

# Course Overview

---

Mengye Ren

NYU

September 5, 2023

# Contents

- 1 Logistics
- 2 Course Overview and Goals
- 3 Introduction to Machine Learning

- Class webpage: <https://nyu-cs2565.github.io/2023-fall>
  - Course materials (lecture slides, homeworks) will be made available on the website
- Announcements via Brightspace
- Discussion / questions on CampusWire

# 6608

 <https://campuswire.com/p/G74AFD6C8>

- Office Hour: Tuesday 1:00-2:00 pm, Room 508, 60 Fifth Ave.

# Logistics

- Instructor:
  - Mengye Ren (mr3182@nyu.edu)
- Graders:
  - Shreya Agarwal (sa6981@nyu.edu)
  - Jash Rathod (jsr10000@nyu.edu)
- All course material, assignment, and exam related questions should be posted on CampusWire
- I will only respond to administration related emails

- 4 assignments (40%)
- Midterm Exam (Oct 24) (30%)
- Final Project (30%)
- Extra credits (2%) answer other students' questions in a substantial and helpful way on Campuswire

- Submit through Gradescope as a **PDF document**
- Late policy: You have 4 late days in total which can be used throughout the semester without penalty (see more details on website).
- You can discuss with other students on the homework assignments, but:
  - Write up the solutions and code on your own;
  - List the names of the students you discussed with.
- If your solution or code is substantially similar to other students then the incident will be reported to the University.

# Final Project

- Groups of maximum three.
- The goal is to apply the ML algorithms in real applications.
- Goals:
  - Find or collect a dataset (can be related to your area of study/research)
  - Survey existing approaches
  - Implement a set of ML algorithms and compare their performance
- Project proposal due Friday, Oct 27, 2023, 11:59PM
- Last lecture: 3min project presentation
- Final report due Wednesday, Dec 15, 2023, 11:59PM



# Prerequisites

- Multivariate Calculus: partial derivatives/gradient.
- Linear Algebra: vector/matrix manipulations, properties.
- Probability Theory: common distributions; Bayes Rule.
- Statistics: expectation, variance, covariance, median; maximum likelihood.
- Programming: Python, numpy

## Course Overview and Goals

# Syllabus (Tentative)

12 weeks of instruction + 1 week midterm exam + 1 week project presentation

- 2 weeks: introduction to **machine learning, optimization**
- 2 weeks: **Linear** methods for binary classification and regression (also **kernel methods**)
- 2 weeks: **Probabilistic models, Bayesian** methods
- 1 week: **Multiclass** classification and introduction to **structured prediction**
- 3 weeks: **Nonlinear** methods (**trees, ensemble** methods, and **neural networks**)
- 1 week: **Unsupervised** learning: **clustering** and **latent variable** models
- 1 week: **Reinforcement** learning
- More detailed schedule on the course website (still subject to change)

# The high level goals of the class

- Our focus will be on the fundamental building blocks of machine learning
- Prepare the fundamental toolkit – fancy new methods are often combination of the techniques
- Understand what kind of problems can ML help solve
- Despite the large number of methods, understand the pros & cons of each method, understand the motivation why we choose one method over the other
- Apply ML in practical problems

# The level of the class

- Many ML algorithms have been implemented in standard libraries (e.g. sklearn)
- Many people only know how to call these library functions.
- We will learn how to implement each ML algorithm **from scratch** using numpy alone, without any ML libraries.
- Once we have implemented an algorithm from scratch once, we will use the sklearn version.

# Introduction to Machine Learning

# What is learning?

*"The activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something."*

*Merriam Webster dictionary*

# What is learning?

*"The activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something."*

*Merriam Webster dictionary*

*"A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ ."*

*Tom Mitchell*



# Applications of machine learning

- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech

# Applications of machine learning

- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech
- Machine learning approach: program an algorithm to automatically learn from data, or from experience. Typically our goal is to solve a prediction problem of the format:
  - Given an **input**  $x$ ,
  - **Predict** an **output**  $y$ .

# Applications of machine learning

- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech
- Machine learning approach: program an algorithm to automatically learn from data, or from experience. Typically our goal is to solve a prediction problem of the format:
  - Given an **input**  $x$ ,
  - **Predict** an **output**  $y$ .
- Why might you want to use a learning algorithm?

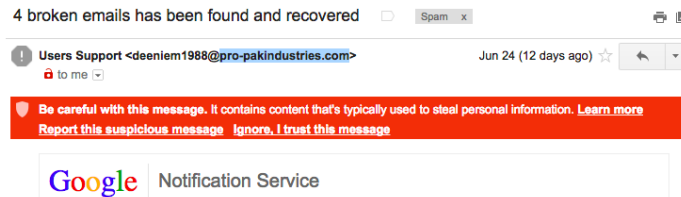
# Applications of machine learning

- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech
- Machine learning approach: program an algorithm to automatically learn from data, or from experience. Typically our goal is to solve a prediction problem of the format:
  - Given an **input**  $x$ ,
  - **Predict** an **output**  $y$ .
- Why might you want to use a learning algorithm?
  - hard to code up a solution by hand (e.g. vision, speech)
  - system needs to adapt to a changing environment (e.g. spam detection)
  - want the system to perform *better* than the human programmers
  - privacy/fairness (e.g. ranking search results)

# Example: Spam Detection

Let's start with a few canonical examples.

- **Input  $x$ :** Incoming email



- **Output  $y$ :** "SPAM" or "NOT SPAM"
- This is a **binary classification** problem: there are two possible outputs.

## Example: Medical Diagnosis

- **Input  $x$ :** Symptoms (fever, cough, fast breathing, shaking, nausea, ...)
- **Output  $y$ :** Diagnosis (pneumonia, flu, common cold, bronchitis, ...)
- A **multiclass classification** problem: choosing an output out of a *discrete* set of possible outputs.

How do we express uncertainty about the output?

- **Probabilistic classification** or **soft classification**:

$$\mathbb{P}(\text{pneumonia}) = 0.7$$

$$\mathbb{P}(\text{flu}) = 0.2$$

$$\vdots \quad \quad \vdots$$

## Example: Predicting a Stock Price

- **Input  $x$ :** History of the stock's prices
- **Output  $y$ :** The price of the stock at the close of the next day
- This is called a **regression** problem (for historical reasons): the output is *continuous*.

# Comparison to Rule-Based Approaches (Expert Systems)

- Consider the problem of medical diagnosis.
  - ① Talk to experts (in this case, medical doctors).
  - ② Understand how the experts come up with a diagnosis.
  - ③ Implement this process as an algorithm (a **rule-based system**): e.g., a set of symptoms  $\rightarrow$  a particular diagnosis.
  - ④ Potentially use logical deduction to infer new rules from the rules that are stored in the knowledge base.



# Rule-Based Approach

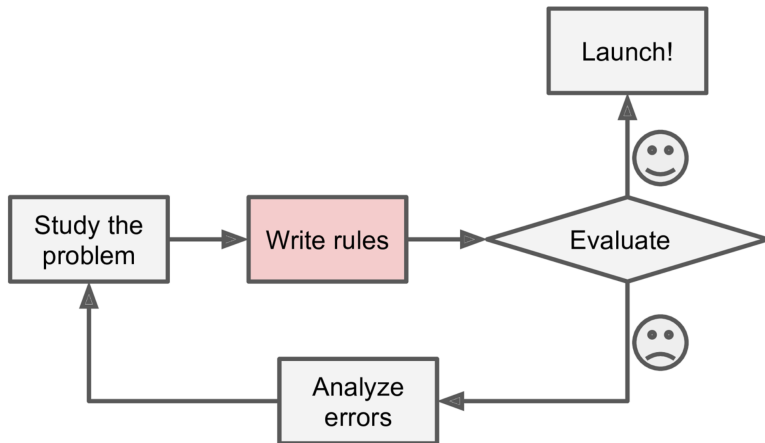


Fig 1-1 from *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurelien Geron (2017).

# Advantages of Rule-Based Approaches

- Leverage existing domain expertise.
- Generally **interpretable**: We can describe the rule to another human
- Produce reliable answers for the scenarios that are included in the knowledge bases.

# Limitations of Rule-Based Systems

- Labor intensive to build: experts' time is expensive.
- Rules work very well for areas they cover, but often do not **generalize** to unanticipated input combinations.
- Don't naturally handle uncertainty.

# The Machine Learning Approach

- Instead of explicitly engineering the process that a human expert would use to make the decision...
- We have the machine **learn** on its own from inputs and outputs (decisions).
- We provide **training data**: many examples of (input  $x$  , output  $y$ ) pairs, e.g.
  - A set of videos, and whether or not each has a cat in it.
  - A set of emails, and whether or not each one should go to the spam folder.
- Learning from training data of this form (inputs and outputs) is called **supervised learning**.

# Machine Learning Algorithm

- A **machine learning algorithm** learns from the training data:
  - **Input:** Training Data (e.g., emails  $x$  and their labels  $y$ )
  - **Output:** A prediction function that produces output  $y$  given input  $x$ .
- The goal of machine learning is to find the “best” (to be defined) prediction function **automatically, based on the training data**
- The success of ML depends on
  - The availability of large amounts of data;
  - **Generalization** to unseen samples (the test set): just memorizing the training set will not be useful.

# Machine Learning Approach

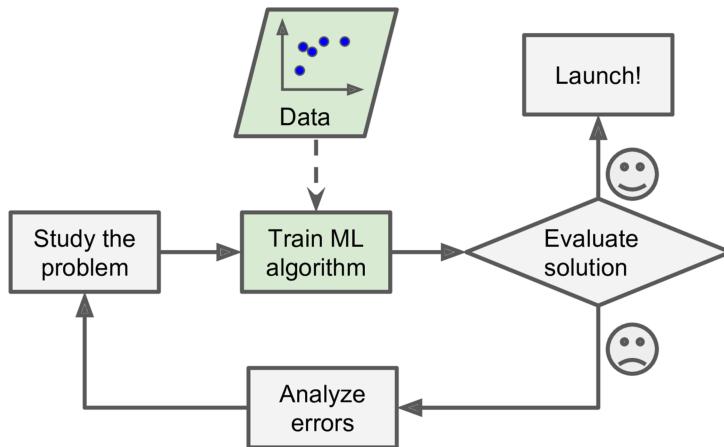


Fig 1-2 from *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurelien Geron (2017).

# Key concepts

- The most common **ML problem types**:

# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)



# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)
  - Regression

# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)
  - Regression
- **Prediction function**: predicts output  $y$  (e.g. spam or not?) given input  $x$  (e.g. email)

# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)
  - Regression
- **Prediction function**: predicts output  $y$  (e.g. spam or not?) given input  $x$  (e.g. email)
- **Training data**: a set of (input  $x$ , output  $y$ ) pairs

# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)
  - Regression
- **Prediction function**: predicts output  $y$  (e.g. spam or not?) given input  $x$  (e.g. email)
- **Training data**: a set of (input  $x$ , output  $y$ ) pairs
- **Supervised learning algorithm**: takes training data and produces a prediction function

# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)
  - Regression
- **Prediction function**: predicts output  $y$  (e.g. spam or not?) given input  $x$  (e.g. email)
- **Training data**: a set of (input  $x$ , output  $y$ ) pairs
- **Supervised learning algorithm**: takes training data and produces a prediction function
- Beyond prediction

# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)
  - Regression
- **Prediction function**: predicts output  $y$  (e.g. spam or not?) given input  $x$  (e.g. email)
- **Training data**: a set of (input  $x$ , output  $y$ ) pairs
- **Supervised learning algorithm**: takes training data and produces a prediction function
- Beyond prediction
  - **Unsupervised learning**: finding structures in data, e.g. clustering

# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)
  - Regression
- **Prediction function**: predicts output  $y$  (e.g. spam or not?) given input  $x$  (e.g. email)
- **Training data**: a set of (input  $x$ , output  $y$ ) pairs
- **Supervised learning algorithm**: takes training data and produces a prediction function
- Beyond prediction
  - **Unsupervised learning**: finding structures in data, e.g. clustering
  - **Reinforcement learning**: optimizing long-term objective, e.g. Go

# Key concepts

- The most common **ML problem types**:
  - Classification (binary and multiclass)
  - Regression
- **Prediction function**: predicts output  $y$  (e.g. spam or not?) given input  $x$  (e.g. email)
- **Training data**: a set of (input  $x$ , output  $y$ ) pairs
- **Supervised learning algorithm**: takes training data and produces a prediction function
- Beyond prediction
  - **Unsupervised learning**: finding structures in data, e.g. clustering
  - **Reinforcement learning**: optimizing long-term objective, e.g. Go
  - **Representation learning**: learning good features of real-world objects, e.g. text



# Core Questions in Machine Learning

Given any task, the following questions need to be answered:

- **Modeling:** What class of prediction functions are we considering?
- **Learning:** How do we learn the “best” prediction function in this class from our training data?
- **Inference:** How do we compute the output of the prediction function for a new input?

# Relations to statistics

- It's similar to statistics...
  - Both fields try to uncover patterns in data
  - Both fields draw heavily on calculus, probability, and linear algebra, and share many of the same core algorithms

- It's similar to statistics...
  - Both fields try to uncover patterns in data
  - Both fields draw heavily on calculus, probability, and linear algebra, and share many of the same core algorithms
- But it's not statistics...
  - Stats is more concerned with **helping scientists and policymakers** draw good conclusions; ML is more concerned with **building autonomous agents**
  - Stats puts more emphasis on **interpretability and mathematical rigor**; ML puts more emphasis on predictive **performance, scalability, and autonomy**

- Nowadays, “machine learning” is often brought up with “artificial intelligence” (AI)

- Nowadays, “machine learning” is often brought up with “artificial intelligence” (AI)
- AI does not often imply a learning based system
  - Symbolic reasoning
  - Rule based system
  - Tree search
  - etc.

- Nowadays, “machine learning” is often brought up with “artificial intelligence” (AI)
- AI does not often imply a learning based system
  - Symbolic reasoning
  - Rule based system
  - Tree search
  - etc.
- Learning based system → learned based on the data → more flexibility, good at solving pattern recognition problems.

## Relations to human learning

- It is tempting to imagine machine learning as a component in AI just like human learning in ourselves.

# Relations to human learning

- It is tempting to imagine machine learning as a component in AI just like human learning in ourselves.
- Human learning is:
  - Very data efficient
  - An entire multitasking system (vision, language, motor control, etc.)
  - Takes at least a few years :)
- For serving specific purposes, machine learning doesn't have to look like human learning in the end.



# Relations to human learning

- It is tempting to imagine machine learning as a component in AI just like human learning in ourselves.
- Human learning is:
  - Very data efficient
  - An entire multitasking system (vision, language, motor control, etc.)
  - Takes at least a few years :)
- For serving specific purposes, machine learning doesn't have to look like human learning in the end.
- It may borrow ideas from biological systems (e.g. neural networks).

# Relations to human learning

- It is tempting to imagine machine learning as a component in AI just like human learning in ourselves.
- Human learning is:
  - Very data efficient
  - An entire multitasking system (vision, language, motor control, etc.)
  - Takes at least a few years :)
- For serving specific purposes, machine learning doesn't have to look like human learning in the end.
- It may borrow ideas from biological systems (e.g. neural networks).
- There may also be biological constraints.

# History of machine learning

- 1957 — Perceptron algorithm (implemented as a circuit!)
- 1959 — Arthur Samuel wrote a learning-based checkers program that could defeat him
- 1969 — Minsky and Papert's book *Perceptrons* (limitations of linear models)

# History of machine learning

- 1957 — Perceptron algorithm (implemented as a circuit!)
- 1959 — Arthur Samuel wrote a learning-based checkers program that could defeat him
- 1969 — Minsky and Papert's book *Perceptrons* (limitations of linear models)
- 1980s — Some foundational ideas
  - Connectionist psychologists explored neural models of cognition
  - 1984 — Leslie Valiant formalized the problem of learning as PAC learning
  - 1988 — Backpropagation (re-)discovered by Geoffrey Hinton and colleagues
  - 1988 — Judea Pearl's book *Probabilistic Reasoning in Intelligent Systems* introduced Bayesian networks

# History of machine learning

- 1990s — the “AI Winter”, a time of pessimism and low funding

# History of machine learning

- 1990s — the “AI Winter”, a time of pessimism and low funding
- A lot of practical ML algorithms were invented in the 90s.
  - Markov chain Monte Carlo, variational inference, kernels and support vector machines, boosting, convolutional networks

# History of machine learning

- 1990s — the “AI Winter”, a time of pessimism and low funding
- A lot of practical ML algorithms were invented in the 90s.
  - Markov chain Monte Carlo, variational inference, kernels and support vector machines, boosting, convolutional networks
- 2000s — applied AI fields (vision, NLP, etc.) adopted ML

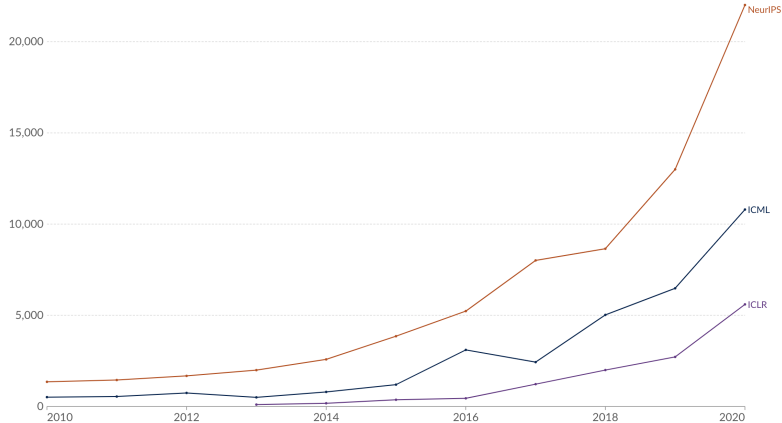
# History of machine learning

- 1990s — the “AI Winter”, a time of pessimism and low funding
- A lot of practical ML algorithms were invented in the 90s.
  - Markov chain Monte Carlo, variational inference, kernels and support vector machines, boosting, convolutional networks
- 2000s — applied AI fields (vision, NLP, etc.) adopted ML
- 2010s — deep learning
  - 2010–2012 — neural nets smashed previous records in speech-to-text and object recognition
  - increasing adoption by the tech industry
  - 2016 — AlphaGo defeated the human Go champion



# History of machine learning

Top ML conferences attendance over year:



# ML problems

- In ML problems, we generally need to:
  - Make a decision:
    - Move email to spam folder?

# ML problems

- In ML problems, we generally need to:
  - Make a decision:
    - Move email to spam folder?
  - Take an action:
    - In a self-driving car, make a right turn
    - Reject the hypothesis that  $\theta = 0$  (classical statistics)

# ML problems

- In ML problems, we generally need to:
  - Make a decision:
    - Move email to spam folder?
  - Take an action:
    - In a self-driving car, make a right turn
    - Reject the hypothesis that  $\theta = 0$  (classical statistics)
  - Produce some output:
    - Whose face is it in the image?
    - The Hindi translation of a Japanese input sentence

# ML problems

- In ML problems, we generally need to:
  - Make a decision:
    - Move email to spam folder?
  - Take an action:
    - In a self-driving car, make a right turn
    - Reject the hypothesis that  $\theta = 0$  (classical statistics)
  - Produce some output:
    - Whose face is it in the image?
    - The Hindi translation of a Japanese input sentence
  - Predicting where a storm will be in an hour (what forms of output are possible here?)

Inputs are often paired with **labels**.

## Examples of labels

- Whether or not the picture actually contains an animal
- The storm's location one hour after they query
- Which, if any, of the suggested URLs were selected

# Evaluation Criterion

Finding “optimal” outputs, under various definitions of optimality.

## Examples of Evaluation Criteria

- Is the classification correct?
- Does the transcription exactly match the spoken words?
  - Should we give partial credit (for getting only some of the words right)? How?
- How far is the storm from the predicted location? (If we're producing a point estimate)
- How likely is the storm's actual location under the predicted distribution? (If we're doing density prediction)

# Typical Sequence of Events

Many problem domains can be formalized as follows:

- ➊ Observe input  $x$ .
  - ➋ Predict an output  $\hat{y}$ .
  - ➌ Observe label  $y$ .
  - ➍ Evaluate output in relation to the label.
- Input space:  $\mathcal{X}$
  - Label space:  $\mathcal{Y}$



# Formalization

## Prediction Function

A **prediction function** gets input  $x \in \mathcal{X}$  and produces an output  $y \in \mathcal{Y}$  :

$$\begin{array}{rcl} f: \mathcal{X} & \rightarrow & \mathcal{Y} \\ x & \mapsto & f(x) \end{array}$$

# Formalization

## Prediction Function

A **prediction function** gets input  $x \in \mathcal{X}$  and produces an output  $y \in \mathcal{Y}$  :

$$\begin{aligned} f: \mathcal{X} &\rightarrow \mathcal{Y} \\ x &\mapsto f(x) \end{aligned}$$

## Loss Function

A **loss function** evaluates the output in the context of the true outcome  $y$ .

$$\begin{aligned} \ell: \mathcal{Y} \times \mathcal{Y} &\rightarrow \mathbb{R} \\ (\hat{y}, y) &\mapsto \ell(\hat{y}, y) \end{aligned}$$

# Evaluating a Prediction Function

**Goal:** Find the optimal prediction function.

**Intuition:** If we can evaluate how good a prediction function is, we can turn this into an optimization problem.

- The loss function  $\ell$  evaluates a *single* output
- How do we evaluate the prediction function *as a whole*?

Define a space where the prediction function is applicable

- Assume there is a **data generating distribution**  $P_{\mathcal{X} \times \mathcal{Y}}$ .
- All input/output pairs  $(x, y)$  are generated i.i.d. from  $P_{\mathcal{X} \times \mathcal{Y}}$ .

One common desideratum is to have a prediction function  $f(x)$  that “does well on average”:

$\ell(f(x), y)$  is usually small, in some sense

How can we formalize this?

## Definition

The **risk** of a prediction function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  is

$$R(f) = \mathbb{E}_{(x,y) \sim P_{\mathcal{X} \times \mathcal{Y}}} [\ell(f(x), y)].$$

In words, it's the **expected loss** of  $f$  over  $P_{\mathcal{X} \times \mathcal{Y}}$ .

**We can't actually compute the risk function:**

Since we don't know  $P_{\mathcal{X} \times \mathcal{Y}}$ , we cannot compute the expectation.

But we can **estimate** it.

# The Bayes Prediction Function

## Definition

A **Bayes prediction function**  $f^* : \mathcal{X} \rightarrow \mathcal{Y}$  is a function that achieves the *minimal risk* among all possible functions:

$$f^* \in \arg \min_f R(f),$$

where the minimum is taken over all functions from  $\mathcal{X}$  to  $\mathcal{Y}$ .

- The risk of a Bayes prediction function is called the **Bayes risk**.
- A Bayes prediction function is often called the “**target function**”, since it’s the best prediction function we can possibly produce.

## Example: Multiclass Classification

- Spaces:  $\mathcal{Y} = \{1, \dots, k\}$
- 0-1 loss:

$$\ell(\hat{y}, y) = 1(\hat{y} \neq y) := \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{otherwise.} \end{cases}$$

## Example: Multiclass Classification

- Spaces:  $\mathcal{Y} = \{1, \dots, k\}$

- 0-1 loss:

$$\ell(\hat{y}, y) = 1(\hat{y} \neq y) := \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{otherwise.} \end{cases}$$

- Risk:

$$\begin{aligned} R(f) &= \mathbb{E}[1(f(x) \neq y)] = 0 \cdot \mathbb{P}(f(x) = y) + 1 \cdot \mathbb{P}(f(x) \neq y) \\ &= \mathbb{P}(f(x) \neq y), \end{aligned}$$

which is just the misclassification error rate.

- The Bayes prediction function returns the most likely class:

$$f^*(x) \in \arg \max_{1 \leq c \leq k} \mathbb{P}(y = c \mid x)$$



## But we can't compute the risk!

- Can't compute  $R(f) = \mathbb{E}[\ell(f(x), y)]$  because we **don't know**  $P_{\mathcal{X} \times \mathcal{Y}}$ .

## But we can't compute the risk!

- Can't compute  $R(f) = \mathbb{E}[\ell(f(x), y)]$  because we **don't know**  $P_{\mathcal{X} \times \mathcal{Y}}$ .
- One thing we can do in ML/statistics/data science is **estimate** it:

## But we can't compute the risk!

- Can't compute  $R(f) = \mathbb{E}[\ell(f(x), y)]$  because we **don't know**  $P_{\mathcal{X} \times \mathcal{Y}}$ .
- One thing we can do in ML/statistics/data science is **estimate** it:

Assume we have sample data:

Let  $\mathcal{D}_n = ((x_1, y_1), \dots, (x_n, y_n))$  be drawn i.i.d. from  $\mathcal{P}_{\mathcal{X} \times \mathcal{Y}}$ .

## But we can't compute the risk!

- Can't compute  $R(f) = \mathbb{E}[\ell(f(x), y)]$  because we **don't know**  $P_{\mathcal{X} \times \mathcal{Y}}$ .
- One thing we can do in ML/statistics/data science is **estimate** it:

Assume we have sample data:

Let  $\mathcal{D}_n = ((x_1, y_1), \dots, (x_n, y_n))$  be drawn i.i.d. from  $\mathcal{P}_{\mathcal{X} \times \mathcal{Y}}$ .

- We draw inspiration from the strong law of large numbers:  
If  $z_1, \dots, z_n$  are i.i.d. with expected value  $\mathbb{E}z$ , then

$$\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n z_i = \mathbb{E}z,$$

with probability 1.

# The Empirical Risk

Let  $\mathcal{D}_n = ((x_1, y_1), \dots, (x_n, y_n))$  be drawn i.i.d. from  $\mathcal{P}_{\mathcal{X} \times \mathcal{Y}}$ .

## Definition

The **empirical risk** of  $f : \mathcal{X} \rightarrow \mathcal{A}$  with respect to  $\mathcal{D}_n$  is

$$\hat{R}_n(f) = \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i).$$

By the strong law of large numbers,

$$\lim_{n \rightarrow \infty} \hat{R}_n(f) = R(f),$$

almost surely.

# Empirical Risk Minimization

## Definition

A function  $\hat{f}$  is an **empirical risk minimizer** if

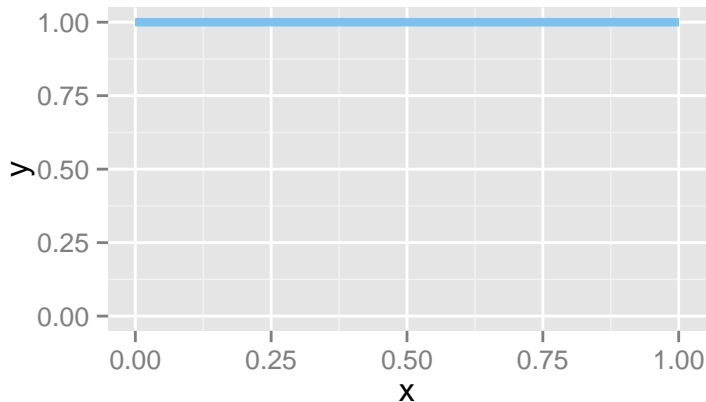
$$\hat{f} \in \arg \min_f \hat{R}_n(f),$$

where the minimum is taken over all functions  $f : \mathcal{X} \rightarrow \mathcal{A}$ .

- In an ideal world we'd want to find the risk minimizer.
- Is the empirical risk minimizer close enough?
- In practice, we always only have a finite sample...

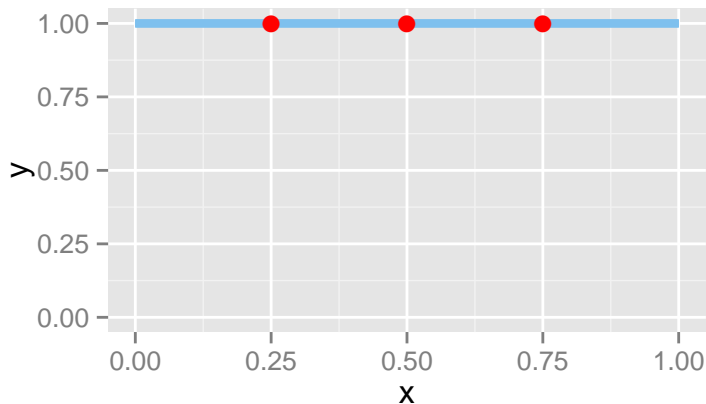
# Empirical Risk Minimization

- $P_{\mathcal{X}} = \text{Uniform}[0, 1]$ ,  $Y \equiv 1$  (i.e.  $Y$  is always 1).
- A plot of  $\mathcal{P}_{\mathcal{X} \times \mathcal{Y}}$ :



# Empirical Risk Minimization

$P_{\mathcal{X}} = \text{Uniform}[0, 1]$ ,  $Y \equiv 1$  (i.e.  $Y$  is always 1).

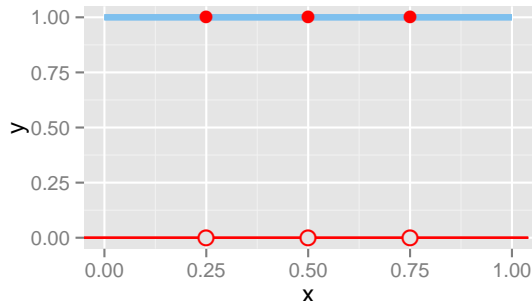


A sample of size 3 from  $\mathcal{P}_{\mathcal{X} \times \mathcal{Y}}$ .



# Empirical Risk Minimization

$P_{\mathcal{X}} = \text{Uniform}[0, 1]$ ,  $Y \equiv 1$  (i.e.  $Y$  is always 1).

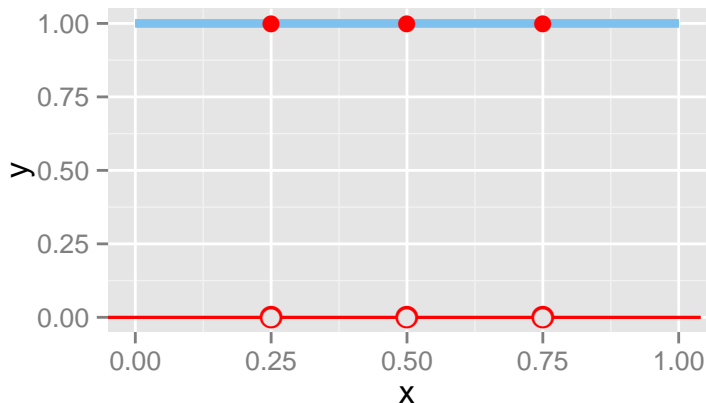


A proposed prediction function:

$$\hat{f}(x) = 1(x \in \{0.25, 0.5, 0.75\}) = \begin{cases} 1 & \text{if } x \in \{0.25, .5, .75\} \\ 0 & \text{otherwise} \end{cases}$$

# Empirical Risk Minimization

$P_{\mathcal{X}} = \text{Uniform}[0, 1]$ ,  $Y \equiv 1$  (i.e.  $Y$  is always 1).



Under either the square loss or the 0/1 loss,  $\hat{f}$  has Empirical Risk = 0 and Risk = 1.

# Empirical Risk Minimization

- In this case, ERM led to a function  $f$  that just **memorized** the data.
- How can we improve **generalization** from the training inputs to new inputs?
- We need to smooth things out somehow!
  - A lot of modeling is about spreading and extrapolating information from one part of the input space  $\mathcal{X}$  into unobserved parts of the space.
- One approach is **constrained ERM**:
  - Instead of minimizing empirical risk over *all* prediction functions,
  - We constrain our search to a particular subset of the space of functions, called a **hypothesis space**.

# Hypothesis Spaces

## Definition

A **hypothesis space**  $\mathcal{F}$  is a set of prediction functions  $\mathcal{X} \rightarrow \mathcal{Y}$  that we consider when applying ERM.

Desirable properties of a hypothesis space:

- Includes only those functions that have the desired “regularity”, e.g. smoothness, simplicity
- Easy to work with (e.g., we have efficient algorithms to find the best function within the space)

Most applied work is about designing good hypothesis spaces for specific tasks.

# Constrained Empirical Risk Minimization

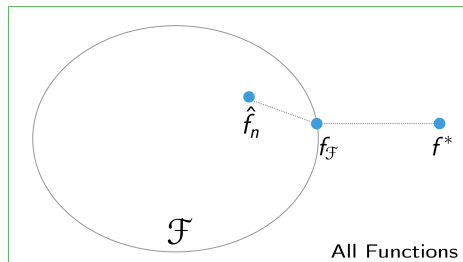
- Given a hypothesis space  $\mathcal{F}$ , a set of prediction functions mapping  $\mathcal{X} \rightarrow \mathcal{A}$ ,
- An **empirical risk minimizer** (ERM) in  $\mathcal{F}$  is a function  $\hat{f}_n$  such that

$$\hat{f}_n \in \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i).$$

- A **risk minimizer** in  $\mathcal{F}$  is a function  $f_{\mathcal{F}}^* \in \mathcal{F}$  such that

$$f_{\mathcal{F}}^* \in \arg \min_{f \in \mathcal{F}} \mathbb{E}[\ell(f(x), y)].$$

# Excess Risk Decomposition



$$f^* = \arg \min_f \mathbb{E} [\ell(f(x), y)]$$

$$f_{\mathcal{F}} = \arg \min_{f \in \mathcal{F}} \mathbb{E} [\ell(f(x), y)]$$

$$\hat{f}_n = \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i)$$

- Approximation error (of  $\mathcal{F}$ ) =  $R(f_{\mathcal{F}}) - R(f^*)$
- Estimation error (of  $\hat{f}_n$  in  $\mathcal{F}$ ) =  $R(\hat{f}_n) - R(f_{\mathcal{F}})$

# Excess Risk Decomposition for ERM

## Definition

The **excess risk** compares the risk of  $f$  to the Bayes optimal  $f^*$ :

$$\text{Excess Risk}(f) = R(f) - R(f^*)$$

- Can excess risk ever be negative?

The excess risk of the ERM  $\hat{f}_n$  can be decomposed:

$$\begin{aligned}\text{Excess Risk}(\hat{f}_n) &= R(\hat{f}_n) - R(f^*) \\ &= \underbrace{R(\hat{f}_n) - R(f_{\mathcal{F}})}_{\text{estimation error}} + \underbrace{R(f_{\mathcal{F}}) - R(f^*)}_{\text{approximation error}}.\end{aligned}$$

- There is a tradeoff between estimation error and approximation error

# Approximation Error

Approximation error  $R(f_{\mathcal{F}}) - R(f^*)$  is

- a property of the class  $\mathcal{F}$
- the penalty for restricting to  $\mathcal{F}$  (rather than considering all possible functions)

*Bigger  $\mathcal{F}$  mean smaller approximation error.*

Concept check: Is approximation error a random or non-random variable?



# Estimation Error

Estimation error  $R(\hat{f}_n) - R(f_{\mathcal{F}})$

- is the performance hit for choosing  $f$  using finite training data
- is the performance hit for minimizing empirical risk rather than true risk

With *smaller*  $\mathcal{F}$  we expect *smaller* estimation error.

*Under typical conditions:* “With infinite training data, estimation error goes to zero.”

Concept check: Is estimation error a random or non-random variable?

- What have we been glossing over by writing “argmin”?

- What have we been glossing over by writing “argmin”?
- In practice, we need a method to find  $\hat{f}_n \in \mathcal{F}$ : this can be very difficult!

- What have we been glossing over by writing “argmin”?
- In practice, we need a method to find  $\hat{f}_n \in \mathcal{F}$ : this can be very difficult!
- For nice choices of loss functions and classes  $\mathcal{F}$ , we can get arbitrarily close to the exact minimizer
  - But that takes time – is it always worth it?

- What have we been glossing over by writing “argmin”?
- In practice, we need a method to find  $\hat{f}_n \in \mathcal{F}$ : this can be very difficult!
- For nice choices of loss functions and classes  $\mathcal{F}$ , we can get arbitrarily close to the exact minimizer
  - But that takes time – is it always worth it?
- For some hypothesis spaces (e.g. neural networks), we don't know how to find  $\hat{f}_n \in \mathcal{F}$ .

# Optimization Error

- In practice, we don't find the ERM  $\hat{f}_n \in \mathcal{F}$ .
- We find  $\tilde{f}_n \in \mathcal{F}$  that we hope is good enough.
- **Optimization error:** If  $\tilde{f}_n$  is the function our optimization method returns, and  $\hat{f}_n$  is the empirical risk minimizer, then

$$\text{Optimization Error} = R(\tilde{f}_n) - R(\hat{f}_n).$$

# Error Decomposition in Practice

- Excess risk decomposition for function  $\tilde{f}_n$  returned by an optimization algorithm in practice:

$$\begin{aligned}\text{Excess Risk}(\tilde{f}_n) &= R(\tilde{f}_n) - R(f^*) \\ &= \underbrace{R(\tilde{f}_n) - R(\hat{f}_n)}_{\text{optimization error}} + \underbrace{R(\hat{f}_n) - R(f_{\mathcal{F}})}_{\text{estimation error}} + \underbrace{R(f_{\mathcal{F}}) - R(f^*)}_{\text{approximation error}}\end{aligned}$$

- It would be nice to observe the error decomposition for a practical  $\tilde{f}_n$ !
- How would we address each type of error?
- Why is this usually impossible?
- But we could construct an artificial example, where we know  $P_{\mathcal{X} \times \mathcal{Y}}$  and  $f^*$  and  $f_{\mathcal{F}} \dots$

- Given a loss function  $\ell : \mathcal{A} \times \mathcal{Y} \rightarrow \mathbb{R}$ ,



# ERM Overview

- Given a loss function  $\ell : \mathcal{A} \times \mathcal{Y} \rightarrow \mathbb{R}$ ,
- Choose a hypothesis space  $\mathcal{F}$ .

# ERM Overview

- Given a loss function  $\ell : \mathcal{A} \times \mathcal{Y} \rightarrow \mathbb{R}$ ,
- Choose a hypothesis space  $\mathcal{F}$ .
- Use an optimization method to find an empirical risk minimizer  $\hat{f}_n \in \mathcal{F}$ :

$$\hat{f}_n = \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i).$$

- Given a loss function  $\ell : \mathcal{A} \times \mathcal{Y} \rightarrow \mathbb{R}$ ,
- Choose a hypothesis space  $\mathcal{F}$ .
- Use an optimization method to find an empirical risk minimizer  $\hat{f}_n \in \mathcal{F}$ :

$$\hat{f}_n = \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i).$$

- (Or find a  $\tilde{f}_n$  that comes close to  $\hat{f}_n$ )

# ERM Overview

- Given a loss function  $\ell : \mathcal{A} \times \mathcal{Y} \rightarrow \mathbb{R}$ ,
- Choose a hypothesis space  $\mathcal{F}$ .
- Use an optimization method to find an empirical risk minimizer  $\hat{f}_n \in \mathcal{F}$ :

$$\hat{f}_n = \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i).$$

- (Or find a  $\tilde{f}_n$  that comes close to  $\hat{f}_n$ )
- The machine learning scientist's job:
  - Choose  $\mathcal{F}$  that balances approximation and estimation error.
  - As we get more training data, we can use a bigger  $\mathcal{F}$ .