What is Machine Learning?

Oxford Spring School in Advanced Research Methods 2024

Dr Thomas Robinson, LSE

Day 1/5 (2025)

Introduction

This course

5×3 hour sessions, covering:

1. Introduction to ML (maximum likelihood estimation)
2. Regularised methods (bias variance trade-off)
3. Tree-based methods (hyperparameter tuning)
4. Neural networks (feature engineering)
5. Ensemble methods

The logic:

- Understand the underlying mechanics of parameter estimation
- Start from a modeling strategy we are familiar with...
- ... and move on to algorithmically more complicated cases
- Build on the same foundational concepts across each day

What won't we cover?

- ML is a broad and contested domain
- We will not cover some topics:
  - Unsupervised methods
  - Clustering algorithms
  - (Text-specific ML models)

This course prioritises:

- Intuitive understanding of popular ML methods
- Transferability of fundamentals
- Relevance to "downstream" social science research problems

Session structure

Each day will be a mixture of:

- Lecture content (approx. 2 hours)
  - Plenty of opportunities for questions
  - Time for us to think through some applied problems
- Coding walkthroughs (approx. 1 hour)
  - Conducted in R using RStudio
  - Hands on experience using different algorithms
- Self-guided learning
  - Applied readings using the methods we discuss in class
  - Completing and consolidating coding exercises

Slides and code are available here:
www.github.com/tsrobinson/ml_oxss25

Today's session

Goals:

1. Introduce the topic of machine learning
   - What is ML?
   - How do we distinguish ML from "traditional" statistics?
   - What sort of problems might we apply it to?
2. Introduce key conceptual distinctions we will make throughout the course
3. Introduce maximum likelihood estimation
   - A key way in which ML parameters are estimated
   - Build our own logistic regression estimator

What is machine learning?

(Machine) learning and statistics

"There are two cultures in the use of statistical modeling to reach ***conclusions from data****. One assumes that the data are generated by a given stochastic data model. The other uses algorithmic models and treats the data mechanism as unknown" – Breiman 2001*

"Statistical learning refers to a set of tools for modeling and understanding ***complex*** *datasets. It is a recently developed area in statistics and blends with parallel developments in* ***computer science*** *and, in particular, machine learning" – James et al. 2013*

"Machine learning is a subfield of ***computer science*** *that is concerned with building algorithms which, to be useful, rely on a collection of examples of some phenomenon. . . " – Burkov 2019*

Machine learning

Expectation: I need a $1m super computer

Reality: It runs in ~~*minutes*~~ seconds on a ~~*personal computer*~~ smartphone

Why machine learning?

Machine learning can be:

- Powerful
  - With respect to computational efficiency
- Flexible
  - With respect to **data generating processes**
- Reduce the burden on the researcher
  - With respect to generating data...
  - and estimating models

But ML is not a panacea!

- Cannot solve problems of poor research design
- Increased flexibility *can* lead to poor interpretability

Machine learning and social science

ML also introduces issues of its own:

Twitter apologises for 'racist' image-cropping algorithm

Users highlight examples of feature automatically focusing on white faces over black ones

Sarah Silverman sues OpenAI and Meta claiming AI training infringed copyright

Prediction and inference

The classic dichotomy when introducing ML:

$$\hat{y_i} = \hat{\beta}_0 + \hat{\beta}_1 x_{1i}$$

- **Inference**: estimating the size/direction of the (causal) relationship between variables ($\hat{\boldsymbol{\beta}}$ problems)
- **Prediction**: estimating the outcome, using the relationships between variables ($\hat{\boldsymbol{y}}$ problems)

These two facets are connected:

- Knowing the size/direction of (all) relationships -> predict the outcome
- But we rarely know the true model
- Sometimes we can get good at $\hat{\boldsymbol{y}}$ problems without knowing $\hat{\boldsymbol{\beta}}$

Going down a philosophical rabbit hole

We might want to ask:

- Is there such a thing as a “true data generating process?” (see Grimmer et al 2021)
- Could a model yield perfect predictions without learning causal relationships?

More specifically for this course:

- Is the “$=$” in $y = \beta x + \ldots$ a convenience more than a scientific claim?
- What is the purpose of a model?

$\hat{y}$ and $\hat{X}$ problems

We can think about where a prediction problem lies:

- $\hat{\boldsymbol{y}}$ (dependent variable)
  - To predict the probability of revolution...
  - ... or the weather tomorrow
  - These are not necessarily *inferential* problems
- $\hat{\boldsymbol{X}}$ (independent variables):
  - Dimensions of interest that may be important to our theory, but which are:
    - Latent (i.e. not directly observable)
    - Difficult to measure "by hand"
  - Use ML to make predictions over $\boldsymbol{X}$
  - These estimates are used *downstream* to test a theory

Classification and prediction

Within both problems, we can distinguish two types:

- **Prediction** – estimating the value of a continuous variable (sometimes referred to as a "regression" problem) e.g.,
  - The ideology of a politician
  - The number of votes received by a candidate
- **Classification** – estimating which *class* of a category an observation belongs to, e.g.,
  - Party identity (Republican/Democrat/Libertarian/Independent)
  - The topic of a tweet (foreign/domestic/personal, positive/negative)
  - Recidivism

Supervised vs unsupervised methods

Supervised methods

- **Training data** are a set of labelled examples, where $\boldsymbol{y}$ is our target prediction
- We use these examples to "learn" the relationship between $\boldsymbol{y}$ and $\boldsymbol{X}$
- Then predict $\boldsymbol{y}$ for a *new* unlabeled dataset (i.e. where the target variable is not observed)

Learning the relationship:

$$\underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}}_{\mathbf{y}^{\mathsf{TRAIN}}} \quad \underbrace{\begin{bmatrix} 3.3 & 1.1 & 0 \\ 2.7 & 0.8 & 0 \\ 1.8 & 0.1 & 1 \\ \vdots & \vdots & \vdots \\ 5 & 1.2 & 0 \end{bmatrix}}_{\mathbf{X}^{\mathsf{TRAIN}}}$$

Predicting on new data

$$\mathbf{X}^{\mathsf{TEST}} = \begin{bmatrix} 3.5 & 1.9 & 1 \\ 5.4 & 0.3 & 0 \\ 1.7 & 0.5 & 1 \end{bmatrix}$$

Notation

Algebra

Throughout the course, we will follow the notation set out in Burkov (2019):

- θ and x are scalar - i.e. a single number
  - E.g., $\theta = 3.141$, $x = 1$ etc.
- $\boldsymbol{\theta}$ and $\boldsymbol{x}$ are vectors - i.e. an ordered list of scalar values
  - E.g., $\boldsymbol{\theta} = [0.5, 3, 2]$
- $\mathbf{\Theta}$ and $\mathbf{X}$ are matrices
  - E.g.,

$$\mathbf{X} = \begin{bmatrix} 1 & 5 \\ 24 & -3 \end{bmatrix}$$

  - $\mathbf{x}^{(k)}$ is the kth column of $\boldsymbol{X}$
  - $\mathbf{x_i}$ is the ith row in matrix $\mathbf{X}$
  - $x_i^{(k)}$ is the kth element of the row vector $\boldsymbol{x_i}$

Probability notation

Let p denote a probability distribution *function* that returns the probability of an event or observation:

- $p(A) = 0.5$ means the probability of event A is 0.5
- All probabilities are bounded between 0 and 1
  - $0 \leq p(A) \leq 1$

Conditional probabilities means the probability of an event **given** the value of another variable

- $p(A|c) = 0.25$

Probability rules

Probabilities have some nice features:

- $p(A \text{ and } B) = p(A)p(B|A)$
- If $p(A) \perp p(B)$, then $p(B|A) = p(B)$

As a result:

- If $p(A) \perp p(B)$, then $p(A \text{ and } B) = p(A)p(B)$
- These rules explain why the probability of two coin-flips is $=$ 0.25:
  - $P(\text{Flip 1} = \text{Heads}) = 0.5$
  - $P(\text{Flip 2} = \text{Heads}|\text{Flip 1} = \text{Heads}) = P(\text{Flip 2} = \text{Heads})$
  - $P(\text{Flip 1} = \text{Heads *and* Flip 2} = \text{Heads}) = 0.5 \times 0.5 = 0.25$

Notation quiz

What are the following:

1. $\mathbf{a}$
2. y_i
3. $\boldsymbol{\beta}$
4. β
5. $\mathbf{\Theta}$

If $p(A) = 0.5$, $p(B) = 0.1$, and $p(B|A) = 0.3$:

6. Is $p(A) \perp p(B)$?
7. What is $p(A \text{ and } B)$?

Maximum Likelihood Estimation (a gentle introduction)

Searching the hypothesis space

A researcher wants to characterise an outcome as the linear combination of predictor variables:

$$\mathbf{y} = \beta_0 + \beta_1 \mathbf{x}^{(1)} + \ldots + \beta_2 \mathbf{x}^{(k)}$$

- We will set aside all inferential/theoretical concerns
- Focused on parsimonious description in a linear space

Since $\mathbf{X}$ is fixed (it's the data we observe), we need to find the **best** β:

- Need some way of searching across all possible values ("**hypotheses**") and finding the one that best *fits* our data
  - Statistical learning!

Bayes Theorem (from a frequentist perspective)

$$\underbrace{P(A|B)}_{\text{Posterior}} = \frac{\overbrace{P(B|A)}^{\text{Likelihood}} \times \overbrace{P(A)}^{\text{Prior}}}{\underbrace{P(B)}_{\text{Evidence}}}$$

We can use Bayes formula to estimate the posterior probability of some parameter $\boldsymbol{\theta}$:

$$p(\boldsymbol{\theta}|\mathbf{X}) \propto p(\mathbf{X}|\boldsymbol{\theta}) \times p(\boldsymbol{\theta}),$$

where $\mathbf{X}$ is the data.

Likelihood function

Let's suppose that we have no prior knowledge over $\boldsymbol{\theta}$, so we'll drop the prior and focus specifically on the likelihood:

$$\mathcal{L}(\boldsymbol{\theta}) = p(\mathbf{X}|\boldsymbol{\theta})$$

How would we calculate this?

Likelihood function

Let's suppose that we have no prior knowledge over $\boldsymbol{\theta}$, so we'll drop the prior and focus specifically on the likelihood:

$$\mathcal{L}(\boldsymbol{\theta}) = p(\mathbf{X}|\boldsymbol{\theta})$$

How would we calculate this?

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= p(\mathbf{x_1}|\boldsymbol{\theta}) \times p(\mathbf{x_2}|\boldsymbol{\theta}) \times \ldots \times p(\mathbf{x_n}|\boldsymbol{\theta}) \\ &= \prod_{i=1 \ldots N} p(\mathbf{x}_i|\boldsymbol{\theta})\end{aligned}$$

i.e. the product of the probability of each observations within $\mathbf{X}$, given $\boldsymbol{\theta}$.

Comparing likelihoods

Suppose we have two alternative values of $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}$. We can calculate the likelihood *ratio* (LR) of these two possible parameter values:

$$LR = \frac{\mathcal{L}(\boldsymbol{\theta}^{(1)})}{\mathcal{L}(\boldsymbol{\theta}^{(2)})}$$

If LR $>$ 1, which set of parameters would we pick?

Comparing likelihoods

Suppose we have two alternative values of $\boldsymbol{\theta}$: $\boldsymbol{\theta^{(1)}}, \boldsymbol{\theta^{(2)}}$. We can calculate the likelihood *ratio* (LR) of these two possible parameter values:

$$LR = \frac{\mathcal{L}(\boldsymbol{\theta^{(1)}})}{\mathcal{L}(\boldsymbol{\theta^{(2)}})}$$

If LR $>$ 1, which set of parameters would we pick?

- $\boldsymbol{\theta}^{(1)}$

Maximum likelihood estimation

We can generalise this for all possible values of θ:

$$\arg\max_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1 \ldots N} p(\mathbf{x}_i | \boldsymbol{\theta})$$

i.e., from the set of all possible parameter values Θ, find the parameter value that maximises the likelihood function.

Hence, **maximum** likelihood estimation.

- How we calculate $p(\mathbf{x}_i | \boldsymbol{\theta})$ will depend on the functional form of the underlying distribution
- We'll explore this specifically with respect to logistic regression later on today

Why is this maximum likelihood a useful concept?

Numeric overflow

Multiplying many small numbers means we soon lose the power to calculate them precisely

- `R` double-precision numbers range from 2×10^{-308} to 2×10^{308}
- If 400 observations have $p_{\boldsymbol{\theta}} = 0.01$, $\mathcal{L}(\boldsymbol{\theta})$ will be outside the computable range

What if we take the log?

- The log function is strictly increasing
- $Log(a \times b) = Log(a) + Log(b)$ so we can simply add the values

With the logged likelihood function we do not have the problem of numeric overflow!

Negative log-likelihood

We can also calculate the *negative* log-likelihood:

- I.e. put a minus sign in front!
- We then *minimise* the negative log-likelihood
- We typically want to minimise rather than maximise because many of our procedures for optimisation are based on the former
- But, broadly, this is just semantics:
  - Minimising the negative log-likelihood is the same as maximising the log-likelihood

Logistic regression

Logistic regression:

- Allows us to estimate β parameters when we have a binary outcome variable
- More broadly, it is a **binary classification** algorithm – what is the probability that $y_i = 1$ given a vector of features $\mathbf{x_i}$?

We can write the logistic regression function as,

$$f_{\boldsymbol{\theta},b}(\mathbf{X}) = \frac{1}{1 + e^{-(\boldsymbol{\theta}\mathbf{X}+b)}}.$$

The goal is to find the *best* values of $\boldsymbol{\theta}$ and b that "explains" the data

- For simplicity, let's include b within $\boldsymbol{\theta}$ s.t. $\boldsymbol{\theta} = \{b, \theta_1, \cdots, \theta_k\}$

MLE of logistic regression I

For a given vector of scalar values $\boldsymbol{\theta}$, we can ask what the likelihood of the data is given those values

- With the optimal parameter choice $\boldsymbol{\theta}^*$:
  - When $y_i = 1$, $f_{\boldsymbol{\theta}^*}(\mathbf{x}_i) = 1$
  - When $y_i = 0$, $f_{\boldsymbol{\theta}^*}(\mathbf{x}_i) = 0$

Why can't we just use the predicted probabilities as the likelihood?

MLE of logistic regression I

For a given vector of scalar values $\boldsymbol{\theta}$, we can ask what the likelihood of the data is given those values

- With the optimal parameter choice $\boldsymbol{\theta}^*$:
  - When $y_i = 1$, $f_{\boldsymbol{\theta}^*}(\mathbf{x}_i) = 1$
  - When $y_i = 0$, $f_{\boldsymbol{\theta}^*}(\mathbf{x}_i) = 0$

Why can't we just use the predicted probabilities as the likelihood?

- Predicted probabilities work well when $y_i = 1$
- But we would erroneously down-weight the likelihood for all $y_i = 0$

How can we fix this?

- When $y_i = 0$, let the likelihood equal $1 - f_{\boldsymbol{\theta}*}(\mathbf{x}_i)$

MLE of logistic regression II

We construct the likelihood for *any* datapoint using a mathematical "logic gate":

$$\mathcal{L}_{\boldsymbol{\theta}} = f_{\boldsymbol{\theta}}(\mathbf{X})^{\boldsymbol{y}} \times (1 - f_{\boldsymbol{\theta}}(\mathbf{X}))^{(1-\boldsymbol{y})},$$

as when $y_i = 0, x^{y_i} = 1$ and $x^{(1-y_i)} = x$, and *vice versa*.

Simplifying, since $f_{\boldsymbol{\theta}}(\boldsymbol{x_i}) = \hat{y}_i$:

$$\mathcal{L}_{\boldsymbol{\theta}} = \hat{\boldsymbol{y}}^{y}(1 - \hat{\boldsymbol{y}})^{1-y}$$

MLE optimization

We can then apply our "tricks" to make the computation easier:

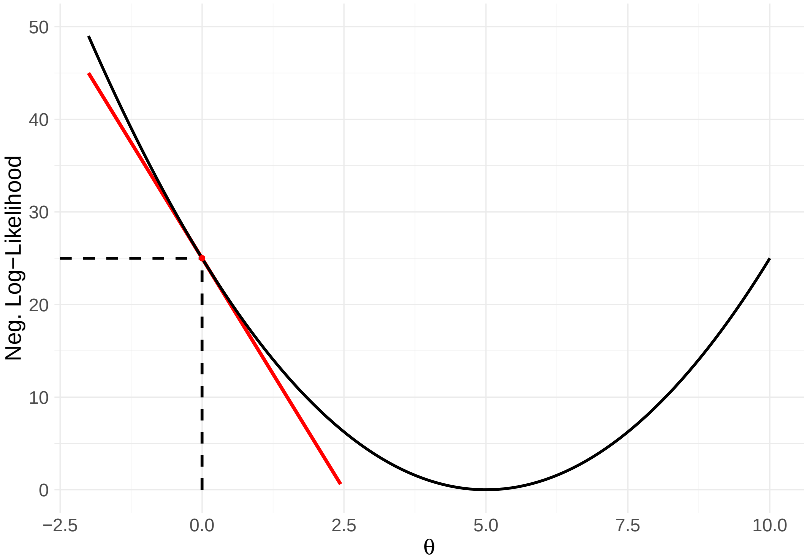
$$-Log(\mathcal{L}_{\boldsymbol{\theta}}) = -\sum_{i=1}^{N} Log(\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{x_i})),$$

with the goal of minimising this quantity through choosing $\boldsymbol{\theta}$.

How exactly do we minimize this function?

- Unlike OLS it is not possible to minimize this function analytically
- We therefore have to use computation to iterate through values of $\boldsymbol{\theta}$ to *approximate* the minima

Minimising the negative log-likelihood in one dimension



Gradient descent algorithm

To find the minimum of the negative log-likelihood we:

1. Choose a value for the starting parameter θ
2. Calculate the slope of the function at that point
3. Adjust our value of θ in the *opposite* direction to the slope coefficient's sign
4. Recalculate the slope, and repeat 2-4

We can generalise this to $\boldsymbol{\theta}$:

- Let $Q(\boldsymbol{\theta})$ be the negative log likelihood function
- Calculate the **gradient vector** of the function in k-dimensions
- Adjust each parameter $\theta_k \in \boldsymbol{\theta}$ by the negative of the corresponding gradient element

$$\theta_k = \theta_k - \frac{\partial Q(\boldsymbol{\theta})}{\partial \theta_k}$$

Logistic regression gradient

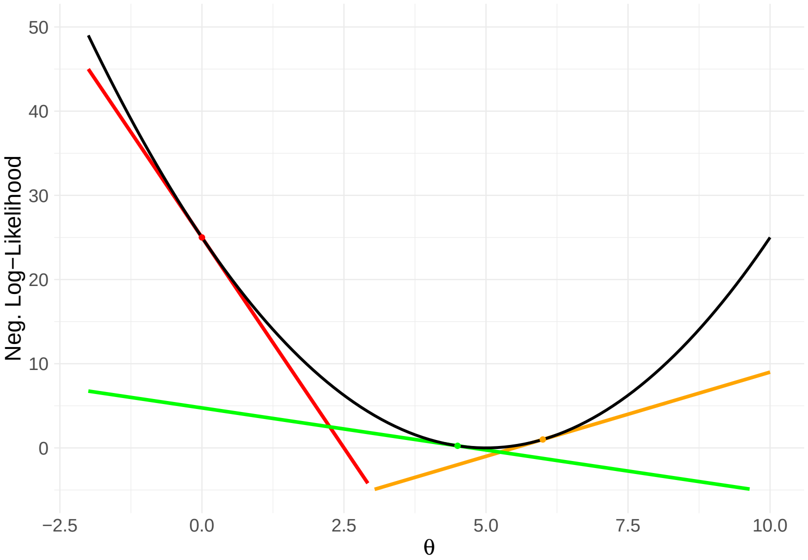
The partial derivative for any predictor $\boldsymbol{x}^{(j)}$ for the *logistic* cost function is:

$$\frac{\partial Q^{\text{Logit}}}{\partial \theta_j} = (f_{\theta_j}(\boldsymbol{X}) - \boldsymbol{y})\boldsymbol{x}^{(j)}$$

Hence the gradient of the function's curve for any vector of logistic parameters $\boldsymbol{\theta}$ can be described as:

$$\boldsymbol{\nabla} = \begin{bmatrix} \frac{\partial Q^{\text{Logit}}(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial Q^{\text{Logit}}(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial Q^{\text{Logit}}(\boldsymbol{\theta})}{\partial \theta_k} \end{bmatrix}$$

Progression of the descent algorithm



Learning rate

As we iteratively adjust the value of our parameter:

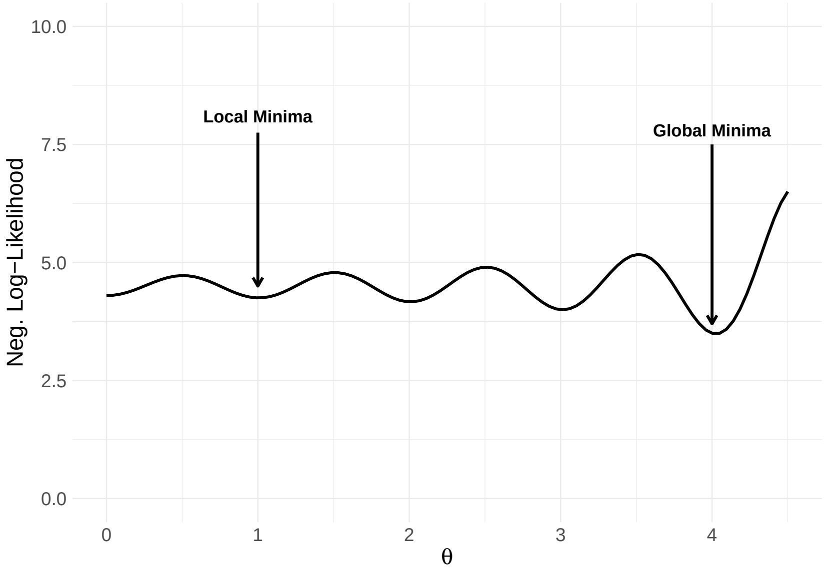
- It's possible we keep jumping over the minima
- Or we get stuck in a rut and the estimator fails to find an even better parameter choice

So we can scale the impact of the current gradient on the new parameter choice:

- Let's call this scalar the **learning rate** (λ)
- $\boldsymbol{\theta}_{\text{New}} = \boldsymbol{\theta} - \lambda \boldsymbol{\nabla}$

The choice of λ is down to the researcher:

- Overly-large values will not converge
- Overly small values may take too long, or risk converging on *local* minima



Local Minima
Global Minima
Neg. Log−Likelihood
θ
10.0
7.5
5.0
2.5
0.0
0
1
2
3
4

Stochastic gradient descent

Gradient descent can be **expensive**:

- We have to evaluate all rows in our training data before making any updates to the parameters
- If we have lots of observations
  1. Each calculation takes a long time
  2. Takes many iterations to optimise
- Instead we can use **stochastic gradient descent** (SGD)
  - Inspect the loss of each observation (or a random subset) individually
  - Update the coefficients based on each observation

Stochastic gradient descent

Under GD, for each iteration:

$$\theta_k \leftarrow \theta_k - \lambda \sum_{i=1}^{N} \Big(f_{\theta_k}(\boldsymbol{x_i}) - y_i \Big) \boldsymbol{x_i}^{(k)}$$

Under SGD, for each iteration:

$$\theta_k \leftarrow \theta_k - \lambda \Big(f_{\theta_k}(\boldsymbol{x_i}) - y_i \Big) \boldsymbol{x_i}^{(k)}, \text{ for } i \in \{1, \ldots, N\}$$

- SGD typically converges a lot faster than GD
  - Every iteration we make N small changes to the parameter estimate
  - Computationally more efficient (we'll cover this more later in the week)
  - At the cost of some additional noise in the optimisation process

Readings

Three suggested readings after today's class

- See course outline for more details
- Happy to answer questions on these tomorrow!

Coding workshop: writing our own logistic regression classifier