Introduction, course logistics

CS771: Introduction to Machine Learning

Nisheeth Srivastava

Course Logistics

- Course Name: Introduction to Machine Learning – CS771
  - An introductory course – supposed to be your first intro to the subject
- 3 lectures every week on Zoom, MWF 1800-1915 (link)
  - Classes will be recorded and uploaded to mooKIT for later viewing
  - mooKIT URL: https://hello.iitk.ac.in/cs771a/ (CC id and password to be used for login)
- Friday lecture will double as an interactive discussion session
- All material will be posted on the mooKIT page for the course
- Q/A and announcements on mooKIT

Course Team

- Abhas (ababhas@cse.iitk.ac.in)
- Abhishek Jaiswal (abhijais@iitk.ac.in)
- Abhishek Krishna (krishnacs20@iitk.ac.in)
- Avideep (avideep@iitk.ac.in)
- Dhanajit (dhanajit@iitk.ac.in)
- Jay (jvora20@iitk.ac.in)

Course Team

- Mahesh (maheshak@cse.iitk.ac.in)
- Mosab (mosab@iitk.ac.in)
- Neeraj (neermat@cse.iitk.ac.in)
- Rahul (rsharma@iitk.ac.in)
- Yatin (yatind@iitk.ac.in)
- Nisheeth (nsrivast@cse.iitk.ac.in)

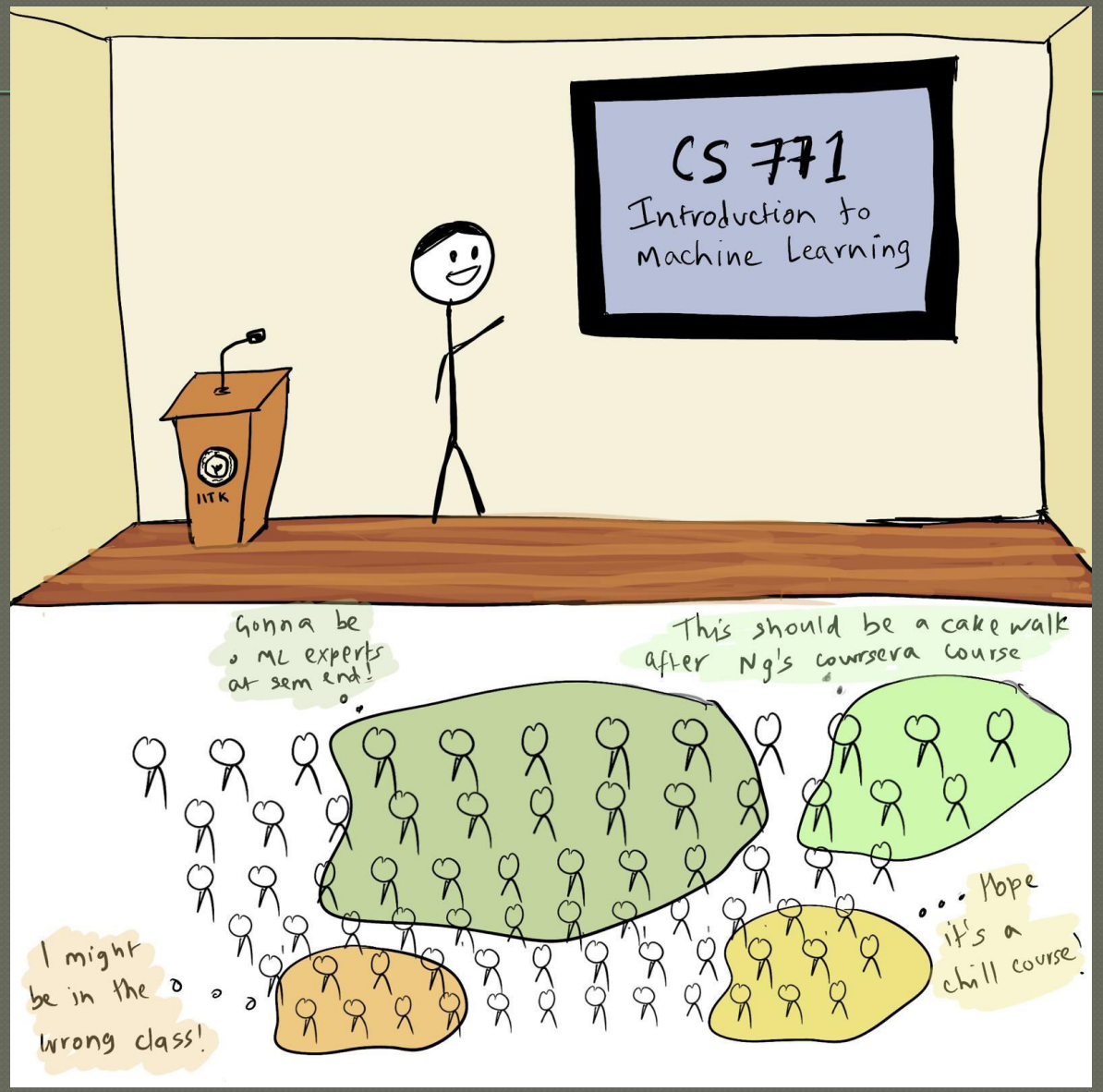
Workload and Grading Policy

- 4 homework assignments (theory + programming) worth 60%
  - Theory part: Derivations/analysis
  - Programming part: Implement/use ML algos, analysis of results. Must be done in Python (learn if not already familiar)
  - Must be typeset in LaTeX (learn if not already familiar)
  - To be submitted via Gradescope (login details will be provided)
  - Will also involve a viva
- Quizzes and exams (mid-sem and end-sem) worth 40%
  - Will be held online – details later
  - Quizzes will be worth 5% of the total grade
  - Mid-sem and end-sem will be worth 10, 20% each of the total gra

Reference materials

- Course slides and notebooks will mostly be sourced from previous iterations of CS771
- No fixed textbook
  - Sections from Bishop, Murphy, Stork, Hart & Duda will be assigned as reading from time to time
- Different books vary in terms of
  - Set of topics covered
  - Flavor (e.g., classical statistics, deep learning, probabilistic/Bayesian, theory)
  - Terminology and notation (beware of this especially)

Course Goals



Credit: Rishika Agarwal (EE, graduated 2017)

Course Goals..

- Introduction to the foundations of machine learning models
- Focus on developing the ability to
  - Understand the underlying principles behind ML models and algos
  - Understand how to implement and evaluate them
  - Understand/develop intuition on choosing the right ML model/algo for your problem
- Focus on how to interpret ML algorithms' outputs in a rigorous manner
  - Over-optimistic interpretation of ML has led to a reproducibility crisis in the field
  - Graduates of this course should be able to separate truth from falsehood (and BS) in ML evaluations

- Not an introduction to popular software frameworks and libraries, such as scikit-learn, PyTorch, Tensorflow, etc
  - Can explore once you have some understanding of various ML techniques

What is Machine Learning

How do we classify?

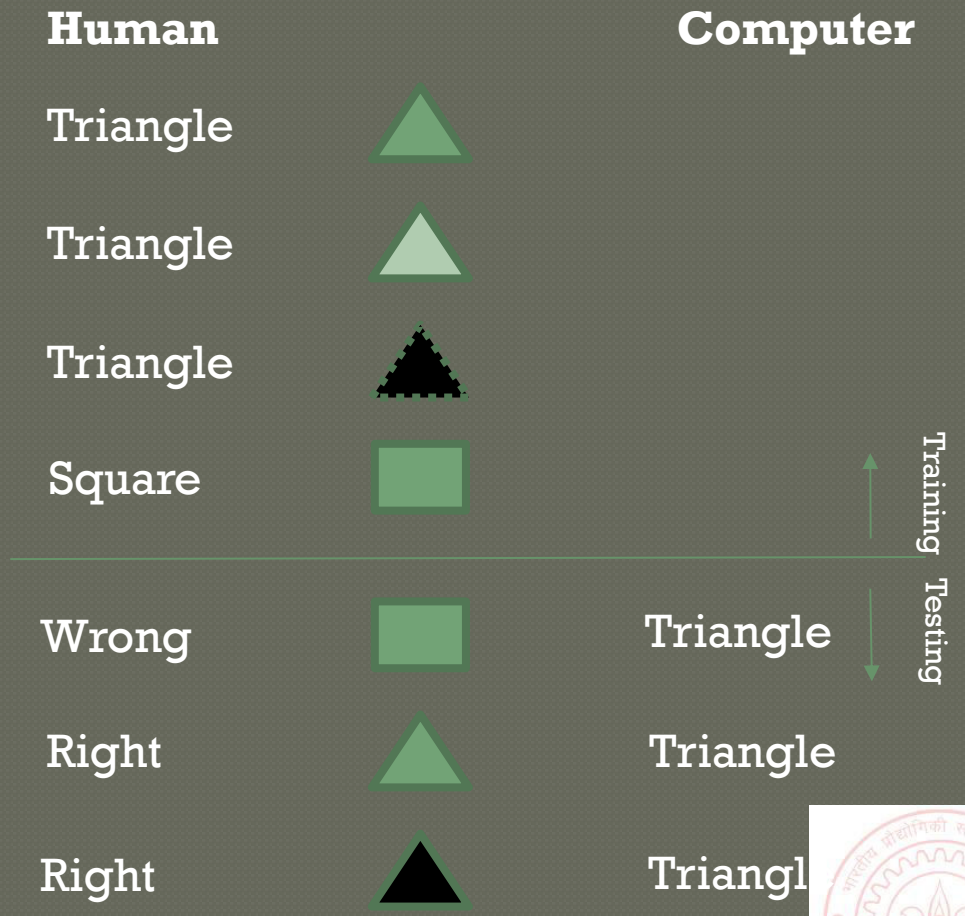
- Let's say we want to classify the climate in a region
- We may say something like
  - If the summer temperature is above 35 C for more than 20 days during May-July, the climate is tropical
  - If the total rainfall is less than 10 cm all year, the climate is desert
- Basic principles
  - Think of simple rules that place thresholds on some measurable attributes of the object
  - Combine rules to maximize coverage of classification

Classification using expert advice

- Who should be screened for COVID-19?
- Attributes
  1. History of international travel
  2. Flu-like symptoms
  3. Contact with anyone reporting 1 or 2
- Advantages:
  - Simple to understand and critique
- Disadvantages:
  - Hard to implement without data and actuators

Supervised classification using machine learning

- Instead of asking experts to generate rules
  - Ask them to say 'wrong' or 'right' whenever a computer program classifies something
- Program is programmed to want to make as few mistakes as possible
- Revises its internal rules such that the expert has to say 'wrong' less often
- Rules are *learned* using expert supervision

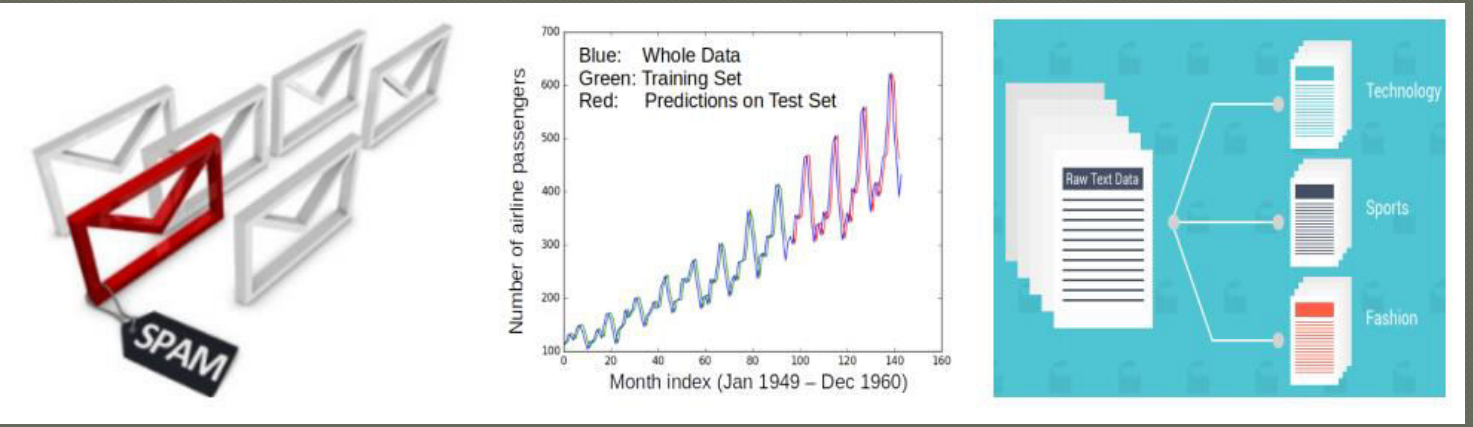


Advantages of ML classification

- Rules are learned from data instead of generated by experts so
  - Rules have a lower chance of being contaminated by an expert's bias
  - Rules are not restricted in complexity
  - Rules are more likely to be testable
  - Rules are likely to have greater predictive accuracy
- But, for the same reason,
  - Rules have a higher chance of missing important aspects of the situation not captured within data
  - Rules can become over-complex and hard to interpret
  - Rules become less likely to generalize well to new situations

Machine Learning (ML)

- Designing algorithms that use data to programmatically learn a model of it
- The learned model can be used to
  - Detect patterns/structures/themes/trends etc. in the data
  - Make predictions about future data and make decisions

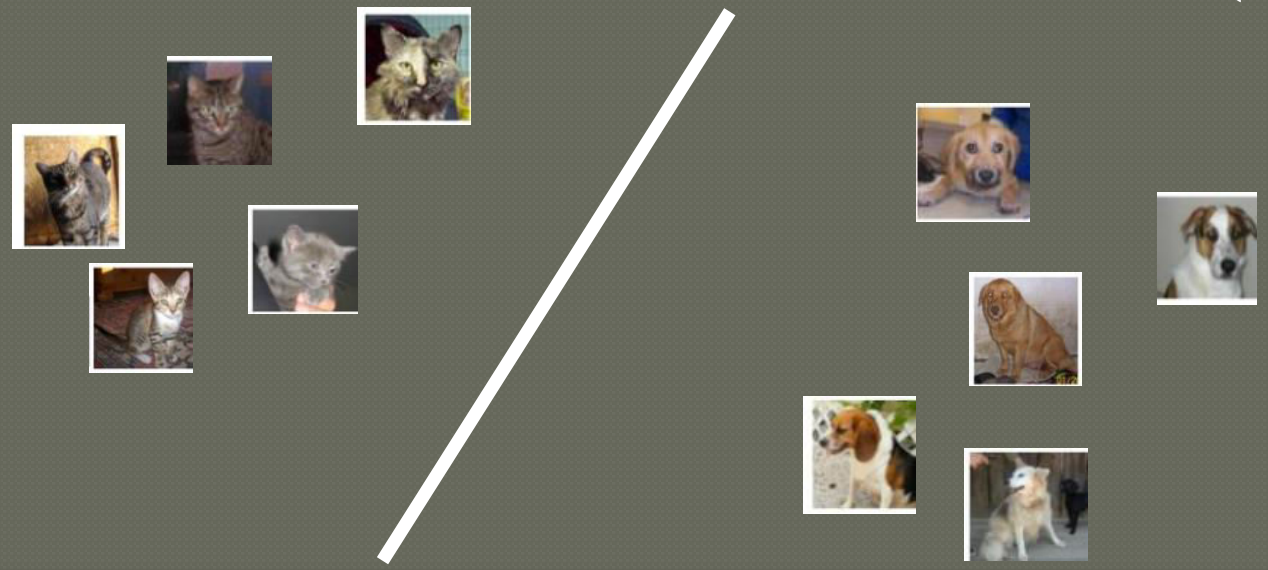


- Modern ML algorithms are heavily "data-driven"
  - No need to pre-define and hard-code all the rules (infeasible/impossible anyway)
  - The rules are not "static"; can adapt as the ML algo ingests more and more data

ML: From What It Does to How It Does It?

- ML enables intelligent systems to be data-driven rather than rule-driven
- How: By supplying training data and building statistical models of data
- Pictorial illustration of an ML model for binary classification:

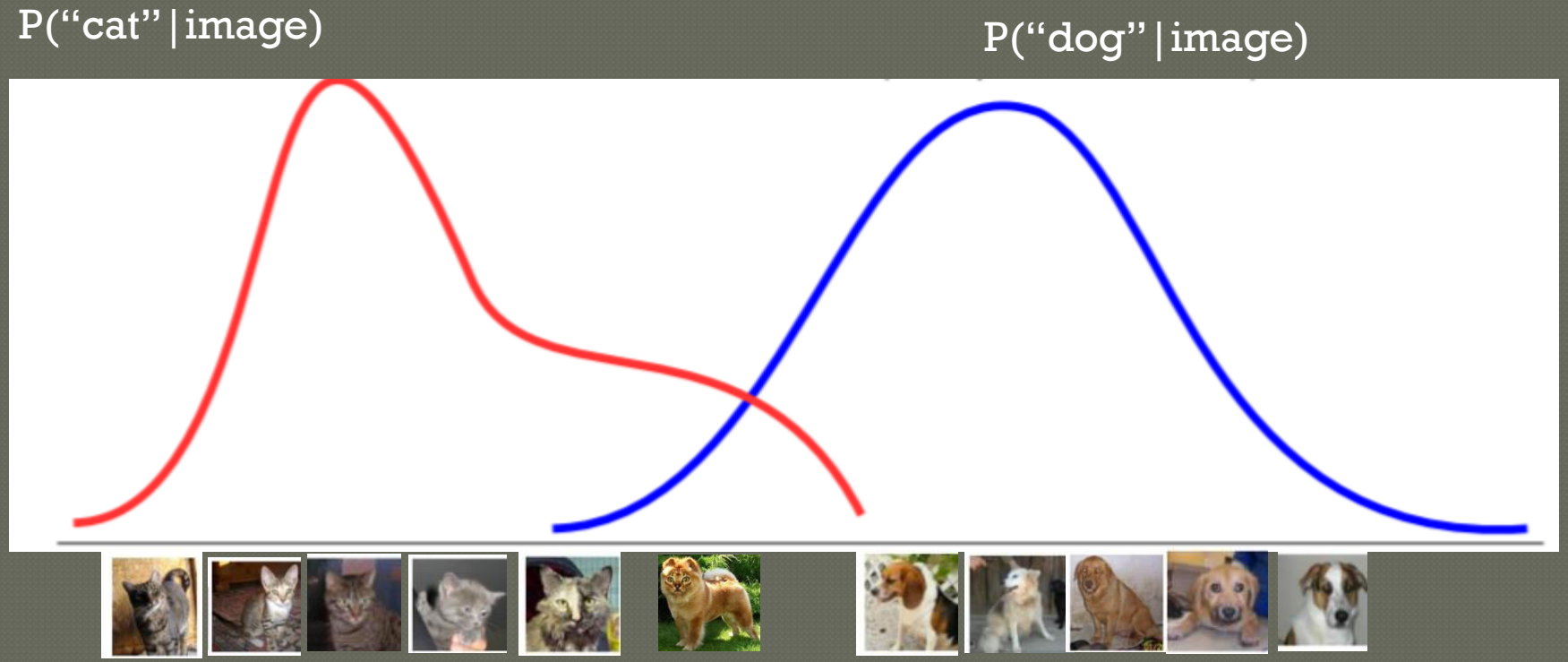
A Linear Classifier (the statistical model)



ML: From What It Does to How It Does It?

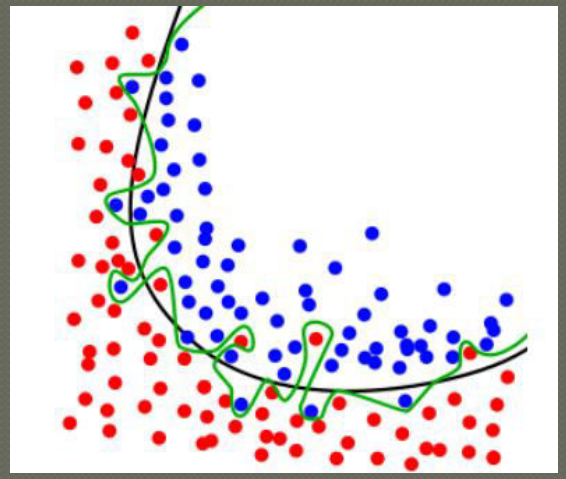
- ML enables intelligent systems to be data-driven rather than rule-driven
- How: By supplying training data and building statistical models of data
- Pictorial illustration of an ML model for binary classification:

A Probabilistic Classifier (the statistical model)



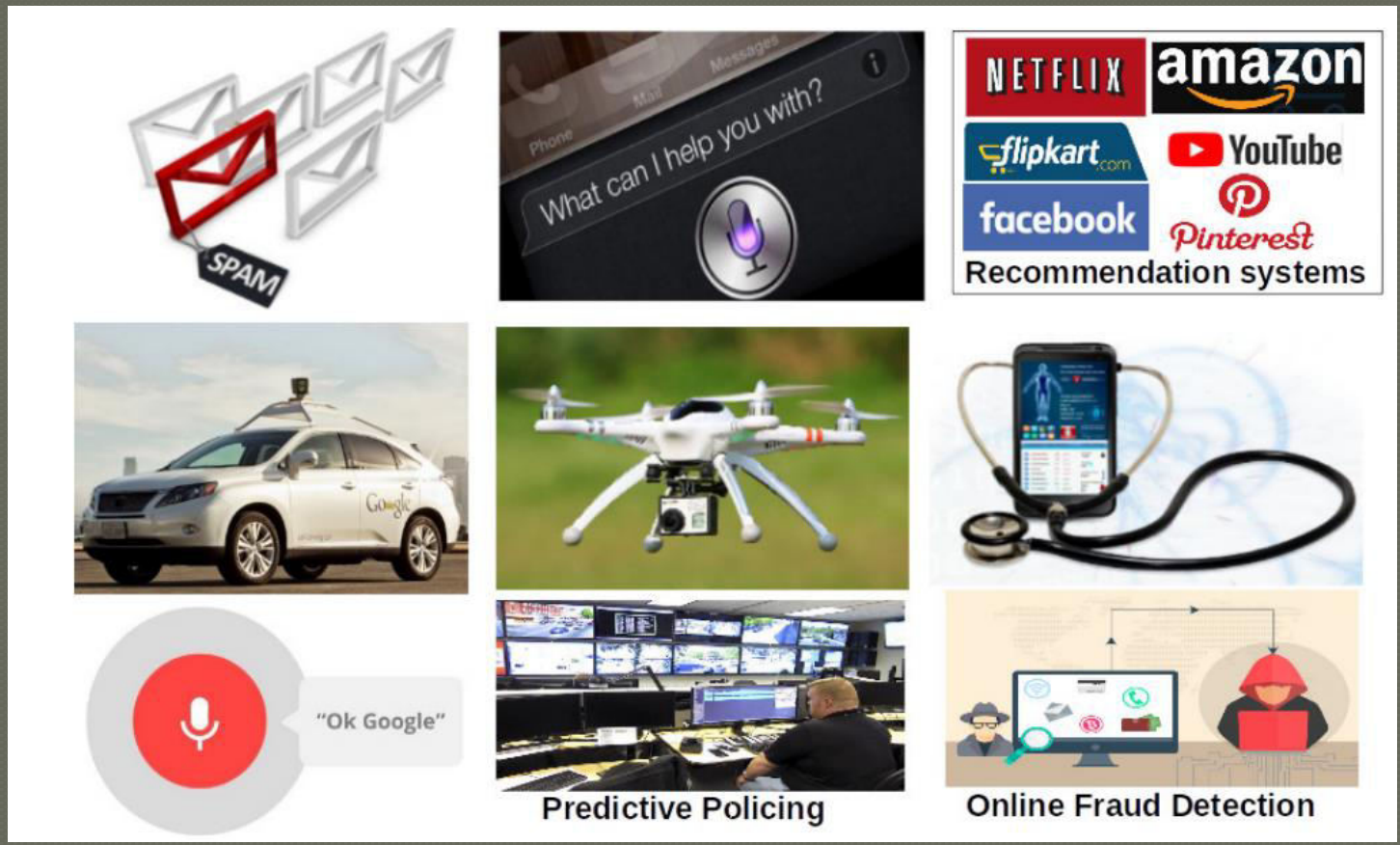
Generalization

- The most accurate model for your training data is a lookup table



- A good ML model must generalize well on unseen (test data)
  - To avoid approximating a lookup table for the training data
- Simpler models should be preferred over more complex
  - An article of religious faith

ML Applications Abound..



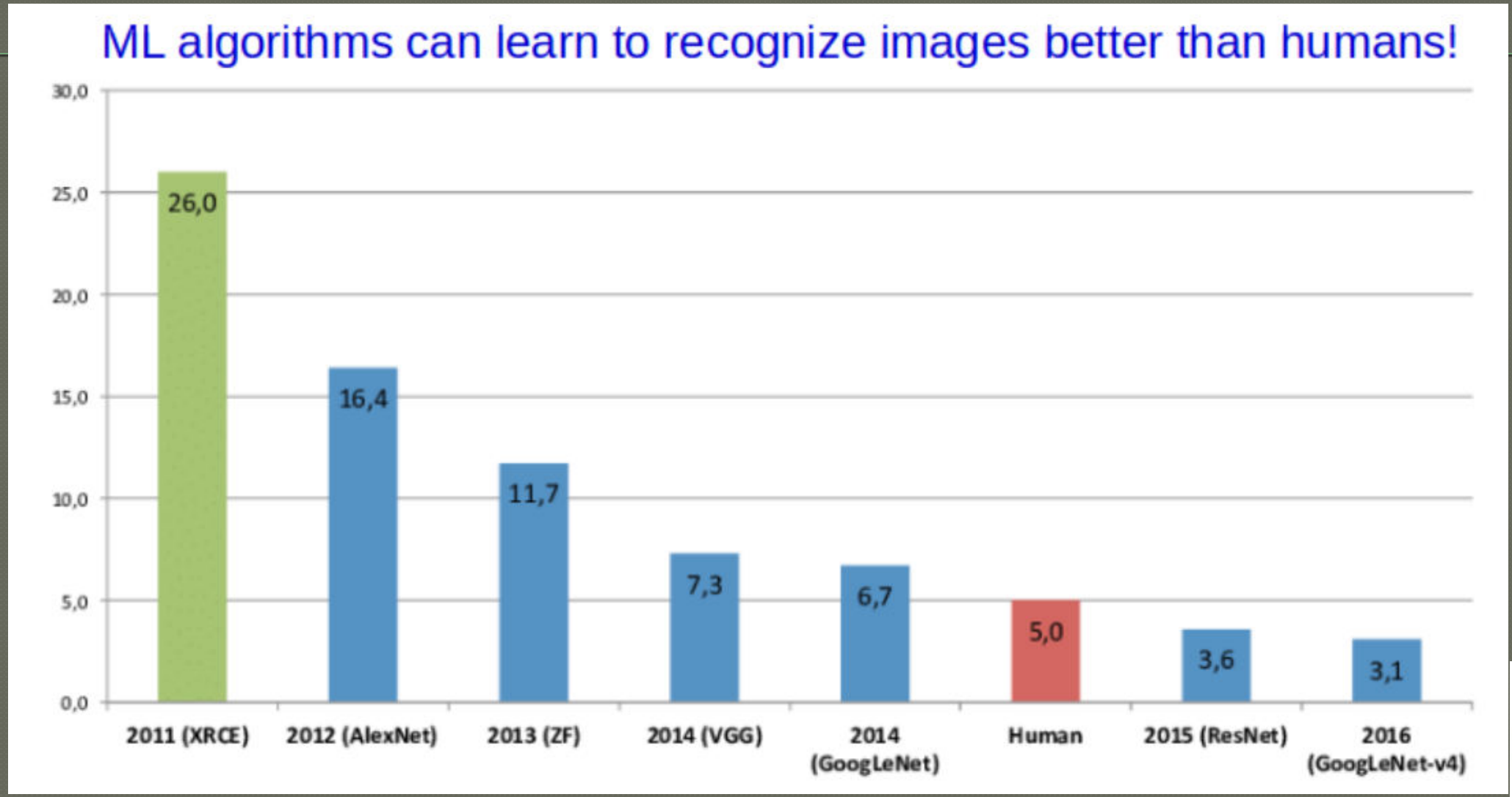
Picture courtesy: gizmodo.com,rcdronearena.com,www.wiseyak.com,www.charlesdong.com

Key Enablers for Modern ML

- Availability of large amounts of data to train ML models

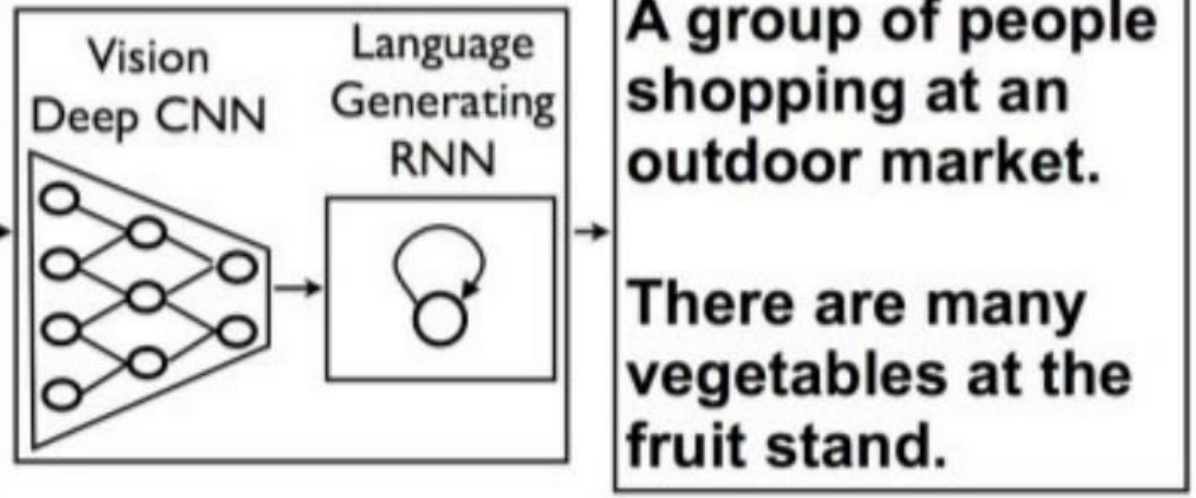
- Increased computing power (e.g., GPUs)

ML: Some Success Stories



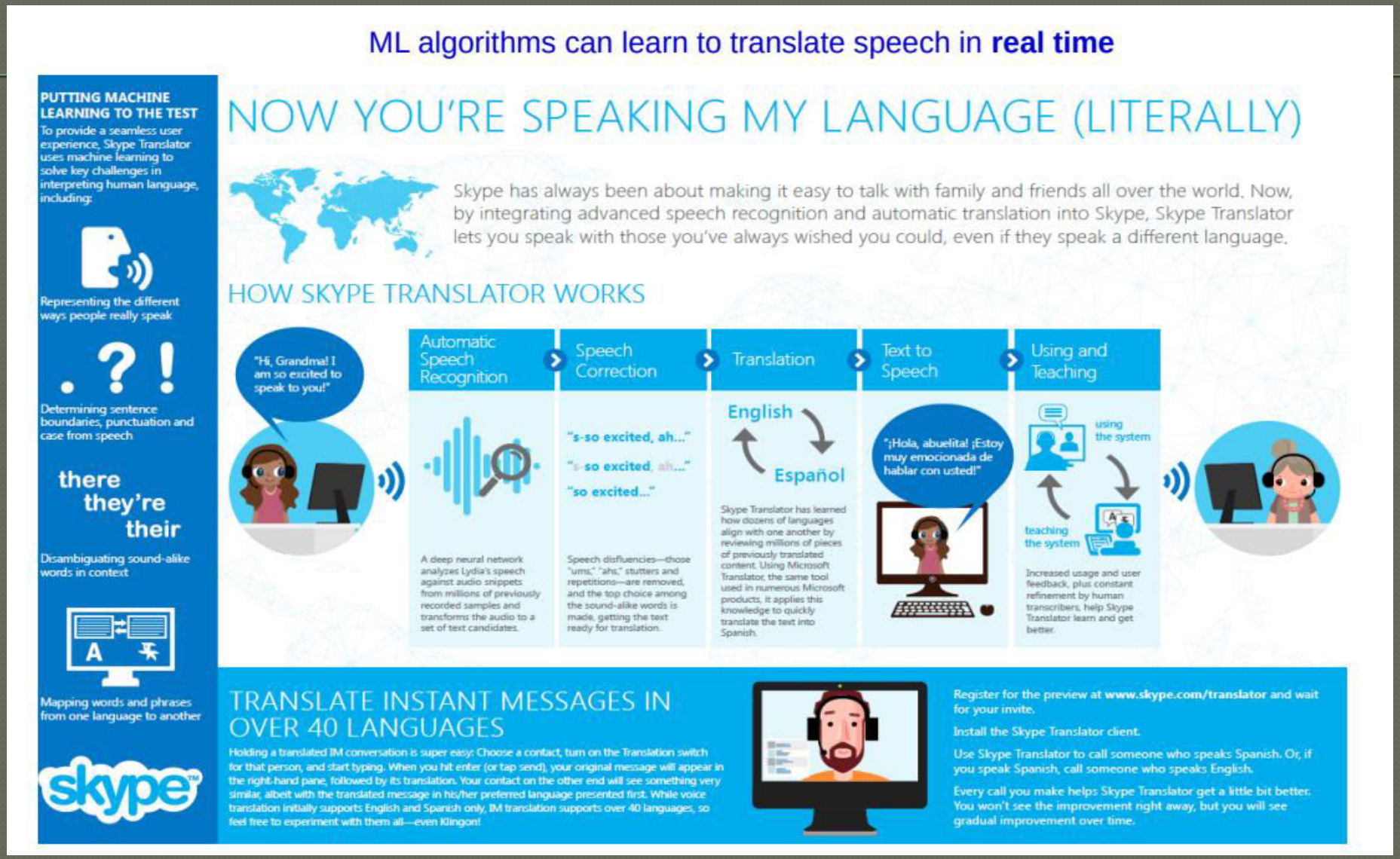
ML: Some Success Stories

ML algorithms can learn to generate captions for images



http://arxiv.org/abs/1411.4555 "Show and Tell: A Neural Image Caption Generator"

ML: Some Success Stories



Picture courtesy: https://news.microsoft.com/

ML: Some Success Stories

- Automatic Program Correction

```
1 #include<stdio.h>
2 int main(){
3  int a;
4  scanf("%d", a );
5  printf("ans=%d",
6   a+10);
7  return 0;
8 }
```

```
1 #include<stdio.h>
2 int main(){
3  int a;
4  scanf("%d", &a );
5  printf("ans=%d",
6   a+10);
7  return 0;
8 }
```

Figure 1: Left: erroneous program, Right: fix by TRACER. The compiler message read: *Line-4, Column-9: warning: format '%d' expects argument of type 'int *', but argument 2 has type 'int'.*

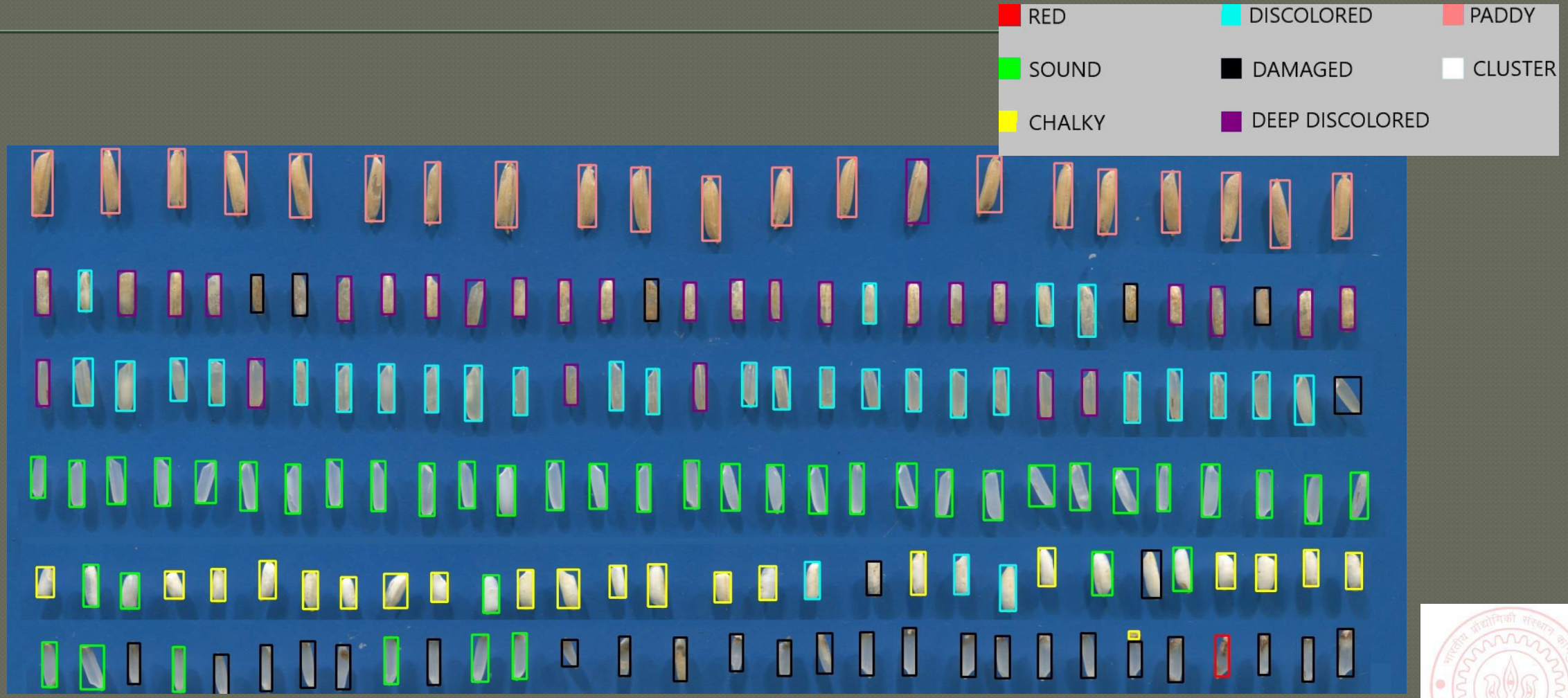
```
1 #include<stdio.h>
2 int main(){
3    int x,x1,d;
4    // ...
5    d=(x-x1)(x-x1);
6    return d;
7 }
```

```
1 #include<stdio.h>
2 int main(){
3    int x,x1,d;
4    // ...
5    d=(x-x1)*(x-x1);
6    return d;
7 }
```

Figure 2: Left: erroneous program, Right: fix by TRACER. The compiler message read: *Line-5, Column-11: error: called object type 'int' is not a function or function pointer.*

Example from "Compilation error repair: for the student programs, from the student programs", Ahmed et al (2018)

ML: Some success stories



** From Pant, Singh & Srivastava (2021)*

Politically correct ML Systems

- Good ML should not just be about getting high accuracies; they are now also expected to be politically correct
- Should also ensure that the ML models are fair and unbiased

An image captioning system should not always assume a specific gender

Don't want a self-driving car that is more likely to hit black people than white people

Criminals?

Not Criminals?

Don't want a predictive policing system that predicts criminality using facial features

- A lot of recent focus on Fairness and Transparency of ML systems

Picture courtesy: Bhargava and Forsyth (2019), https://www.thestranger.com/, Xiaolin Wu and Xi Zhang, "Automated Inference on Criminality Using Face Images"

Learning about ML

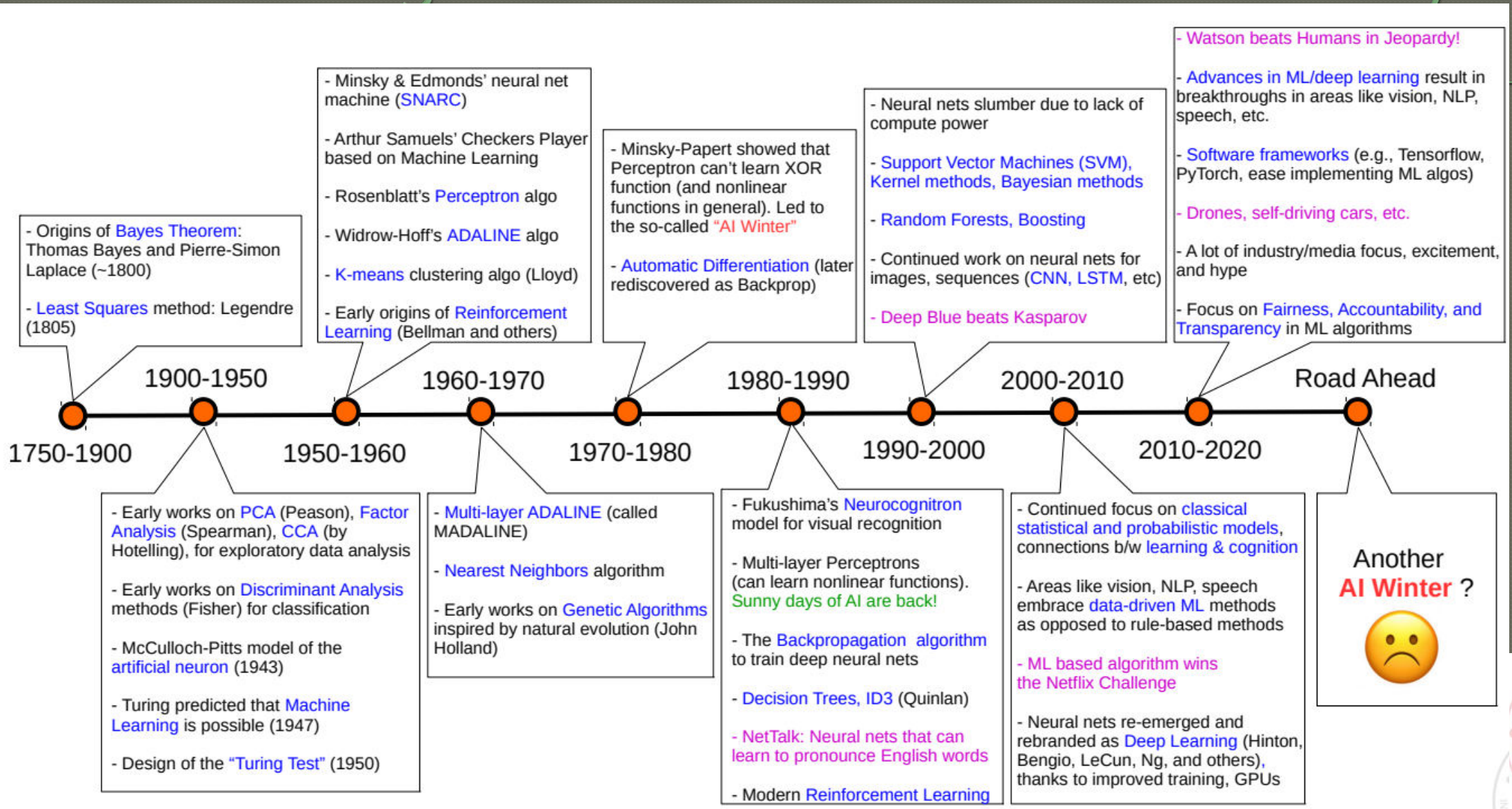
We will also focus on these

- Data
- Representations
- Algorithms
- Outputs

We will also focus on these

- Interpretation

Looking Back Before We Start: History of ML



Next Class

- Various Flavors of ML problems
- Data and features
- Basic mathematical operations on data and features

Working with Data and Features

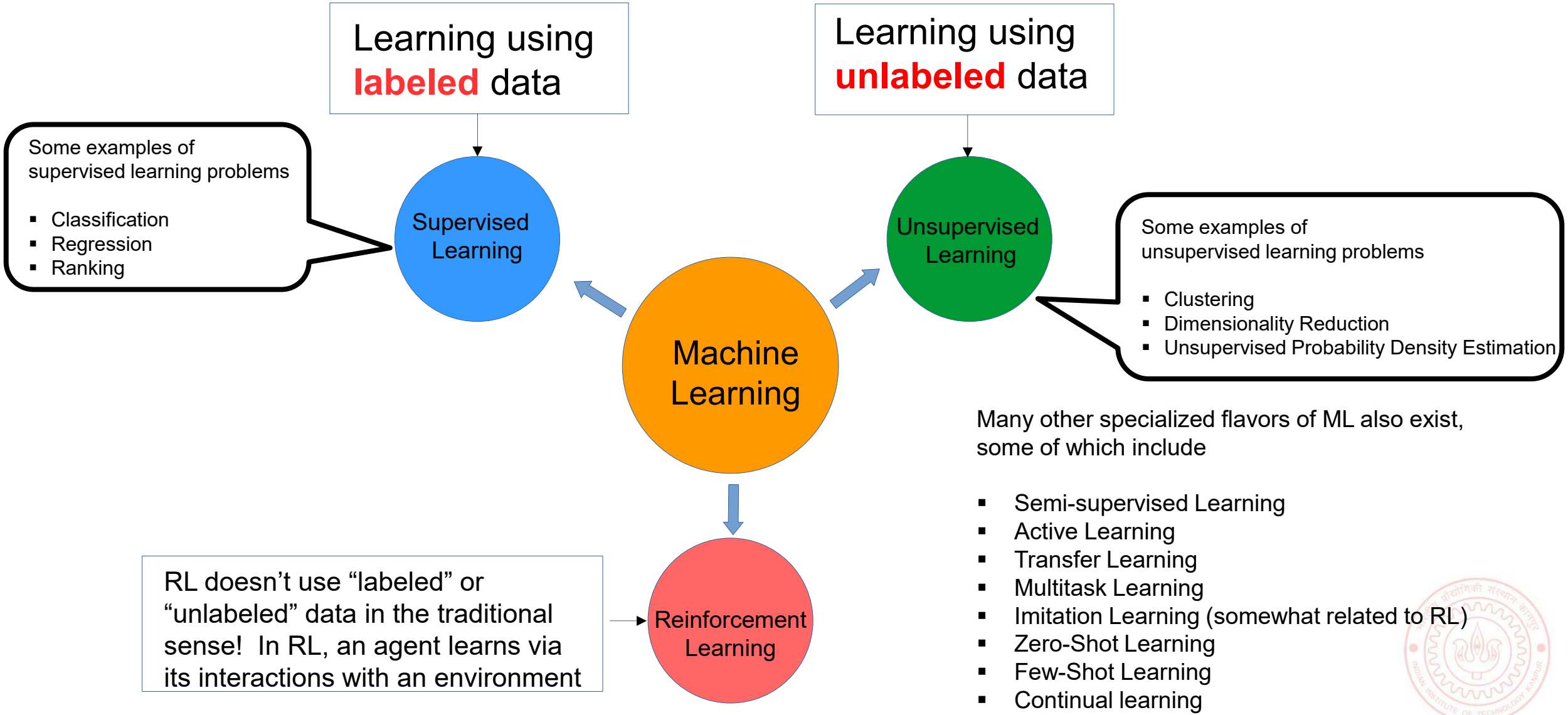
CS771: Introduction to Machine Learning

Nisheeth Srivastava

Plan for today

- Types of ML problems
- Typical workflow of ML problems
- Various perspectives of ML problems
- Data and Features
- Some basic operations of data and features
- People who need a math refresher can use this handy tutorial

A Loose Taxonomy of ML

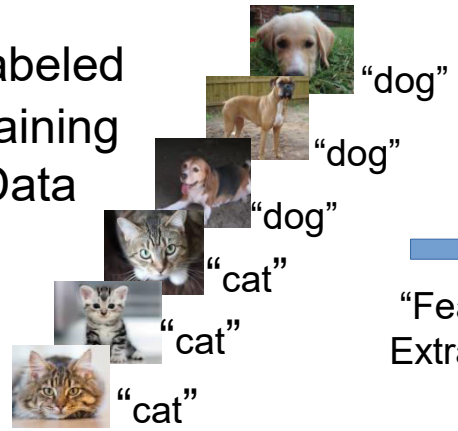


Many other specialized flavors of ML also exist, some of which include

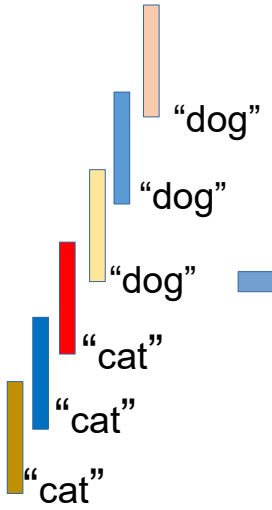
- Semi-supervised Learning
- Active Learning
- Transfer Learning
- Multitask Learning
- Imitation Learning (somewhat related to RL)
- Zero-Shot Learning
- Few-Shot Learning
- Continual learning

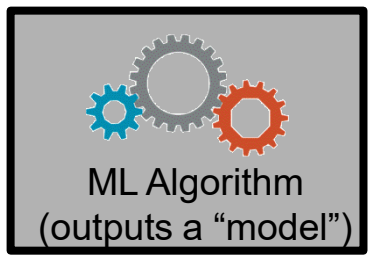
A Typical Supervised Learning Workflow

Note: This example is for the problem of binary classification, a supervised learning problem



"Feature" Extraction



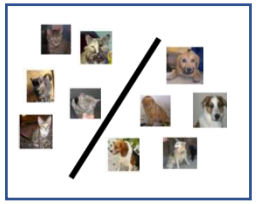


Can you think of a problem you would try to solve using supervised learning?

Feature extraction converts raw inputs to a numeric representation that the ML algo can understand and work with. More on feature extraction later.

Test Image

"Feature" Extraction

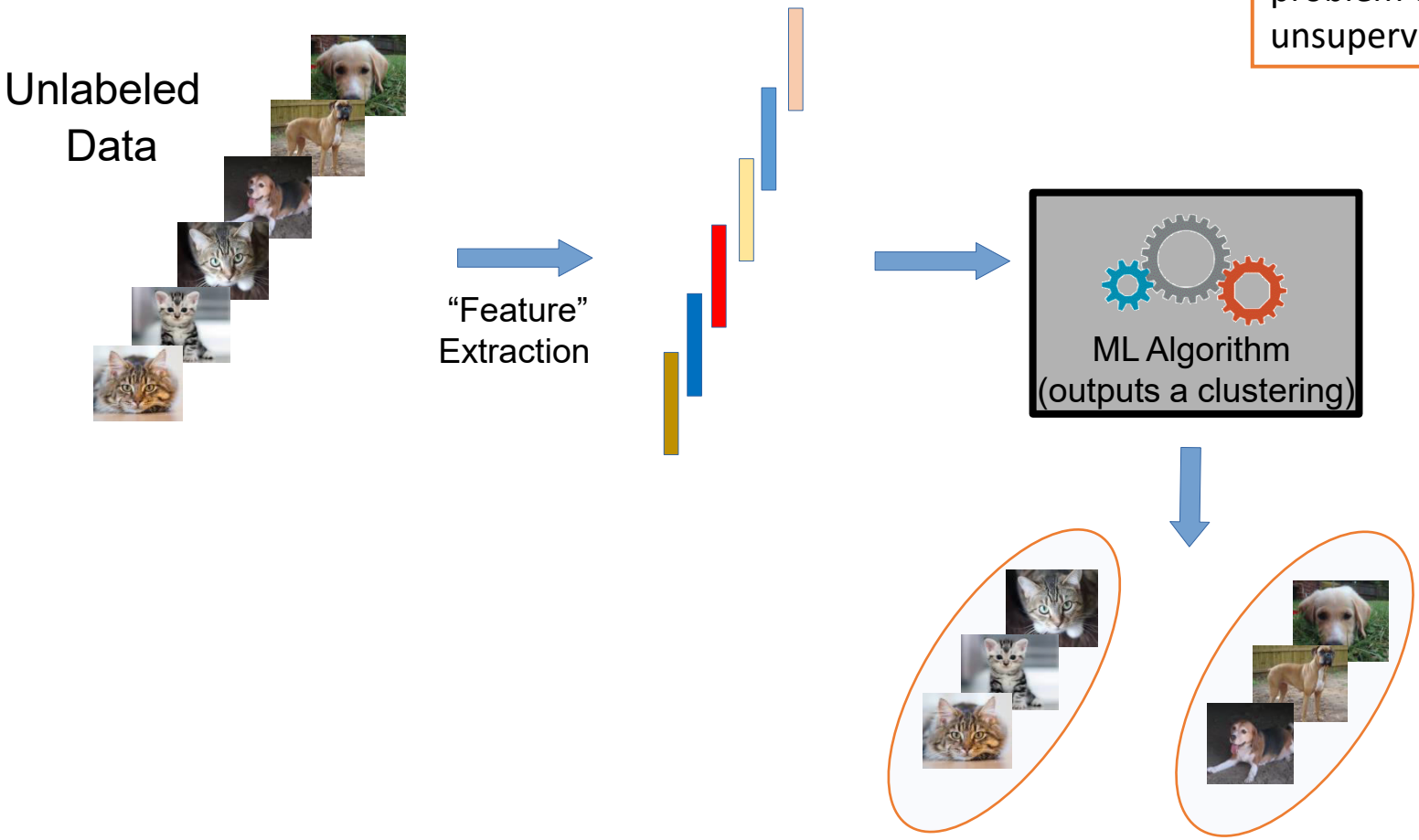


Cat vs Dog Prediction model

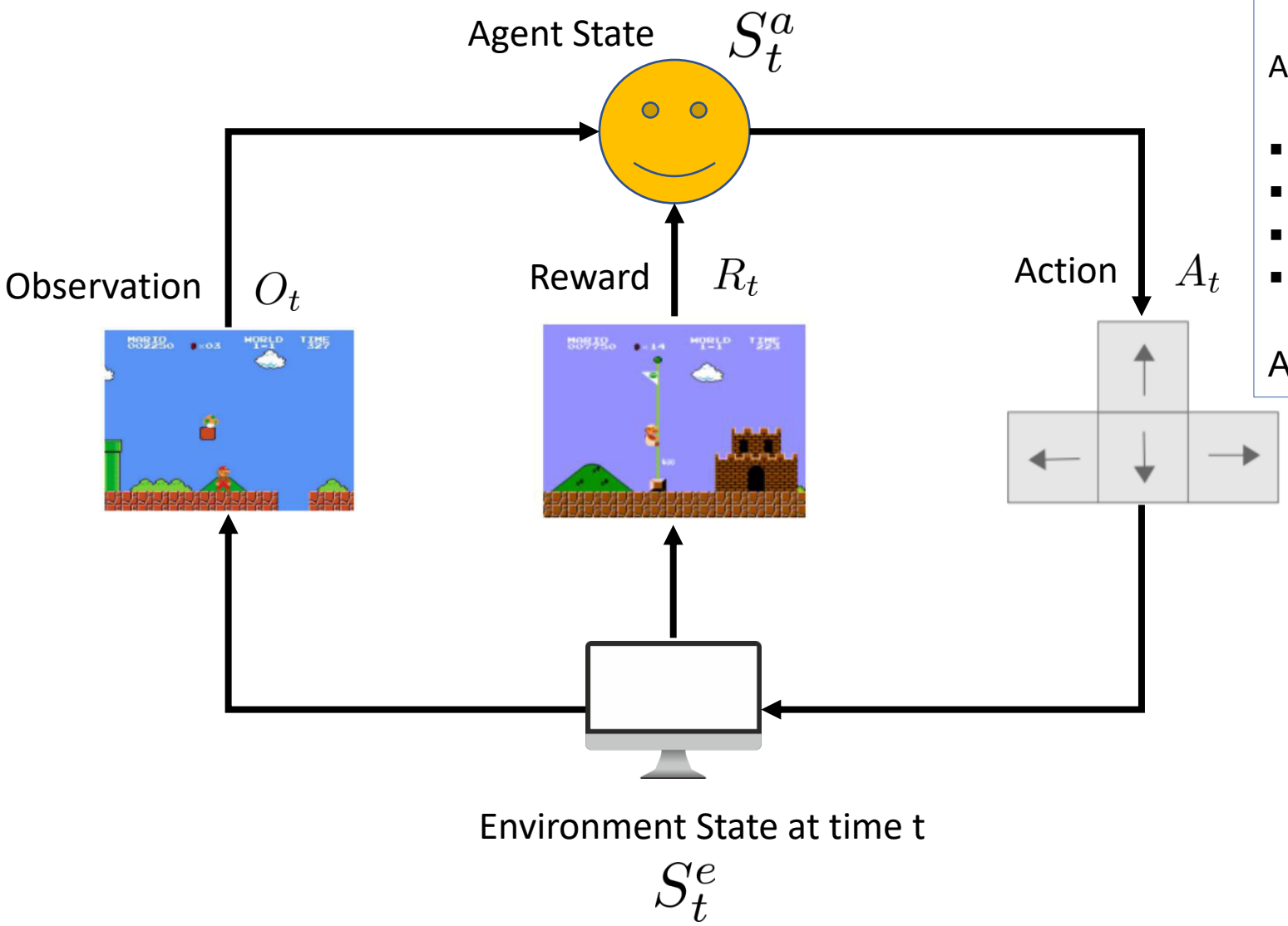
Predicted Label (cat/dog)

A Typical Unsupervised Learning Workflow

Note: This example is for the problem of data clustering, an unsupervised learning problem



A Typical Reinforcement Learning Workflow



Wish to teach an agent optimal policy for some task

Agent does the following repeatedly

- Senses/observes the environment
- Takes an action based on its current policy
- Receives a reward for that action
- Updates its policy

Agent's goal is to maximize its overall reward

There IS supervision, not explicit (as in Supervised Learning) but rather implicit (feedback based)

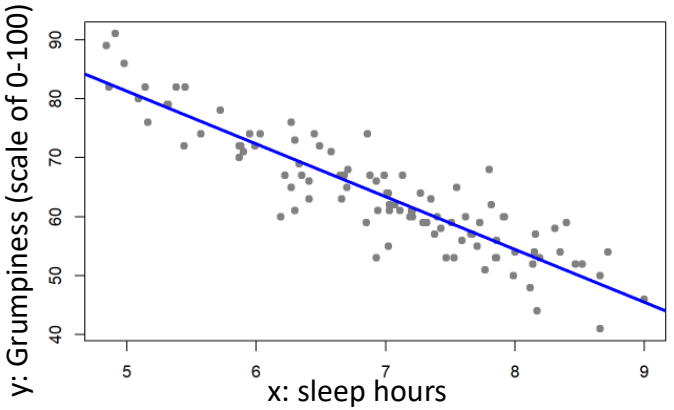
ML: Some Perspectives

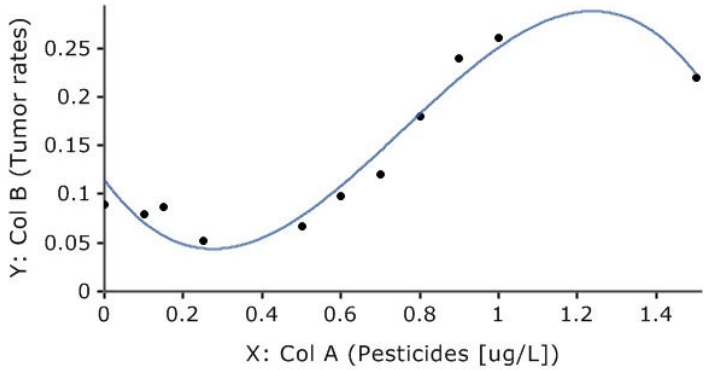
Geometric Perspective

Recall that feature extraction converts inputs into a numeric representation

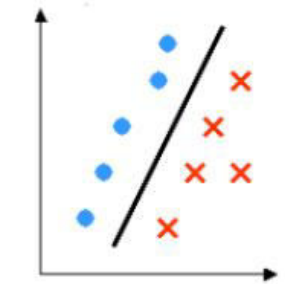
- Basic fact: Inputs in ML problems can often be represented as points or vectors in some vector space
- Doing ML on such data can thus be seen from a geometric view

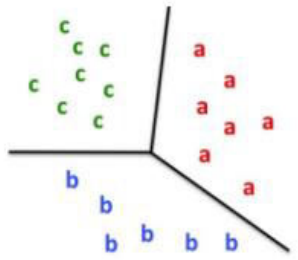
Regression: A supervised learning problem. Goal is to model the relationship between input (x) and real-valued output (y). This is akin to a line or curve fitting problem

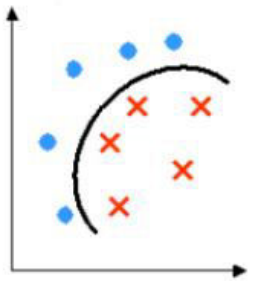


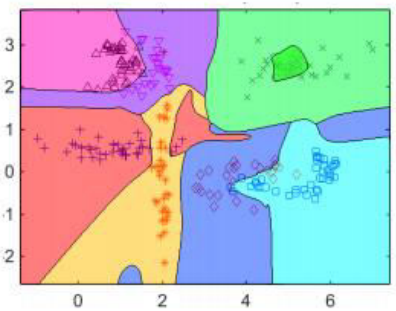


Classification: A supervised learning problem. Goal is to learn a to predict which of the two or more classes an input belongs to. Akin to learning linear/nonlinear separator for the inputs


Two-Class (binary) Linear Classification


Multi-Class Linear Classification

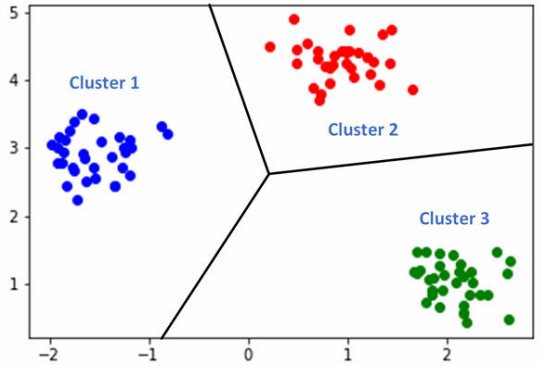

Two-Class (binary) Nonlinear Classification


Multi-Class Nonlinear Classification

Pic from: https://learningstatisticswithr.com/book/regression.html, https://maxstat.de/

Geometric Perspective

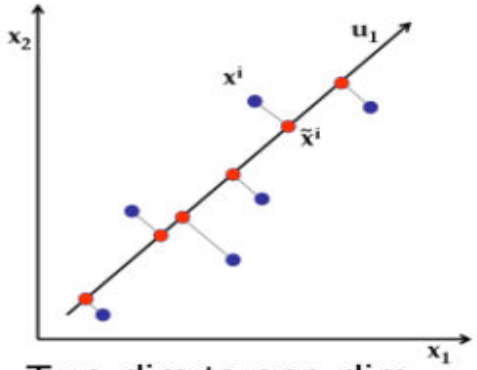
Clustering: An unsupervised learning problem. Goal is to group inputs in a few clusters based on their similarities with each other



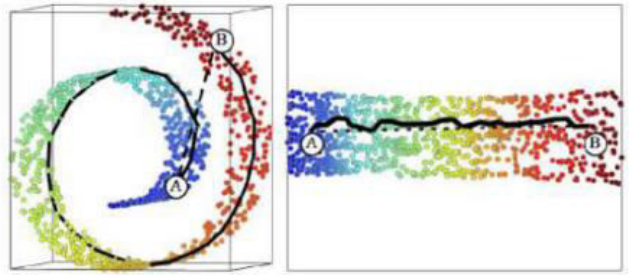
Clustering looks like classification to me. Is there any difference?

Yes. In clustering, we don't know the labels. Goal is to separate them without any labeled "supervision"

Dimensionality Reduction: An unsupervised learning problem. Goal is to compress the size of each input without losing much information present in the data



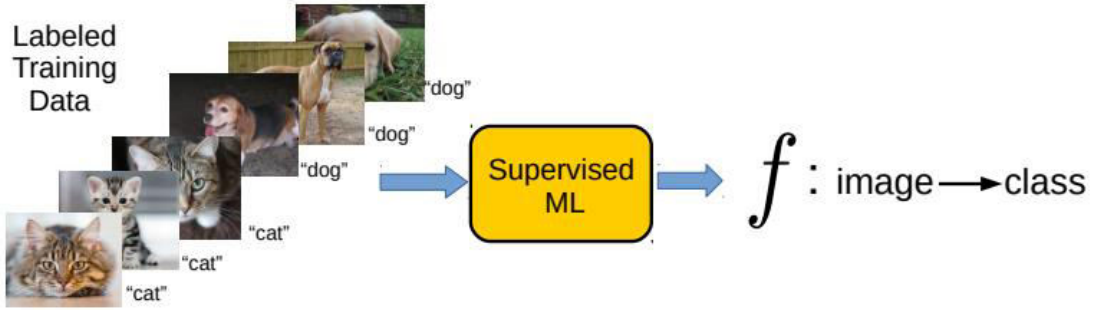
Two-dim to one-dim linear projection



Three-dim to two-dim nonlinear projection (a.k.a. manifold learning)

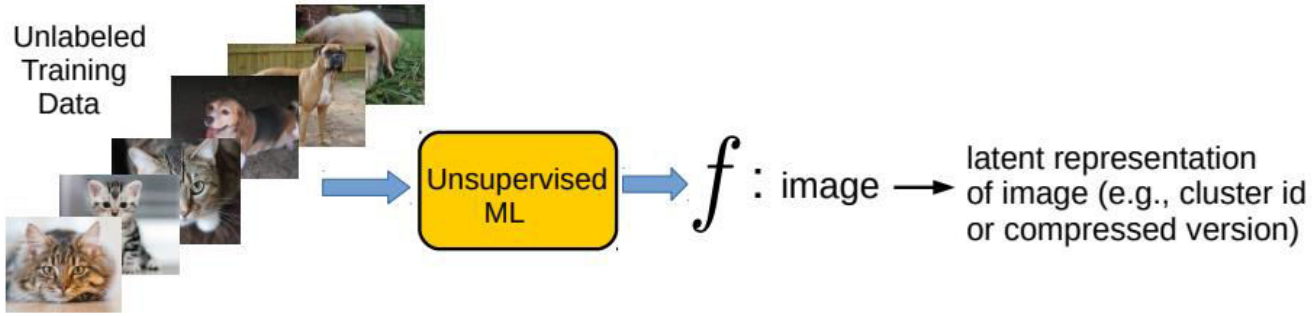
Perspective as function approximation

- Supervised Learning ("predict output given input") can be usually thought of as learning a function *f* that maps each input to the corresponding output



- Unsupervised Learning ("model/compress inputs") can also be usually thought of as learning a function *f* that maps each input to a compact representation

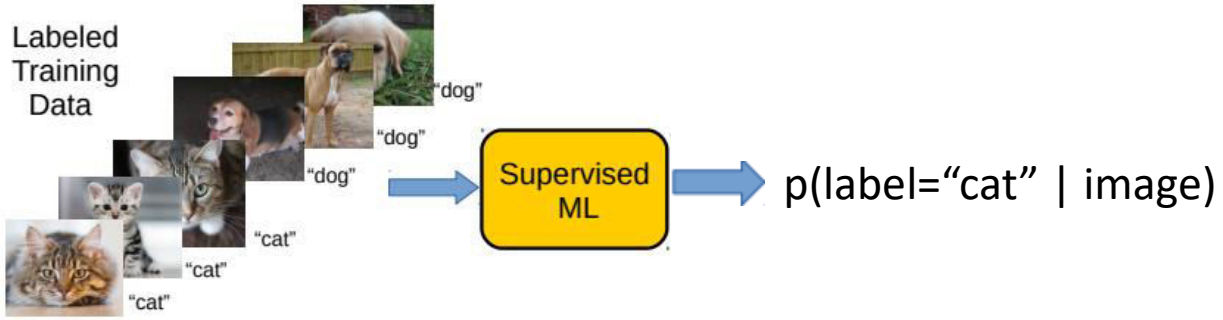
Harder since we don't know the labels in this case



- Reinforcement Learning can also be seen as doing function approximation

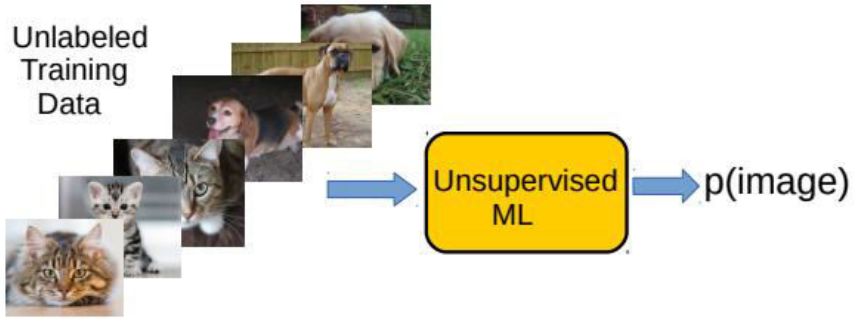
Perspective as probability estimation

- Supervised Learning ("predict output given input") can be thought of as estimating the conditional probability of each possible output given an input



- Unsupervised Learning ("model/compress inputs") can be thought of as estimating the probability density of the inputs

Harder since we don't know the labels in this case



- Reinforcement Learning can also be seen as estimating probability densities

Data and Features

Data and Features

- ML algos require a numeric feature representation of the inputs

- Features can be obtained using one of the two approaches
  - Approach 1: Extracting/constructing features manually from raw inputs
  - Approach 2: Learning the features from raw inputs

- Approach 1 is what we will focus on primarily for now

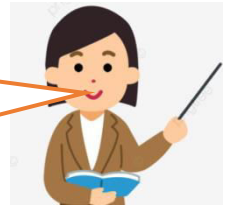
- Approach 2 is what is followed in Deep Learning algorithms (will see later)

- Approach 1 is not as powerful as Approach 2 but still used widely

Example: Feature Extraction for Text Data

- Consider some text data consisting of the following sentences:
  - John likes to watch movies
  - Mary likes movies too
  - John also likes football

BoW is just one of the many ways of doing feature extraction for text data. Not the most optimal one, and has various flaws (can you think of some?), but often works reasonably well



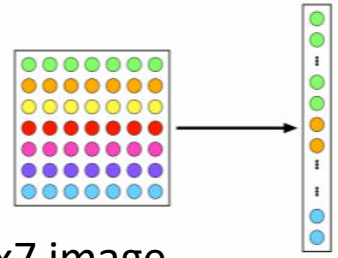
- Want to construct a feature representation for these sentences
- Here is a "bag-of-words" (BoW) feature representation of these sentences

	John	likes	to	watch	movies	Mary	too	also	football
Sentence 1	1	1	1	1	1	0	0	0	0
Sentence 2	0	1	0	0	1	1	1	0	0
Sentence 3	1	1	0	0	0	0	0	1	1

- Each sentence is now represented as a binary vector (each feature is a binary value, denoting presence or absence of a word). BoW is also called "unigram" rep.

Example: Feature Extraction for Image Data

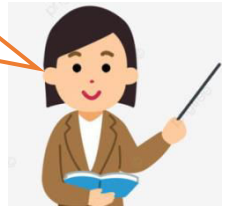
- A very simple feature extraction approach for image data is flattening



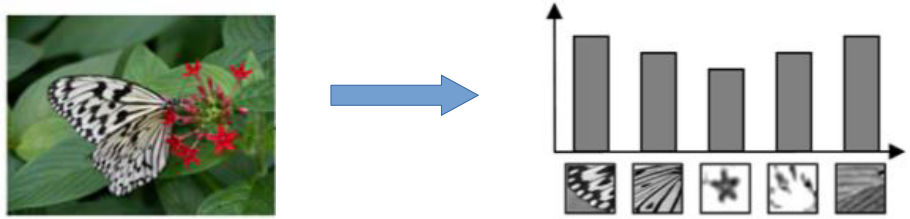
7x7 image (49 pixels)

Vector of pixel intensities

Flattening and histogram based methods destroy the spatial information in the image but often still work reasonably well



- Histogram of visual patterns is another popular feature extr. method for images



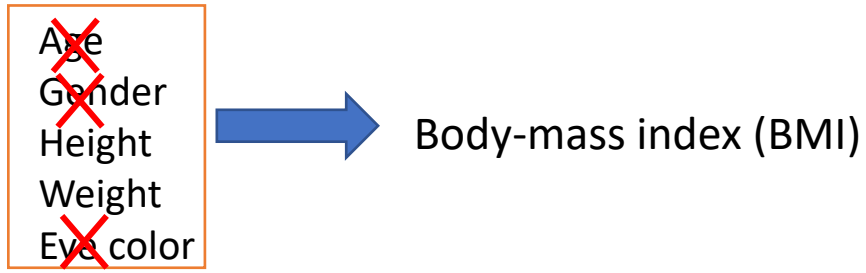
- Many other manual feature extraction techniques developed in computer vision and image processing communities (SIFT, HoG, and others)

Pic credit: cat.uab.cat/Research/object-recognition

Feature Selection

- Not all the extracted features may be relevant for learning the model (some may even confuse the learner)

- Feature selection (a step after feature extraction) can be used to identify the features that matter, and discard the others, for more effective learning

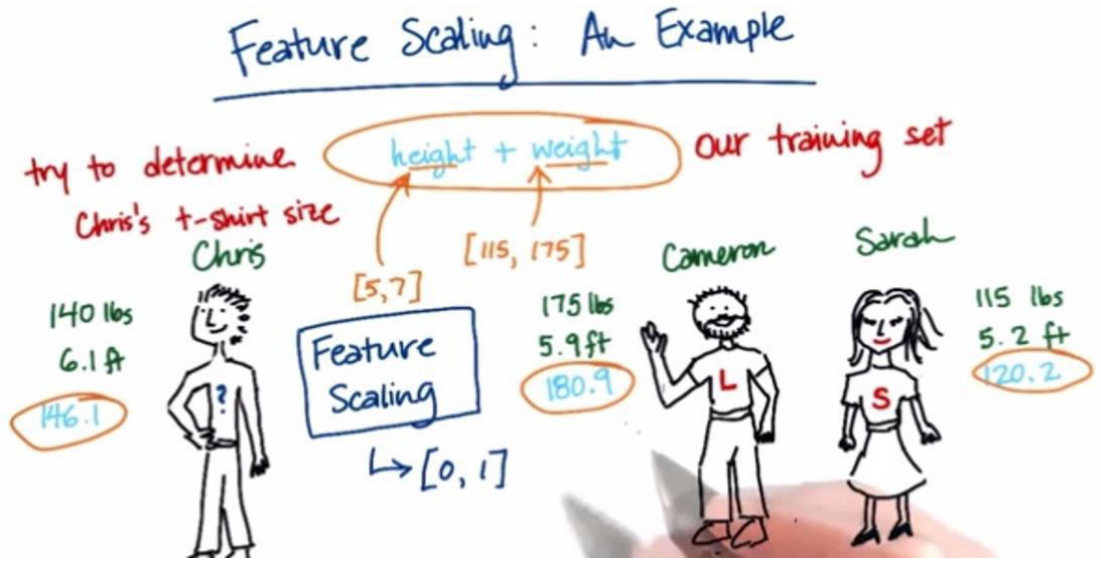


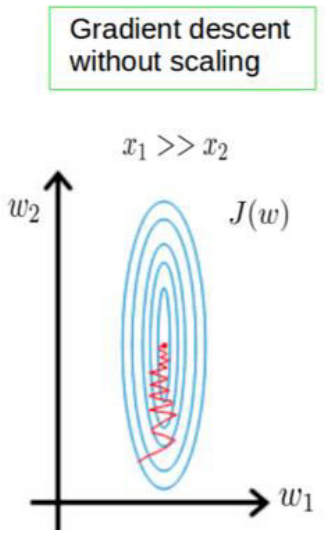
- Many techniques exist – some based on intuition, some based on algorithmic principles (will visit feature selection later)

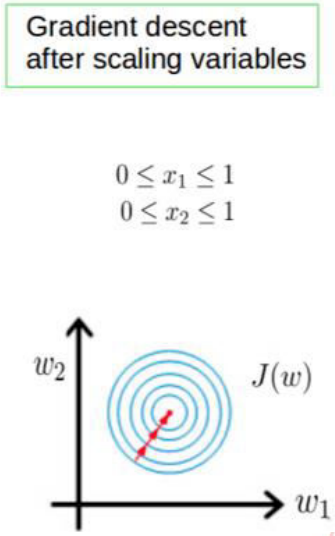
- More common in supervised learning but can also be done for unsup. learning

Some More Postprocessing: Feature Scaling

- Even after feature selection, the features may not be on the same scale
- This can be problematic when comparing two inputs – features that have larger scales may dominate the result of such comparisons
- Therefore helpful to standardize the features (e.g., by bringing all of them on the same scale such as between 0 to 1)







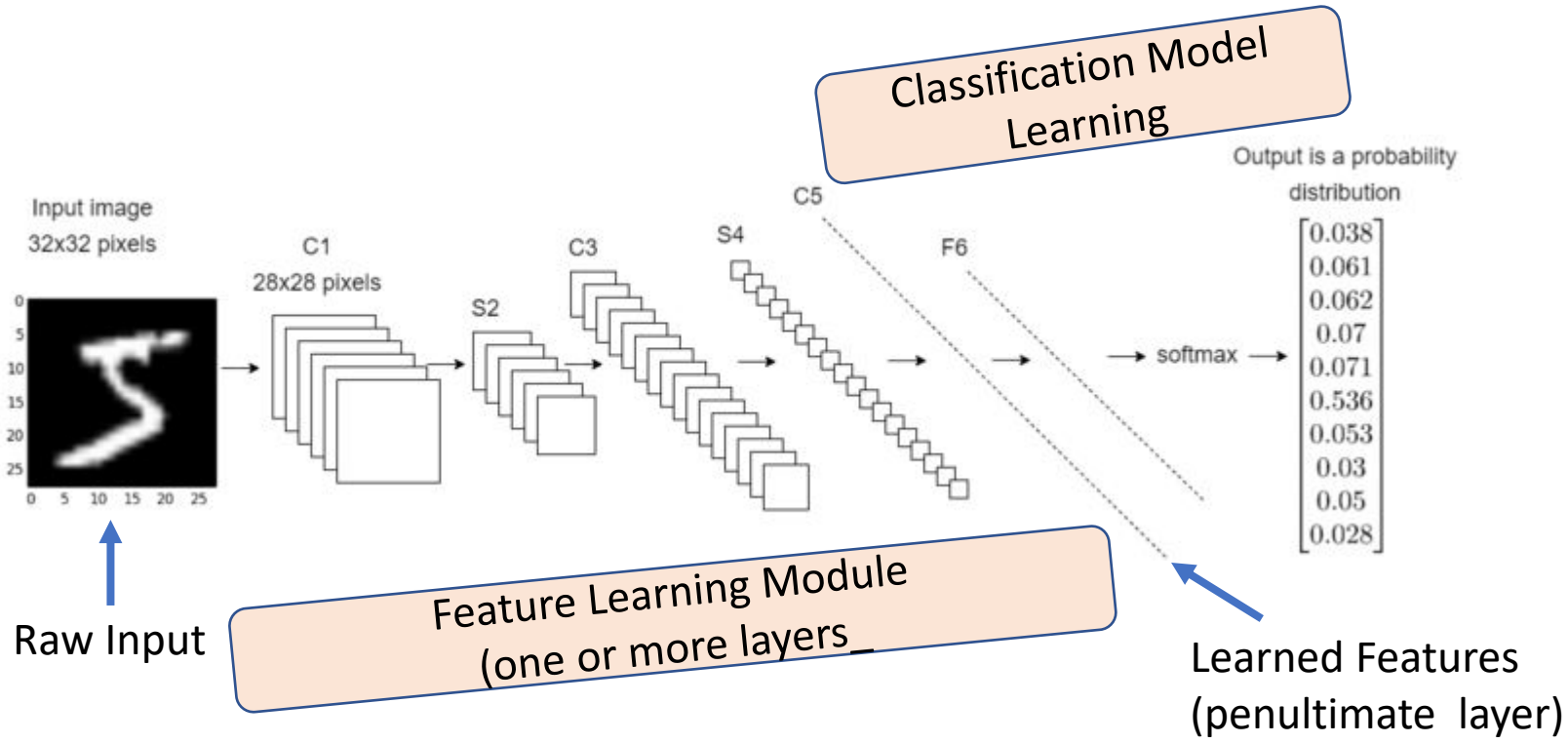
- Also helpful for stabilizing the optimization techniques used in ML algos

Pic credit: https://becominghuman.ai/demystifying-feature-scaling-baff53e9b3fd, https://stackoverflow.com/

Deep Learning: An End-to-End Approach to ML

Deep Learning = ML with automated feature learning from the raw inputs

Feature extraction part is automated via the feature learning module



Pic an adaptation of the original from: https://deepai.org/

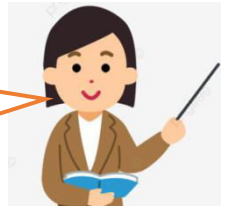
Some Notation/Nomenclature/Convention

- Sup. learning requires training data as N input-output pairs $\{(\mathbf{x_n}, y_n)\}_{n=1}^{N}$
- Unsupervised learning requires training data as N inputs $\{\mathbf{x_n}\}_{n=1}^{N}$

RL and other flavors of ML problems also use similar notation

- Each input $\mathbf{x_n}$ is (usually) a vector containing the values of the features or attributes or covariates that encode properties of the it represents, e.g.,
  - For a 7 × 7 image: $\mathbf{x_n}$ can be a 49 × 1 vector of pixel intensities

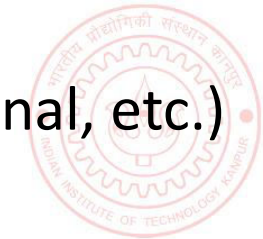
Size or length of the input $\mathbf{x_n}$ is commonly known as data/input dimensionality or feature dimensionality



- (In sup. Learning) Each y_n is the output or response or label associated with input $\mathbf{x_n}$ (and its value is known for the training inputs)
  - Output can be a scalar, a vector of numbers, or even an structured object (more on this later)

Types of Features and Types of Outputs

- Features as well as outputs can be real-valued, binary, categorical, ordinal, etc.
- Real-valued: Pixel intensity, house area, house price, rainfall amount, temperature, etc
- Binary: Male/female, adult/non-adult, or any yes/no or present/absent type value
- Categorical/Discrete: Zipcode, blood-group, or any “one from a finite many choices“ value
- Ordinal: Grade (A/B/C etc.) in a course, or any other type where relative values matter
- Often, the features can be of mixed types (some real, some categorical, some ordinal, etc.)

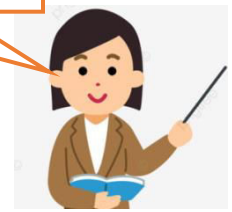


Some Basic Operations of Inputs

- Assume each input feature vector $\mathbf{x}_n \in R^D$ to of size D

- Given N inputs $\{\mathbf{x}_n\}_{n=1,}^{N}$ their average or mean can be computed as

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$$



- Can compute the Euclidean distance between any pair of inputs $\mathbf{x}_n$ and $\mathbf{x}_m$

$$d(\boldsymbol{x}_n, \boldsymbol{x}_m) = ||\boldsymbol{x}_n - \boldsymbol{x}_m|| = \sqrt{(\boldsymbol{x}_n - \boldsymbol{x}_m)^\top (\boldsymbol{x}_n - \boldsymbol{x}_m)} = \sqrt{\sum_{d=1}^{D} (x_{nd} - x_{md})^2}$$

- .. or Euclidean distance between an input $\mathbf{x}_n$ and the mean $\boldsymbol{\mu}$ of all inputs

- .. and various other operations that we will look at later..

Next Class

- Introduction to Supervised Learning

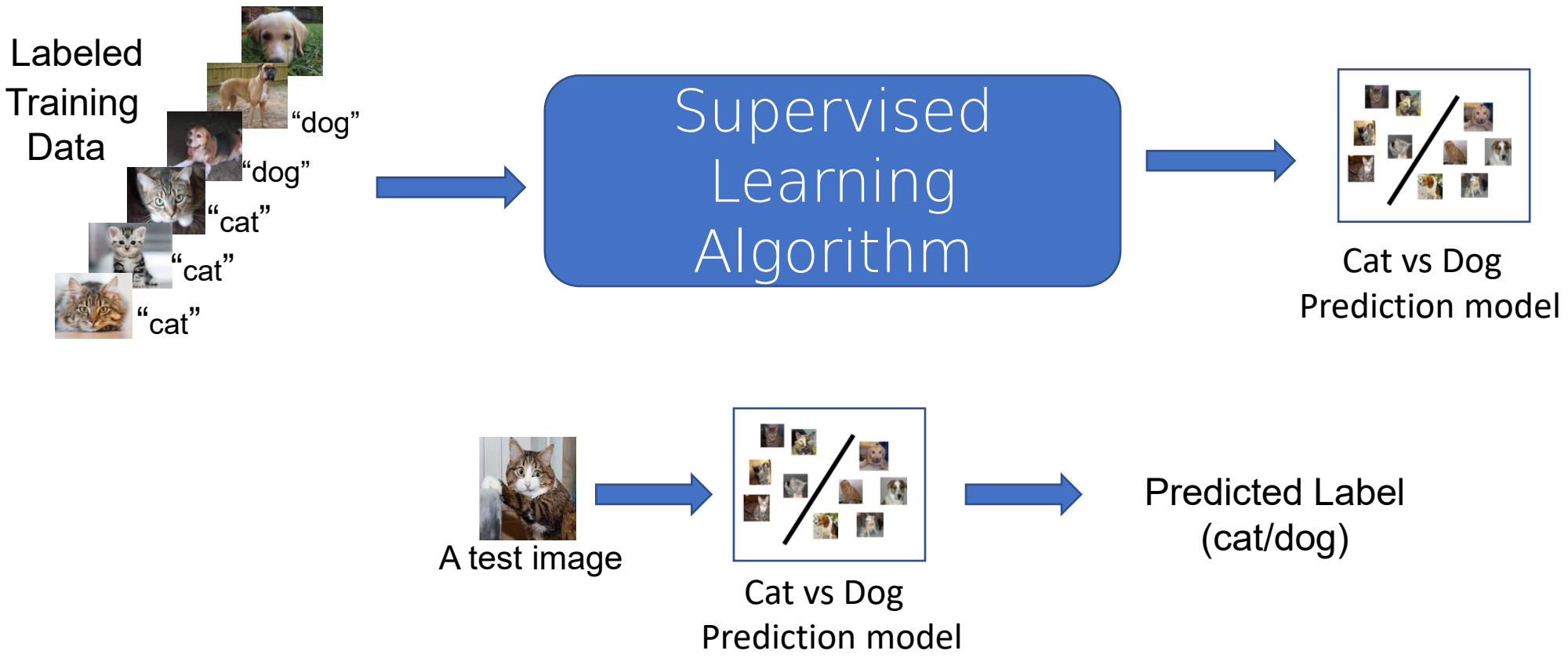
- A simple Supervised Learning algorithm based on computing distances

Learning with Prototypes

CS771: Introduction to Machine Learning

Nisheeth

Supervised Learning

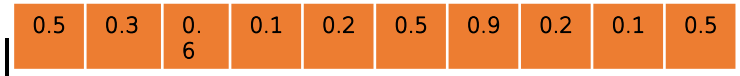


Some Types of Supervised Learning Problems

- Consider building an ML module for an e-mail client
- Some tasks that we may want this module to perform
  - Predicting whether an email of spam or normal: Binary Classification
  - Predicting which of the many folders the email should be sent to: Multi-class Classification
  - Predicting all the relevant tags for an email: Tagging or Multi-label Classification
  - Predicting what's the spam-score of an email: Regression
  - Predicting which email(s) should be shown at the top: Ranking
  - Predicting which emails are work/study-related emails: One-class Classification
- These predictive modeling tasks can be formulated as supervised learning problems
- Today: A very simple supervised learning model for binary/multi-class classification

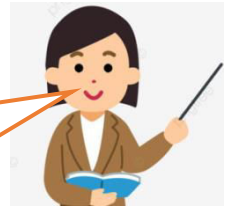
Some Notation and Conventions

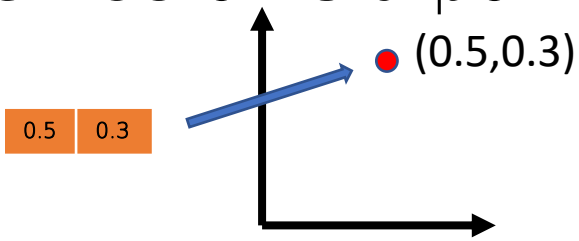
- In ML, inputs are usually represented by vectors
- A vector consists of an array of scalar values

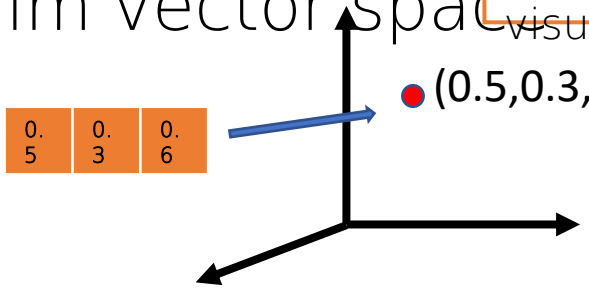


- Geometrically, a vector is just a point in a vector space, e.g.,
  - A length 2 vector is a point in 2-dim vector space
  - A length 3 vector is a point in 3-dim vector space

Likewise for higher dimensions, even though harder to visualize







- Unless specified otherwise
  - Small letters in bold font will denote vectors, e.g., $\mathbf{x}, \mathbf{a}, \mathbf{b}$ etc.
  - Small letters in normal font to denote scalars, e.g. x, a, b, etc
  - Capital letters in bold font will denote matrices (2-dim arrays)

Some Notation and Conventions

- A single vector will be assumed to be of the form $\mathbf{x} = [x_1, x_2, \ldots, x_D]$

- Unless specified otherwise, vectors will be assumed to be column vectors
  - So we will assume $\mathbf{x} = [x_1, x_2, \ldots, x_D]$ to be a column vector of size $D \times 1$
  - Assuming each element to be real-valued scalar, $\mathbf{x} \in \mathbb{R}^{D \times 1}$ or $\mathbf{x} \in \mathbb{R}^{D}$ ($\mathbb{R}$: space of reals)

- If $\mathbf{x} = [x_1, x_2, \ldots, x_D]$ is a feature vector representing, say an image, then
  - D denotes the dimensionality of this feature vector (number of features)
  - x_i (a scalar) denotes the value of i^{th} feature in the image

- For denoting multiple vectors, we will use a subscript with each

Some Basic Operations on Vectors

- Addition/subtraction of two vectors gives another vector of the same size

- The mean μ(average or centroid) of N vectors $\{\mathbf{x}_n\}_{n=1}^N$

$$\mu = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$$ (of the same size as each $\mathbf{x}_n$)

- The inner/dot product of two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^\top \boldsymbol{b} = \sum_{i=1}^{D} a_i b_i$$ (a real-valued number denoting how "similar" $\boldsymbol{a}$ ar

Assuming both $\boldsymbol{a}$ and $\boldsymbol{b}$ have unit Euclidean norm

- For a vector $\boldsymbol{a} \in \mathbb{R}^D$, its Euclidean norm is defined via its inner product with itself

$$\|\boldsymbol{a}\|_2 = \sqrt{\boldsymbol{a}^\top \boldsymbol{a}} = \sqrt{\sum_{i=1}^{d} a_i^2}$$

Computing Distances

- Euclidean (L2 norm) distance between two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b} \in \mathbb{R}^D$

$$d_2(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||_2 = \sqrt{\sum_{i=1}^{D} (a_i - b_i)^2} = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top (\boldsymbol{a} - \boldsymbol{b})} = \sqrt{\boldsymbol{a}^\top \boldsymbol{a} + \boldsymbol{b}^\top \boldsymbol{b} - 2\boldsymbol{a}^\top \boldsymbol{b}}$$

Sqrt of Inner product of the difference vector

Another expression in terms of inner products of individual vectors

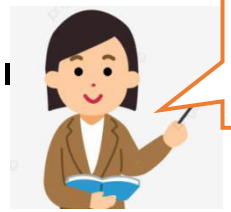
- Weighted Euclidean distance between two vect $\in \mathbb{R}^D$

$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{D} w_i (a_i - b_i)^2} = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top \mathbf{W} (\boldsymbol{a} - \boldsymbol{b})}$$

$\mathbf{W}$ is a DxD diagonal matrix with weights w_i on its diagonals. Weights may be known or even learned from data (in ML problems)

- distance between two vectors $\boldsymbol{a} \in \mathbb{R}^D$ and $\boldsymbol{b}$

$$d_1(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||_1 = \sum_{i=1}^{D} |a_i - b_i|$$



L1 norm distance is also known as the Manhattan distance or Taxicab norm
(it's a very natural notion of distance between two points in some vector space)

Our First Supervised Learner

Prelude: A Very Primitive Classifier

The idea also applies to multi-class classification: Use one image per class, and predict label based on the distances of the test image from all such images

- Consider a binary classification problem – ca
- Assume training data with just 2 imag – one  nd one
- Given a new test image (cat/dog), how do we predict its label?
- A simple idea: Predict using its distance from each of the 2 training images

$d($ Test image ,  $) < d($ Test image ,  $)$? Predict cat else dog

Wait. Is it ML? Seems to be like just a simple "rule". Where is the "learning" part in this?

Some possibilities: Use a feature learning/selection algorithm to extract features, and use a Mahalanobis distance where you learn the W matrix (instead of using a predefined W), using "distance metric learning"

Even this simple model can be learned. For example, for the feature extraction/selection part and/or for the distance computation part

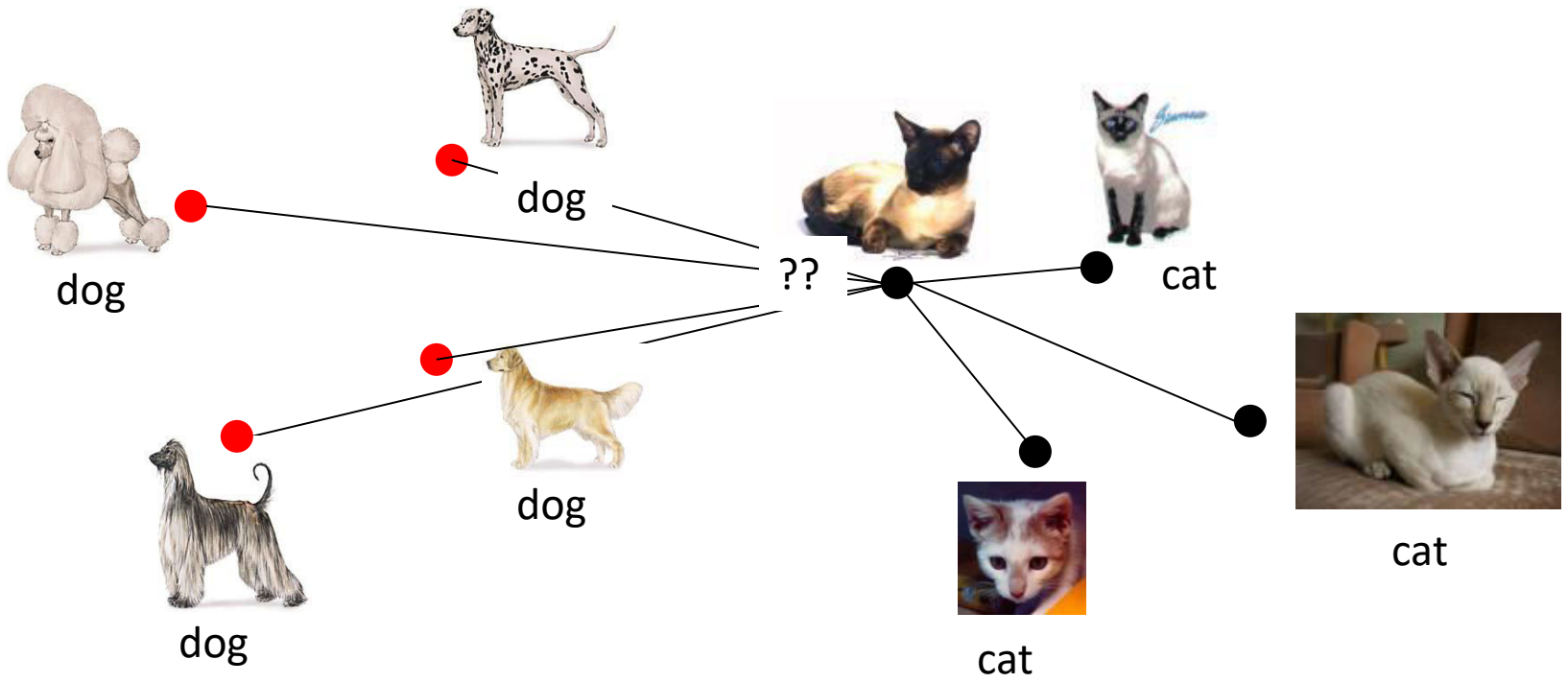
Improving Our Primitive Classifier

- Just one input per class may not sufficiently capture variations in a class

- A natural improvement can be by using more inputs per class

- We will consider two approaches to do this
  - Learning with Prototypes (LwP)
  - Nearest Neighbors (NN)
- Both LwP and NN will use multiple inputs per class but in different ways

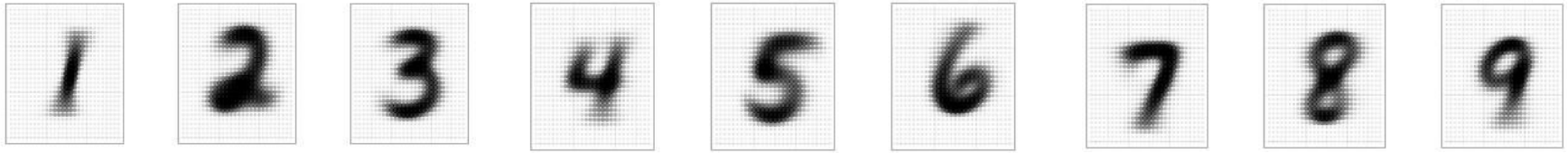
Learning to predict categories



Learning with Prototypes (LwP)

- Basic idea: Represent each class by a "prototype" vector

- Class Prototype: The "mean" or "average" of inputs from that class



Averages (prototypes) of each of the handwritten digits 1-9

- Predict label of each test input based on its distances from the class prototypes
  - Predicted label will be the class that is the closest to the test input

- How we compute distances can have an effect on the accuracy of this model (may need to try Euclidean, weight Euclidean

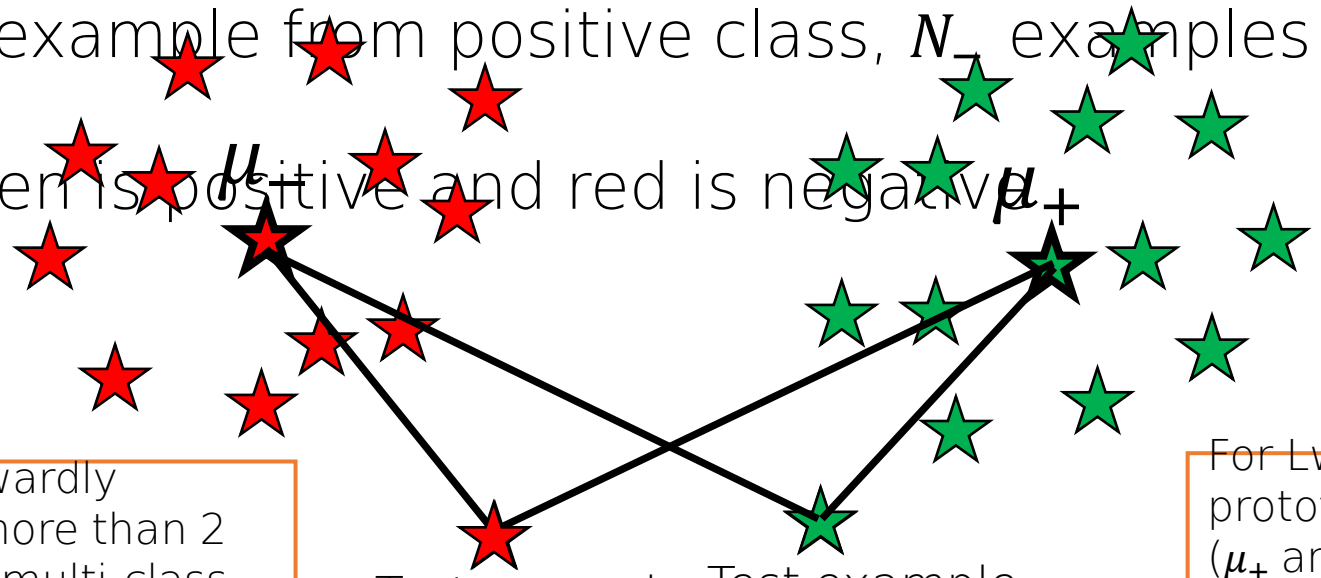
Pic from: https://www.reddit.com/r/dataisbeautiful/comments/3wgbv9/average_handwritten_digit_oc/

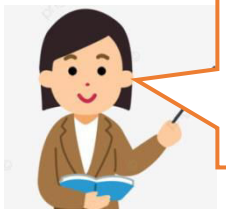
Learning with Prototypes (LwP): An Illustration

- Suppose the task is binary classification (two classes assumed pos and neg)

- Training data: N labelled examples $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, $\mathbf{x}_n \in \mathbb{R}^D$, $y_n \in \{-1, +1\}$
  - Assume N_+ example from positive class, N_- examples from negative class
  - Assume green is positive and red is negative

$$\mu_- = \frac{1}{N_-} \sum_{y_n=-1} \mathbf{x}_n \qquad \mu_+ = \frac{1}{N_+} \sum_{y_n=+1} \mathbf{x}_n$$





LwP straightforwardly generalizes to more than 2 classes as well (multi-class classification) – K prototypes for K classes

For LwP, the prototype vectors (μ_+ and μ_- here) define the "model"

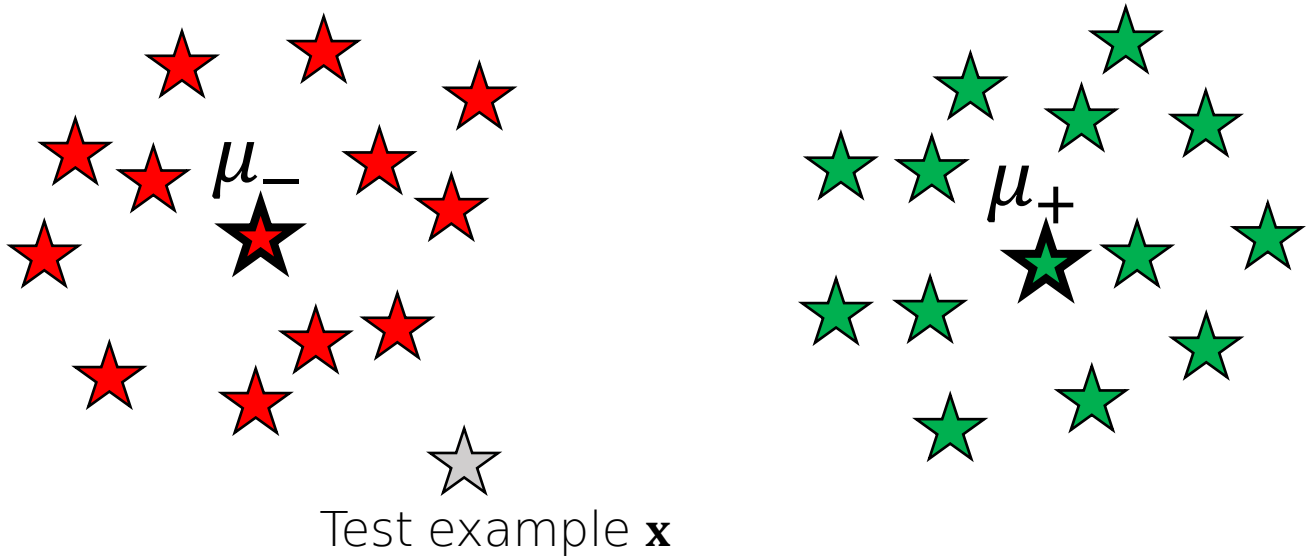
LwP: The Prediction Rule, Mathematically

- What does the prediction rule for LwP look like mathematically?

- Assume we are using Euclidean distances here

$$||\boldsymbol{\mu}_- - \mathbf{x}||^2 = ||\boldsymbol{\mu}_-||^2 + ||\mathbf{x}||^2 - 2\langle \boldsymbol{\mu}_-, \mathbf{x} \rangle$$

$$||\boldsymbol{\mu}_+ - \mathbf{x}||^2 = ||\boldsymbol{\mu}_+||^2 + ||\mathbf{x}||^2 - 2\langle \boldsymbol{\mu}_+, \mathbf{x} \rangle$$



Prediction Rule: Predict label as +1 if $f(\mathbf{x}) = ||\boldsymbol{\mu}_- - \mathbf{x}||^2 - ||\boldsymbol{\mu}_+ - \mathbf{x}||^2 > 0$ otherwi

LwP: The Prediction Rule, Mathematically

- Let's expand the prediction rule expression a bit more

$$
\begin{aligned}
f(\mathbf{x}) &= \left|\left|\boldsymbol{\mu}_- - \mathbf{x}\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+ - \mathbf{x}\right|\right|^2 \\
&= \left|\left|\boldsymbol{\mu}_-\right|\right|^2 + \left|\left|\mathbf{x}\right|\right|^2 - 2\langle\boldsymbol{\mu}_-, \mathbf{x}\rangle - \left|\left|\boldsymbol{\mu}_+\right|\right|^2 - \left|\left|\mathbf{x}\right|\right|^2 + 2\langle\boldsymbol{\mu}_+, \mathbf{x}\rangle \\
&= 2\langle\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-, \mathbf{x}\rangle + \left|\left|\boldsymbol{\mu}_-\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+\right|\right|^2 \\
&= \langle\mathbf{w}, \mathbf{x}\rangle + b
\end{aligned}
$$

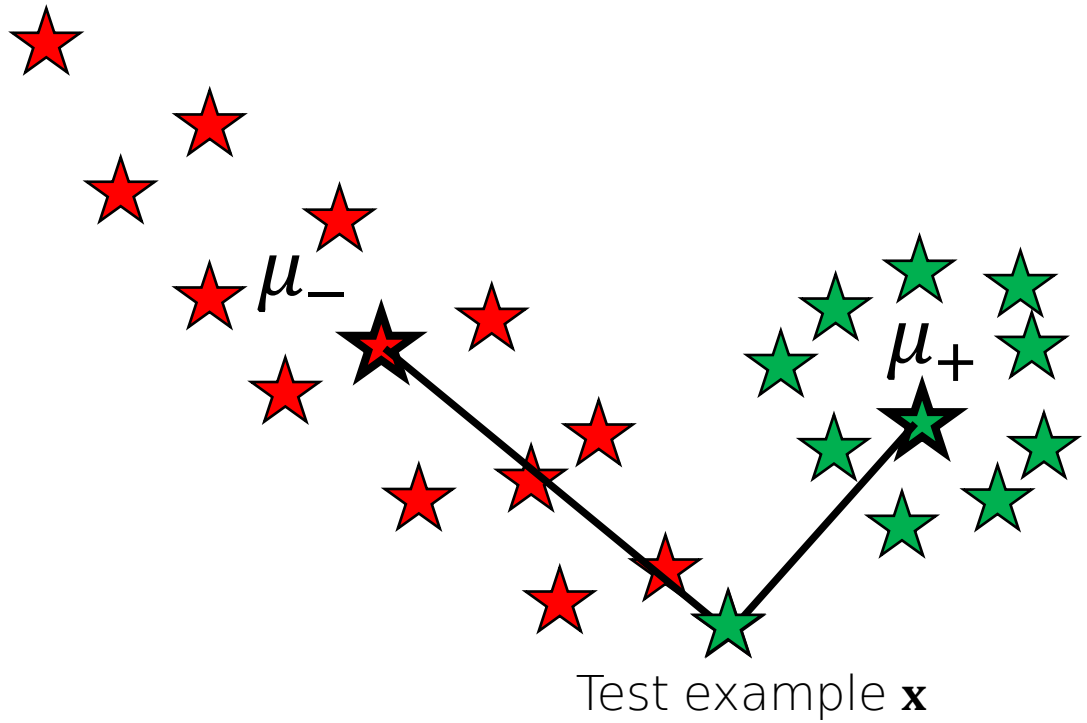
- Thus LwP with Euclidean distance is equivalent to a linear model with
  - Weight vector $\mathbf{w} = 2(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$
  - Bias term $b = \left|\left|\boldsymbol{\mu}_-\right|\right|^2 - \left|\left|\boldsymbol{\mu}_+\right|\right|^2$

Will look at linear models more formally and in more detail later

- Prediction rule therefore is: Predict +1 if $\langle\mathbf{w}, \mathbf{x}\rangle + b > 0$, else

LwP: Some Failure Cases

- Here is a case where LwP with Euclidean distance may not work well



Can use feature scaling or use Mahalanobis distance to handle such cases (will discuss this in the next lecture)

- In general, if classes are not equisized and spherical, LwP with Euclidean distance will usually not work well (but improvements possible; will discuss later)

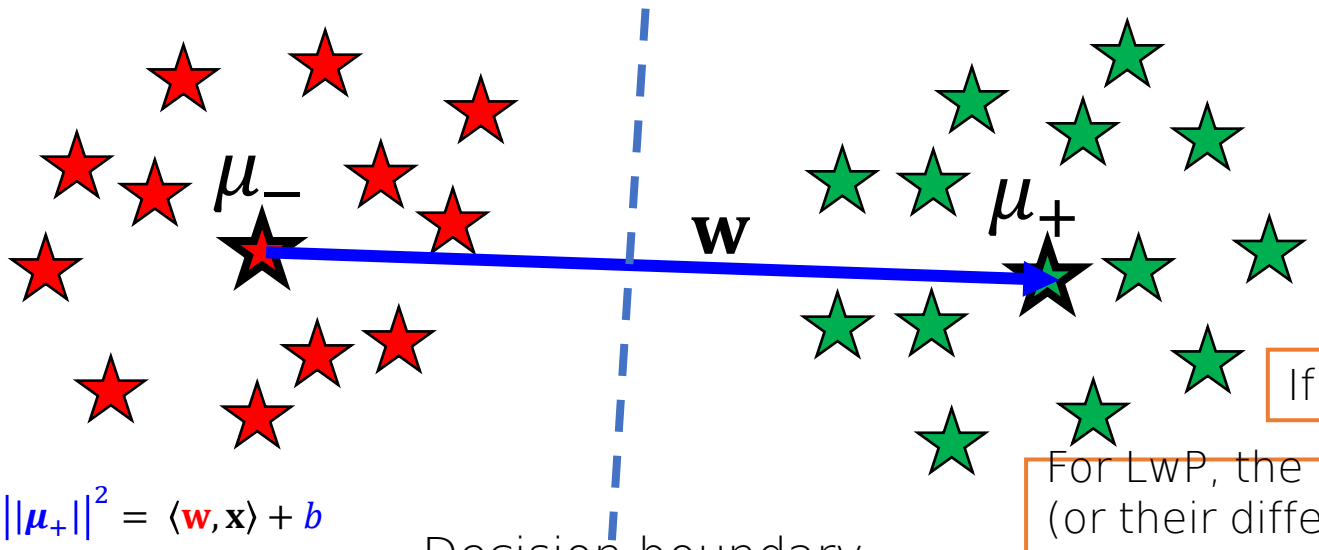
LwP: Some Key Aspects

- Very simple, interpretable, and lightweight model
  - Just requires computing and storing the class prototype vectors
- Works with any number of classes (thus for multi-class classification as well)
- Can be generalized in various ways to improve it further, e.g.,
  - Modeling each class by a probability distribution rather than just a prototype vector
  - Using distances other than the standard Euclidean distance (e.g., Mahalanobis)
- With a learned distance function, can work very well even with very few examples from each class (used in some "few-shot

Learning with Prototypes (LwP)

$$\mu_- = \frac{1}{N_-}\sum_{y_n=-1}\mathbf{x}_n$$

$$\mu_+ = \frac{1}{N_+}\sum_{y_n=+1}\mathbf{x}_n$$



$$\mathbf{w} = \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-$$

If Euclidean distance used

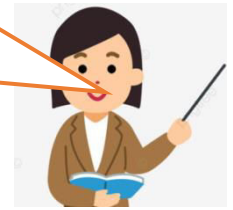
Prediction rule for LwP (for binary classification with Euclidean distance)

$$f(\mathbf{x}) = 2\langle \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-, \mathbf{x}\rangle + ||\boldsymbol{\mu}_-||^2 - ||\boldsymbol{\mu}_+||^2 = \langle \mathbf{w}, \mathbf{x}\rangle + b$$

($f(\mathbf{x}) > 0$ then predict +1 otherwise -1)

Decision boundary (perpendicular bisector of line joining the class prototype vectors)

For LwP, the prototype vectors (or their difference) define the "model". μ_+ and μ_- (or just $\mathbf{w}$ in the Euclidean distance case) are the model parameters.



Exercise: Show that for the bin. classfn case

$$f(\mathbf{x}) = \sum_{n=1}^{N}\alpha_n\langle \mathbf{x}_n, \mathbf{x}\rangle + b$$

Note: Even though $f(\mathbf{x})$ can be expressed in this form, if $N > D$, this may be more expensive to compute (O(N) time)as compared to $\langle \mathbf{w}, \mathbf{x}\rangle + b$ (O(D) time).

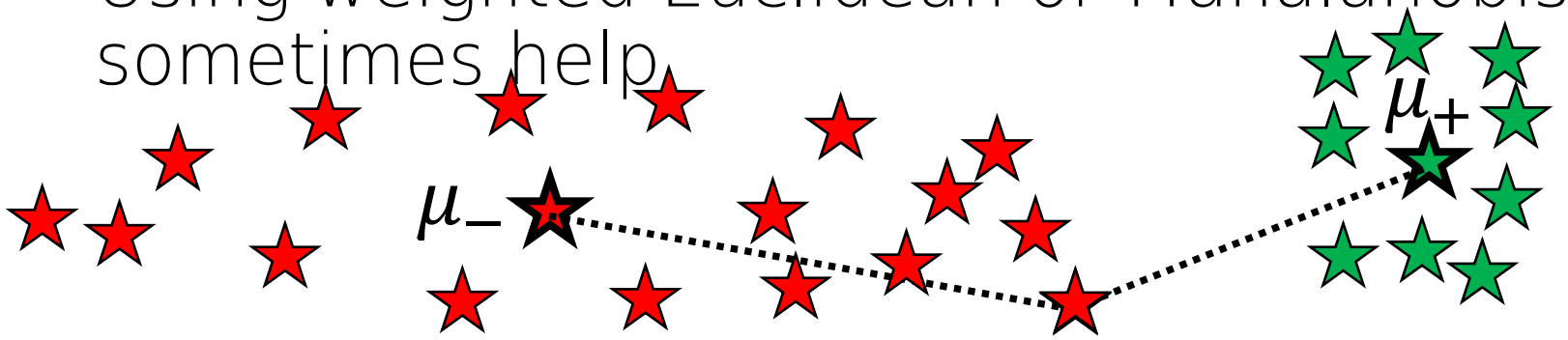
Can throw away training data after computing the prototypes and just need to keep the model parameters for the test time in such "parametric" models

So the "score" of a test point $\mathbf{x}$ is a weighted sum of its similarities with each of the N training inputs. Many supervised learning models have $f(\mathbf{x})$ in this form as we will see later

However the form $f(\mathbf{x}) = \sum_{n=1}^{N}\alpha_n\langle \mathbf{x}_n, \mathbf{x}\rangle + b$ is still very useful as we will see later when we discuss kernel methods

Improving LwP when classes are complex-shaped

- Using weighted Euclidean or Mahalanobis distance can sometimes help



$$d_w(\boldsymbol{a},\boldsymbol{b}) = \sqrt{\sum_{i=1}^{D} w_i (a_i - b_i)^2}$$

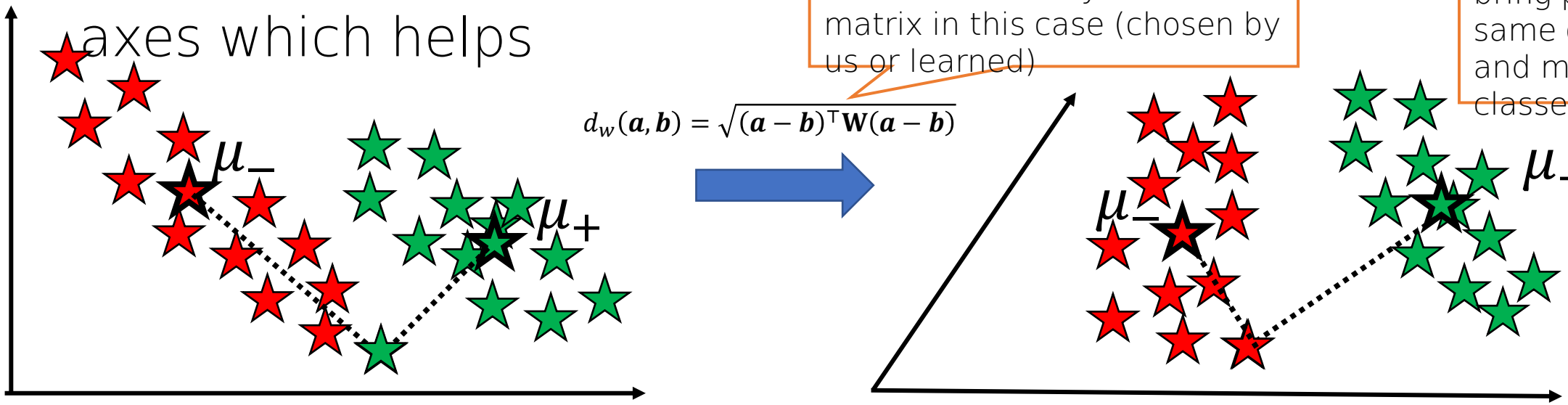
Use a smaller w_i for the horizontal axis feature in this example

- Note: Mahalanobis distance also has the effect of ro axes which helps

$$d_w(\boldsymbol{a},\boldsymbol{b}) = \sqrt{(\boldsymbol{a}-\boldsymbol{b})^\top \mathbf{W} (\boldsymbol{a}-\boldsymbol{b})}$$

$\mathbf{W}$ will be a 2x2 symmetric matrix in this case (chosen by us or learned)

A good $\mathbf{W}$ will help bring points from same class closer and move different classes apart

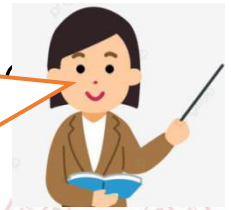


Improving LwP when classes are complex-shaped

- Even with weighted Euclidean or Mahalanobis dist, LwP still a linear classifier ☹
- Exercise: Prove the above fact. You may use the following hint
  - Mahalanobis dist can be written as $d_{\mathbf{W}}(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(\boldsymbol{a}-\boldsymbol{b})^\top \mathbf{W} (\boldsymbol{a}-\boldsymbol{b})}$
  - $\mathbf{W}$ is a symmetric matrix and thus can be written as $\mathbf{A}\mathbf{A}^\top$ for any matrix $\mathbf{A}$
  - Showing for Mahalabonis is enough. Weighted Euclidean is a special case with diag $\mathbf{W}$

- LwP can be extended to learn nonlin… …ri… we use nonlinear distances/similarit… …en talk about kernels)

Note: Modeling each class by not just a mean by a probability distribution can also help in learning nonlinear decision boundaries. More on this when we discuss probabilistic models for classification



LwP as a subroutine in other ML models

- For data-clustering (unsupervised learning), *K*-means clustering is a popular algo

- *K*-means also computes means/centres/prototypes of groups of unlabeled points
- Harder than LwP since labels are unknown. But we can do
  - Guess the label of each point, compute means using guess labels
  - Refine labels using these means (assign each point to the current closest mean)
  - Repeat until means don't change anymore

Next Lecture

- Nearest Neighbors

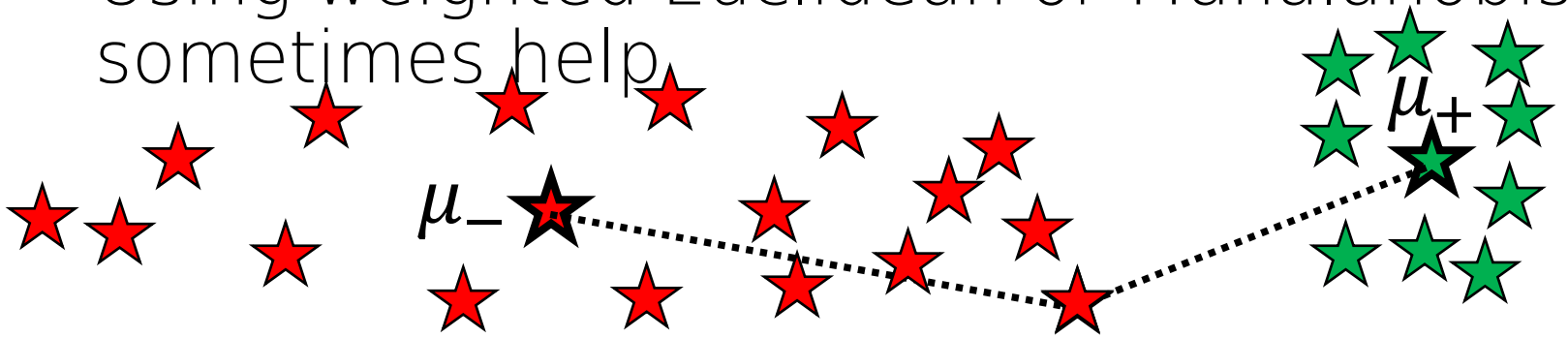
Exotic distances and nearest neighbors

CS771: Introduction to Machine Learning

Nisheeth

Improving LwP when classes are complex-shaped

- Using weighted Euclidean or Mahalanobis distance can sometimes help



$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{D} w_i (a_i - b_i)^2}$$

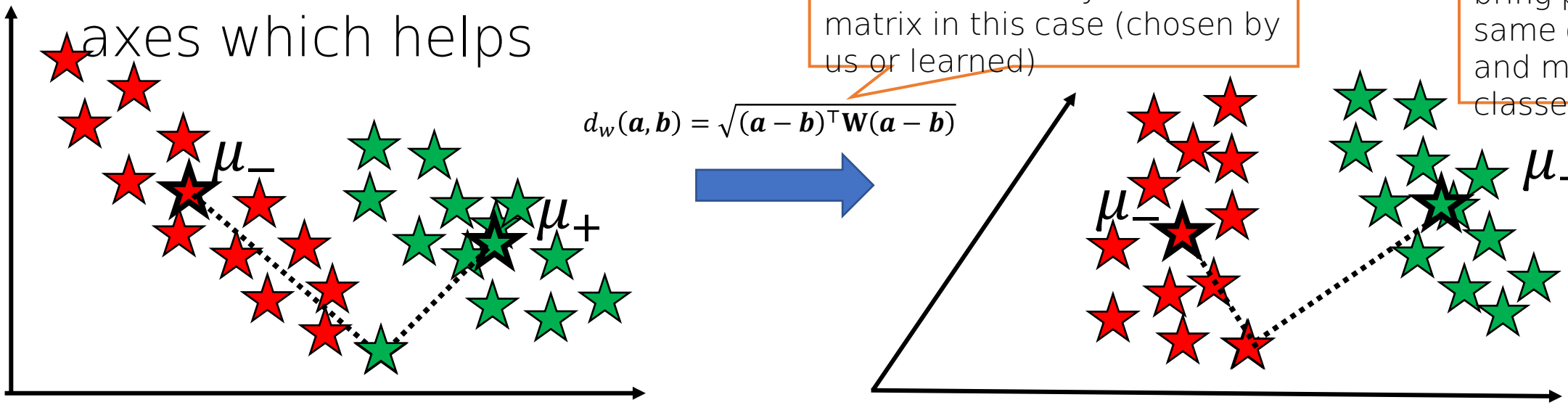
Use a smaller w_i for the horizontal axis feature in this example

- Note: Mahalanobis distance also has the effect of ro… axes which helps

$$d_w(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^\top \mathbf{W} (\boldsymbol{a} - \boldsymbol{b})}$$

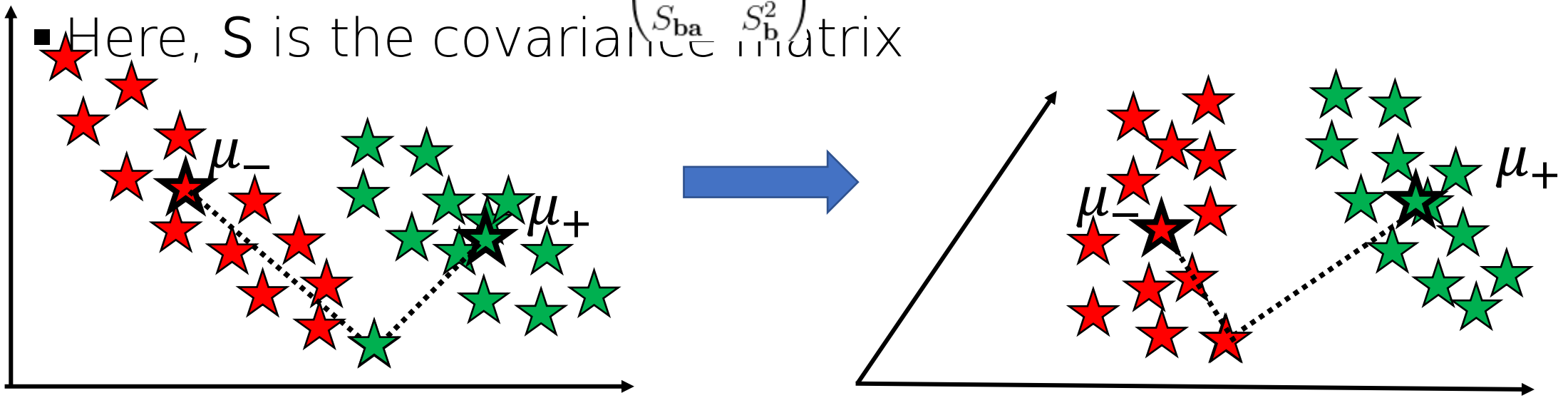
$\mathbf{W}$ will be a 2x2 symmetric matrix in this case (chosen by us or learned)

A good $\mathbf{W}$ will help bring points from same class closer and move different classes apart



What is Mahalanobis Distance?

- Recall $\text{Euclidean } D(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^T(\mathbf{a}-\mathbf{b})}$
- And its generali$\text{Weighted Euclidean } D(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^T\mathbf{W}(\mathbf{a}-\mathbf{b})}$
  - Where **W** is a diagonal matrix
- The Mahalanobis distance further generalizes the weighted Euclidean distar$\text{Mahalanobis } D(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^T\mathbf{S}^{-1}(\mathbf{a}-\mathbf{b})}$
- Here, **S** is the covariance matrix $\begin{pmatrix} S_{\mathbf{a}}^2 & S_{\mathbf{ab}} \\ S_{\mathbf{ba}} & S_{\mathbf{b}}^2 \end{pmatrix}$



Supervised Learning using Nearest Neighbors

Nearest Neighbors

- Another supervised learning technique based on computing distances

- Very simple idea. Simply do the following at test time
  - Compute distance of of the test point from all the training points
  - Sort the distances to find the "nearest" input(s) in training data
  - Predict the label using majority or avg label of these inputs

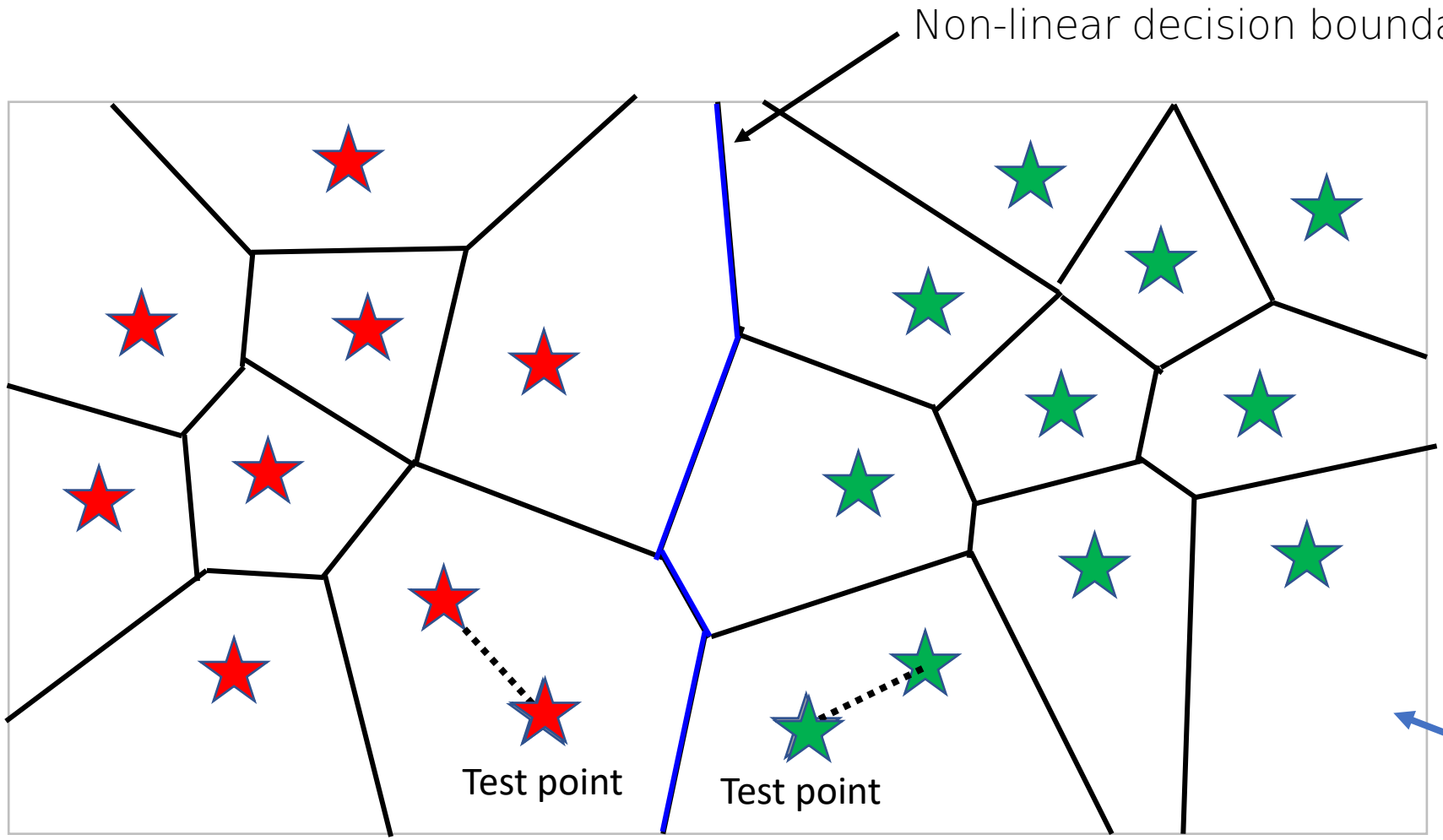
- Can use Euclidean or other dist (e.g., Mahalanobis). Choice important just like LwP

- Unlike LwP which does prototype based comparison, nearest neighbors method looks at the labels of individual training inputs to make prediction

Nearest Neighbors for Classification

Reference material

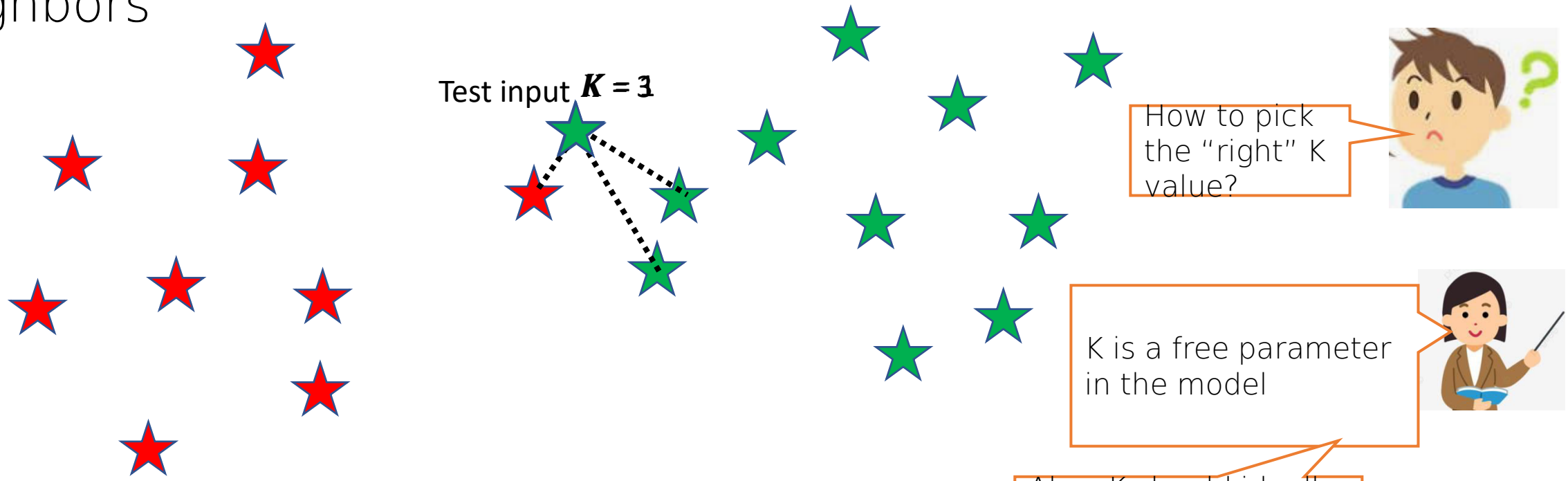
Nearest Neighbor (or "One" Nearest Neighbor)



Nearest neighbour approach induces a Voronoi tessellation/partition of the input space (all test points falling in a cell will get the label of the training input in that cell)

K Nearest Neighbors (*K*NN)

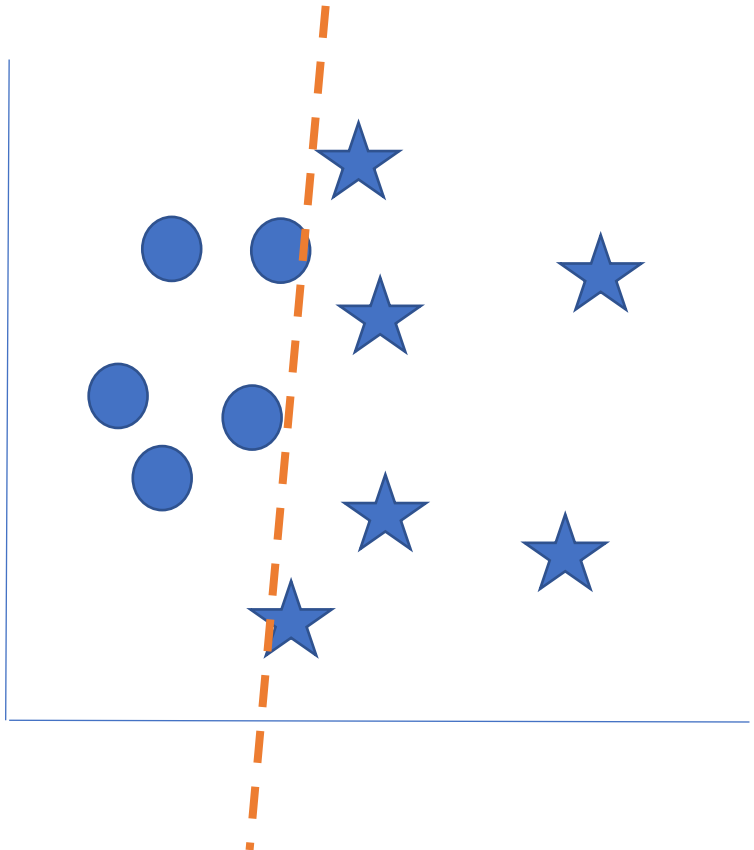
- In many cases, it helps to look at not one but $K > 1$ nearest neighbors



- Essentially, taking more votes helps!
  - Also leads to smoother decision boundaries (less chances of overfitting on training data)

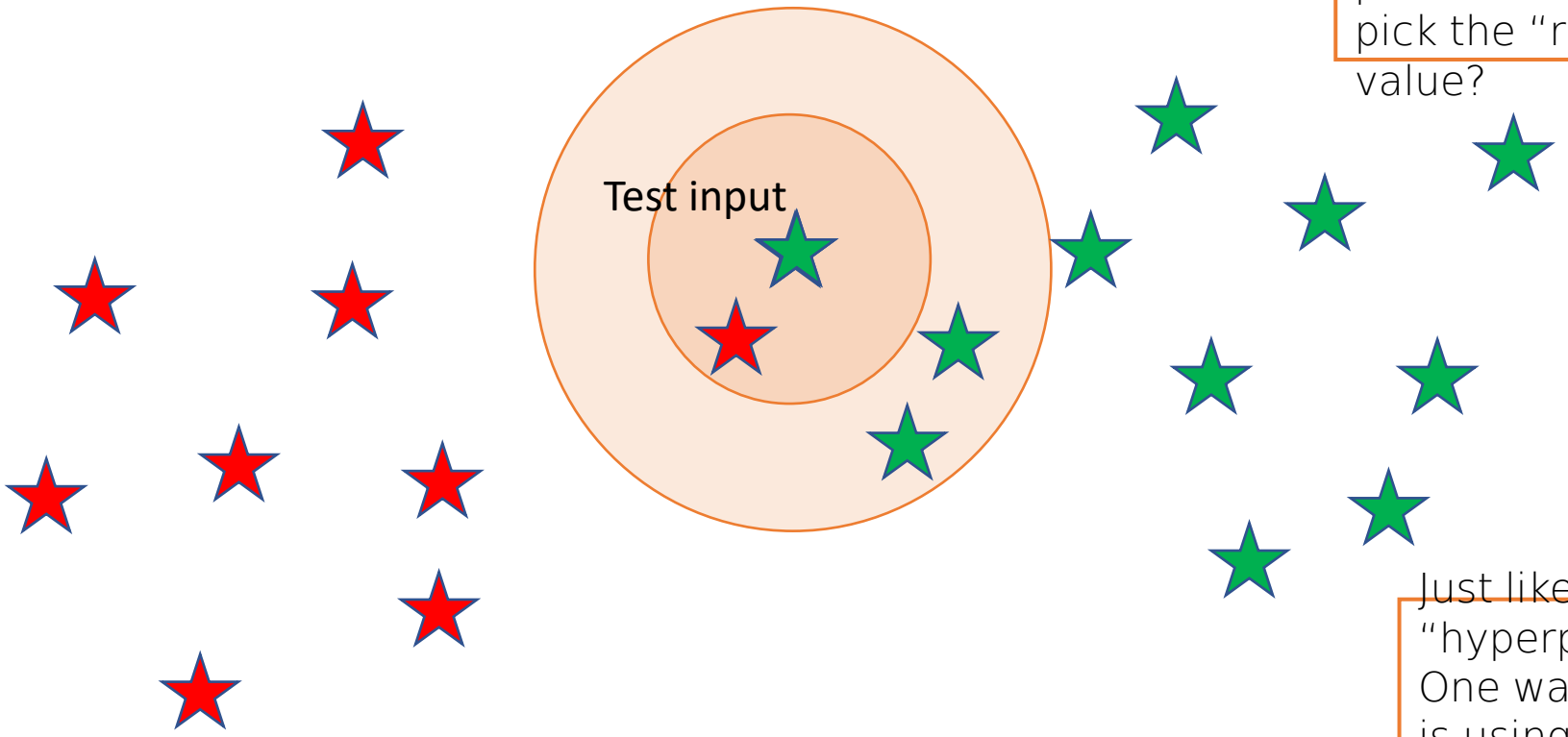
Setting parameter values

- The black magic of machine learning
  - "Predictions are hard, especially about the future" – Yogi Berra
- Basic idea: find parameter values for which you make the fewest possible mistakes on training data
- Pray that training data is representative
- If it's not, your model will work badly
- We will see some nice math that helps
- Soon!



ϵ-Ball Nearest Neighbors (ϵ-NN)

- Rather than looking at a fixed number K of neighbors, can look inside a ball of a given radius ϵ, around the test input

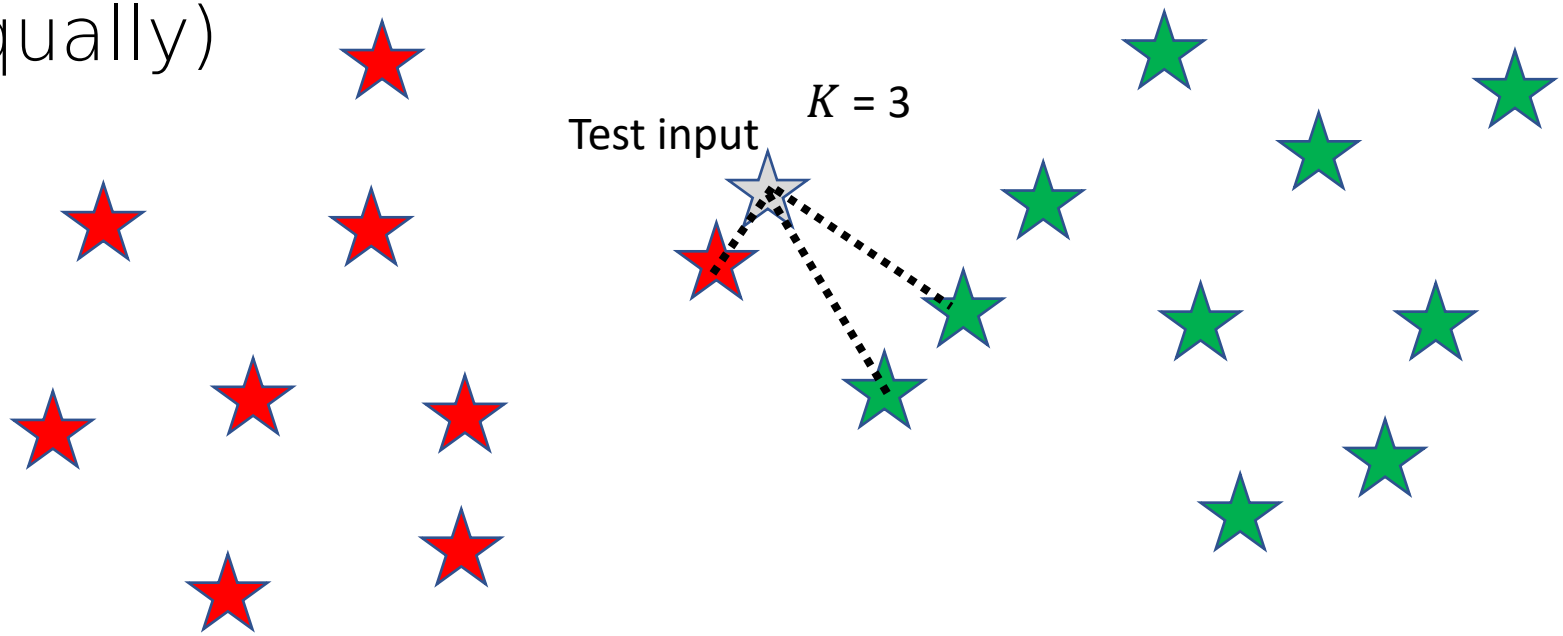


So changing ϵ may change the prediction. How to pick the "right" ϵ value?

Just like K, ϵ is also a "hyperparameter". One way to choose it is using "cross-validation" (will see shortly)

Distance-weighted *K*NN and ϵ-NN

- The standard KNN and ϵ-NN treat all nearest neighbors equally (all vote equally)



- An improvement: When voting, give more importance to closer training inputs

Unweighted KNN prediction: $\frac{1}{3}$ ★ + $\frac{1}{3}$ ★ + $\frac{1}{3}$ ★ = ★

Weighted KNN prediction: $\frac{3}{5}$ ★ + $\frac{1}{5}$ ★ + $\frac{1}{5}$ ★ = ★

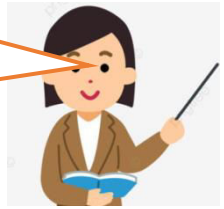
In weighted approach, a single red training input is being given 3 times more importance than the other two green inputs since it is sort of "three times" closer to the test input than the other two green inputs

ϵ-NN can also be made weighted likewise

*K*NN/ϵ-NN for Other Supervised Learning Problems

- Can apply KNN/ϵ-NN for other supervised learning problems as well, such as
  - Multi-class classification
  - Regression
  - Tagging/multi-label learning

We can also try the weighted versions for such problems, just like we did in the case of binary classification



- For multi-class, simply used the same majority rule like in binary classfication case
  - Just a simple difference that now we have more than 2 classes

- For regression, simply compute the average of the outputs of nearest neighbors

- For multi-label learning, each output is a binary vector

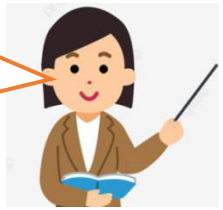
*K*NN Prediction Rule: The Mathematical Form

- Let's denote the set of K nearest neighbors of an input $\mathbf{x}$ by $N_K(\mathbf{x})$

- The unweighted KNN prediction $\mathbf{y}$ can be written as

$$\mathbf{y} = \frac{1}{K} \sum_{i \in N_K(\mathbf{x})} \mathbf{y}_i$$

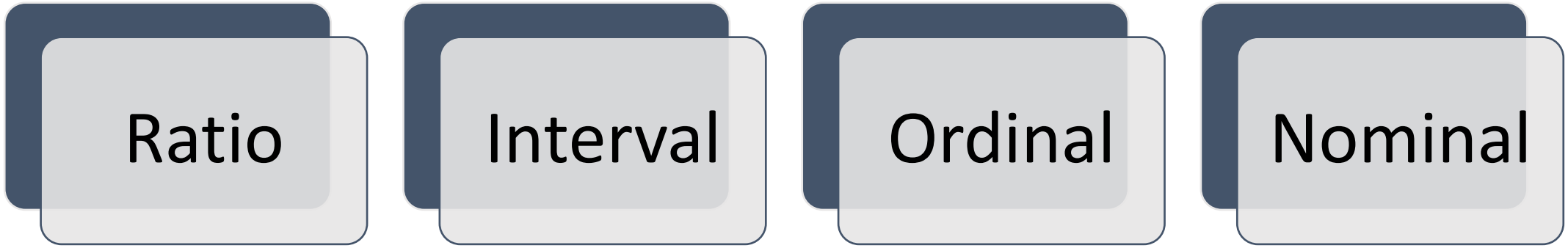
For discrete labels with 5 possible values, the one-hot representation will be an all zeros vector of size 5, except a single 1 denoting the value of the discrete label, e.g., if label = 3 then one-hot vector = [0,0,1,0,0]



- This form makes direct sense of regression and for cases where the each output is a vector (e.g., multi-class classification where each output is a discrete value which can be represented as a one-hot vector, or tagging/multi-label classification where each output is a binary vector)

Nearest Neighbours: Some Comments

- An old, classic but still very widely used algorithm
  - Competitive with state-of-the-art approaches most of the time
- Can work very well in practice with the right distance function
  - How do we pick the right distance function?
- Requires lots of storage (need to keep all the training data at test time)
- Prediction step can be slow at test time
  - For each test point, need to compute its distance from all the training points
- Clever data-structures or data-summarization techniques can provide speed-ups



Data types determine distance choice

Ratio

- Ratio data can be reasonably compared as multiples or fractions of each other
- Measure quantities
- Zero has a meaning
- Examples:
  - Someone's weight
  - A bottle's capacity

Interval

- Can be measured on a scale
- Differences between values are meaningful
- Multiples and fractions of measurements are not meaningful
- Examples:
  - Customer satisfaction ratings
  - Temperature, in Fahrenheit

Ordinal

- Can be expressed in terms of rank or position
- Only relative ranks are meaningful
- Cannot do arithmetic with it
- Examples:
  - Positions in a race
  - Protein sequence in genome

Nominal

- Items in a nominal scale belong to a category
- No other information about them
- Can be numeric, but this does not make them ordinal
- Examples:
  - Phone numbers
  - People in CS771

From data to distance

Attribute Type	**Dissimilarity**	**Similarity**
Nominal	$d = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$	$s = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$
Ordinal	$d = \lvert x - y \rvert / (n-1)$ (values mapped to integers 0 to $n-1$, where n is the number of values)	$s = 1 - d$
Interval or Ratio	$d = \lvert x - y \rvert$	$s = -d$, $s = \frac{1}{1+d}$, $s = e^{-d}$, $s = 1 - \frac{d - min\_d}{max\_d - min\_d}$

Nominal and ordinal values are usually converted to one-hot vector representations

Have to be careful this representation does not cloud their meaning

For ratio and interval-scaled features

- Euclidean distance is your basic workhorse
  - Use it if you can think of nothing else
- Squared Euclidean distance will also work fine if your features are range normalized
  - Useful when you want to save the compute of taking square roots
- Manhattan distance is also useful when time presses
  - Only simple additions and subtractions involved
- More exotic metrics may be needed in specific situations
  - Read this for some examples

Similarity for binary-valued vectors

- Independent of feature type
- Compute the following quantities
  - F01 = number of attributes where x is 0 and y is 1
  - F10 = number of attributes where x is 1 and y is 0
  - F00 = number of attributes where x is 0 and y is 0
  - F11 = number of attributes where x is 1 and y is 1

For binary vectors

Simple matching coefficient

$$\frac{F11 + F00}{F01 + F10 + F11 + F00}$$

Jaccard Coefficient

$$\frac{F11}{F01 + F10 + F11}$$

For general vector representations

- We use a cosine similarity measure

$$cos(\mathbf{d_1}, \mathbf{d_2}) = \frac{< \mathbf{d_1}, \mathbf{d_2} >}{\|\mathbf{d_1}\|\|\mathbf{d_2}\|}$$

- Where <> indicates a dot product and || || measures the distance of each vector to the origin
- Bigger numbers mean the two items are closer
- Invert it to obtain a distance measure
  - Additively, multiplicatively or exponentially

Next Lecture

- Hyperparameter/model selection via cross-validation
- Learning with Decision Trees

Sources of error

CS771: Introduction to Machine Learning

Nisheeth

Plan for today

- Understanding error in machine learning
- Cross-validation
- Learning with Decision Trees

Hyperparameter Selection

- Every ML model has some hyperparameters that need to be tuned, e.g.,
  - K in KNN or ϵ in ϵ-NN
  - Choice of distance to use in LwP or nearest neighbors

- Would like to choose h.p. values that would give best performance on test data

Generalization

Training set (labels known)

Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

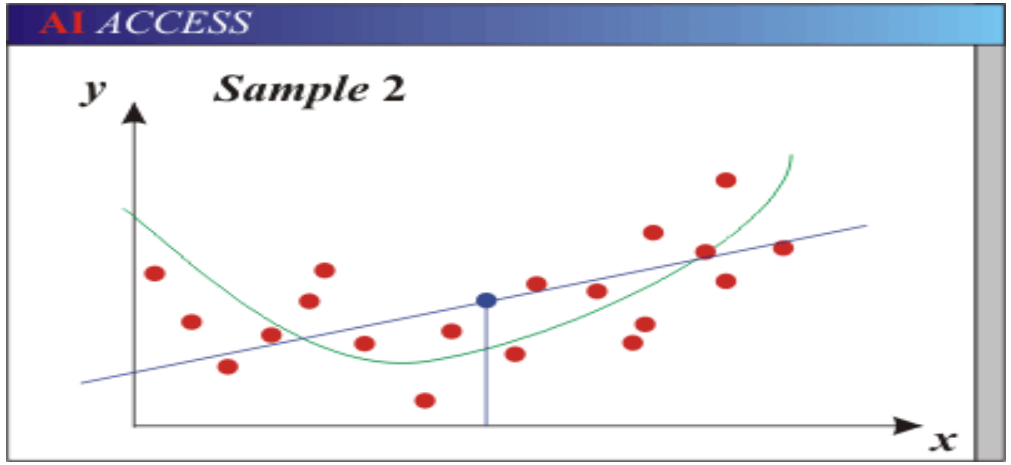
Slide credit: L. Lazebnik

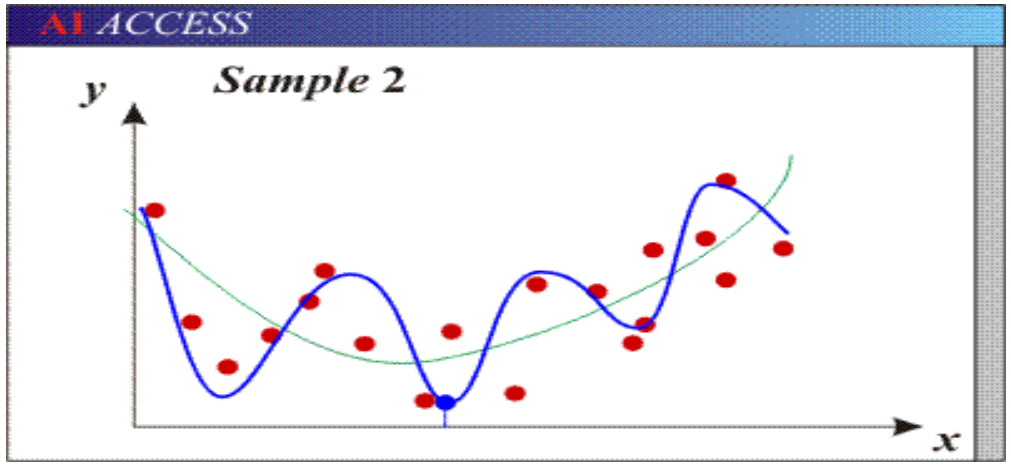
Generalization

- Components of generalization error
  - **Bias:** how much the average model over all training sets differ from the true model?
    - Error due to inaccurate assumptions/simplifications made by the model
  - **Variance:** how much models estimated from different training sets differ from each other
- **Underfitting:** model is too "simple" to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error
- **Overfitting:** model is too "complex" and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

Slide credit: L. Lazebnik

Bias-Variance Trade-off





- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).
- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).

Slide credit: D. Hoiem

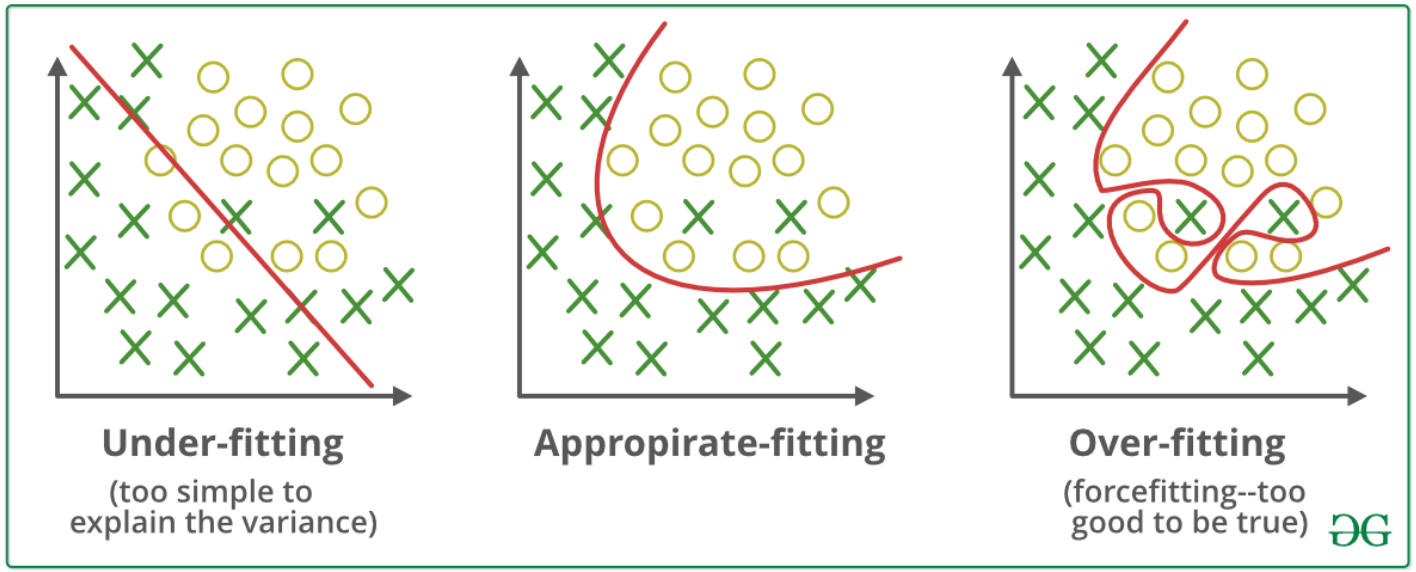
Bias-Variance Trade-off

$$E(MSE) = noise^2 + bias^2 + variance$$

Unavoidable error

Error due to incorrect assumptions

Error due to variance of training samples

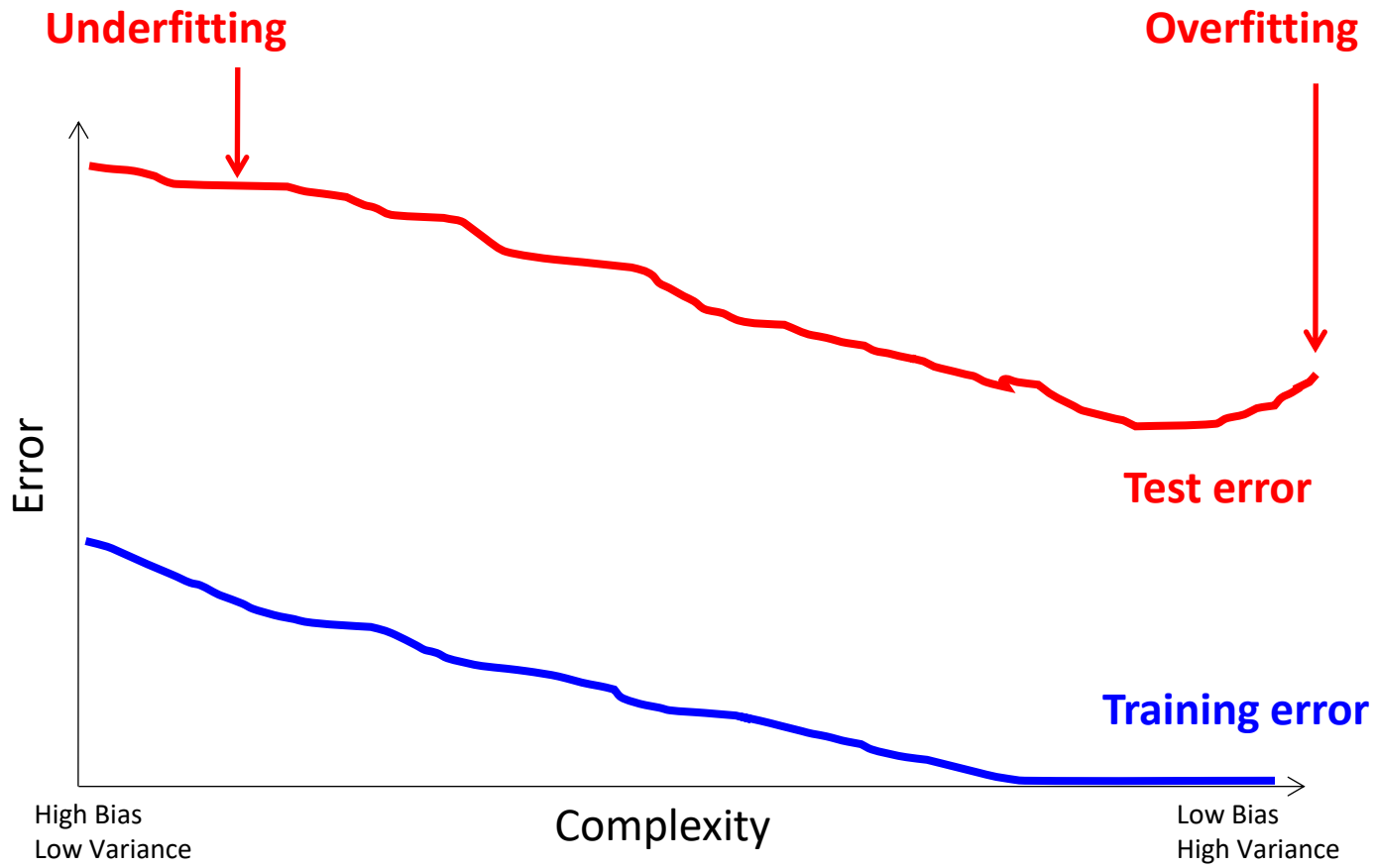


See the following for explanations of bias-variance (also Bishop's "Neural Networks" book):

•http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

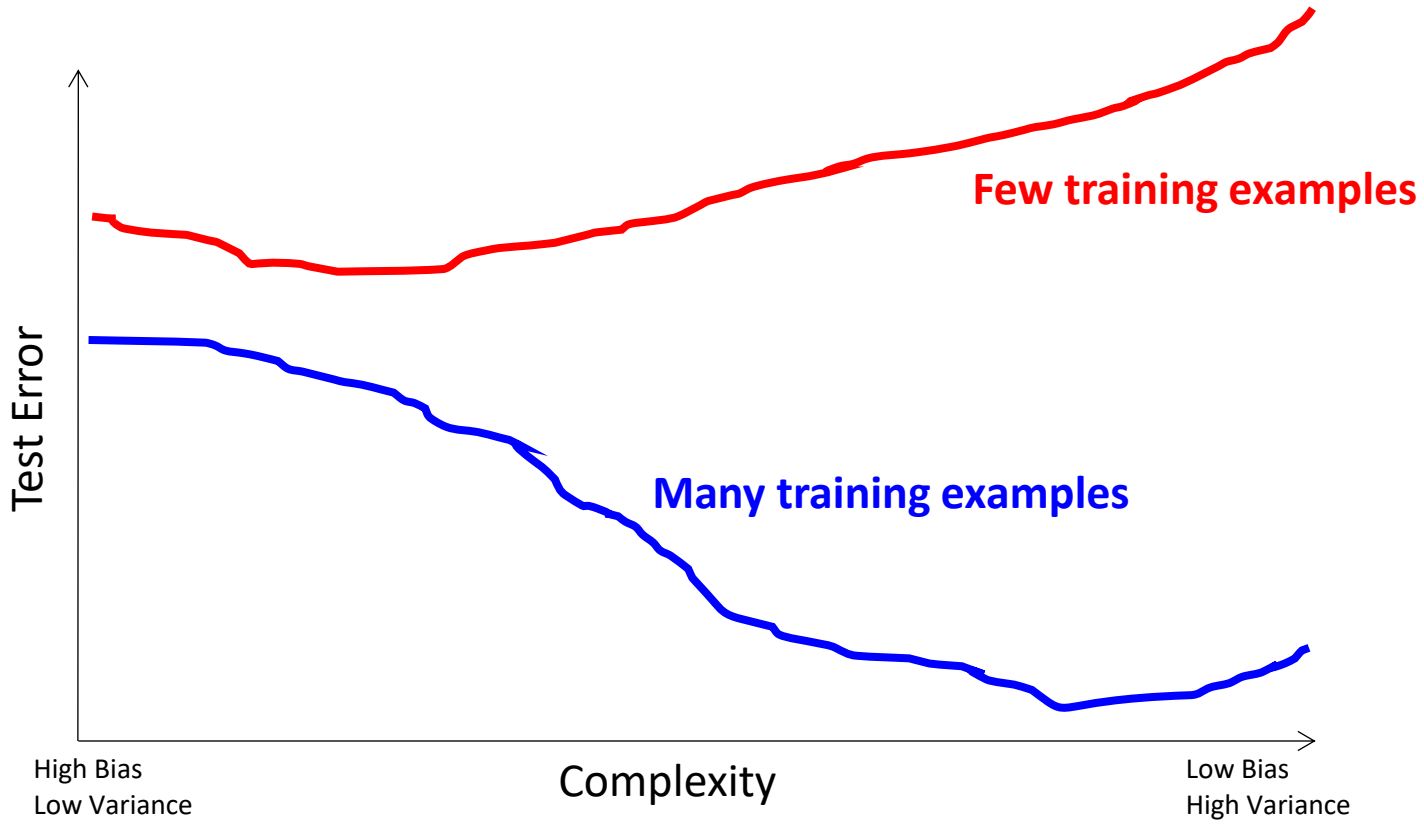
Image credit: geeksforgeeks.com
Slide credit: D. Hoiem

Bias-variance tradeoff



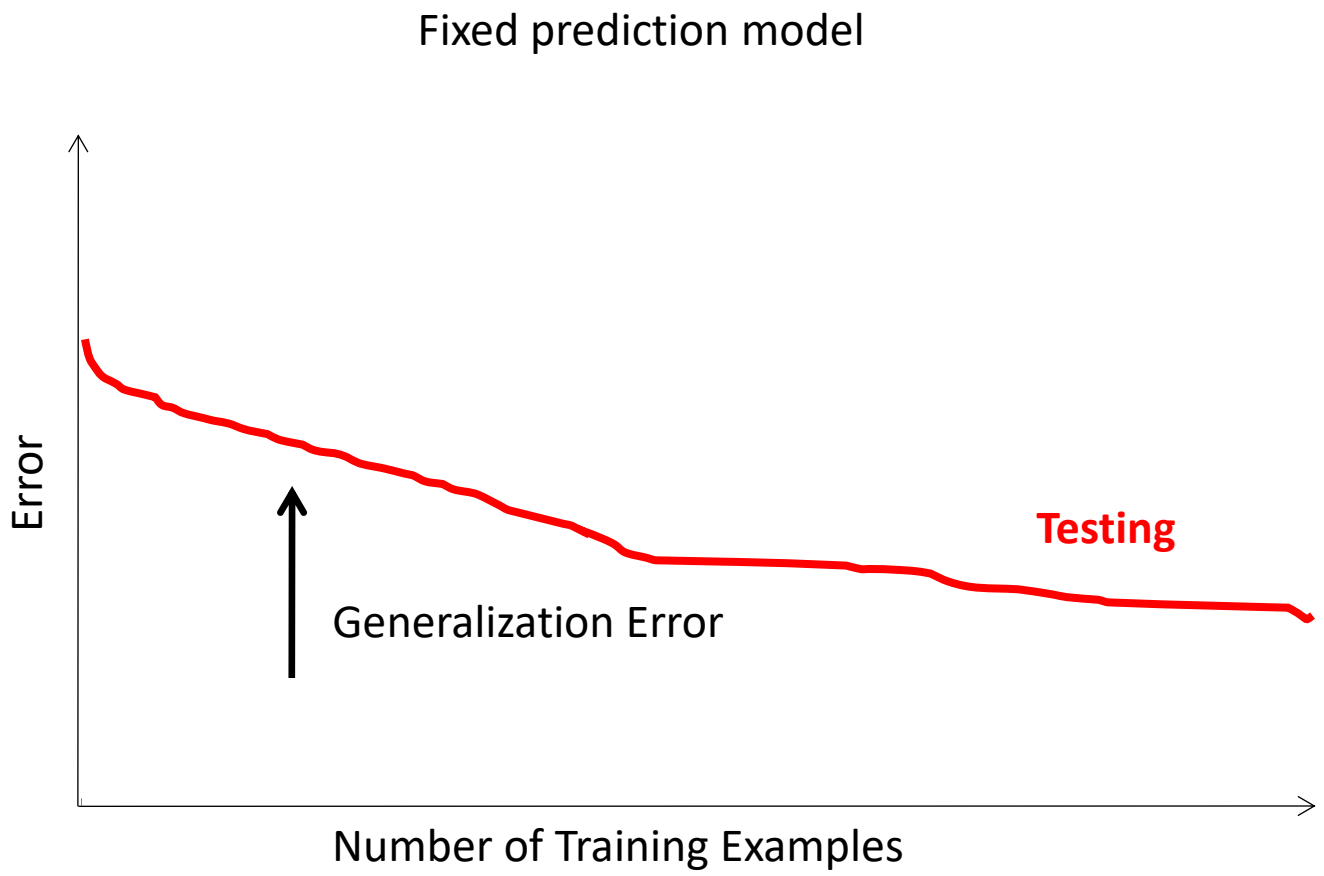
Slide credit: D. Hoiem

Bias-variance tradeoff



Slide credit: D. Hoiem

Effect of Training Size



Slide credit: D. Hoiem

Remember…

- No classifier is inherently better than any other: you need to make assumptions to generalize

- Three kinds of error
  - Inherent: unavoidable
  - Bias: due to over-simplifications
  - Variance: due to inability to perfectly estimate parameters from limited data

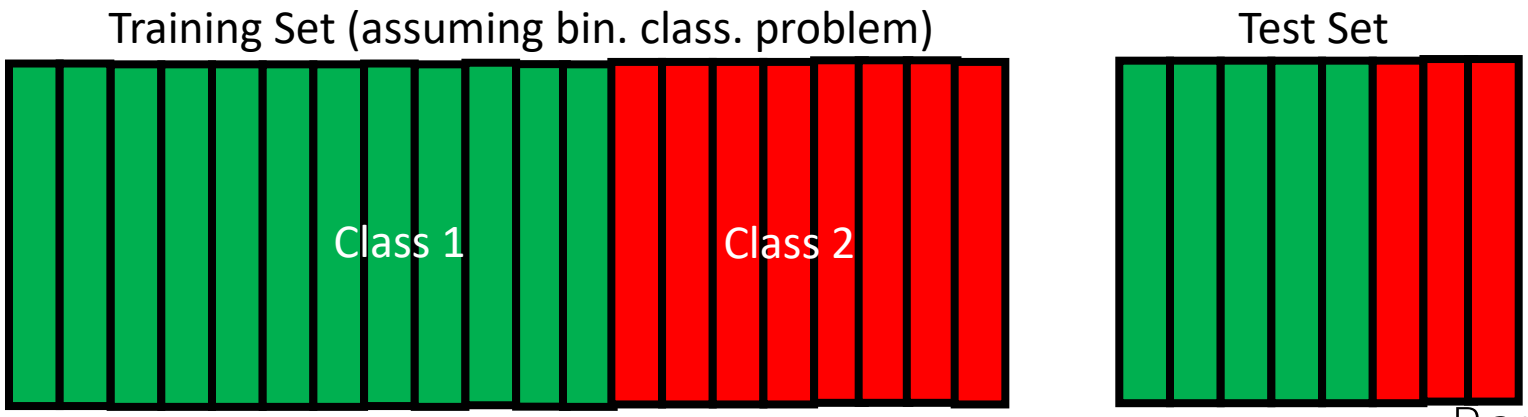
How to reduce variance?

- Choose a simpler classifier
- Cross-validate the parameters
- Get more training data

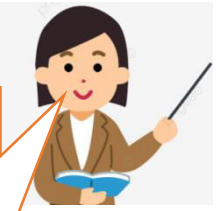
Slide credit: D. Hoiem

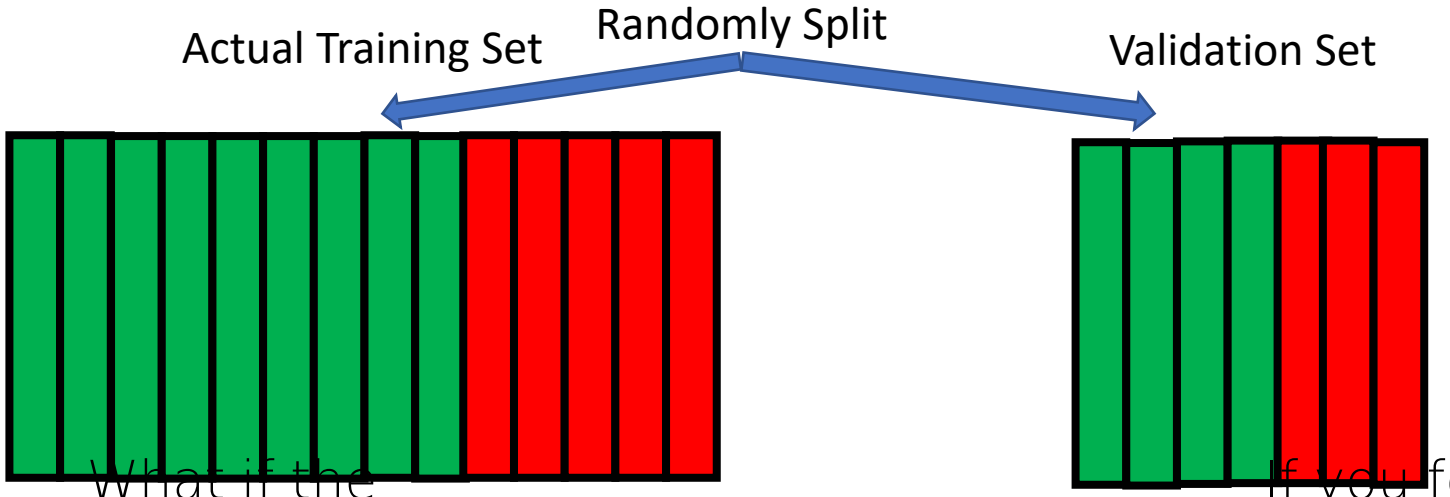
Cross-Validation

No peeking while building the model



Note: Not just h.p. selection; we can also use CV to pick the best ML model from a set of different ML models (e.g., say we have to pick between two models we may have trained - LwP and nearest neighbors. Can use CV to choose the better one.





Randomly split the training data into actual training set and validation set. Using the actual training set, train several times, each time using a different value of the hyperparam. Pick the hyperparam value that gives best accuracy on the validation set

What if the random split is unlucky (i.e., validation data is not like test data)?

If you fear an unlucky split, try multiple splits. Pick the hyperparam value that gives the best average CV accuracy across all such splits. If you are using N splits, this is called N–fold cross validation

Learning using Decision Trees

CS771: Introduction to Machine Learning

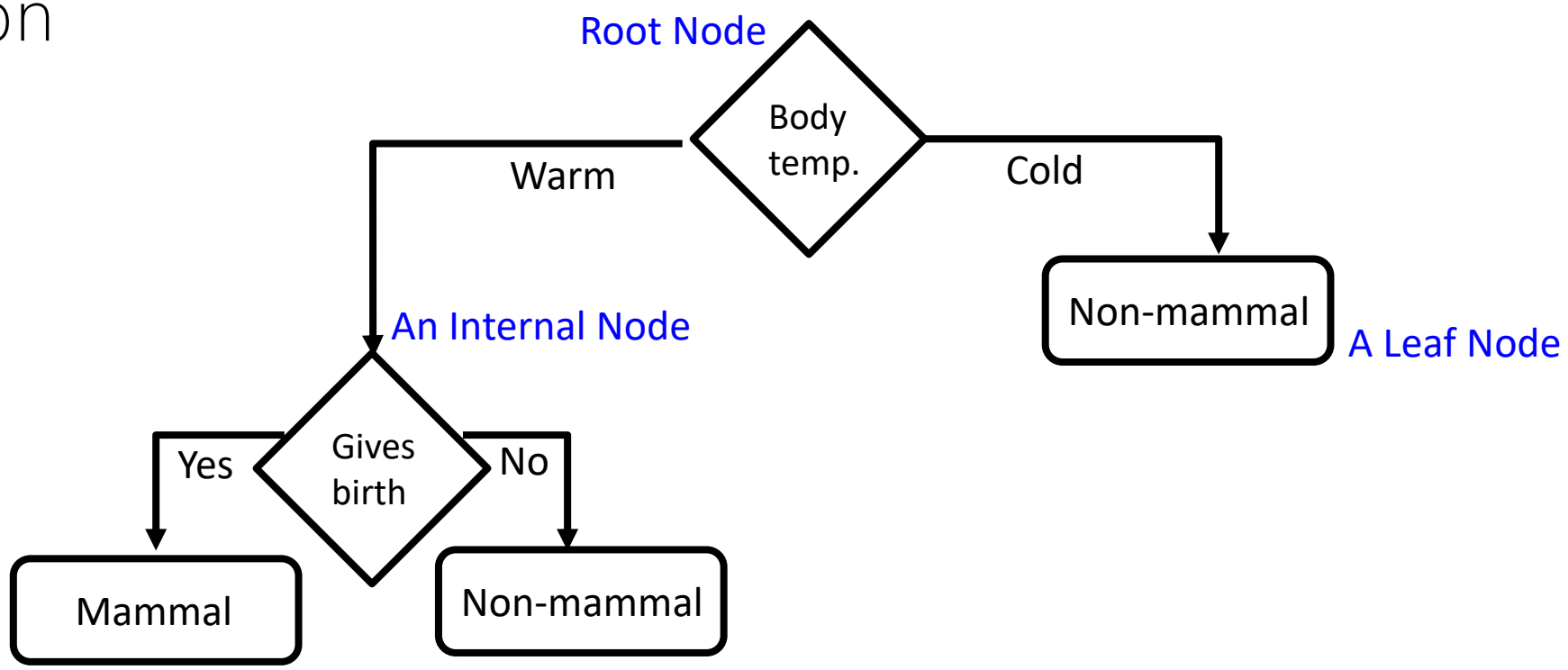
Nisheeth

Announcements

- Today: learning with Decision Trees
- Quiz 1: Next Friday (27th Aug, in-class hours)
  - Syllabus: everything we will have covered by the end of today
- No class this Friday (Muharram holiday)

Decision Trees

- A Decision Tree (DT) defines a hierarchy of rules to make a prediction



- Root and internal nodes test rules. Leaf nodes make predictions

- Decision Tree (DT) learning is about learning such a tree from

A decision tree friendly problem

Loan approval prediction

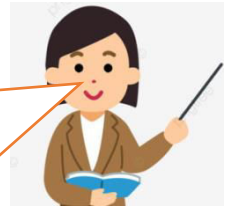
ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

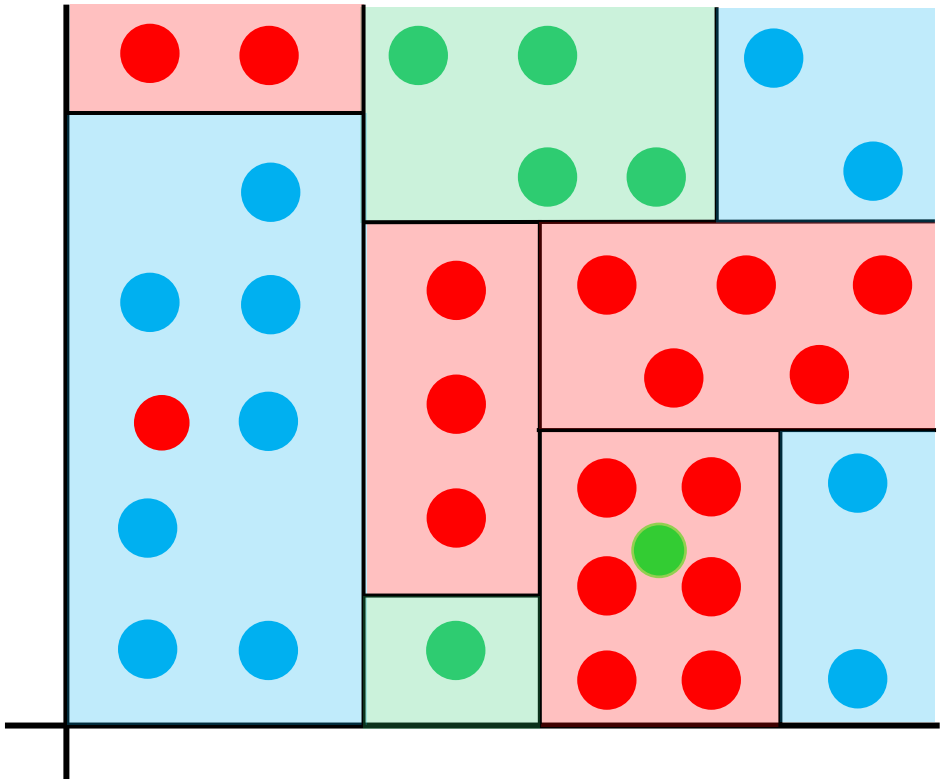
Learning Decision Trees with Supervision

- The basic idea is very simple
- Recursively partition the training data into homogeneous regions

What do you mean by "homogeneous" regions?

A homogeneous region will have all (or a majority of) training inputs with the same/similar outputs

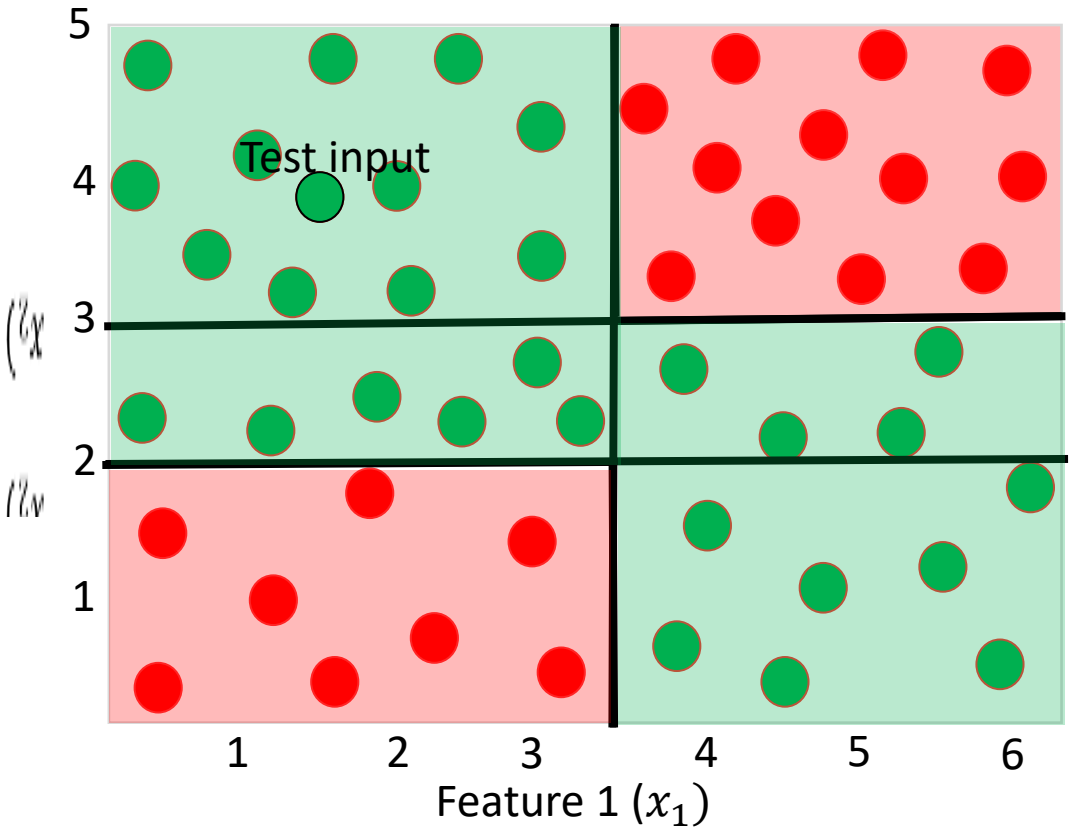


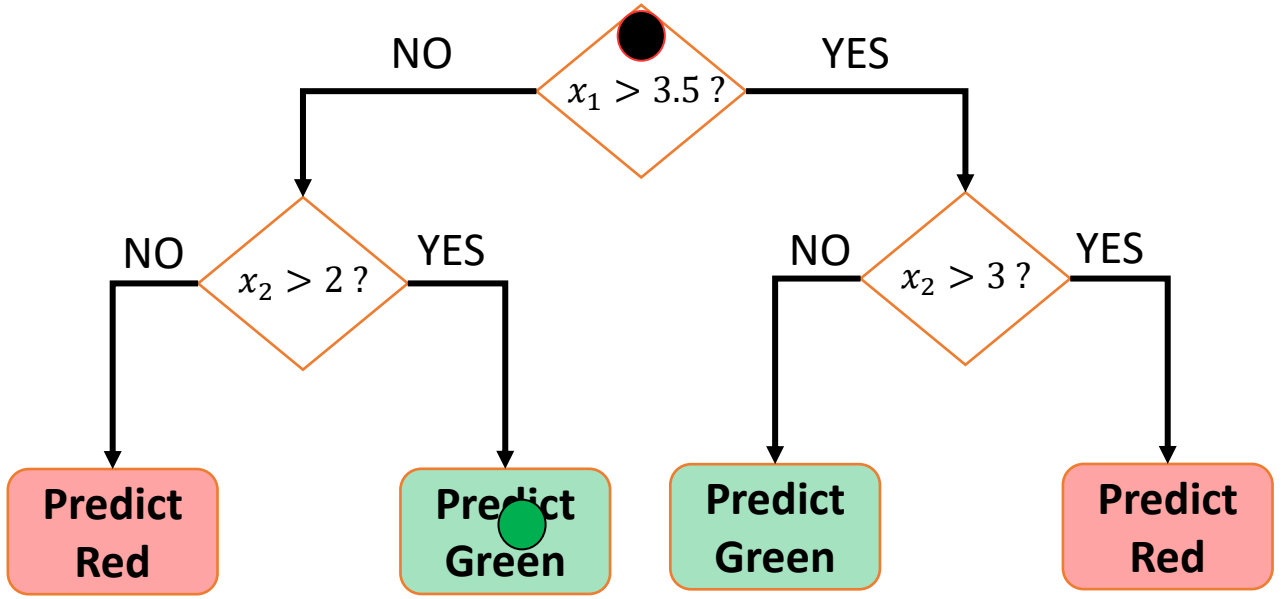


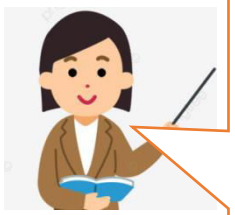
Even though the rule within each group is simple, we are able to learn a fairly sophisticated model overall (note in this example, each rule is a simple horizontal/vertical classifier but the overall decision boundary is rather sophisticated)

- Within each group, fit a simple supervised learner (e.g., predict

Decision Trees for Classification







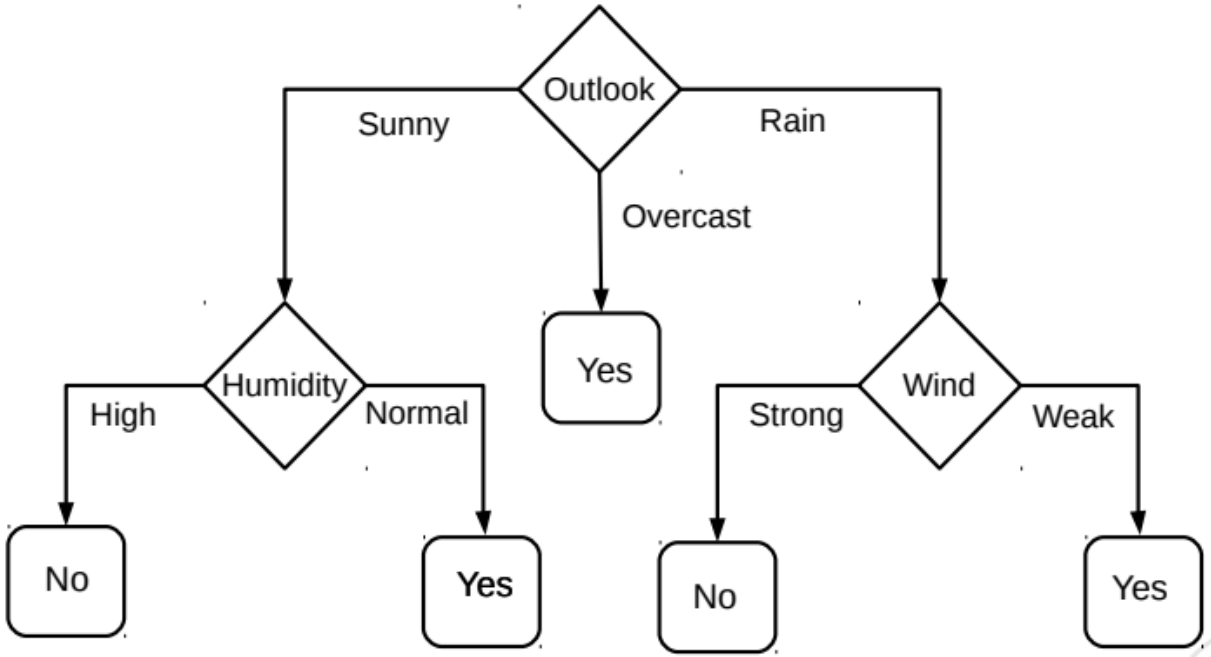
DT is very efficient at test time: To predict the label of a test point, nearest neighbors will require computing distances from 48 training inputs. DT predicts the label by doing just 2 feature-value comparisons! Way faster!

Remember: Root node contains all training inputs
Each leaf node receives a subset of training inputs

Decision Trees for Classification: Another Example

- Deciding whether to play or not to play Tennis on a Saturday
- Each input (Saturday) has 4 categorical features: Outlook, Temp., Humidity, Wind
- A binary classification problem (play vs no-play)
- Below Left: Training data. Below Right: A decision tree constructed for this data

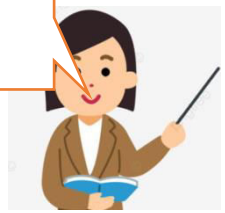
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



Example credit: Tom Mitchell

Decision Trees: Some Considerations

Usually, cross-validation can be used to decide size/shape



- What should be the size/shape of the DT?
  - Number of internal and leaf nodes
  - Branching factor of internal nodes
  - Depth of the tree

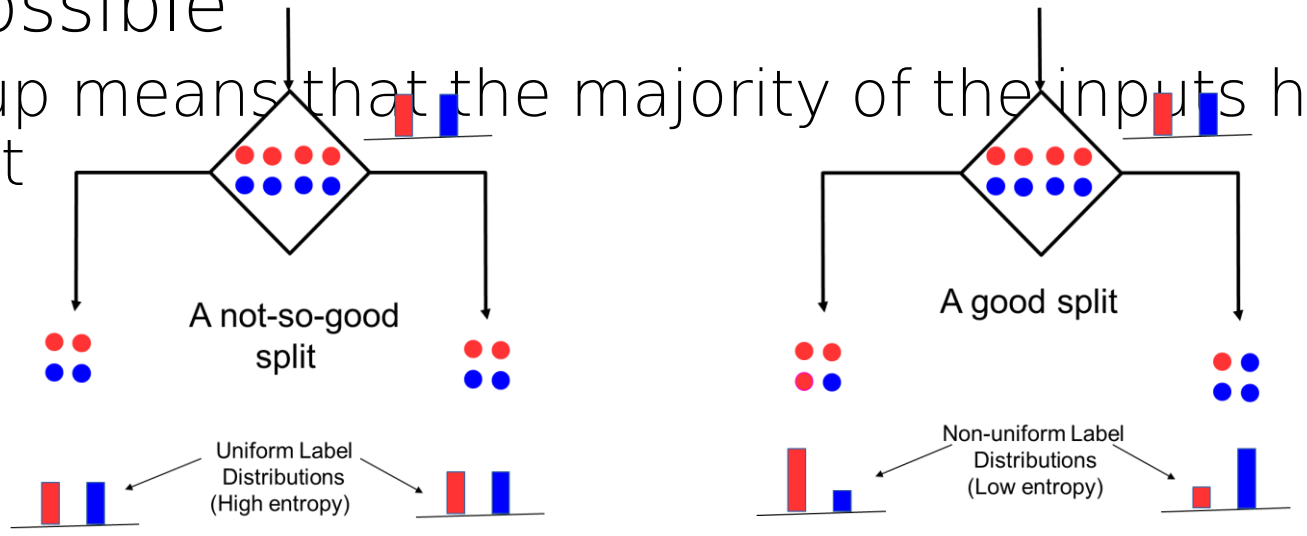
- Split criterion at internal no
  - Use another classifier?
  - Or maybe by doing a simpler test?

- What to do at the leaf node? Some options:
  - Make a constant prediction for each test input re
  - Use a nearest neighbor based prediction using training inputs at tha leaf node
  - Train and predict using some other sophisticated supervised learner on that node

Usually, constant prediction at leaf nodes used since it will be very fast

How to Split at Internal Nodes?

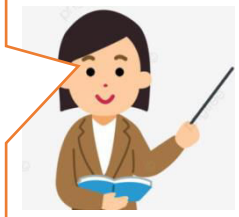
- Recall that each internal node receives a subset of all the training inputs
- Regardless of the criterion, the split should result in as "pure" groups as possible
  - A pure group means that the majority of the inputs have the same label/output



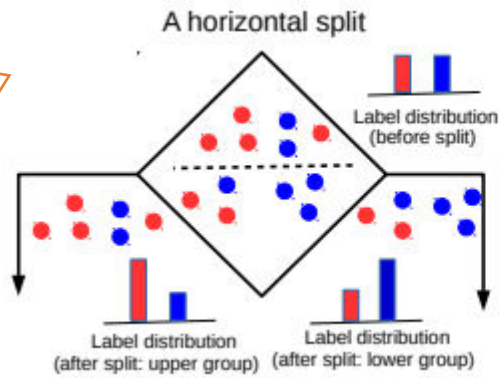
- For classification problems (discrete outputs), entropy is a measure of purity

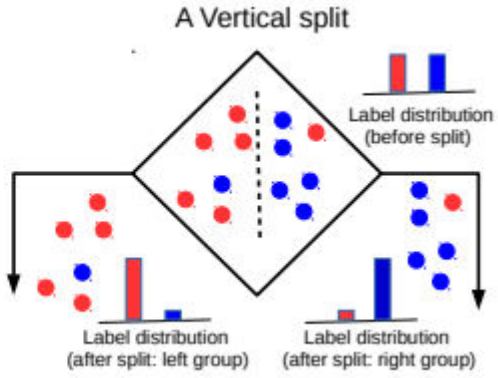
Techniques to Split at Internal Nodes

- Each internal node decides which outgoing branch an input should be sent to
- This decision/split can be done using various ways, e.g.,
  - Testing the value of a single feature at a time (such internal node called DT methods based on testing a single feature at each internal node are faster and more popular (e.g., ID3, C4.5 algos)

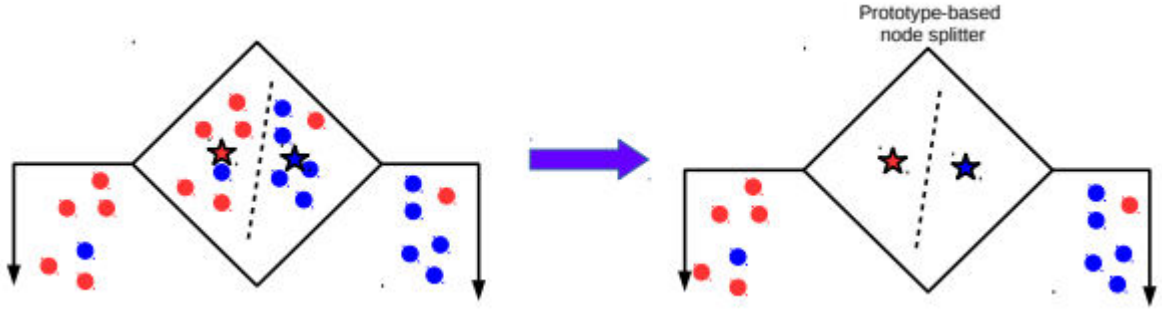


With this approach, all features and all possible values of each feature need to be evaluated in selecting the feature to be tested at each internal node
(can be slow but can be made faster using some tricks)



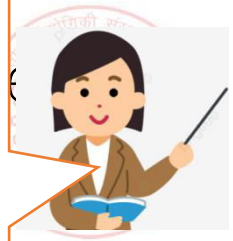


  - Learn

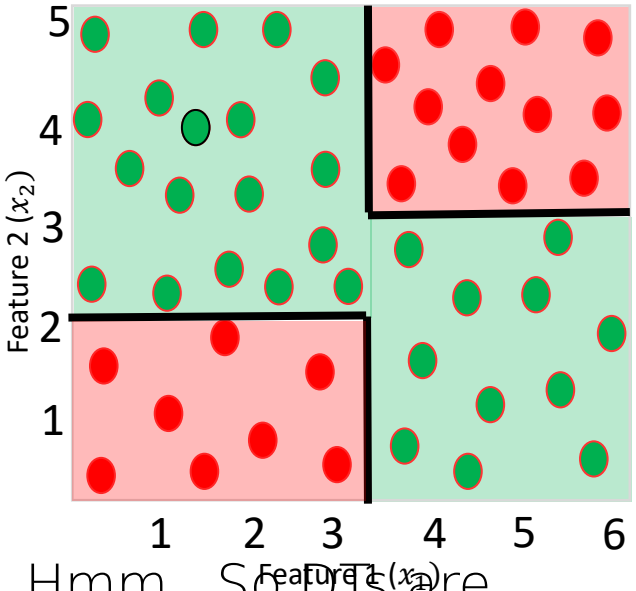


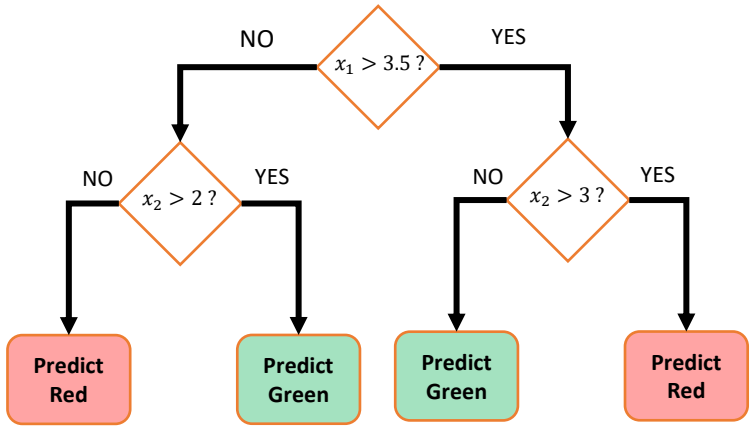
re

DT methods based on learning and using a separate classifier at each internal node are less common. But this approach can be very powerful and are sometimes used in some advanced DT methods



Constructing Decision Trees





Given some training data, what's the "optimal" DT?

How to decide which rules to test for and in what order?

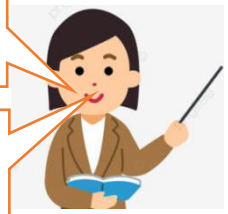
How to assess informativeness of a rule?

Hmm.. So DTs are like the "20 questions" game (ask the most useful questions first)

The rules are organized in the DT such that most informative rules are tested first

Informativeness of a rule is of related to the extent of the purity of the split arising due to that rule. More informative rules yield more pure splits

In general, constructing DT is an intractable problem (NP-hard)



Often we can use some "greedy" heuristics to construct a "good" DT

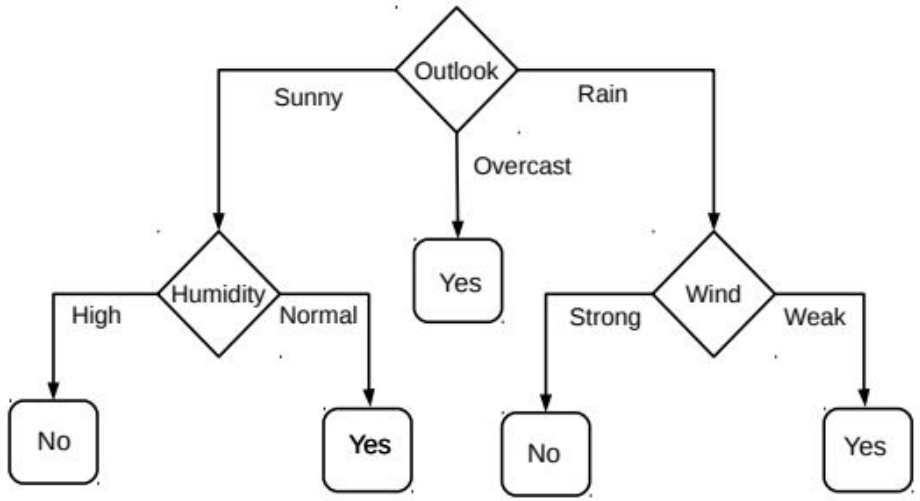
To do so, we use the training data to figure out which rules should be tested at each node

The same rules will be applied on the test inputs to route them along the tree until they reach some leaf node where the prediction is made

Decision Tree Construction: An Example

- Let's consider the playing Tennis example
- Assume each internal node will test the value of one of the feat

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



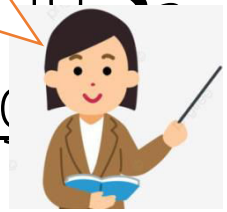
- Question: Why does it make more sense to test the feature "outlook" first?
- Answer: Of all the 4 features, it's the most informative

Entropy and Information Gain

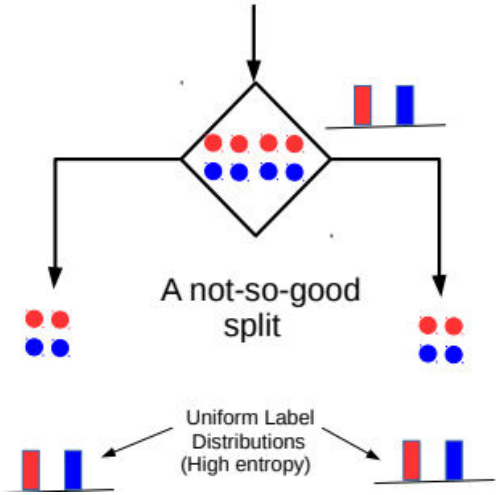
- Assume a set of labelled inputs $\boldsymbol{S}$ from C classes, p_c as fraction of class c inputs
- Entropy of the set $\boldsymbol{S}$ is defined as $\mathbf{H}(\boldsymbol{S}) = -\sum_{c \in C} p_c \log$
- Suppose a rule splits $\boldsymbol{S}$ into two smaller disjoint sets $\boldsymbol{S_1}$
- Reduction in entropy after the split is called information

$$IG = H(S) - \frac{|S_1|}{|S|} H(S_1) - \frac{|S_2|}{|S|} H(S_2)$$

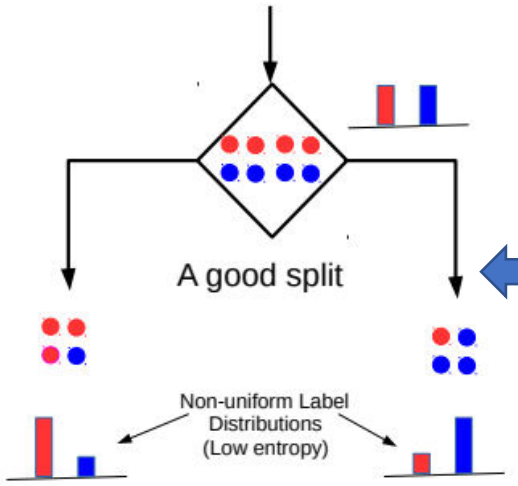
Uniform sets (all classes roughly equally present) have high entropy; skewed sets low



This split has a low IG (in fact zero IG)



VS



This split has higher IG

Entropy and Information Gain

- Let's use IG based criterion to construct a DT for the Tennis example
- At root node, let's compute IG of each of the 4 featur
- Consider feature "wind". Root contains all examples

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

$$H(S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.94$$

$$\boldsymbol{S}_{\text{weak}} = [6+, 2-] \Rightarrow \text{H}(\boldsymbol{S}_{\text{weak}}) = 0.811$$

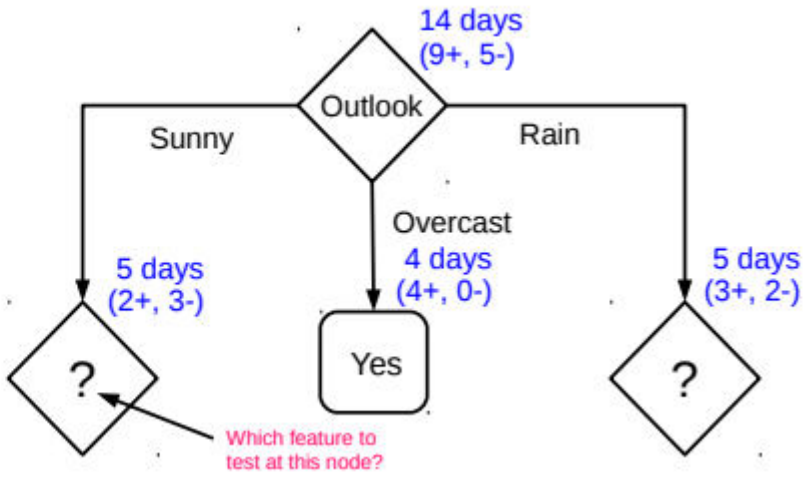
$$\boldsymbol{S}_{\text{strong}} = [3+, 3-] \Rightarrow \text{H}(\boldsymbol{S}_{\text{strong}}) = 1$$

$$IG(S, wind) = H(S) - \frac{|S_{\text{weak}}|}{|S|} H(S_{\text{weak}}) - \frac{|S_{\text{strong}}|}{|S|} H(S_{\text{strong}}) = 0.94 - 8/14 * 0.811 - 6/14 * 1 = 0.048$$

- Likewise, at root: IG(S, outlook) = 0.246, IG(S, humidity) = 0.151, IG(S,temp) = 0.029
- Thus we choose "outlook" feature to be tested at the root node
- Now how to grow the DT, i.e., what to do at the next level? Which feature to test next?

Growing the tree

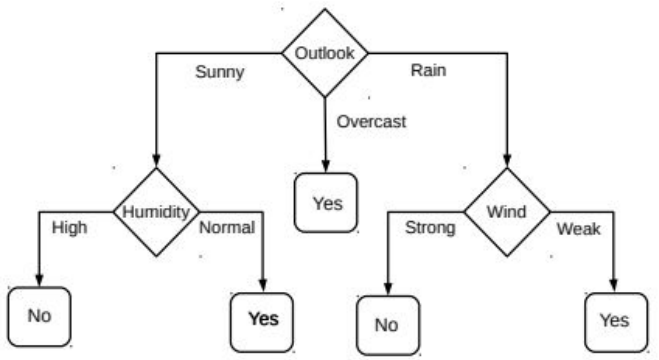
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Proceeding as before, for level 2, left node, we can verify that
  - IG(S,temp) = 0.570, IG(S, humidity) = 0.970, IG(S, wind) = 0.019
- Thus humidity chosen as the feature to be tested at level 2, left node
- No need to expand the middle node (already "pure" - all "yes" training examples)
- Can also verify that wind has the largest IG for the right node
- Note: If a feature has already been tested along a path earlier, we don't

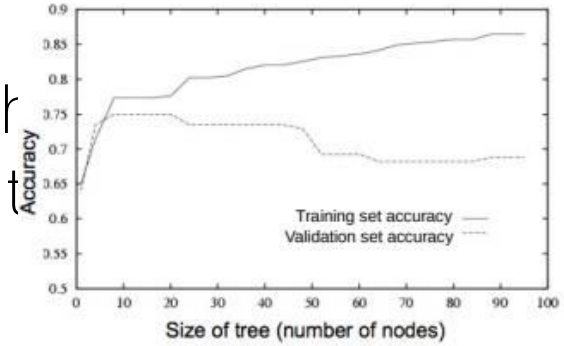
When to stop growing the tree?

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

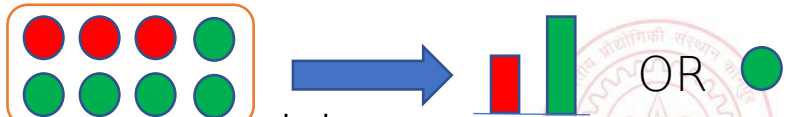


- Stop expanding a node further (i.e., make it a leaf node) when
  - It consist of all training examples having the same label (the node becomes "pure")
  - We run out of features to test along the path to th
  - The DT starts to overfit (car onit the validation set accuracy, too much!

To help prevent the tree from growing



- Important: No need to obsess too much for purity
  - It is okay to have a leaf node that is not fully pure, e.g., this
  - At test inputs that reach an impure leaf, can predict probability of belonging to each class (in above example, p(red) = 3/8, p(green) =



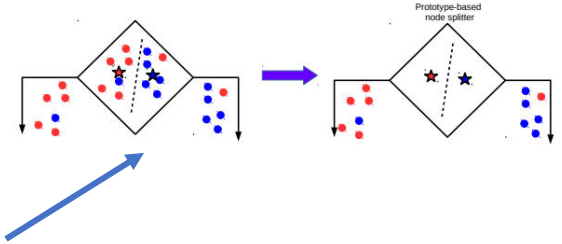
Avoiding Overfitting in DTs

- Desired: a DT that is not too big in size, yet fits the training data reasonably
- Note: An example of a very simple DT is "decision-stump"
  - A decision-stump only tests the value of a single feature (or a simple rule)
  - Not very powerful in itself but often used in large ensembles of decision stumps
- Mainly two approaches to prune a complex DT
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)
- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn't hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the

Either can be done using a validation set

Decision Trees: Some Comments

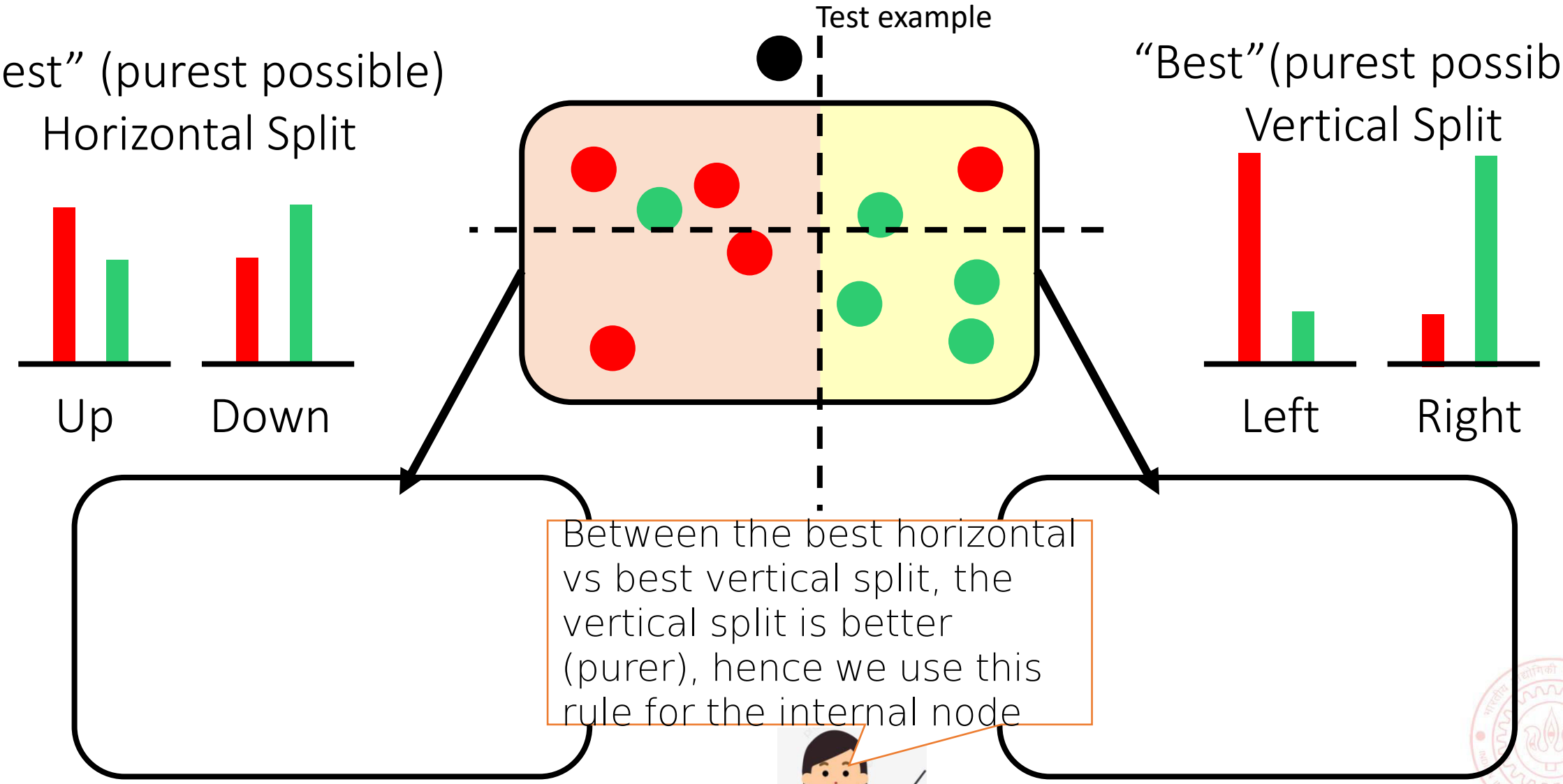
- Gini-index defined as $\sum_{c=1}^{C} p_c(1-p_c)$ can be an alternative to Entropy
- For DT regression[1], variance in the outputs can be used to assess impurity
- When features are real-valued (no finite possible values to try), things are a bit more tricky
  - Can use tests based on thresholding feature values (recall our synthetic data examples)
  - Need to be careful w.r.t. number of threshold points, ho range is, etc.

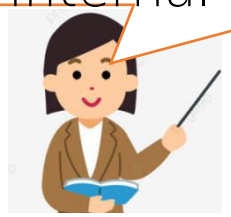


- More sophisticated decision rules at the internal nodes can also be used
  - Basically, need some rule that splits inputs at an internal node into homogeneous groups
  - The rule can even be a machine learning classification algo (e.g., LwP or a deep learner)

[1]Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees

An Illustration: DT with Real-Valued Features



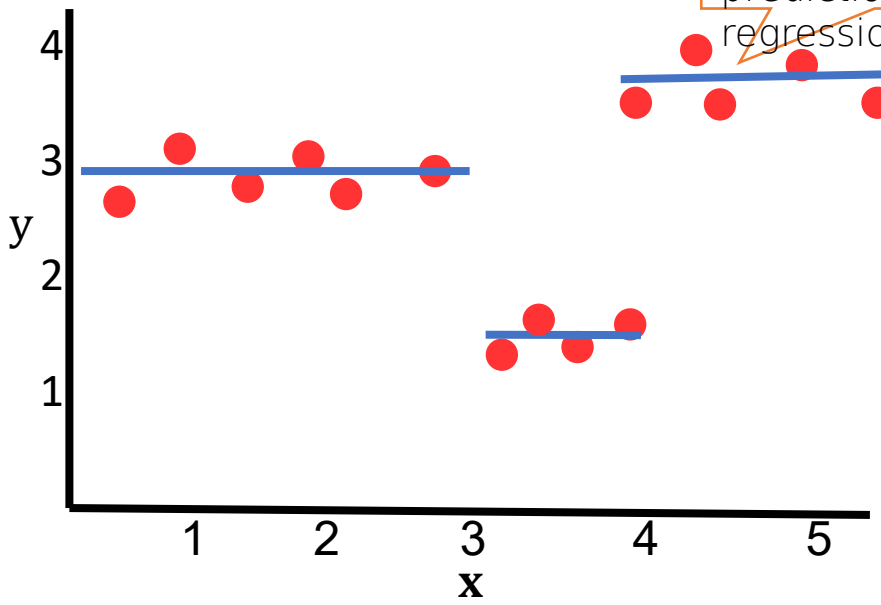


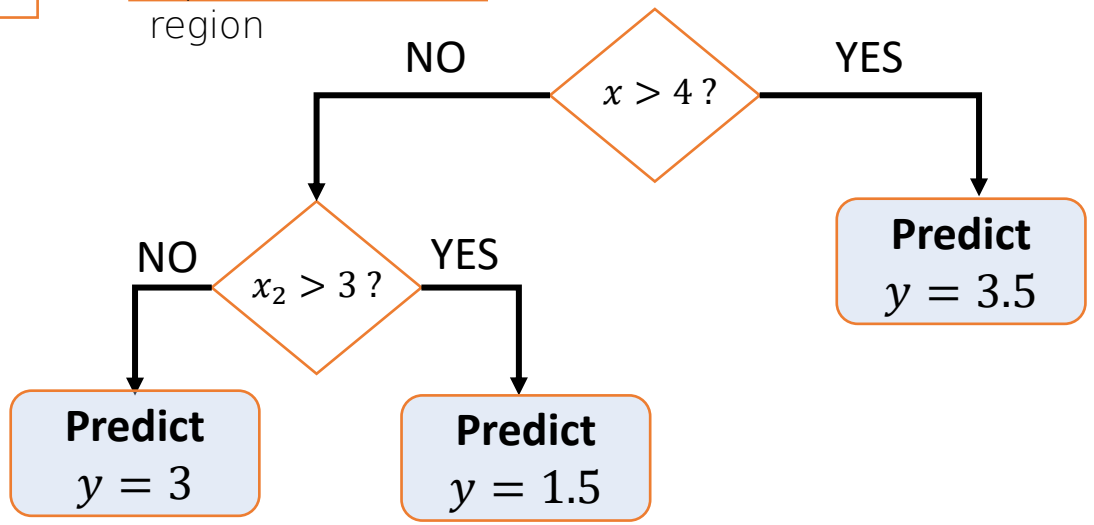
This illustration's credit: Purushottam Kar

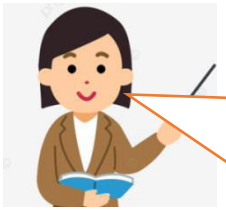
Decision Trees for Regression

Can use any regression model but would like a simple one, so let's use a constant prediction based regression model

Another simple option can be to predict the average output of the training inputs in this region







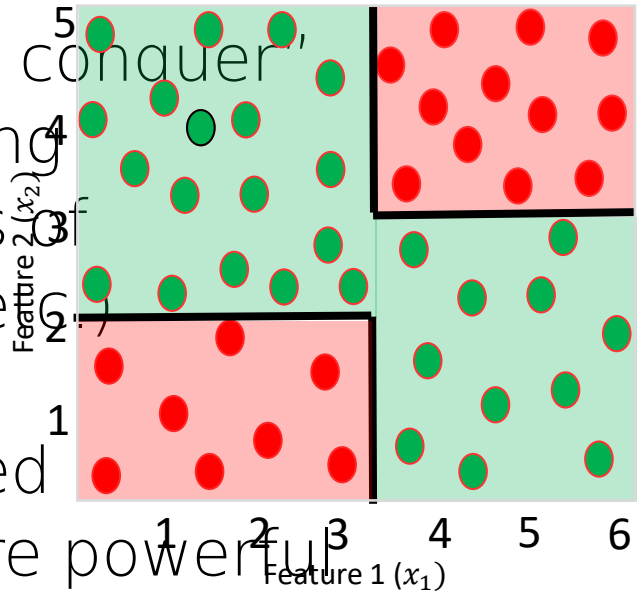
To predict the output for a test point, nearest neighbors will require computing distances from 15 training inputs. DT predicts the label by doing just at most feature-value comparisons! Way faster!

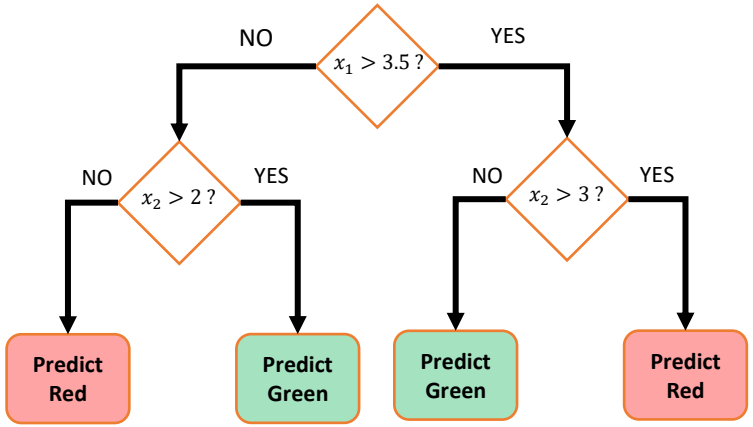
Decision Trees: A Summary

.. thus helping us learn complex rule as a combination of several simpler rules

Some key strengths:

- Simple and easy to interpret
- Nice example of "divide and conquer" paradigm in machine learning
- Easily handle different types of features (real, categorical, etc.)
- Very fast at test time
- Multiple DTs can be combined via ensemble methods: more powerful (e.g., Decision Forests; will see later)
- Used in several real-world ML applications, e.g., recommender systems, gaming (Kinect)





Human-body pose estimation

Some key weaknesses:

- Learning optimal DT is (NP-hard) intractable. Existing algos mostly greedy heuristics

Linear Regression

CS771: Introduction to Machine Learning

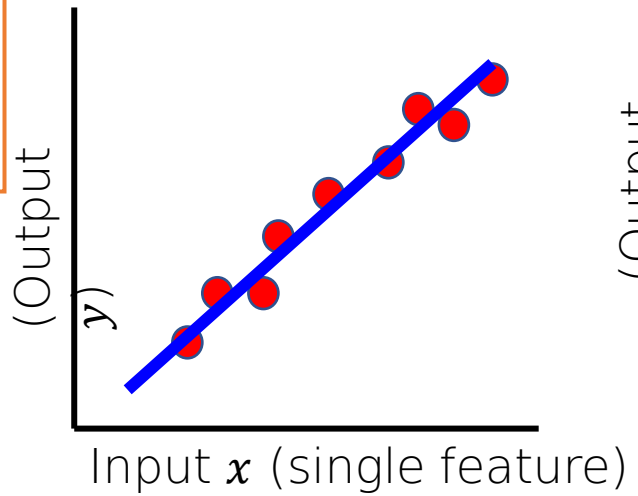
Nisheeth

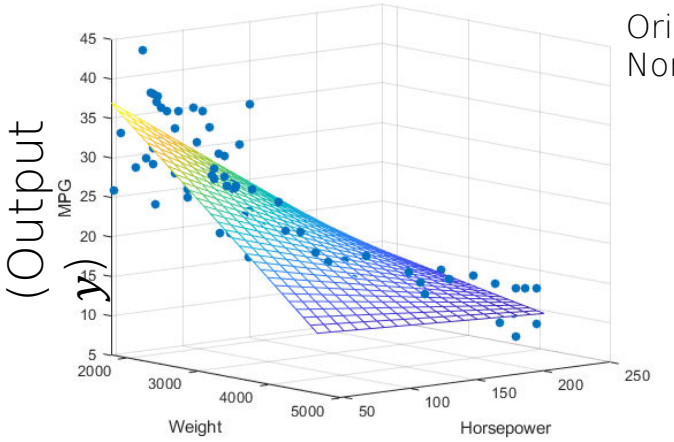
Linear Regression: Pictorially

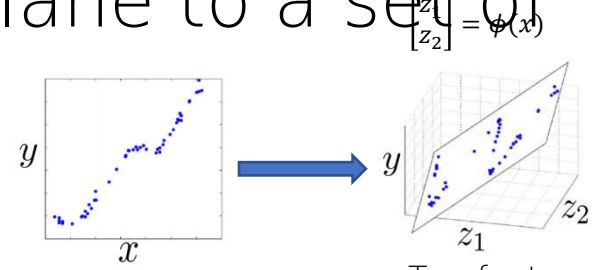
- Linear regression is like fitting a line or (hyper)plane to a set of points

What if a line/plane doesn't model the input-output relationship very well, e.g., if their relationship is better modeled by a nonlinear curve or curved surface?

Do linear models become useless in such cases?







Original (single) feature
Nonlinear curve needed

Two features
Can fit a plane (linea

No. We can even fit a curve using a linear model after suitably transforming the inputs

$\phi(x)$

The transformation $\phi(.)$ can be predefined or learned (e.g., using kernel methods or a deep neural network based feature extractor). More on this later

- The line/plane must also predict outputs the unseen (test)

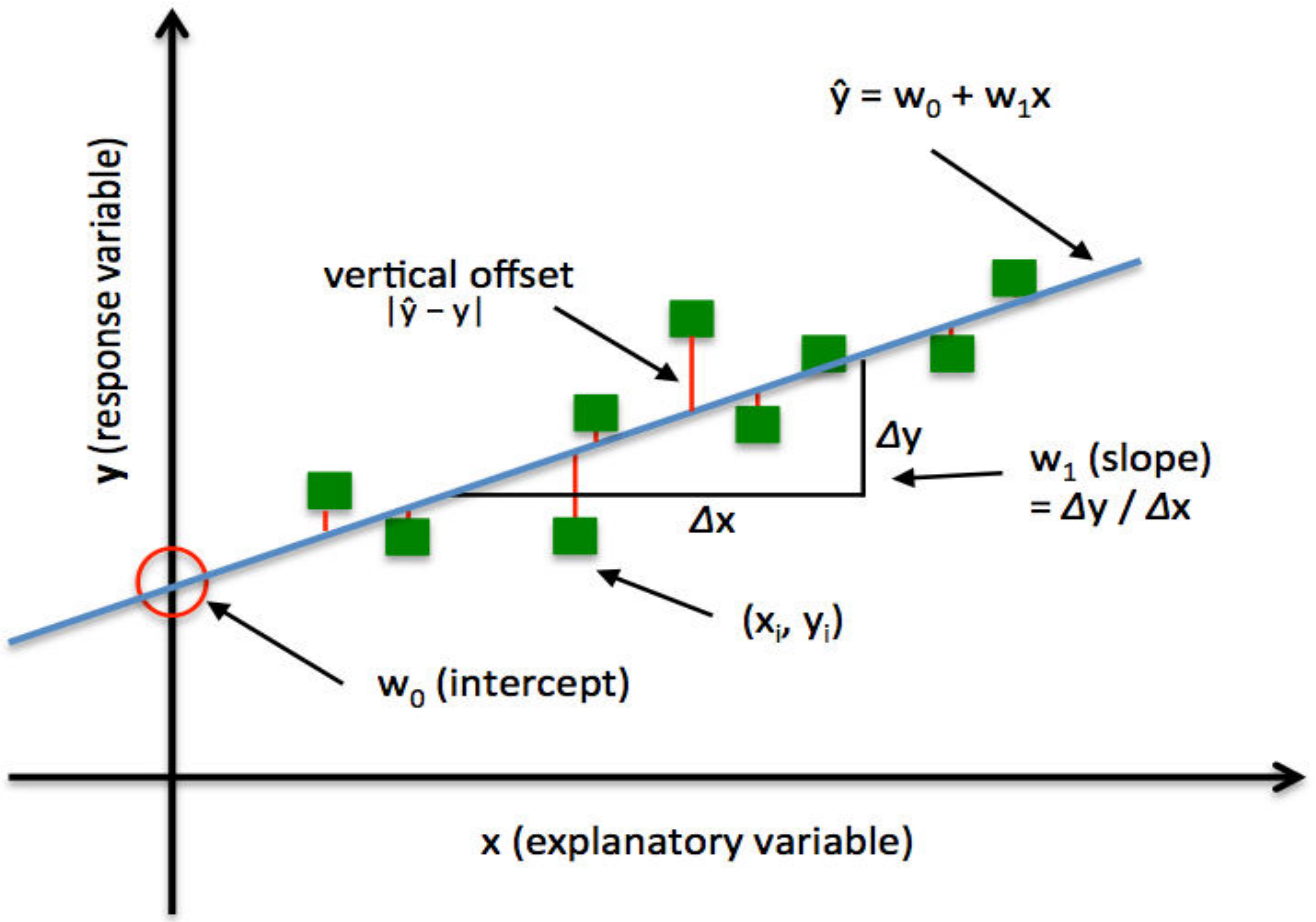
Simplest Possible Linear Regression Model

- This is the base model for all statistical machine learning
- x is a one feature data variable
- y is the value we are trying to predict
- The regression model is

$$y = w_0 + w_1 x + \varepsilon$$

Two parameters to estimate – the slope of the line w_1 and the y-intercept w_0

- ε is the unexplained, random, or error component.

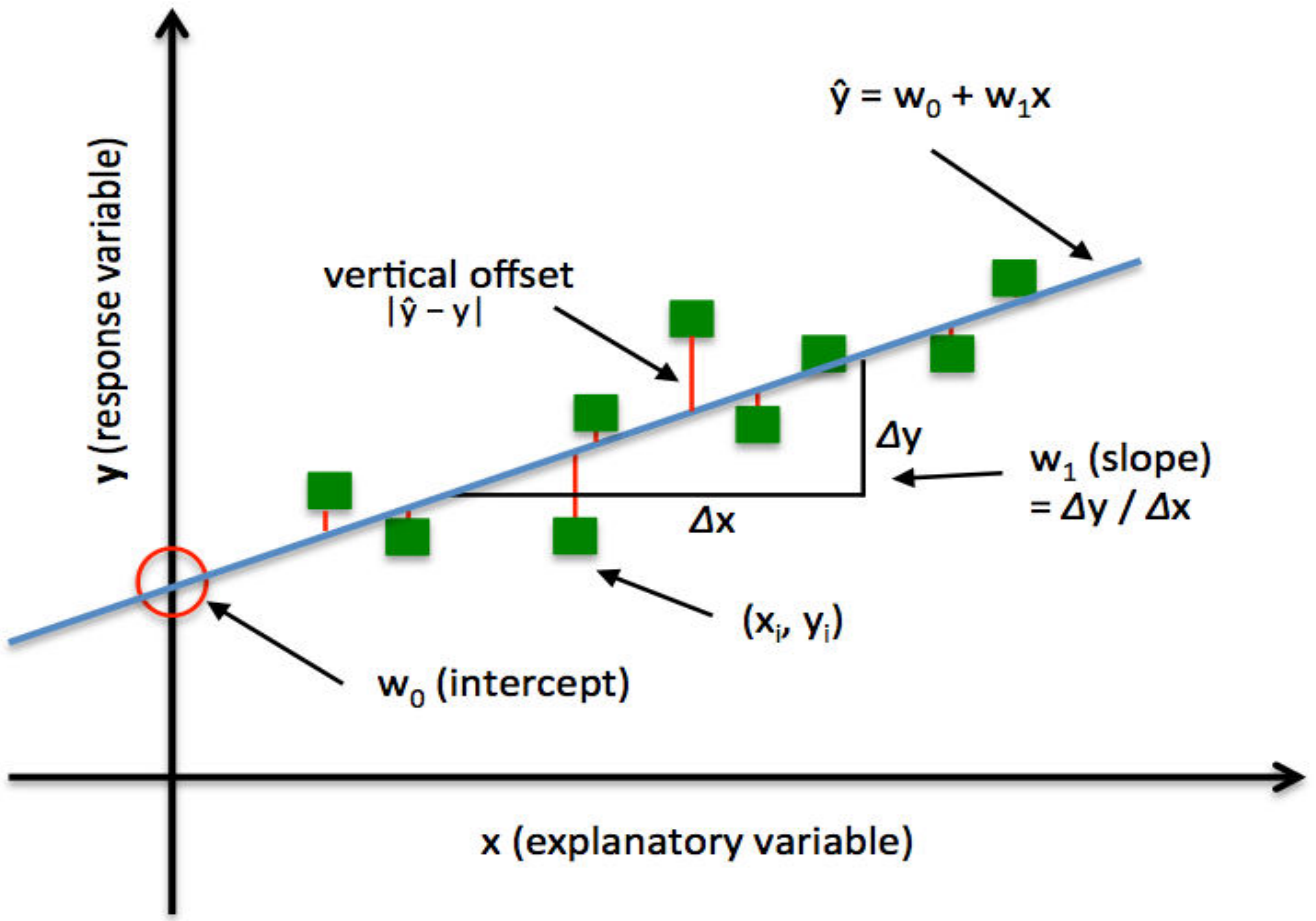


Solving the regression problem

- We basically want to find {w0, w1} that minimize deviations from the predictor line

$$\arg\min_{w_0,w_1} \sum_{i}^{n} (y_i - w_0 - w_1 x_i)^2$$

- How do we do it?
  - Iterate over all possible w values along the two dimensions?
  - Same, but smarter? [next class]
  - No, we can do this in *closed form* with just plain calculus
- Very few optimization problems in ML have closed form solutions
  - The ones that do are interesting for that reason



Parameter estimation via calculus

- We just need to set the partial derivatives to zero (full derivation)

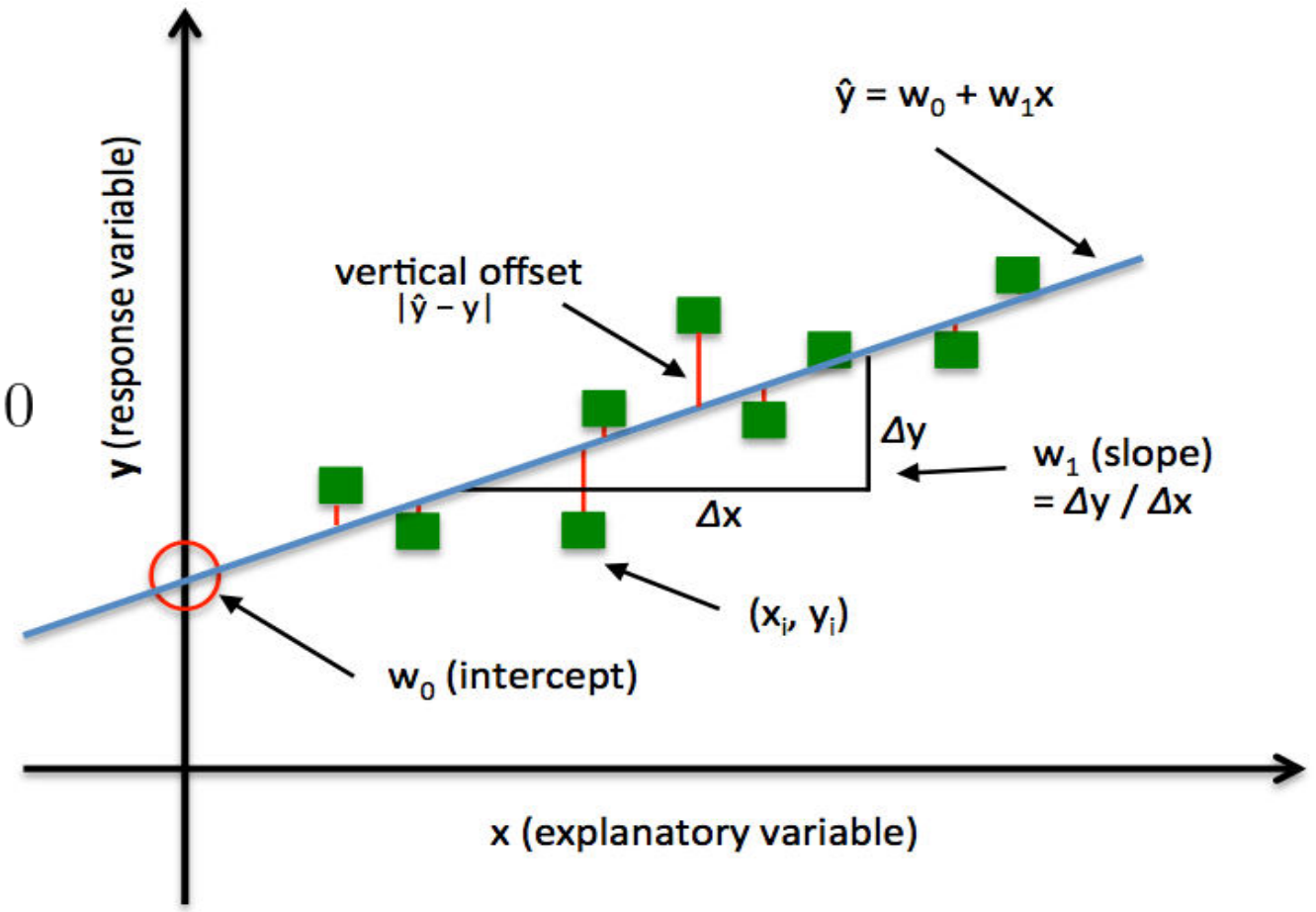
$$\frac{\partial \epsilon^2}{\partial w_0} = \sum_i^n -2(y_i - w_0 - w_1 x_i) = 0$$

$$\frac{\partial \epsilon^2}{\partial w_1} = \sum_i^n -2x_i(y_i - w_0 - w_1 x_i) = 0$$

- Simplifying

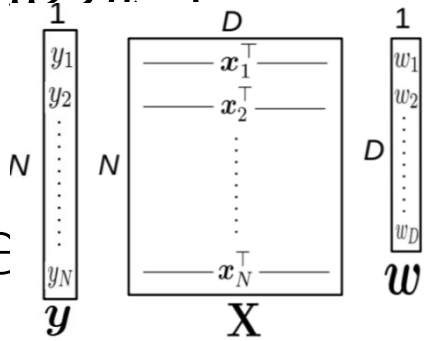
$$w_0 = \bar{y} - w_1 \bar{x}$$

$$w_1 = \frac{n\sum_i^n x_i y_i - \sum_i^n x_i \sum_i^n y_i}{n\sum_i^n x_i x_i - \sum_i^n x_i \sum_i^n x_i}$$



More generally

- Given: Training data with N input-output pairs $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$



- Goal: Learn a model to predict the output for new te

- Assume the function that approximates the I
be a linear model $y_n \approx f(\boldsymbol{x}_n) = \boldsymbol{w}^\top \boldsymbol{x}_n \quad (n = 1,2,\ldots,N)$

Can also write all of them compactly using matrix-vector notation as $\boldsymbol{y} \approx \boldsymbol{X}\boldsymbol{w}$

Goal of learning is to find the $\boldsymbol{w}$ that minimizes this loss + does well on test data

Total error or loss of this m $L(\boldsymbol{w}) = \sum_{n=1}^N \ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n)$

Unlike models like KNN and DT, here we have an explicit problem-specific objective (loss function) that we wish to optimize for

$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n)$ measures the prediction error or "loss" or "deviation" of the model on a single training input $(\boldsymbol{x}_n, y_n)$

Linear Regression with Squared Loss

- In this case, the loss func will b

In matrix-vector notation, can write it compactly as
$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$

$$L(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- Let us find the $\boldsymbol{w}$ that optimizes (minimizes) the above squared loss

The "least squares" (LS) problem Gauss-Legendre, 18th century)

- We need calculus and optimization to do this!
- The LS problem can be solved easily and has a closed solution

$$\boldsymbol{w}_{LS} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

Link to a nice derivation

$$\boldsymbol{w}_{LS} = \left(\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top\right)^{-1}\left(\sum_{n=1}^{N} y_n \boldsymbol{x}_n\right) = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$$

$D \times D$ matrix inversion – can be expensive. Ways to handle this. Will see later

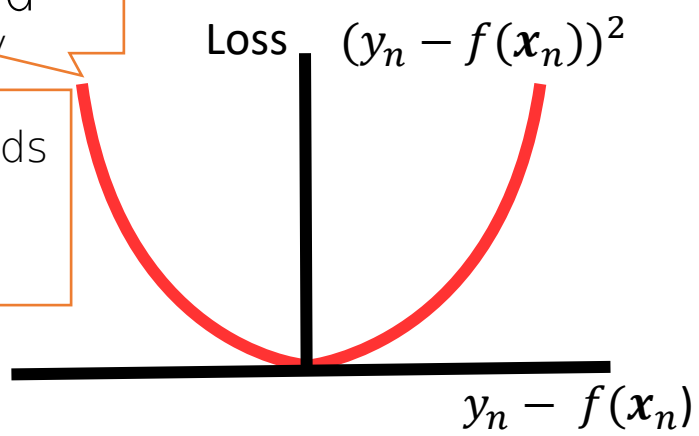
Alternative loss functions

- Many possible loss functions for regression

Choice of loss function usually depends on the nature of the data. Also, some loss functions result in easier optimization problem than others

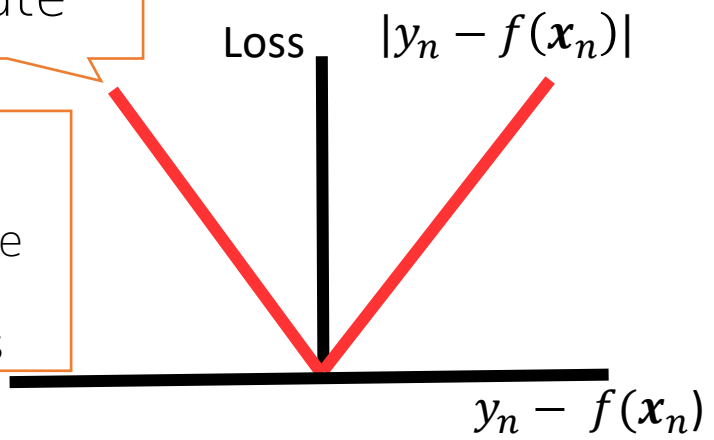
Squared loss

Very commonly used for regression. Leads to an easy-to-solve optimization problem



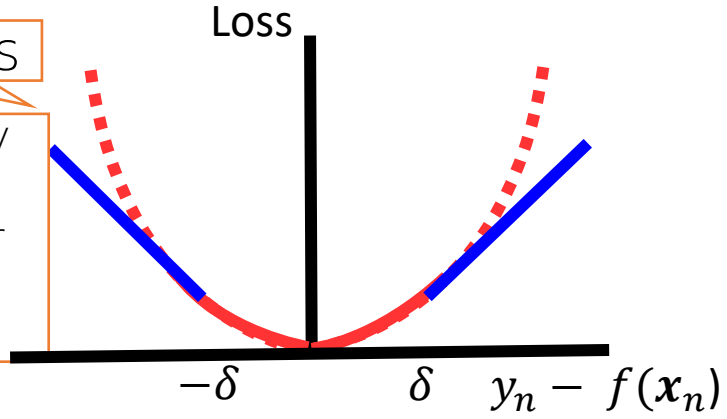
Absolute loss

Grows more slowly than squared loss. Thus better suited when data has some outliers (inputs on which model makes large errors)



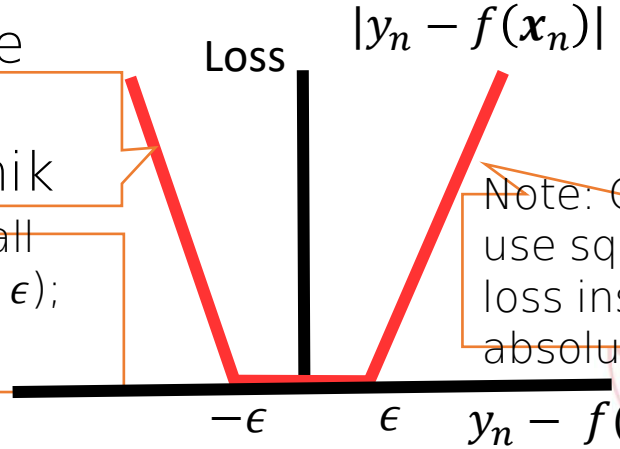
Huber loss

Squared loss for small errors (say up to δ); absolute loss for larger errors. Good for data with outliers



ϵ-insensitive loss (a.k.a. Vapnik loss)

Zero loss for small errors (say up to ϵ); absolute loss for larger errors



Note: Can also use squared loss instead of absolute loss

How do we ensure generalization?

- We minimized the objective $L(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$ w.r.t. $\boldsymbol{w}$ and got

$$\boldsymbol{w}_{LS} = (\sum_{n=1}^{N} \boldsymbol{x}_n \, \boldsymbol{x}_n^\top)^{-1} (\sum_{n=1}^{N} y_n \boldsymbol{x}_n) \;\; = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \, \boldsymbol{X}^\top \boldsymbol{y}$$

- Problem: The matrix $\boldsymbol{X}^\top \boldsymbol{X}$ may not be invertible
  - This may lead to non-unique solutions for $\boldsymbol{w}_{opt}$
- Problem: Overfitting since we only minimized loss defined on training data
  - Weights $\boldsymbol{w} = [w_1, w_2, \ldots, w_D]$ may become arbitrarily large to fit the training data perfectly
  - Such weights may perform poorly on the test data however
- One Solution: Minimize a **regularized objective** $L(\boldsymbol{w})$

$R(\boldsymbol{w})$ is called the Regularizer and measures the "magnitude" of $\boldsymbol{w}$

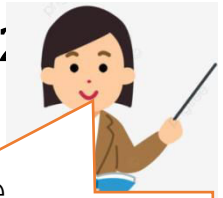
$\lambda \geq 0$ is the reg. hyperparam. Controls how much we wish to regularize (needs to be tuned via cross-validation)

Regularized Least Squares (a.k.a. Ridge Regression)

- Recall that the regularized objective is of the form $L_{reg}(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda\, R(\boldsymbol{w})$

- One possible/popular regularizer: the squared Euclidean (ℓ_2 squared) norm of $\boldsymbol{w}$ $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2 = \boldsymbol{w}^\top \boldsymbol{w}$

Why is the method called "ridge" regression



Look at the form of the solution. We are adding a small value λ to the diagonals of the DxD matrix $\boldsymbol{X}^\top\boldsymbol{X}$(like adding a ridge/mountain to some land)

- With this regularizer, we have the regularized

$$\boldsymbol{w}_{ridge} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) + \lambda\, R(\boldsymbol{w})$$

$$= \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

- Proceeding just like the LS case, we can find the optimal $\boldsymbol{w}$ which is given by

$$\boldsymbol{w}_{ridge} = \left(\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top + \lambda I_D\right)^{-1} \left(\sum_{n=1}^{N} y_n \boldsymbol{x}_n\right) = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda I_D)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

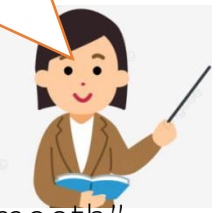
A closer look at ℓ_2 regularization

- The regularized objective we minimized is

$$L_{reg}(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

- Minimizing $L_{reg}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ gives a solution for $\boldsymbol{w}$ that
  - Keeps the training error small
  - Has a small ℓ_2 squared norm $\boldsymbol{w}^\top \boldsymbol{w} = \sum_{d}^{D}$
- Small entries in $\boldsymbol{w}$ are good since they lead to "smooth

Remember – in general, weights with large magnitude are bad since they can cause overfitting on training data and may not work well on test data



Good because, consequently, the individual entries of the weight vector $\boldsymbol{w}$ are also prevented from becoming too large

Not a "smooth" model since its test data predictions may change drastically even with small changes in some feature's value

A typical $\boldsymbol{w}$ learned without ℓ_2 reg.

3.2	1.8	1.3	2.1	10000	2.5	3.1	0.1

Just to fit the training data where one of the inputs was possibly an outlier, this weight became too big. Such a weight vector will possibly do poorly on normal test inputs

$\boldsymbol{x}_n =$

1.2	0.5	2.4	0.3	0.8	0.1	0.9	2.1

$\boldsymbol{x}_m =$

1.2	0.5	2.4	0.3	$0.8 + \epsilon$	0.1	0.9	2.1

Exact same feature vectors only differing in just one feature by a small amount

$y_n = 0.8$

$y_m = 100$

Very different outputs though (maybe one of these two training ex. is an outlier)

Other Ways to Control Overfitting

- Use a regularizer $R(\boldsymbol{w})$ defined by other norms, e.g.,

ℓ_1 norm regularizer

$$\|\boldsymbol{w}\|_1 = \sum_{d=1}^{D} |w_d|$$

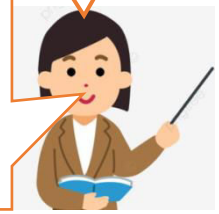
When should I used these regularizers instead of the ℓ_2 regularizer?

$$\|\boldsymbol{w}\|_0 = \#\text{nnz}(\boldsymbol{w})$$

ℓ_0 norm regularizer (counts number of nonzeros in $\boldsymbol{w}$)

Note that optimizing loss functions with such regularizers is usually harder than ridge reg. but several advanced techniques exist (we will see some of those later)

Use them if you have a very large number of features but many irrelevant features. These regularizers can help in automatic feature selection



Using such regularizers gives a sparse weight vector $\boldsymbol{w}$ as solution

sparse means many entries in $\boldsymbol{w}$ will be zero or near zero. Thus those features will be considered irrelevant by the model and will not influence prediction

- Use non-regularization based approaches
  - Early-stopping (stopping training just when we have a decent val. set accuracy)
  - Dropout (in each iteration, don't update some of the
  - Injecting noise in the inputs

All of these are very popular ways to control overfitting in deep learning models. More on these later when we talk about deep learning

Linear Regression as Solving System of Linear Eqs

- The form of the lin. reg. model $\boldsymbol{y} \approx \boldsymbol{Xw}$ is akin to a system of linear equation
- Assuming N training examples with D features each, we have

First training example: $y_1 = x_{11}w_1 + x_{12}w_2 + \ldots + x_{1D}w_D$

Second training example: $y_2 = x_{21}w_1 + x_{22}w_2 + \ldots + x_{2D}w_D$

$\vdots$

N-th training example: $y_N = x_{N1}w_1 + x_{N2}w_2 + \ldots + x_{ND}w_D$

Note, here x_{nd} denotes the d^{th} feature of the n^{th} training example

N equations and D unknowns here $(w_1, w_2, \ldots, w_D)$

- However, in regression, we rarely have $N = D$ but rather $N > D$ or $N < D$
  - Thus we have an underdetermined ($N < D$) or overdetermined ($N > D$) system
  - Methods to solve (over/under-determined systems can be used for lin. reg. as well
  - Many of these methods don't require expensive matrix inversion

Solving lin. reg. as system of lin. eq.

$\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$ → $\boldsymbol{Aw} = \boldsymbol{b}$ where $\boldsymbol{A} = \boldsymbol{X}^\top \boldsymbol{X}$, and $\boldsymbol{b} =$

Now solve this!

System of lin. Eqns with D equations and D unknowns

Next Lecture

- Solving linear regression using iterative optimization methods
  - Faster and don't require matrix inversion
- Brief intro to optimization techniques

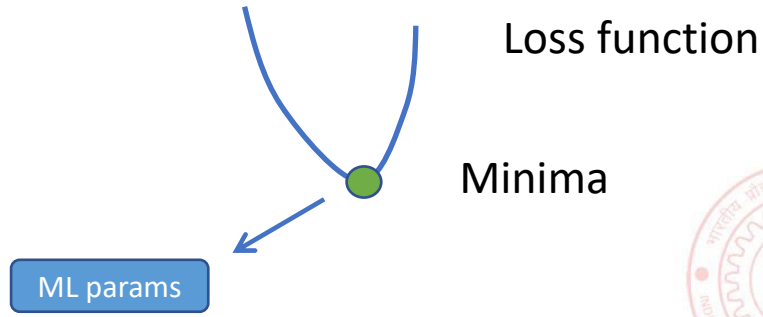
Optimization for ML

CS771: Introduction to Machine Learning

Nisheeth

Today's class

- In the last class, we saw that parameter estimation for the linear regression model is possible in closed form
- This is not always the case for all ML models. What do we do in those cases?
- We treat the parameter estimation problem as a problem of function optimization
- There is lots of math, but it's very intuitive
- Don't be intimidated



Nice reference for today's material.

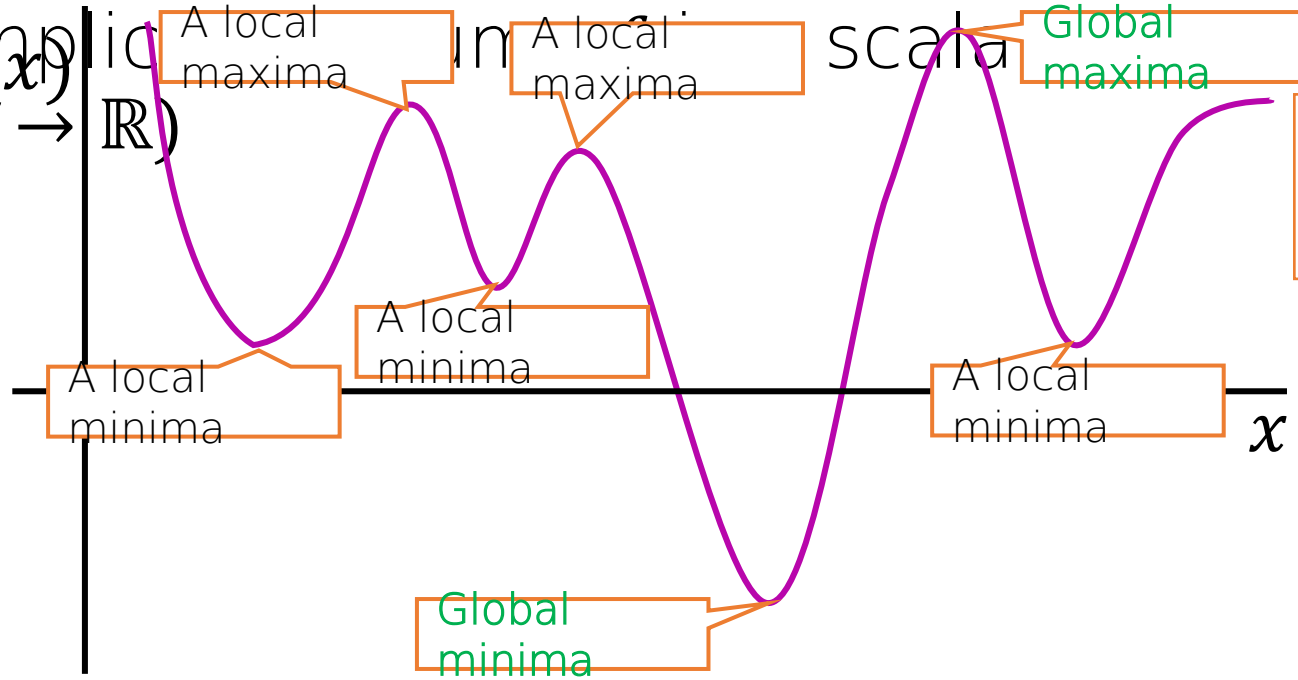
For those of you interested in a deeper dive in the math, see Ch 3 in this book

Functions and their optima

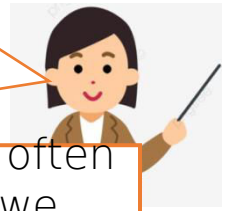
The objective function of the ML problem we are solving (e.g., squared loss for regression)

Assume unconstrained for now, i.e., just a real-valued number/vector

- Many ML problems require us to optimize a function f of some variable(s) x
- For simplicity, let's assume f is a scalar function of a scalar x ($f: \mathbb{R} \rightarrow \mathbb{R}$)



Usually interested in global optima but often want to find local optima, too



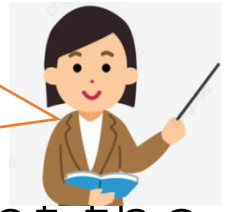
For deep learning models, often the local optima are what we can find (and they usually suffice) – more later

Will see what these are later

- Any function has one/more optima (maxima, minima), and maybe saddle points

Derivatives

Will sometimes use $f'(x)$ to denote the derivative

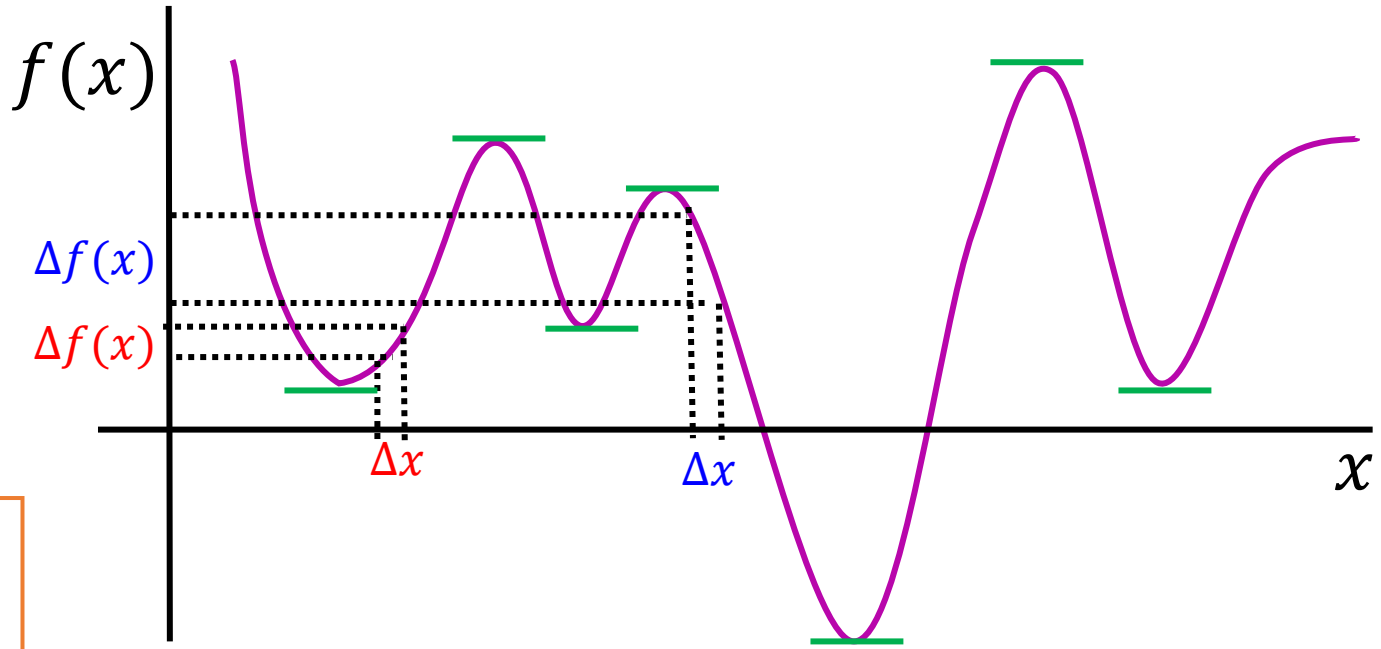


- Magnitude of derivative at a point is the rate of change of the func at that point

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{\Delta f(x)}{\Delta x}$$

Sign is also important: Positive derivative means f is increasing at x if we increase the value of x by a very small amount; negative derivative means it is decreasing

Understanding how f changes its value as we change x is helpful to understand optimization (minimization/maximization) algorithms

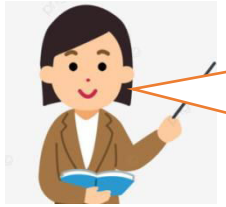


- Derivative becomes zero at stationary points (optima or saddle points)
  - The function becomes "flat" ($\Delta f(x) = 0$ if we change x by a very little at

Rules of Derivatives

Some basic rules of taking derivatives

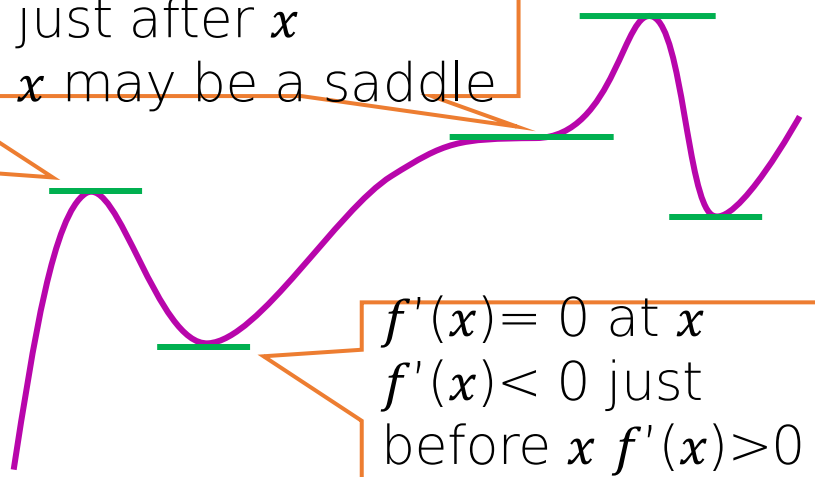
- Sum Rule: $\big(f(x)+g(x)\big)' = f'(x)+g'(x)$
- Scaling Rule: $\big(a \cdot f(x)\big)' = a \cdot f'(x)$ if a is not a function of x
- Product Rule: $\big(f(x) \cdot g(x)\big)' = f'(x) \cdot g(x) + g'(x) \cdot f(x)$
- Quotient Rule: $\big(f(x)/g(x)\big)' = \big(f'(x) \cdot g(x) - g'(x)f(x)\big)/\big(g(x)\big)^2$
- Chain Rule: $\Big(f\big(g(x)\big)\Big)' \stackrel{\text{def}}{=} (f \circ g)'(x) = f'\big(g(x)\big) \cdot g'(x)$



We already used some of these (sum, scaling and chain) when calculating the derivative for the linear regression model

Derivatives

- How the derivative itself changes tells us about the function's optima



$f'(x) = 0$ at x, $f'(x) > 0$ just before x $f'(x) < 0$ just after x
x is a maxima

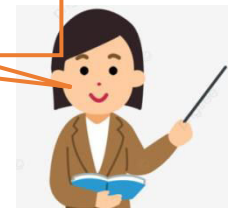
$f'(x) = 0$ at x
$f'(x) = 0$ just before x $f'(x) = 0$ just after x
x may be a saddle

$f'(x) = 0$ at x
$f'(x) < 0$ just before x $f'(x) > 0$ just after x
x is a minima

$f'(x) = 0$ and $f''(x) < 0$
x is a maxima

$f'(x) = 0$ and $f''(x) > 0$
x is a minima

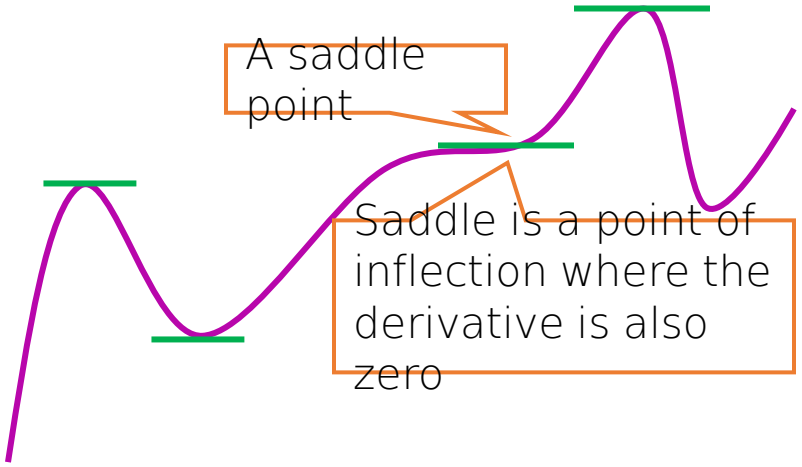
$f'(x) = 0$ and $f''(x) = 0$
x may be a saddle. May need higher derivatives

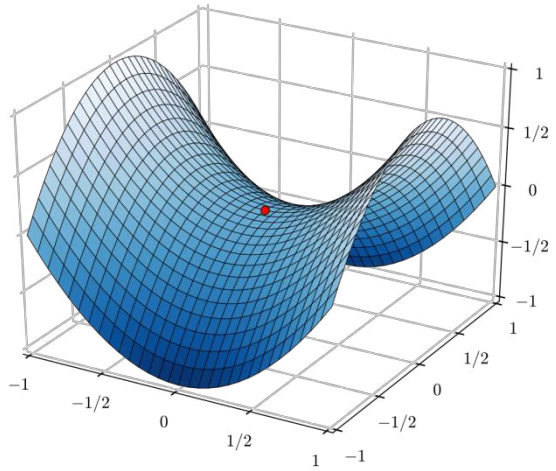


- The second derivative $f''(x)$ can provide this information

Saddle Points

- Points where derivative is zero but are neither minima nor maxima





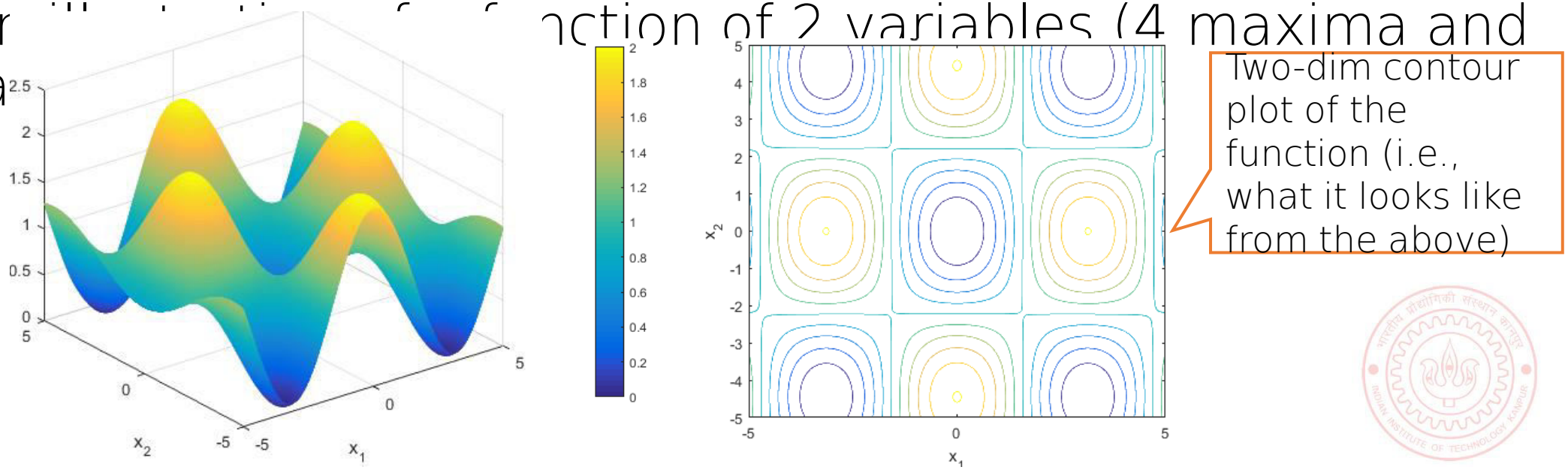
- Saddle points are very common for loss functions of deep learning models
  - Need to be handled carefully during optimization

Multivariate Functions

- Most functions that we see in ML are multivariate function
- Example: Loss fn $\boldsymbol{L(w)}$ in lin-reg was a multivar function of $\boldsymbol{D}$-dim vector $\boldsymbol{w}$

$$L(\boldsymbol{w}): \mathbb{R}^D \rightarrow \mathbb{R}$$

- Here is an illustration of a function of 2 variables (4 maxima and 5 minima)



Two-dim contour plot of the function (i.e., what it looks like from the above)

Plot courtesy: http://benchmarkfcns.xyz/benchmarkfcns/griewankfcn.html

Derivatives of Multivariate Functions

- Can define derivative for a multivariate functions as well via the gradient

- Gradient of a function $f(\boldsymbol{x}): \mathbb{R}^D \rightarrow \mathbb{R}$ is a $D \times 1$ vector of partial derivatives

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_D}\right)$$

Each element in this gradient vector tells us how much f will change if we move a little along the corresponding (akin to one-dim case)

- Optima and saddle points defined similar to one-dim case
  - Required properties that we saw for one-dim case must be satisfied along all the directions

- The second derivative in this case is known as the Hessian

The Hessian

- For a multivar scalar valued function $f(\boldsymbol{x})$: $\mathbb{R}^D \rightarrow \mathbb{R}$, Hessian is a $D \times D$ matrix

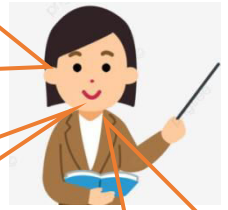
$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_D} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_D x_1} & \frac{\partial^2 f}{\partial x_D x_2} & \cdots & \frac{\partial^2 f}{\partial x_D^2} \end{bmatrix}$$

Gives information about the curvature of the function at point $\boldsymbol{x}$

Note: If the function itself is vector valued, e.g., $f(\boldsymbol{x})$: $\mathbb{R}^D \rightarrow \mathbb{R}^K$ then we will have K such $D \times D$ Hessian matrices, one for each output dimension of f

A square, symmetric $D \times D$ matrix M is PSD if $\boldsymbol{x}^\top M \boldsymbol{x} \geq \boldsymbol{0}$ $\forall\, \boldsymbol{x} \in \mathbb{R}^D$

Will be NSD if $\boldsymbol{x}^\top M \boldsymbol{x} \leq \boldsymbol{0}$ $\forall\, \boldsymbol{x} \in \mathbb{R}^D$

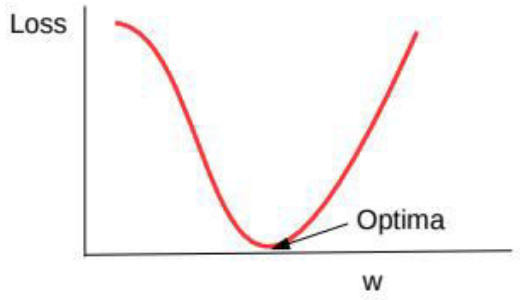


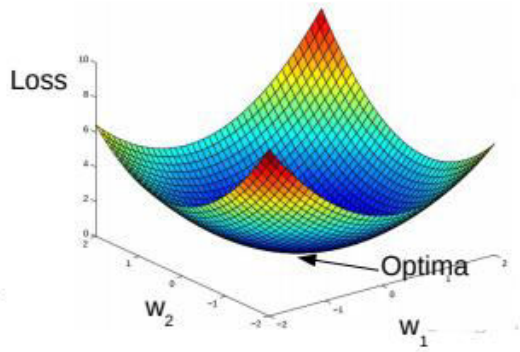
PSD if all eigenvalues are non-negative

- The Hessian matrix can be used to assess the optima/saddle points
  - $\nabla f(\boldsymbol{x}) = 0$ and $\nabla^2 f(\boldsymbol{x})$ is a positive semi-definite (PSD) matrix then $\boldsymbol{x}$ is a minima
  - $\nabla f(\boldsymbol{x}) = 0$ and $\nabla^2 f(\boldsymbol{x})$ is a negative semi-definite (NSD) matrix then $\boldsymbol{x}$ is

Convex and Non-Convex Functions

- A function being optimized can be either convex or non-convex
- Here are a couple of examples of convex functions

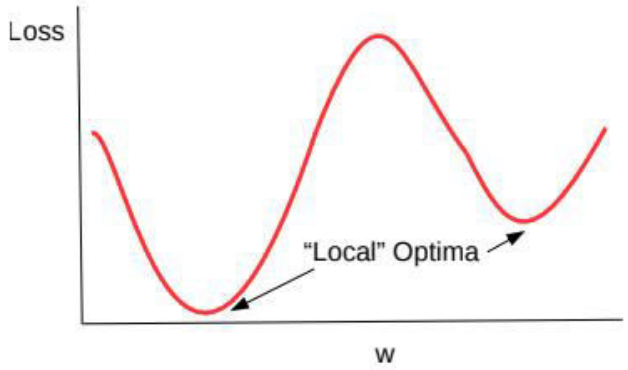


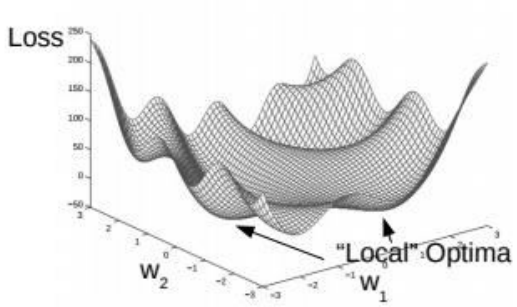


Convex functions are bowl-shaped. They have a unique optima (minima)

Negative of a convex function is called a concave function, which also has a unique optima (maxima)

- Here are a couple of examples of non-convex functions





Non-convex functions have multiple minima. Usually harder to optimize as compared to convex functions

Loss functions of most deep learning models are non-convex

Convex Sets

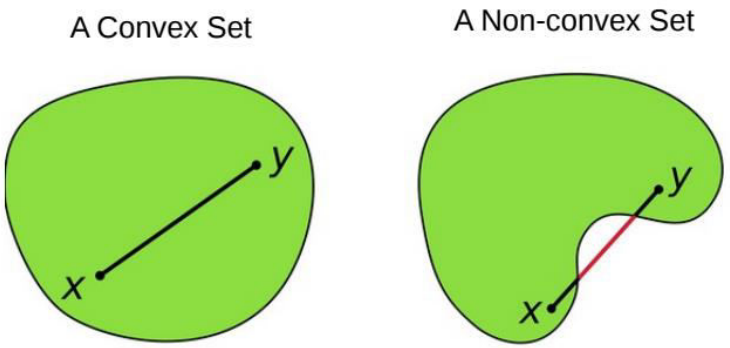
- A set S of points is a convex set, if for any two points $x, y \in S$, and $0 \leq \alpha \leq 1$

z is also called a "convex combination" of two points

$$z = \alpha x + (1 - \alpha) y \in S$$

Can also define convex combination of N points $x_1, x_2, \ldots, x_N$ as $z = \sum_{i=1}^{N} \alpha_i x_i$

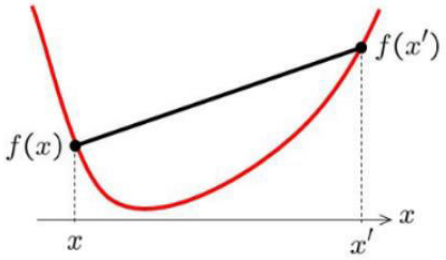
- Above means that all points on the line-segment between x and y lie within S



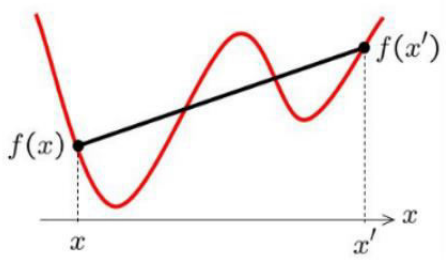
Convex Functions

- Informally, $f(x)$ is convex if all of its chords lie above the function everywhere

Convex Function



Non-convex Function



Note: "Chord lies above function" more formally means

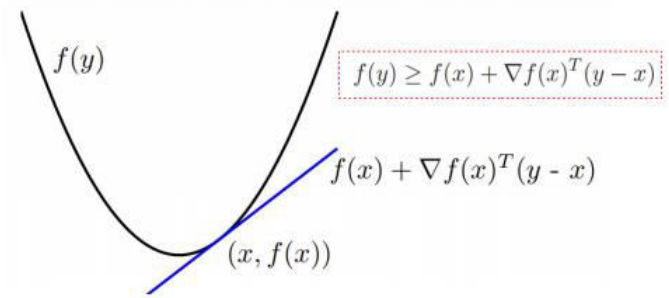
If f is convex then given

$$\alpha_1, \ldots, \alpha_n \text{ s.t } \sum_{i=1}^{n} \alpha_i = 1$$

$$f\left(\sum_{i=1}^{n} \alpha_i x_i\right) \leq \sum_{i=1}^{n} \alpha_i f(x_i)$$

Jensen's Inequality

- Formally, (assuming differentiable function), some tests for convexity:
  - First-order convexity



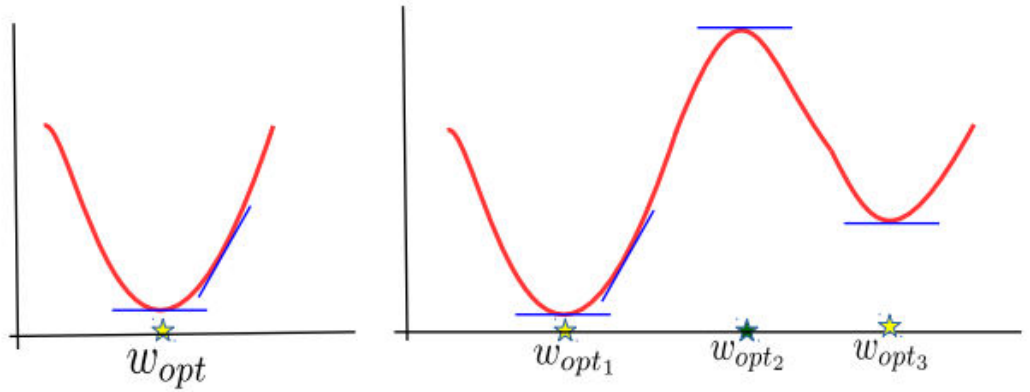
bove all the

nts)

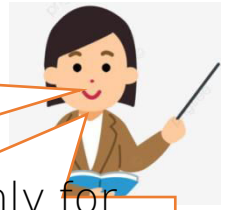
Exercise: Show that ridge regression objective is convex

Optimization Using First-Order Optimality

- Very simple. Already used this approach for linear and ridge regressior



Called "first order" since only gradient is used and gradient provides the first order info about the function being optimized



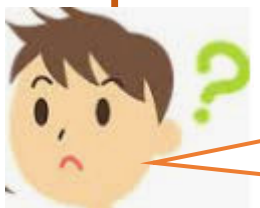
The approach works only for very simple problems where the objective is convex and there are no constraints on the values $\boldsymbol{w}$ can take

- First order optimality: The gradient $\boldsymbol{g}$ must be equal to zero at the optima

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}}[L(\boldsymbol{w})] = \mathbf{0}$$

- Sometimes, setting $\boldsymbol{g} = \mathbf{0}$ and solving for $\boldsymbol{w}$ gives a closed form solution

Optimization via Gradient Descent



Can I used this approach to solve maximization problems?

Fact: Gradient gives the direction of steepest change in function's value

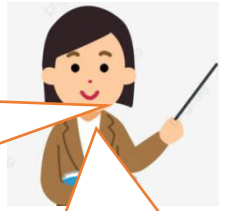
For max. problems we can use gradient ascent

$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \boldsymbol{g}^{(t)}$

Will move in the direction of the gradient

Iterative since it requires several steps/iterations to find the optimal solution

For convex functions, GD will converge to the global minima



Good initialization needed for non-convex

Gradient Descent

- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2, \ldots$ (or until convergence)
  - Calculate the gradient $\boldsymbol{g}^{(t)}$ using the current iterates $\boldsymbol{w}^{(t)}$
  - Set the learning rate η_t
  - Move in the opposite direction of gradient

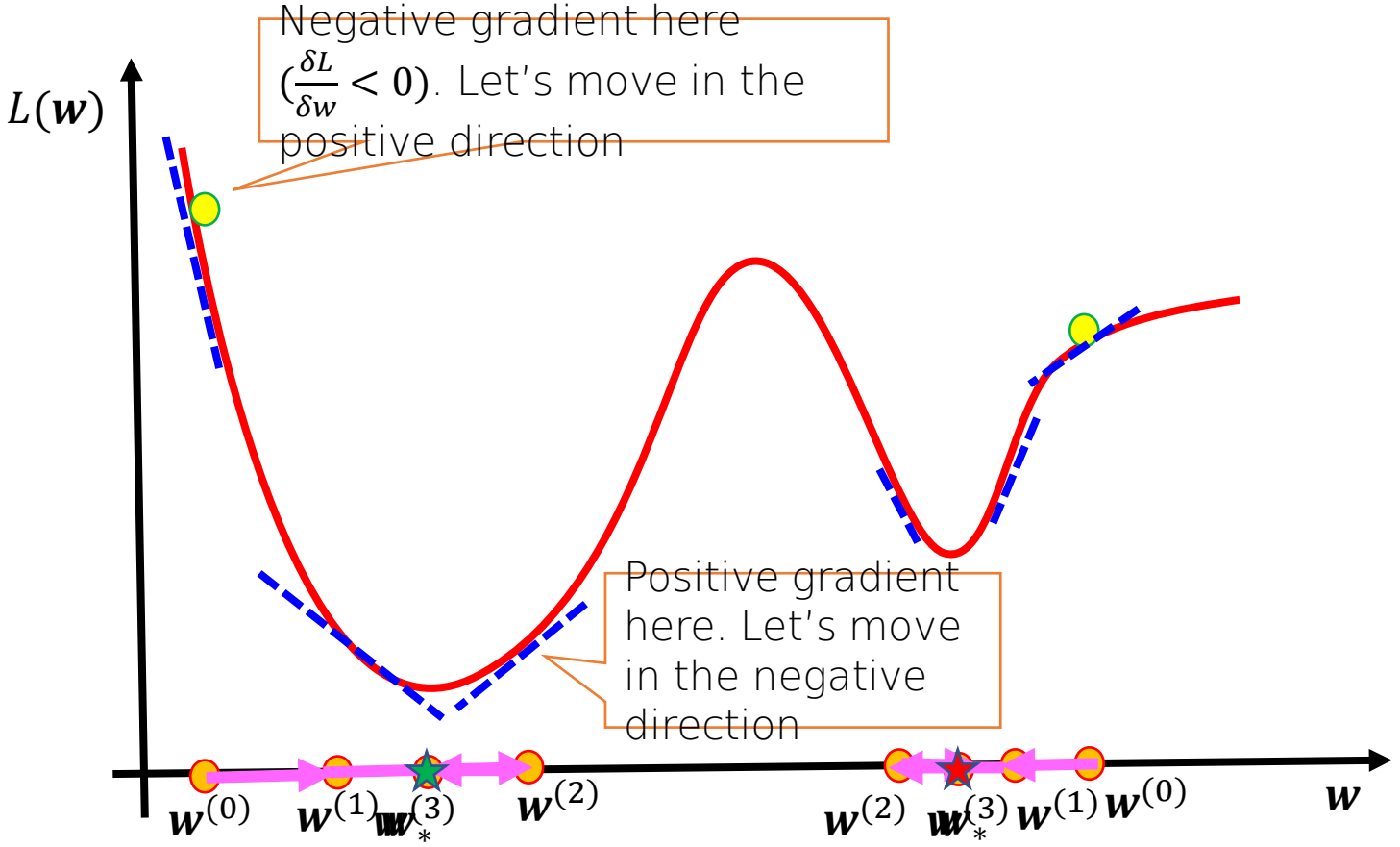
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

The learning rate very imp. Should be set carefully (fixed or chosen adaptively). Will discuss some strategies lat

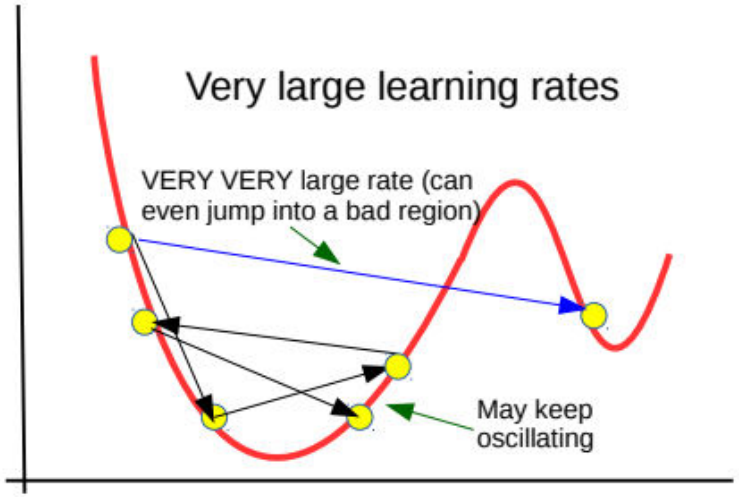
Will see the justification shortly

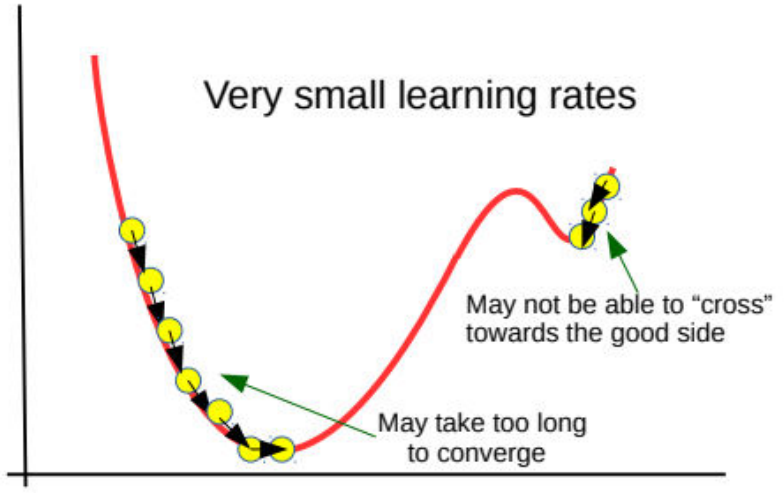
Sometimes may be tricky to to assess convergence? Will see some methods later

Gradient Descent: An Illustration



Learning rate is very importa





GD: An Example

- Let's apply GD for least squares linear regression

$$\boldsymbol{w}_{ridge} = \arg\min_{\boldsymbol{w}} L_{reg}(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- The gradient: $\boldsymbol{g} = -\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n$

- Each GD update will be of the form

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \sum_{n=1}^{N} 2 \left(y_n - \boldsymbol{w}^{(t)\top} \boldsymbol{x}_n\right) \boldsymbol{x}_n$$

Prediction error of current model $\boldsymbol{w}^{(t)}$ on the n^{th} training example

Training examples on which the current model's error is large contribute more to the update

- Exercise: Assume $N = 1$, and show that GD update improves prediction on the training input $(\boldsymbol{x}_n, y_n)$, i.e, y_n is closer to $\boldsymbol{w}^{(t+1)\top} \boldsymbol{x}_n$ than to $\boldsymbol{w}^{(t)\top} \boldsymbol{x}_n$
  - This is sort of a proof that GD updates are "corrective" in nature (and it actually is true not just for linear regression but can also be shown for

Coming up next

- Gradients when the function is non-differentiable
- Solving optimization problems
- Iterative optimization algorithms, such as gradient descent and its variants

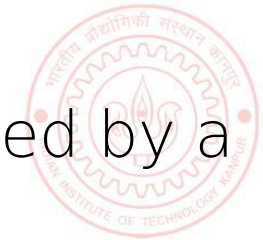
Optimization for ML (contd.)

CS771: Introduction to Machine Learning

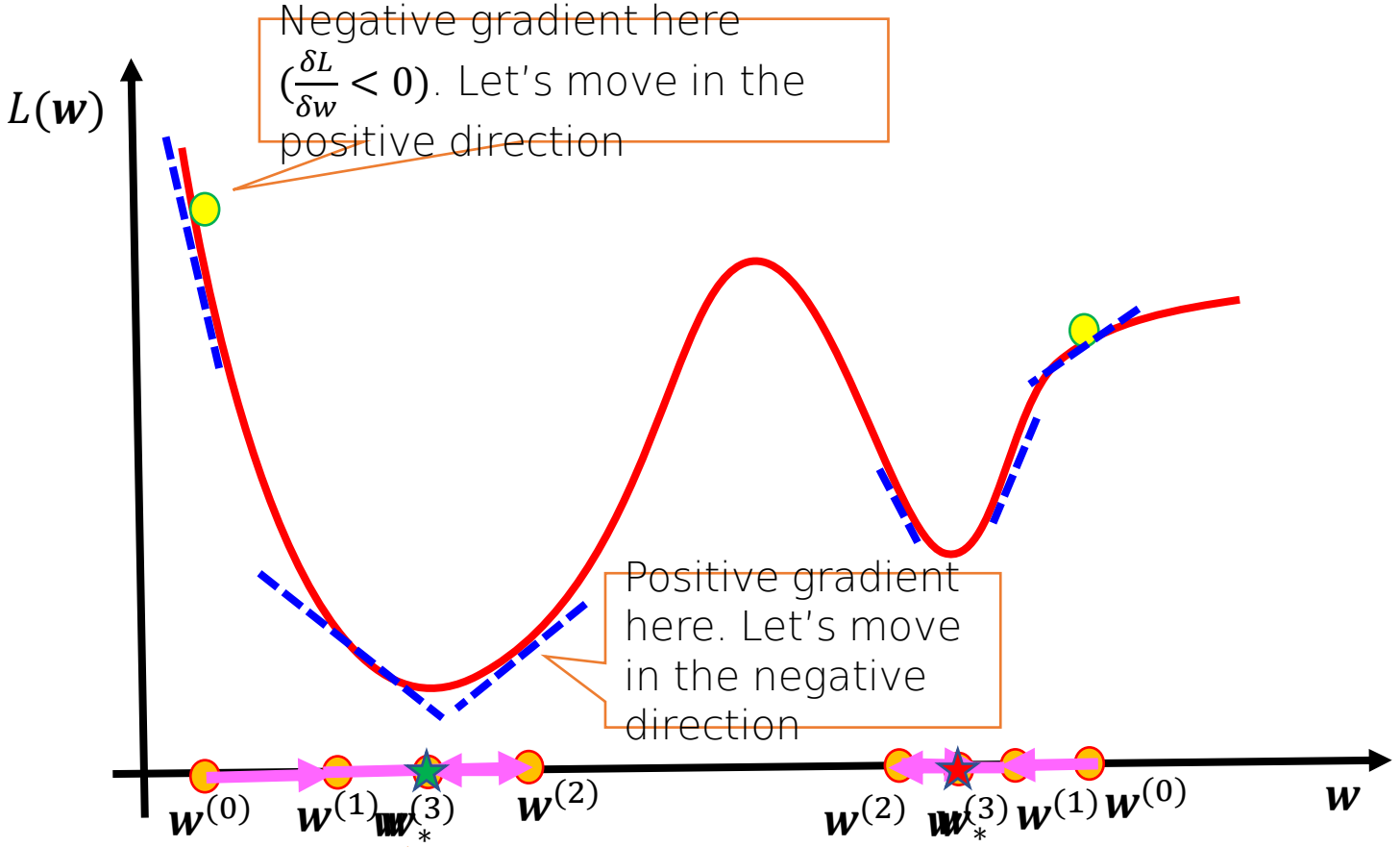
Nisheeth

Announcements

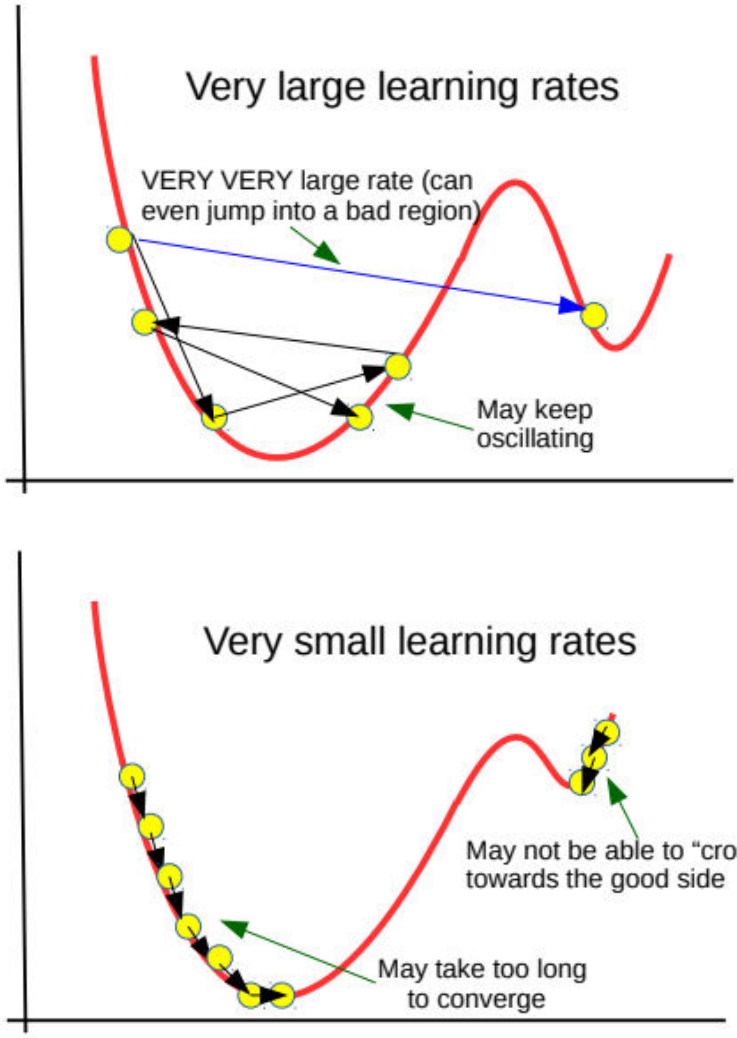
- Quiz 1 pro-rate policy
  - I will pro-rate marks for those who were able to attempt more than 15 marks worth of problems in Quiz 1 based on Quiz 1 mark percentiles (assumption: people would have performed at the same percentile of the class for unseen problems as seen problems)
  - I will pro-rate marks for those who were able to attempt less than or equal to 15 marks worth of problems in Quiz 1 based on Mid Sem exam percentiles
  - If anyone has an objection to this policy, please let me know ASAP
- Quiz 1 is graded; marks will show up on HelloIITK soon
  - 8 people scored full marks, suggesting the paper was not that hard
- Mid Sem exam is on 12th Sept
  - We will follow the same pattern as Quiz 1: real-time exam followed by a take-home assignment
- Syllabus will include everything covered till 10th Sept



Recall: Gradient Descent



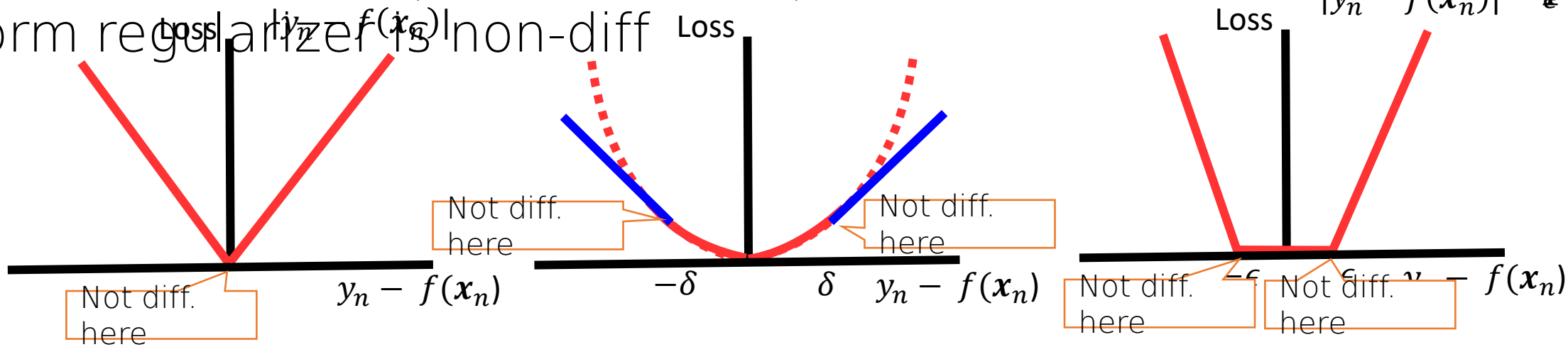
Learning rate is very importa



Dealing with Non-differentiable Functions

- In many ML problems, the objective function will be non-differentiable

- Some examples that we have already seen: Linear regression with absolute loss, or Huber loss, or ϵ-insensitive loss; even ℓ_1 norm regularizer is non-diff

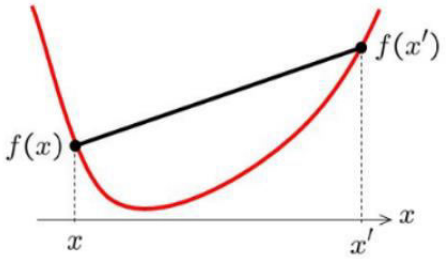


- Basically, any function in which there are points with kink is non-diff

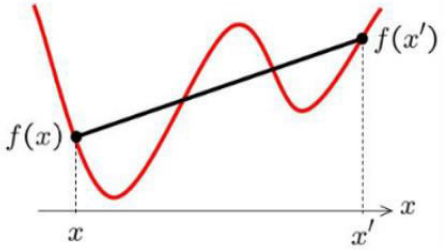
Recall: Convex Functions

- Informally, $f(x)$ is convex if all of its chords lie above the function everywhere

Convex Function



Non-convex Function



Note: "Chord lies above function" more formally means

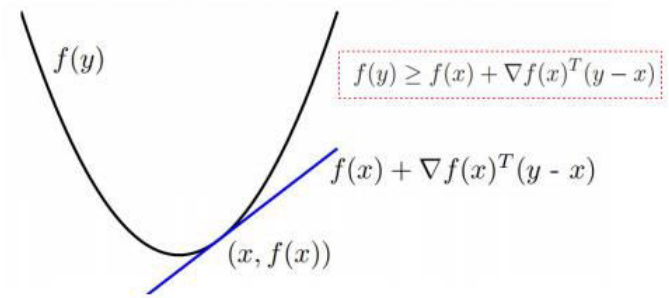
If f is convex then given

$$\alpha_1, \ldots, \alpha_n \quad \text{s.t} \ \sum_{i=1}^{n} \alpha_i = 1$$

$$f\left(\sum_{i=1}^{n} \alpha_i x_i\right) \leq \sum_{i=1}^{n} \alpha_i f(x_i)$$

Jensen's Inequality

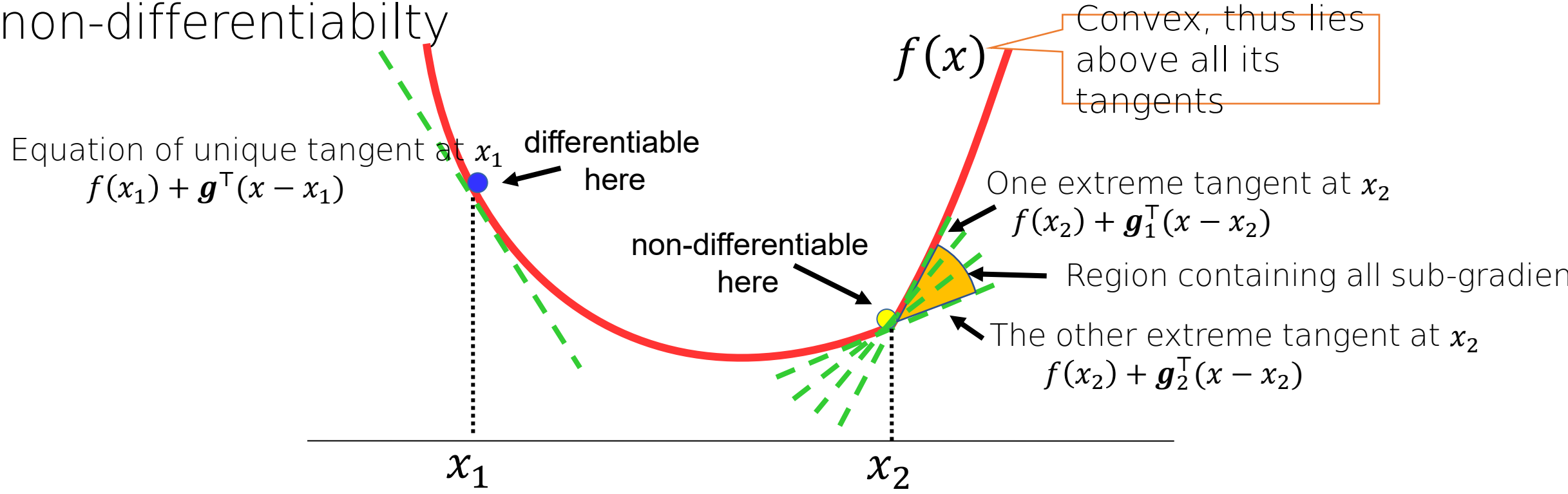
- Formally, (assuming differentiable function), some tests for convexity:
  - First-order convexity



bove all the tangents)

Sub-gradients

- For convex non-diff fn, can define sub-gradients at point(s) of non-differentiabilty



- For a convex, non-diff function $f(\boldsymbol{x})$, sub-gradient at $\boldsymbol{x}_*$ is <u>any</u> vector $\boldsymbol{g}$ s.t. $\forall \boldsymbol{x} \;\; f(\boldsymbol{x}) \geq f(\boldsymbol{x}_*) + \boldsymbol{g}^\top(\boldsymbol{x} - \boldsymbol{x}_*)$

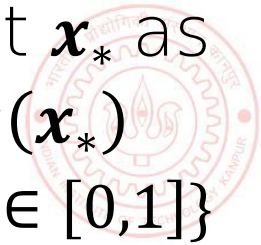
Sub-gradients, Sub-differential, and Some Rules

- Set of all sub-gradient at a non-diff point $\boldsymbol{x}_*$ is called the sub-differential

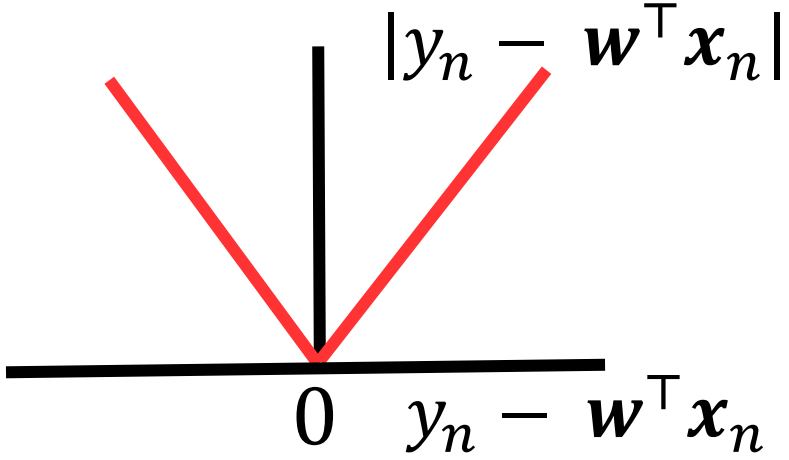
$$\partial f(\boldsymbol{x}_*) \triangleq \{\boldsymbol{g} : f(\mathbf{x}) \geq f(\boldsymbol{x}_*) + \boldsymbol{g}^\top(\boldsymbol{x} - \boldsymbol{x}_*) \quad \forall \mathbf{x}\}$$

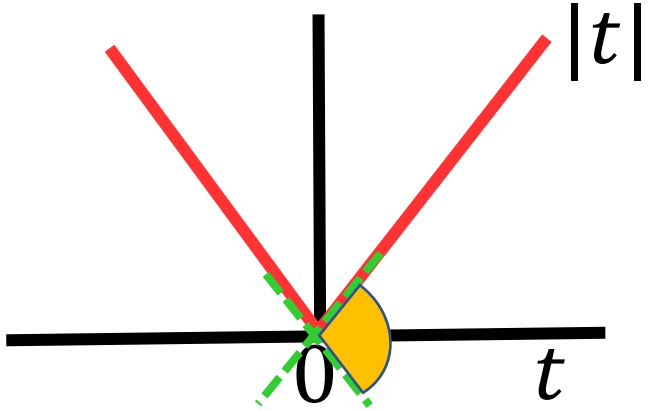
- Some basic rules of sub-diff calculus to keep in mi
  - Scaling rule: $\partial\big(c \cdot f(\mathbf{x})\big) = c \cdot \partial f(\mathbf{x}) = \{c \cdot \mathbf{v} : \mathbf{v} \in \partial f(\mathbf{x})\}$
  - Sum rule: $\partial\big(f(\mathbf{x}) + g(\mathbf{x})\big) = \partial f(\mathbf{x}) + \partial g(\mathbf{x})$ $= \{\mathbf{u} + \mathbf{v} : \mathbf{u} \in \partial f(\mathbf{x}), \mathbf{v} \in \partial g(\mathbf{x})\}$
  - Affine trans: $\partial f(\mathbf{a}^\top\mathbf{x} + b) = \partial f(t) \cdot \mathbf{a} = \{c \cdot \mathbf{a} : c \in \partial f(t)\}$, where $t = \mathbf{a}^\top\mathbf{x} + b$
  - Max rule: If $h(\boldsymbol{x}) = \max\{f(\boldsymbol{x}), g(\boldsymbol{x})\}$ then we calculate $\partial h(\boldsymbol{x})$ at $\boldsymbol{x}_*$ as
    - If $f(\boldsymbol{x}_*) > g(\boldsymbol{x}_*), \partial h(\boldsymbol{x}_*) = \partial f(\boldsymbol{x}_*)$, If $g(\boldsymbol{x}_*) > f(\boldsymbol{x}_*), \partial h(\boldsymbol{x}_*) = \partial g(\boldsymbol{x}_*)$
    - If $f(\boldsymbol{x}_*) = g(\boldsymbol{x}_*), \partial h(\boldsymbol{x}_*) = \{\alpha\mathbf{a} + (1-\alpha)\mathbf{b} : \mathbf{a} \in \partial f(\boldsymbol{x}_*), \mathbf{b} \in \partial g(\boldsymbol{x}_*), \alpha \in [0,1]\}$

The affine transform rule is a special case of the more general chain rule



Sub-Gradient For Absolute Loss Regression





$$\partial|t| = \begin{cases} 1 & \text{if } t > 0 \\ -1 & \text{if } t < 0 \\ [-1, +1] & \text{if } t = 0 \end{cases}$$

- The loss function for linear reg. with absolute loss: $L(\boldsymbol{w}) = |y_n - \boldsymbol{w}^\top \boldsymbol{x}_n|$
- Non-differentiable at $y_n - \boldsymbol{w}^\top \boldsymbol{x}_n = 0$
- Can use the affine transform rule of sub-diff calculus
- Assume $t = y_n - \boldsymbol{w}^\top \boldsymbol{x}_n$. Then $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_n \partial|t|$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_n \times 1 = -x_n$ if $t > 0$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_n \times -1 = x_n$ if $t < 0$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_n \times c = -cx_n$ where $c \in [-1, +1]$ if $t = 0$

Sub-Gradient Descent

- Suppose we have a non-differentiable function $L(\boldsymbol{w})$
- Sub-gradient descent is almost identical to GD except we use subgradients

Sub-Gradient Descent

- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2, \ldots$ (or until convergence)
  - Calculate the sub-gradient $\boldsymbol{g}^{(t)} \in \partial L(\boldsymbol{w}^{(t)})$
  - Set the learning rate $\eta^{(t)}$
  - Move in the opposite direction of subgradient

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

Stochastic Gradient Descent (SGD)

Writing as an average instead of sum. Won't affect minimization of $\boldsymbol{L(w)}$

- Consider a loss function of the form $\boldsymbol{L(w) = \frac{1}{N}\sum_{n=1}^{N} \ell_n(w)}$

Expensive to compute – requires doing it for all the training examples in each iteration

- The (sub)gradient in this case can be written as

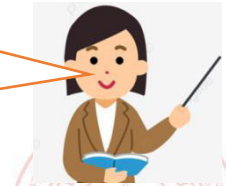
$$\boldsymbol{g = \nabla_w L(w) = \nabla_w [\frac{1}{N}\sum_{n=1}^{N} \ell_n(w)] = \frac{1}{N}\sum_{n=1}^{N} g_n}$$

(Sub)gradient of the loss on $\boldsymbol{n^{th}}$ training example

- Stochastic Gradient Descent (SGD) approximates $\boldsymbol{g}$ using a single training example

- At iter. $\boldsymbol{t}$, pick an index $\boldsymbol{i \in \{1,2,\ldots,N\}}$ uniformly ra
- approximate $\boldsymbol{g}$ as $\boldsymbol{g \approx g_i = \nabla_w \ell_i(w)}$

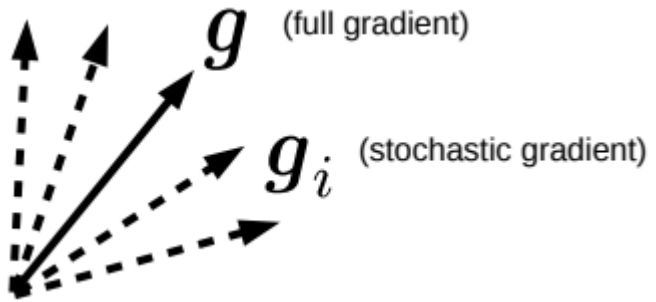
Can show that $\boldsymbol{g_i}$ is an unbiased estimate of $\boldsymbol{g}$, i.e., $\boldsymbol{\mathbb{E}[g_i] = g}$



- May take more iterations than GD to converge but each iteration is

Minibatch SGD

- Gradient approximation using a single training example may be noisy



The approximation may have a high variance – may slow down convergence, updates may be unstable, and may even give sub-optimal solutions (e.g., local minima where GD might have given global minima)

- We can use $B > 1$ unif. rand. chosen train. ex. with indices $\{i_1, i_2, \ldots, i_B\} \in \{1,2,\ldots,N\}$
- Using this "minibatch" of examples, we can compute a minibatch gradient

$$\boldsymbol{g} \approx \frac{1}{B}\sum_{b=1}^{B} \boldsymbol{g}_{i_b}$$

- Averaging helps in reducing the variance in the stochastic

Constrained Optimization

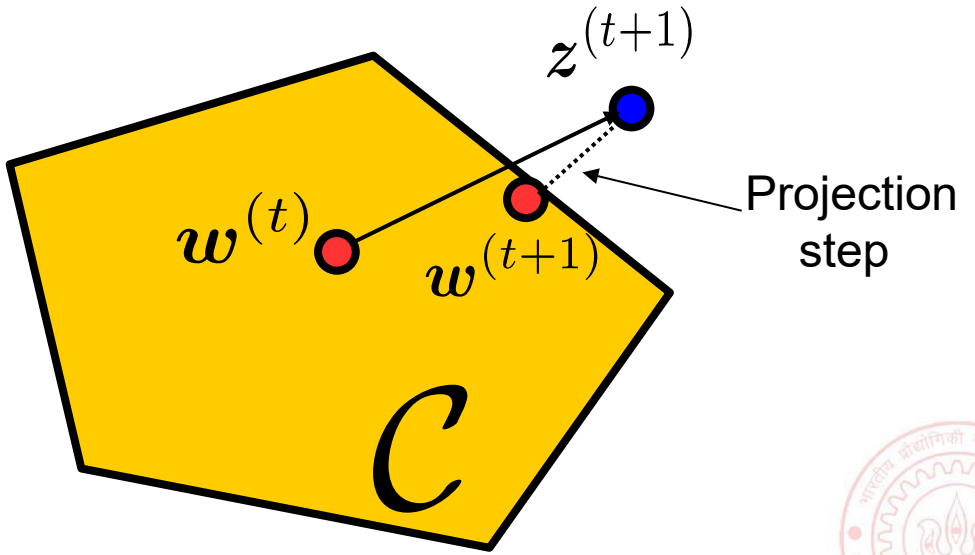
Projected Gradient Descent

- Consider an optimization problem of the form

$$w_{opt} = \arg\min_{w \in \mathcal{C}} L(\boldsymbol{w})$$

- Projected GD is very similar to GD with an extra projection step
- Each iteration t will be of the form
  - Perform update: $\boldsymbol{z}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$
  - Check if $\boldsymbol{z}^{(t+1)}$ satisfies constr
    - If $\boldsymbol{z}^{(t+1)} \in \mathcal{C}$, set $\boldsymbol{w}^{(t+1)} = \boldsymbol{z}^{(t+1)}$
    - If $\boldsymbol{z}^{(t+1)} \notin \mathcal{C}$, project as $\boldsymbol{w}^{(t+1)} = \Pi_{\mathcal{C}}[\boldsymbol{z}^{(t+1)}]$

Projecti on operato r

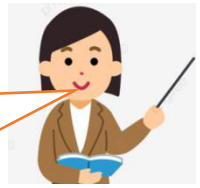


Projected GD: How to Project?

- Here projecting a point means finding the "closest" point from the constraint set

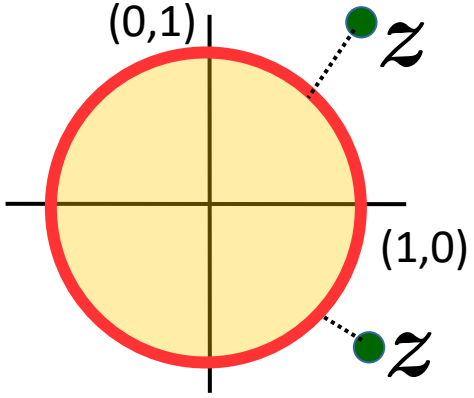
$$\Pi_{\mathcal{C}}[\boldsymbol{z}] = \arg\min_{\boldsymbol{w}\in\mathcal{C}} \|\boldsymbol{z}-\boldsymbol{w}\|^2$$

Projected GD commonly used only when the projection step is simple and efficient to compute



- For some sets $\mathcal{C}$, the projection step is easy

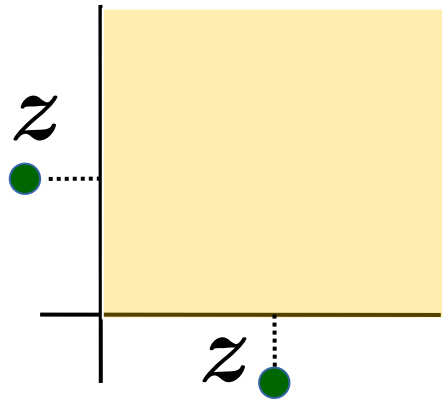
$\mathcal{C}$: Unit radius ℓ_2 ball



Projection = Normalize to unit Euclidean length vector

$$\hat{\mathbf{x}} = \begin{cases} \mathbf{x} & \text{if } \|\mathbf{x}\|_2 \le 1 \\ \frac{\mathbf{x}}{\|\mathbf{x}\|_2} & \text{if } \|\mathbf{x}\|_2 > 1 \end{cases}$$

$\mathcal{C}$: Set of non-negative reals



Projection = Set each negative entry in $\boldsymbol{z}$ to be zer

$$\hat{\mathbf{x}}_i = \begin{cases} \mathbf{x}_i & \text{if } \mathbf{x}_i \ge 0 \\ 0 & \text{if } \mathbf{x}_i < 0 \end{cases}$$

Proximal Gradient Descent

- Consider minimizing a regularized loss function of the form

$$\arg\min_w L(\boldsymbol{w}) + R(\boldsymbol{w})$$

Note: The reg. hyperparam. λ assumed part of $R(\boldsymbol{w})$ itself

- Proximal GD popular when regularizer $R(\boldsymbol{w})$ is non-differentiable
- Basic idea: Do GD on $L(\boldsymbol{w})$ and use a prox. operator to regularize via $R(\boldsymbol{w})$
- For a func. R, its prox. operator is

$$\text{prox}_R(\boldsymbol{z}) = \arg\min_{\boldsymbol{w}} \left[R(\boldsymbol{w}) + \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{w}\|_2^2 \right]$$

Proximal GD

Assume reg. is $R(\boldsymbol{w})$

- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2,\ldots$ (or until convergence)
  - Calculate the (sub)gradient of train. Loss (w/o reg.)

$$\boldsymbol{g}^{(t)} \in \partial L(\boldsymbol{w}^{(t)})$$

  - Set learning rate η_t
  - Step 1: $\boldsymbol{z}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$
  - Step 2: $\boldsymbol{w}^{(t+1)} = \text{prox}_{R(\boldsymbol{w})}(\boldsymbol{z}^{(t+1)})$

Special Cases

For $R(\boldsymbol{w}) = 0.5 \times \|\mathbf{w}\|_2^2$
$\text{prox}_R(\boldsymbol{z}) = \boldsymbol{z}/2$ i.e. scaling

That is, regularize by reducing the value of each component of the the vector $\boldsymbol{z}$ by half

If $R(\boldsymbol{w})$ defines a set based constraint

$$R(\boldsymbol{w}) := \boldsymbol{w} \in \mathcal{C}$$

$$\text{prox}_R(\boldsymbol{z}) = \arg\min_{\boldsymbol{w}\in\mathcal{C}} \|\boldsymbol{z} - \boldsymbol{w}\|^2$$

Prox. GD becomes equivalent to projected GD

Constrained Opt. via Lagrangian

- Consider the following constrained minimization problem (using f instead of L)

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} f(\boldsymbol{w}), \quad \text{s.t.} \quad g(\boldsymbol{w}) \leq 0$$

- Note: If constraints of the form $g(w) \geq 0$, use $-g(w) \leq 0$
- Can handle multiple inequality and equality constraints too (will see later)
- Can transform th

$$\hat{\boldsymbol{w}} = \arg\min f(\boldsymbol{w}) + c(\boldsymbol{w})$$

ing equivalent unconstrained pr

$$c(\boldsymbol{w}) = \max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) = \begin{cases} \infty, & \text{if } g(\boldsymbol{w}) > 0 \quad \text{(constraint violated)} \\ 0 & \text{if } g(\boldsymbol{w}) \leq 0 \quad \text{(constraint satisfied)} \end{cases}$$

- Our problem can

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \left\{ f(\boldsymbol{w}) + \arg\max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) \right\}$$

Constrained Opt. via Lagrangian

The Lagrangian: $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}$

- Therefore, we can write our original problem as

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \left\{ f(\boldsymbol{w}) + \arg\max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) \right\} = \arg\min_{\boldsymbol{w}} \left\{ \arg\max_{\alpha \geq 0} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\}$$

- The Lagrangian is now optimized w.r.t. $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ (Lagrange multiplier)
- We can defined Primal and Dual problem as

$$\hat{\boldsymbol{w}}_P = \arg\min_{\boldsymbol{w}} \left\{ \arg\max_{\alpha \geq 0} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\} \quad \text{(primal problem)}$$

$$\hat{\boldsymbol{w}}_D = \arg\max_{\alpha \geq 0} \left\{ \arg\min_{\boldsymbol{w}} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\} \quad \text{(dual problem)}$$

Both equal if $f(w)$ and the set $g(w) \leq 0$ are convex

Constrained Opt. with Multiple Constraints

- We can also have multiple inequality and equality constraints

$$\begin{aligned} \hat{\boldsymbol{w}} &= \arg\min_{\boldsymbol{w}} f(\boldsymbol{w}) \\ \text{s.t.} \quad & g_i(\boldsymbol{w}) \leq 0, \quad i = 1, \ldots, K \\ & h_j(\boldsymbol{w}) = 0, \quad j =, 1, \ldots, L \end{aligned}$$

- Introduce Lagrange multipliers $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_K]$ and $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_L]$
- The L [illegible] be

$$\hat{\boldsymbol{w}}_P = \arg\min_{\boldsymbol{w}} \{\arg\max_{\boldsymbol{\alpha}\geq 0, \boldsymbol{\beta}} \{f(\boldsymbol{w}) + \sum_{i=1}^{K} \alpha_i g_i(\boldsymbol{w}) + \sum_{j=1}^{L} \beta_j h_j(\boldsymbol{w})\}\}$$

$$\hat{\boldsymbol{w}}_D = \arg\max_{\boldsymbol{\alpha}\geq 0, \boldsymbol{\beta}} \{\arg\min_{\boldsymbol{w}} \{f(\boldsymbol{w}) + \sum_{i=1}^{K} \alpha_i g_i(\boldsymbol{w}) + \sum_{j=1}^{L} \beta_j h_j(\boldsymbol{w})\}\}$$

Some other useful optimization methods

Co-ordinate Descent (CD)

- Standard gradient descent update for : $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$
- CD: In each iter, update only one entry (co-ordinate) of $\boldsymbol{w}$. Keep all others fixed

$$w_d^{(t+1)} = w_d^{(t)} - \eta_t g_d^{(t)} \qquad d \in \{1,2,\ldots,D\}$$

$g_d = \nabla_{w_d} L(\boldsymbol{w})$ -- partial derivative w.r.t. the d^{th} element of vector $\boldsymbol{w}$ (or the d^{th} element of the gradient vector $\boldsymbol{g}^{(t)}$)

$w^{(t+1)}$ = $w^{(t)}$ − η_t

- Cost of each update is now independent of D
- In each iter, can choose co-ordinate to update unif. randomly or in cyclic order
- Instead of updating a single co-ord, can also update "blocks" of co-ordinates
  - Called Block co-ordinate descent (BCD)
- To avoid $O(D)$ cost of gradient computation, can cache previous

Alternating Optimization (ALT-OPT)

- Consider opt. problems with several variables, say two variables $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$

$$\{\hat{\boldsymbol{w}}_1, \hat{\boldsymbol{w}}_2\} = \arg\min_{\boldsymbol{w}_1, \boldsymbol{w}_2} \mathcal{L}(\boldsymbol{w}_1, \boldsymbol{w}_2)$$

- Often, this "joint" optimization is hard/impossible to solve
- We c ve such prob

ALT-OPT

1. Initialize one of the variables, e.g., $\boldsymbol{w}_2 = \boldsymbol{w}_2^{(0)}, t = 0$
2. Solve $\boldsymbol{w}_1^{(t+1)} = \arg\min_{\boldsymbol{w}_1} \mathcal{L}(\boldsymbol{w}_1, \boldsymbol{w}_2^{(t)})$ $\quad //\boldsymbol{w}_2$ "fixed" at its most recent value $\boldsymbol{w}_2^{(t)}$
3. Solve $\boldsymbol{w}_2^{(t+1)} = \arg\min_{\boldsymbol{w}_2} \mathcal{L}(\boldsymbol{w}_1^{(t+1)}, \boldsymbol{w}_2)$ $\quad //\boldsymbol{w}_1$ "fixed" at its most recent value $\boldsymbol{w}_1^{(t+1)}$
4. $t = t + 1$. Go to step 2 if not converged yet.

- Usually converges to a local optima. But very very useful. Will

Some Practical Aspects: Initialization

- Iterative opt. algos like GD, SGD, etc need to be initialized to "good" values
  - Bad initialization can result on bad local optima
- Mainly a concern for non-convex loss functions, not so much for convex loss functions
- Transfer Learning: Initialize using params of a model trained on a related dataset
- Initialize using solution of a simpler but related model
  - E.g., for multitask regression (say T coupled regression problems), initialize using the solutions of the T independently trained regression problems
- For deep learning models, initialization is very important
  - Transfer learning approach is often used (initialize using "pre-trained" model

Some Practical Aspects: Assessing Convergence

- Various ways to assess convergence, e.g. consider converged if
  - The objective's value (on train set) ceases to change much across iterations

$$L(\boldsymbol{w}^{(t+1)}) - L(\boldsymbol{w}^{(t)}) < \epsilon$$ (for some small pre-defined ϵ)

  - The parameter values cease to change much across iterations

$$\|\boldsymbol{w}^{(t+1)} - \boldsymbol{w}^{(t)}\| < \tau$$ (for some small pre-defined τ)

  - Above condition is also equivalent to saying that the gradients are close to zero

$$\|\boldsymbol{g}^{(t)}\| \rightarrow 0$$

  - The objective's value has become small enough that we are happy with
  - Use a validation set to assess if the model's performance is acceptable (early stopping)

Some Practical Aspects: Learning Rate (Step Size)

- Some guidelines to select good learning rate (a.k.a. st
- For convex functions, setting η_t something like C/t or $C/\sqrt{t}$ often works well
  - These step-sizes are actually theoretically optimal in some settings
  - In general, we want the learning rates to satisfy the following conditions
    - $\eta_t \to 0$ as t becomes very very large
    - $\sum \eta_t = \infty$ (needed to ensure that we can potentially reach anywhere in the parameter space)
  - Sometimes carefully chosen constant learning rates (usually small, or initially large and later small) also work well in practice
- Can also search for the "best" step-size by solving

$$\eta_t = \arg\min_{\eta \geq 0} f\left(\mathbf{w}^{(t)} - \eta \cdot \mathbf{g}^{(t)}\right)$$

C is a hyperparameter

Also called "line search"

A one-dim optimization problem (note that $\mathbf{w}^{(t)}$ and $\mathbf{g}^{(t)}$ are fixed)

- An faster alternative to line search is the Armijo-Goldstein rule

Some Practical Aspects: Adaptive Gradient Methods

- Can also use different learning rate in different dime

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \boldsymbol{e}^{(t)} \odot \boldsymbol{g}^{(t)}$$

Vector of learning rates along each dimension

Element-wise product of two vectors

$$e_d^{(t)} = \frac{1}{\sqrt{\epsilon + \sum_{\tau=1}^{t} \left(g_d^{(t)}\right)^2}}$$

If some dimension had big updates recently (marked by large gradient values), show down along those directions by using smaller learning rates - AdaGrad (Duchi et al, 2011)

- Can use a momentum term to stabilize gradients by reusing info from past grads
  - Move faster along directions that were previously goo
  - Slow down along directions where gradient has c

β usually set as 0.9

The "momentum" term. Set to 0 at initialization

$$\boldsymbol{m}^{(t)} = \beta \boldsymbol{m}^{(t-1)} + \eta_t \boldsymbol{g}^{(t)}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{m}^{(t)}$$

In an even faster version of this, $\boldsymbol{g}^{(t)}$ is replaced by the gradient computed at the next step if previous direction were used, i.e., $\nabla L(\boldsymbol{w}^{(t)} - \beta \boldsymbol{m}^{(t-1)})$. Called Nesterov's Accelerated Gradient (NAG) method

- Also exists several more advanced methods that combine the above methods

Optimization for ML: Summary

- Gradient methods are simple to understand and implement
- More sophisticated optimization methods also often use gradient methods
- Backpropagation algo used in deep neural nets is GD + chain rule of differentiation
- Use subgradient methods if function not differentiable
- Alternating optimization helps in situations where joint optimization is intractable

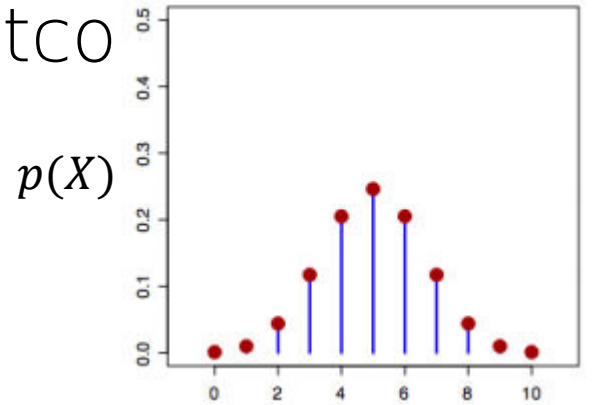
Probability Basics

CS771: Introduction to Machine Learning

Nisheeth

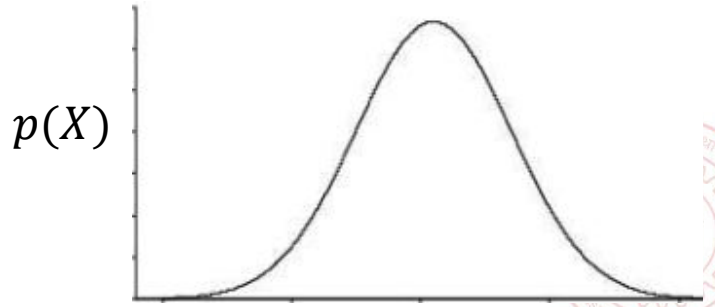
Random Variables

- Informally, a random variable (r.v.) X denotes possible outcomes of an event
- Can be discrete (i.e., finite many possible outco continuous
- Some examples of discrete r.v.
  - $X \in \{0, 1\}$ denoting outcomes of a coin-toss
  - $X \in \{1, 2, \ldots, 6\}$ denoting outcome of a dice roll



X (a discrete r.v.)

- Some examples of continuous r.v.
  - $X \in (0, 1)$ denoting the bias of a coin
  - $X \in \mathbb{R}$ denoting heights of students in CS771
  - $X \in \mathbb{R}$ denoting time to get to your hall from the department

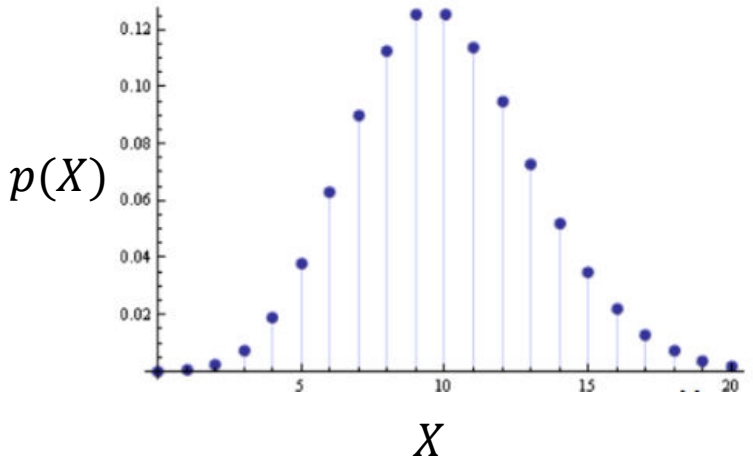


X (a continuous r.v.)

Discrete Random Variables

- For a discrete r.v. X, $p(x)$ denotes $p(X = x)$ - probability that $X = x$
- $p(X)$ is called the probability mass function (PMF) of r.v. X
  - $p(x)$ or $p(X = x)$ is the value of the PMF at x

$$p(x) \geq 0$$
$$p(x) \leq 1$$
$$\sum_x p(x) = 1$$



Continuous Random Variables

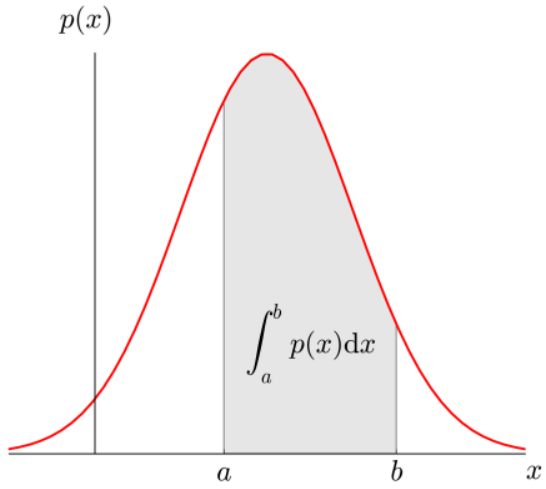
- For a continuous r.v. X, a *probability* $p(X = x)$ or $p(x)$ is meaningless

- For cont. r.v., we talk in terms of prob. within an interval $X \in (x, x + \delta x)$
  - $p(x)\delta x$ is the prob. that $X \in (x, x + \delta x)$ as $\delta x \to 0$
  - $p(x)$ is the probability density at $X = x$

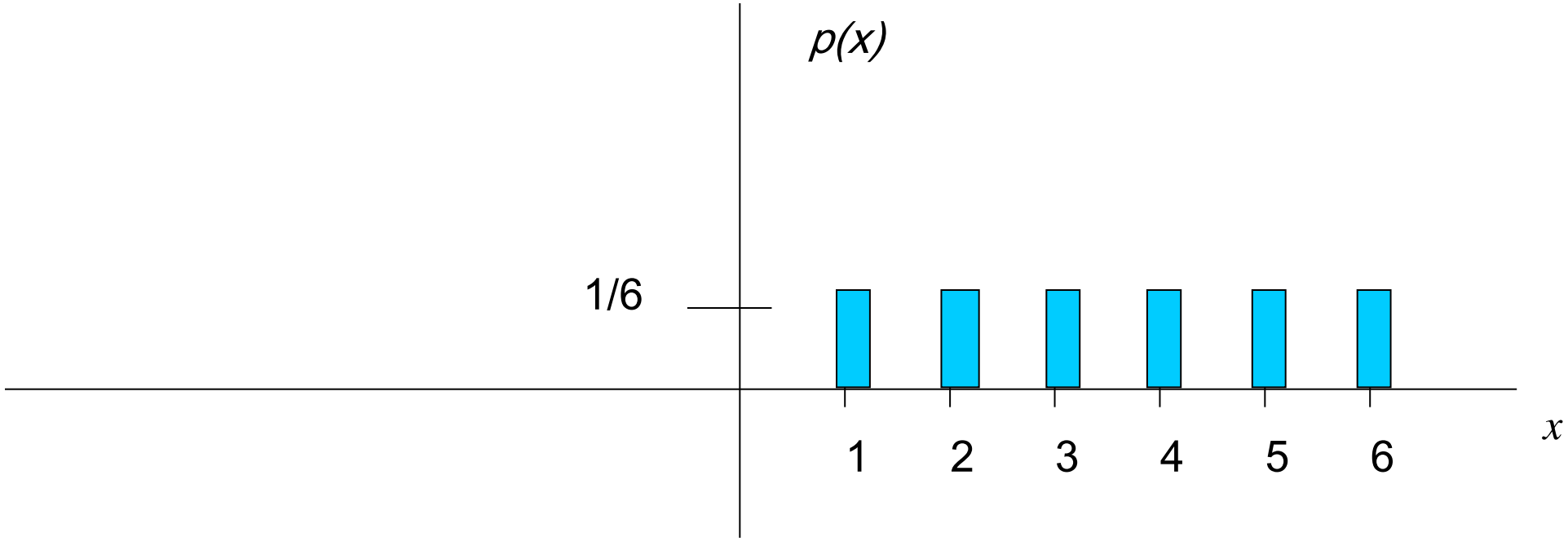
$$p(x) \geq 0$$

$$\cancel{p(x) \leq 1}$$

$$\int p(x)dx = 1$$



Discrete example: roll of a die



$$\sum_{\text{all } x} P(x) = 1$$

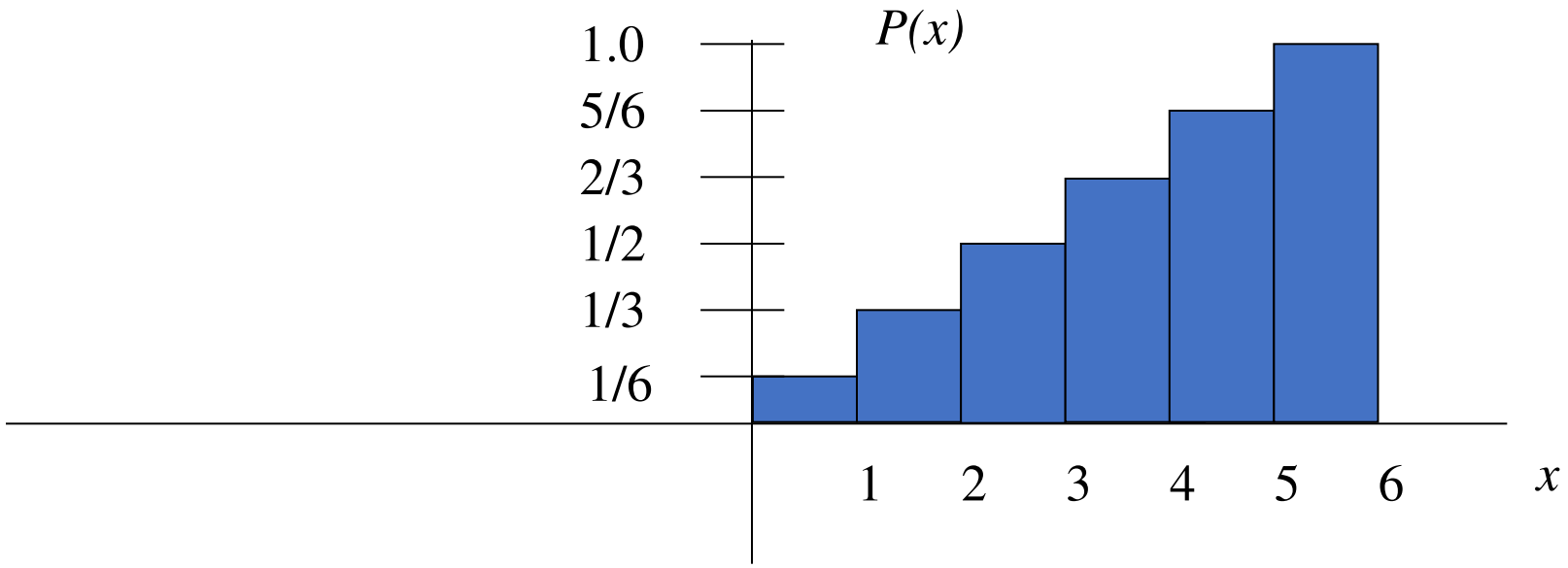
Probability mass function (pmf)

x	$p(x)$
1	$p(x=1)=1/6$
2	$p(x=2)=1/6$
3	$p(x=3)=1/6$
4	$p(x=4)=1/6$
5	$p(x=5)=1/6$
6	$p(x=6)=1/6$

Cumulative distribution function

x	$P(x \leq A)$
1	$P(x \leq 1) = 1/6$
2	$P(x \leq 2) = 2/6$
3	$P(x \leq 3) = 3/6$
4	$P(x \leq 4) = 4/6$
5	$P(x \leq 5) = 5/6$
6	$P(x \leq 6) = 6/6$

Cumulative distribution function (CDF)



Practice Problem

- The number of patients seen in a clinic in any given hour is a random variable represented by *x*. The probability distribution for *x* is:

x	10	11	12	13	14
P(x)	.4	.2	.2	.1	.1

Find the probability that in a given hour:

a. exactly 14 patients arrive $p(x=14)= .1$

b. At least 12 patients arrive $p(x\geq 12)= (.2 + .1 + .1) = .4$

c. At most 11 patients arrive $p(x\leq 11)= (.4 + .2) = .6$

Continuous case

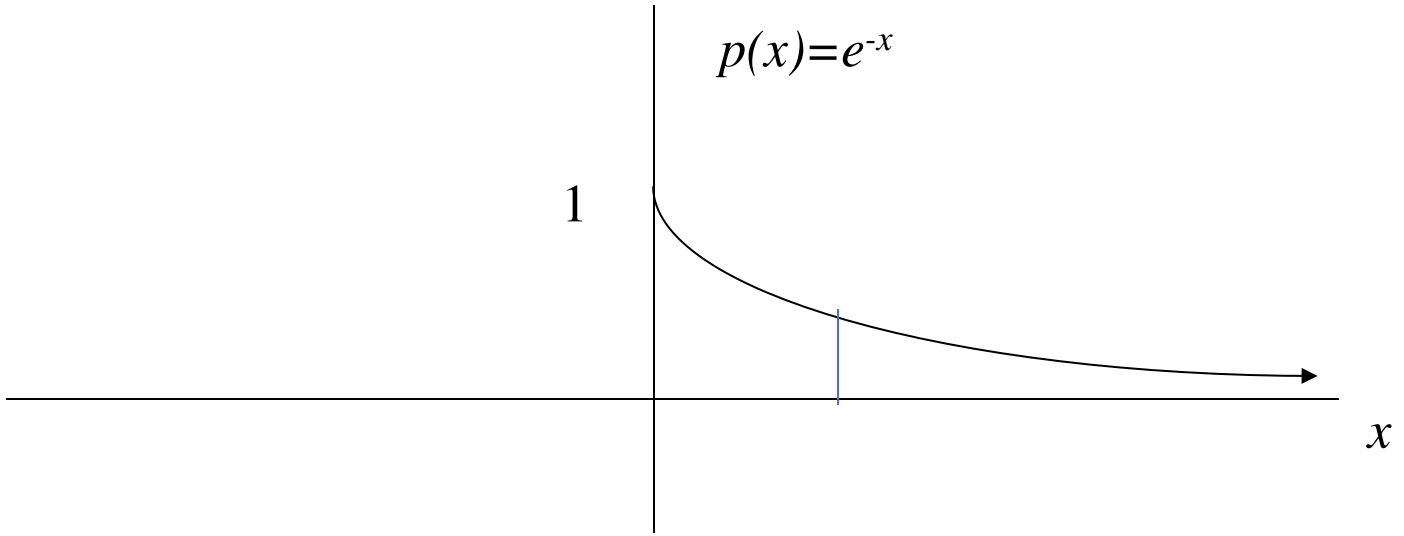
- The probability function that accompanies a continuous random variable is a continuous mathematical function that integrates to 1.
  - For example, recall the negative exponential function (in probability, this is called an “exponential distribution”):

$$f(x) = e^{-x}$$

  - This function integrates to 1:

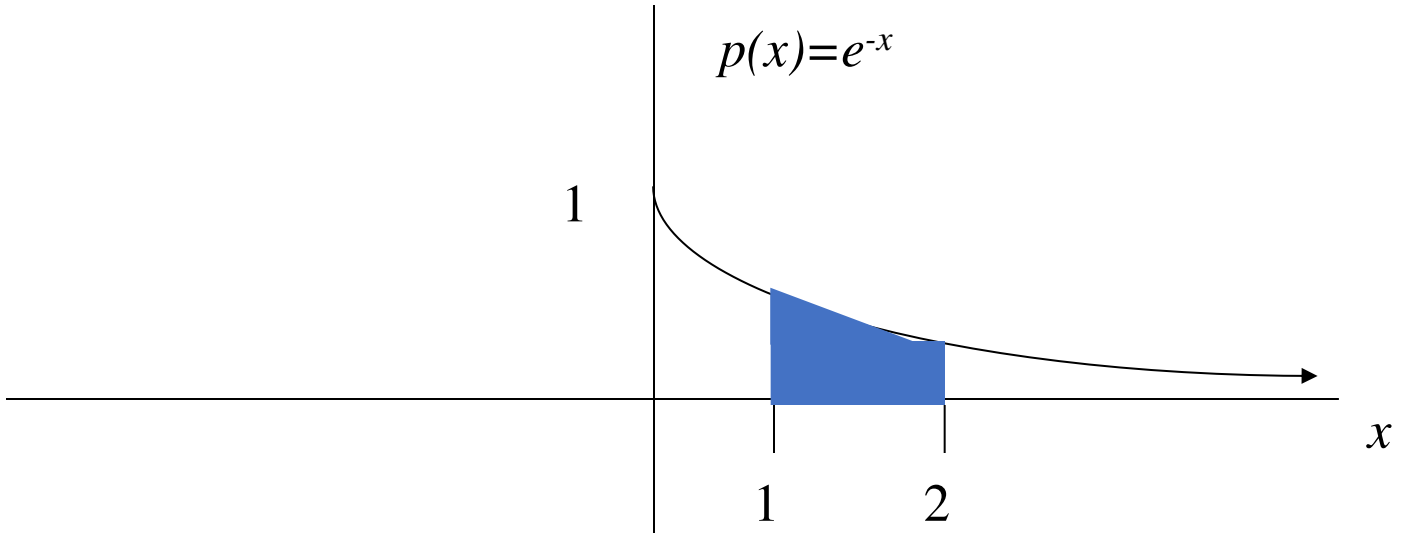
$$\int_0^{+\infty} e^{-x} = -e^{-x} \Big|_0^{+\infty} = 0 + 1 = 1$$

Continuous case: "probability density function" (pdf)



The probability that x is any exact particular value (such as 1.9976) is 0; we can only assign probabilities to possible ranges of x.

For example, the probability of x falling within 1 to 2:

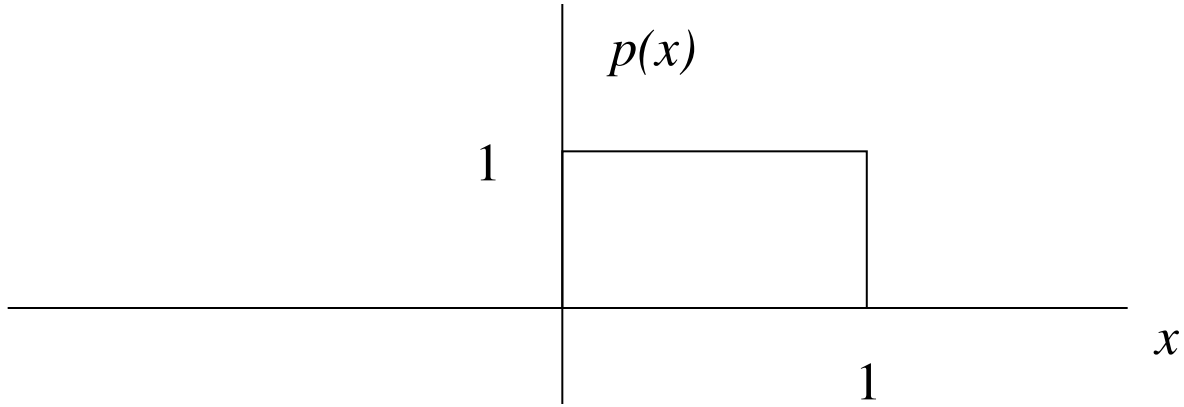


$$P(1 \le x \le 2) = \int_1^2 e^{-x} = -e^{-x} \Big|_1^2 = -e^{-2} - -e^{-1} = -.135 + .368 = .23$$

Example 2: Uniform distribution

The uniform distribution: all values are equally likely.

$f(x) = 1$, for $1 \geq x \geq 0$

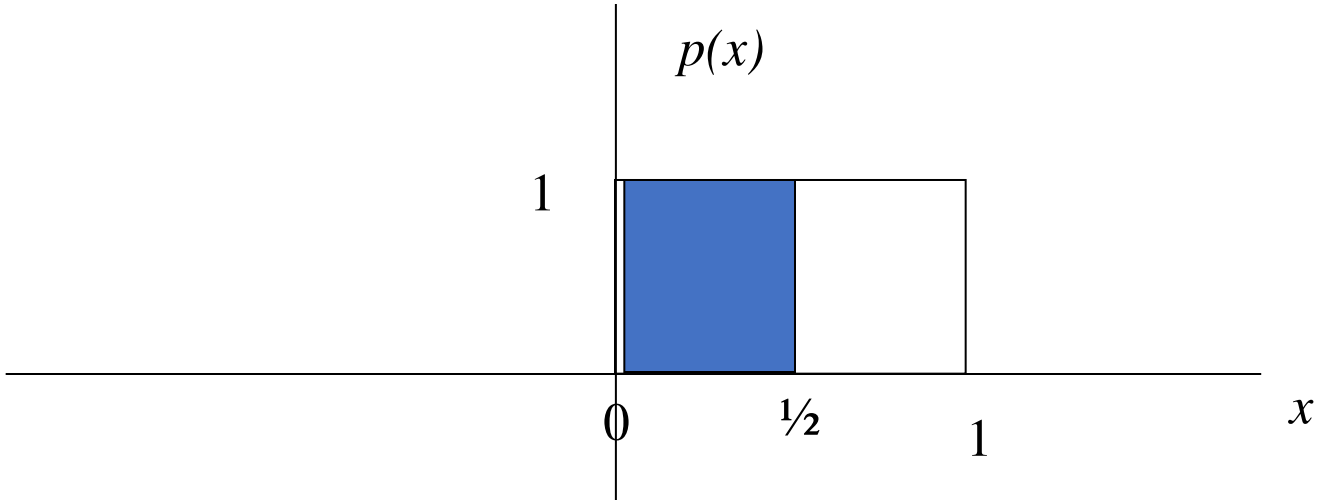


We can see it's a probability distribution because it integrates to 1 (the area under the curve is 1):

$$\int_0^1 1 = x \Big|_0^1 = 1 - 0 = 1$$

Example: Uniform distribution

What's the probability that x is between 0 and ½?

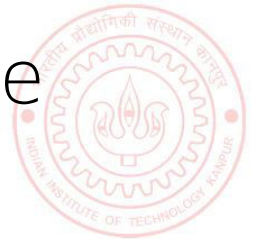


$$P(½ \geq x \geq 0) = ½$$

A word about notation

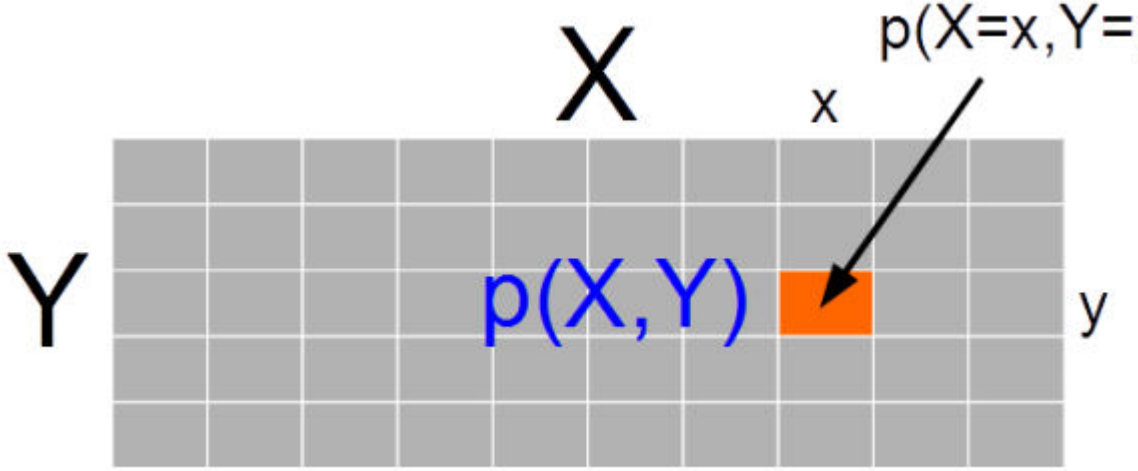
- $p(.)$ can mean different things depending on the context
- $p(X)$ denotes the distribution (PMF/PDF) of an r.v. X
- $p(X = x)$ or $p_X(x)$ or simply $p(x)$ denotes the prob. or prob. density at value x
  - Actual meaning should be clear from the context (but be careful)
- Exercise same care when $p(.)$ is a specific distribution (Bernoulli, Gaussian, etc.)
- The following means generating a random sample from the distribution $p(X)$

$$x \sim p(X)$$



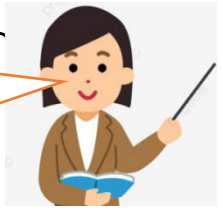
Joint Probability Distribution

- Joint prob. dist. $p(X, Y)$ models probability of co-occurrence of two r.v. X, Y
- For discrete r.v., the joint PMF $p(X, Y)$ is ... ur



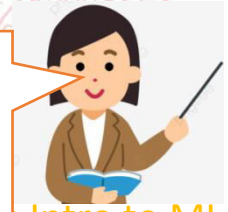
$$\sum_x \sum_y p(X = x, Y = y) = 1$$

For 3 r.v.'s; we will likewise have a "cube" for the PMF. For more than 3 r.v.'s too, similar analogy holds



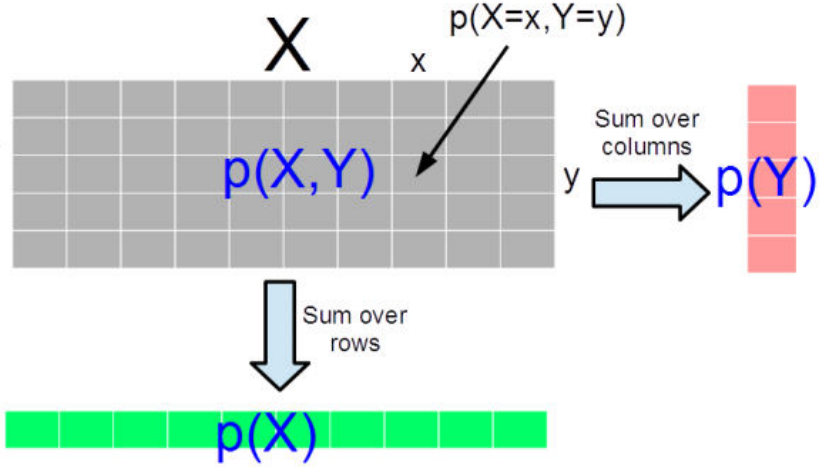
- For two cont $\int_x \int_y p(X = x, Y = y) dx dy = 1$ e have

For more than two r.v.'s, we will likewise have a multi-dim integral for this property



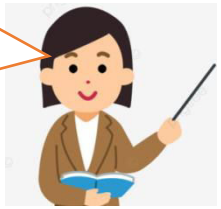
Marginal Probability Distribution

- Consider two r.v.'s X and Y (discrete/continuous – both need not of same type)
- Marg. Prob. is PMF/PDF of one r.v. accounting for all possibilities of the other r.v.
- For discrete r.v.'s $p(X) = \sum_y p(X, Y = y)$ and $p(Y) = \sum_x p(X = x, Y)$
- For disc … he … rows/co …



The definition also applied for two sets of r.v.'s and marginal of one set of r.v.'s is obtained by summing over all possibilities of the second set of r.v.'s

For discrete r.v.'s, marginalization is called summing over, for continuous r.v.'s, it is called "integrating out"

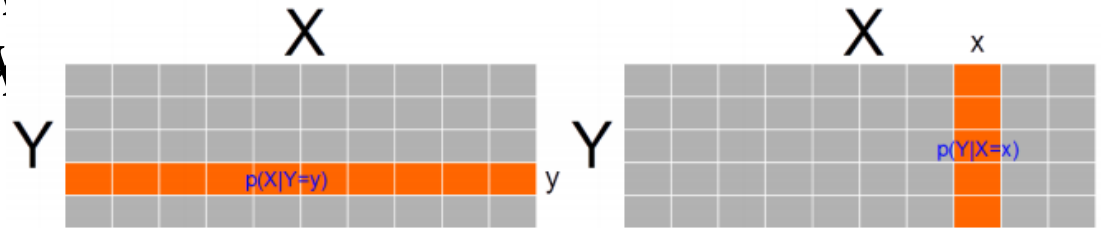


$$p(X) = \int_y p(X, Y = y)dy, \quad p(Y) = \int_x p(X = x, Y)dx$$

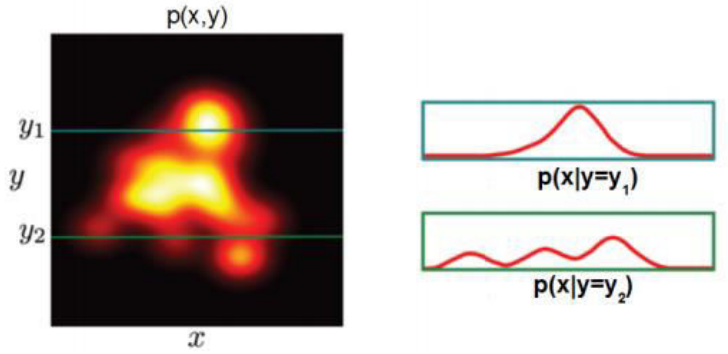
Conditional Probability Distribution

- Consider two r.v.'s X and Y (discrete/continuous – both need not of same type)
- Conditional PMF/PDF $p(X|Y)$ is the prob. dist. of one r.v. X, fixing other r.v. Y
- $p(X|Y = y)$ or $p(Y|X = x)$ like taking a slice: $p(X|Y = y)$

Discrete Random Variables



Continuous Random Variables



- Note: A conditional PMF/PDF may also be conditioned on multiple r.v.'s, e.g., $p(Y|X_1, X_2, X_3)$

We will see cond. dist. of output y given weights w(r.v.) and features X written as $p(y|w, X)$

An example

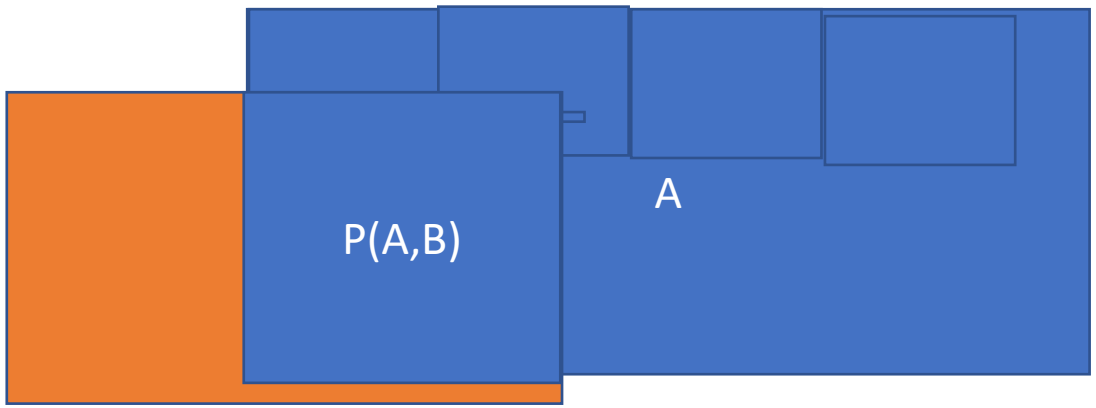
X/Y	raining	sunny
Umbrella	0.5	0.1
No umbrella	0.2	0.2

$P(x) = \{0.6, 0.4\}$
$P(y) = \{0.7, 0.3\}$

$P(X|Y = \text{raining}) = \{0.5, 0.2\}$

$P(X = \text{umbrella}|Y = \text{raining})$

$P(B|A)p(A) = p(A|B)p(B)$



Some Basic Rules

- **Sum Rule**: Gives the marginal probability distribution from joint probability distribution

For discrete r.v.: $p(X) = \sum_Y p(X, Y)$

For continuous r.v.: $p(X) = \int_Y p(X, Y) dY$

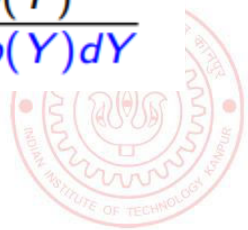
- **Product Rule**: $p(X, Y) = p(Y|X)p(X) = p(X|Y)p(Y)$

- **Bayes' rule**: Gives conditional probability distribution (can derive it from

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

For discrete r.v.: $p(Y|X) = \frac{p(X|Y)p(Y)}{\sum_Y p(X|Y)p(Y)}$

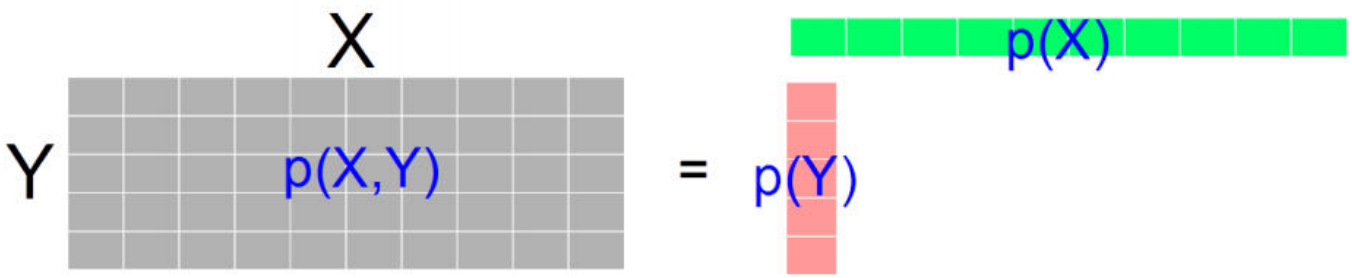
For continuous r.v.: $p(Y|X) = \frac{p(X|Y)p(Y)}{\int_Y p(X|Y)p(Y)dY}$



Independence

- X and Y are independent when knowing one tells nothing about the other

$$p(X|Y=y) = p(X)$$
$$p(Y|X=x) = p(Y)$$
$$p(X,Y) = p(X)p(Y)$$



- The above is the marginal independence $(X \perp\!\!\!\perp Y)$

- Two r.v.'s X and Y may not be marginally indep but may be given the value of another r.v. Z

$$p(X,Y|Z=z) = p(X|Z=z)p(Y|Z=z) \qquad X \perp\!\!\!\perp Y|Z$$

Expectation

- Expectation of a random variable tells the expected or average value it takes
- Expectation of a discrete random variable having PMF $p(X)$

$$\mathbb{E}[X] = \sum_{x \in S_X} x p(x)$$

Probability that $X = x$

- Expectation of a continuous random variable having PDF $p(X)$

$$\mathbb{E}[X] = \int_{x \in S_X} x p(x) dx$$

Probability density at $X = x$

Note that this exp. is w.r.t. the distribution $p(f(X))$ of the r.v. $f(X)$

Often the subscript is omitted but do keep in mind the underlying distribution

- The definition applies to functions of r.v. too (e.g., $\mathbb{E}[f(X)]$)

Expectation: A Few Rules

X and Y need not be even independent. Can be discrete or continuous

- Expectation of sum of two r.v.'s: $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
- Proof is as follows
  - Define $Z = X + Y$

$$\mathbb{E}[Z] = \sum_{z \in S_Z} z \cdot p(Z = z) \quad \text{s.t. } z = x + y \text{ where } x \in S_X \text{ and } y \in S_Y$$

$$= \sum_{x \in S_X} \sum_{y \in S_Y} (x + y) \cdot p(X = x, Y = y)$$

$$= \sum_x \sum_y x \cdot p(X = x, Y = y) + \sum_x \sum_y y \cdot p(X = x, Y = y)$$

$$= \sum_x x \sum_y p(X = x, Y = y) + \sum_y y \sum_x p(X =$$

Used the rule of marginalization of joint dist. of two r.v.'s

$$= \sum_x x \cdot p(X = x) + \sum_y y \cdot p(Y = y)$$

$$= \mathbb{E}[X] + \mathbb{E}[Y]$$

Expectation: A Few Rules (Contd)

- Expectation of a scaled r.v.: $\mathbb{E}[\alpha X] = \alpha\mathbb{E}[X]$
- Linearity of expectation: $\mathbb{E}[\alpha X + \beta Y] = \alpha\mathbb{E}[X] + \beta\mathbb{E}[Y]$
- (More General) Lin. of exp.: $\mathbb{E}[\alpha f(X) + \beta g(Y)] = \alpha\mathbb{E}[f(X)] + \beta\mathbb{E}[g(Y)]$
- Exp. of product of two independent r.v.'s: $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$
- Law of the Unconscious Statistician (LOTUS): Given an r.v. X with a known prob. dist. $p(X)$ and an[illegible]dom va[illegible] $g(X)$ for some function g

$$\mathbb{E}[Y] = \mathbb{E}[g(X)] = \sum_{y \in S_Y} y p(y) = \sum_{x \in S_X} g(x) p(x)$$

α is a real-valued scalar

α and β are real-valued scalars

f and g are arbitrary functions.

Requires finding $p(Y)$

Requires only $p(X)$ which we already have

LOTUS also applicable for continuous r.v.'s

- Rule of iterated expectation: $\mathbb{E}_{p(X)}[X] = \mathbb{E}_{p(Y)}[\mathbb{E}_{p(X|Y)}[X|Y]]$

Variance and Covariance

- Variance of a scalar r.v. tells us about its spread around its mean value $\mathbb{E}[X] = \mu$

$$\mathbf{var}[X] = \mathbb{E}[(X - \mu)^2] = \mathbb{E}[X^2] - \mu^2$$

- Standard deviation is simply the square root is variance
- For two scalar r.v.'s X and Y, the covariance is defined by

$$\mathbf{cov}[X, Y] = \mathbb{E}[\{X - \mathbb{E}[X]\}\{Y - \mathbb{E}[Y]\}] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$$

- For two vector r.v.'s X and Y (assume column vec), the covariance matrix is defined by

$$\mathbf{cov}[X, Y] = \mathbb{E}[\{X - \mathbb{E}[X]\}\{Y^\top - \mathbb{E}[Y^\top]\}] = \mathbb{E}[XY^\top] - \mathbb{E}[X]\mathbb{E}[Y^\top]$$

- Cov. of components of a vector r.v. X: $\mathbf{cov}[X] = \mathbf{cov}[X, X]$
- Note: The definitions apply to functions of r.v. too (e.g., $\mathbf{var}[f(X)]$)

Important result

Transformation of Random Variables

- Suppose $Y = f(X) = AX + b$ be a linear function of a vector-valued r.v. X (A is a matrix and b is a vector, both constants)
- Suppose $\mathbb{E}[X] = \mu$ and $\mathbf{cov}[X] = \Sigma$, then for the vector-valued r.v. Y

$$\mathbb{E}[Y] = \mathbb{E}[AX + b] = A\mu + b$$

$$\mathbf{cov}[Y] = \mathbf{cov}[AX + b] = A\Sigma A^{\top}$$

- Likewise, if $Y = f(X) = a^{\top}X + b$ be a linear function of a vector-valued r.v. X (a is a vector and b is a scalar, both constants)
- Suppose $\mathbb{E}[X] = \mu$ and $\mathbf{cov}[X] = \Sigma$, then for the scalar-valued r.v. Y

$$\mathbb{E}[Y] = \mathbb{E}[a^{\top}X + b] = a^{\top}\mu + b$$

$$\mathbf{var}[Y] = \mathbf{var}[a^{\top}X + b] = a^{\top}\Sigma a$$

Common Probability Distributions

Important: We will use these extensively to model data as well as parameters of models

- Some common discrete distributions and what they can model
  - **Bernoulli**: Binary numbers, e.g., outcome (head/tail, 0/1) of a coin toss
  - **Binomial**: Bounded non-negative integers, e.g., # of heads in n coin tosses
  - **Multinomial/multinoulli**: One of K (>2) possibilities, e.g., outcome of a dice roll
  - **Poisson**: Non-negative integers, e.g., # of words in a document
- Some common continuous distributions and what they can model
  - **Uniform**: numbers defined over a fixed range
  - **Beta**: numbers between 0 and 1, e.g., probability of head for a biased coin
  - **Gamma**: Positive unbounded real numbers
  - **Dirichlet**: vectors that sum of 1 (fraction of data points in different clusters)
  - **Gaussian**: real-valued numbers or real-valued vectors

Expectations and parameter estimation

CS771: Introduction to Machine Learning

Nisheeth

Expectation

- Expectation of a random variable tells the expected or average value it takes
- Expectation of a discrete random variable having PMF $p(X)$

$$\mathbb{E}[X] = \sum_{x \in S_X} xp(x)$$

Probability that $X = x$

- Expectation of a continuous random variable having PDF $p(X)$

$$\mathbb{E}[X] = \int_{x \in S_X} xp(x)dx$$

Probability density at $X = x$

Note that this exp. is w.r.t. the distribution $p(f(X))$ of the r.v. $f(X)$

Often the subscript is omitted but do keep in mind the underlying distribution

- The definition applies to functions of r.v. too (e.g., $\mathbb{E}[f(X)]$)

Expectation: intuition

- Recall the following probability distribution of patient arrivals:

x	10	11	12	13	14
P(x)	.4	.2	.2	.1	.1

$$\sum_{i=1}^{5} x_i\, p(x) = 10(.4) + 11(.2) + 12(.2) + 13(.1) + 14(.1) = 11.3$$

Non-trivial discrete probabilities

Take the example of 5 coin tosses. What's the probability that you flip exactly 3 heads in 5 coin tosses?

A discrete distribution: binomial

- A fixed number of observations (trials), n
  - e.g., 15 tosses of a coin; 20 patients; 1000 people surveyed
- A binary outcome
  - e.g., head or tail in each toss of a coin; disease or no disease
  - Generally called "success" and "failure"
  - Probability of success is p, probability of failure is 1 – p
- Constant probability for each observation
  - e.g., Probability of getting a tail is the same each time we toss the coin

Binomial distribution

Solution:

One way to get exactly 3 heads: HHHTT

What's the probability of this exact arrangement?

P(heads)xP(heads) xP(heads)xP(tails)xP(tails) $=(1/2)^3 \; x \; (1/2)^2$

Another way to get exactly 3 heads: THHHT

Probability of this exact outcome = $(1/2)^1 \; x \; (1/2)^3 \; x \; (1/2)^1 = (1/2)^3 \; x \; (1/2)^2$

Binomial distribution

In fact, $(1/2)^3 \; x \; (1/2)^2$ is the probability of each unique outcome that has exactly 3 heads and 2 tails.

So, the overall probability of 3 heads and 2 tails is:

$(1/2)^3 \; x \; (1/2)^2 \; + (1/2)^3 \; x \; (1/2)^2 + (1/2)^3 \; x \; (1/2)^2 \; +$ ….. for as many unique arrangements as there are—but how many are there??

$\binom{5}{3}$ ways to arrange 3 heads in 5 trials

$_5C_3 = 5!/3!2! = 10$

Outcome	Probability
THHHT	$(1/2)^3 \; x \; (1/2)^2$
HHHTT	$(1/2)^3 \; x \; (1/2)^2$
TTHHH	$(1/2)^3 \; x \; (1/2)^2$
HTTHH	$(1/2)^3 \; x \; (1/2)^2$
HHTTH	$(1/2)^3 \; x \; (1/2)^2$
HTHHT	$(1/2)^3 \; x \; (1/2)^2$
THTHH	$(1/2)^3 \; x \; (1/2)^2$
HTHTH	$(1/2)^3 \; x \; (1/2)^2$
HHTHT	$(1/2)^3 \; x \; (1/2)^2$
THHTH	$(1/2)^3 \; x \; (1/2)^2$
10 arrangements	$x \; (1/2)^3 \; x \; (1/2)^2$

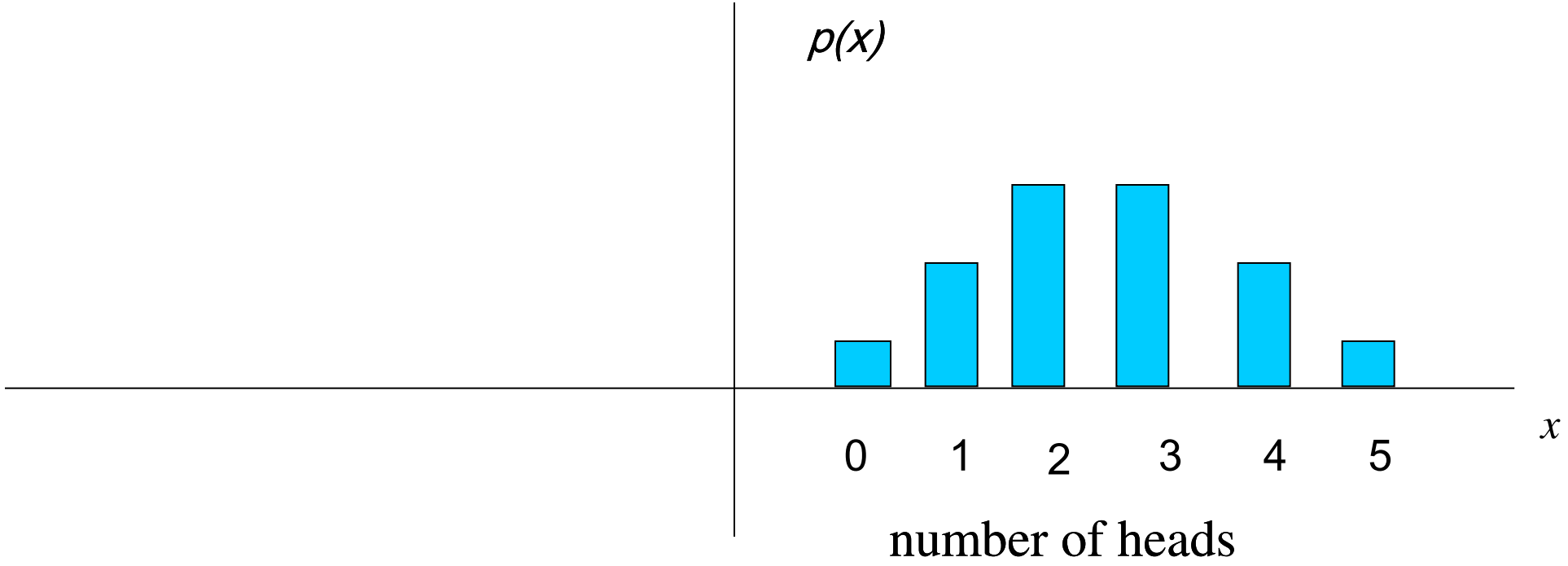
The probability of each unique outcome (note: they are all equal)

$\therefore$ P(3 heads and 2 tails) = $\binom{5}{3}$ x P(heads)^3 x P(tails)^2 =

$10 \times (1/2)^5 = 31.25\%$

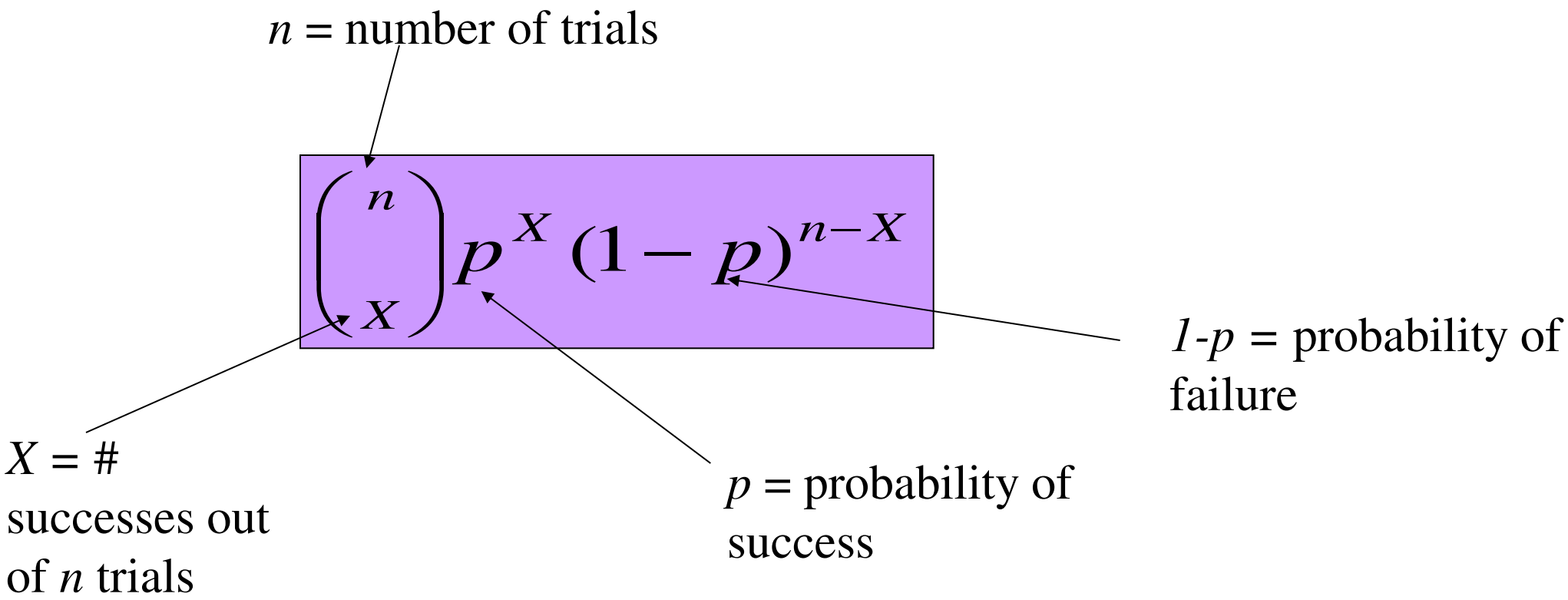
Binomial distribution function:

X= the number of heads tossed in 5 coin tosses



Binomial distribution, generally

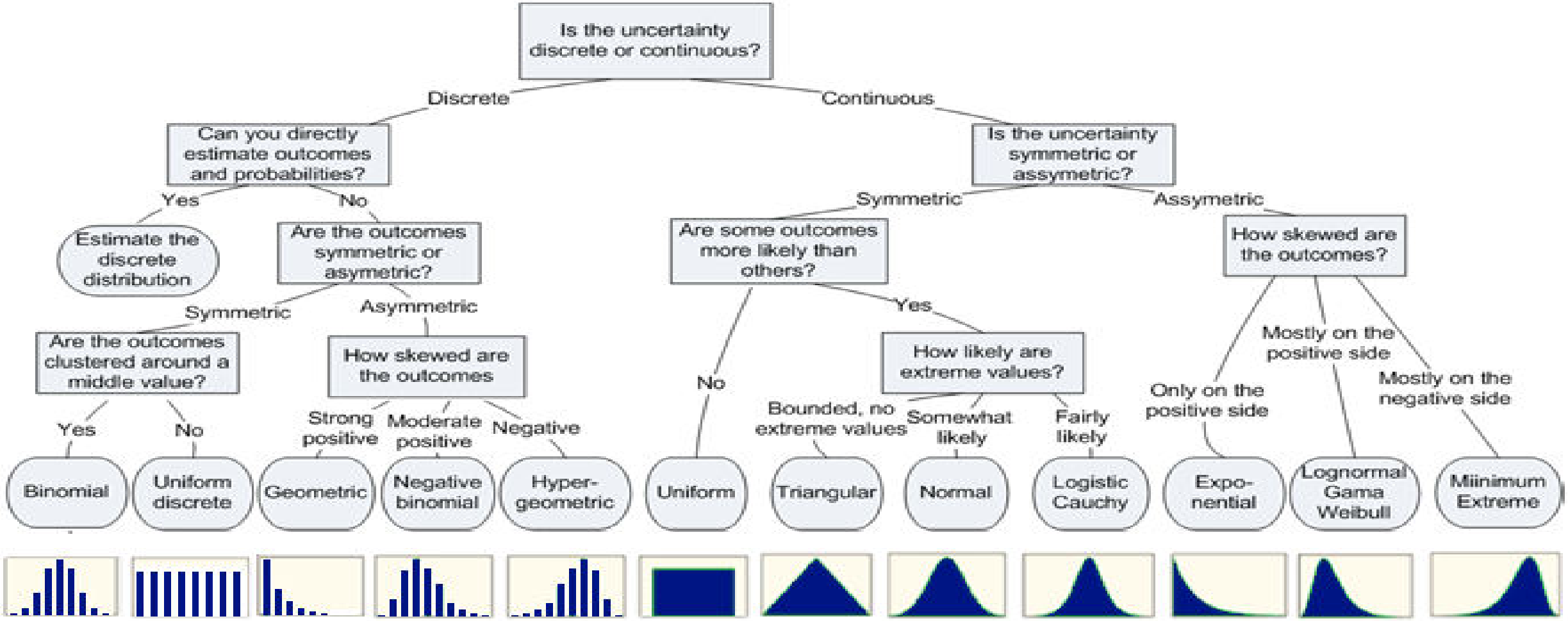
Note the general pattern emerging → if you have only two possible outcomes (call them 1/0 or yes/no or success/failure) in n independent trials, then the probability of exactly X "successes"=



Common Probability Distributions

Important: We will use these extensively to model data as well as parameters of models

- Some common discrete distributions and what they can model
  - **Bernoulli**: Binary numbers, e.g., outcome (head/tail, 0/1) of a coin toss
  - **Binomial**: Bounded non-negative integers, e.g., # of heads in n coin tosses
  - **Multinomial/multinoulli**: One of K (>2) possibilities, e.g., outcome of a dice roll
  - **Poisson**: Non-negative integers, e.g., # of words in a document
- Some common continuous distributions and what they can model
  - **Uniform**: numbers defined over a fixed range
  - **Beta**: numbers between 0 and 1, e.g., probability of head for a biased coin
  - **Gamma**: Positive unbounded real numbers
  - **Dirichlet**: vectors that sum of 1 (fraction of data points in different clusters)
  - **Gaussian**: real-valued numbers or real-valued vectors


Is the uncertainty discrete or continuous?
Discrete
Continuous
Can you directly estimate outcomes and probabilities?
Is the uncertainty symmetric or assymetric?
Yes
No
Symmetric
Assymetric
Estimate the discrete distribution
Are the outcomes symmetric or asymetric?
Are some outcomes more likely than others?
How skewed are the outcomes?
Symmetric
Asymmetric
Yes
No
Are the outcomes clustered around a middle value?
How skewed are the outcomes
How likely are extreme values?
Only on the positive side
Mostly on the positive side
Mostly on the negative side
Yes
No
Strong positive
Moderate positive
Negative
Bounded, no extreme values
Somewhat likely
Fairly likely
Binomial
Uniform discrete
Geometric
Negative binomial
Hyper-geometric
Uniform
Triangular
Normal
Logistic Cauchy
Expo-nential
Lognormal Gama Weibull
Miinimum Extreme

Expectation: A Few Rules

X and Y need not be even independent. Can be discrete or continuous

- Expectation of sum of two r.v.'s: $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
- Proof is as follows
  - Define $Z = X + Y$

$$\mathbb{E}[Z] = \sum_{z \in S_Z} z \cdot p(Z = z) \quad \text{s.t. } z = x + y \text{ where } x \in S_X \text{ and } y \in S_Y$$

$$= \sum_{x \in S_X} \sum_{y \in S_Y} (x + y) \cdot p(X = x, Y = y)$$

$$= \sum_x \sum_y x \cdot p(X = x, Y = y) + \sum_x \sum_y y \cdot p(X = x, Y = y)$$

$$= \sum_x x \sum_y p(X = x, Y = y) + \sum_y y \sum_x p(X =$$

Used the rule of marginalization of joint dist. of two r.v.'s

$$= \sum_x x \cdot p(X = x) + \sum_y y \cdot p(Y = y)$$

$$= \mathbb{E}[X] + \mathbb{E}[Y]$$

Expectation: A Few Rules (Contd)

- Expectation of a scaled r.v.: $\mathbb{E}[\alpha X] = \alpha\mathbb{E}[X]$
- Linearity of expectation: $\mathbb{E}[\alpha X + \beta Y] = \alpha\mathbb{E}[X] + \beta\mathbb{E}[Y]$
- (More General) Lin. of exp.: $\mathbb{E}[\alpha f(X) + \beta g(Y)] = \alpha\mathbb{E}[f(X)] + \beta\mathbb{E}[g(Y)]$
- Exp. of product of two independent r.v.'s: $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$
- Law of the Unconscious Statistician (LOTUS): Given an r.v. X with a known prob. dist. $p(X)$ and an[illegible]dom va[illegible] $g(X)$ for some function g

$$\mathbb{E}[Y] = \mathbb{E}[g(X)] = \sum_{y \in S_Y} y p(y) = \sum_{x \in S_X} g(x) p(x)$$

- Rule of iterated expectation: $\mathbb{E}_{p(X)}[X] = \mathbb{E}_{p(Y)}[\mathbb{E}_{p(X|Y)}[X|Y]]$

Variance and Covariance

- Variance of a scalar r.v. tells us about its spread around its mean value $\mathbb{E}[X] = \mu$

$$\mathbf{var}[X] = \mathbb{E}[(X - \mu)^2] = \mathbb{E}[X^2] - \mu^2$$

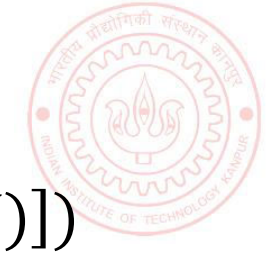
- Standard deviation is simply the square root is variance
- For two scalar r.v.'s X and Y, the covariance is defined by

$$\mathbf{cov}[X, Y] = \mathbb{E}[\{X - \mathbb{E}[X]\}\{Y - \mathbb{E}[Y]\}] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$$

- For two vector r.v.'s X and Y (assume column vec), the covariance matrix is defined by

$$\mathbf{cov}[X, Y] = \mathbb{E}[\{X - \mathbb{E}[X]\}\{Y^\top - \mathbb{E}[Y^\top]\}] = \mathbb{E}[XY^\top] - \mathbb{E}[X]\mathbb{E}[Y^\top]$$

- Cov. of components of a vector r.v. X: $\mathbf{cov}[X] = \mathbf{cov}[X, X]$
- Note: The definitions apply to functions of r.v. too (e.g., $\mathbf{var}[f(X)]$)
- Note: Variance of sum of independent r.v.'s: $\mathbf{var}[X + Y] = \mathbf{var}[X]$



Transformation of Random Variables

- Suppose $Y = f(X) = AX + b$ be a linear function of a vector-valued r.v. X (A is a matrix and b is a vector, both constants)
- Suppose $\mathbb{E}[X] = \mu$ and $\mathbf{cov}[X] = \Sigma$, then for the vector-valued r.v. Y

$$\mathbb{E}[Y] = \mathbb{E}[AX + b] = A\mu + b$$

$$\mathbf{cov}[Y] = \mathbf{cov}[AX + b] = A\Sigma A^{\top}$$

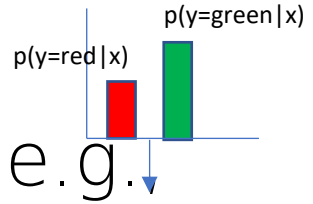
- Likewise, if $Y = f(X) = a^{\top}X + b$ be a linear function of a vector-valued r.v. X (a is a vector and b is a scalar, both constants)
- Suppose $\mathbb{E}[X] = \mu$ and $\mathbf{cov}[X] = \Sigma$, then for the scalar-valued r.v. Y

$$\mathbb{E}[Y] = \mathbb{E}[a^{\top}X + b] = a^{\top}\mu + b$$

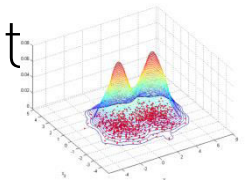
$$\mathbf{var}[Y] = \mathbf{var}[a^{\top}X + b] = a^{\top}\Sigma a$$

Probabilistic ML: Some Motivation

- In many ML problems, we want to model and reason about data probabilistically



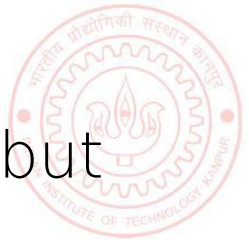
- At a high-level, this is the density estimation view of ML, e.g.,
  - Given input-output pairs $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$ estimat conditional $p(y|\boldsymbol{x})$



  - Given inputs $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, estimate the distribution $p(\boldsymbol{x})$ of the inputs
  - Note 1: These dist. will depend on some parameters θ (to be estimated), and written as

$$p(y|\boldsymbol{x}, \theta) \quad \text{or} \quad p(\boldsymbol{x}|\theta)$$

  - Note 2: These dist. sometimes assumed to have a specific form, but sometimes not



Probabilistic Modeling: The Basic Idea

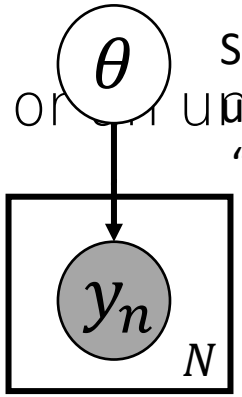
- Assume N observations $\boldsymbol{y} = \{y_1, y_2, \ldots, y_N\}$, generated from a presumed prob. model

$$y_n \sim p(y|\theta) \qquad \forall n$$

(assumed independently & identically distributed)

- Here $p(y|\theta)$ is a conditional distribution, conditioned on params θ (to be le
  - N
  - Po

The parameters may themselves depend on other unknown/known parameters (called hyperparameters), which may depend on other unknowns, and so on. ☺ This is essentially "hierarchical" modeling (will see various examples later)

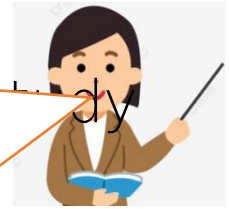


Such diagrams are usually called the "plate notation"

known or an unknown and do

ady

The Predictive dist. tells us how likely each possible value of a new observation y_* is. Example: if y_* denotes the outcome of a coin toss, then what is $p(y_* = "head"|\boldsymbol{y})$, given N previous coin tosses $\boldsymbol{y} = \{y_1, y_2, \ldots, y_N\}$



- Some of the tasks that we may be interested in
  - Parameter estimation: Estimating the unknown parameters θ (and other

Parameter Estimation in Probabilistic Models

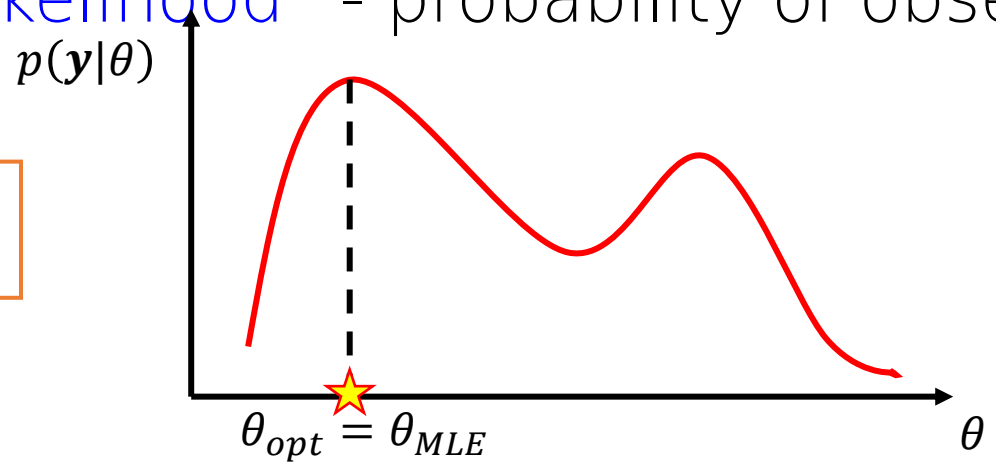
- Since data is assumed to be i.i.d., we can write down its total probability as

$$p(\boldsymbol{y}|\theta) = p(y_1, y_2, \ldots, y_N|\theta) = \prod_{n=1}^{N} p(y_n|\theta)$$

This now is an optimization problem essentially (θ being the unknown)

- $p(\boldsymbol{y}|\theta)$ called "likelihood" - probability of observed data as a function of params θ

How do I find the best θ ?

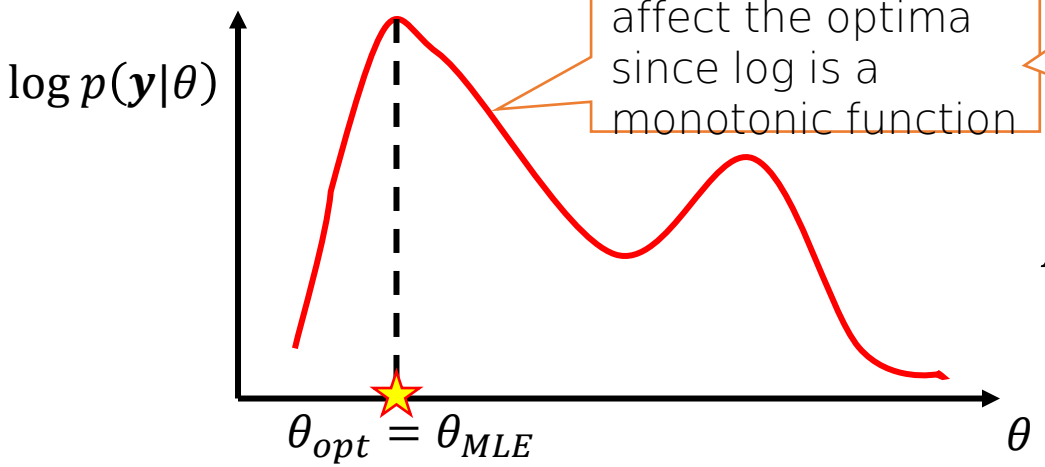


Well, one option is to find the θ that maximizes the likelihood (probability of the observed data) – basically, which value of θ makes the observed data most likely to have come from the assumed distribution $p(\boldsymbol{y}|\theta)$ --- Maximum Likelihood Estimation (MLE)

- In parameter estimation, the goal is to find the "best" θ, given observed data $\boldsymbol{y}$

Maximum Likelihood Estimation (MLE)

- The goal in MLE is to find the optimal θ by maximizing the likelihood
- In practice, we maximize the log of the likelihood (log-likelihood in short)



Taking log doesn't affect the optima since log is a monotonic function

Leads to simpler algebra/calculus, and also yields better numerical stability when implementing it on computer (dealing with log of probabilities)

$$LL(\theta) = \log p(\boldsymbol{y}|\theta) = \log \prod_{n=1}^{N} p(y_n|\theta)$$

$$= \sum_{n=1}^{N} \log p(y_n|\theta)$$

- Thus the MLE problem is

$$\theta_{MLE} = \operatorname{argmax}_{\theta} LL(\theta) = \operatorname{argmax}_{\theta} \sum_{n=1}^{N} \log p(y_n|\theta)$$

- This is now an optimization (maximization problem). Note: θ may

Maximum Likelihood Estimation (MLE)

Negative Log-Likelihood (NLL)

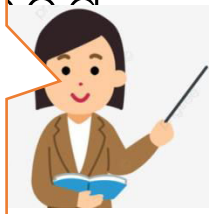
- The MLE problem can also be easily written as a minimization problem

$$\theta_{MLE} = \operatorname{argmax}_{\theta} \sum_{n=1}^{N} \log p(y_n|\theta) = \operatorname{argmin}_{\theta} \sum_{n=1}^{N} -\log p(y_n|\theta)$$

- Thus MLE can also be seen as minimizing the negative log-likelihood (NLL)

$$\theta_{MLE} = \operatorname{argmin}_{\theta} NLL(\theta)$$

Indeed. It may overfit. Several ways to prevent it: Use regularizer or other strategies to prevent overfitting. Alternatives, use "prior" distributions on the parameters θ that we are trying to estimate (which will kind of act as a regularizer as we will see shortly)



Such priors have various other benefits as we will see later

- NLL is analogous to a loss function
  - The negative log-lik $(-\log p(y_n|\theta))$ is akin to the

Does it mean MLE could overfit? If so, how to prevent this?

- Thus doing MLE is akin to minimizing training loss

MLE: An Example

- Consider a sequence of N coin toss outcomes (observations)
- Each observation y_n is a binary random variable. Head: $y_n = 1$, Tail: $y_n = 0$
- Each y_n is assumed generated by a Bernoulli distribution with param $\theta \in (0,1)$ $\quad p(y_n|\theta) = \text{Bernoulli}(y_n|\theta) = \theta^{y_n}(1-\theta)^{1-y_n}$
- Here θ the unknown param (probability of head). Want to estimate it using MLE
- Log-likelihood: $\sum_{n=1}^{N} \log p(y_n|\theta) = \sum_{n=1}^{N} [y_n \log \theta + (1-y_n)\log(1-\theta)]$
- Minimizing NLL w.r.t. θ gives $\theta_{MLE} = \frac{\sum_{n=1}^{N} y_n}{N}$

Probability of a head

Take deriv. set it to zero and solve. Easy optimization

I tossed a coin 5 times – gave 1 head and 4 tails. Does it means $\theta = 0.2$?? The MLE approach says so. What is I see 0 head and 5 tails. Does it mean $\theta = 0$?

Thus MLE solution is simply the fraction of heads! Makes intuitive sense!

Indeed – if you want to trust MLE solution. But with small number of training observations, MLE may overfit and may not be reliable. We will soon see better alternatives that use