CS4/5780 — Lecture 2

Supervised Learning

Spring 2022

CS4/5780 — Lecture 2

Machine Learning Intro Continued

Spring 2022

Outlook

“**A breakthrough in machine learning would be worth ten Microsofts.**” (Bill Gates, Microsoft)

“It will be the basis and fundamentals of every successful huge IPO win in 5 years.” (Eric Schmidt, Google / Alphabet)

“AI and machine learning are going to change the world and we really have not begun to scratch the surface.” (Jennifer Chayes, UC Berkeley)

“ML is transforming sector after sector of the economy, and the rate of progress only seems to be accelerating.” (Daphne Koller, Stanford / Coursera/ Insitro)

“Machine learning is the next Internet” (Tony Tether, DARPA)


DANGER

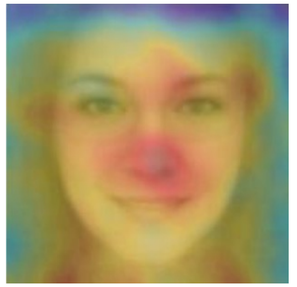
Data privacy / misuse

Learning models leak training data (Fredrickson et al. ‘15)

Leaked data

Real image

Learning algorithms detect sexual orientation better than people (Wang & Kosinski’17)



Formal definitions of data privacy:

- **K- anonymity (Sweeney)**
- **Differential Privacy (Dwork, McSherry, Nissim, Smith).**

Latanya Sweeney

Cynthia Dwork

Frank McSherry

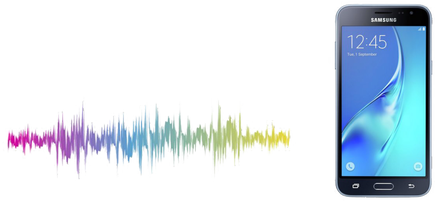
Kobbi Nissim

Adam Smith

Robust and Secure ML

Image Recognition
Misreading traffic signs
(Eykholt et al)

Speech recognition
Hide commands in noise (Carlini & Wagner)



Poisoning Attacks
Tay (chat bot) became inflammatory in 16 hr.

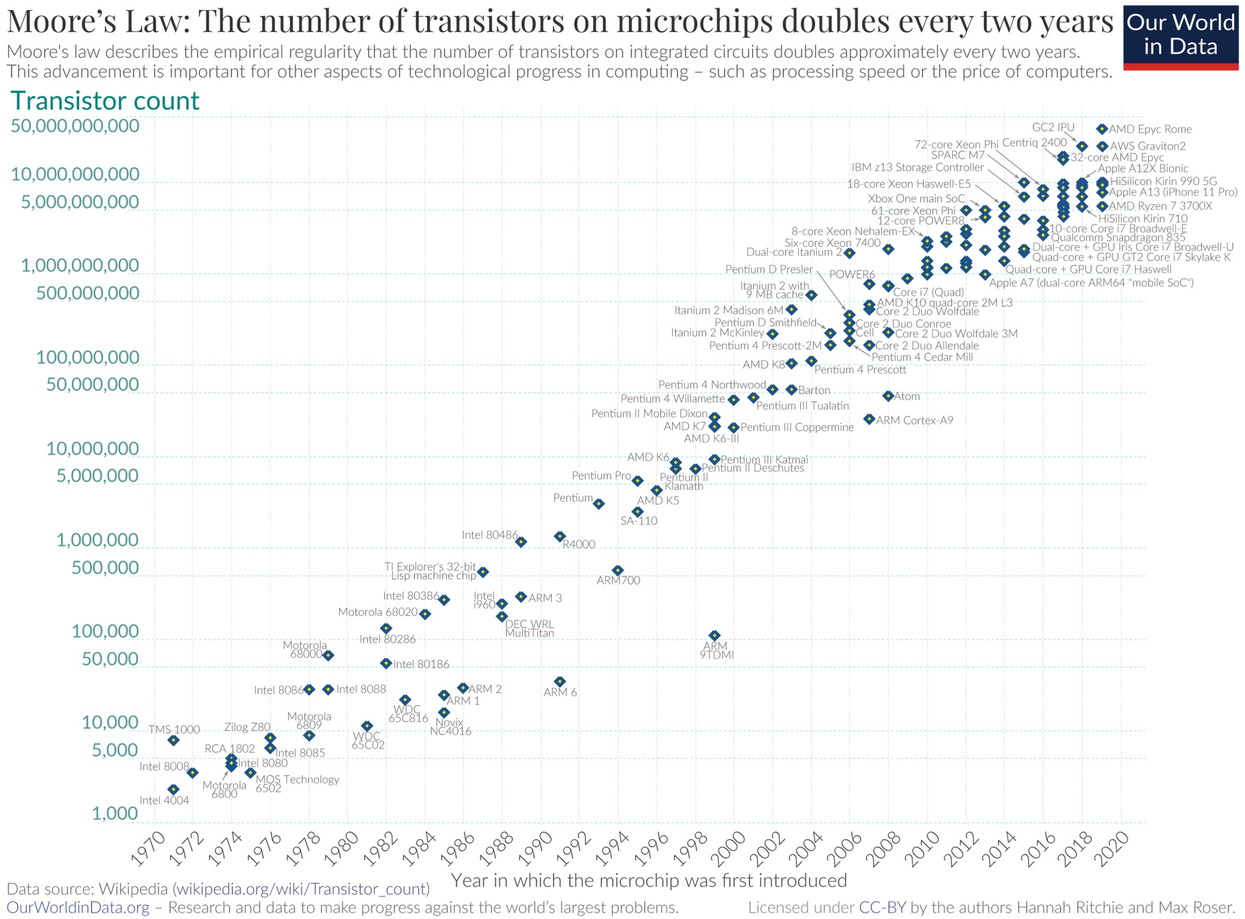
How to create robust and secure machine learning algorithms?

Learning and the Society

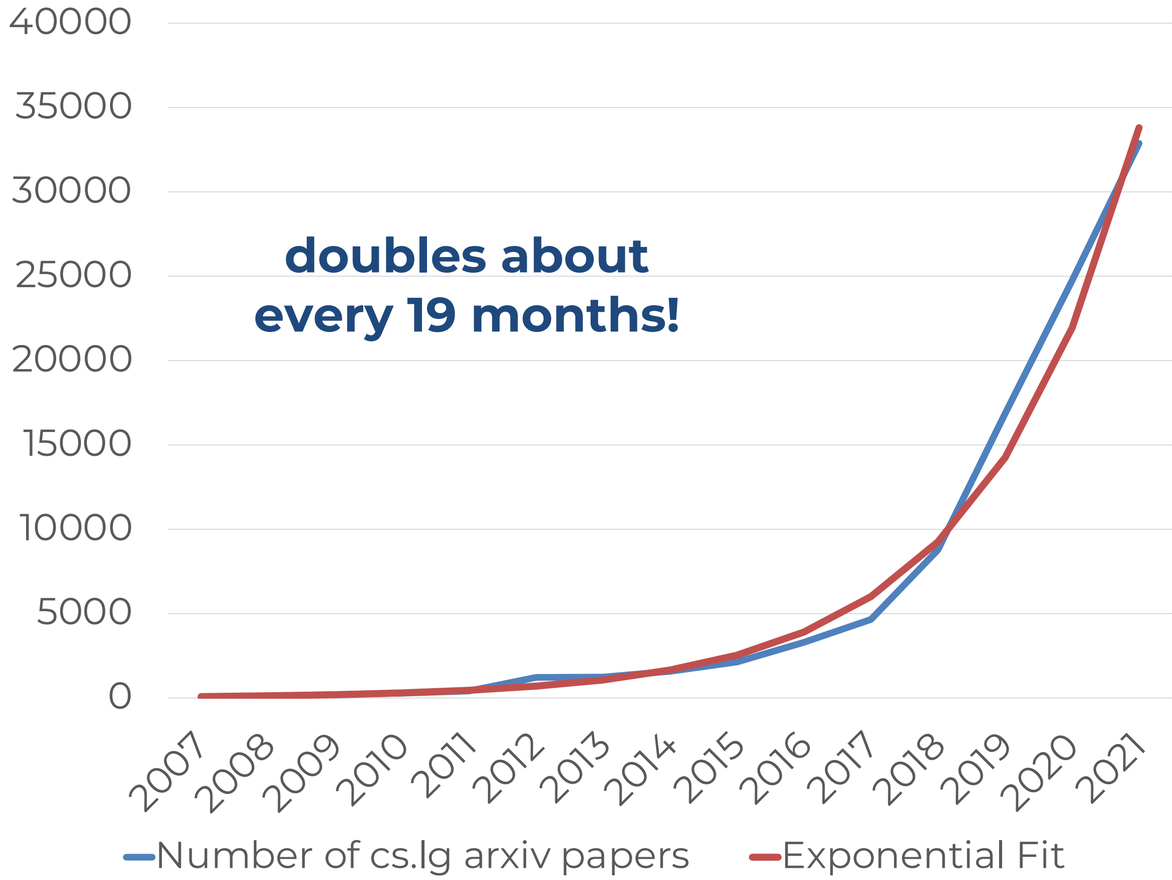
- Bad dynamics perpetuate and worsen stereotypes and biases.
- Who carries the burden of bad prediction?
- How to design good dynamics?

Will AI take over the world?

The Original Moore's Law

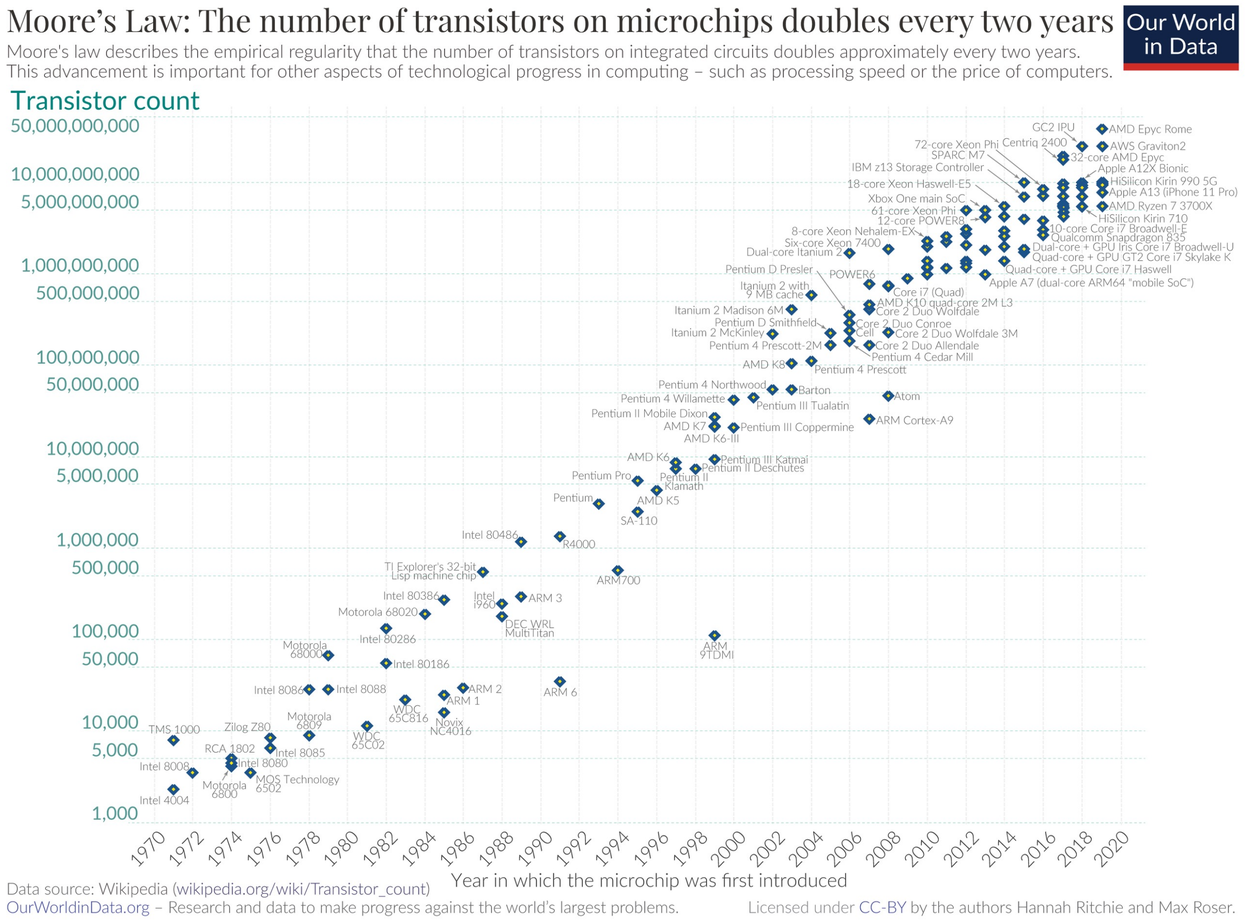


CS Machine Learning Papers
The New Moore's Law?

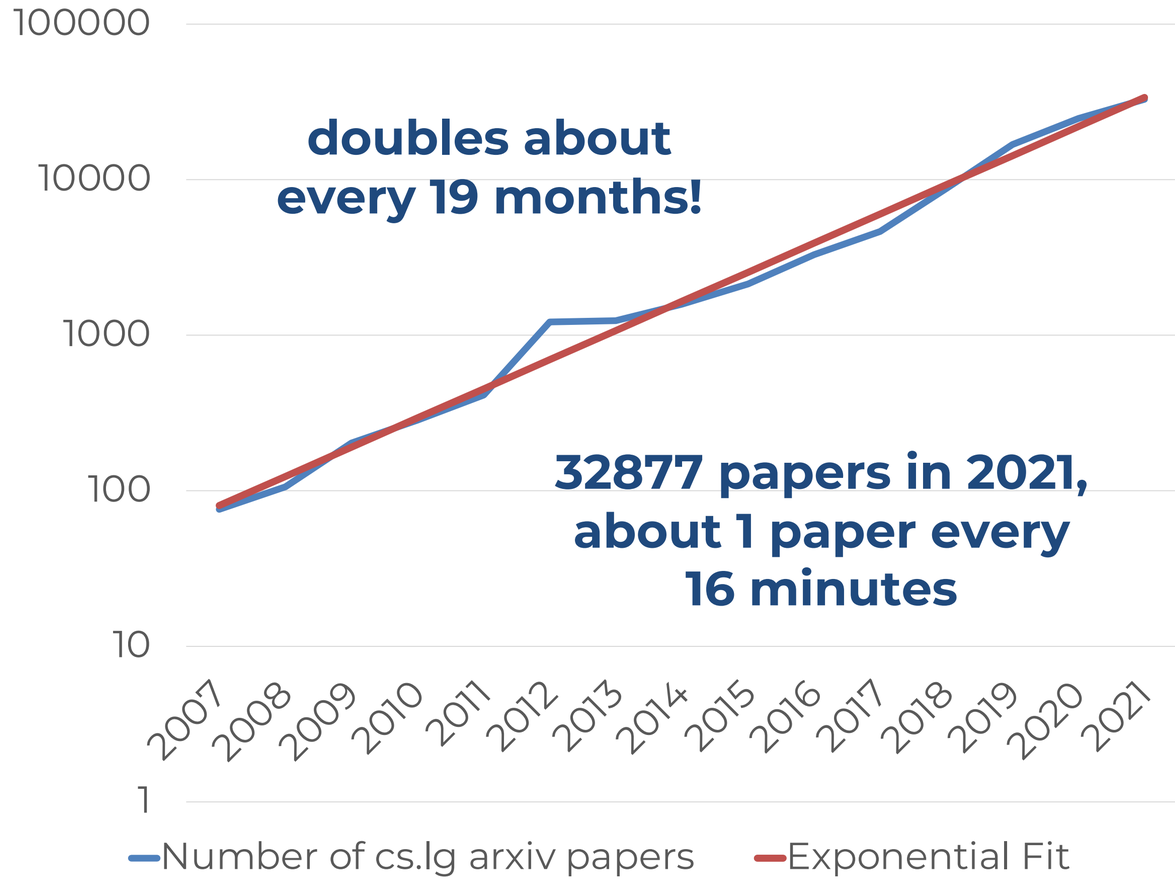


Will AI take over the world?

The Original Moore's Law



CS Machine Learning Papers
The New Moore's Law?



Will AI take over the world?

Good news:

AI is nowhere near to general Intelligence (no real progress)

Bad news:

AI doesn't have to be smarter than us to be harmful

CS4/5780 — Lecture 2

Supervised Learning

Spring 2022

What types of ML are there?

As far as this course is concerned:

- **supervised learning:** given labeled examples, find the right prediction of an unlabeled example.
  - e.g. *Given annotated images learn to detect faces.*
- **unsupervised learning:** given data try to discover similar patterns, structure, sub-spaces
  - e.g. *automatically cluster news articles by topic*
- **reinforcement learning:** learn from delayed feedback
  - e.g. *robot learns to walk, fly, play chess*

One way to think of it:

Learn a function from input-output pairs

```
In [1]: def f(n):
   ...:
   ...:
```

unknown

DATA

```
In [2]: f(1)
Out[2]: 1.0

In [3]: f(2)
Out[3]: 2.0

In [4]: f(3)
Out[4]: 4.0

In [5]: f(4)
Out[5]: 8.0
```

I've written a function in python, and here are some of its outputs for some inputs.

Can we make a **hypothesis** as to what the function might be?

$$h(x) = 2^{x-1}$$

We can't expect to learn the function perfectly all the time!

Another way to think of it:

Make predictions from data

- **Dataset** of emails from some source

- Each email was **labeled** at “spam” or “not spam”
  - Perhaps by human labelers

- We want to learn a function that can **classify future emails** from the same source as either “spam” or “not spam” — matching what human labelers would do

Formalizing Supervised Learning

Input instances (a.k.a. examples) x in some set $\mathcal{X}$.

- E.g. for spam classification, $\mathcal{X}$ could be the set of all strings.

Output labels y in some set $\mathcal{Y}$.

- E.g. for spam classification, $\mathcal{Y} = \{\text{spam}, \text{notspam}\}$.

Training dataset $\mathcal{D}$ consists of input-label pairs (x, y)

- $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$.

Formalizing Supervised Learning: The Way We Typically Do It

Input instances (a.k.a. examples) $x \in \mathcal{X} = \mathbb{R}^d$.

- E.g. before doing spam classification, we'd first map (embed) each email into a representation in some vector space $\mathbb{R}^d$.
- It's this vector representation we use for learning.

Output labels $y \in \mathcal{Y}$; usually $\mathcal{Y} \subseteq \mathbb{Z}$ or $\mathcal{Y} \subseteq \mathbb{R}$

- E.g. for spam classification, $\mathcal{Y} = \{-1, +1\}$.
- We let $+1$ mean spam and

Training dataset $\mathcal{D}$ consists of input-label pairs (x, y)

- $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \subset \mathbb{R}^d \times \mathcal{Y}$.

Input examples = Feature Vectors

- Let's look at a simple feature vector.
  - Say we want to predict house prices.

$$x = \begin{bmatrix} \text{square feet} \\ \text{number of bedrooms} \\ \text{number of bathroooms} \\ \text{construction date} \end{bmatrix} = \begin{bmatrix} 2000 \\ 3 \\ 2 \\ 2005 \end{bmatrix} \in \mathbb{R}^4.$$

Input examples = Feature Vectors

- Let's look at a simple feature vector.
  - Say we want to predict house prices.

$$x = \begin{bmatrix} \text{square feet} \\ \text{number of bedrooms} \\ \text{number of bathroooms} \\ \text{construction date} \\ \text{zip code?} \end{bmatrix} = \begin{bmatrix} 2000 \\ 3 \\ 2 \\ 2005 \\ 14853 \end{bmatrix} \in \mathbb{R}^5.$$

Input examples = Feature Vectors

- Let's look at a simple feature vector.
  - Say we want to predict house prices.

$$x = \begin{bmatrix} \text{square feet} \\ \text{number of bedrooms} \\ \text{number of bathroooms} \\ \text{construction date} \\ \text{zip code} = 00000? \\ \vdots \\ \text{zip code} = 14853? \\ \vdots \\ \text{zip code} = 99999? \end{bmatrix} = \begin{bmatrix} 2000 \\ 3 \\ 2 \\ 2005 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{100004}.$$

More Types of Feature Vectors

Text document in **bag-of-words format**

- Vector dimension d is the number of words in the dictionary.
- The ith feature $x^{(i)}$ is the number of times the ith word in the dictionary appears in the document.
- Feature vectors x are large (typically $d \approx 10^6$) and very **sparse**.

Images

- For a color image, $d = 3 \times W \times H$, where W is the width and H is the height of the image in pixels. (3 comes from RGB color.)
- Each feature $x^{(i)}$ represents the values of a particular color channel (red, green, or blue) for a particular pixel.
- Feature vectors x are large and dense.

Output labels

Binary classification. Typically $\mathcal{Y} = \{-1, 1\}$ or $\mathcal{Y} = \{0, 1\}$.

- E.g. spam filtering: an email is spam or it isn't.

Multi-class classification. Typically $\mathcal{Y} = \{1, \ldots, K\}$ or $\mathcal{Y} = \{0, \ldots, K-1\}$

- Here $K \geq 2$ is the number of classes.
- E.g. image classification. An image can be a cat, or a dog, etc.

Regression. Typically $\mathcal{Y} = \mathbb{R}$

- E.g. predict the sale price of a house.

Formalizing Supervised Learning: Where does the data come from?

Input instances (a.k.a. examples) $x \in \mathcal{X} = \mathbb{R}^d$.

Output labels $y \in \mathcal{Y}$; usually $\mathcal{Y} \subseteq \mathbb{Z}$ or $\mathcal{Y} \subseteq \mathbb{R}$

Training dataset $\mathcal{D}$ consists of input-label pairs (x, y)

Data (x_i, y_i) are drawn from some unknown **distribution** $\mathcal{P}$ over $\mathcal{X} \times \mathcal{Y}$.

- This is a joint distribution over examples and labels.
- We usually suppose the data are drawn **independently**, i.e. (x_i, y_i) and (x_j, y_j) are independent for $i \neq j$.

Formalizing Supervised Learning: What is the goal?

Input instances (a.k.a. examples) $x \in \mathcal{X} = \mathbb{R}^d$.

Output labels $y \in \mathcal{Y}$; usually $\mathcal{Y} \subseteq \mathbb{Z}$ or $\mathcal{Y} \subseteq \mathbb{R}$

Training dataset $\mathcal{D}$ consists of input-label pairs (x, y)

Data (x_i, y_i) are drawn from some unknown **distribution** $\mathcal{P}$ over $\mathcal{X} \times \mathcal{Y}$.

Goal is to pick a **hypothesis** $h : \mathcal{X} \to \mathcal{Y}$ such that for a freshly drawn data sample (x, y) from $\mathcal{P}$, the predicted label $\hat{y} = h(x)$ will have $\hat{y} = y$ (or, sometimes, $\hat{y} \approx y$) with high probability.

- I.e. we learn a program that "gets it right" most of the time.

Formalizing Supervised Learning: Hypothesis Classes

Input instances (a.k.a. examples) $x \in \mathcal{X} = \mathbb{R}^d$.

Output labels $y \in \mathcal{Y}$; usually $\mathcal{Y} \subseteq \mathbb{Z}$ or $\mathcal{Y} \subseteq \mathbb{R}$

Training dataset $\mathcal{D}$ consists of input-label pairs (x, y)

Data (x_i, y_i) are drawn from some unknown **distribution** $\mathcal{P}$ over $\mathcal{X} \times \mathcal{Y}$.

Goal is to pick a **hypothesis** $h : \mathcal{X} \to \mathcal{Y}$ such that for $(x, y) \sim \mathcal{P}$, we have $h(x) = y$ with high probability.

The hypothesis h is restricted to lie in a **hypothesis class** $\mathcal{H}$, i.e. $h \in \mathcal{H}$.

- Before we can learn h, we must specify what type of function we are looking for—don't want to have to consider *all* functions.
- E.g. for a linear classifier, $\mathcal{H}$ is the set of linear functions.

Formalizing Supervised Learning: Learning Algorithms

Input instances (a.k.a. examples) $x \in \mathcal{X} = \mathbb{R}^d$.

Output labels $y \in \mathcal{Y}$; usually $\mathcal{Y} \subseteq \mathbb{Z}$ or $\mathcal{Y} \subseteq \mathbb{R}$

Training dataset $\mathcal{D}$ consists of input-label pairs (x, y)

Data (x_i, y_i) are drawn from some unknown **distribution** $\mathcal{P}$ over $\mathcal{X} \times \mathcal{Y}$.

Goal is to pick a **hypothesis** $h : \mathcal{X} \to \mathcal{Y}$ such that for $(x, y) \sim \mathcal{P}$, we have $h(x) = y$ with high probability.

The hypothesis h is restricted to lie in a **hypothesis class** $\mathcal{H}$, i.e. $h \in \mathcal{H}$.

Then: a **supervised learning algorithm** is an algorithm that given a dataset $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$ outputs a hypothesis $h \in \mathcal{H}$.

How most learning is done:
Learning Via Optimization

1. Set up a supervised learning setting
   - Collect a **dataset**
   - Define a **hypothesis class**

2. Define some notion of “how good” a particular hypothesis is on the dataset
   - Lets us compare two hypotheses to see which is better
   - Called a **loss function** (or risk function)

3. Use optimization to select the ”best” hypothesis
   - The one that minimizes the loss

Common Loss Functions in ML

0/1 Loss

- The fraction of examples in the training set the **hypothesis gets wrong**
  - a.k.a. the **error rate**

$$\mathcal{L}_{0/1}(h; \{(x_1, y_1), \ldots, (x_n, y_n)\}) = \frac{1}{n} \sum_{i=1}^{n} \delta_{h(\mathbf{x}_i) \neq y_i}$$

$$\delta_{h(\mathbf{x}_i) \neq y_i} = \begin{cases} 1 & \text{if } h(\mathbf{x}_i) \neq y_i \\ 0 & \text{otherwise} \end{cases}$$

Square Loss (a.k.a. the L2 loss)

- Usually used **for regression**
  - **Sum of the squares of the error**

$$\mathcal{L}_{\mathsf{sq}}(h; \{(x_1, y_1), \ldots, (x_n, y_n)\}) = \frac{1}{n} \sum_{i=1}^{n} (h(\mathbf{x}_i) - y_i)^2$$

Absolute Loss (a.k.a. the L1 loss)

- Usually used **for regression**
  - **Sum of the absolute values of the error**
  - Helps to deal with outliers

$$\mathcal{L}_{\text{abs}}(h; \{(x_1, y_1), \ldots, (x_n, y_n)\}) = \frac{1}{n} \sum_{i=1}^{n} |h(\mathbf{x}_i) - y_i|$$

Memorization: A Silly Learning Algorithm

Given a dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, output the hypothesis

$$h(x) = \begin{cases} y_i & \text{if } x = x_i \text{ for some } i \in \{1, \ldots, n\} \\ y_1 & \text{otherwise} \end{cases}$$

What is the loss of this algorithm?

Zero loss! But should we expect this algorithm to perform well on freshly drawn examples from the source distribution?

Generalization

We don't just care about the loss on the training dataset.

What we want is low loss on **fresh examples drawn from the same source** distribution $\mathcal{P}$.

That is, what we really want to be small is

$$\mathbf{E}_{(x,y)\sim\mathcal{P}}\left[\mathcal{L}(h;(x,y))\right].$$

But we can't minimize this because **we don't have access to** $\mathcal{P}$.

- We only have access to the dataset drawn from $\mathcal{P}$.

Train/Test Split

- We can't directly compute the expected loss over our distribution, but we can **estimate** it

- Idea: split the dataset into two parts
  - A training dataset
  - A test dataset

- Use **only the training dataset** to select the hypothesis

- Then we can estimate the expected loss over *P* as

$$\mathbf{E}_{(x,y)\sim\mathcal{P}}\left[\mathcal{L}(h;(x,y))\right] \approx \mathcal{L}(h;\mathcal{D}_{\text{test}})$$

Breakout: Why does this work?

- Examples in our dataset are statistically **independent**
  - Consequently, training set is independent of test set
  - So, the hypothesis is independent of the test set

- The loss on the test set is an **unbiased estimator** of the expected loss over the source distribution

$$\mathbf{E}_{\mathcal{D}_{\text{test}} \text{ sampled from } \mathcal{P}} \left[\mathcal{L}(h; \mathcal{D}_{\text{test}})\right] = \mathbf{E}_{(x,y)\sim\mathcal{P}} \left[\mathcal{L}(h; (x, y))\right]$$

- Will be a good estimator for large test sets

Train/Validation/Test Split

- In practice we use a three-way split
  - Train set
  - **Validation set**
  - Test set

- The validation set is **used to revise the hypothesis**
  - If validation loss is too high, we can change our assumptions/settings and try training/validation again.
  - **But we only test once!** (Otherwise we lose independence.)

- Usually 80%/10%/10% train/validation/test is good

How to Split the Data

- The test set must simulate a real test scenario
  - It should simulate what you will encounter in real life.

- E.g. for an email spam filter, you train a system on past data to predict if future email is spam.
  - Must split train / test temporally
  - Only try to predict the future from the past.

- If the task is not temporal, it is a good default approach to **split uniformly at random**.

A Complete* Recipe for Supervised Learning

1. Assemble a **dataset**
2. Choose a **hypothesis class** and a **loss function.**
3. **Split** the dataset into train/validation/test sets.
4. Find* the hypothesis h that **minimizes the loss on the training set**
5. Evaluate h on the **validation set**
   - loss too high, retry at 4 with different assumptions/settings
6. Output h, and evaluate it on the **test set**
   - we should only use the test set here — not to choose h

No Free Lunch Theorem

- There is **no single supervised learning algorithm that will work across all distributions**.
  - No algorithm outperforms any other on all tasks or even on “most” tasks.

- Consequence: **you must make assumptions in order to learn**
  - Which learning algorithm you choose will depend on your assumptions.

ARTICLE Communicated by Steven Nowlan

The Lack of A Priori Distinctions Between Learning Algorithms

David H. Wolpert
The Santa Fe Institute, 1399 Hyde Park Rd., Santa Fe, NM, 87501, USA

This is the first of two papers that use off-training set (OTS) error to investigate the assumption-free relationship between learning algorithms. This first paper discusses the senses in which there are no a priori distinctions between learning algorithms. (The second paper discusses the senses in which there are such distinctions.) In this first paper it is shown, loosely speaking, that for any two algorithms A and B, there are “as many” targets (or priors over targets) for which A has lower expected OTS error than B as vice versa, for loss functions like zero-one loss. In particular, this is true if A is cross-validation and B is “anti-cross-validation” (choose the learning algorithm with largest cross-validation error). This paper ends with a discussion of the implications of these results for computational learning theory. It is shown that one *cannot* say: if empirical misclassification rate is low, the Vapnik–Chervonenkis dimension of your generalizer is small, and the training set is large, then with high probability your OTS error is small. Other implications for “membership queries” algorithms and “punting” algorithms are also discussed.