02-750 Week 1
Automation of Scientific Research

Aidan Jan

January 20, 2026

Context

There are two types of automation.

1. Systems for performing repetitive tasks
    - E.g., milking cows, grinding and welding, liquid handling
    - These systems operate in highly **controlled environments**
    - For the most part, these machines do not need to solve problems (i.e., handle novel situations)
2. Systems that can 'think' and handle novel challenges
    - This is an emerging area of automation that leverages advances in **computing hardware, artificial intelligence**, and **machine learning**
    - e.g., IBM Watson, NewsBot, Amazon Robots, Self-driving Cars
    - These systems operate in largely **uncontrolled environments**
    - They **do** encounter novel situations and solve problems

The Scientific Method

- Hypothesis generation leads to experiment selection and execution through **predictions**
- Experiment selection and execution leads back to hypothesis generation through **observations**

We can automate this!

- Hypothesis generation can be done via **machine learning**
- Experiment selection can be done via **artificial intelligence**
- Experiment execution can be done via **automated instruments (i.e., robots)**

But what next? This course focuses on algorithms to perform the *Experiment Selection with AI* step

Why Should We Automate Scientific Research?

- To save money, time, and space
- To increase quality, safety, and reproducibility
- To manage the complexity of living systems (this objective requires the second kind of automation)

Biological Systems are Complex

For example, a hormone goes through many steps to generate a response. You can make a hypothesis on what the steps are. However, how do you test the hypothesis?

- How do you predict the system's behavior under untested conditions?
- How do you select the next experiment?
- How do you generate an alternative hypothesis, if this one is invalidated?

Questions about biological systems are very hard to answer, even for small systems. Things only get harder, if we have a really big system. We will study techniques that let us

- Turn observations into hypotheses
- Make accurate predictions
- Select informative experiment(s)

Automated Science for Drug Discovery

Drug discovery is expensive and takes a long time.

- 5k - 10k compounds are tested to produce one FDA-approved drug.
- Average cost of this one drug is $1.3 - 1.8 Billion.

The cost of drug discovery is driven by late-stage failures. The leading cause of failures is not a lack of efficacy, but side effects.

- Drugs may have unwanted, off-target interactions
- Thus, drug development is not just about finding compounds that hit a desired target; it is also about finding compounds that miss all other targets.

Active Learning

If we have unlimited money, we can measure each element of the (target, compound) matrix by running many experiments. However, we don't have unlimited money. **Active learning** will select a subset of elements that will help us learn a model of the entire matrix.

Our model is the (drug, target, response) matrix. Rows are drugs, columns are targets, and responses are the values of each cell. We can use active learning to select experiments by:

- Quantifying the uncertainty of the model's predictions for each unobserved combination
- Running the experiment corresponding to the least certain prediction

With only 2.5% of the matrix covered, scientists can identify 57% of the active compounds!

- Saves time and money over other methods, and improves the model faster.

Supervised Learning

Training data comes in pairs of inputs (x, y), where $x \in \mathbb{R}^d$ is the input instance and $y \in \mathfrak{C}$ its label. The entire training data is denoted as

$$D = \{(x_1, y_1), \cdots, (x_n, y_n)\} \subseteq \mathbb{R}^d \times \mathfrak{C}$$

where

- $\mathbb{R}^d$: d-dimensional feature space
- x_i: input vector of the i-th sample
- y_i: label of the i-th sample
- $\mathfrak{C}$: label space

Typical scenarios of the label space $\mathfrak{C}$

Scenario	Label Space
Binary Classification	$\mathfrak{C} = \{0, 1\}$ $\mathfrak{C} = \{+1, -1\}$
Multi-class Classification	$\mathfrak{C} = \{1, \cdots, k\}$ $(K \geq 2)$
Regression	$\mathfrak{C} = \mathbb{R}$

The goal is to find a function $h : \mathbb{R}^d \to \mathfrak{C}$, such that $h(x_i) \approx y_i \forall (x_i, y_i) \in D$. Ideally, our function would also work for points not in D, e.g., not overfitting the training data.

Machine Learning as Optimization

Machine learning algorithms use training data D to guide a **search** for the 'best' model in a particular hypothesis space, $\mathcal{H}$. Each algorithm specifies:

- The hypothesis space, $\mathcal{H}$
  - Ex. linear classifiers, decision trees, neural networks, policies, etc.
- A **loss function** (aka. cost function or error function) that can be used to evaluate each $h \in \mathcal{H}$
  - Ex. 0-1 loss, hinge loss, squared error, logistic loss, etc.
- A **risk objective** (aka. decision rule) that defines that 'best' means
  - Ex. Expected loss, minimax loss, maximum likelihood, etc.
- A **search method** to find the h that minimizes the risk
  - Ex. gradient descent

Offline versus Online Learning

- **Offline Machine Learning:** The learning algorithm receives all the training data at the same time, and then learns a model
- **Online Machine Learning:** The learning algorithm receives the data in a **sequenctial** fashion, and iteratively updates the model
  - Ex. for supervised learning: $D_i = D_{i-1} \cup \{(x_i, y_i)\}$
  - D_{i-1} are old data instances, and $\{x_i, y_i\}$ are new training instance(s).
- We will assume that the training data $D = \{(x_1, y_1), \cdots, (x_T, y_T)\}$ are independent and identically distributed (iid).
  - **Independent**: the occurrence of one instance $(x_i, y_i) \in D$, does not affect the probability of occurrence of any other instance $(x_j, y_j) \in D$
  - **Identically Distributed**: Each instance in D is sampled from the same (typically unknown) probability distribution.

Regret

Here, l will be the 0-1 loss function defined as

$$l(h, x, y) := \begin{cases} 1 & \text{if } h(x) \neq y \\ 0 & \text{if } h(x) = y \end{cases}$$

Basic framework for online machine learning

- The model makes a prediction on the new instance: $h_{i-1}(x_i) = \hat{y_i}$
- We compare $\hat{y_i}$ to the true label, y_i, and calculate the loss, l
- We update the model: h_i using $D_i = D_{i-1} \cup \{(x_i, y_i)\}$

In the online learning setting, we can ask the question: how well *could* I have done (in terms of total errors), if I had all the data at the beginning (i.e., as in offline learning)?

Informally, **regret** is the difference between the cumulative loss produced using online learning, versus the offline learning setting.

More formally, given a loss function l, **regret** is defined as

$$R^T = \sum_{t=1}^{T} l(h_t, x_t, y_t) - \sum_{t=1}^{T} l(h^*, x_t, y_t)$$

where T is the number of data points, h_t is the model that was used to make the prediction for instance x_t and

$$h^* = \arg\min_{h \in \mathcal{H}} \sum_{t=1}^{T} l(h_t, x_t, y_t)$$

Additionally, note that regret being zero does **not** imply that loss is zero.

No Regret Online Learning

A sequence of models is said to have **no regret** if $\lim_{T \to \infty} \frac{R^T}{T} = 0$.

- This will be true if $R^T \in o(T)$
  - Informally, a function f is $o(g)$ if function g grows much faster than f

Aside: Big-O Notation vs. Little-O Notation

$f \in O(g)$ means $\exists c > 0$ and $x_0 \in \mathbb{R}$ such that $|f(x)| \leq cg(x) \forall x \geq x_0$

- Ex. if $f \equiv 4x^3 - 2x^2$ and $g \equiv x^3$, then $f \in O(g)$ for $c = 6$ and $x_0 = 1$.

$f \in o(g)$ means $\forall \epsilon > 0 \exists N$ such that $|f(x)| \leq \epsilon g(x) \forall x \geq N$

- Ex. if $f \equiv 2x$ and $g \equiv x^2$, then $f \in o(g)$.

Three Simple Online Algorithms for Learning Classifiers

- A toy algorithm (Halving)
- Weighted Majority Algorithm
- Hedge Algorithm

Toy Online Learning Algorithm (Halving)

This is a toy algorithm, but it is useful to study because it is easy to understand and to analyze. Key assumptions:

- $\mathcal{H}$ is a **finite** set of classifiers of the form: $h : \mathbb{R}^d \to \mathfrak{C}$
- $\mathcal{H}$ contains a perfect model h^*
  - Note that $\mathcal{L}(h^*, x, y) = \frac{1}{T}\sum_{t=1}^{T} l(h^*, x_t, y_t) = 0$ (i.e., perfect model doesn't make any mistakes)
  - Of course, we don't know *which* model is perfect, *a priori*
- The goal is to find $h^* \in \mathcal{H}$ in an online fashion

Basic Strategy:

- Assign a binary weight, $w_i \in \{0, 1\}$, to each model $h_i \in \mathcal{H}$
  - Each weight is initialized to 1, thus, $\sum_{h_i \in \mathcal{H}} w_i = |\mathcal{H}|$
  - A weight of 1 means the model **might** be perfect
- For each new training example, (x_t, y_t)
  - Output the **majority label** amongst the models with **weight 1**.
    - The majority might be right, or it might be wrong!
    - We will think of the majority label as having come from a virtual model $\hat{h}_t$
  - Update the weights as follows:
    - Set w_i to 0 for any model such that $h_i(x_t) \neq y_t$

For each new training instance, (x_t, y_t)

1. Output the majority label rom model $\hat{h}_t$

$$\hat{y} = \arg\max_{c \in \mathfrak{C}} \sum_{h_i \in \mathcal{H}} w_i \cdot \mathbb{1}(h_i(x_t) = c)$$

2. Re-weight each model in $\mathcal{H}$ using the following rule:

$$w_i(t) = \min(w_i(t-1), \mathbb{1}(h_i(x_t) = y_t))$$

Notice that at any given time t, the sum over all $W_t = \sum_{h_i \in \mathcal{H}} w_i(t)$ is a measure of the number of **potentially perfect** models in $\mathcal{H}$ (i.e., those with weight of 1). This number decreases monotonically over time.

Halving Example

Suppose we have a collection of models for predicting whether two molecules bind

Models			Weights			Majority Label	True Label
h_1	h_2	h_3	w_1	w_2	w_3	$\hat{h}_i(x_t) = \hat{y}$	y_t
1	0	1	1	1	1	1	1
0	1	1	1	0	1	Tie	1
1	0	1	0	0	1	1	1
0	1	1	0	0	1	1	1
1	0	1	0	0	1	1	1

Each iteration we only fill one row, and the weights at each iteration are determined by the results of the previous rows. Since we know there is a perfect model, we set a weight to zero once a model is wrong (e.g., we know that one must not be the perfect model).

If we run a regret analysis:

- We assume that there is a perfect model (i.e., zero loss), so regret is

$$R^T = \sum_{t=1}^{T} l(\hat{h_t}, x_t, y_t) - \sum_{t=1}^{T} l(h^*, x_t, y_t) = \sum_{t=1}^{T} l(\hat{h_t}, x_t, y_t)$$

  - the second sum goes to zero
  - Notice that once the weights of all the imperfect models have been set to zero, the majority will always be right for any subsequent input.
- Thus, the regret for this simple algorithm corresponds to the number of mistakes made by $\hat{h_1}, \cdots, \hat{h_t}$
- Let's compute an upper bound on the number of mistakes...