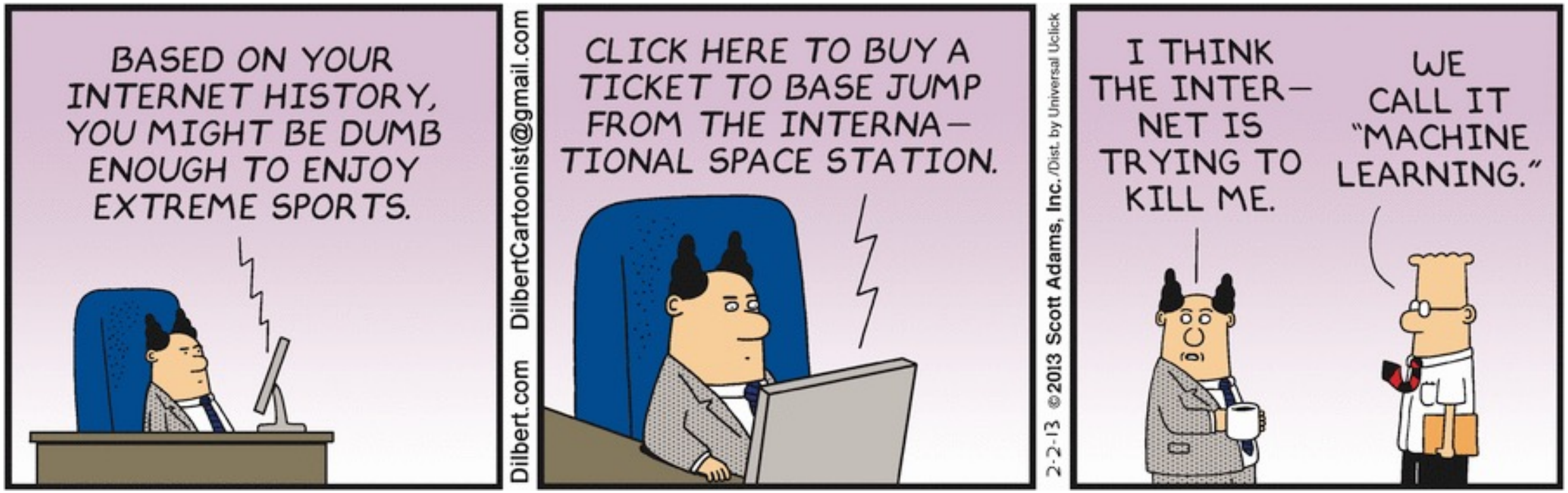
COMP 341 Intro to AI
Machine Learning



Asst. Prof. Barış Akgün

Koç University

What is Machine Learning?

- Seriously, what is machine learning?
- “field of study that gives computers the ability to **learn** without being explicitly programmed” – Arthur Samuel 1959
- “The capacity of a computer to modify its processing on the basis of newly acquired information.” – Oxford Dict.
- Names coined: AI 1956, ML 1959

Who is Arthur Samuel?

- Pioneer Computer Scientist
- Computers for non-numerical programming
- Inventor of alpha-beta pruning
- Checkers AI: alpha-beta pruning + evaluation function
- Increased IBM stock by 15 points overnight!

ML in Checkers

First ML Ideas:

- Rote Learning: Memorize the states and the outcome of the game

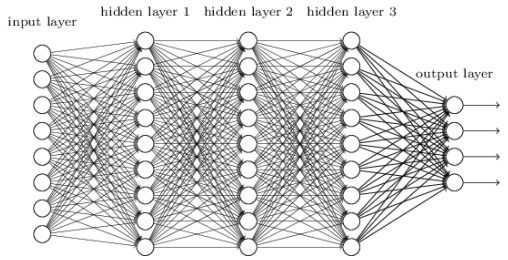
- Self-Learning: Play against itself to tune the weights of the evaluation function

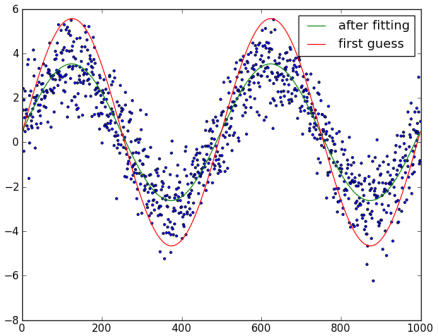
$$J(x) = w_1 f_1(x) + w_2 f_2(x) + \ \ldots + w_n f_n(x) = \sum_{i=1}^{n} w_i f_i(x)$$

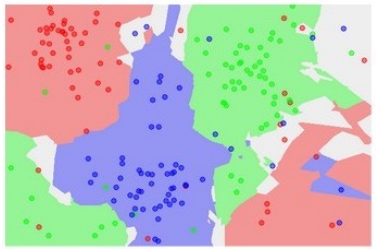
Fun fact: By the end of training, his own program was beating Arthur Samuel

Brief History of ML

- 1763: Underpinnings of Bayes' Theorem (probability)
- 1805: Least Squares (optimization, fitting)
- 1896: Underpinnings of Linear Regression (statistics, fitting)
- 1913: Markov Chains (sequence analysis)
- 1950: Turing's Predictions
- 1951: First Neural Network "Machine"
- 1952: Arthur Samuel starts his checkers work
- 1957: Discovery of perceptron (Neural Networks)
- 1967: Nearest Neighbors
- Late 1960s: Perceptron Book: 1st death of NNs
- 1970: "Backpropagation" – NN training (widespread use in 1986)
- 1971: VC Dimension (Theory of learning capacity – wide scale use later)

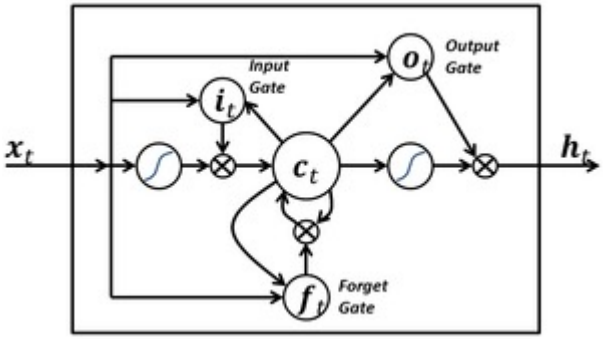


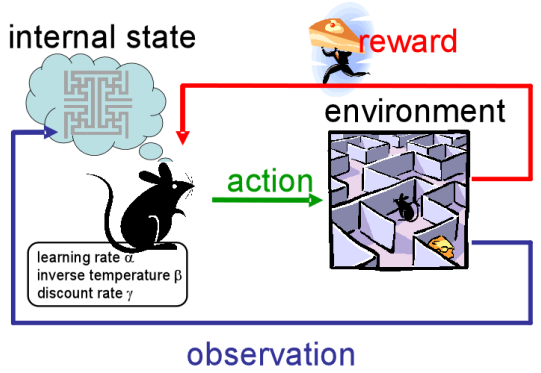


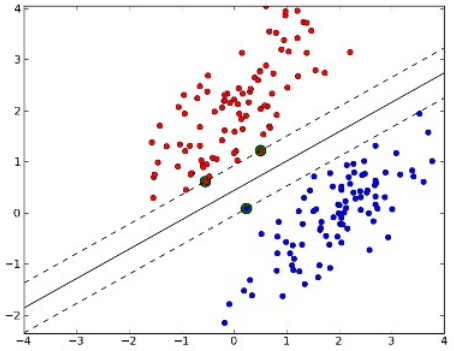


Brief History of ML

- 1980: Explanation Based Learning
- 1981: Neocognitron (Inspiration for Deep Nets)
- 1982: Recurrent NNs (Sequence/Memory for NNs)
- 1984: Probably Approximately Correct Learning (Theory of why ML is possible)
- 1986: Backpropagation: Rebirth of NNs
- 1989: Reinforcement Learning
- 1995: Support Vector Machines (2nd death of NNs)
- 1997: LSTM (For NNs)
- Late 1990s-2010s: Widespread applications of ML and many new methods
- 2012: Deep Learning's World Debut (2nd rebirth of NNs)
- 2016: Deep Reinforcement Learning and AlphaGo
- 2017: Transformers







Out of space for more stuff but you get the idea

Modern Applications

Literally too many to list

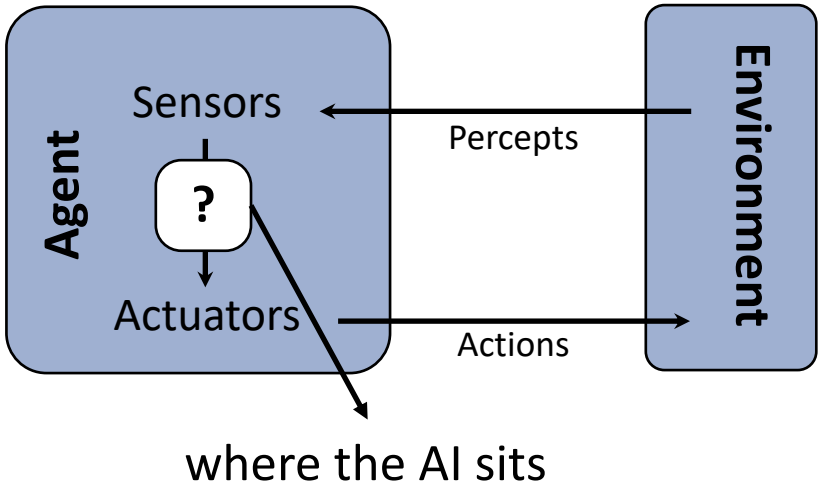
- Self-driving cars
- Genome analysis
- Effective web search Fraud detection
- Recommender systems
- Intelligent Assistants
- Algorithmic Trading
- Cybersecurity
- Image Generation
- Chatbots
- Healthcare
- Image/Video analysis
- Robotics
- Game playing
- Adaptive/Intelligent User Interfaces
- Information retrieval
- Decision aids

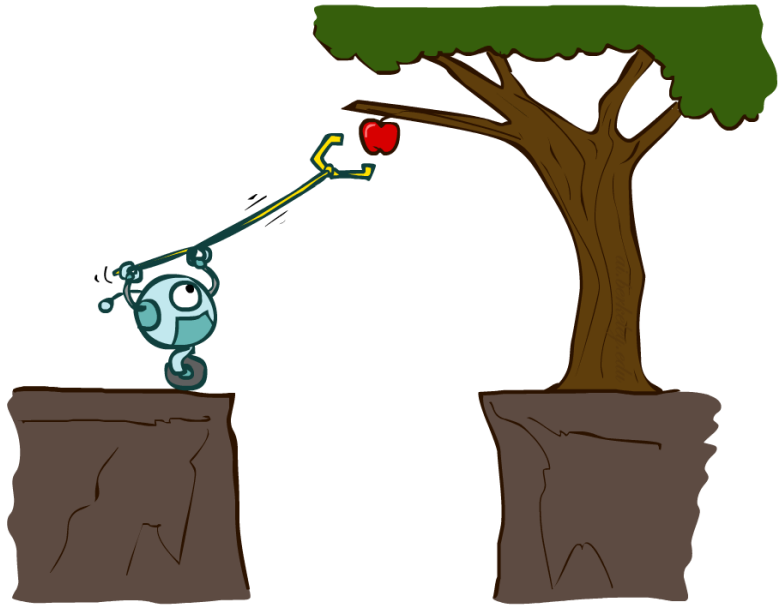
Relation to Agent-Based AI

- A **rational** agent selects actions that maximize its (expected) utility

- AI: Mapping percept histories to actions (deciding which action to take)

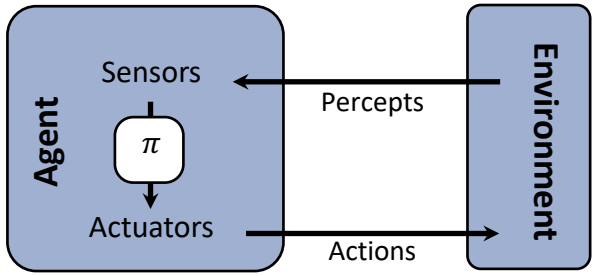
- So far:
  - Search / Adversarial Search
  - Local Search / CSPs
  - Bayesian Networks / HMMs





Agents and ML

- So far, we have used **models** and **hand designed evaluation functions** to make decisions and reason about the environment
- ML:
  - Can we acquire the **models** from data/experience?
  - Can we learn heuristics/evaluation functions?
  - Can we learn π from data/experience? (directly learning to act)
  - Can we learn "features" from low-level sensor percepts? (e.g., pedestrian detection)
- Observations can be used for improving performance of an agent over time
- Needed for unknown environments, or when designer of the agent can't predict everything
- Can be used to directly program the agent; expose it to reality rather than writing it down
- (Note: Models good enough to do search/planning on, are very difficult to learn for most problems and is an active area of research. However, ML can be used as their subcomponents e.g. for self-driving)



What is learned?

- Learning parameters
  - Assume your model has a specific model and learn its parameters
  - E.g. CPTs, coefficients of a polynomial, weights of a neural network
- Learning structure
  - Learn the relationships between the variables of interest
  - E.g. BN topology, HMM transition models
- Learning patterns (e.g. clustering)

(This is not an exhaustive list)

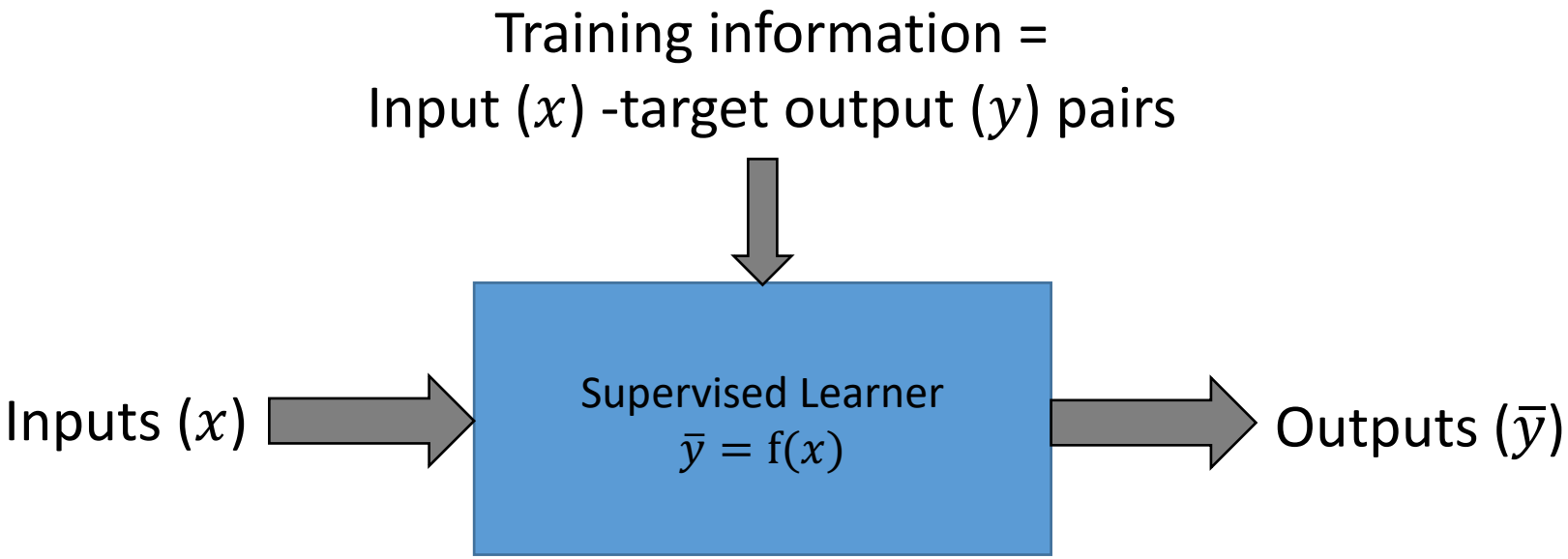
Simple Applications with Coins

1. What is the monetary value of a coin?
2. Which coins are similar to each other?
3. How can I maximize my coins?
4. Create a new coin for me

Each question is a different machine learning problem type!
(There are more ML problem types)

Supervised Learning

- Finding a mapping between inputs and outputs



- Minimize the "loss" between target output (y) and the actual output ($\bar{y}$)

"Improving some **measure of performance** for a given task given **training experience**"

Supervised Learning

- Given data and labels, predict future labels. Eg: Coins and their monetary value

1 kuruş

25 kuruş

5 kuruş

50 kuruş

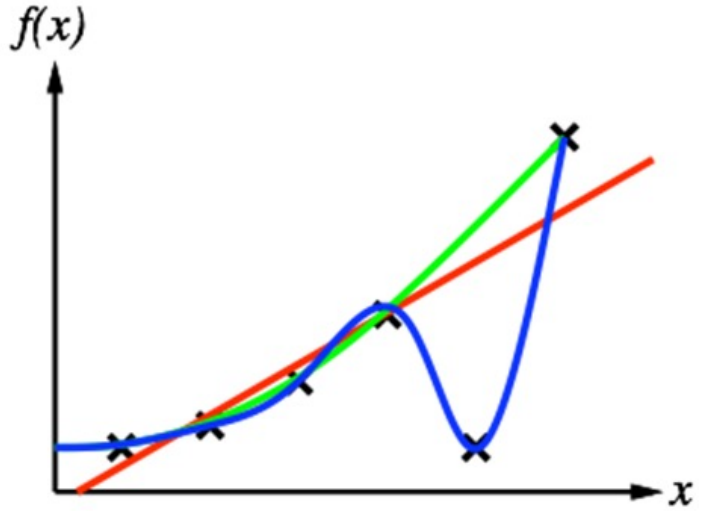
10 kuruş

1 TL

?

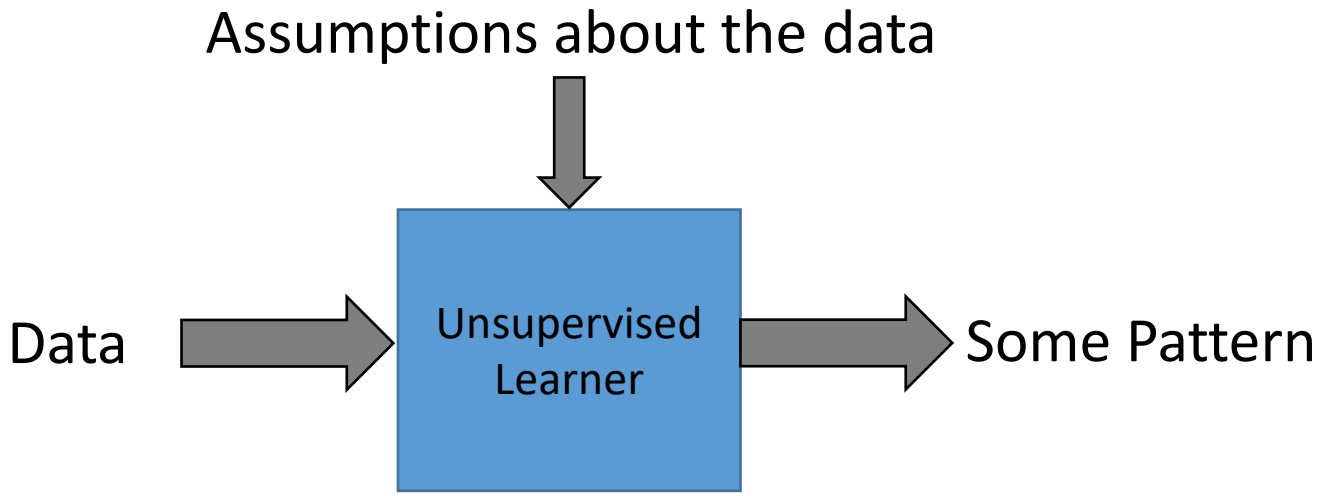
Supervised Learning

- Classification
  - Inputs are mapped to discrete outputs (or classes)
  - E.g. spam filtering (mail spam or not)
  - E.g. coin classification
- Regression
  - Learn a continuous mapping from inputs to outputs.
  - E.g. Predicting tomorrow's market value



Unsupervised Learning

- Given data find the underlying patterns:
  - Clustering
  - Anomaly Detection
  - Dimensionality Reduction (includes **Representation/Embedding Learning**)
  - Density Estimation



- Performance metric depends on these assumptions (e.g. minimizing distance to cluster means)

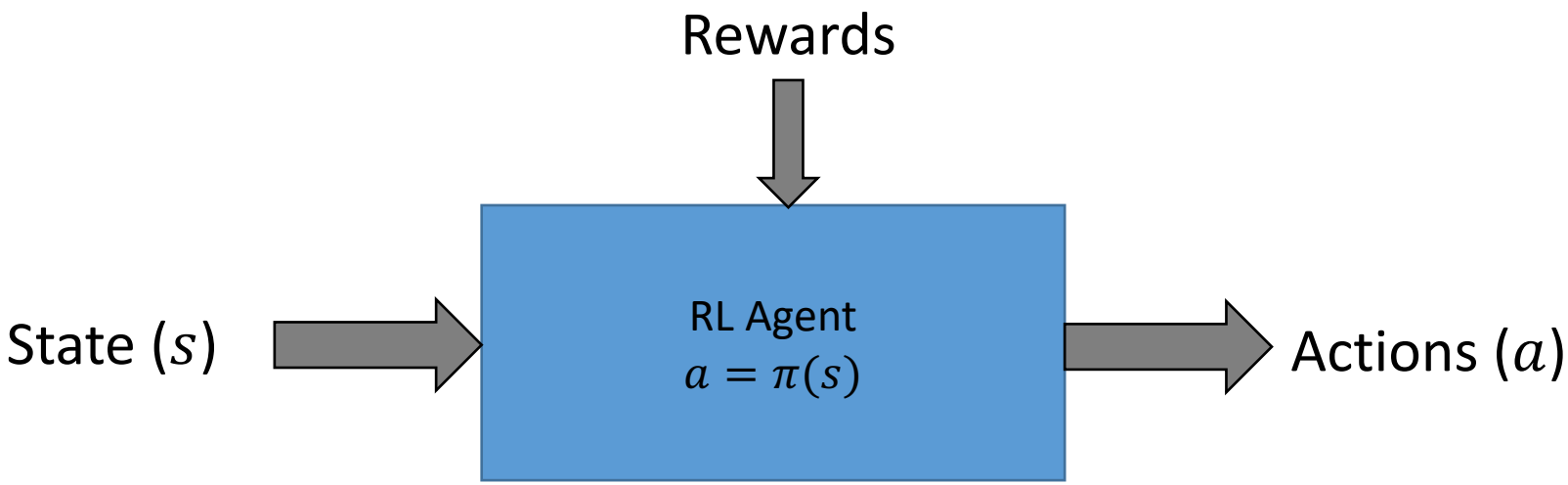
Unsupervised Learning

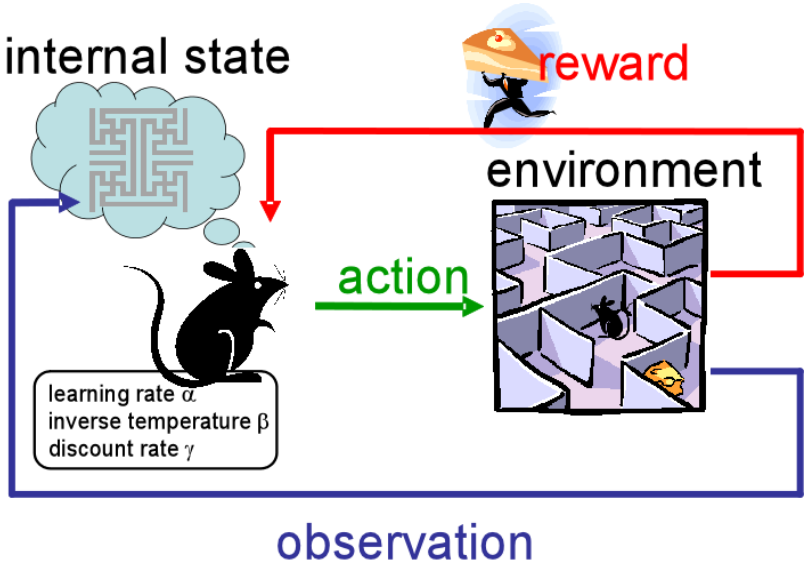
- Given data, find patterns. E.g. group similar coins together - Clustering

- Some other examples
  - Representation learning (includes dimensionality reduction and embeddings)
  - Distribution learning
  - Recommendation

Reinforcement Learning

- Learning a mapping between states and actions to maximize reward (and min. penalty)





- Can learn without training data on which actions to take at which state! (but can also incorporate it)

Reinforcement Learning

- Given reward information, learn how to act. E.g., maximizing the number of coins

RL Example

Initial

Final

OR: https://www.youtube.com/watch?v=gn4nRCC9TwQ

Brainstorm: Coins

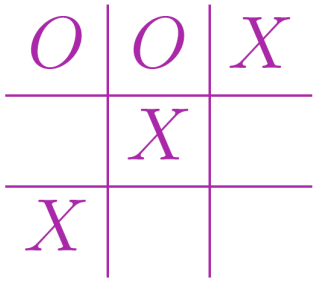
- How to represent these coins?
- What properties to look at?
- Size, shape, weight, shine (luster), …
- These are called features

Features

- Properties being measured or calculated, or characteristics being observed
- Need to be informative and discriminative
- Shape vs Diameter for Turkish coins?

But be careful!

Supervised Learning

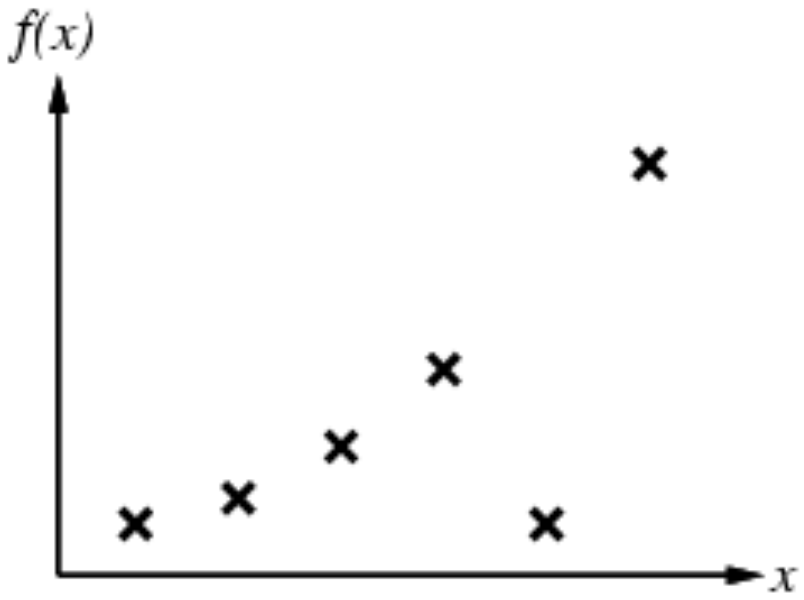
- Learn a function from labelled examples
- x is the input f is the *target* function
- A training example is a pair $(x, f(x))$, e.g.,  , $+1$
- Problem: find a hypothesis h such that $h \approx f$, given a set of training examples

- This is a highly simplified model of real learning:
  - Ignores prior knowledge
  - Assumes a deterministic, observable/environment
  - Assumes examples are given
  - Assumes that the agent wants to learn f -why?

Supervised Learning

Construct/adjust h to agree with f on training set
(h is consistent if it agrees with f on all examples)

E.g., curve fitting:

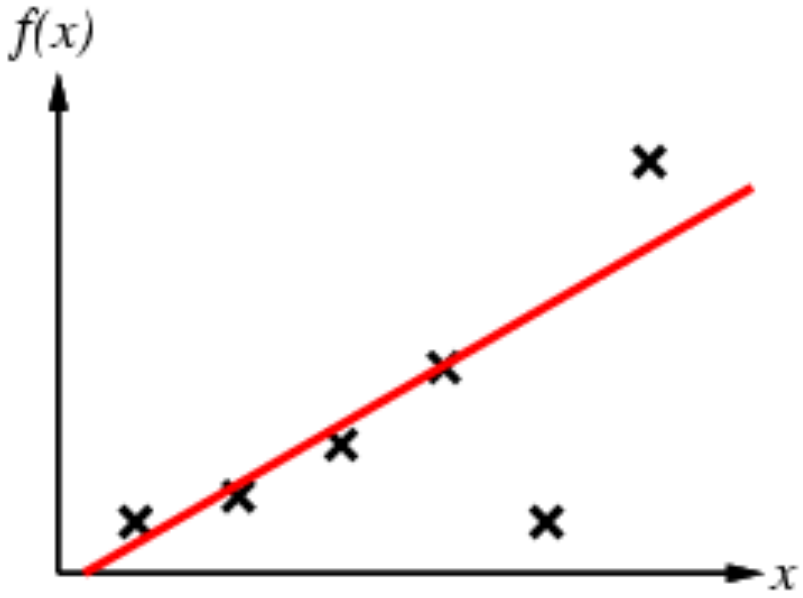


Supervised Learning

Construct/adjust h to agree with f on training set
(h is consistent if it agrees with f on all examples)

E.g., curve fitting:

$h = ax + b$

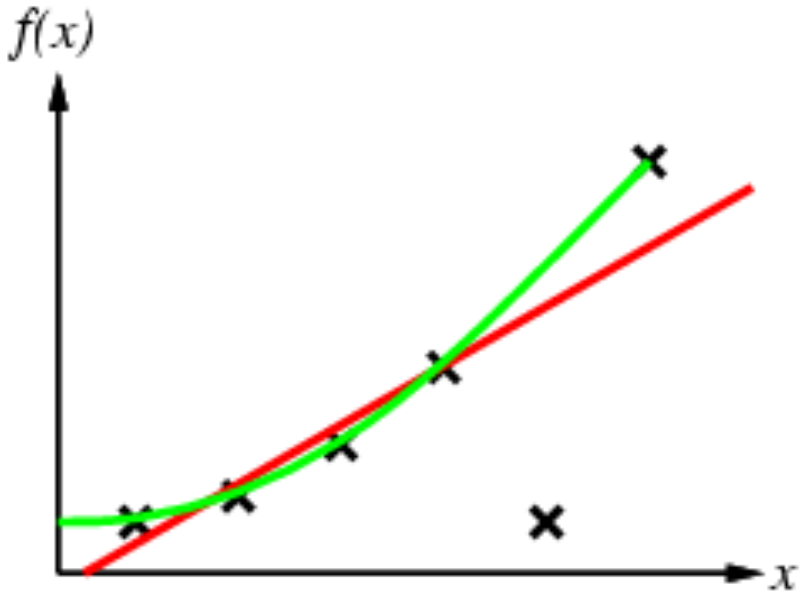


Supervised Learning

Construct/adjust h to agree with f on training set
(h is consistent if it agrees with f on all examples)

E.g., curve fitting:

$h = ax^2 + bx + c$

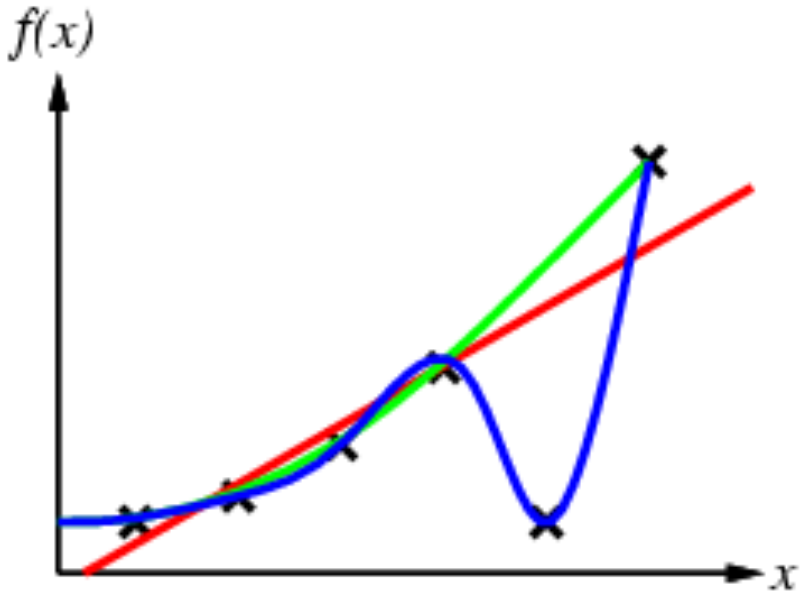


Supervised Learning

Construct/adjust h to agree with f on training set
(h is consistent if it agrees with f on all examples)

E.g., curve fitting:

$$h = \sum_{i=0}^{n} a_n x^n$$

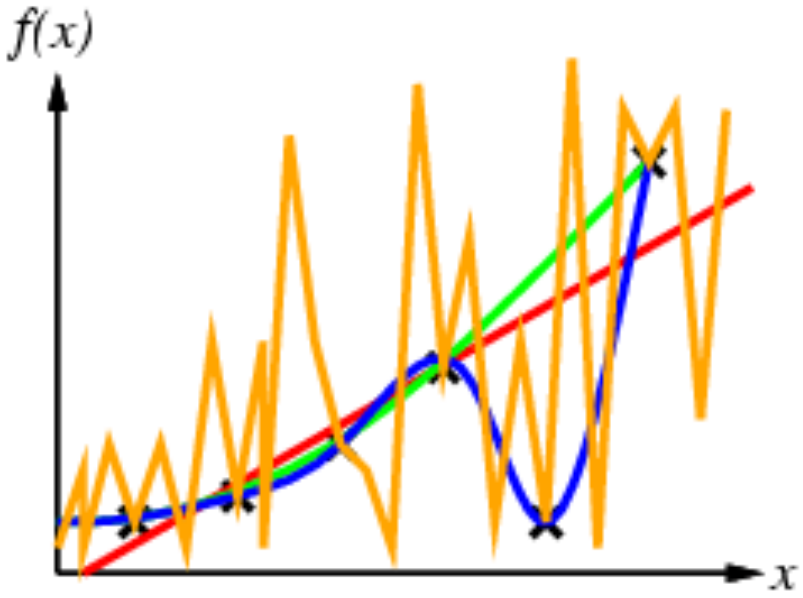


Supervised Learning

Construct/adjust h to agree with f on training set
(h is consistent if it agrees with f on all examples)

E.g., curve fitting:

$h = ?$



What are the differences between h?
Which one would you pick?

Ockham's Razor:
maximize both consistency and simplicity

Supervised Learning

Ockham's Razor:
maximize both consistency and simplicity

simpler

$h = ax + b$

$h = ax^2 + bx + c$

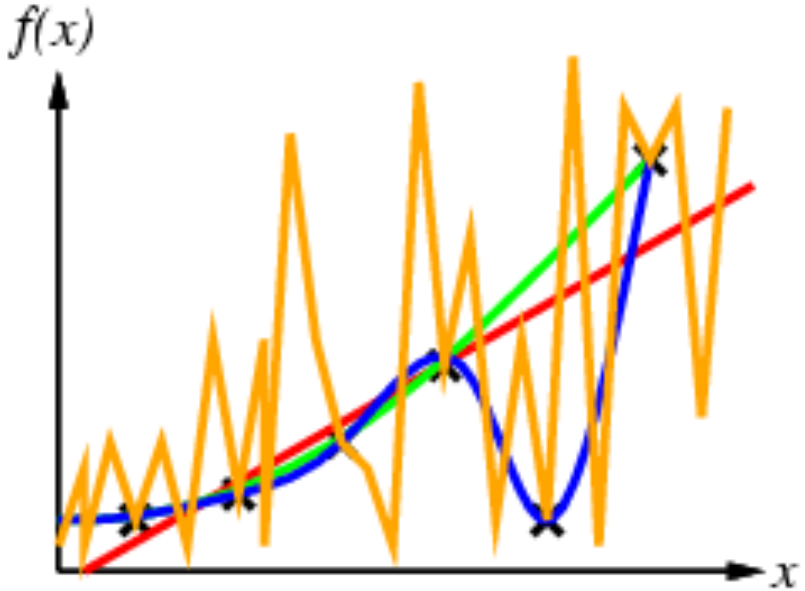
$h = \sum_{i=0}^{n} a_n x^n$

$h = noisy\_zigzag(x)$

more complex

less consistent

more consistent

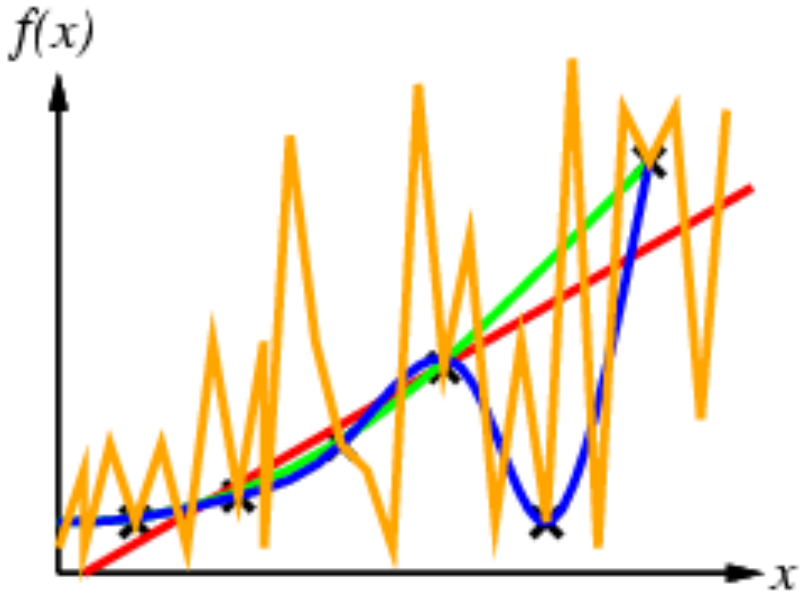


Which one would you choose?
Why?

Supervised Learning

- Underfitting: Cannot capture the underlying trend of the data, cannot generalize well
- Overfitting: Captures the noise or locality of the data, or memorizes the data, cannot generalize well

- Both are bad!
- We have solutions
  - Regularization
  - Cross-validation
  - Train-test-validate



Classification Example

cat

dog

x: images
$f(x)$: class

big cat

big dog

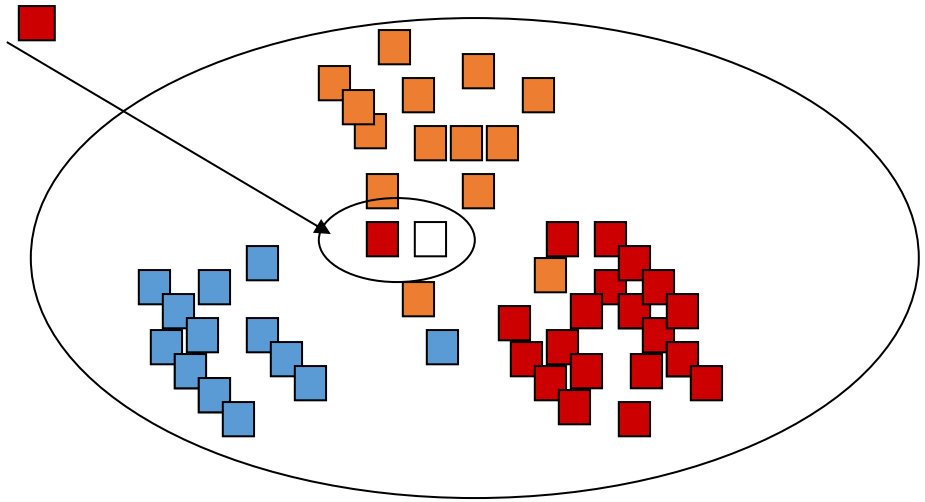
K-Nearest Neighbors

- An SL method, abbreviated as k-NN (not to be confused with Neural Networks)
- A top contender for the easiest ML algorithm

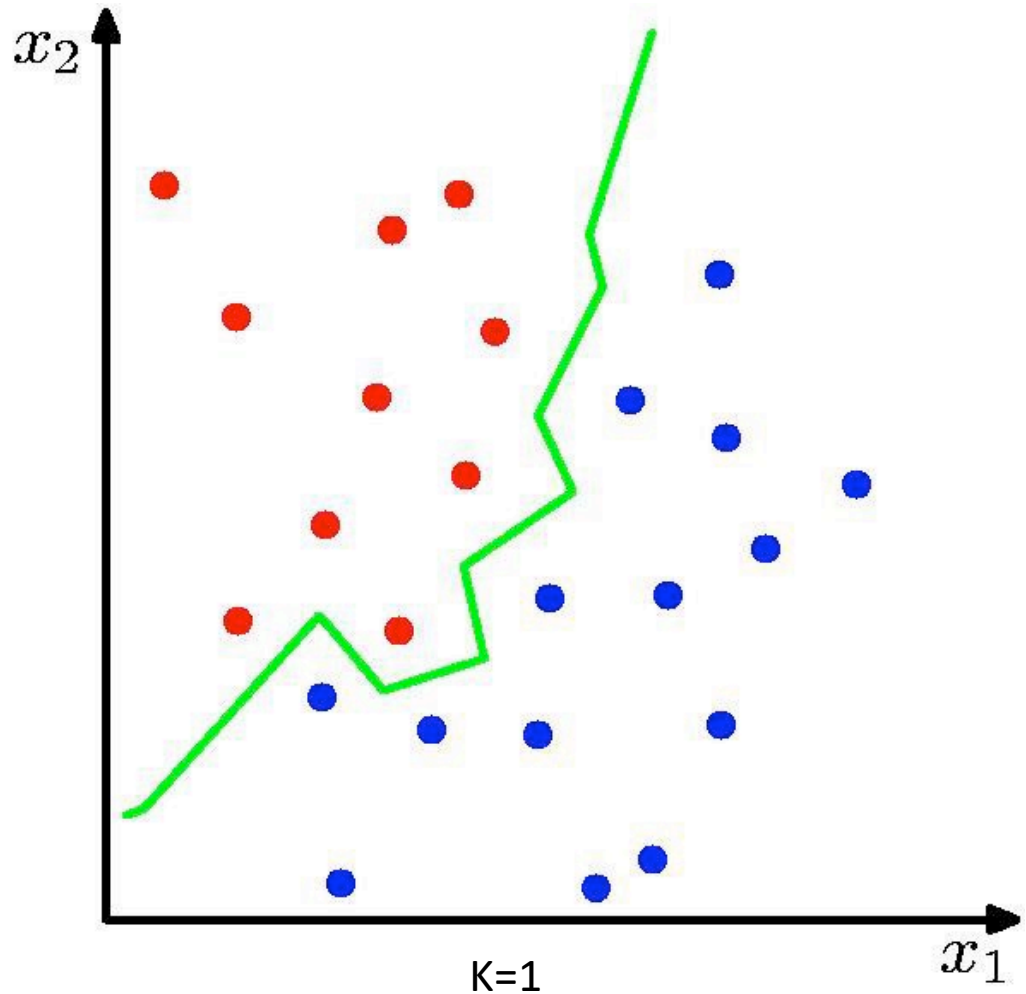
- Let's recall some supervised learning problems
  - Input: $x = \{x_1, \dots, x_n\}, x_i \in \mathrm{X}$
  - Output: $f(x) = \{f(x_1), \dots, f(x_n)\}, f(x_i) \in \mathrm{Y}$
  - Learn a mapping: $h(x)\colon \mathrm{X} \to \mathrm{Y}, \text{s.t. } h \approx f$
  - Classification
    - Binary: $\mathrm{Y} = \{-1, +1\}$ (or true/false, or 1/0 …)
    - Multi-class: $\mathrm{Y} = \{1,2, \dots, C\}$ (cat, dog, bunny, robot …)
  - Regression: $\mathrm{X} = \mathbb{R}^d, Y = \mathbb{R}$ (Y can be multi-D as well, X domain may vary)

Nearest Neighbor Classifier

Idea: For a new data point, predict its label as the closest's label in our training set



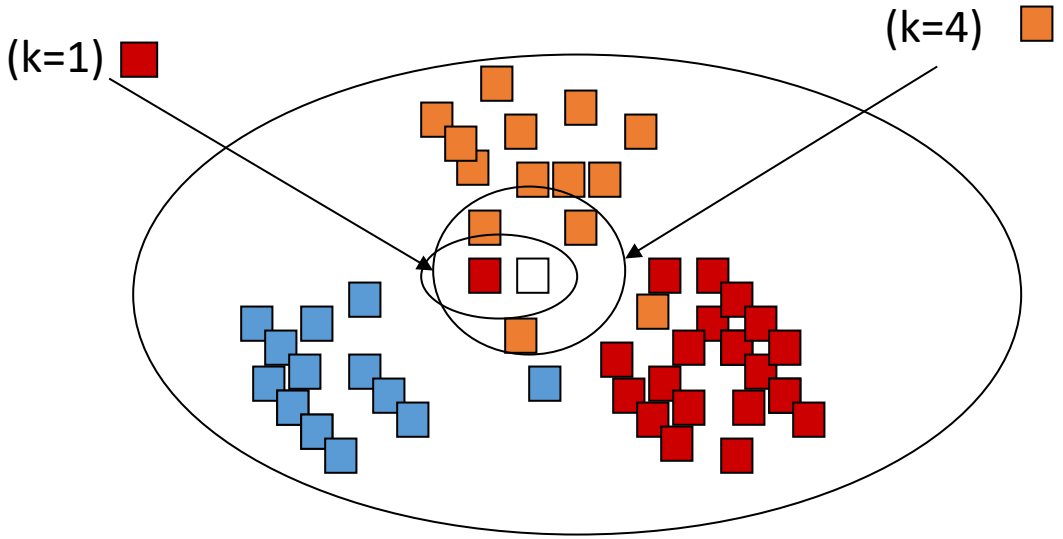
kNN Classifier



K=1

K Nearest Neighbor Classifier

Now look at k neighbors and predict the label as the majority's

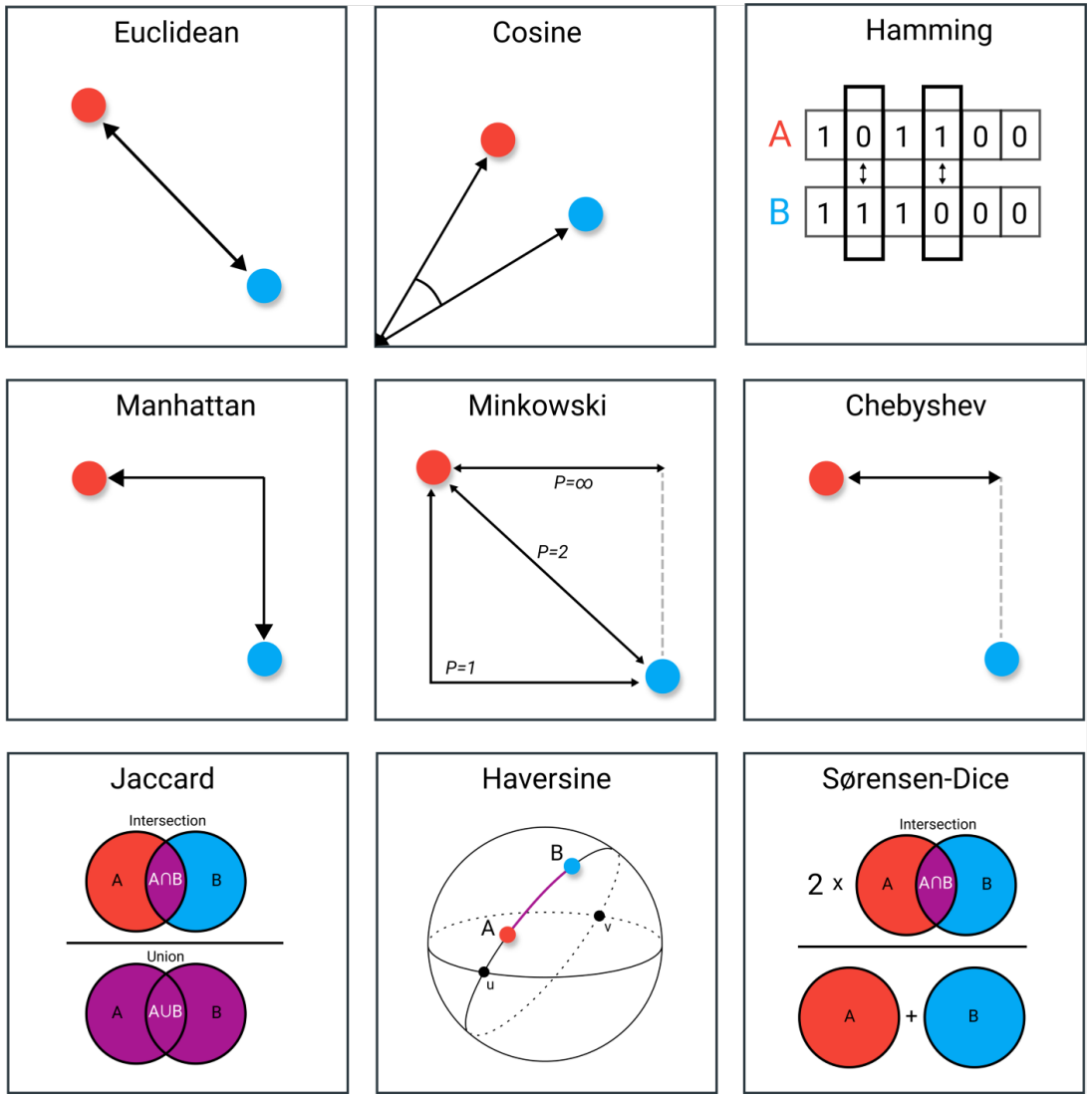


kNN Algorithm Sketch

- Let $\mathrm{D} = \{(x_1, f(x_1)), \ldots, (x_\mathrm{n}, f(x_n))\}$ be the training samples and x be the input sample to be classified

- Calculate distances between the inputs of x_i and x, sort them, then count the occurrences of classes at the top k
  - $d_i = d(x_i, x), L = \{d_1, \ldots, d_n\}, L' = \mathrm{sort}(L, ascending)$
  - Let ind be the indices of the first k elements after sorting and $\mathrm{C} = \{c \mid c \in f(x_{ind})\}$
  - Then take the most common element, $h(y) = mode(\mathrm{C})$

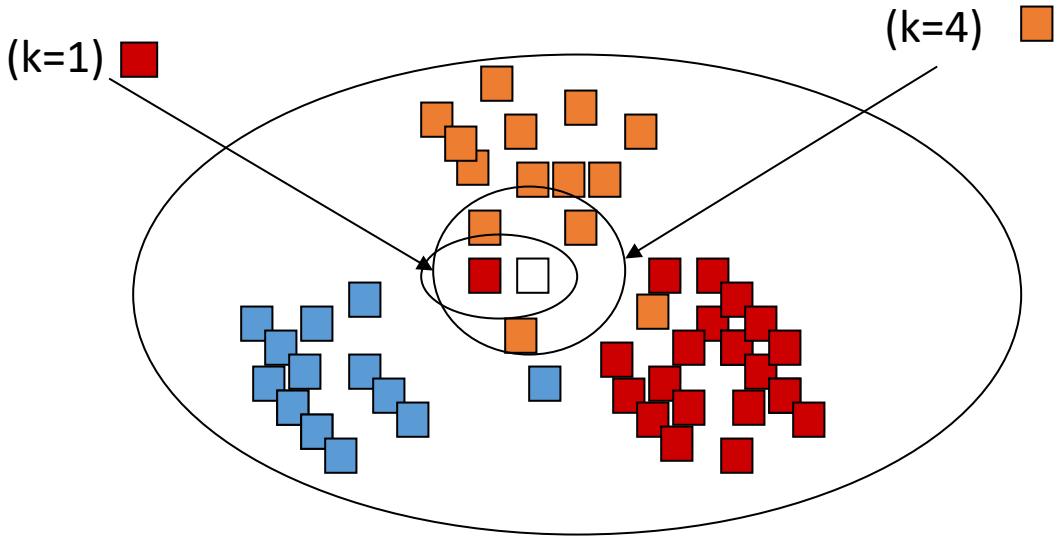
Example Distances Visualized



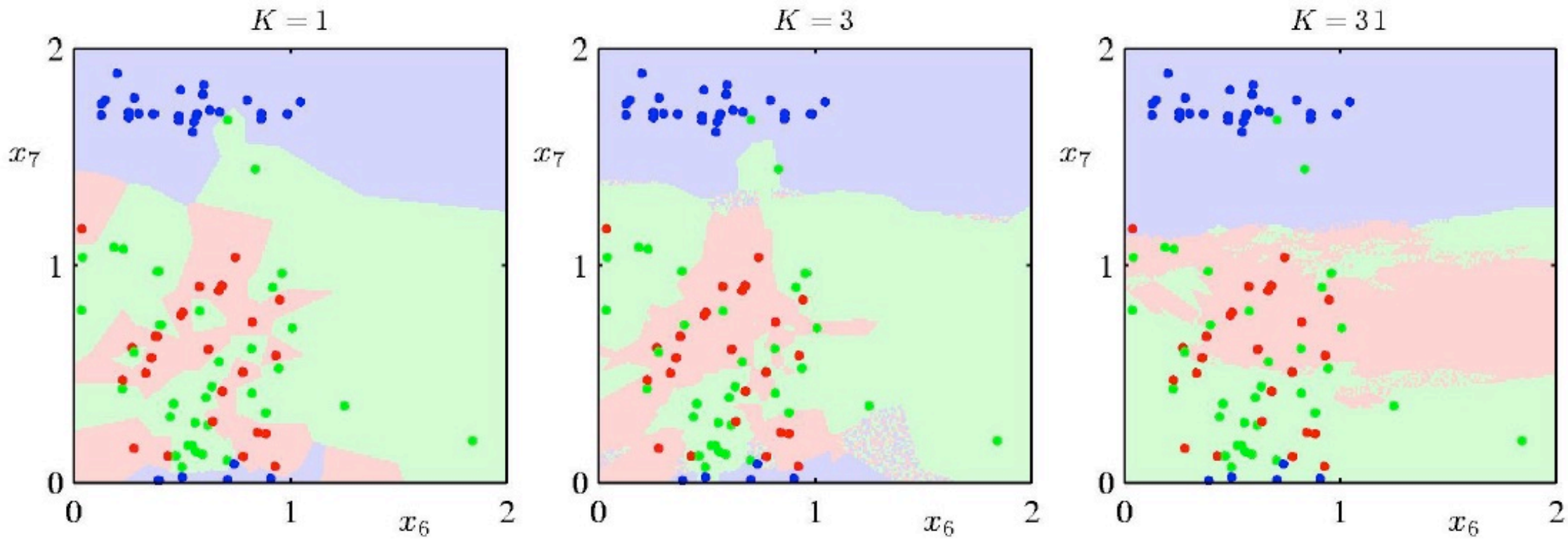
From Maarten Grootendorst

K Nearest Neighbor Classifier

How many neighbors should we count ?



kNN Classifier



- K acts as a smother
- How to choose it?

How to Choose k?

- “Hyper-parameter selection ”: k is a hyper-parameter of the kNN method

- Need to measure performance:
  - On the training data?
  - On some “test” data?

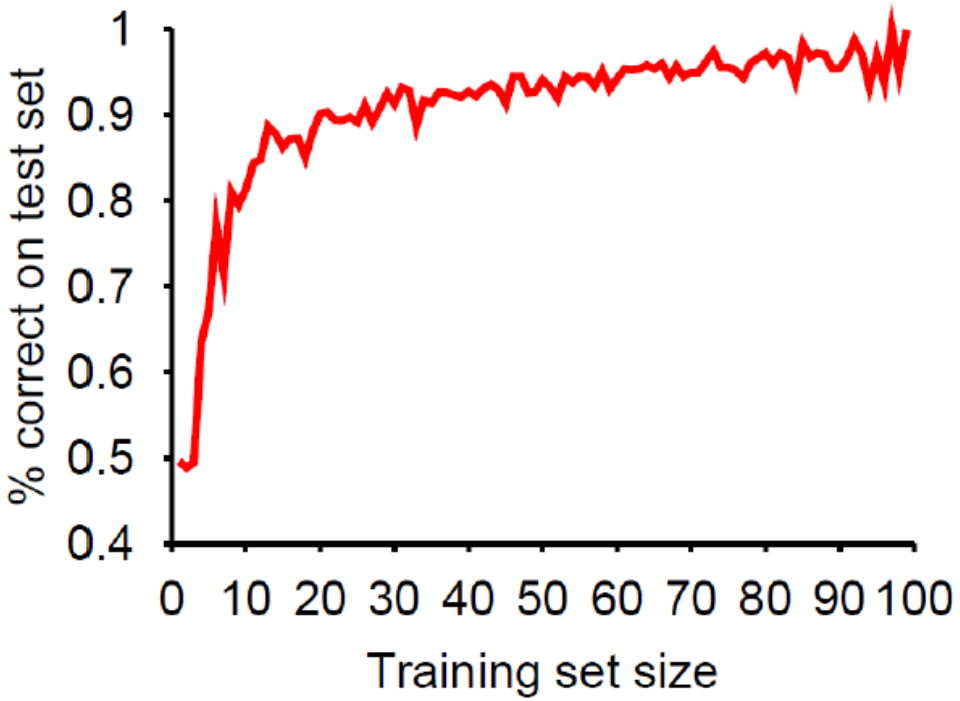
Performance Measurement

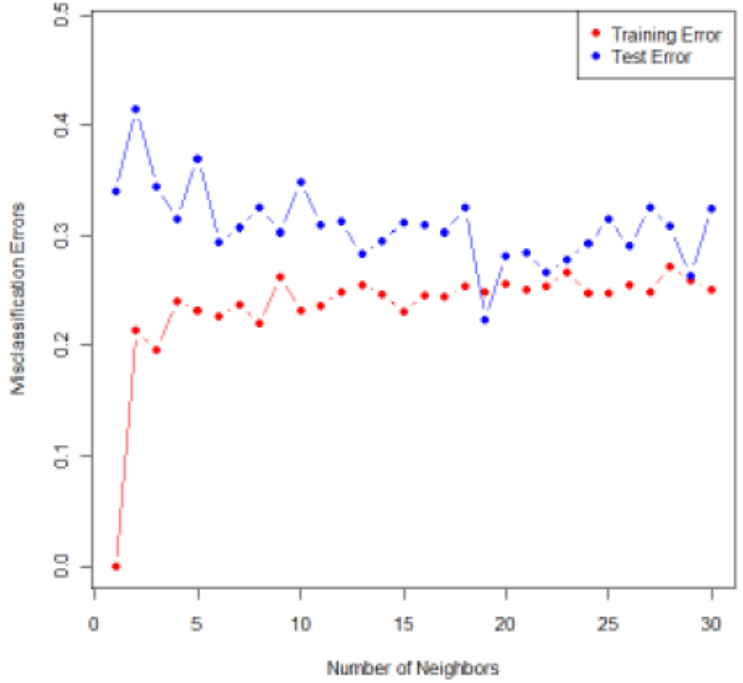
How do we know $h \approx f$?

1. Use theorems of computational/statistical learning theory (can't right now!)
   also, not practical for most methods
2. Try h on a new *test set* of examples

   (need to use the same distribution over the same example space as the training set)

Learning curve = % correct on test set as a function of training set size





How to Choose k?

- “Hyper-parameter selection ”: k is a hyper-parameter of the kNN method

- Need to measure performance:
  - On the training data?
  - On some “test” data?

- Vary k from 1 to K
- Pick the best performing one!
- Any problems?
  - k could be train-test split specific

Cross Validation

Technique to reduce over fitting

- Use 1/k examples for testing, rest for training
- Repeat k times so you've trained and tested with all of the data
- Called k-fold Cross Validation
- Average the performance results for a more accurate measure

Other variants are possible!

The k here is not the "k" in the kNN method and can be picked independently. I did not want to change the name with respect to the existing jargon.

“Train-Validate-Test”

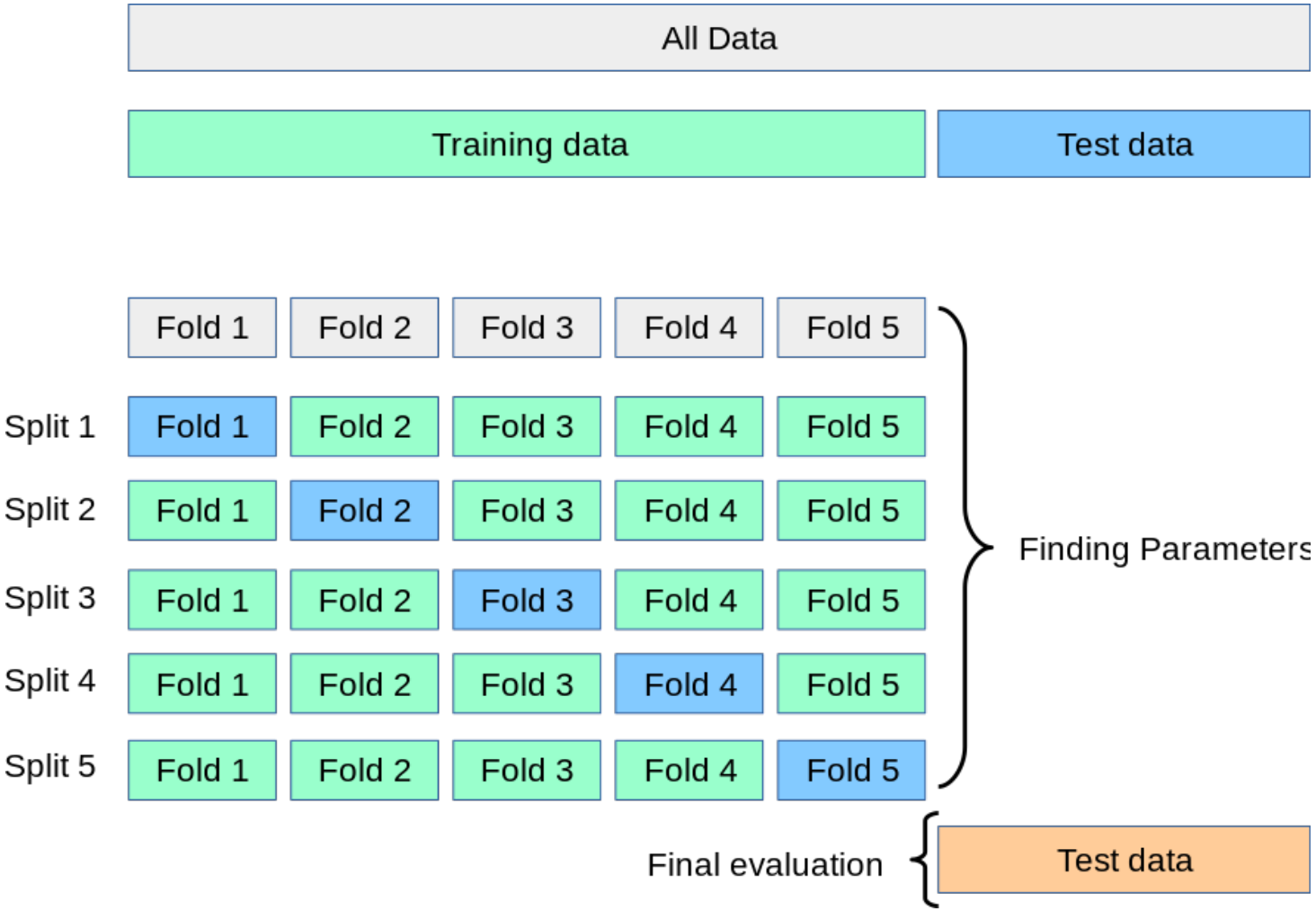
You can alternatively split the data into three sets:

- **Training Set**: Used for learning your model
- **Validation Set**: Used for selecting the hyper-parameters of your model
- **Test Set**: Used for assessing the final performance of your model

You should not do any further fine-tuning to increase your score on the test set! If the desired performance is not achieved, you need to go back and change your features, change your algorithm or collect more data!

Cross-validation and the train-validate-test approaches can be used with any ML method, not just kNN!

Combining



You can do further combinations, however at some point this becomes inefficient

It is already not feasible for deep learning

Nonparametric vs Parametric Methods

- Simple distinction for ML (see statistics sources for a better distinction):
  - Parametric: Uses a fixed number of parameters for getting an output
  - Non-parametric: Uses a flexible number of parameters for getting an output (name is misleading!)

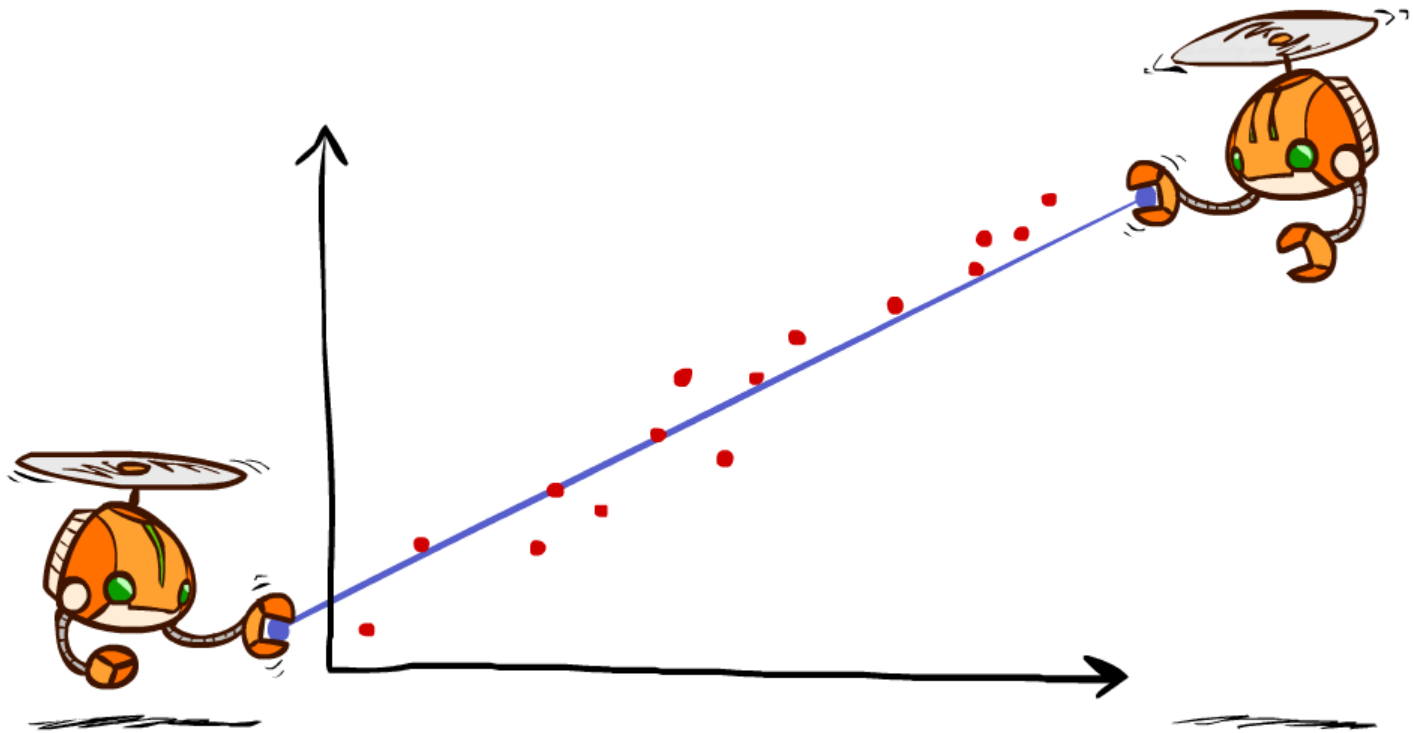
- In the kNN method, we use the entire dataset to decide on the output, thus it is a non-parametric method.

- In parametric methods, we chose a parameterized function, and learn its parameters from the training data.

Linear Regression

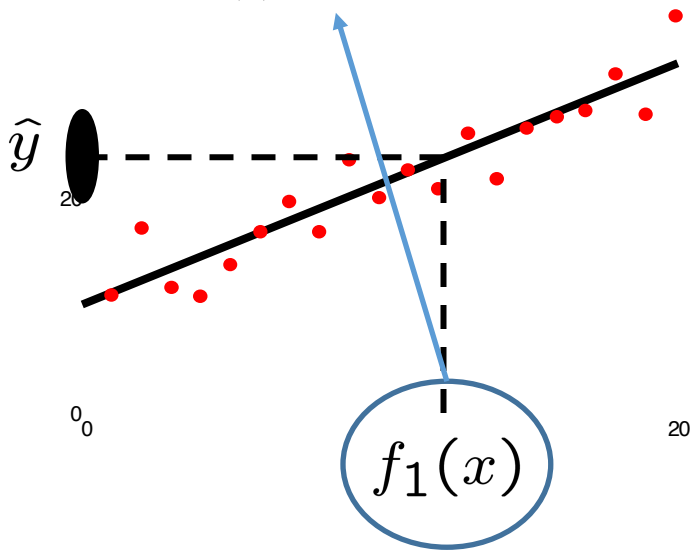
Given n data-label pairs $\{x_i, y_i\}$, find a function of the form $\hat{y} = w_1 x + w_0$

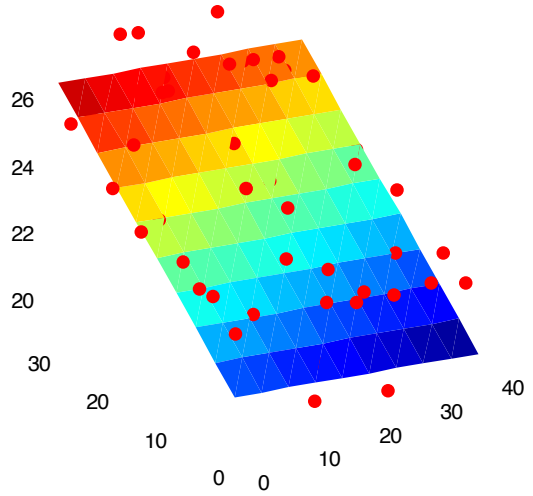
This function can then be used to explain the data or predict the values of future data



Multiple Linear Regression

e.g.
x: coin
$f_{1(x)}$: diameter





Prediction:

$$\widehat{y} = w_0 + w_1 f_1(x)$$

Prediction:

$$\widehat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$$

Can include many more features

Linear Regression

- Find parameters $w_0, w_1, \ldots w_d$ that **best** explains the data pairs $\{x_i, y_i\}$

$$\hat{y} = w_0 + w_1 f_1(x) + w_2 f_2(x) + \cdots + w_d f_d(x) = \sum_{k=0}^{d} w_k f_k(x) = w^T f(x)$$

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \ldots \\ w_d \end{bmatrix}, f(x) = \begin{bmatrix} f_{0(x)} \\ f_1(x) \\ \ldots \\ f_d(x) \end{bmatrix}, f_0(x) = 1$$

How to find w?

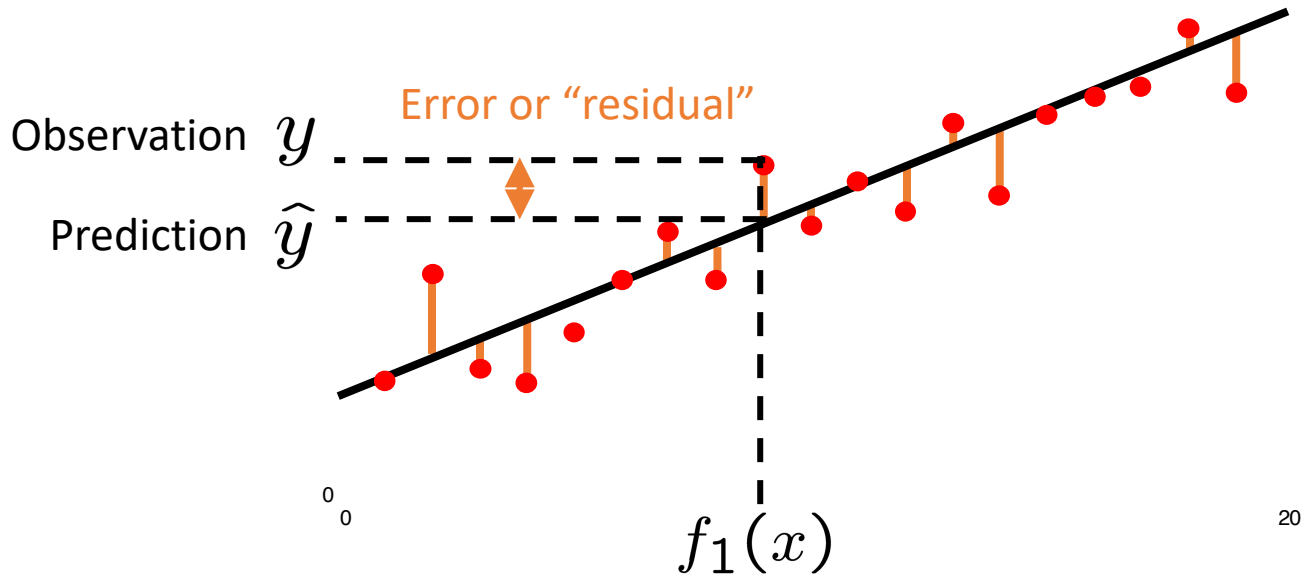
- Given data $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and the model $\hat{y} = w^T f(x)$:
- Find parameters w that **best** explains the data pairs $\{x_i, y_i\}$
- Idea of best: Minimize the difference between predicted y's ($\hat{y}$'s) and the given/observed y's
- Define total error:

$$total\ error = J(D, w) = \sum_{i=1}^{n} (y - \hat{y})^2 = \sum_{i=1}^{n} (y - w^T f(x))^2$$

- Find w that minimizes $\mathrm{J}(D, w)$

Least Squares Optimization

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



Differentiate the total error with respect to the parameters and equate to 0

LSO for Linear Regression

- Take the partial derivative of $J(D, w)$ with respect to each w_i and equate it to 0
- Plug in the input data and do some manipulation to get $\mathrm{X}w - Y = 0$, where

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}, f(x) = \begin{bmatrix} f_{0(x)} \\ f_1(x) \\ \vdots \\ f_d(x) \end{bmatrix}, f_0(x) = 1, Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} f^T(x_1) \\ \vdots \\ f^T(x_n) \end{bmatrix}$$

- 1's in X are for the bias term (w_0)
- X and Y come from the data. We want to find w such that $(\mathrm{X}w - Y)^T(\mathrm{X}w - Y)$ is minimized
- Home exercise: Why minimizing this is equivalent to minimizing $J(D, w)$?

One Way of Calculating w From the Data

- Differentiate $(\mathrm{X}w - Y)^T(\mathrm{X}w - Y)$ wrt w and equate to 0 (during the class)

$$\Rightarrow w = (X^T X)^{-1} X^T Y$$

- Where $X^+ = (X^T X)^{-1} X^T$ is called the pseudo-inverse of X
- Even though this is the mathematical form, do not ever use it in your numerical calculations. Use an existing function from a library or read up on Singular Value Decomposition and pseudo-inverse

Adding Nonlinearities?

- What if the relation between x and y is nonlinear?
- The answer is hidden in the formulation! $\hat{y} = w^T f(x)$
- The features can be nonlinear, even though the model is linear wrt w

- However, this puts the burden on feature extraction

What about smoothing?

- Large weights lead to overfitting, penalize the norm of the weights:

$$J(D, w) = \sum_{i=1}^{n} (y - w^T f(x))^2 + \lambda w^T w$$

- Resulting estimator:

$$w = (X^T X + \lambda \mathrm{I})^{-1} X^T Y$$

Where I is the identity matrix. (Derivation is similar as before with the hint $\lambda w = (\lambda \mathrm{I}) w$)

- This is called ridge regression or Tikhonov regularization ($w^T w = \left|\left|w\right|\right|_2^2$)
- There are other versions with different regularization terms. Some common ones
  - Lasso regression with $\lambda \left|\left|w\right|\right|_1$: Useful when the weights are desired to be sparse
  - Elastic-net regression with: $\lambda_1 \left|\left|w\right|\right|_1 + \lambda_2 \left|\left|w\right|\right|_2^2$

Notes on Linear Regression

- Old and widely used method
- Take a statistics class to dive deeper into regression, very useful
  - Unfortunately, we do not have the time

- Cross-validation can help in choosing features as well as regularization hyperparameters (How?)

- The weight regularization idea is common in other parametric methods as well

What to do if f(x,w) is nonlinear in w?

- Hope that there is a closed form solution to the least-squares?
- Optimization! Minimize the error iteratively.
- Example you have probably heard about: **Gradient Descent**
- General objective with regularization:

$$J(D, w) = \sum_{i=1}^{n} \ell_D\left(y, f(x, w)\right) + \lambda \ell_w(w)$$

- ℓ_D data dependent "loss", e.g. squared error
- ℓ_w weight regularization, e.g. L2-norm
- Assuming ℓ's are differentiable, we can apply gradient descent
- We can add other terms as well (and we do in practice)

Gradient Descent/Ascent

- Local Search on continuous state spaces
- State x: multivariate and continuous
- Objective function J(x): Differentiable around x (e.g. x is the weight vector)

- **Idea**: Move x in the direction of decreasing/increasing f
- **How**: Derivatives!

- Need to calculate the **gradient:**

$$\nabla J(x) = \left(\frac{\partial J(x)}{\partial x_1}, \ldots, \frac{\partial J(x)}{\partial x_d}\right)$$

- Can be computed *analytically* or *numerically*

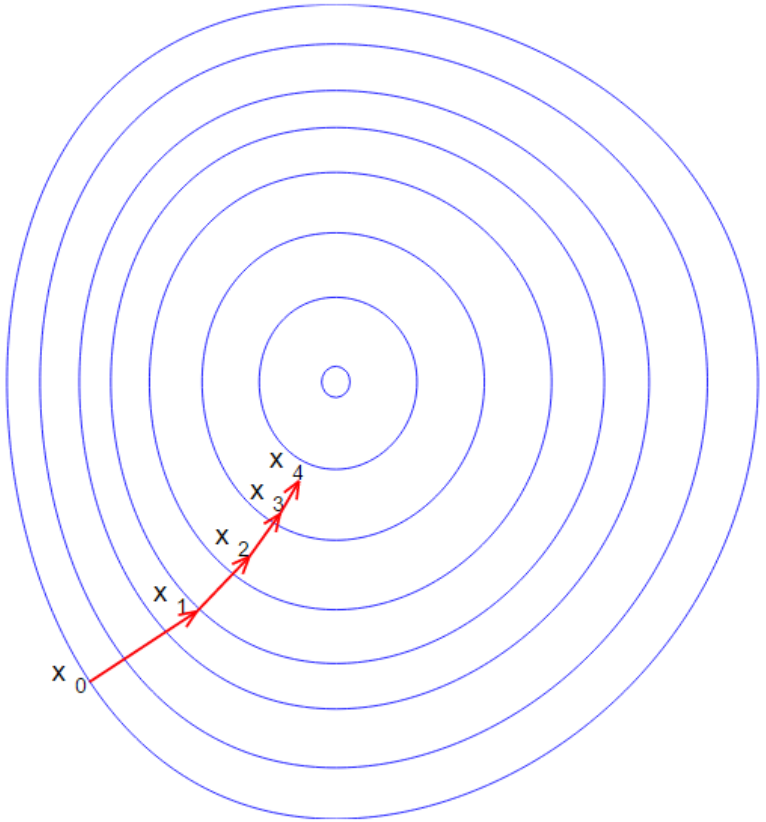
Gradient Descent/Ascent

- Implementation:

While SomeCondition:

$$x(t+1) = x(t) \pm \alpha \nabla J(x)$$

- SomeCondition:
  - Maximum iterations
  - |x(t+1) –x(t)| < small
  - …



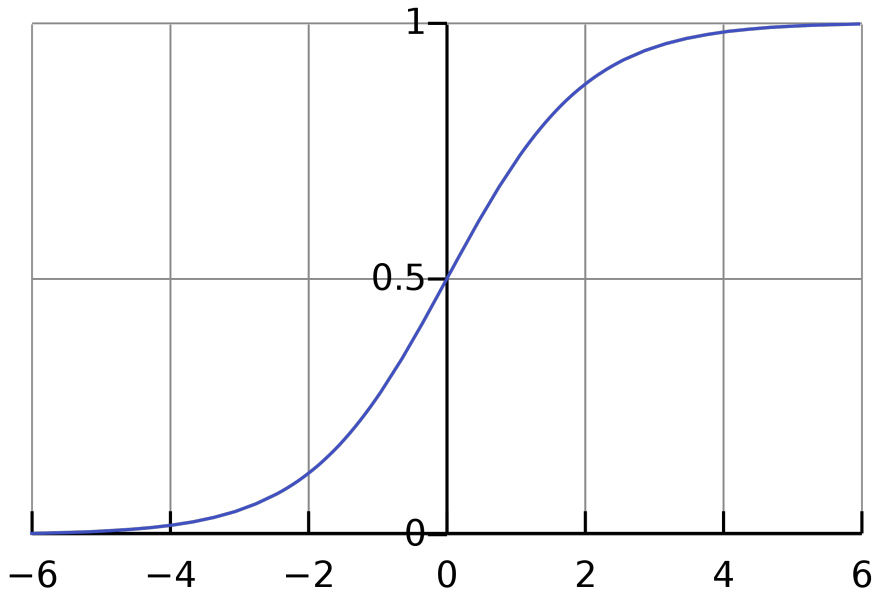
- More complicated versions exists e.g., adaptive step size, momentum, calculating higher order derivatives (see *Newton-Raphson*) etc.

Brainstorm

- How to use linear regression for (binary) classification? (Logistic Regression)

Sigmoid and Logistic Functions

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



$$g(x) = \frac{1}{1 + e^{-(w_o + w_1 x)}}$$

$$x = -\frac{w_0}{w_1} \rightarrow g(x) = 0.5$$

w_1 affects the “slope”

Logistic Regression

- Target: 0 or 1
- Model: $\mathrm{g}(f(x)) = \frac{1}{1+e^{-(w^t f(x))}}$ (sometimes interpreted as $\mathrm{P}(y=1|x)$)
- Loss as the log-loss: (see the entropy concept within information theory)

$$\ell(y_i, f(x_i)) = -y_i \ln\big(g(f(x_i))\big) - (1-y_i)\ln(1-g(f(x_i)))$$

- Learning: Gradient descent on the total log-loss for w
- Can apply regularization as well

A few Words on Gradient Descent/Ascent

- This family of algorithms is used in **a lot of** applications!
- Very simple to code (but modifications may not be so much)
- Works in any number of dimensions
  - Even in infinite-dimensions, just need to change the derivative
- Inefficient
- Some state-space landscapes wreak havoc
- Diminishing or blowing gradients in higher-dimensions

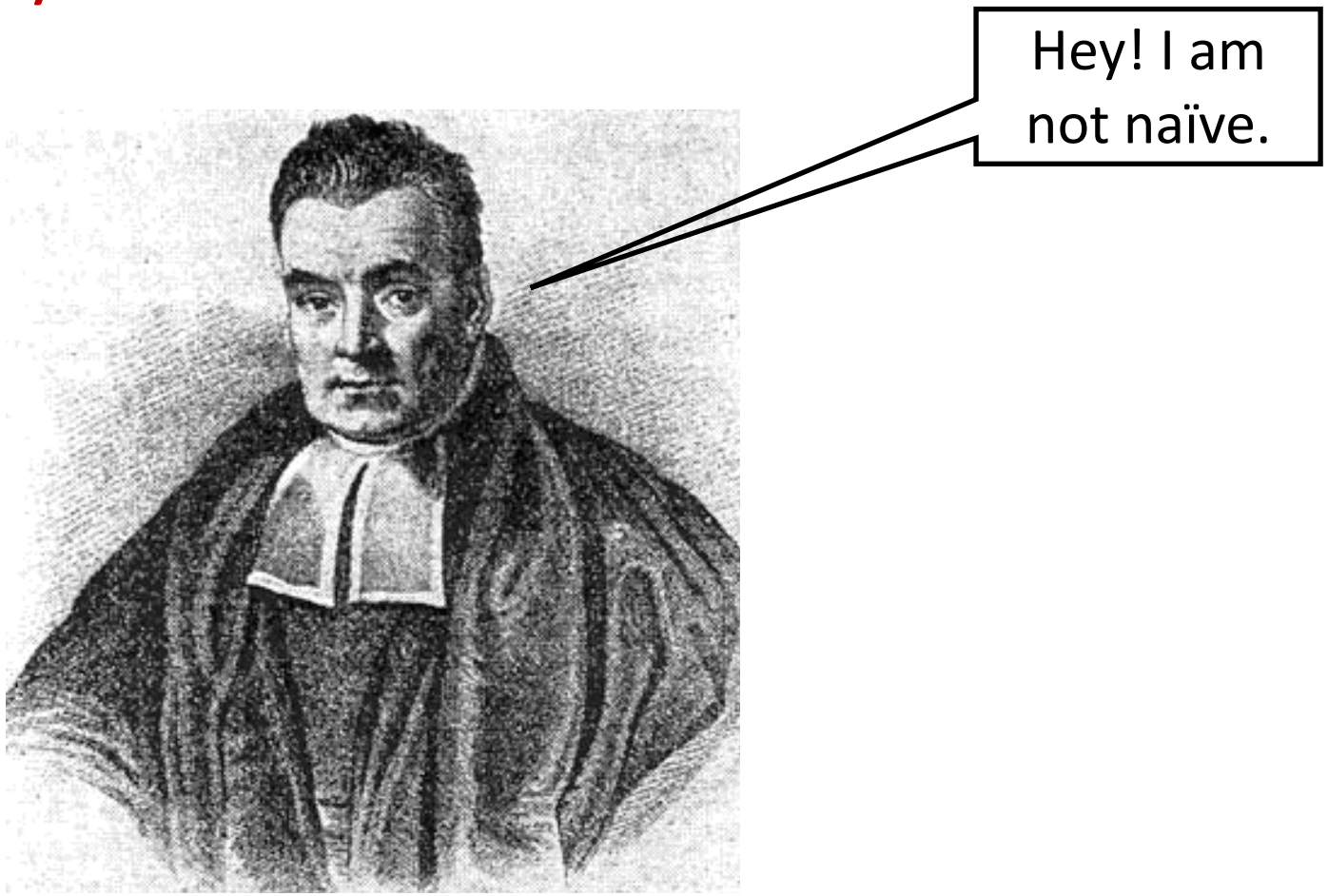
- In any case, a good first tool to tackle an applicable problem!

Discriminative vs Generative Methods

- Recall
  - Input: $\{x_1, x_2, \dots, x_n\}, x_i \in X$
  - Output: $\{y_1, y_2, \dots, y_n\}, y_i \in Y$
- Classification: $\text{argmax}_y P(y|x)$, i.e., most likely output given the input
- Discriminative: When we learn $P(y|x)$
  - Directly plug-in!
- Generative: When we can learn $P(y, x)$ or $P(x|y)$
  - Can "generate" new data given label!

  $P(y|x) = \frac{P(y,x)}{P(x)} = \frac{P(x|y)P(y)}{P(x)} \propto P(x|y)P(y)$, ($P(x)$ is the same for all y!)
  - Plug-in whichever in whichever format you have in the classification equation

Naïve Bayes



The slides with an asterisk (*) will not be mathematically included in the exams.
However, True/False or other open-ended questions may require you to know the relevant ideas.

Naïve Bayes - Introduction

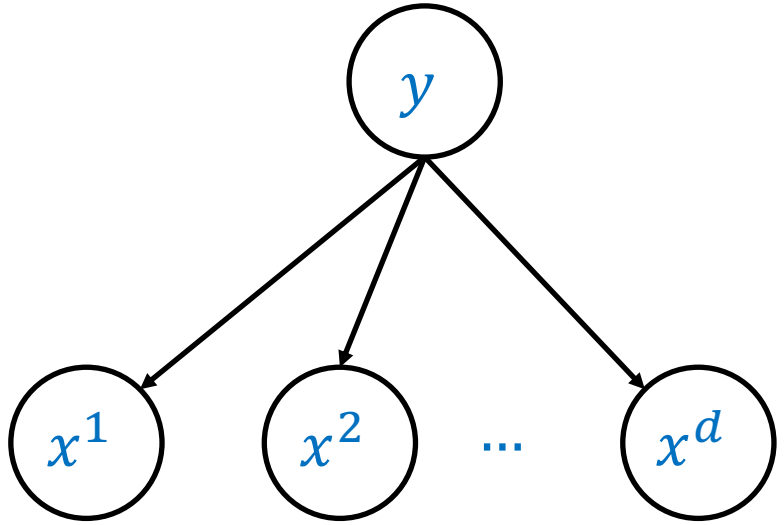
- The input space is multi-dimensional. E.g.
  - Classification: Image features like color histogram, edge histogram etc. Output: Cat, dog, robot
  - Regression –Housing Prices: Coordinates as input, price as output

- Idea: Use a BN to represent the relationship between inputs and the output
- Issues:
  - What is the structure?
  - How do we learn its parameters?

Naïve Bayes

- Idea: Use a BN to represent the relationship between inputs and the output
- Assume that all the features/attributes (i.e. the dimensions of the input) are independent effects of the output

$$x = [x^1, x^2, \dots, x^d]^T$$



Where the input is d dimensional

- Naïve Bayes Model:

$$P(y|x) = P\left(y\middle|x^1, x^2, \dots, x^d\right) = \propto P(y)\prod_{j=1}^{d} P(x^j|y)$$

- Structure is assumed, what is to be learned?

Naïve Bayes

$$P(f(x), x) = P(f(x), x_1, x_2, .., x_d) = P(f(x)) \prod_{i=1} P(x_i | f(x))$$

- Specify how each feature depends on its class
  - E.g. P(Trait = Selfish | Type = Cat) = 0.9, P(Trait = Super Smart | Type = Robot) = 0.99
  - E.g. P(Locomotion = Legged | Type = Cat) = 1.0, P(Locomotion = Wheeled | Type = Robot) = 0.8
- Need to learn $O(d)$ values – why?
- Model is simple and the assumption is often wrong
  - E.g. Input: Code grade and Report grade, Output: Final Project Grade
- Yet it still works!

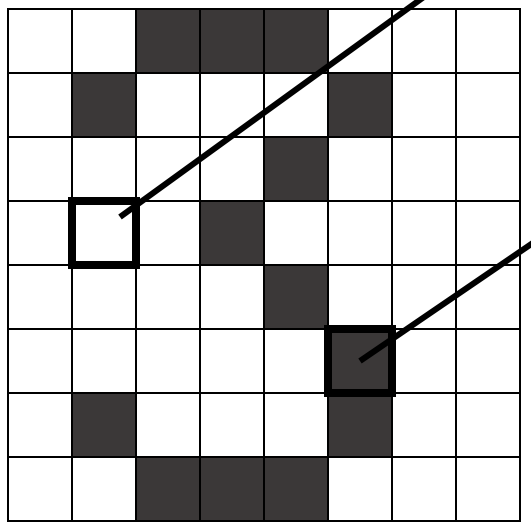
General Naïve Bayes

- What do we need in order to use Naïve Bayes?
  - Inference method (we know this part!)
    - Start with a bunch of probabilities: P(Y) and the $P(F_i|Y)$ tables
    - Use standard inference to compute $P(Y|F_1 \ldots F_n)$
    - Nothing new here
  - Estimates of local conditional probability tables
    - P(Y), the prior over labels
    - $P(F_i|Y)$ for each feature (evidence variable)
    - These probabilities are collectively called the *parameters* of the model and denoted by θ
    - Up until now, we assumed these appeared by magic, but…
    - …they typically come from training data counts: we'll look at this soon

Example Digit Recognition

$P(Y)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



$P(F_{3,1} = on|Y)$

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

$P(F_{5,5} = on|Y)$

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

Input: image pixels with binary values, 8 x 8

Learning Parameters

- Maximum Likelihood Estimate of a Random Variable

$$P(x) = \frac{count(x)}{total\ samples} \qquad P(x|y) = \frac{\boxed{count(x,y)}}{count(y)}$$

Number of x and y seen together

- Note that you can also have a probability function, e.g., Gaussian
- In that case, you estimate its parameters using the data

Parameter Estimation for a Known Distribution

- Given a distribution $P(x, w)$ and independently and identically distributed (iid) data points $\mathrm{D} = \{x_1, \ldots, x_n\}$
- Maximize the likelihood:

$$L(D, w) = \prod_{i=1}^{n} P(x_i, w)$$

- Or equivalently minimize

$$\ln\left(\prod_{i=1}^{n} P(x_i, w)\right) = \sum_{i=1}^{n} \ln(P(x_i, w))$$

- The equations from the previous slides can be derived this way!

Parameter Estimation for a Gaussian*

$$P(x) = N(\mu, \sigma) = \frac{1}{\sqrt{(2\sigma^2\pi)}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Take its natural logarithm, then differentiate wrt μ, σ then equate to 0

$$\ln(P(x)) = \ln\left(\frac{1}{\sqrt{(2\sigma^2\pi)}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) = -\frac{1}{2}\ln(2\pi) - \ln(\sigma) - \frac{(x-\mu)^2}{2\sigma^2}$$

- Differentiate wrt μ and equate to 0 for all the data:

$$\sum_{i=1}^{n} \frac{x_i - \mu}{\sigma^2} = 0$$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Parameter Estimation for a Gaussian*

$$\ln(P(x)) = \ln\left(\frac{1}{\sqrt{(2\sigma^2\pi)}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) = -\frac{1}{2}\ln(2\pi) - \ln(\sigma) - \frac{(x-\mu)^2}{2\sigma^2}$$

- Differentiate wrt σ and equate to 0 for all the data:

$$\sum_{i=1}^{n} -\frac{1}{\sigma} + \frac{(x_i-\mu)^2}{\sigma^3} = -\frac{n}{\sigma} + \sum_{i=1}^{n} \frac{(x_i-\mu)^2}{\sigma^3}$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2$$

- This assumes the prior on μ and σ, is uniform! Let $\mathrm{w} = [\mu, \sigma]$
  - We have done $w = argmax(P(D|w))$, same as $w = argmax(P(w|D))$, with uniform prior
  - If we had a prior: $w = argmax\big(P(w|D)\big) = argmax(P(D|w)P(w))$!

Learning Parameters

- Maximum Likelihood Estimate of a Random Variable

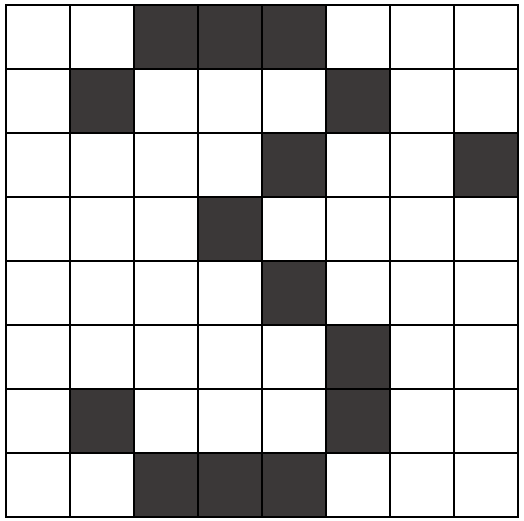
$$P(x) = \frac{count(x)}{total\ samples} \qquad P(x|y) = \frac{count(x,y)}{count(y)}$$

Number of x and y seen together

- Note that you can also have a probability function, e.g., Gaussian
- In that case, you estimate its parameters using the data
  - $P(x|y) = N(\mu(y), \Sigma(y))$

Generalization and Overfitting

- Relative frequency parameters will overfit the training data!
- This approach will assign 0 probability to unseen events
  - E.g. Digit recognition: Due to noise some pixels may have an erroneous reading
  - E.g. spam filtering: Many words, some might not be in the training set!



- To generalize better: we need to smooth or regularize the estimates

- What if we assume that some events are possible, even if we do not have them in our training data set?

Extended Laplace Smoothing*

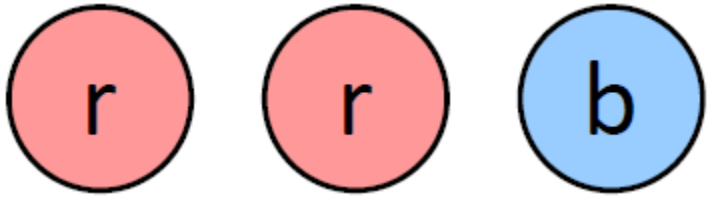
- MLE for a discreet random variable X, where c denotes the count, p_j denotes the values that it can take and n denotes the total number of samples:

$$P(X = p_j) = \frac{c\ (X = p_j)}{n}, j = 1 \dots r$$

- Laplace Smoothing: Each event has a prior of happening k times. Then:

$$P(X = p_j) = \frac{c\ (X = p_j) + k}{n + kr}$$

- k is the strength of the prior
  - $k = 0$, frequency based estimation (overfit)
  - $k = \infty$, uniform probability, no effect of the data (underfit)



$$P_{LAP,0}(X) = (\frac{2}{3}, \frac{1}{3})$$

$$P_{LAP,1}(X) = (\frac{3}{5}, \frac{2}{5})$$

$$P_{LAP,100}(X) = (\frac{102}{203}, \frac{101}{203})$$

How does this affect conditional probabilities?

Linear Interpolation Smoothing*

- Laplace Smoothing performs poorly for conditionals $P(X|Y)$
  - When $|X| = r$ is large and/or when $|Y| = s$ is large, empirical probabilities are affected too much
- Linear Interpolation
  - Get empirical $P(X)$ from data
  - Make sure $P(X|Y)$ isn't too different from the empirical $P(X)$ (attributing some of the prior to the conditional even if not observed)

$$P_{LIN}(x|y) = \alpha P(x|y) + (1 - \alpha)P(x)$$

- How would this behave for $\alpha = 1$ and $\alpha = 0$?
- There are even better ways! That's for another class though…

Notes on NB

- Another simple ML approach
- The naïve Bayes assumption takes all features to be independent given the class label. Often wrong but it still works!

$$P(y|x) = P\left(y\middle|x^1, x^2, \dots, x^d\right) = \propto P(y)\prod_{j=1}^{d} P(x^j|y)$$

- Can learn $P(y)$ and $P\left(x^j\middle|y\right)$, separately
- There are relaxation methods to help with overfitting

More on “Generative Models”

- Models that can generate/sample the data that were trained on!
- Supervised case (Given right before the NB slides)
- Unsupervised example: Fitting a distribution to data P(x)
  - For example fitting a Gaussian Mixture Model to the data
- Series Forecasting: Given a history of input, what is the next value? (e.g. stock price prediction, next word prediction) – Similar to discriminitive learning in methods but application is generative
- Example: BNs, HMMs as Generative Models

What about Large Language Models?

$$P(x_t | x_{t-w:t-1}, \theta)$$

- Where:
  - x_t: next token to generate (tokens are usually words or “sub-words”)
  - $x_{t-w:t-1}$: Previous tokens (either generated or given, incase of “conversational models” both, can also have additional information)
  - θ: Parameters, representing the model (was dropped in the previous slides)
- This is why some people call LLMs “statistical parrots”
- Looks simple but …

- Multi-Modal Models: Multiple types of inputs and outputs

Intentionally Left Empty

Relaxation/Smoothing/Regularization

- The relaxation methods and ideas are general and can be applicable to other methods
- Such as putting priors on parameter estimations
  - E.g. as we have seen a glimpse of in the Gaussian Distribution case (in the slides)
  - E.g. small parameters for basis function regression (see Tikhonov Regularization or ridge regression)

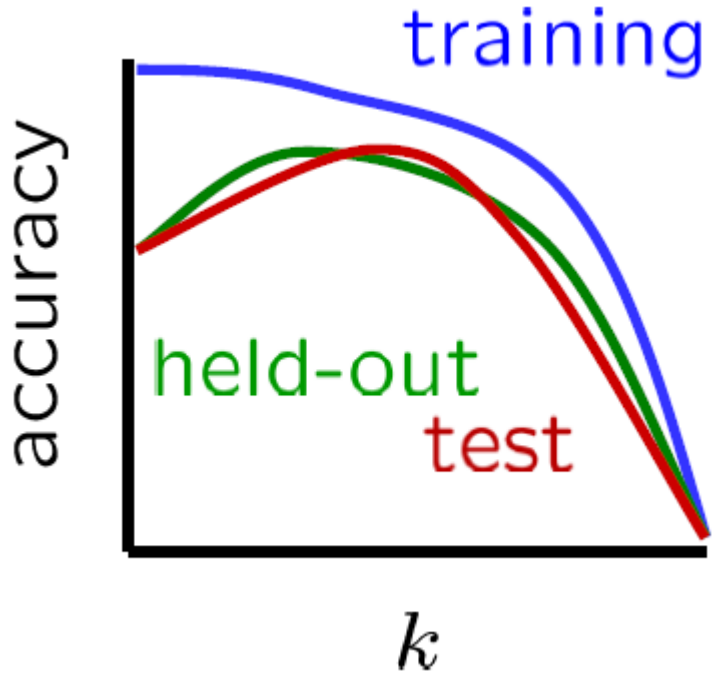
$$J(D) = \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda w^T w$$

  - E.g. Google Dirichlet Distribution (for latent variable models)

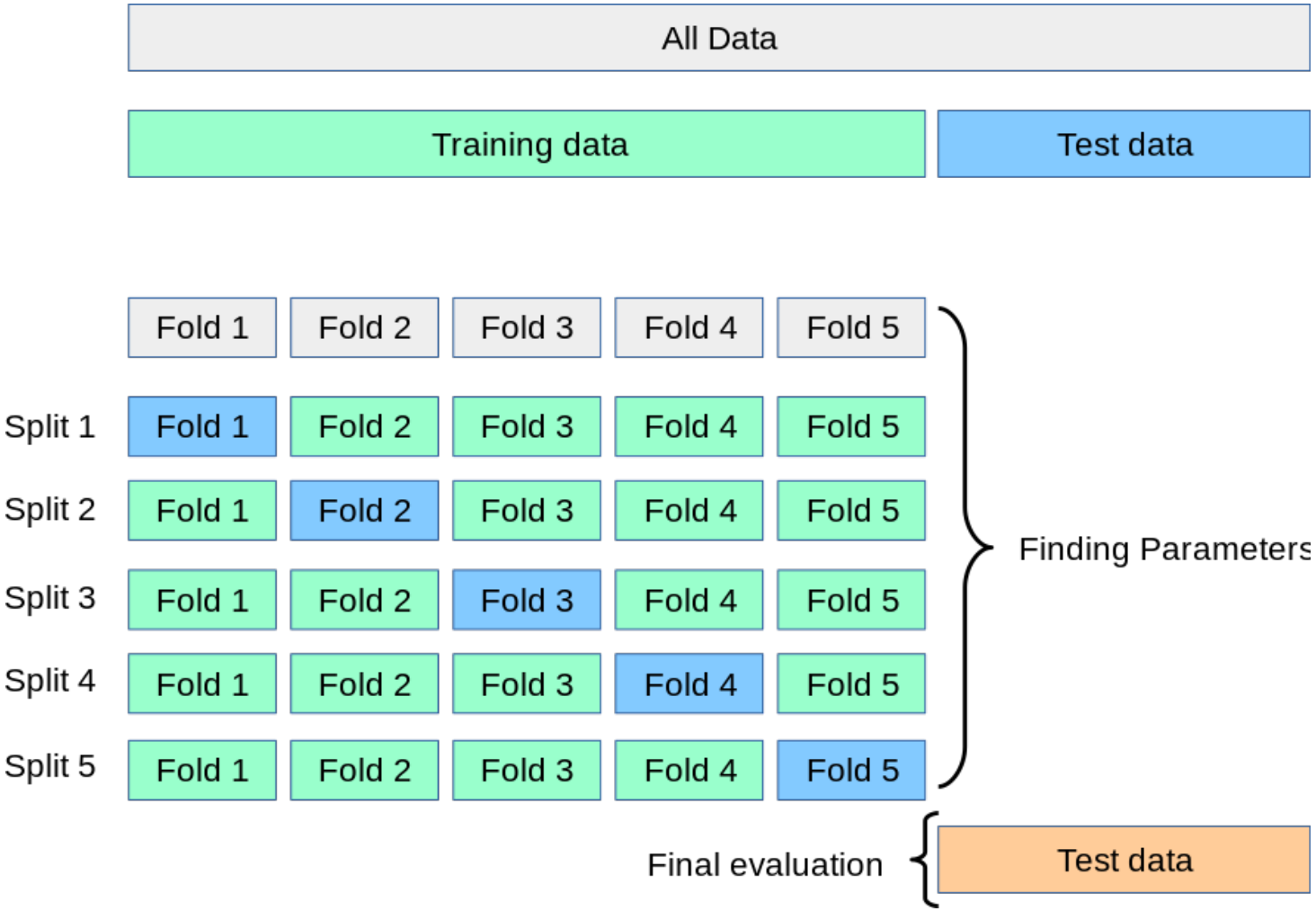
- Why did we want this? –Overfitting!
- How do we decide on the smoothing parameters?
  - Priors, smoothing values, regularization parameters etc.

Tuning on Validation Data

- Two kinds of unknowns
  - Parameters (if parametric): e.g. the probabilities $P(X|Y), P(Y)$
  - Hyperparameters: e.g. the amount / type of smoothing to do, k, α
- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data –Why?
  - For each value of the hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data
- Take it a step further, do cross-validation at each step! Not always justifiable with amount of data and time



Repeating:

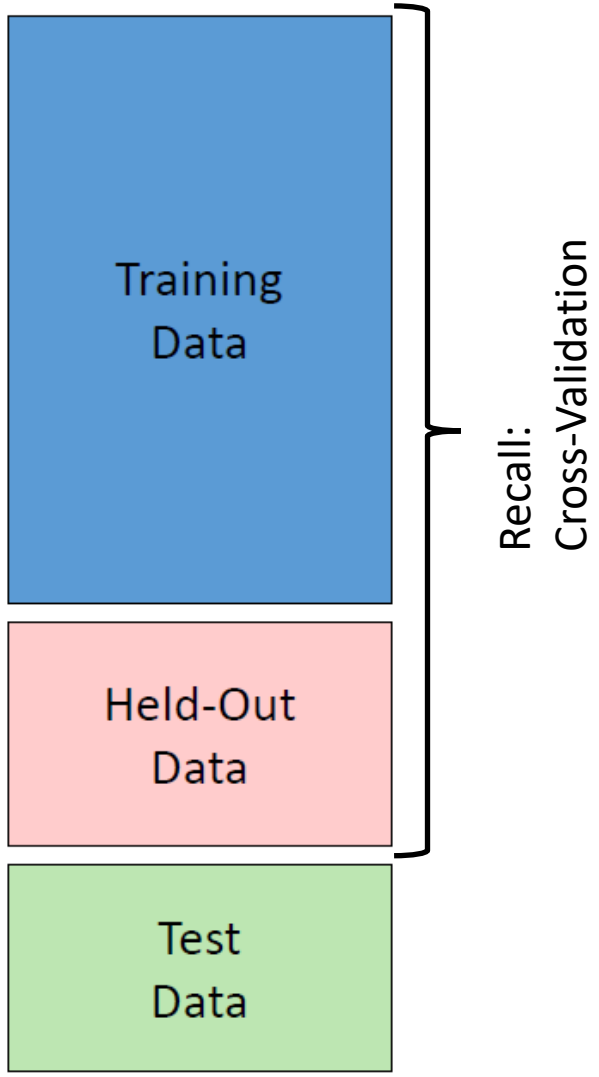


You can do further combinations, however at some point this becomes inefficient

It is already not feasible for deep learning

Supervised Training

- Data: labeled instances, divide it up
  - Training set
  - Held out set or Validation Set
  - Test set
- Features: attributes which characterize each x
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - Tune hyperparameters on held-out (validation) set
  - Compute performance of test set
  - Very important: never "peek" at the test set!
- Performance Evaluation
  - Accuracy: fraction of instances predicted correctly
  - Total Error: Total Difference between targets and predictions
  - Log-likelihood of the data
  - There are others, usually referred to as the "loss function"
- Overfitting and generalization
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well



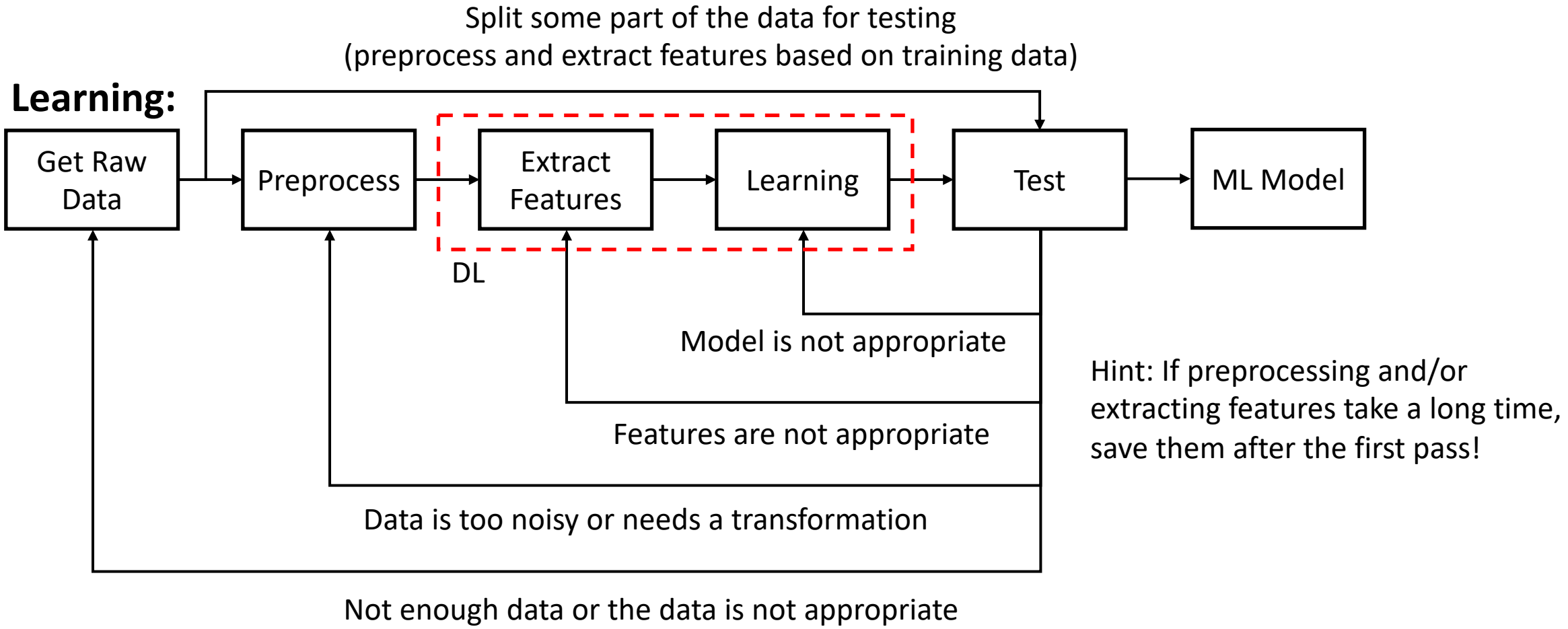
Baselines

- First step: get a baseline
  - Baselines are very simple “straw man” procedures
  - Help determine how hard the task is
  - Help know what a “good” accuracy is
- Weakest Baseline: Random selection
  - If you cannot do better than random, either the problem is random or you should give up ML
- Very Weak baseline: most frequent label classifier
  - Gives all test instances whatever label was most common in the training set
- Other Simple Baselines:
  - kNN: if you can calculate distances
  - Decision Trees
  - Naïve Bayes
  - Multivariate Gaussians: if continuous input
  - Linear Regression

- Established Baselines: (Widely Used Methods)
  - SVMs (getting a bit old now)
  - Random Forests and Gradient Boosted Trees
  - Gaussian Processes
  - MLPs
- For real research, usually use previous work as a (strong) baseline

- Note that you should do your best (e.g. hyper-parameter selection) for your baseline as well!

Simplified ML Pipelines



Inference: