An Introduction to Machine Learning Concepts with Linear Regression

Nathan K. Chan, HBSc, MSc

Motivation

- Scientists frequently use **statistical models to infer associations** between dependent and independent variables

- **Linear regression is a ubiquitous model** in scientific publications
  - E.g. generalized linear models, ANOVA, ANCOVA, t-tests, etc.

- Let’s use linear regression as a “launching point” for understanding key concepts in machine learning.

Learning Outcomes

- To understand **loss functions** and their applications
- To describe the **bias-variance trade-off**
- To describe **hyperparameters** and **hyperparameter tuning**
- To understand **cross-validation**
- To introduce *very* basic machine learning algorithms (if time permits)

Part 1:
Review of Linear Regression

Multivariable linear regression

Given n observations of p explanatory variables from data set:

$$\{y_i, x_{i1}, \cdots, x_{ip}\}_{i=1}^{n}, (m = 1, \cdots, p)$$

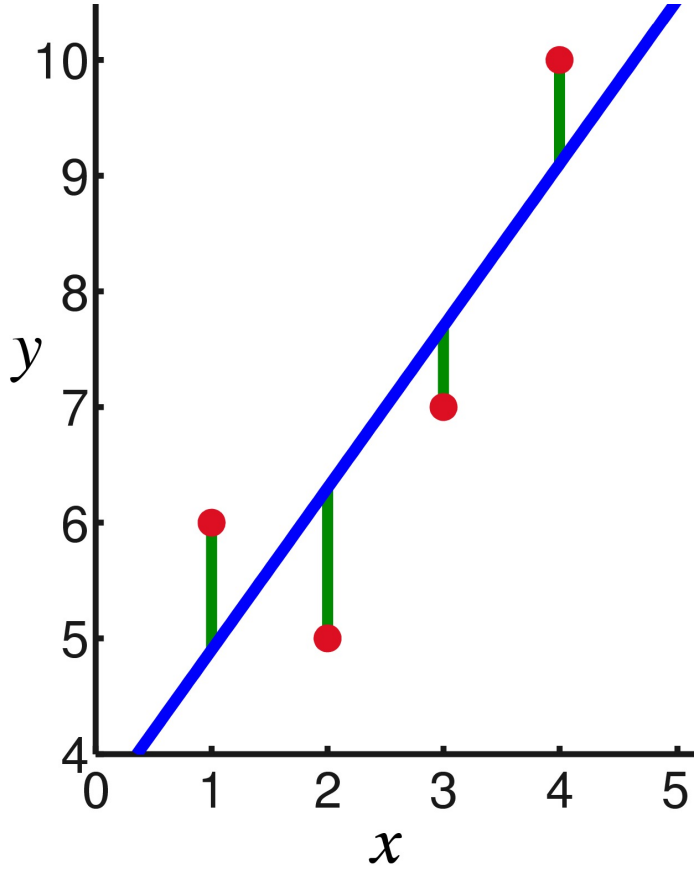
We can determine how linear changes of each x_m vary with $\boldsymbol{y}$ given its respective coefficient β_m:

$$y_i = \beta_0 1 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \sum_{m=1}^{p} \beta_m x_{im} + \varepsilon_i$$

$$\boldsymbol{y} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}$$

- y_i is the i-th observation of the response variable
- x_{im} is the i-th observation of the m-th explanatory variable
- β_m is the partial derivative of $\boldsymbol{y}$ with respect to the m-th explanatory variable
  - *i.e.* the coefficient of the m-th explanatory variable
- ε_i is the i-th residual of y_i given the sum of all $\beta_m x_{im}$

We don't know the values of $\boldsymbol{\beta}$; how can we estimate $\boldsymbol{\beta}$?



Source: https://en.wikipedia.org/wiki/Linear_regression#/media/File:Linear_least_squares_example2.png

Loss functions

$\boldsymbol{\varepsilon}$ is the residual of $\boldsymbol{y}$ given $\mathbf{X}\boldsymbol{\beta}$.

$$\boldsymbol{\varepsilon} = \boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}$$

Multiple solutions of $\boldsymbol{\beta}$ are possible.
We need a "loss function" such that a solution for $\boldsymbol{\beta}$ is defined.

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} S(\boldsymbol{\beta})$$

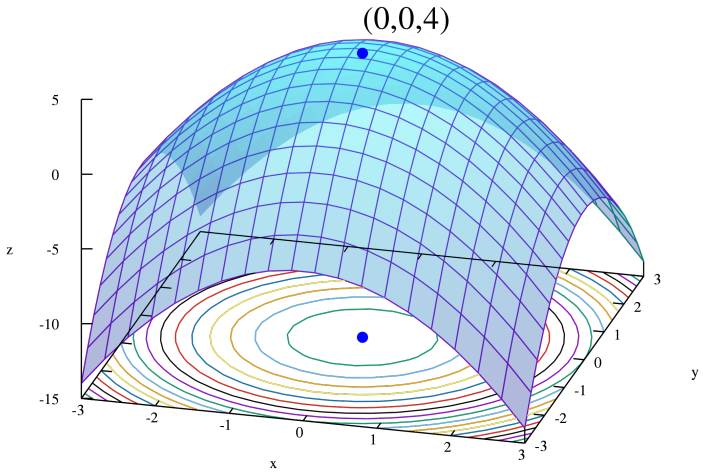
- S is a "loss function" of $\boldsymbol{\beta}$ that constrains $\boldsymbol{\beta}$ under some condition
- $\widehat{\boldsymbol{\beta}}$ (read as "beta-hat") is the value of $\boldsymbol{\beta}$ that minimizes $S(\boldsymbol{\beta})$

$\widehat{\boldsymbol{\beta}}$ provides "fitted values" of $\boldsymbol{y}$ and lets us define residuals from fits:

$$\widehat{\boldsymbol{y}} = \mathbf{X}\widehat{\boldsymbol{\beta}}$$

$$\hat{\boldsymbol{\varepsilon}} = \mathbf{y} - \mathbf{X}\widehat{\boldsymbol{\beta}} = \boldsymbol{y} - \widehat{\boldsymbol{y}}$$

- $\widehat{\boldsymbol{y}}$ is the fitted/predicted value of $\boldsymbol{y}$ given $\mathbf{X}\widehat{\boldsymbol{\beta}}$
- $\hat{\boldsymbol{\varepsilon}}$ is the residual of $\boldsymbol{y}$ given $\widehat{\boldsymbol{y}}$



Source:
https://en.wikipedia.org/wiki/Mathematical_optimization#/media/File:Max_paraboloid.svg

Ordinary Least Squares (OLS)

The OLS estimator defines the loss function $S(\boldsymbol{\beta})$ as:

$$S_{OLS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left| y_i - \sum_{m=1}^{p} x_{im}\beta_m \right|^2 = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = \|\boldsymbol{\varepsilon}\|_2^2$$

This is the "**residual sum of squares**" (RSS).
It is the remaining variance of $\boldsymbol{y}$ given the explanatory variables.

Thus, the OLS estimator of $\widehat{\boldsymbol{\beta}}$ is defined as:

$$S_{OLS}(\widehat{\boldsymbol{\beta}}_{OLS}) = \left\|\boldsymbol{y} - \mathbf{X}\widehat{\boldsymbol{\beta}}_{OLS}\right\|_2^2 = \|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|_2^2 = \|\hat{\boldsymbol{\varepsilon}}\|_2^2$$

$$\widehat{\boldsymbol{\beta}}_{OLS} = \arg\min_{\boldsymbol{\beta}} S_{OLS}(\boldsymbol{\beta})$$

What properties make $\widehat{\boldsymbol{\beta}}$ from S_{OLS} an "ideal" $\boldsymbol{\beta}$?
(Why is OLS the way we've done regression for so long?)

If "ideal" means a "**Best Linear Unbiased Estimator**"…

The $\|\quad\|_2$ notation is called the "L^2 norm", defined as:

$$\|\boldsymbol{\theta}\|_2 = \sqrt{\theta_1^2 + \cdots + \theta_n^2}$$

For n observations and p explanatory variables, the "mean square error" (MSE) is defined as:

$$MSE = \frac{RSS}{df_{RSS}}, df_{RSS} = n - p$$

- df_{RSS} is the degrees of freedom of the RSS

If df_{RSS} is constant, then minimizing RSS also minimizes MSE.

$$\arg\min_{\theta} RSS = \arg\min_{\theta} MSE$$

Best Linear Unbiased Estimators (BLUEs)

$\widehat{\boldsymbol{\theta}}$ is an estimator with true value $\boldsymbol{\theta}$.

Define "mean square error" (MSE) as:

$$\mathrm{MSE}(\widehat{\boldsymbol{\theta}}) = \mathrm{E}\left[(\widehat{\boldsymbol{\theta}} - \boldsymbol{\theta})^2\right] = \mathrm{Var}(\widehat{\boldsymbol{\theta}}) + \mathrm{Bias}(\widehat{\boldsymbol{\theta}}, \boldsymbol{\theta})^2$$

An estimator is "**unbiased**" when its expectation is equal to zero:

$$\mathrm{Bias}(\widehat{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \mathrm{E}[\widehat{\boldsymbol{\theta}}] - \boldsymbol{\theta} = 0$$

The "**best**" estimator is one that minimizes MSE:

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$$

If an estimator is unbiased, it implies that:

$$\mathrm{MSE}(\widehat{\boldsymbol{\theta}}_{unbiased}) = \mathrm{Var}(\widehat{\boldsymbol{\theta}}_{unbiased}) \because \mathrm{Bias}(\widehat{\boldsymbol{\theta}}_{unbiased}, \boldsymbol{\theta}) = 0$$

Therefore, a BLUE is an estimator that minimizes its variance.

$$\therefore \widehat{\boldsymbol{\theta}}_{BLUE} = \arg\min_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathrm{Var}(\boldsymbol{\theta})$$

Under the assumptions that…

1. $\mathrm{E}[\boldsymbol{\varepsilon}] = 0$ (residual mean is zero);
2. $\mathrm{Var}(\boldsymbol{\varepsilon}) < \infty$ (residual variance is finite);
3. $\mathrm{Cov}(\varepsilon_i, \varepsilon_j) = 0, \forall\, i \neq j$ (residuals are uncorrelated);

The OLS estimator,

$$\widehat{\boldsymbol{\beta}}_{OLS} = \arg\min_{\boldsymbol{\beta}} S_{OLS}(\boldsymbol{\beta})$$

can be proven* to be the BLUE.

*See Gauss-Markov Theorem and proof:
https://web.stanford.edu/class/stats253/lectures/lect2.pdf

To summarize…

Regressions require a loss function, S, to estimate the coefficients for regression, $\widehat{\boldsymbol{\beta}}$.

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} S(\boldsymbol{\beta})$$

"**Best**":

$$\widehat{\boldsymbol{\beta}}_{OLS} = \arg\min_{\boldsymbol{\beta}} S_{OLS}(\boldsymbol{\beta}) = \arg\min_{\boldsymbol{\beta}} MSE\ (\boldsymbol{\beta})$$

The OLS estimator of $\widehat{\boldsymbol{\beta}}$ is a ***best*** linear ***unbiased*** estimator of $\boldsymbol{\beta}$.

"**Unbiased**":

$$\mathrm{Bias}\left(\widehat{\boldsymbol{\beta}}_{OLS}, \boldsymbol{\beta}\right) = \mathrm{E}\left[\widehat{\boldsymbol{\beta}}_{OLS}\right] - \boldsymbol{\beta} = 0$$

$$\therefore \mathrm{MSE}\left(\widehat{\boldsymbol{\beta}}_{OLS}\right) = \mathrm{Var}\left(\widehat{\boldsymbol{\beta}}_{OLS}\right)$$

OLS estimates $\boldsymbol{\beta}$ by minimizing the squared error of the model — the L^2 norm of the residuals.

$$S_{OLS}\left(\widehat{\boldsymbol{\beta}}_{OLS}\right) = \left\|\boldsymbol{y} - \mathbf{X}\widehat{\boldsymbol{\beta}}_{OLS}\right\|_2^2 = \|\hat{\boldsymbol{\varepsilon}}\|_2^2$$

$$\widehat{\boldsymbol{\beta}}_{OLS} = \arg\min_{\boldsymbol{\beta}} S_{OLS}(\boldsymbol{\beta}_{OLS})$$

Part 2:
Bias and Regularization

Best Linear Estimators

Recall that "best" means minimizing MSE:

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$$

$$\mathrm{MSE}(\widehat{\boldsymbol{\theta}}) = \mathrm{E}\left[(\widehat{\boldsymbol{\theta}} - \boldsymbol{\theta})^2\right] = \mathrm{Var}(\widehat{\boldsymbol{\theta}}) + \mathrm{Bias}(\widehat{\boldsymbol{\theta}}, \boldsymbol{\theta})^2$$

$$\mathrm{MSE}(\widehat{\boldsymbol{\theta}}_{unbiased}) = \mathrm{Var}(\widehat{\boldsymbol{\theta}}_{unbiased}) \because \mathrm{Bias}(\widehat{\boldsymbol{\theta}}_{unbiased}, \boldsymbol{\theta}) = 0$$

Issue 1:

A better estimator may exist if we relax the "unbiased" condition – a **best linear** ~~unbiased~~ **estimator**.

We can introduce bias to reduce the coefficients of less important predictors by "penalizing" (*i.e.* "regularizing") $\boldsymbol{\beta}$ in the least squares algorithm.

How do we **penalize** less important predictors?

$$\mathrm{MSE}(\widehat{\boldsymbol{\theta}}_{with\ bias}) \overset{?}{\leq} \mathrm{MSE}(\widehat{\boldsymbol{\theta}}_{unbiased})$$

Issue 2:

Consider an "overdetermined" regression with many predictors.

If any predictors are perfectly collinear, then OLS cannot provide a unique solution. Random predictors could also "take" variance from true predictors, reducing the variance attributed to any individual predictor.

How do we ***select*** the best predictors?

Regularized/Penalized Least Squares (RLS/PLS)

The OLS estimator defines the loss function $S(\boldsymbol{\beta})$ as:

$$S_{OLS}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

Recall…

$$\mathrm{MSE}(\widehat{\boldsymbol{\theta}}) = \mathrm{E}\left[\left(\widehat{\boldsymbol{\theta}} - \boldsymbol{\theta}\right)^2\right] = \mathrm{Var}(\widehat{\boldsymbol{\theta}}) + \mathrm{Bias}(\widehat{\boldsymbol{\theta}}, \boldsymbol{\theta})^2$$

If bias is added as a constraint on $\boldsymbol{\beta}$, it limits the values $\boldsymbol{\beta}$ can take for solving the loss function $S_{OLS}(\boldsymbol{\beta})$.

Let's try adding a term to penalize $\boldsymbol{\beta}$.

$$S_{RLS}(\boldsymbol{\beta}) = S_{OLS}(\boldsymbol{\beta}) + \lambda R(\boldsymbol{\beta})$$

- S_{RLS} is the loss function for regularized least squares (RLS)
- R is the "penalty function" or "regularization function"
- λ is the weight of the penalty function for $R(\boldsymbol{\beta})$

The $\| \quad \|_2$ notation is called the "L^2 norm", a.k.a. the "Euclidean distance", defined as:

$$\|\boldsymbol{\theta}\|_2 = \sqrt{\theta_1^2 + \cdots + \theta_n^2}$$

The general form, $\| \quad \|_p$, is called the "L^p norm":

$$\|\boldsymbol{\theta}\|_p = \left(\sum_{i=1}^{n} |\theta_i|^p\right)^{\frac{1}{p}}$$

Thus, the "L^1 norm", a.k.a. the "Manhattan distance", is defined as:

$$\|\boldsymbol{\theta}\|_1 = \sum_{i=1}^{n} |\theta_i|$$

*LASSO is the “Least Absolute Shrinkage and Selection Operator”.

Regularization/Penalty functions

$$S_{RLS}(\boldsymbol{\beta}) = S_{OLS}(\boldsymbol{\beta}) + \lambda R(\boldsymbol{\beta})$$

How do we define $R(\boldsymbol{\beta})$?

We can constrain $\boldsymbol{\beta}$ such that the squared L^2 norm does not exceed t:

$$\|\boldsymbol{\beta}\|_2^2 \leq t$$

$$R_{Ridge}(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_2^2$$

We can constrain $\boldsymbol{\beta}$ such that the L^1 norm does not exceed t:

$$\|\boldsymbol{\beta}\|_1 \leq t$$

$$R_{LASSO}(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1$$

- t is some constraint that limits a function of $\boldsymbol{\beta}$

- $\|\boldsymbol{\beta}\|_2^2$ is the squared absolute sum of the coefficients.

- $\|\boldsymbol{\beta}\|_1$ is the absolute sum of the coefficients.

Regression with this penalized and squared L^2 norm is called “ridge regression”:

$$S_{Ridge}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2$$

$$\widehat{\boldsymbol{\beta}}_{Ridge} = \arg\min_{\boldsymbol{\beta}} S_{Ridge}(\boldsymbol{\beta})$$

Regression with this penalized L^1 norm is called “LASSO”*:

$$S_{LASSO}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1$$

$$\widehat{\boldsymbol{\beta}}_{LASSO} = \arg\min_{\boldsymbol{\beta}} S_{LASSO}(\boldsymbol{\beta})$$

t and λ are described as “**hyperparameters**”.
The relationship between t and λ is data dependent.

λ is any non-negative real number

Recall:

$$S_{Ridge}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2$$

$$S_{LASSO}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1$$

If λ is very small, ridge regression and LASSO approximate OLS.

$$\lim_{\lambda\to 0} S_{Ridge}(\boldsymbol{\beta}) = \lim_{\lambda\to 0}(\|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = S_{OLS}(\boldsymbol{\beta})$$

$$\lim_{\lambda\to 0} S_{LASSO}(\boldsymbol{\beta}) = \lim_{\lambda\to 0}(\|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = S_{OLS}(\boldsymbol{\beta})$$

λ can take any value defined in the set of non-negative real numbers:

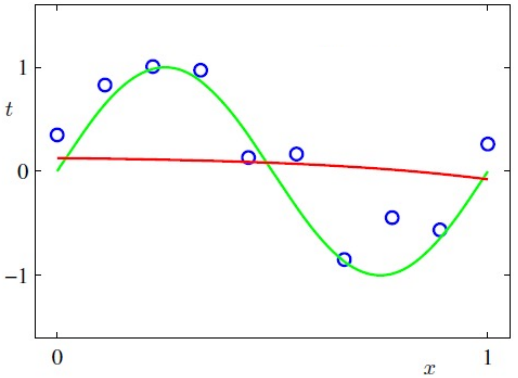
$$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$$

$$\boldsymbol{\lambda} = \{\lambda \in \mathbb{R}_{\geq 0}\}$$

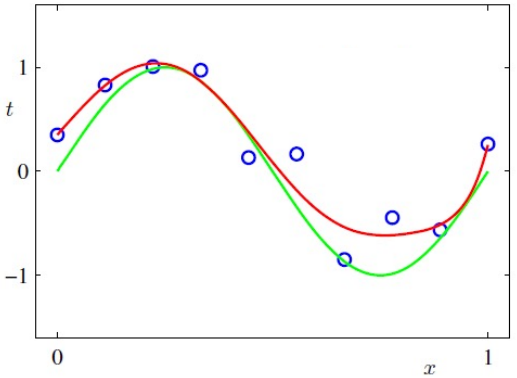
Source: https://stats.stackexchange.com/questions/108364/demonstration-of-benefits-of-ridge-regression-over-ordinary-regression

Predictors from a 9th order polynomial fit

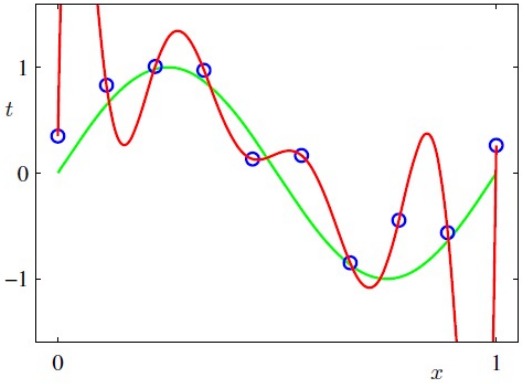
$\lambda = 1$
$\ln(\lambda) = 0$



$\lambda = 1.523 \cdot 10^{-8}$
$\ln(\lambda) = -18$



$\lambda = 0$
$\ln(\lambda) = -\infty$



Why do we constrain $\|\boldsymbol{\beta}\|_2^2$ and $\|\boldsymbol{\beta}\|_1$?

Consider a model with 2 highly collinear predictors:

$$\boldsymbol{\beta} = \{\beta_1, \beta_2\}$$

The squared L^2 norm is the formula for a circle (n-sphere):

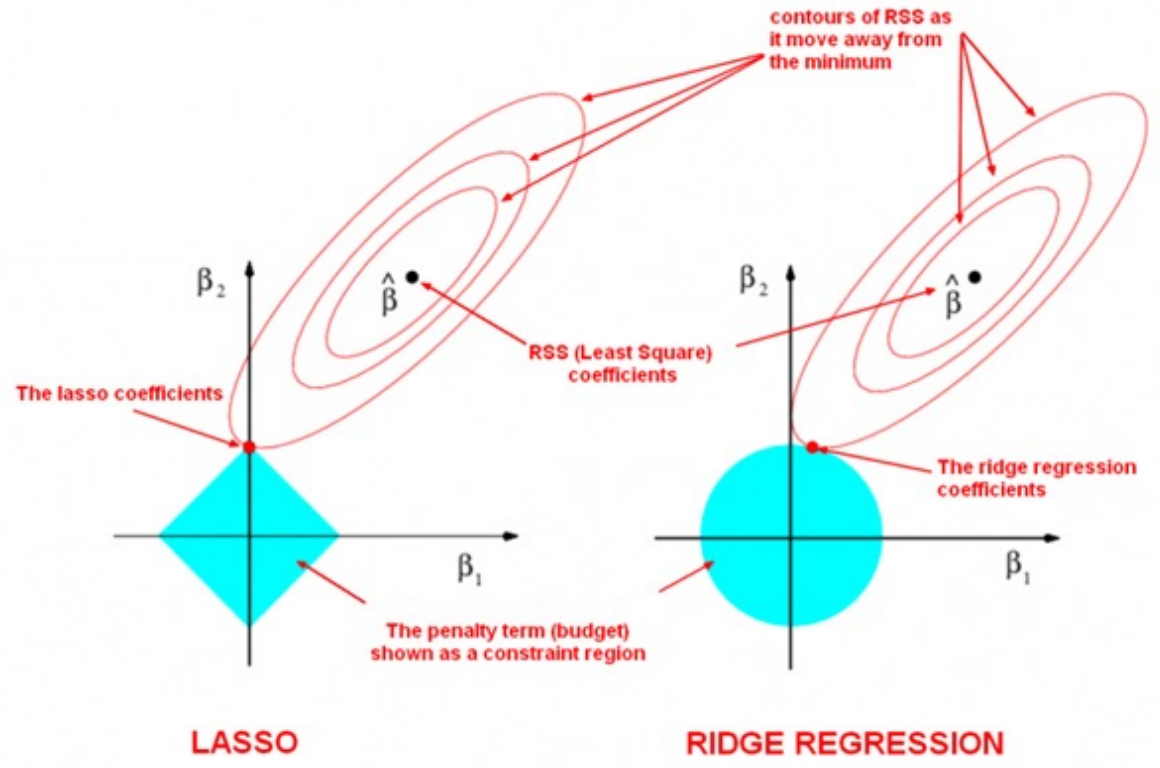
$$\|\boldsymbol{\beta}\|_2^2 = \beta_1^2 + \beta_2^2 \leq t$$
$$r^2 = x^2 + y^2$$

Thus, ridge regression **penalizes** the "less important" coefficients in sets of collinear predictors by reducing the coefficients of worse predictors.

The L^1 norm is the formula for a square (hypercube):

$$\|\boldsymbol{\beta}\|_1 = |\beta_1| + |\beta_2| \leq t$$
$$r = x + y$$

Thus, LASSO **penalizes** and **selects** the "less important" coefficients in sets of collinear predictors by setting some coefficients exactly to zero.



Source: https://www.quora.com/How-would-you-describe-the-difference-between-linear-regression-lasso-regression-and-ridge-regression

To summarize…

Introducing bias can provide better model fits, *i.e.* produce coefficients whose minimized MSE is less than the minimized MSE from BLUEs.

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$$

$$\mathrm{MSE}(\widehat{\boldsymbol{\theta}}) = \mathrm{E}\left[(\widehat{\boldsymbol{\theta}} - \boldsymbol{\theta})^2\right] = \mathrm{Var}(\widehat{\boldsymbol{\theta}}) + \mathrm{Bias}(\widehat{\boldsymbol{\theta}}, \boldsymbol{\theta})^2$$

$$\mathrm{MSE}(\widehat{\boldsymbol{\theta}}_{Bias\neq 0}) \overset{?}{\leq} \mathrm{MSE}(\widehat{\boldsymbol{\theta}}_{Bias=0})$$

Ridge regression penalizes coefficients of "worse" collinear predictors to potentially provide better regression fits.

$$S_{Ridge}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2$$

$$\widehat{\boldsymbol{\beta}}_{Ridge} = \arg\min_{\boldsymbol{\beta}} S_{Ridge}(\boldsymbol{\beta})$$

LASSO penalizes and introduces sparsity in the regression by setting some coefficients exactly to zero. **LASSO is a feature selector.**

$$S_{LASSO}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1$$

$$\widehat{\boldsymbol{\beta}}_{LASSO} = \arg\min_{\boldsymbol{\beta}} S_{LASSO}(\boldsymbol{\beta})$$

$\boldsymbol{\lambda}$ is any **non-negative real number**.

$$\boldsymbol{\lambda} = \{\lambda \in \mathbb{R}_{\geq 0}\}$$

Part 3:
Statistics to Machine Learning

Bias-Variance Trade-off

Recall:

$$\mathrm{MSE}(\hat{\boldsymbol{\theta}}) = \mathrm{Var}(\hat{\boldsymbol{\theta}}) + \mathrm{Bias}(\hat{\boldsymbol{\theta}}, \boldsymbol{\theta})^2$$

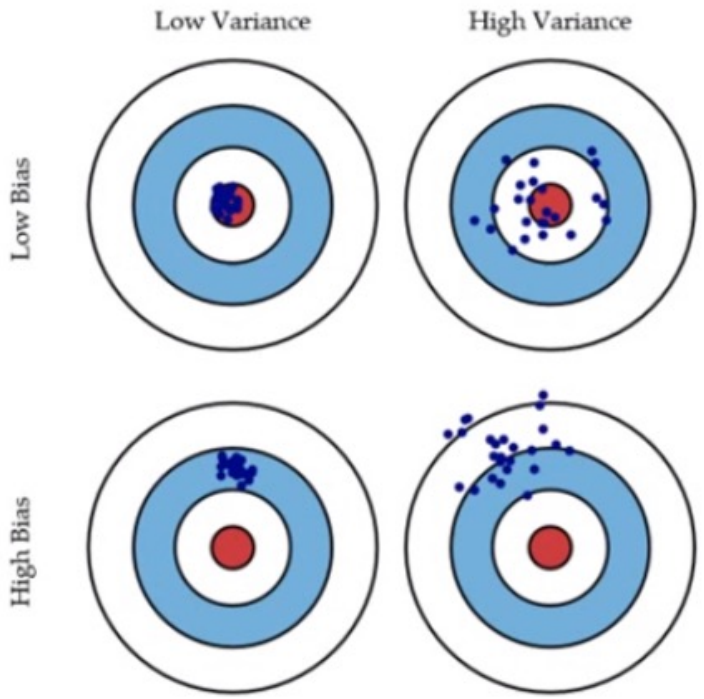
Assume MSE is constant.
If variance is large, bias must be small.
If bias is large, variance must be small.

Recall:

$$S_{RLS}(\boldsymbol{\beta}) = S_{OLS}(\boldsymbol{\beta}) + \lambda R(\boldsymbol{\beta})$$

If λ is small, bias is low.
Few predictors are penalized, and we risk "overfitting": modelling the noise rather than the pattern.

If λ is large, bias is high.
Many predictors are penalized, and we risk "underfitting": failing to model the pattern.

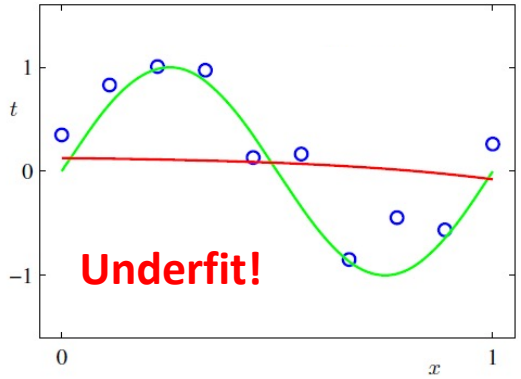


Source:
https://www.datasciencecentral.com/profiles/blogs/intuition-behind-bias-variance-trade-off-lasso-and-ridge

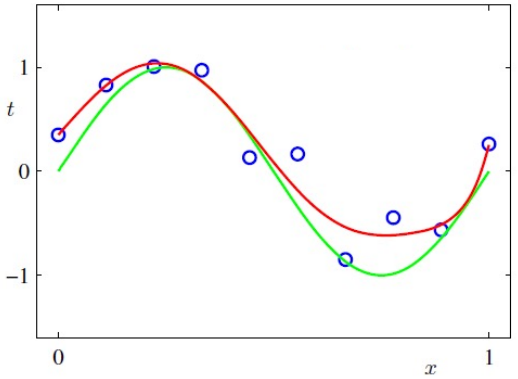
Source:
https://stats.stackexchange.com/questions/108364/demonstration-of-benefits-of-ridge-regression-over-ordinary-regression

Over- and Underfitting

Predictors from a 9th order polynomial fit



$\lambda = 1$
$\ln(\lambda) = 0$



$\lambda = 1.523 \cdot 10^{-8}$
$\ln(\lambda) = -18$

$\lambda = 0$
$\ln(\lambda) = -\infty$

Recall:

$$S_{RLS}(\boldsymbol{\beta}) = S_{OLS}(\boldsymbol{\beta}) + \lambda R(\boldsymbol{\beta})$$

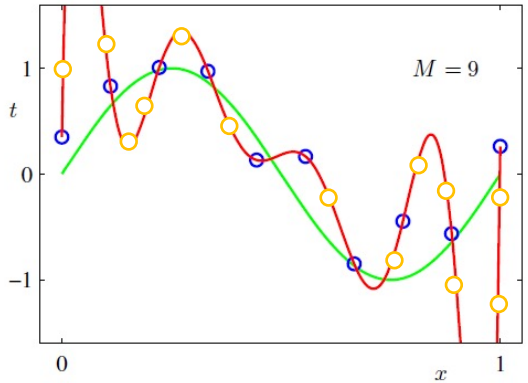
If λ is large, bias is high.
Many predictors are penalized, and we risk "**underfitting**": failing to model the pattern.

If λ is small, bias is low.
Few predictors are penalized, and we risk "**overfitting**": modelling the noise rather than the pattern.

Source:
https://stats.stackexchange.com/questions/108364/demonstration-of-benefits-of-ridge-regression-over-ordinary-regression

Assessing fit requires a reference

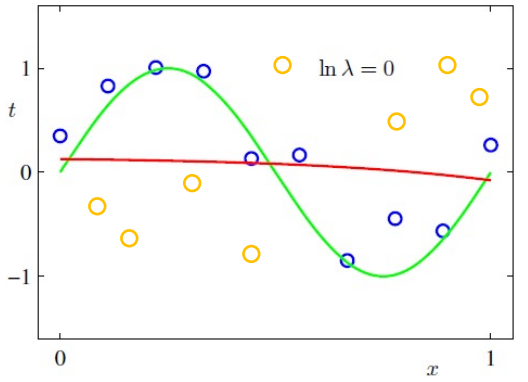
Assume our sample includes most of the population.



Without the green line for reference,
a truly complex model is hard to distinguish from an overfit model.

(Because we have most of the population modelled, a model that precisely predicts each observation might not be overfit!)

Assume our sample is extremely sparse.



Without the green line for reference,
a truly sparse model is hard to distinguish from an underfit model.

(Because we have insufficient data, a model that fails to accurately predict any observations might not be underfit!)

We need reference data to establish whether we are over- or underfit.
We need to assess the "**generalizability**" of the model.

Cross-Validation for Hyperparameter Tuning

How can we choose λ to avoid over- and underfitting?

Recall:

$$\boldsymbol{\lambda} = \{\lambda \in \mathbb{R}_{\geq 0}\}$$

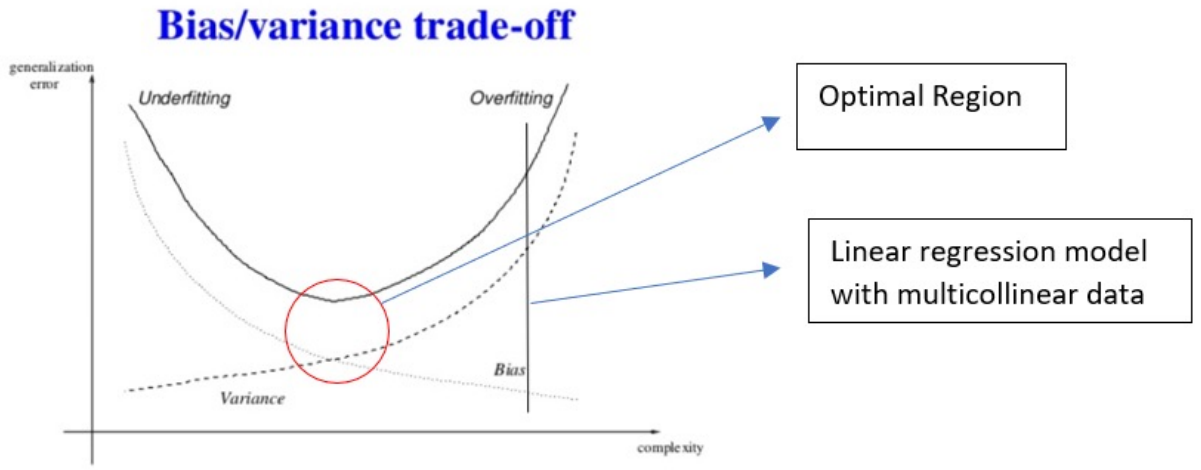
We need to define a loss function for $\boldsymbol{\lambda}$!

$$\hat{\lambda} = \arg\min_{\boldsymbol{\lambda}} S(\boldsymbol{\lambda})$$

How about minimizing the model's error on a new or different dataset?

This "generalization error" (a.k.a. "out-of-sample error") can be estimated with **cross-validation** (CV, a.k.a. "out-of-sample testing").

Solving for $\hat{\lambda}$ is called "**hyperparameter tuning**".



Source: https://www.datasciencecentral.com/profiles/blogs/intuition-behind-bias-variance-trade-off-lasso-and-ridge

Training and Validation

But… how do we generate new or different data?

Resample the sample!

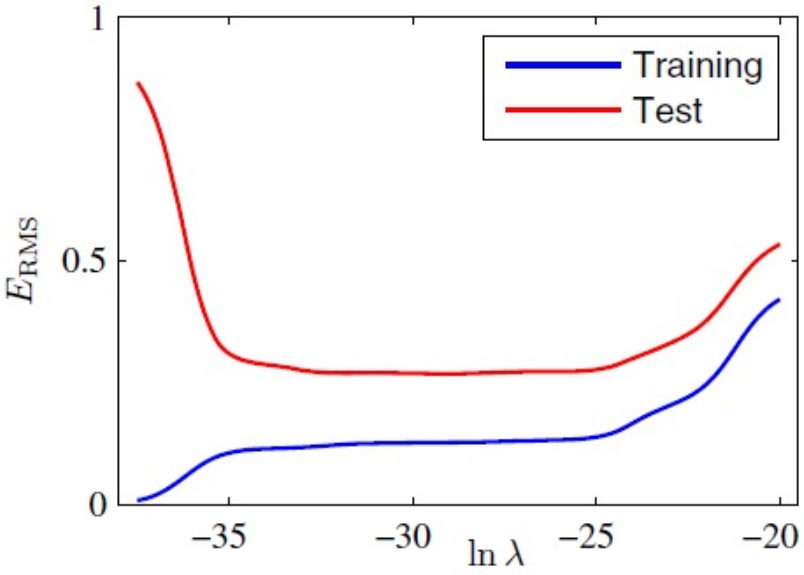
The most basic approach is the "**holdout** method":

1. Divide the sample into separate subsets, e.g.:
   - 70% training, 30% validation
   - 60% training, 40% validation

2. Generate the model using the training data and establish a range of potential values of $\boldsymbol{\lambda}$

3. Find $\hat{\lambda}$ by solving for the λ that minimizes our loss function (e.g. MSE) in the validation data (i.e. hyperparameter tuning)

What if we picked a "bad" training or validation set?



Source: https://stats.stackexchange.com/questions/108364/demonstration-of-benefits-of-ridge-regression-over-ordinary-regression

Source: http://www.ebc.cat/2017/01/31/cross-validation-strategies/

Exhaustive Cross-Validation

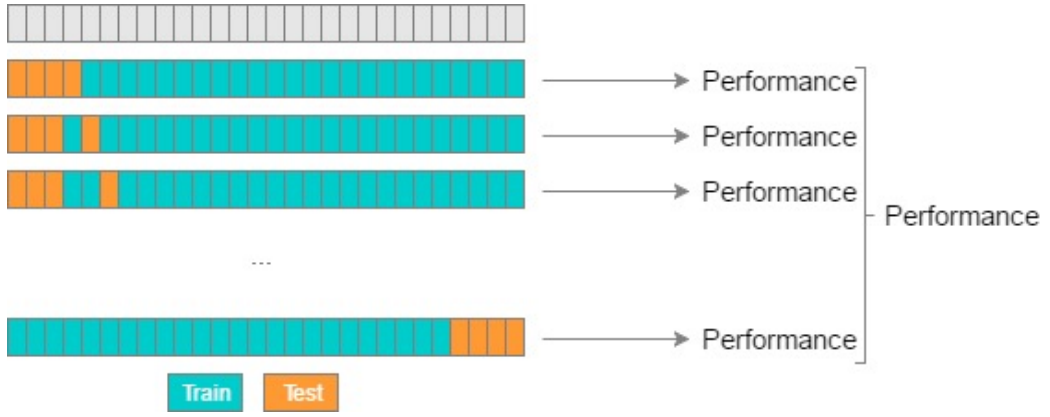
Let's test all possible combinations of subsets as validation sets!

"**Leave-p-out Cross Validation**" (LpO CV)

1. Take p observations from the full dataset of n observations.
2. Generate the model with $n - p$ observations as the training set.
3. Find $\hat{\lambda}$ using the p observations as the validation set

Let this $\hat{\lambda}$ be $\hat{\lambda}_1$, such that:

$$\hat{\boldsymbol{\lambda}} = \{\hat{\lambda}_i\}_{i=1}^{\binom{n}{p}}$$



Source: http://www.ebc.cat/2017/01/31/cross-validation-strategies/

4. Take a different set of p observations
5. Generate a new model with $n - p$ observations as the training set
6. Find $\hat{\lambda}_2$ using the new p observations as the validation set

...

...

7. Repeat until all combinations of p observations have been used for validation
8. Use the average $\hat{\boldsymbol{\lambda}}$ as the $\hat{\lambda}$ for the final model

$$\hat{\lambda}_{final} = \mathrm{E}[\hat{\boldsymbol{\lambda}}]$$

The $\binom{\ }{\ }$ notation means all combinations of p elements of n:

$$\binom{n}{p} = C_p^n = \frac{n!}{p!\,(n-p)!}$$

This is sometimes read as "n choose p".

Exhaustive Cross-Validation

"Leave-p-out Cross Validation" is computationally expensive!

If we did LpO CV with 70% training/30% validation sets and 100 observations…

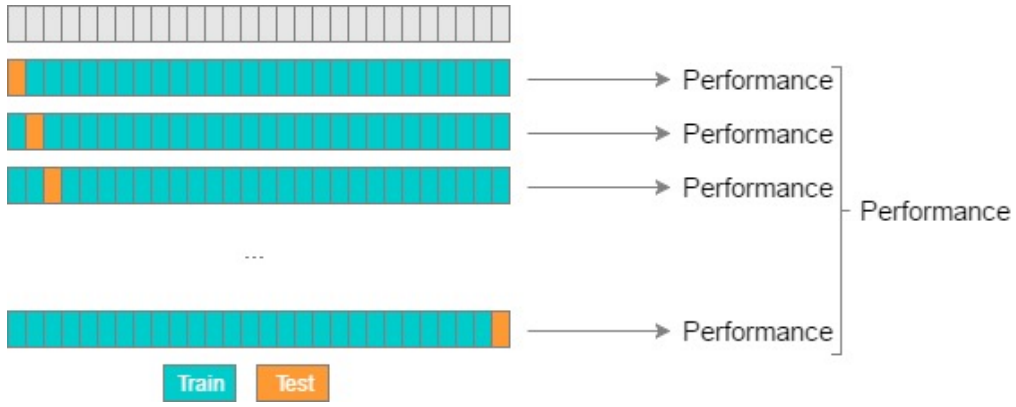
$$\binom{100}{30} = 2.94 \cdot 10^{25}$$

That's larger than the largest SI prefix!!
(The mass of oceans on Earth is 1.4 yottagrams (Yg) – that is, $1.4 \cdot 10^{24}$ g.)

How can we make LpO CV more feasible?

1. Make p smaller
   - "**Leave-one-out Cross Validation**" is when $p = 1$. This is usually feasible. (Does this actually achieve the goal of generalizability?)

2. Don't be exhaustive; approximate LpO CV instead



Source: http://www.ebc.cat/2017/01/31/cross-validation-strategies/

Non-Exhaustive Cross-Validation

"k-**fold Cross-Validation**"

1. Split data into k equal sets
2. Hold one set out and generate the model with remaining data
3. Find $\hat{\lambda}_1$ using the holdout set as the validation set
4. Hold another set out and generate the model with remaining data
5. Find $\hat{\lambda}_2$ using the new holdout set as the validation set

   …
6. Repeat until all k sets have been used for validation

   …
7. Use the average $\hat{\boldsymbol{\lambda}}$ as the $\hat{\lambda}$ for the final model

Source: http://www.ebc.cat/2017/01/31/cross-validation-strategies/

Given n observations, when $k = n$, k-fold cross-validation is equivalent to leave-one-out cross-validation.

Most k-fold CV implementations split data into k equal sets randomly.
As such, **results will vary each time k-fold CV is performed!**

Non-Exhaustive Cross-Validation

"**Repeated random sub-sampling validation**"
a.k.a. "**Monte-Carlo Cross-Validation**"

1. Split data randomly into training and validation sets
2. Generate the model with the training set
3. Find $\hat{\lambda}_1$ using the validation set
4. Split data randomly *again* into training and validation sets
5. Generate the model with the new training set
6. Find $\hat{\lambda}_2$ using the new validation set
   …
7. Perform i iterations of steps (4) to (6), or set a "convergence criteria" for $\hat{\boldsymbol{\lambda}}$ (i.e. a loss function, such as coefficient of variation)
   …
8. Use the average $\hat{\boldsymbol{\lambda}}$ as the $\hat{\lambda}$ for the final model

The proportion of data in each training/validation set is not fixed!
This method can be done *with* or *without replacement*.

Because the training/validation sets are chosen at random,
results will vary with each run of Monte-Carlo CV!

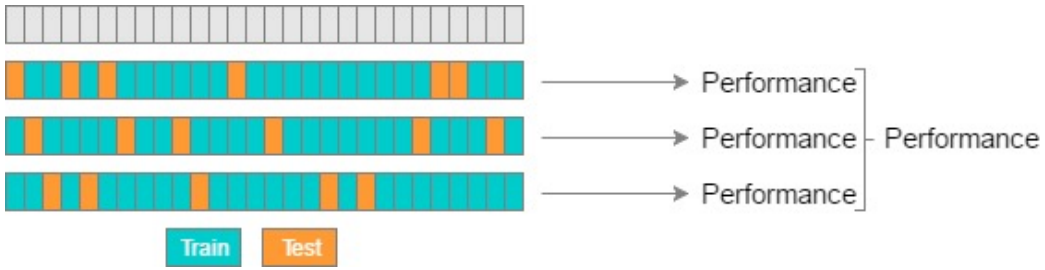


*figure shows repeated random sub-sampling without replacement

Source: http://www.ebc.cat/2017/01/31/cross-validation-strategies/

Take home message:

We must consider the validation approach when assessing the quality of studies that uses machine learning.

To summarize…

λ **balances bias and variance**.

Too much bias or variance can lead to **over- and underfitting**.

Over- and underfitting can only be assessed with a **reference dataset**.

$\boldsymbol{\lambda}$ is a **hyperparameter** that can be “tuned” by **cross-validation**.

Some cross-validation methods are better than others.

Part 4:
Machine Learning, a very, very, quick introduction

From algorithms to machines

What does “machine” in “machine learning” mean?

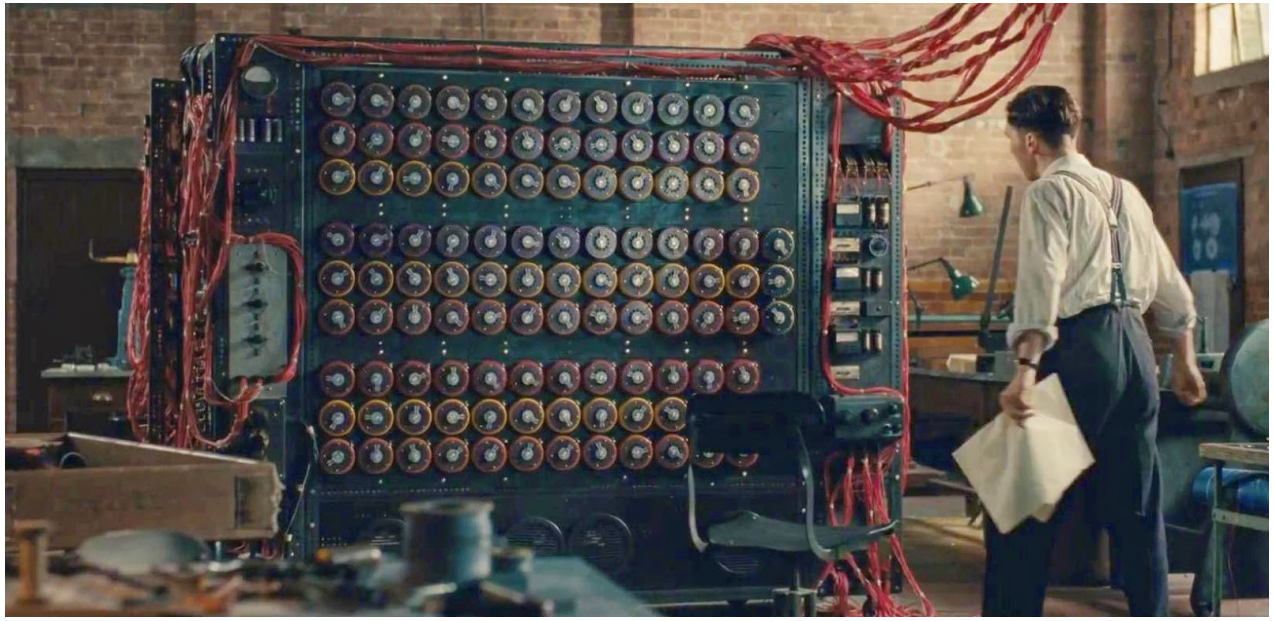
OED defines “machine” as:
“A complex device, consisting of **a number of interrelated parts**, each having a definite function […] to perform a certain kind of work”

Computers are machines!

Regression is a machine!

Regression is “just another algorithm”…

Many algorithms together make a “machine learning algorithm”!



The “bombe” that cracked the Enigma from in World War II was a *very early* computing machine.

Source: *The Imitation Game* (2014)

Regression to Neural Network

Consider a model with 2 predictors:

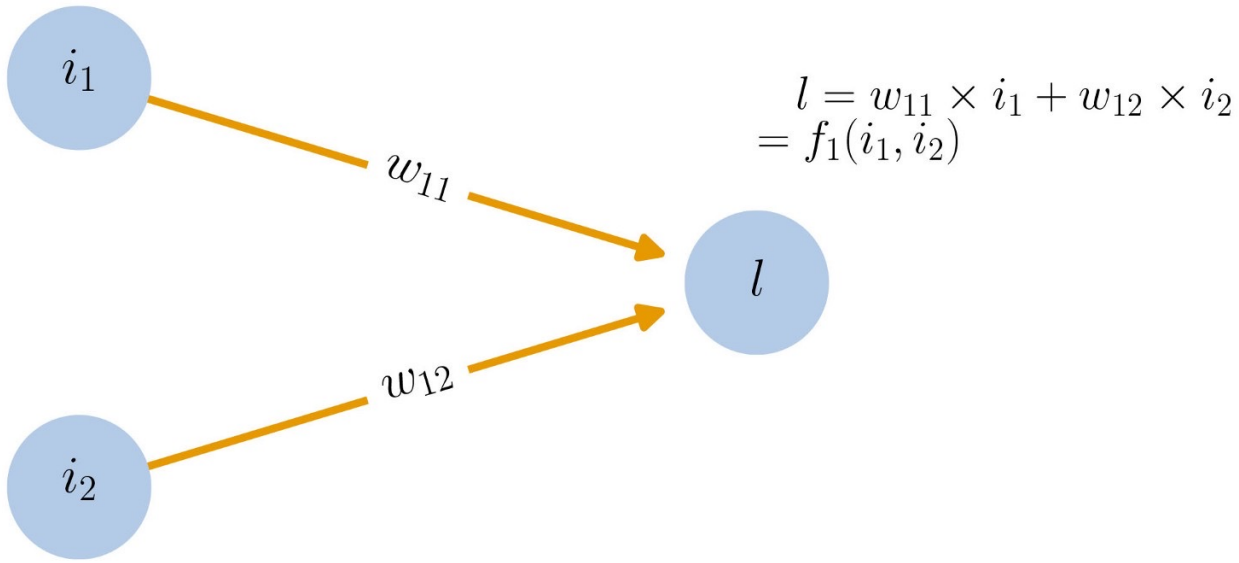
$$\{y_i, x_{i1}, x_{i2}\}_{i=1}^{n}$$

$$\boldsymbol{y} = \beta_0 1 + \beta_1 \boldsymbol{x_1} + \beta_2 \boldsymbol{x_2} + \boldsymbol{\varepsilon}$$

For the sake of simplicity, we exclude the intercept β_0.

Then, given $m = \{1,2\}$…

$$\boldsymbol{x_m} = i_m$$
$$\beta_{1m} = \omega_{1m}$$
$$\boldsymbol{y} = f(i_1, i_2)$$



Source: https://towardsdatascience.com/a-gentle-journey-from-linear-regression-to-neural-networks-68881590760e

A neuron can be a linear regression machine!

Activation Functions

In context of a "neural network", the output of the neuron, l, is modified by an "activation function" a.

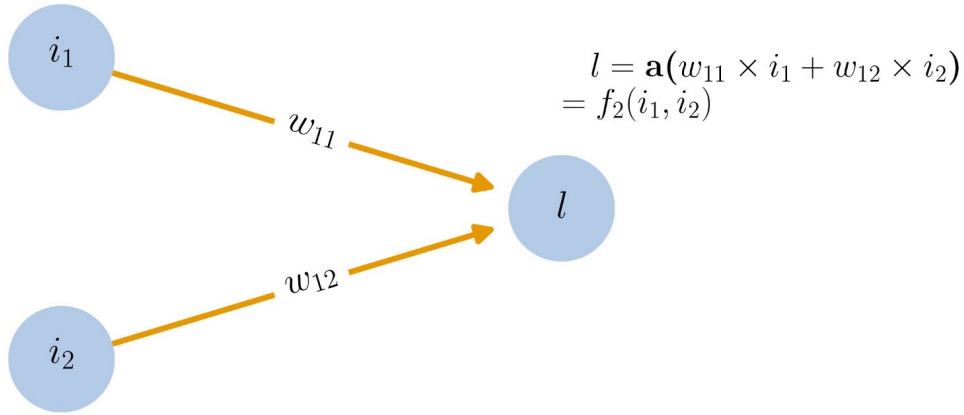
$$\boldsymbol{y} = f(i_1, i_2) \Longrightarrow$$
$$l = a(\boldsymbol{y}) = a\big(f(i_1, i_2)\big)$$

The multivariate linear regression is a special case where:

$$a(\boldsymbol{y}) = \boldsymbol{y}$$

However… the activation function can be any function that changes $\boldsymbol{y}$ in some way!

If you did regressions with different activation functions as outputs, you could potentially use these altered outputs to model non-linear functions.



Source: https://towardsdatascience.com/a-gentle-journey-from-linear-regression-to-neural-networks-68881590760e

A basic neural network

When you stack many regressions together…

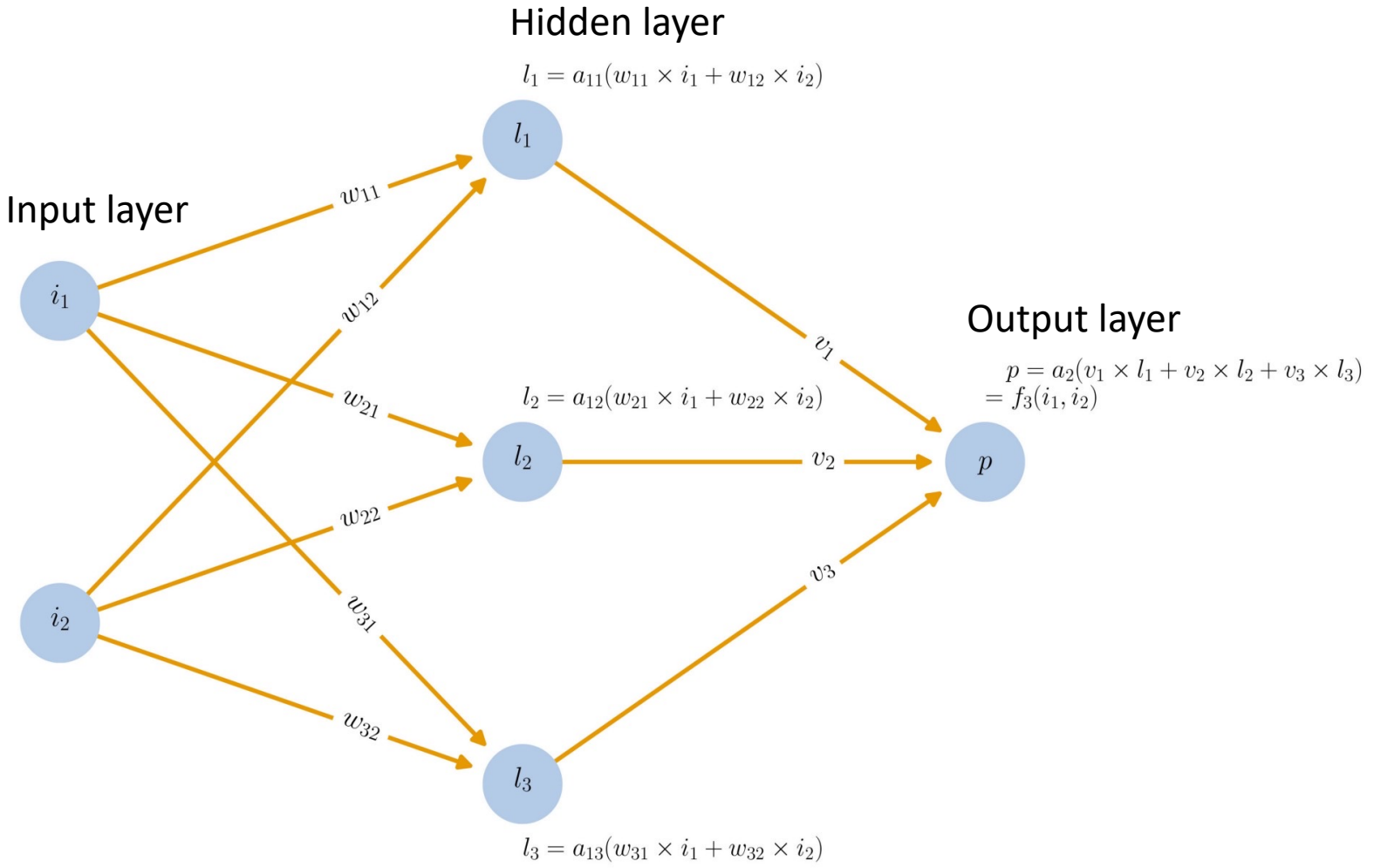
You get a **neural network**!

The regression outputs, l, are a "hidden layer".

Each l in the hidden layer is summed to produce an output p.

"Solving" the neural network requires "**backpropagation**" – using a loss function to minimize the errors across layers.
(This is often described as "**gradient descent**".)

The use of backpropagation to adjust weights is how the neural network "learns".



Types of Learning

Supervised

"You have some set of inputs and outputs that you know.
You want the algorithm to tell you the outputs of a new set of inputs."

Often, these are classification or regression problems.

Unsupervised

"You have some set of inputs that you know.
You want the algorithm to tell you patterns among the inputs."

Often, these are clustering problems.

Reinforcement

"You have some set of inputs.
You perform some action given the inputs, producing outputs.
You assign a 'value' to the outputs (i.e. reward or punishment).
Based on the output value, you change your inputs."

Often, this is used for complex real-world problems (e.g. artificial intelligence).

Thanks for listening ☺

Additional Resources

Ordinary Least Squares:
https://web.stanford.edu/class/stats253/lectures/lect2.pdf

LASSO:
http://www.math.mcgill.ca/yyang/regression/extra/lasso.pdf

Ridge Regression:
https://stats.stackexchange.com/questions/108364/demonstration-of-benefits-of-ridge-regression-over-ordinary-regression

Bias-Variance Trade-off:
https://www.datasciencecentral.com/profiles/blogs/intuition-behind-bias-variance-trade-off-lasso-and-ridge

Cross-Validation:
http://www.ebc.cat/2017/01/31/cross-validation-strategies/

Additional Resources

Machine Learning “cheat sheet”:
https://ml-cheatsheet.readthedocs.io/en/latest/index.html

Intro to neural networks:
https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15381-s06/www/nn.pdf
https://towardsdatascience.com/a-gentle-journey-from-linear-regression-to-neural-networks-68881590760e

A more detailed description of neural network math:
https://imada.sdu.dk/~rolf/Edu/DM534/E16/DM534-marco.pdf