Computing for Economists

Machine Learning Methods

Toren Fronsdal and Ruby Zhang

Harvard University

Table of Contents

Introduction

Introduction

Goal: a non-exhaustive helicopter tour of machine learning methods relevant to economics research.

Introduction

(Supervised) machine learning methods address the problem of **prediction**.

- Produce predictions of y from x

Economics often focuses on **parameter estimation** or **causality**.

- Produce good estimates of parameters β that underlie the relationship between y and x

Applying machine learning to economics requires either:

- Applying to $\hat{y}$-relevant tasks (prediction), or
- Methods to use ML for causal problems

Prediction in Economics

- Predict mortality and late-in-life health care spending (Einav et al., 2018)
- Predict loan repayment using mobile phone data (Bjorkegren and Grissen, 2017)
- Predict poverty using satellite images (Jean et al., 2016)
- Predict restaurant health code violations (Glaeser et al., 2016)
- Predict recidivism (Kleinberg et al., 2018)
- Predict probability of protest (Andırın et al., 2023)

Types of Machine Learning

Supervised

- Regression / prediction: $\mathbb{E}[Y|X = x]$
- Classification: $\mathbb{P}[Y = k|X = x], k \in \{1, 2, ..., K\}$

Unsupervised

- Clustering
- Exploratory data analysis
- Dimensionality reduction

Supervised Learning

Setup

- X_i: predictors (features / inputs / covariates)
- y_i: outcome (target / response)
- $L(\hat{y}, y)$: loss function

A machine learning model takes a set of data and a loss function and searches over a set of functions

$$y = \hat{f}(X) + \varepsilon$$

to find the lowest expected prediction loss $\mathbb{E}_{(y,x)}[L(\hat{f}(x), y)]$ on *new* data ("out of sample").

Decomposing Model Errors

Suppose the loss function is mean-squared error:

$$L(\hat{y}, y) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2.$$

If the true model is $y = f(x) + \varepsilon$ (with $f(x) = \mathbb{E}[y|X = x]$), where $\varepsilon \sim N(0, \sigma^2)$, then we can decompose the error for a given x as

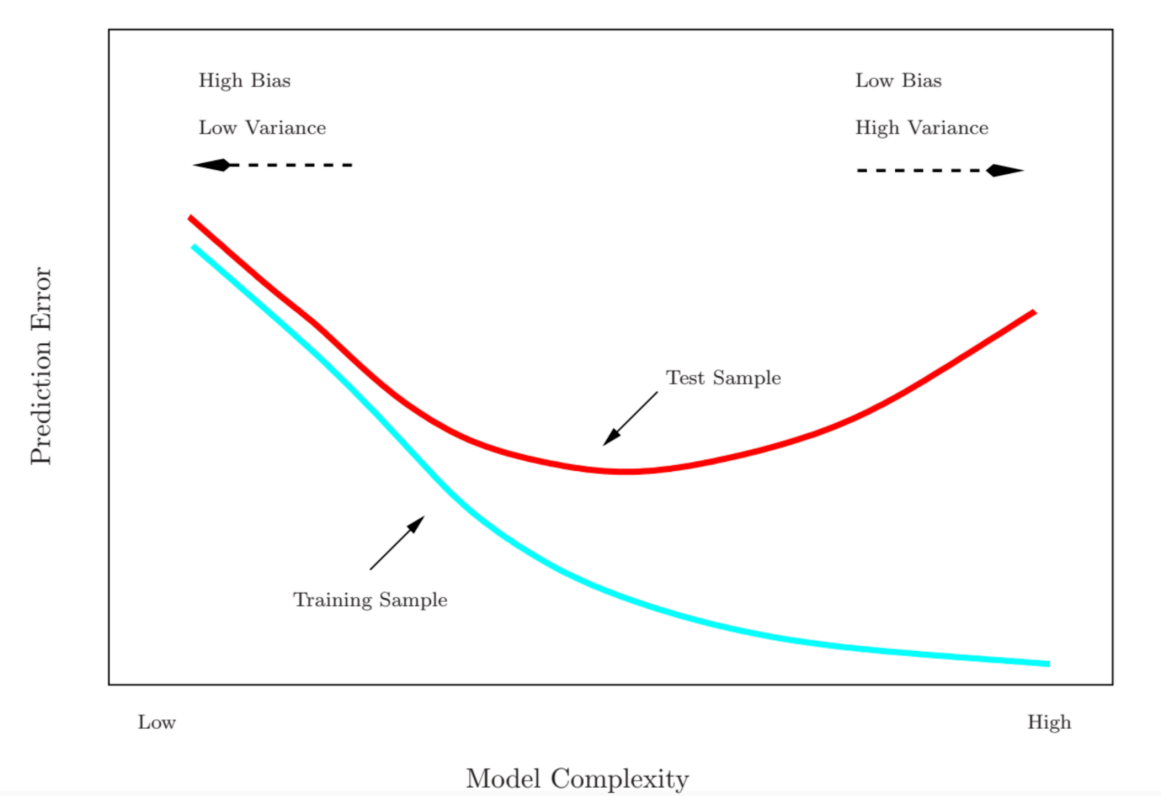
$$\mathbb{E}\left[(y - \hat{f}(X))^2 \mid X = x\right] = \underbrace{\mathbb{E}\left[[f(x) - \hat{f}(x)]^2\right]}_{\text{Reducible}} + \underbrace{\text{Var}(\varepsilon)}_{\text{Irreducible}}$$

Bias-Variance Tradeoff

If the true model is $y = f(x) + \varepsilon$ (with $f(x) = \mathbb{E}[y|X = x]$), where $\varepsilon \sim N(0, \sigma^2)$, then

$$\begin{aligned}\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] &= \mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)^2\right] + \sigma^2 \\ &= \underbrace{\left(f(x) - \mathbb{E}[\hat{f}(x)]\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}\left[\left(\mathbb{E}[\hat{f}(x)] - \hat{f}(x)\right)^2\right]}_{\text{Variance}} + \sigma^2\end{aligned}$$

Train vs. Test Performance



Source: Hastie, Tibshirani, and Friedman (2009)

Model Selection and Regularization

Consider the simple case of linear regression: $\hat{Y} = X^T\hat{\beta}$ for a vector of inputs $X^T = (X_1, X_2, \ldots, X_p)$.

With a large p, we risk **overfitting** to the training data and having poor out-of-sample performance (low bias but high variance).

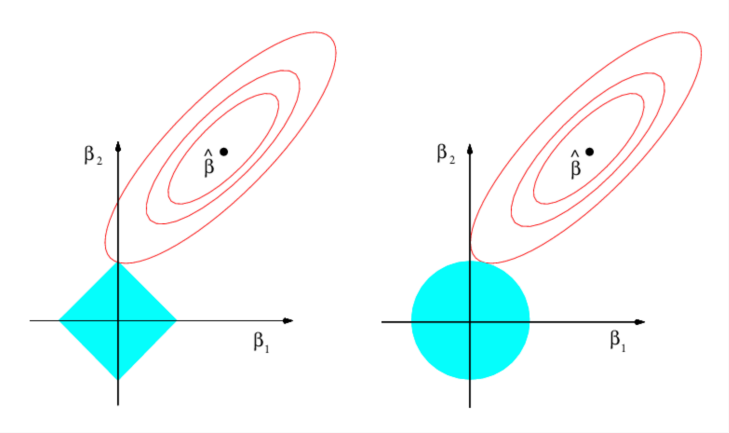
There are two main approaches to choosing the model complexity to achieve the optimal bias-variance tradeoff:

1. **Model selection**. For example, identify a subset of the p predictors most related to the response (subset and stepwise selection).
2. **Shrinkage methods**. For example, the model may include all p predictors, but the estimated coefficients are shrunken towards zero.

Regularization

Reduce variance by constraining or *regularizing* the parameter estimates (at the expense of bias).

- ℓ_1 (Lasso): $\sum_{i=1}^{n} L(\hat{y}_i, y_i) + \lambda \sum_{j=1}^{p} |\beta_j|$
  - Can shrink feature weights to zero
- ℓ_2 (Ridge): $\sum_{i=1}^{n} L(\hat{y}_i, y_i) + \lambda \sum_{j=1}^{p} \beta_j^2$
  - Encourages small but non-zero weights
- Elastic Net: a combination of L1 and L2
  - Provides a balance between feature selection (sparsity) and feature weighting



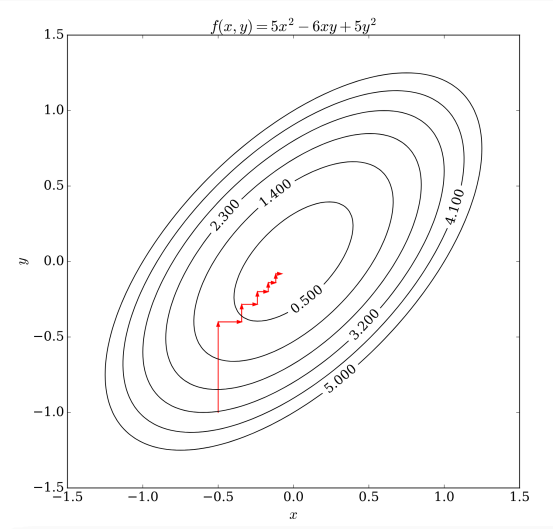
Source: Hastie, Tibshirani, and Friedman (2009)

λ is a "tuning" parameter that is typically chosen by out-of-sample cross-validation.

Coordinate Descent for Lasso

$$\min_{\beta} \sum_{i=1}^{N} \left(y_i - \beta_0 - x_i^T \beta\right)^2 + \lambda \|\beta\|_1$$

Approach: Convert multivariate optimization problem to a loop of univariate optimization problems. Optimize each parameter separately, holding all the others fixed. Cycle through parameters repeatedly until convergence.

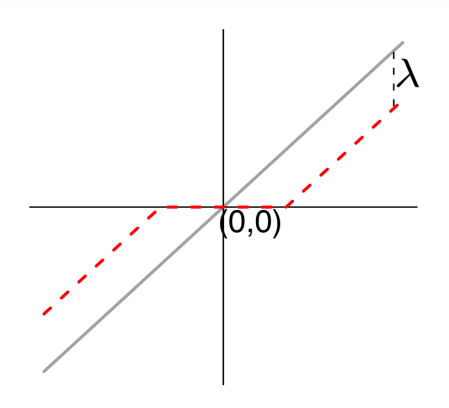


Source: Wikipedia

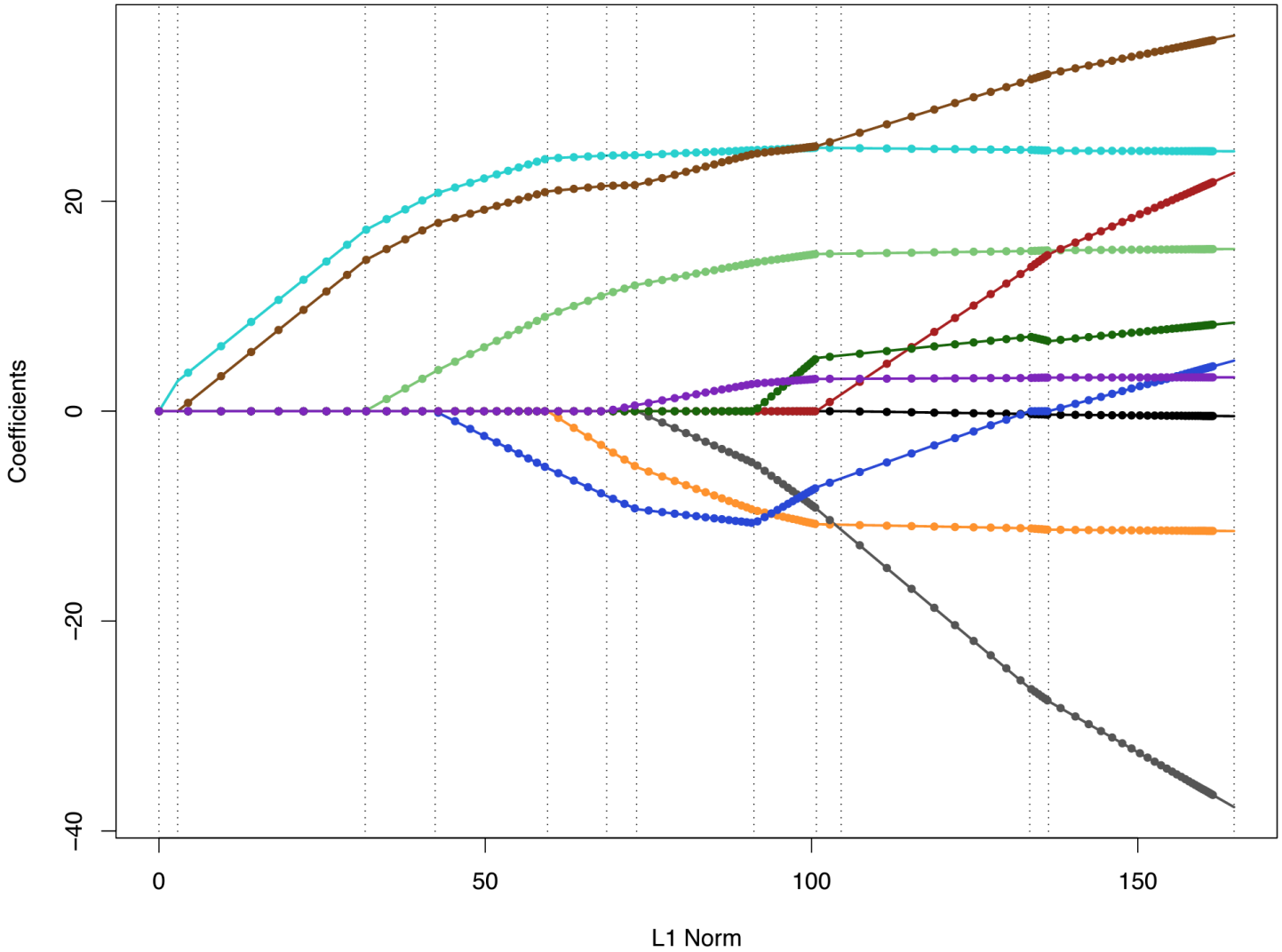
Coordinate Descent for Lasso

- Standardize the p predictors and response variables. Initialize $\hat{\beta} = 0$.
- Repeat until convergence:
  - For $j = 1, 2, ..., p$:
    - Compute the partial residuals $r_{ij} = y_i - \sum_{k \neq j} x_{ik} \beta_k$
    - Compute the simple least squares coefficient of these residuals on jth predictor: $\beta_j^* = \frac{1}{N} \sum_{i=1}^{N} x_{ij} r_{ij}$
    - Update $\hat{\beta}_j$ by soft-thresholding:

$$\hat{\beta}_j \leftarrow S\left(\beta_j^*, \lambda\right) \equiv \textbf{sign}\left(\beta_j^*\right)\left(\left|\beta_j^*\right| - \lambda\right)_+$$



Regularization Path



Pathwise Coordinate Optimization

Compute the solutions for a decreasing sequence of values for λ, starting with λ_{max} (smallest λ for which $\hat{\beta} = \mathbf{0}$).

- Exploits "warm start" $\rightarrow$ solutions don't change much from one λ to the next
- Can be faster to compute path down to λ than the solution only at λ for small values

K-Fold Cross Validation

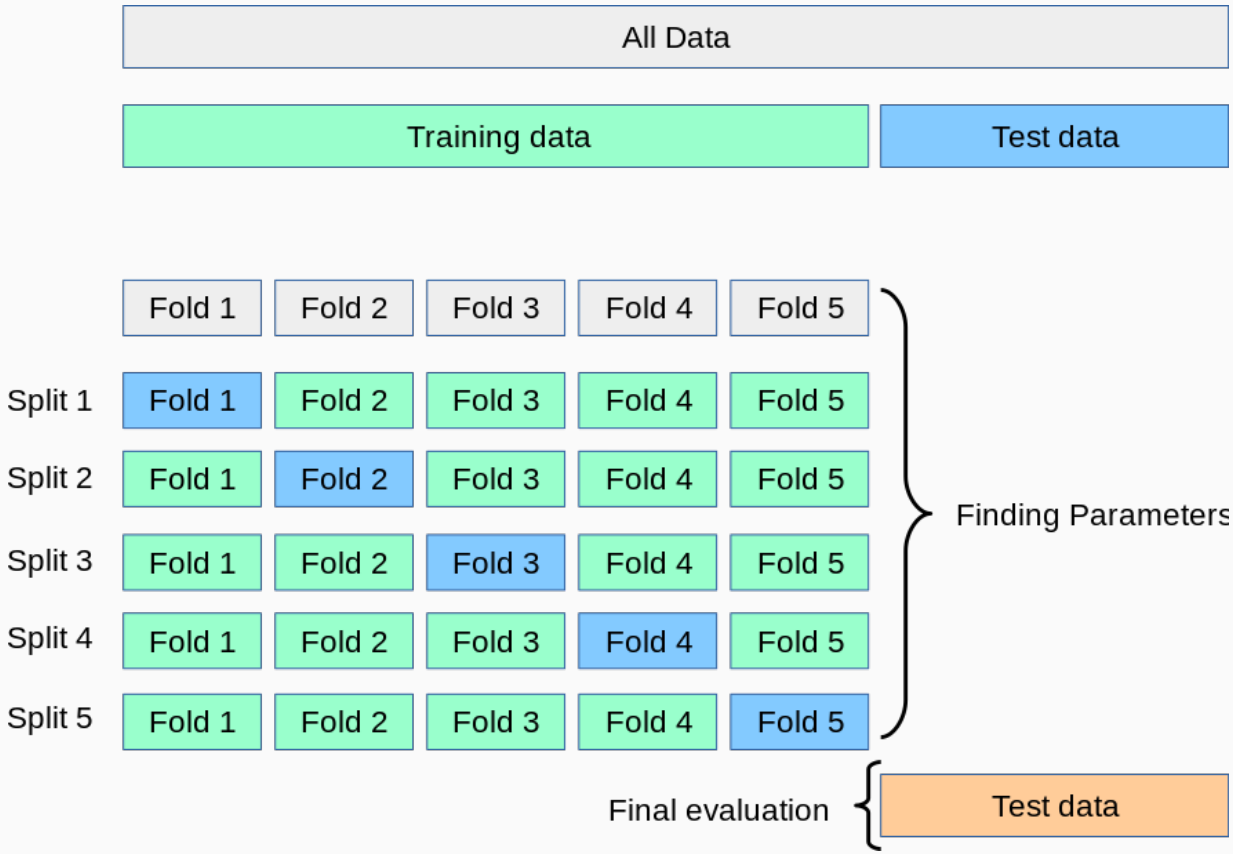
Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.

- For example, choosing the ℓ_1 penalty λ that achieves the best out-of-sample performance

Idea is to randomly divide the data into K equal-sized groups (folds). We leave out part k, fit the model to the other $K - 1$ parts (combined), and then obtain predictions for the left-out kth part.

Leave-one-out cross-validation: K-fold CV taken to its logical extreme with $K = n$.

K-Fold Cross Validation



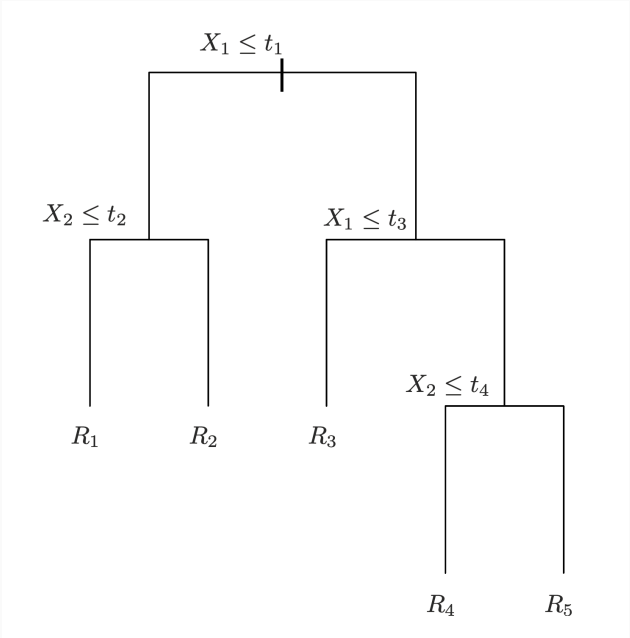
Source: scikit-learn

Tree-Based Methods

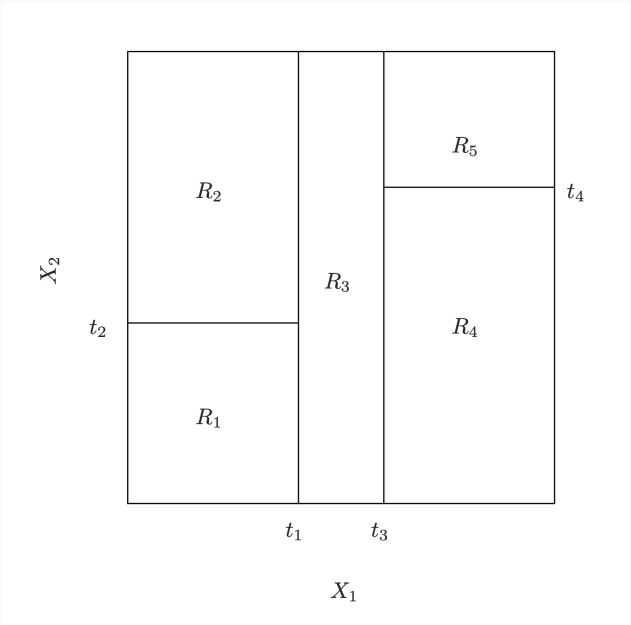
Tree-based methods involve **stratifying** or **segmenting** the predictor space into a number of simple regions.

Conceptually simple and especially powerful when combined with **ensemble methods**.

Can be used for **regression** and **classification** tasks.



(a) recursive binary splitting



(b) two-dimensional view

Basic idea:

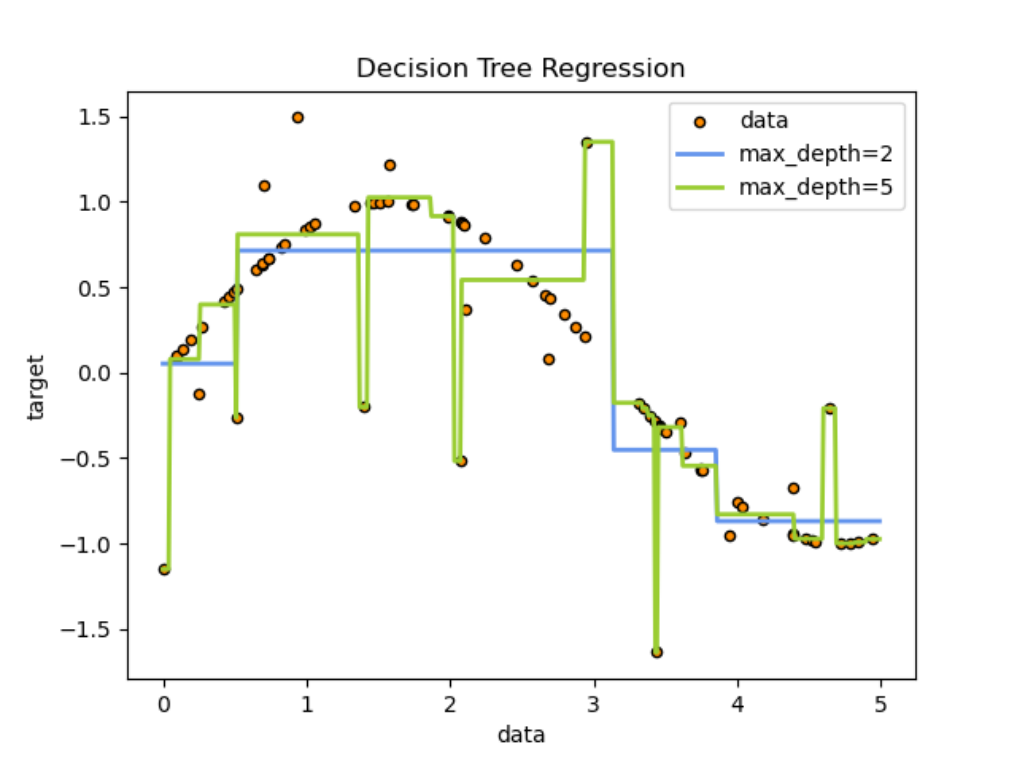
1. Divide the space of covariates $(X_1, X_2, \ldots, X_p)$ into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, we make the same prediction, which is the mean of the outcome variable for the observations in R_j.

3. The objective is to choose the R_j such that we minimize:

$$L(\hat{y}, y) = \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j}\right)^2$$

Tree-Based Methods



Source: scikit-learn

Tree-Based Methods

How does one optimally divide the space of covariates?

- Computationally infeasible to consider every possible partition of the feature space into J regions (NP-complete)
- Rely on **top-down, greedy algorithm** to split the data
  - **Top-down**: begins at the top of the tree and then successively splits the predictor space
  - **Greedy**: the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step

Recursive Binary Splitting

Approach: Choose the splitting variable j and the split point θ to minimize:

$$\min_{j,\theta}\left[\min_{c_1}\sum_{x_i\in R_1(j,\theta)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,\theta)}(y_i-c_2)^2\right],$$

where $R_1(j,\theta)=\{X\mid X_j\leq\theta\}$ and $R_2(j,\theta)=\{X\mid X_j>\theta\}$.

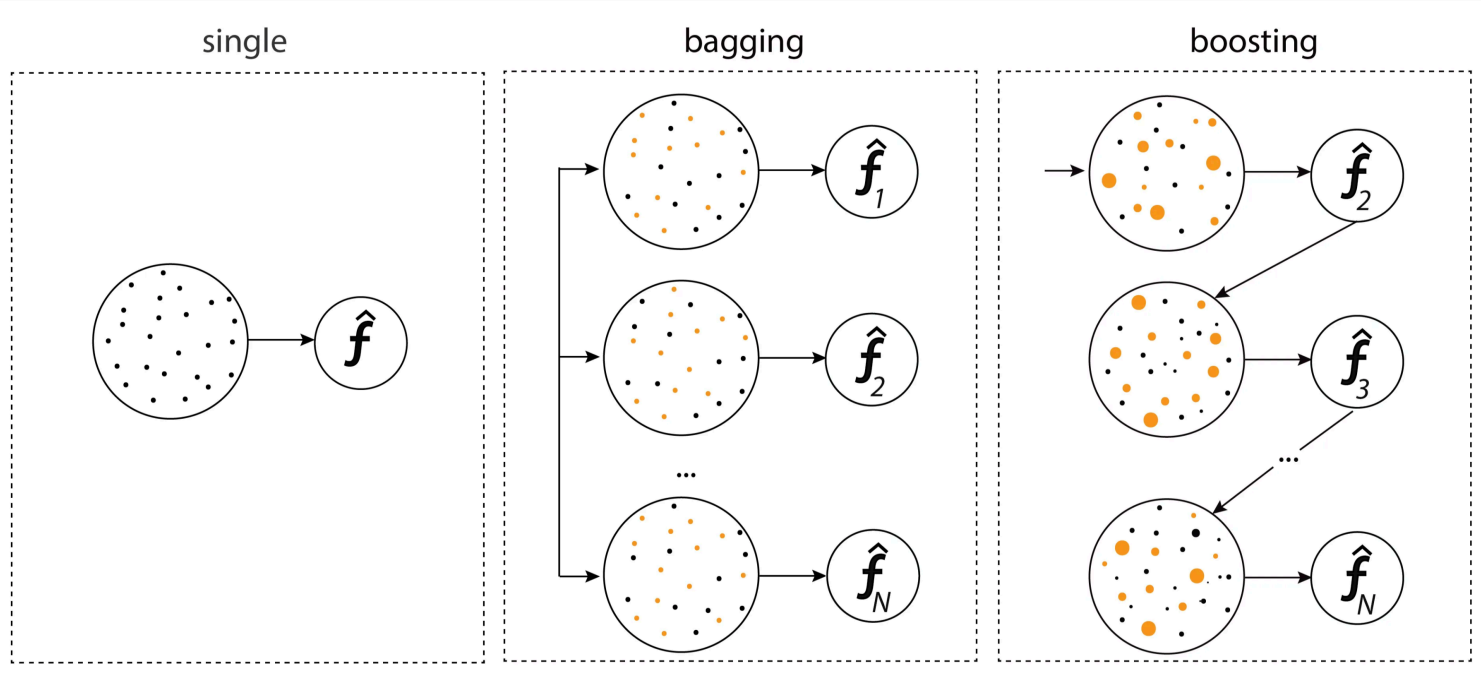
For any j and θ, the inner minimization are solved by

$$\hat{c}_1=\frac{1}{|R_1(j,\theta)|}\sum_{x_i\in R_1(j,\theta)}y_i \text{ and } \hat{c}_2=\frac{1}{|R_2(j,\theta)|}\sum_{x_i\in R_2(j,\theta)}y_i$$

Ensemble Models

Ensemble learning reduces model instability by averaging predictions over multiple instances of similar models.

Combines “weak” learners (high bias, low variance) to produce a powerful “committee”.



Bagging

Name comes from “bootstrap aggregation”.

Approach: bootstrap data and train separate tree on each bootstrapped data set, then average all predictions:

$$\hat{f}_{\text{bag}}(x) = \sum_{b=1}^{B} \beta_b \hat{F}_b(x),$$

where $\hat{F}_b$ is the weak learners trained on the bth bootstrapped sample and, typically, $\beta_b = 1/B$.

- Lowers variance by ensuring reduced risk of influence of a few data points

Random Forests

Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees.

- Each time a split in a tree is formed, a random selection of $m < p$ predictors are considered
- Rule of thumb: $m \approx \sqrt{p}$
- Random forests tend to outperform bagging when decision trees are the base learner

Boosting

Similarly, boosting outputs of many "weak" learners to produce a powerful "committee."

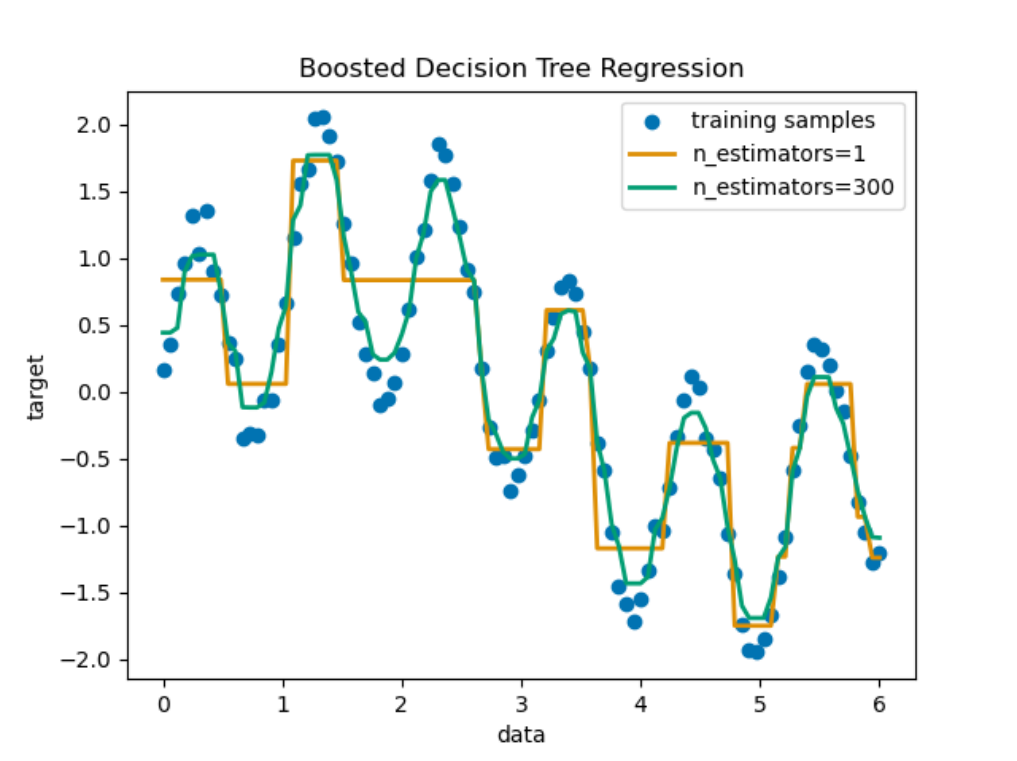
Shallow trees are grown *sequentially*, with new trees fit with the residuals from the previous tree:

$$\hat{f}_i(x) = \hat{f}_{i-1}(x) + \lambda \hat{F}_i(x)$$

Necessitates tuning three main parameters, which can be chosen with cross-validation:

- B: number of trees
- d: number of splits per tree (typically small relative to p)
- λ: learning rate (typically 0.01 or 0.001)

Boosting



Source: scikit-learn

Boosting Algorithm

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   2.1 Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Gradient Boosting Algorithm

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

   - Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$
   - For $j = 1, 2, \ldots, J_m$ compute

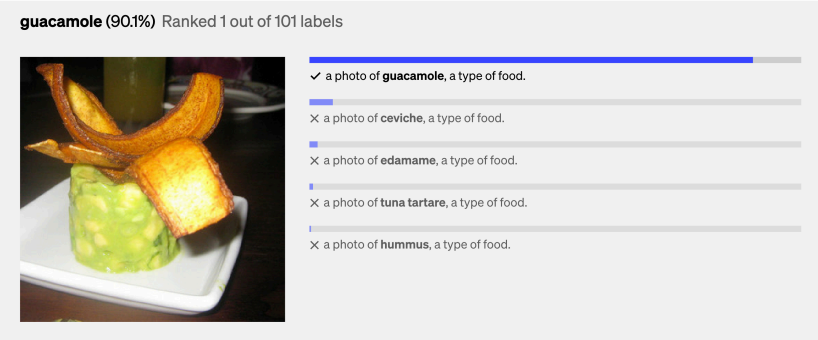
$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   - Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
3. Output $\hat{f}(x) = f_M(x)$

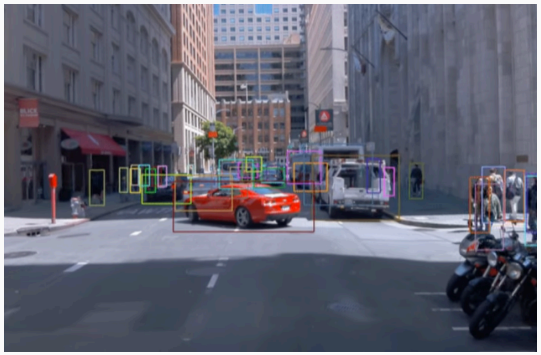
Deep Learning and Neural Networks

Most common in computer vision and natural language processing tasks → tasks with *unstructured* data

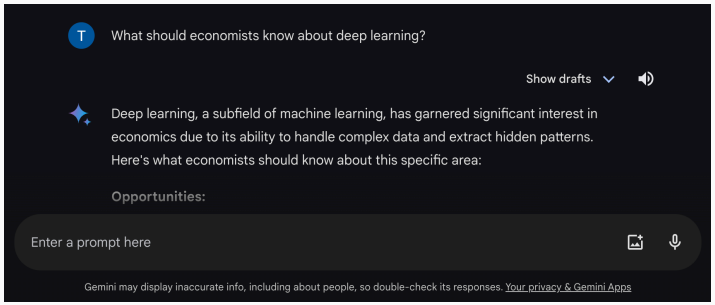
Neural networks are very flexible → proven to be *universal function approximators*



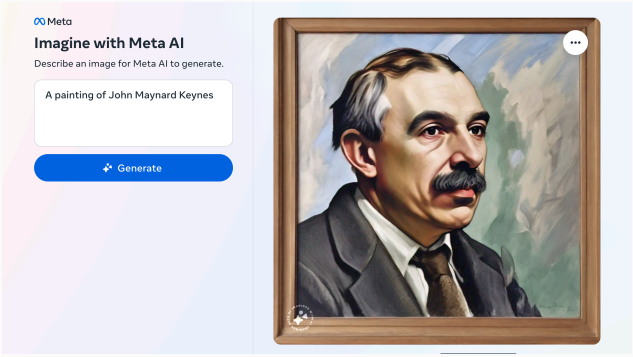
Source: OpenAI



Source: Waymo

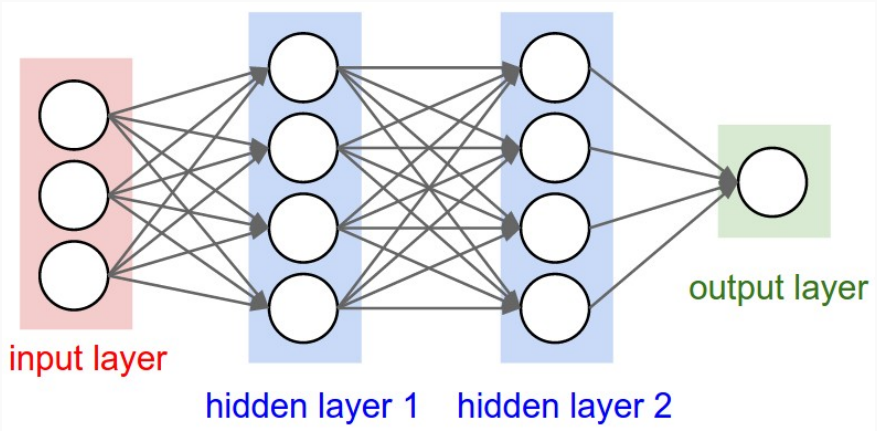


Source: Google Gemini



Source: Meta AI

Feed-Forward Neural Networks



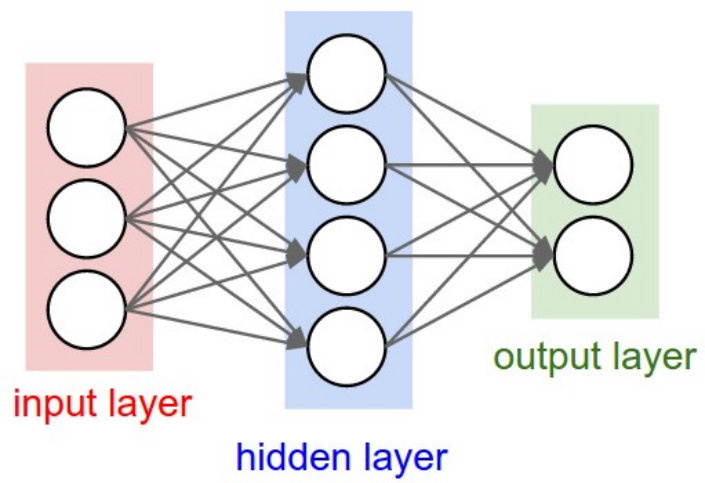
Source: CS 231N

Series of connected **layers** that transform the inputs **x** to the output **y**.

Each layer is made up of **nodes**, which each have a non-linear **activation function**.

Nodes in different layers are connected by **connections**, which each have an associated **weight** (parameter to learn).

Feed-Forward Neural Networks



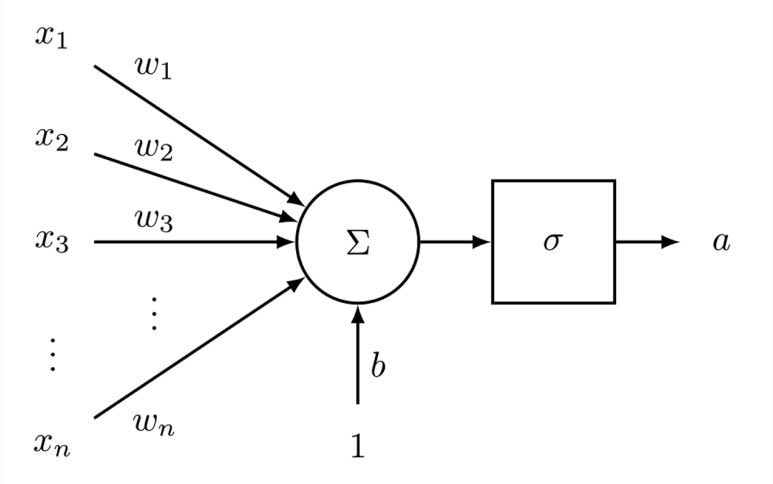
Source: CS 231N

The number of nodes in the output layer depends on the dimension of y. For example, a K-class classification problem has K output nodes.

Logistic Regression as a Neural Network

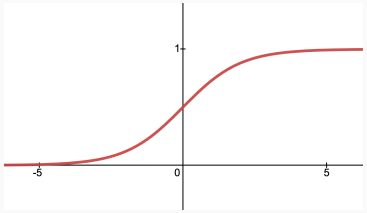
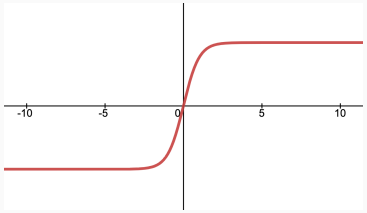
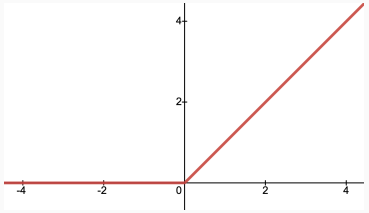
Logistic regression can be represented as a neural network with one neuron:

$$a = \sigma(b + w^T x) = \frac{1}{1 + \exp\left(-\left(b + w^T x\right)\right)}$$



- x = input vector
- w = weight vector
- b = bias scalar
- σ = (non-linear) activation function
- a = scalar activation (output)

Common Activation Functions

	Sigmoid	Hyperbolic Tangent (tanh)	Rectified Linear Unit (ReLU)
Function	$\sigma(x) = \frac{1}{1+e^{-x}}$	$\tanh(x) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$\mathrm{ReLU}(x) = \max(0, x)$
			
Range	(0, 1)	(-1, 1)	$[0, \infty)$
Advantages		Zero-centered. Faster convergence than sigmoid.	No vanishing gradient problem. Very fast to compute.
Disadvantages	Vanishing gradient problems.	Vanishing gradient problems.	"Dying ReLU" problem.

Training a Neural Network

The overall network can be represented as

$$\hat{y} = g^L \left(w'^L g^{L-1} \left(w'^{L-1} \cdots g^1 \left(w'^1 x \right) \cdots \right) \right)$$

where L is the number of layers, g^l is the activation function, and w^l are the weights between layer $l-1$ and l.

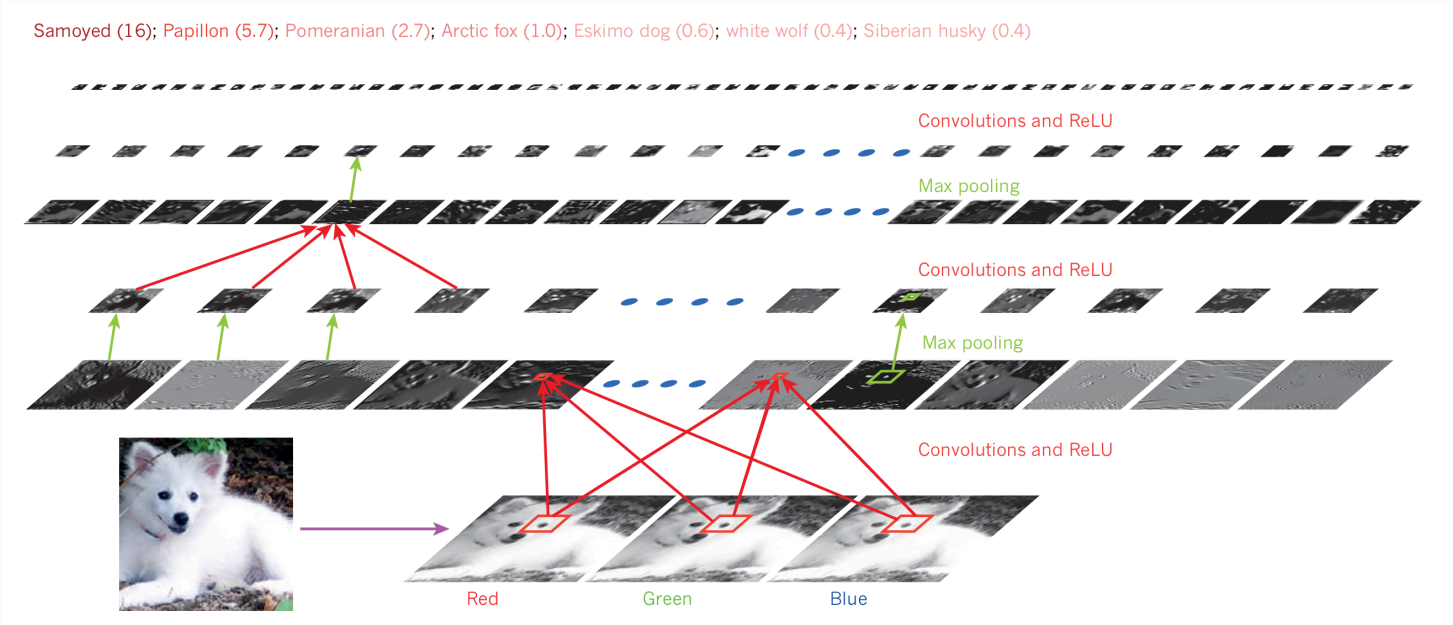
The loss function is $\mathcal{L}(y, \hat{y})$.

Update weights via **stochastic gradient descent**:

$$w_{n+1} = w_n - \eta \nabla_{w_n} \mathcal{L},$$

where $\eta > 0$ is the learning rate (step size) and we use **backpropagation** (form of automatic differentiation) to find the negative gradient of the loss function.

Convolutional Neural Networks



Source: LeCun, Bengio, and Hinton (2015)

“Machine Learning as a Tool for Hypothesis Generation”

Source: Ludwig and Mullainathan (2024)

Unsupervised Learning

Unsupervised Learning

No outcome variable, just a set of features.

Objective is more fuzzy:

- find groups of samples that behave similarly
- find features that behave similarly
- find linear combinations of features with the most variation

Can be useful as a pre-processing step for supervised learning.

Unsupervised Learning Examples

- Categorize news articles with a clustering algorithm (Athey, Mobius, and Pál, 2017)
- Cluster CEO behaviors using latent Dirichlet allocation (Bandiera et al., 2020)
- Topic analysis of FOMC statements with latent Dirichlet allocation (Hansen, McMahon, and Prat, 2018)
- Creating a low-dimensional representation of the latent product space for ready-to-eat cereals using tSTE (Magnolfi, McClure, and Sorenson, 2022)

Some Unsupervised Learning Methods

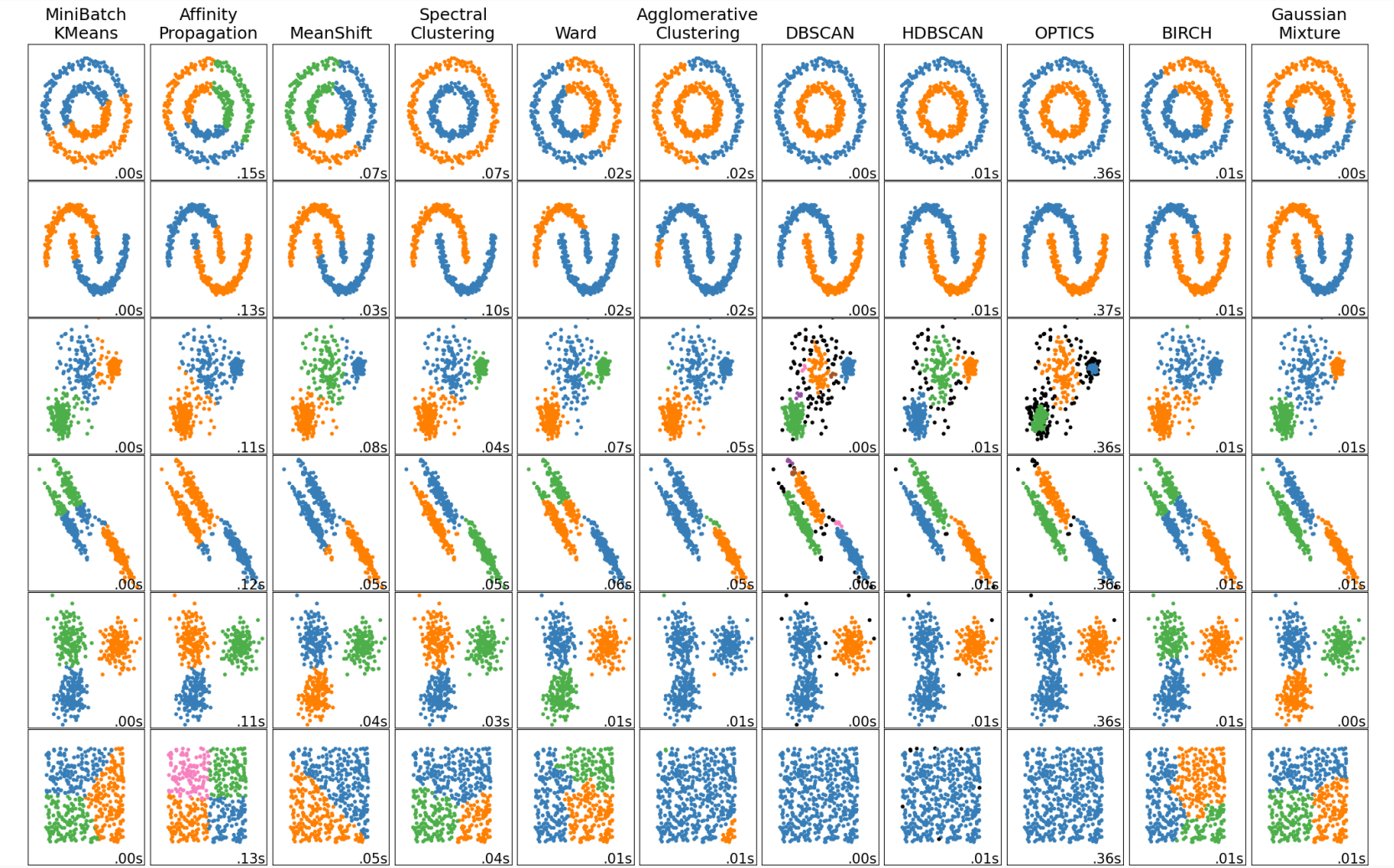
Clustering

- K-Means Clustering
- Mixture Modeling
- Agglomerative Hierarchical Clustering
- Divisive Hierarchical Clustering

Dimensionality Reduction

- Principal Component Analysis (PCA)
- Factor Analysis
- t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Autoencoders

Clustering Algorithms in `sklearn`



Source: scikit-learn

K-Means Clustering

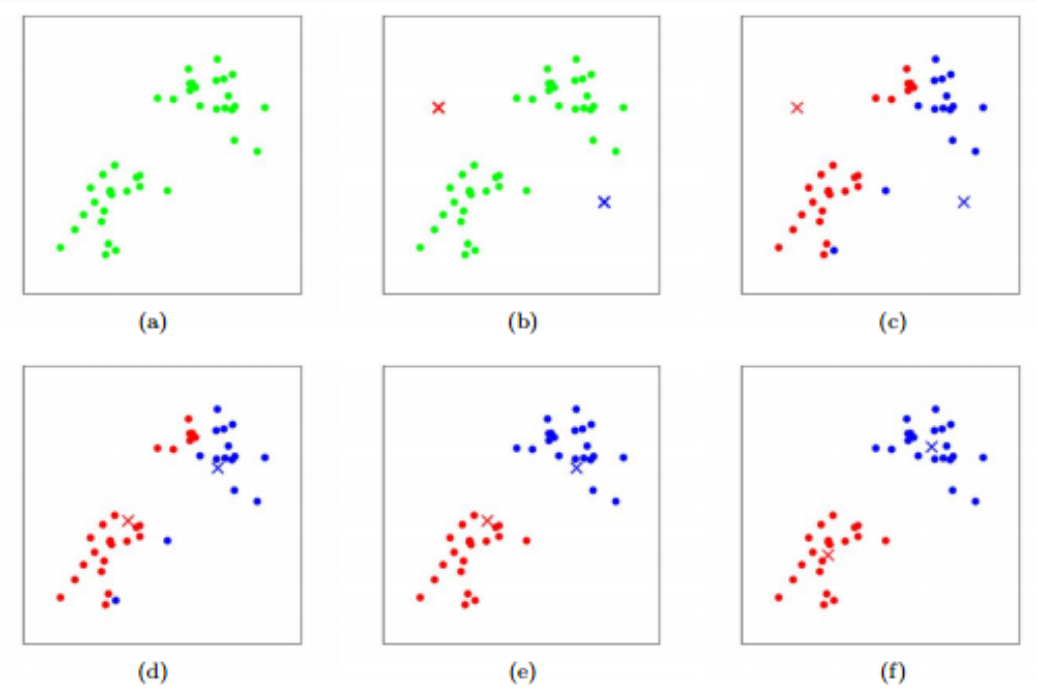
Goal is to partition the feature space into K subspaces.

Clustering methods are often used for exploratory data analysis, creating new features, and recommendation engines.

Algorithm:

1. Initialize a set of K centroids, b_1, b_2, ..., b_K, randomly
2. Repeat until convergence:
   - Assign each unit to the cluster that minimizes $C_i = \textbf{arg min}_{c \in \{1,\ldots,K\}} \|X_i - b_c\|^2$
   - Update the centroids as the average of the X_i in each of the clusters: $b_c = \sum_{i:C_i=c} X_i / \sum_{i:C_i=c} 1$

K-Means Clustering



Source: Hastie, Tibshirani, and Friedman (2009)

Mixture Models

Mixture models suppose the data is an i.i.d sample from some population described by a probability density function f.

This density function is a **mixture** of K component density functions, $f_1, f_2, ..., f_K$. The component densities represent **clusters**.

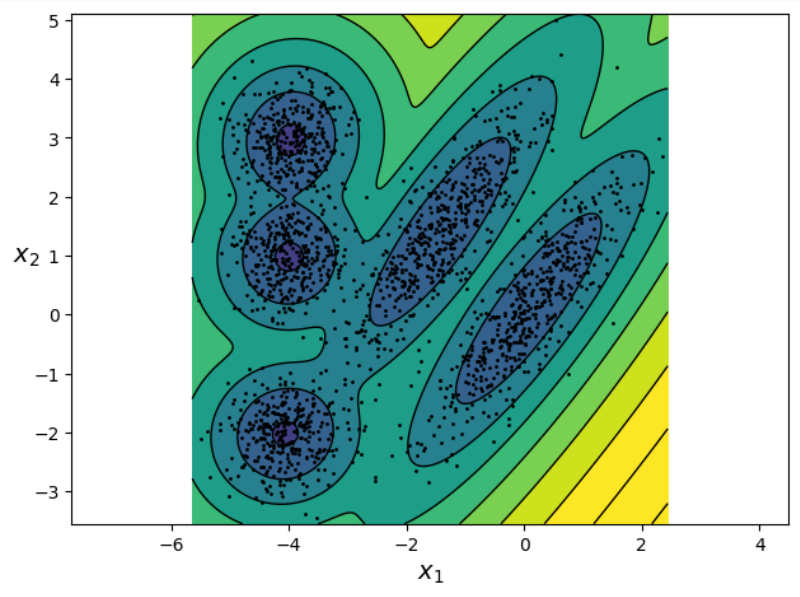
If f_k are in the same parametric family, but with different parameters, we can write the model as

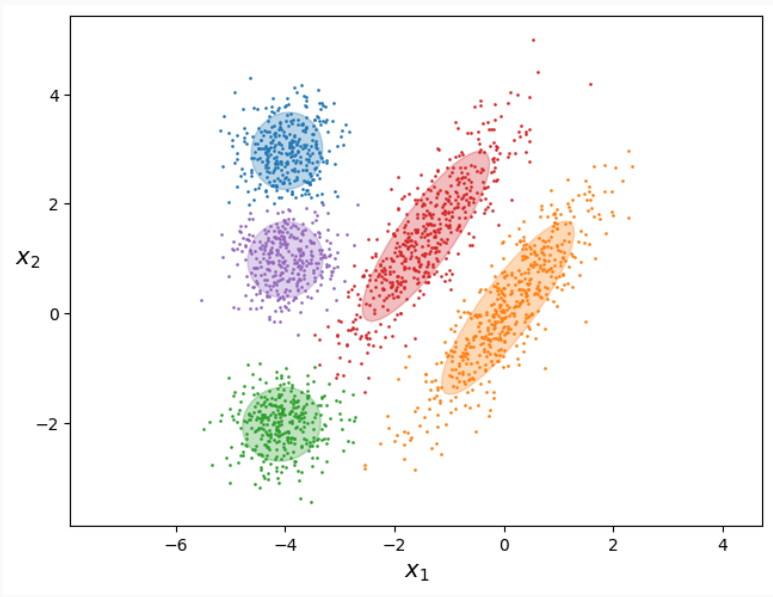
$$f(x) = \sum_{k=1}^{K} \lambda_k f(x; \theta_k)$$

where λ_k is the mixing weight and θ_k is the parameter vector governing f_k.

Estimate $\theta = (\lambda_1, \lambda_2, ..., \lambda_K, \theta_1, \theta_2, ..., \theta_K)$ via maximum likelihood, using the EM algorithm, or corresponding Bayesian approaches.

Mixture Models





Variational Inference

x is a vector of n observations, z is a vector of m hidden variables and α are additional fixed parameters.

We are interested in the posterior distribution

$$p(z|x,\alpha) = \frac{p(z,x|\alpha)}{\int_z p(z,x|\alpha)}$$

Not always straightforward to compute the posterior for many models (e.g., integral may not have closed form).

The main idea behind variational methods is to pick a family of distributions over the latent variables with its own variational parameters,

$$q(z|\nu)$$

then find the parameters that make q close to the posterior of interest. Use q with the fitted parameters as a proxy for the posterior.

Latent Dirichlet Allocation (LDA)

A Bayesian topic model to group words and documents in a corpus into topics.

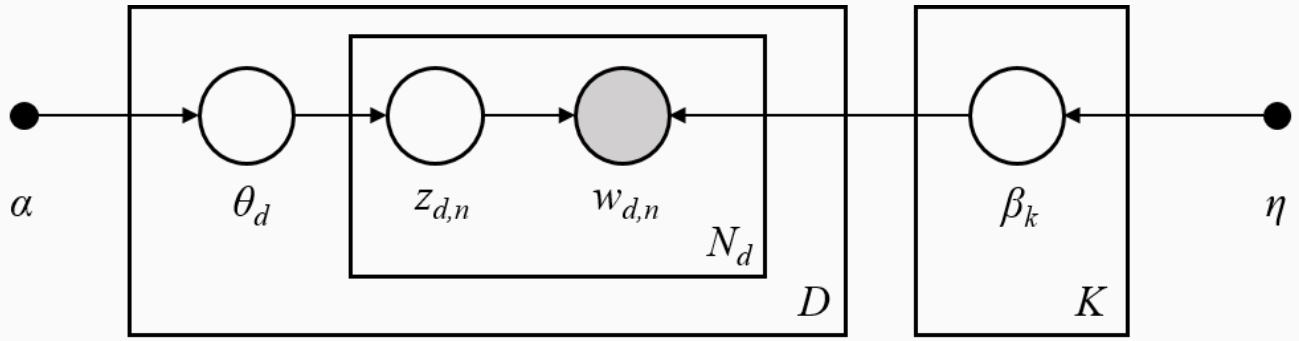
Every document is a mixture of topics and every topic is a mixture of words.

"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

Source: Blei, Ng, and Jordan (2003)

Latent Dirichlet Allocation (LDA)



- Collection of D documents
- Document is a sequence of N words
- K topics in a corpus
- Boxes represent repeated sampling
- Shaded nodes are observed random variables and unshaded nodes are latent variables

Latent Dirichlet Allocation (LDA)

Data generating process:

1. For each topic $k \in K$, draw $\beta_k \sim \text{Dirichlet}(\eta)$, the distribution over the words, i.e. the probability of a word appearing in topic k, and η is the prior of the topic word distribution
2. For each document $d \in D$, draw the topic proportions $\theta_d \sim \text{Dirichlet}(\alpha)$.
3. For each word i in document d:
   3.1 Draw the topic assignment $z_{di} \sim \text{Multinomial}(\theta_d)$.
   3.2 Draw the observed word $w_{ij} \sim \text{Multinomial}(\beta_{z_{di}})$.

Parameter estimating with posterior distribution $p(z, \theta, \beta | w, \alpha, \eta) = \frac{p(z, \theta, \beta | \alpha, \eta)}{p(w | \alpha, \eta)}$. Since posterior is intractable, use variational Bayes method with simpler distribution $q(z, \theta, \beta | \lambda, \phi, \gamma)$ to approximate it (minimizing KL divergence between $q(z, \theta, \beta)$ and the true posterior).

Machine Learning in Python

Packages

scikit-learn: In addition to providing implementations of supervised and unsupervised learning models, provides tools for model fitting, data preprocessing, model selection, and model evaluation.

XGBoost and *LightGBM*: Provide efficient implementation of gradient boosting frameworks.

PyTorch and *TensorFlow*: Provide deep learning frameworks (*PyTorch* has become dominant).

Example with `lightgbm` and `sklearn`

Setup

```
import lightgbm as lgb
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.datasets import fetch_california_housing

# Load a sample dataset (California Housing dataset)
# Target: median house value per Census block
# Attributes: MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude
data = fetch_california_housing(as_frame=True)

X, y = data.data, data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Example with `lightgbm` and `sklearn`

```
# Define a LightGBM regressor
lgb_model = lgb.LGBMRegressor()

# Define a parameter grid for grid search
param_grid = {
    "num_leaves": [15, 31, 50],  # Vary the number of leaves
    "learning_rate": [0.1, 0.01, 0.001],  # Vary the learning rate
    "n_estimators": [100, 500, 1000]  # Vary the number of estimators (trees)
}

# Initialize a GridSearchCV object
grid_search = GridSearchCV(
    estimator=lgb_model,
    param_grid=param_grid,
    scoring="neg_mean_squared_error",
    cv=5 # 5-fold cross-validation
)

# Perform grid search
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:")
print(grid_search.best_params_)
>> Best Hyperparameters:
>> {'learning_rate': 0.1, 'n_estimators': 500, 'num_leaves': 31}

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions on the test set
y_test_pred = best_model.predict(X_test)
```

Further Resources

Further Resources

Machine Learning:

- *The Elements of Statistical Learning*, by Trevor Hastie, Robert Tibshirani, and Jerome Friedman
- *Probabilistic Machine Learning*, by Kevin P. Murphy
- *Applied Causal Inference Powered by ML and AI*, by Victor Chernozhukov et al.

Deep Learning:

- *Neural Networks and Deep Learning*, by Michael Nielson
- *Deep Learning*, by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
- *Deep Learning with PyTorch*, by Eli Stevens, Luca Antiga, and Thomas Viehmann
- Neural Networks: Zero to Hero, by Andrej Karpathy
- Neural Networks, by 3Blue1Brown
- Stanford CS231N: Convolutional Neural Networks for Visual Recognition
- Stanford CS224N: Natural Language Processing with Deep Learning
- Harvard Economics 2355: Unleashing Novel Data at Scale

Self-Supervised Learning and Foundation Models

- "A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT"
- "On the Opportunities and Risks of Foundation Models"

Machine Learning in Economics

Reviews of ML in economics:

- Mullainathan and Spiess (2017); Athey (2019); Athey and Imbens (2019); Kleinberg et al. (2015)

ML and causal inference:

- Chernozhukov et al. (2018a); Chernozhukov et al. (2018b); Athey, Tibshirani, and Wager (2019); Künzel et al. (2018)