Machine Learning Fundamentals

N. Rich Nguyen, PhD

SYS 6016

Learning Outcomes

1. Be familiar with some basic ML vocabulary and fundamental concepts
2. Take a look at the optimization procedure of a simple learning algorithm
3. Make some connection to the field of statistical analysis
4. Understand model's ability for generalization

1. Machine Learning Overview

Machine Learning (ML) Definition

"Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed."

-- Arthur Samuel, Gaming and AI Pioneer, 1959

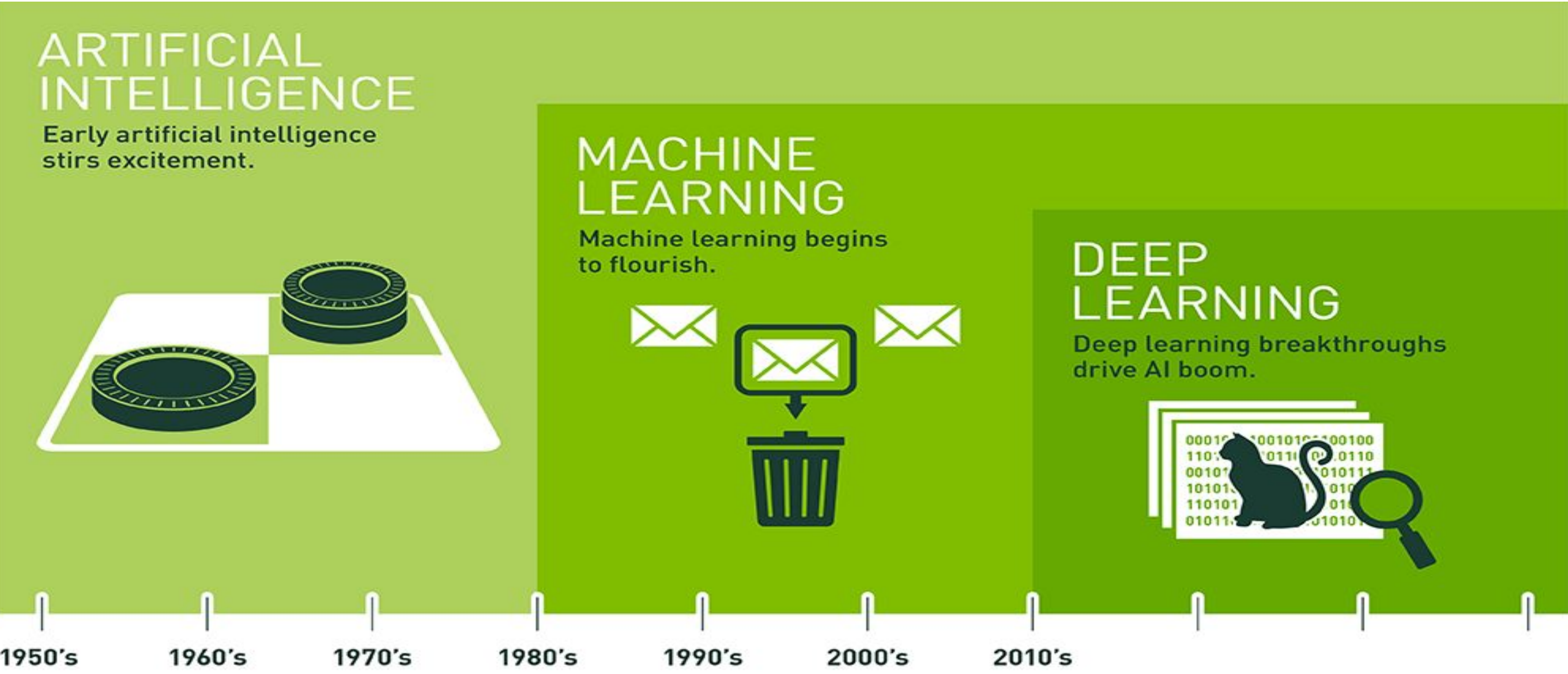
Traditional Programing

Data → Computer → Answer
Program →

Machine Learning

Data → Computer → Program
Answer →

ML Timeline

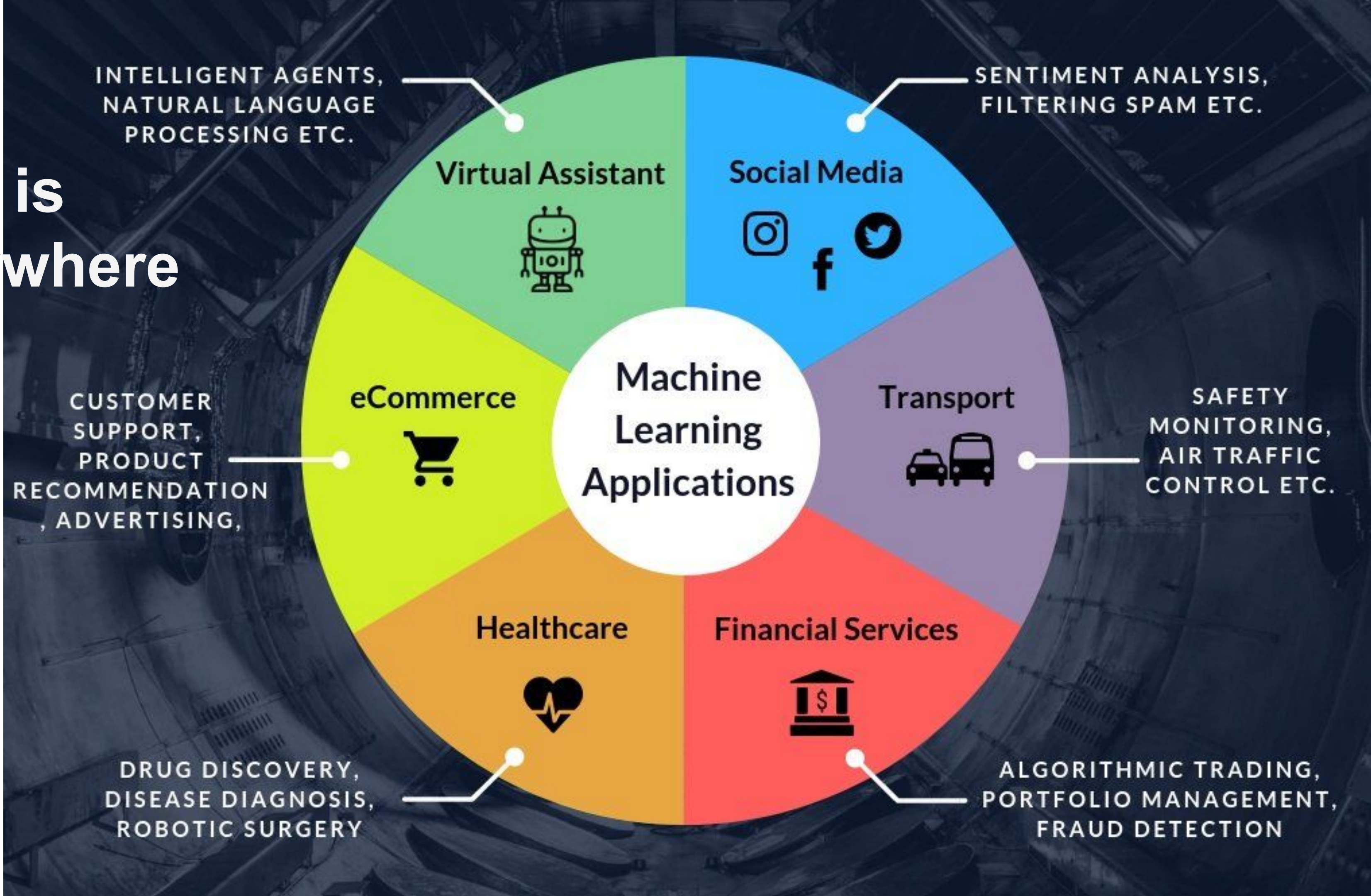


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Can you think of an AI/ML/DL application you find most interesting?

It could be on the news, magazines, social media, or you might have learned it from a friend.

ML is everywhere



Why do you study ML?

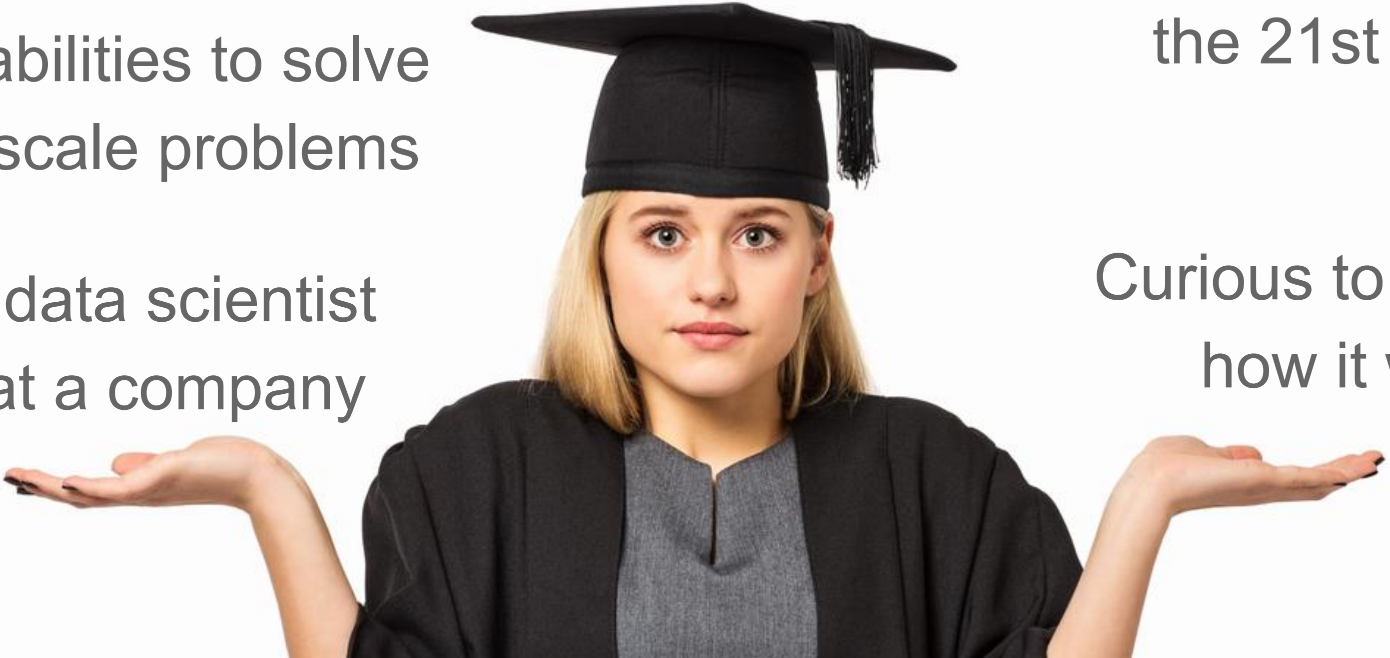
[your reason here]

It's one of the best careers for the 21st century

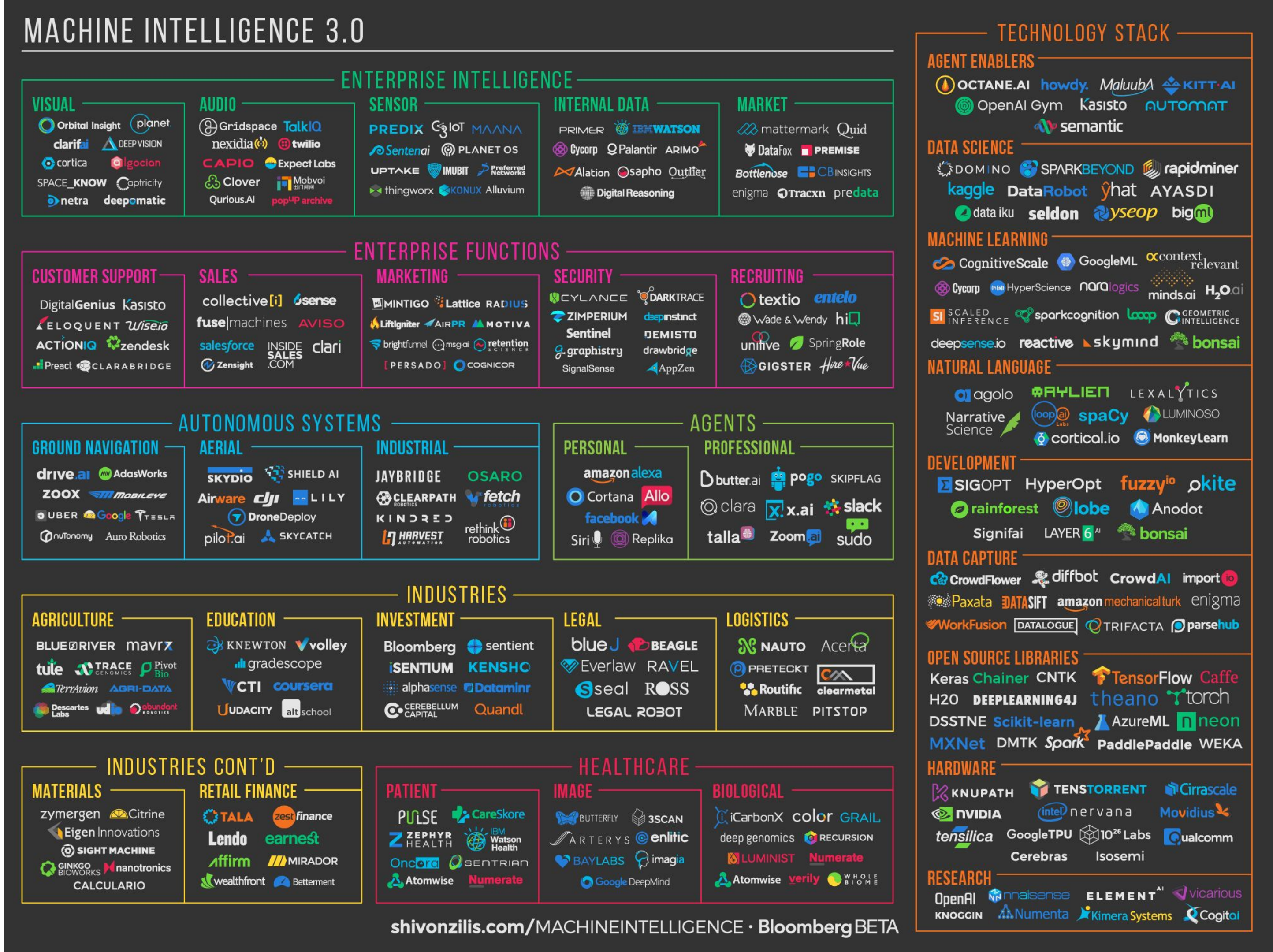
Gain abilities to solve large-scale problems

Obtain a data scientist position at a company

Curious to know how it works



ML Landscape



Learning Algorithms

"A computer program is said to **learn** from *experience* ***E*** with respect to some *task* ***T*** and some *performance* ***P***, if its performance on ***T***, as measured by ***P***, improves with ***E***."

-- Tom Mitchell, Computer Scientist, 1997

Learning Algorithms: The Task, T

Not the process of learning itself, but learning is the means to perform the task (i.e. If we want a robot to be able to walk, then walking is the task).

Described in terms of how the ML systems should process an **example**, which is a collection of **features** represented by $\mathbf{x} \in \mathbb{R}^n$

Many kinds of tasks can be solved with machine learning:

- Regression
- Classification
- Transcription
- Machine Translation
- Anomaly Detection
- Synthesis and Sampling
- Density Estimation

Learning Algorithms: The Performance, P

Quantitative measures to evaluate the abilities of a learning algorithm

- Classification or transcription tasks → **accuracy** or error rate (expected 0-1 loss)
- Regression tasks → **mean square error**
- Density Estimation tasks → **average log-probability**

Interested in how well the learning algorithm performs on data that it has not seen before (**test set**)

Learning Algorithms: The Experience, E

- Most learning algorithms are allowed to experience a **dataset**, a collection of many **examples** or **data points**.
- Depends on what kind of experience learning algorithms are allowed to have during the learning process, they can be broadly categorized as **unsupervised** or **supervised learning**.
- Some learning algorithms do not just experience a fixed dataset, instead they interact with an environment, so there is a feedback loop between the learning systems and their experience. Such algorithms are called **reinforcement learning.**

Classes of Learning Algorithms

Supervised Learning:

Data: (**x**, y)

x is data, y is label

Goal: Learn function to map **x** → y

Example:

This thing is an apple.

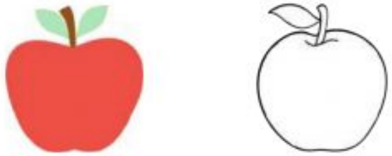
Unsupervised Learning:

Data: **x**

x is data, no labels!

Goal: Learn underlying structure

Example:



This thing is like the other thing.

Reinforcement Learning:

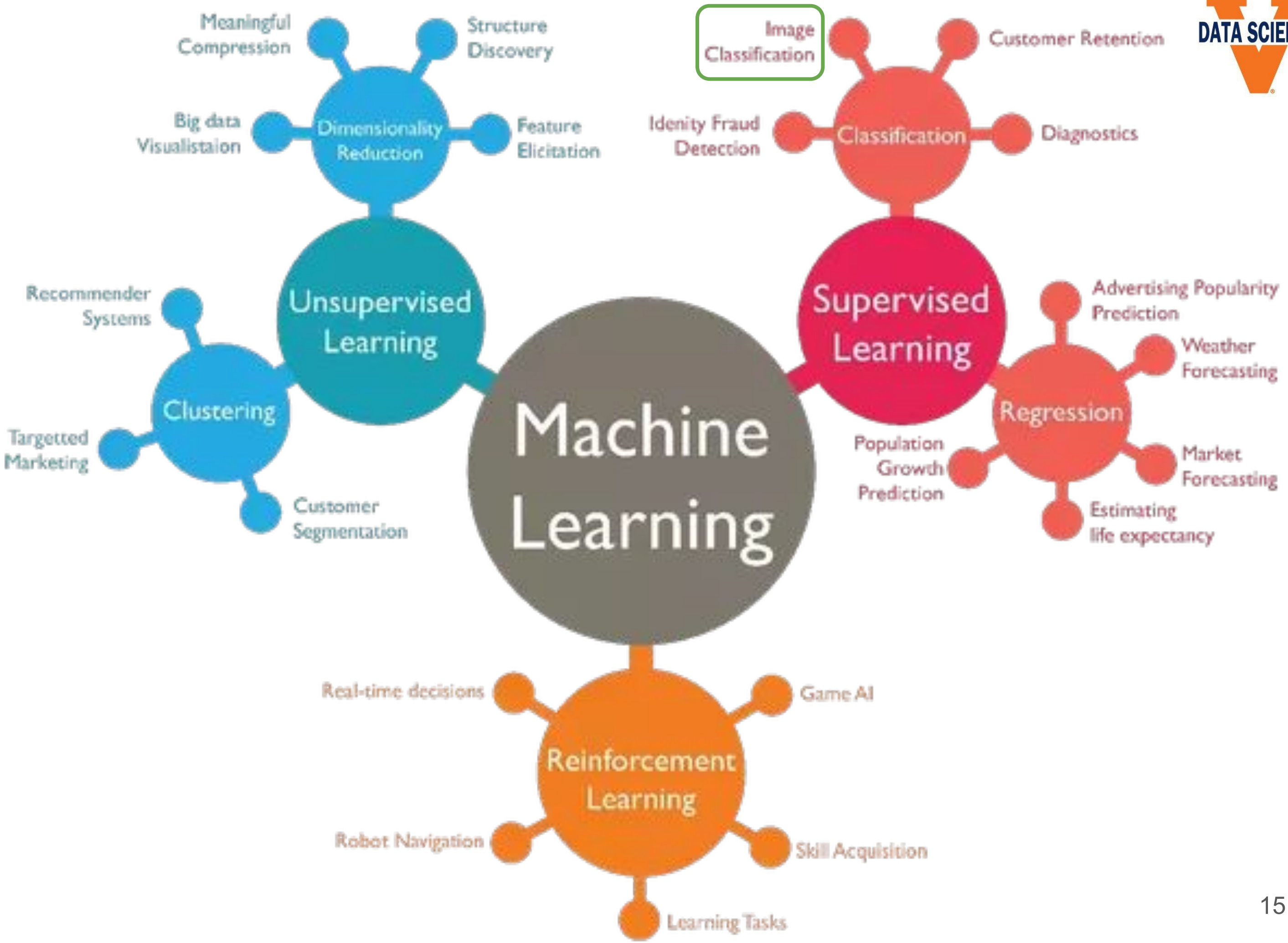
Data: state-action pairs

Goal: Maximize future rewards over many steps

Example:

Eat this thing because it will keep you alive.

ML Taxonomy


Meaningful Compression
Structure Discovery
Big data Visualistaion
Dimensionality Reduction
Feature Elicitation
Recommender Systems
Unsupervised Learning
Clustering
Targetted Marketing
Customer Segmentation
Image Classification
Customer Retention
Idenity Fraud Detection
Classification
Diagnostics
Supervised Learning
Advertising Popularity Prediction
Weather Forecasting
Regression
Population Growth Prediction
Market Forecasting
Estimating life expectancy
Machine Learning
Real-time decisions
Game AI
Reinforcement Learning
Robot Navigation
Skill Acquisition
Learning Tasks

2. Building a learning algorithm

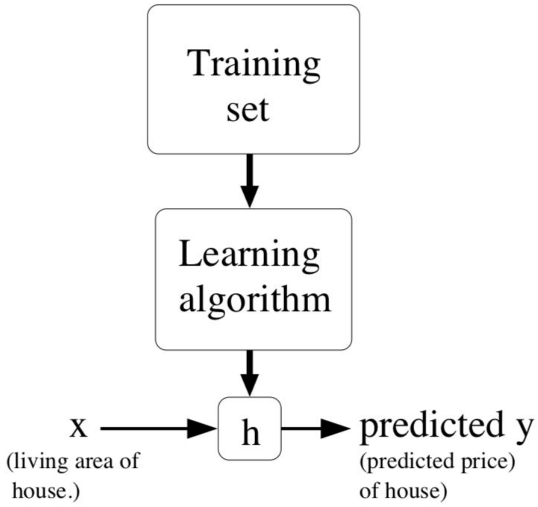
Data Representation: From Table to Matrix Form

	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$		$\mathbf{x_n}$	$\mathbf{y}$	
	total_bedrooms	**population**	**households**		**median_income**	**median_house_value**	
$\mathbf{x}^{(1)}$	129.0	322.0	126.0	. . .	8.3252	452600.0	$y^{(1)}$
$\mathbf{x}^{(2)}$	1106.0	2401.0	1138.0	. . .	8.3014	358500.0	$y^{(2)}$
$\mathbf{x}^{(3)}$	190.0	496.0	177.0	. . .	7.2574	352100.0	$y^{(3)}$
	⋮	⋮	⋮		⋮	⋮	
$\mathbf{x}^{(m)}$	280.0	565.0	259.0	. . .	3.8462	342200.0	$y^{(m)}$

$\mathbf{X}$

*data from the California Housing dataset under Public Domain License

Linear Regression as a learning algorithm



A generalized linear model:

$$h_\theta(\mathbf{x}) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

OR a more concisely vectorized (short) form of the model/hypothesis:

$$\hat{y} = h_\theta(\mathbf{x}) = \theta^\top \mathbf{x}$$

Hypothesis function

***(n+1) x 1* Parameter Vector**

***(n+1) x 1* Feature Vector**

Performance Measure: Mean Square Error (MSE)

$$\mathrm{MSE}(\mathbf{X}, \mathbf{y}) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

Feature Matrix

Label vector

Prediction of instance i^th

Label of instance i^th

$$\mathrm{MSE}(\mathbf{X}, \mathbf{y}) = \frac{1}{m}||\hat{\mathbf{y}} - \mathbf{y}||_2^2$$

Euclidean distance between the predictions and the targets

On the training data, we **minimize** this error by solving for where its **gradient is zero**

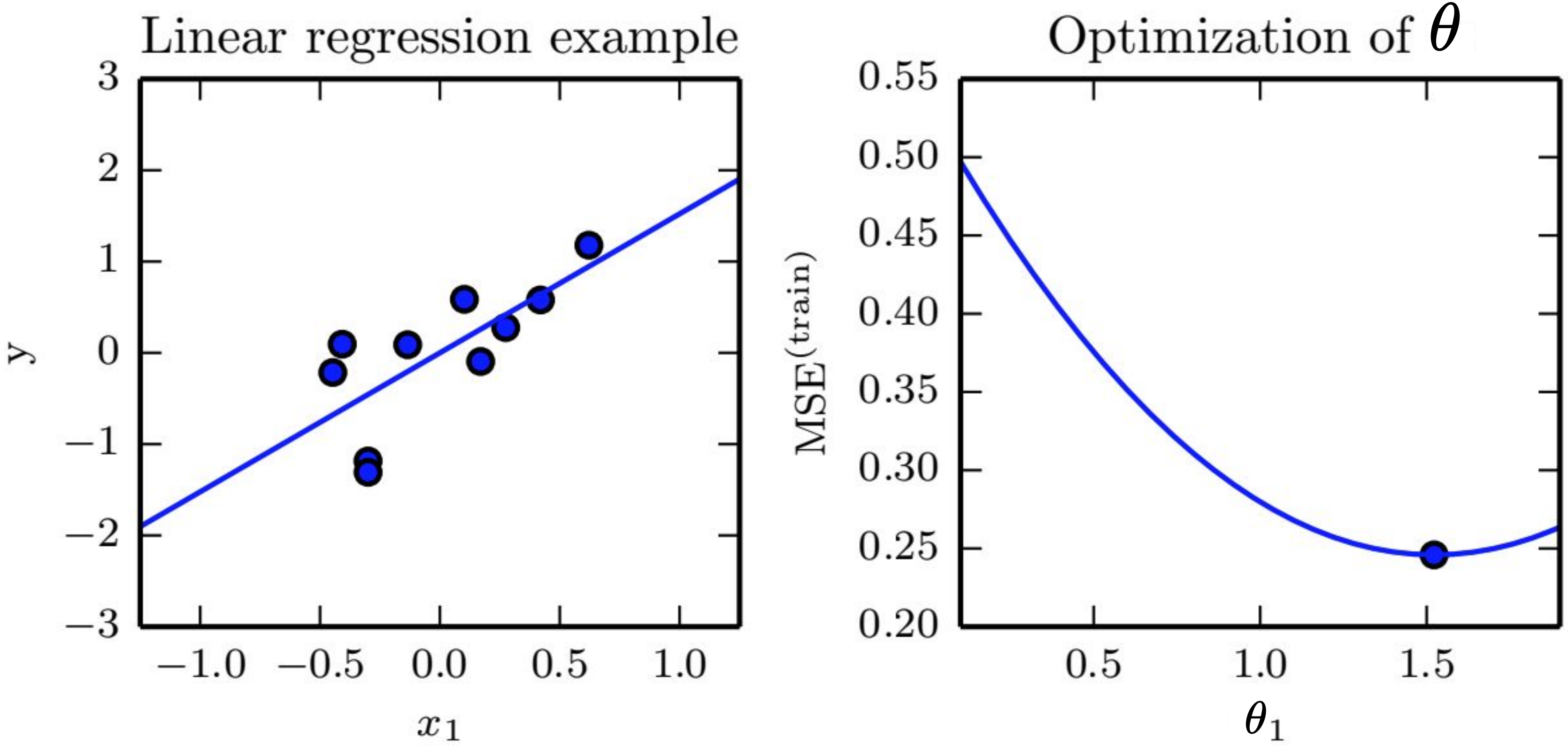
$$\nabla_\theta \mathrm{MSE}(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}}) = 0$$

Close-form Deviation of the Normal Equation

$$
\begin{aligned}
&\nabla_\theta \mathrm{MSE}(\mathbf{X}, \mathbf{y}) = 0 \\
\Rightarrow\ &\nabla_\theta \frac{1}{m} ||\hat{\mathbf{y}} - \mathbf{y}||_2^2 = 0 \\
\Rightarrow\ &\nabla_\theta \frac{1}{m} ||\mathbf{X}\theta - \mathbf{y}||_2^2 = 0 \\
\Rightarrow\ &\nabla_\theta (\mathbf{X}\theta - \mathbf{y})^\top (\mathbf{X}\theta - \mathbf{y}) = 0 \\
\Rightarrow\ &\nabla_\theta \left(\theta^\top \mathbf{X}^\top \mathbf{X}\theta - 2\theta^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y}\right) = 0 \\
\Rightarrow\ &2\mathbf{X}^\top \mathbf{X}\theta - 2\mathbf{X}^\top \mathbf{y} = 0 \\
\Rightarrow\ &\theta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}
\end{aligned}
$$

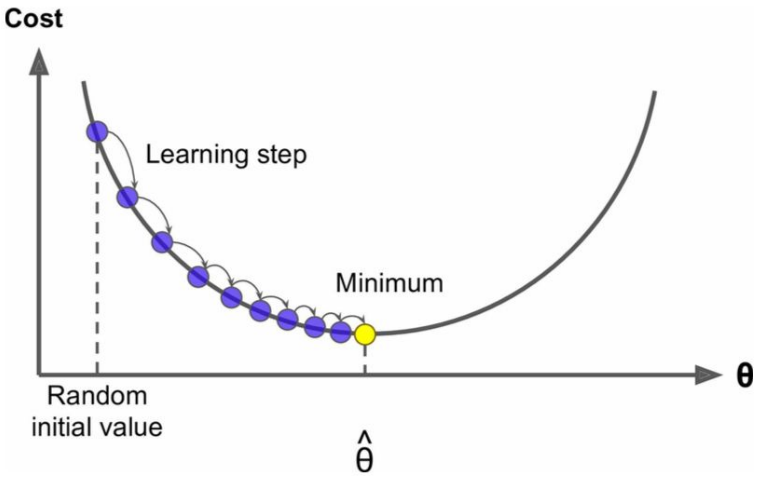
$\leftarrow$ **It's a beautiful thing!**

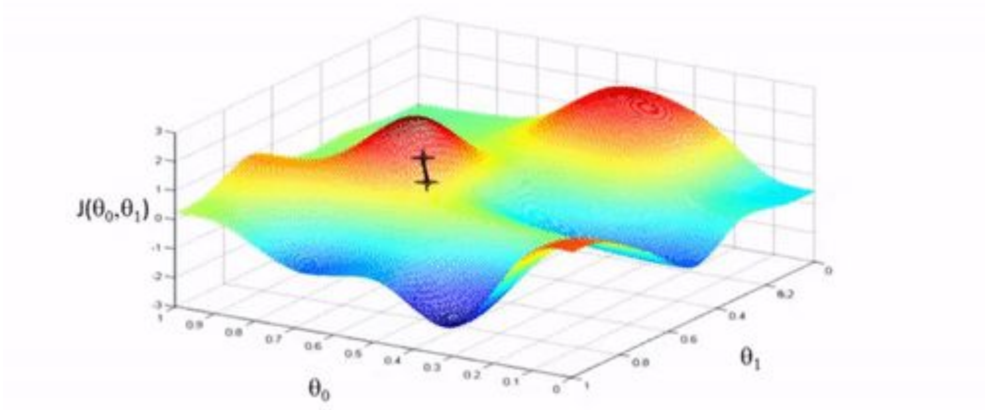
Minimizing MSE with Normal Equation



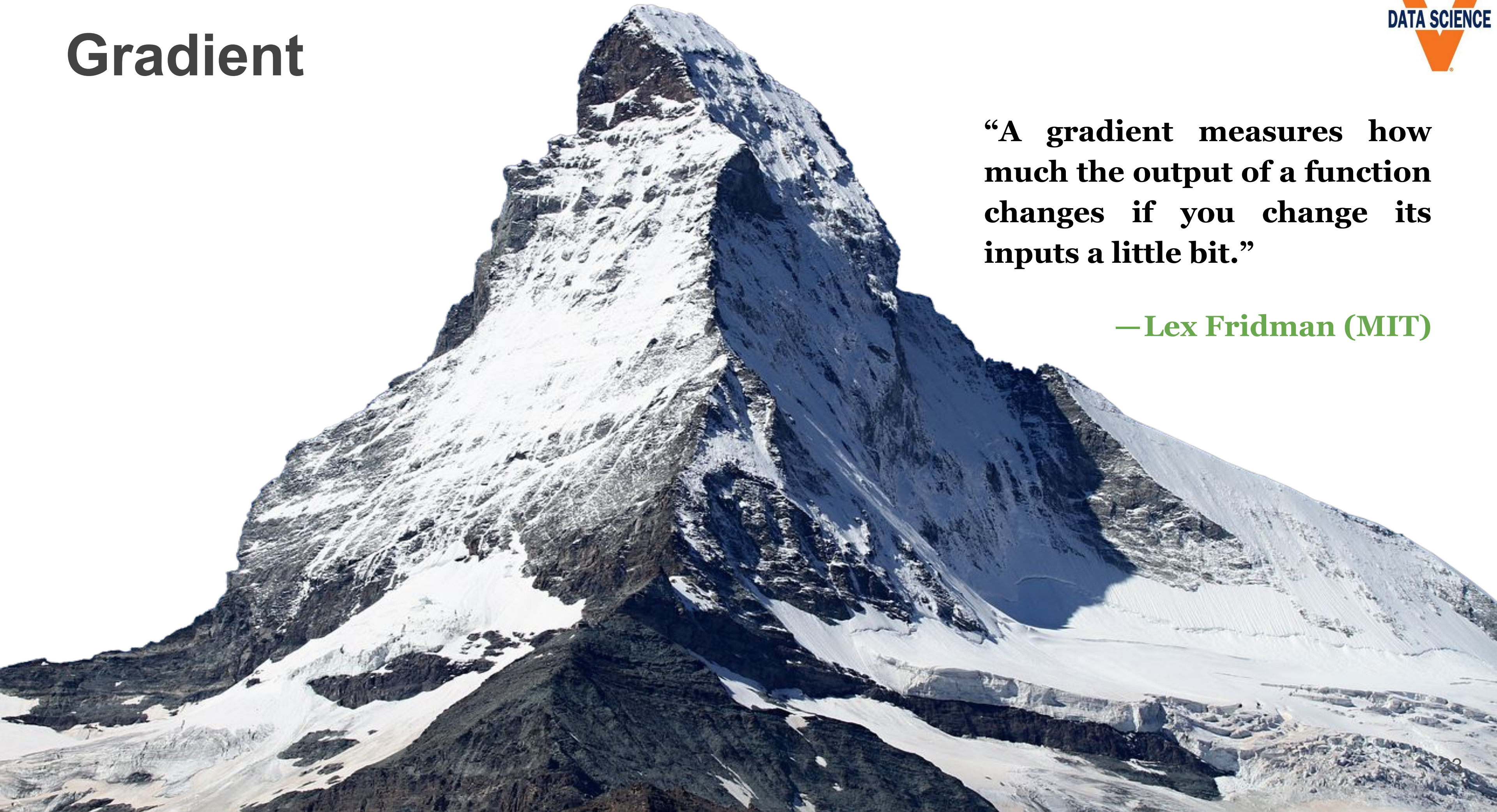
Gradient Descent (GD)

- The closed-form optimization will not typically be feasible, so an iterative optimization algorithm can be used to find the optimal solution.
- The core idea is to tweak parameters **iteratively** to **minimize** a loss function.
- Determined by a ***learning rate*** (how large each learning step should be)





Gradient

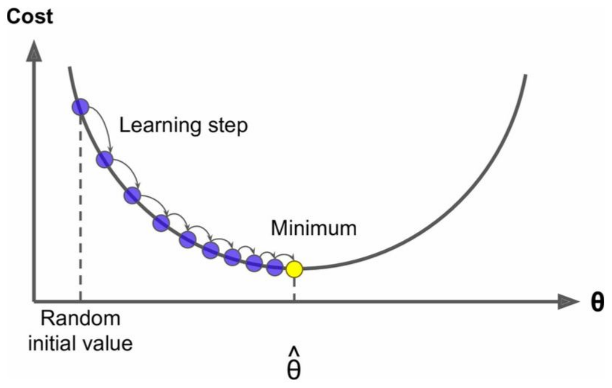


"A gradient measures how much the output of a function changes if you change its inputs a little bit."

—Lex Fridman (MIT)

Batch Gradient Descent (BGD)

- Use all of training data → batch
- Calculate how much the loss function will change if we change a parameter just a bit (ie. partial derivatives)
- Same as: "what is the slope of the mountain if I take a step to the east?" then as the same question for other directions



Minus sign because we want to move downhill, or the opposite direction of the partial derivative

$$\theta_j := \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta), (j = 1 \ldots n)$$

Learning Rate

Cost Function

BGD Formulation

$$\theta_j := \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta), (j = 1 ... n)$$

Expand the loss function:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \left(\frac{1}{m} \sum_{i=1}^{m} \left(\theta^\top \mathbf{x}^{(i)} - y^{(i)} \right)^2 \right)$$

Take partial derivative:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^\top \mathbf{x}^{(i)} - y^{(i)} \right) \boxed{x_j^{(i)} \leftarrow \textbf{Why is this term here?}}$$

Gradient Vector $\nabla J(\theta)$

- Need to calculate over the full training set $\mathbf{X}$
- Use the whole batch at every step → can be slow on large data set

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial \theta_1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) \end{bmatrix} = \begin{bmatrix} \frac{2}{m} \sum_i \left(\theta^\top \mathbf{x}^{(i)} - y^{(i)}\right) x_0^{(i)} \\ \frac{2}{m} \sum_i \left(\theta^\top \mathbf{x}^{(i)} - y^{(i)}\right) x_1^{(i)} \\ \vdots \\ \frac{2}{m} \sum_i \left(\theta^\top \mathbf{x}^{(i)} - y^{(i)}\right) x_n^{(i)} \end{bmatrix} = \frac{2}{m} \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$$

Gradient Descent (GD) Step

$$\theta = \theta - \eta \nabla J(\theta)$$

$$\theta = \theta - \eta \frac{2}{m} \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$$

```
eta = 0.1
n_iterations = 1000
m = 100
theta = np.random.randn(2,1)

for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
```

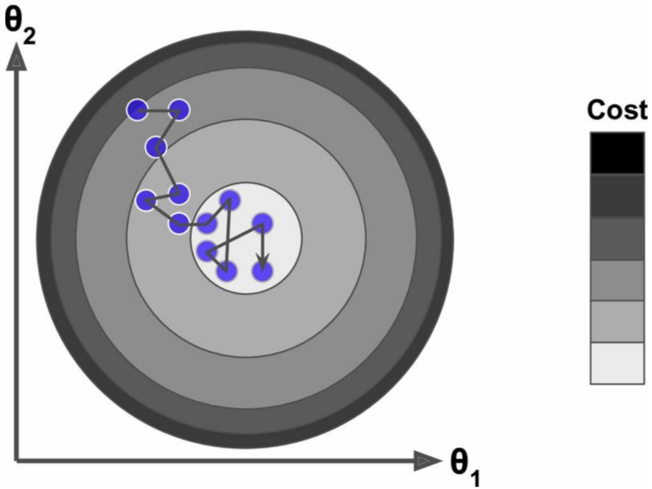
Stochastic Gradient Descent (SGD)

- Instead of using the whole training set, SGD picks **a random sample** in the training set at every step and compute the gradients based on **that** sample.
- It's extremely fast, but is "**stochastic**" (random) in nature, its final parameter values are bounce around the minimum, which are good, but not optimal.

$$\theta = \theta - \eta \frac{2}{m} \mathbf{X}^\top (\mathbf{X}\theta - \mathbf{y})$$

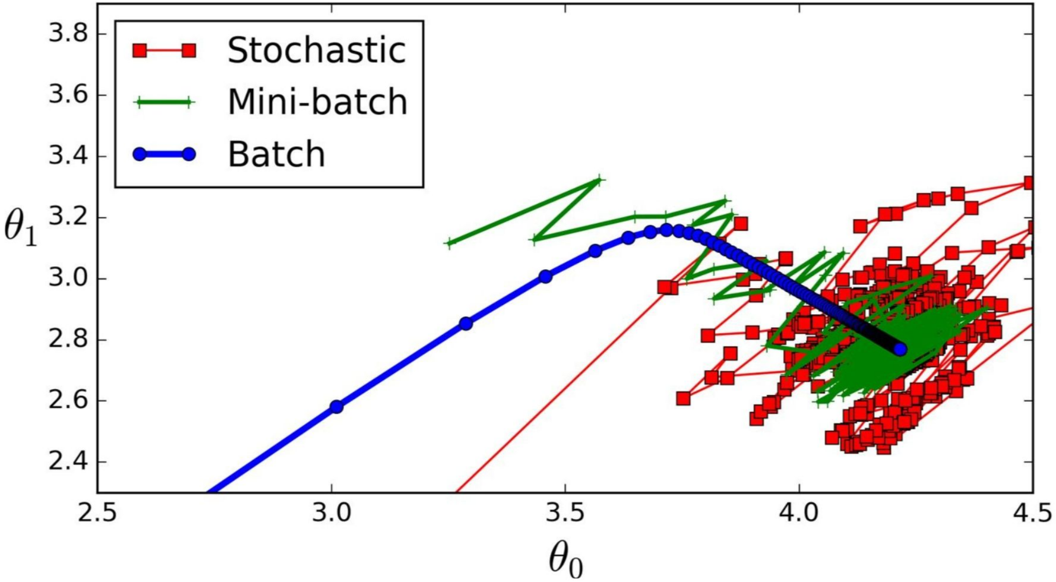
$$\Downarrow$$

$$\theta = \theta - \eta 2\mathbf{x}^{(i)} (\theta^\top \mathbf{x}^{(i)} - y)$$

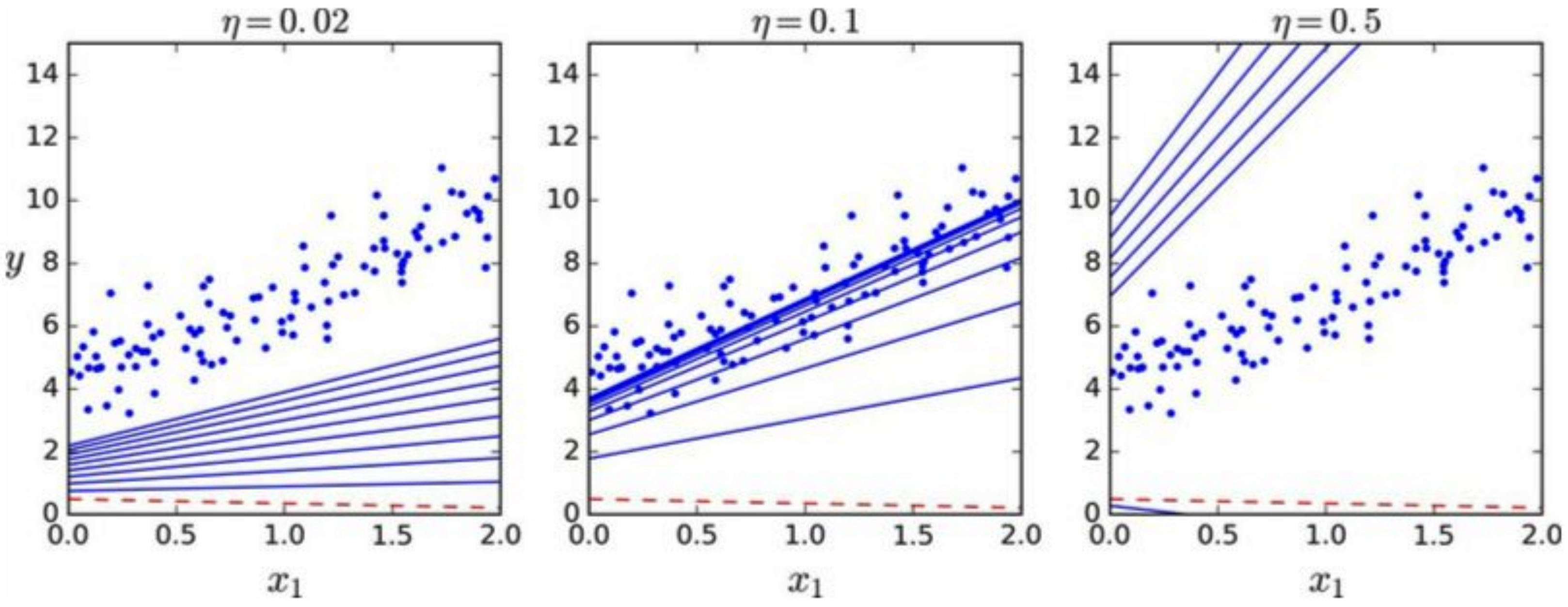


Mini-batch Gradient Descent

Instead of training on the full set (Batch GD) or based on just one sample (Stochastic GD), Mini-batch GD computes gradients on **small random sets of examples** (10-100 in size) called mini-batches → best of both world



GD with various learning rate



3. Connections to Statistics

Maximum Likelihood Estimation (MLE)

- Most optimization in ML can be interpreted as MLE. Let $P(X|\boldsymbol{\theta})$ be a parametric family of probability distributions over the same space indexed by $\boldsymbol{\theta}$.
- $P(X|\ \boldsymbol{\theta})$ maps any configuration of the model to a real number estimating a true distribution of the data.
- The MLE for parameter $\boldsymbol{\theta}$ is defined as:

$$
\begin{aligned}
\theta_{\text{MLE}} &= \arg\max_{\theta} P(\mathbf{X}|\theta) \\
&= \arg\max_{\theta} \prod_{i=1}^{m} P(\mathbf{x}^{(i)}|\theta) \\
&= \arg\max_{\theta} \sum_{i=1}^{m} \log P(\mathbf{x}^{(i)}|\theta) \\
&= \arg\min_{\theta} \sum_{i=1}^{m} -\log P(\mathbf{x}^{(i)}|\theta)
\end{aligned}
$$

We can think of the maximization process as a minimization of the negative log-likelihood

Conditional Log-Likelihood and MSE

In order to predict label **y** given data **X**, we can try to estimate a conditional probability P(**y** | **X**; **θ**):

$$\theta_{\text{MLE}} = \arg\max_{\theta} P(\mathbf{y}|\mathbf{X};\theta) \;\; = \arg\max_{\theta} \sum_{i=1}^{m} \log P(y^{(i)}|\mathbf{x}^{(i)};\theta)$$

For the linear regression problem, we define: $P(y|\mathbf{x}) = \mathcal{N}(y; \hat{y}(\mathbf{x};\theta), \sigma^2)$

$$\theta_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\hat{y}^{(i)} - y^{(i)})^2}{2\sigma^2}\right)$$

$$= \arg\max_{\theta} -m \log\sigma - \frac{m}{2}\log(2\pi) \boxed{- \frac{1}{2\sigma^2} \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||_2^2}$$

Comparing this log-likelihood to: $\arg\min_{\theta} \text{MSE}(\mathbf{X},\mathbf{y}) = \arg\min_{\theta} \boxed{\frac{1}{m} \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||_2^2}$

We can see that the two criteria have different values but the same optimum location, thus justify the use of MSE as a MLE procedure.

Maximum A Posteriori (MAP) Estimation

A Bayesian approach to the point estimate of $\boldsymbol{\theta}$. Recall Bayes' Rule:

$$P(\theta|\mathbf{X}) = \frac{P(\mathbf{X}|\theta)P(\theta)}{P(\mathbf{X})}$$

$$\propto P(\mathbf{X}|\theta)P(\theta)$$

$$\begin{aligned}
\theta_{\text{MAP}} &= \arg\max_{\theta} P(\mathbf{X}|\theta)P(\theta) \\
&= \arg\max_{\theta} \log P(\mathbf{X}|\theta) + \log P(\theta) \\
&= \arg\max_{\theta} \log \prod_i P(\mathbf{x}^{(i)}|\theta) + \log P(\theta) \\
&= \arg\max_{\theta} \sum_i \log P(\mathbf{x}^{(i)}|\theta) + \text{const} \\
&= \arg\max_{\theta} \sum_i \log P(\mathbf{x}^{(i)}|\theta) \\
&= \theta_{\text{MLE}}
\end{aligned}$$

We can conclude that MLE is a special case of MAP where the prior is uniform!

4. Model's Generalization Ability

Generalization

- The ability to perform well on unobserved inputs is called **generalization.**
- Error measured on the training set is called **training error.**
- We typically estimate the **generalization error**, also called **test error**, of a model by measuring its performance on a **test set**.
- The training and test sets are generated by a probability distribution over datasets called the **data-generating process.** Under this process, the examples from each set are assumed to be independent and identically distributed (**i.i.d. assumptions**).

Underfitting and Overfitting

The factors determining how well a machine learning algorithm will perform are

1. Its ability to make the training error small
2. Its ability to make the gap between training and test error small

These two factors correspond to two central challenges in machine learning:

1. **Underfitting**: when the model is not able to obtain sufficient training error.
2. **Overfitting**: when the gap between training and test error too large.

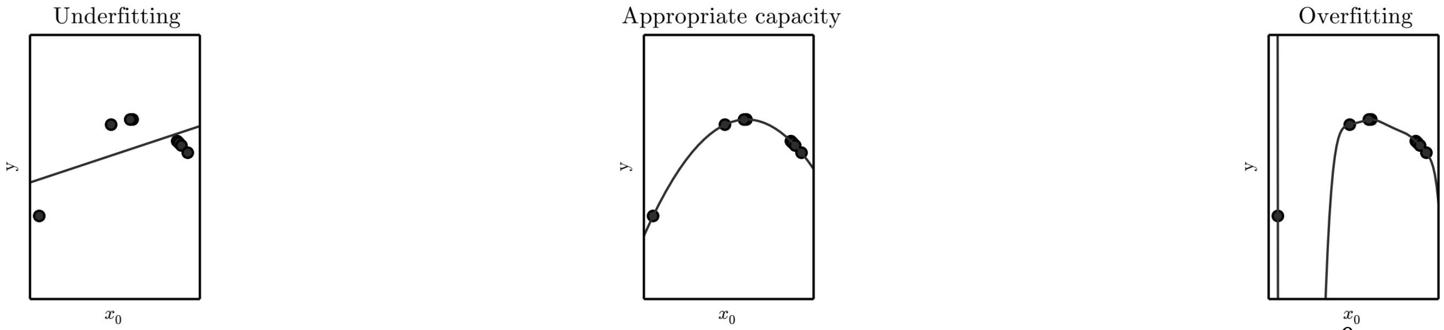
We can control whether a model is more likely to overfit or underfit by altering its **capacity**

Model Capacity

A model capacity is its ability to fit a wide variety of functions:

- Models with **low capacity** may struggle to fit the training set (Underfitting)
- Models with **high capacity** can overfit by memorizing properties of the training set that do not serve them well on the test set (Overfitting)

One way to control model capacity is by choosing its **hypothesis space**, the set of functions that the learning algorithms is allowed to select as being the solution.

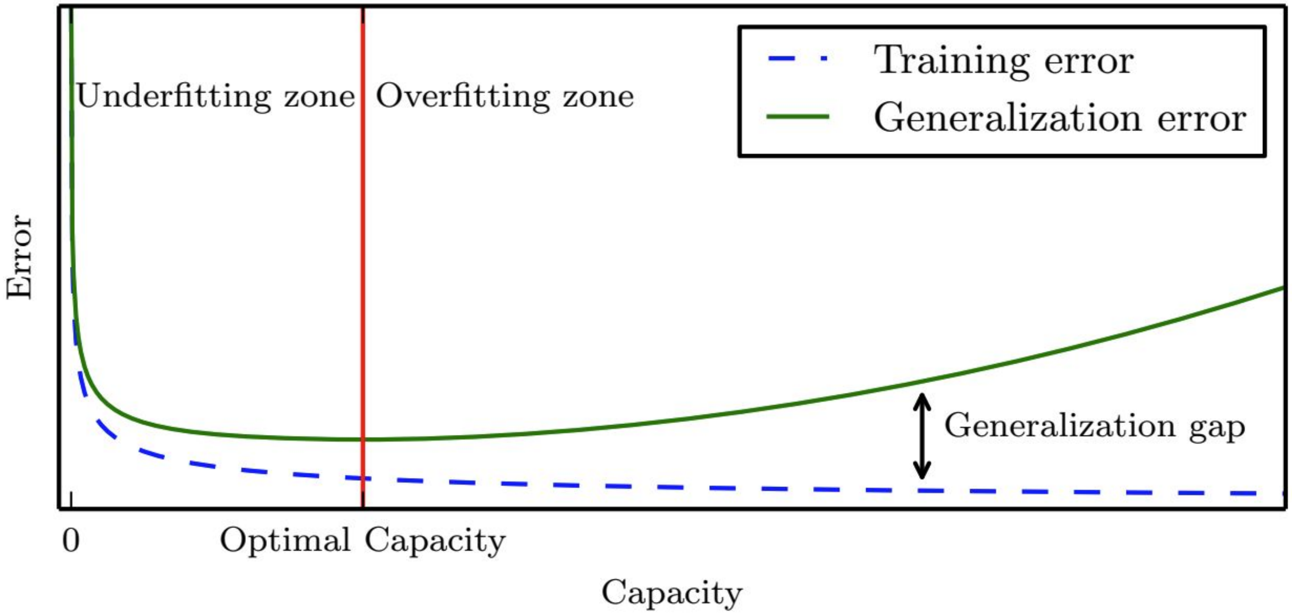


$$\hat{y} = \theta_0 + \theta_1 x_0$$

$$\hat{y} = \theta_0 + \theta_1 x_0 + \theta_2 x_0^2$$

$$\hat{y} = \theta_0 + \sum_{i=1}^{9} \theta_i x^i$$

Relationship between capacity and error



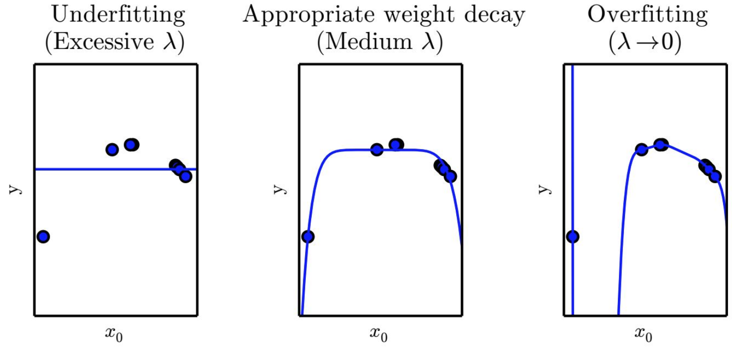
As we increase the model capacity, training error decreases but the gap between training and generalization error increases.

Regularization

We can give a learning algorithm a preference for one solution over another in its hypothesis space. While both functions are eligible, one is preferred.

For example, we can modify the loss function for linear regression to include a preference for the weights to have smaller L2 norm as a **regularizer**:

$$J(\theta) = \mathrm{MSE}(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}}) + \lambda\theta^\top\theta$$



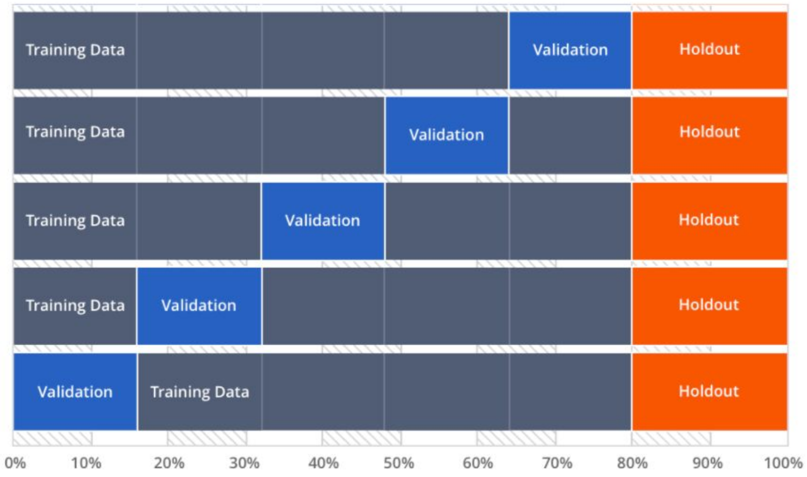
Regularization is intended to reduce generalization error but <u>not</u> training error.

Testing and Validating

- The λ value used to control the regularizer is called a **hyperparameter**, a setting that we can used to control the learning algorithm's behavior.
- Hyperparameters should not be learned from the training set as that learning process will cause overfitting. Instead we need a **validation set.**
- In practice, we typically split a dataset into 3 partitions:
  - **Training Set:** train your model → obtain **training error**
  - **Validation Set:** tune the value of hyperparameters → avoid **overfitting**
  - **Test Set:** evaluate your model → obtain **generalization error**

Cross Validation for small dataset

1. Split data into complementary subsets, train a model on a different combination of these subsets, and **validate** against the remaining parts.
2. Select the model type and hyperparameters which yields small training errors
3. The final model is trained using the hyperparameters on full training set
4. Measure the generalized error on the **test set (holdout)**.



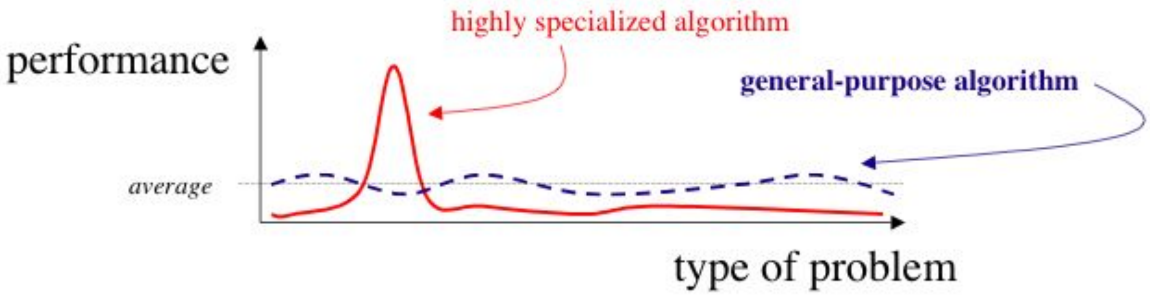
The Recipe to build Learning Algorithms

Nearly all ML algorithms can be described as instances of a simple recipe:

1. A representation of **a dataset** consisting of X and y (or no y provided for unsupervised problems)
2. **A cost function** includes at least a term to perform statistical estimation (ie. negative log-likelihood), and may also include regularization terms (ie. L2 norm of parameters).
3. **A optimization procedure** includes a closed form (ie. normal equation), or an iterative numerical optimization (ie. gradient descent)
4. **A model** specification (ie. Gaussian, linear, or nonlinear polynomial)

By replace any of these components mostly independently from the others, we can obtain a wide range of algorithms.

No Free Lunch (NFL) Theorem

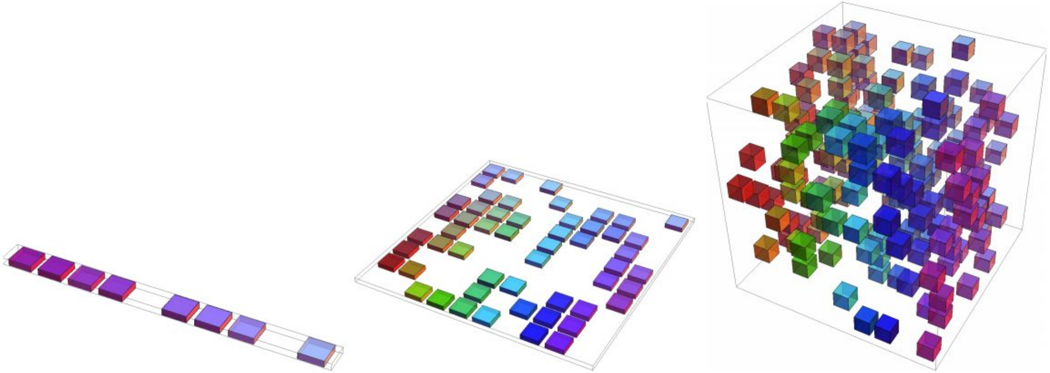


- A model is just a **simplified** version of the observations (data)→ decide which part of the data to keep and which to discard → make some **assumptions** (ie. Linear Assumption in Linear Regression)
- **NFL Theorem** (David Wolpert, 1996): if you make absolutely no assumption about the data, then there is no reason to prefer one model over any other.
- The only way to know for sure which model is best is to **evaluate them all**. In practice, you make some **reasonable assumptions** about the data and evaluate **only a few reasonable models**.

Caveat: The Curse of Dimensionality

Many machine learning problems become exceedingly difficult when the dimensions in the data is high → curse of dimensionality

A statistical challenge arises when the number of possible configurations of $\mathbf{x}$ is **much larger** than the number of training examples → some configuration cells have no observed example. Learning algorithms that fail to scale to this statistical challenge will have difficulties in generalizing complicated AI-level tasks.



Summary: Learning Outcomes

- ✓ Be familiar with some basic ML vocabulary and fundamental concepts
- ✓ Take a look at the optimization procedure of a simple learning algorithm
- ✓ Make some connection to the field of statistical analysis
- ✓ Understand model's abilities for generalization

Discussion: Bias and Variance Tradeoff

- You will have a chance to discuss the tradeoff between bias and variance as a post on **Piazza**!
- You may start by reading the *Deep Learning* book on section **5.4.4**. You are encouraged to do some research on the topic on your own before posting your perspective. Hopefully, we will have an interesting discussion.

The modern machine learning landscape

A great way to get a sense of the current landscape of ML is to look at the competitions on **Kaggle**. Kaggle offers a realistic way to assess what works and what doesn't.

Starting 2016, Kaggle was dominated by two approaches: (1) gradient boosting machines and (2) deep learning. Gradient boosting is used for problems where **structured** data is available while **deep learning** is used for **perceptual** problems such as image classification.

We are on our way to learn about **deep learning...**


Brace Yourselves
Deep Learning is coming...

Bonus Content

Challenges Motivating Deep Learning

- To generalize well, learning algorithms need to guided by implicit "priors" such as the **smoothness prior** (or local constancy), which means that the function we learn should not change very much within a small neighborhood region.
- Many simpler algorithms rely exclusively on this **prior**, and as a result, fail to scale to the statistical challenge involved in solving AI-level tasks.
- The key insight is that a very large number of regions $O(2^k)$, can be defined with $O(k)$ examples, so long as we introduce some dependencies between the regions through additional **assumptions** about the underlying data distribution.
- In order to capture this insight, deep learning algorithms provide **implicit or explicit assumptions** that are reasonable for a broad range of AI tasks.

The assumption for Deep Learning

Deep learning assumes that the data was generated by the **composition of features**, potentially at multiple levels in a hierarchy.

These apparently **mild assumptions** allow an exponentially gain in the relationship between the number of examples and the number configurations that can be distinguished.

The exponential advantages conferred by the use of **deep distributed feature representations**.