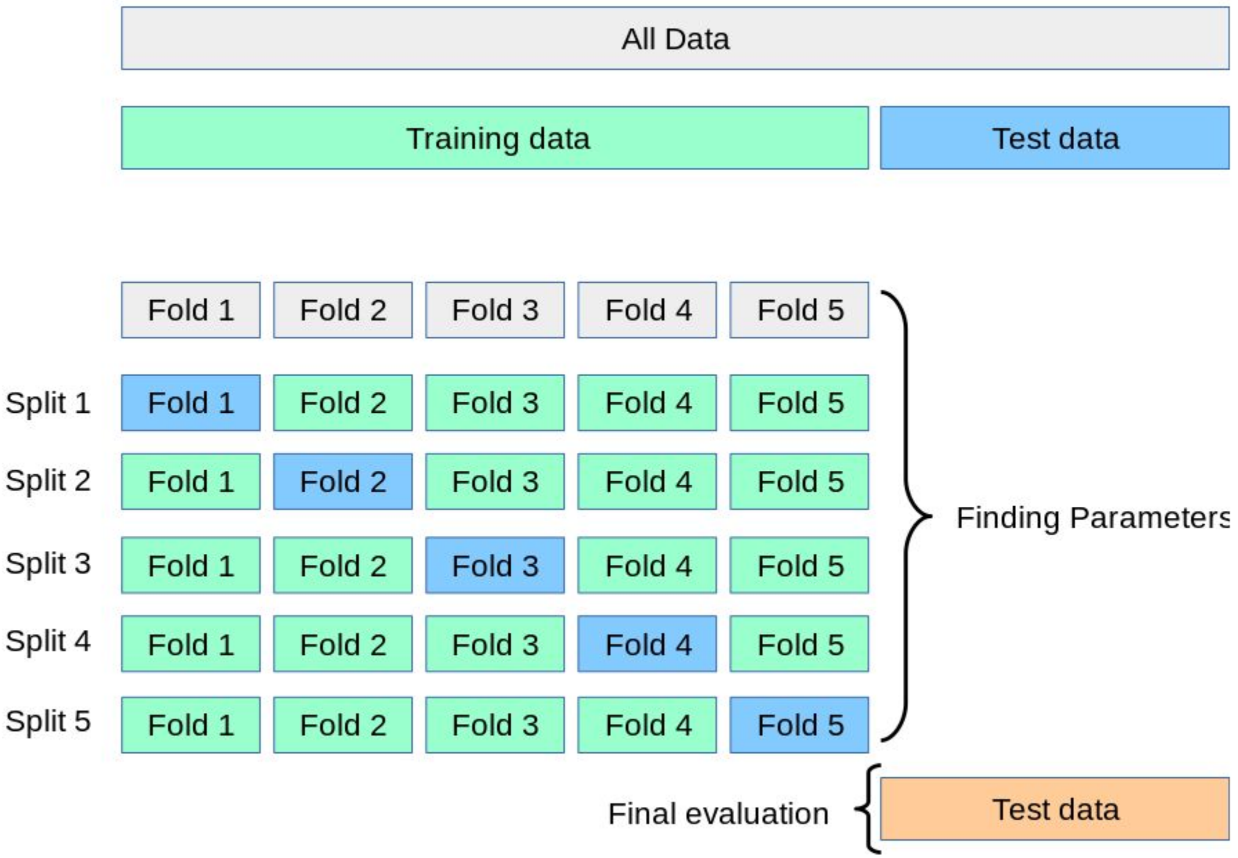
Tutorial #6
Explanations

Validation

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.

Note that the held-out subsets are called either validation or test subsets. There is no consensus and may depend on the exact use of k-fold cross validation. Sometimes people use k-fold cross validation to choose between different models/parameters to then apply to held-out test data and sometimes people report the averaged error on the held-out subsets as the model performance. If you are doing the former (using k-fold cross validation for model selection), you must report performance on held-out test data! In this text/code, we will refer to them as validation subsets to differentiate from our completely held-out test data.

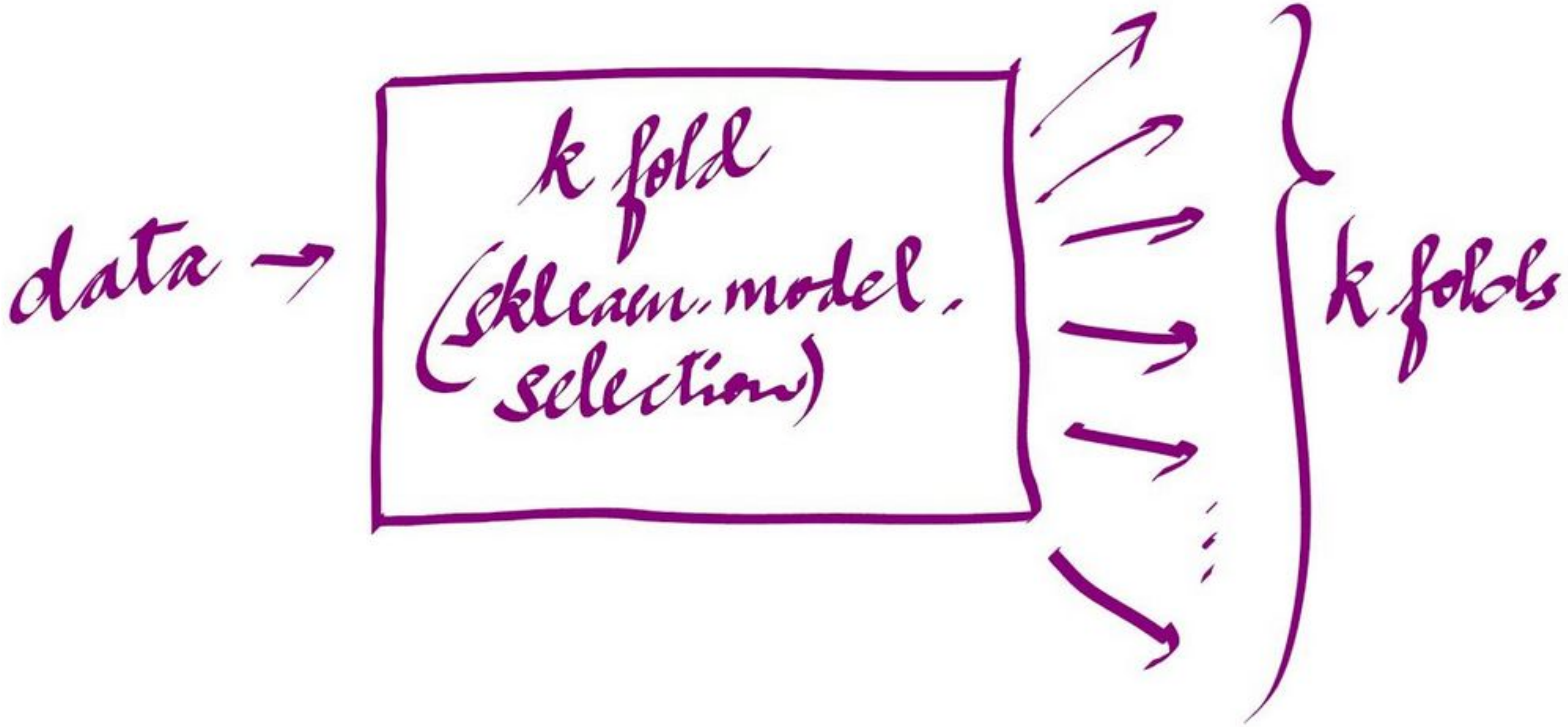

All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Split 2
Split 3
Split 4
Split 5
Finding Parameters
Final evaluation
Test data

Precautions and considerations on data division

Importantly, we need to be very careful when dividing the data into subsets. The held-out subset should not be used in any way to fit the model. We should not do any preprocessing (e.g. normalization) before we divide into subsets or the held-out subset could influence the training subsets. *A lot of false-positives in cross-validation come from wrongly dividing.*

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. 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.

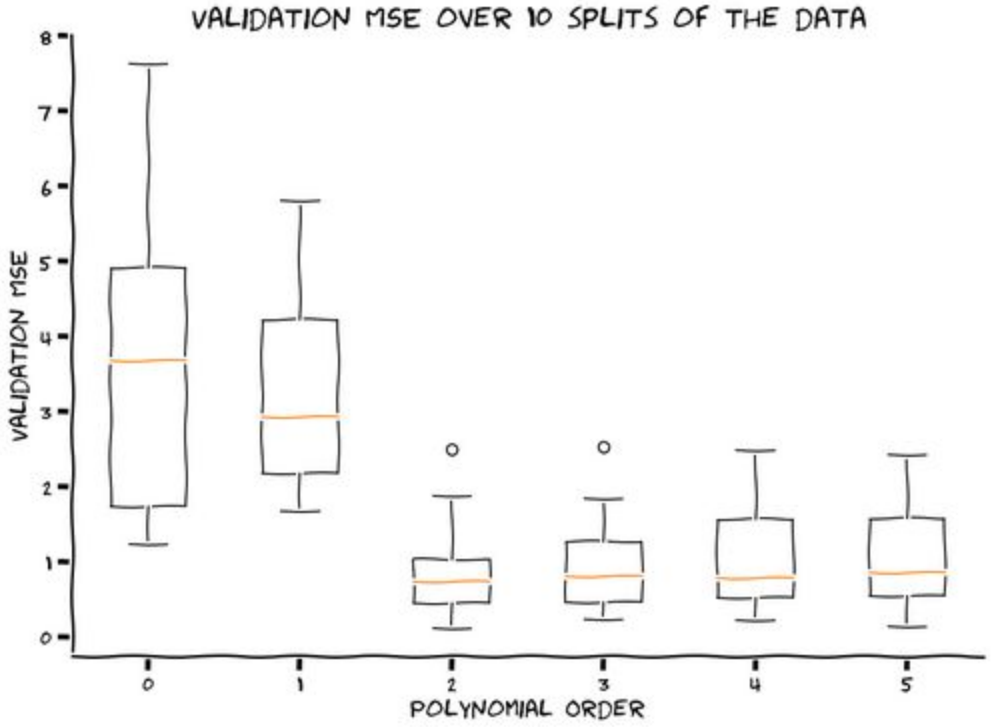
Implement cross validation



Implementation of kfold.split

The Kfold.split method returns an iterator which we can loop through. On each loop, this iterator assigns a different subset as validation and returns new training and validation indices with which to split the data.

We will loop through the 10 train/validation splits and fit several different polynomial regression models (with different orders) for each split.


VALIDATION MSE OVER 10 SPLITS OF THE DATA
VALIDATION MSE
POLYNOMIAL ORDER
0
1
2
3
4
5
6
7
8

AIC (Akaike's Information Criterion)

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 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). 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:

$$AIC = 2K - 2log(L)$$

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

Derivation: AIC

There is a link between mean squared error and the likelihood estimates for linear regression models that we can take advantage of.

We start with our formula for AIC from above:

$$AIC = 2k - 2logL$$

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

$$\log L = -\frac{n}{2}\log(2\pi) - \frac{n}{2}log(\sigma^2) - \sum_i^N \frac{1}{2\sigma^2}(y_i - \tilde{y}_i)^2$$

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:

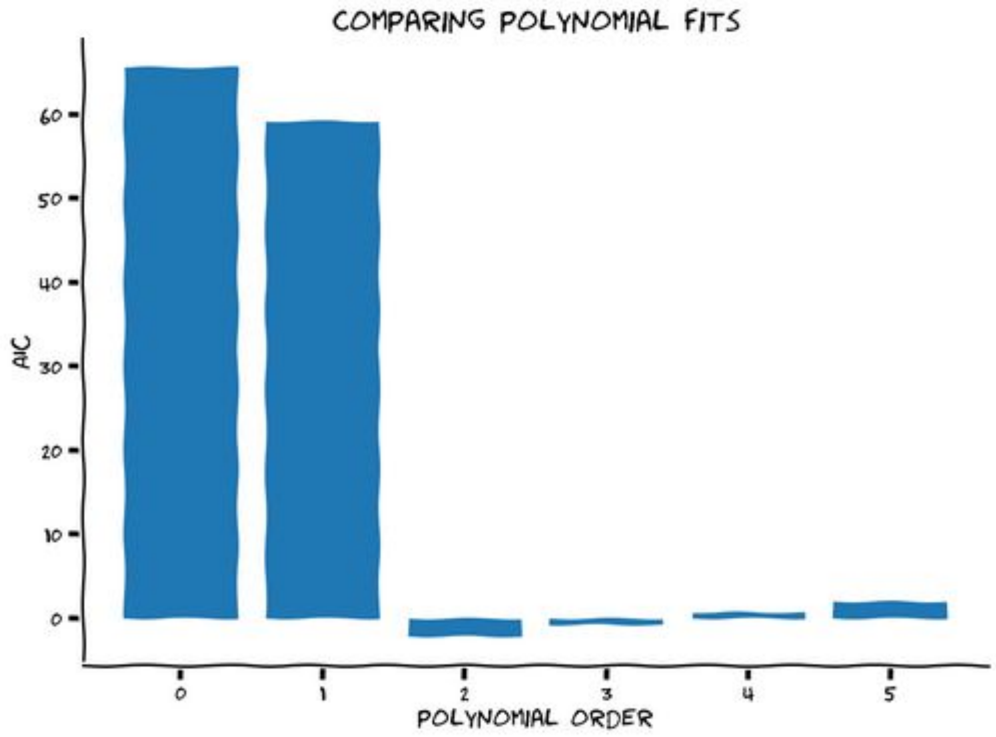
$$AIC = 2k + nlog(\sigma^2)$$

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:

$$AIC = 2K + nlog(\frac{SSE}{n})$$

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

Comparing polynomial fits



Observations

We compute the AIC for each model and choose the model with the lowest AIC (3).

Summary

Summary Lesson #1

- Linear least squares regression is an optimization procedure that can be used for data fitting:
  a. Task: predict a value for y given x
  b. Performance measure: MSE
  c. Procedure: minimize MSE by solving the normal equations
- **Key point**: We fit the model by defining an *objective function* and minimizing it.
- **Note**: In this case, there is an *analytical* solution to the minimization problem and in practice, this solution can be computed using *linear algebra*. This is *extremely* powerful and forms the basis for much of numerical computation throughout the sciences.

Summary Lesson #2

- Likelihood vs probability
  - $L(\vartheta|x,y)=p(y|\vartheta,x)$
  - $p(y|\vartheta,x)$ -> "probability of observing the response y given parameter ϑ and input x"
  - $L(\vartheta|x,y)$ -> "likelihood model that parameters ϑ produced response y from input x"
- Log-likelihood maximization
  - We take the log of the likelihood function for computational convenience
  - The parameters ϑ that maximize $\log L(\vartheta|x,y)$ are the model parameters that maximize the probability of observing the data.
- **Key point**:
  - the log-likelihood is a flexible cost function, and is often used to find model parameters that best fit the data.

Summary Lesson #3

- Bootstrapping is a resampling procedure that allows to build confidence intervals around inferred parameter values
- it is a widely applicable and very practical method that relies on computational power and pseudo-random number generators (as opposed to more classical approaches than depend on analytical derivations)

Summary Lesson #4

- Linear regression generalizes naturally to multiple dimensions
- Linear algebra affords us the mathematical tools to reason and solve such problems beyond the two dimensional case

- To change from a linear regression model to a polynomial regression model, we only have to change how the input data is structured

- We can choose the complexity of the model by changing the order of the polynomial model fit

- Higher order polynomial models tend to have lower MSE on the data they're fit with

Note: In practice, multidimensional least squares problems can be solved very efficiently (thanks to numerical routines such as LAPACK).

Summary Lesson #5

- Training data is the data used for fitting, test data is held-out data.
- We need to strike the right balance between bias and variance. Ideally we want to find a model with optimal model complexity that has both low bias and low variance
    - Too complex models have low bias and high variance.
    - Too simple models have high bias and low variance.

Note

- Bias and variance are very important concepts in modern machine learning, but it has recently been observed that they do not necessarily trade off (see for example the phenomenon and theory of "double descent")

Summary Lesson #6

We need to use model selection methods to determine the best model to use for a given problem.

Cross-validation focuses on how well the model predicts new data.