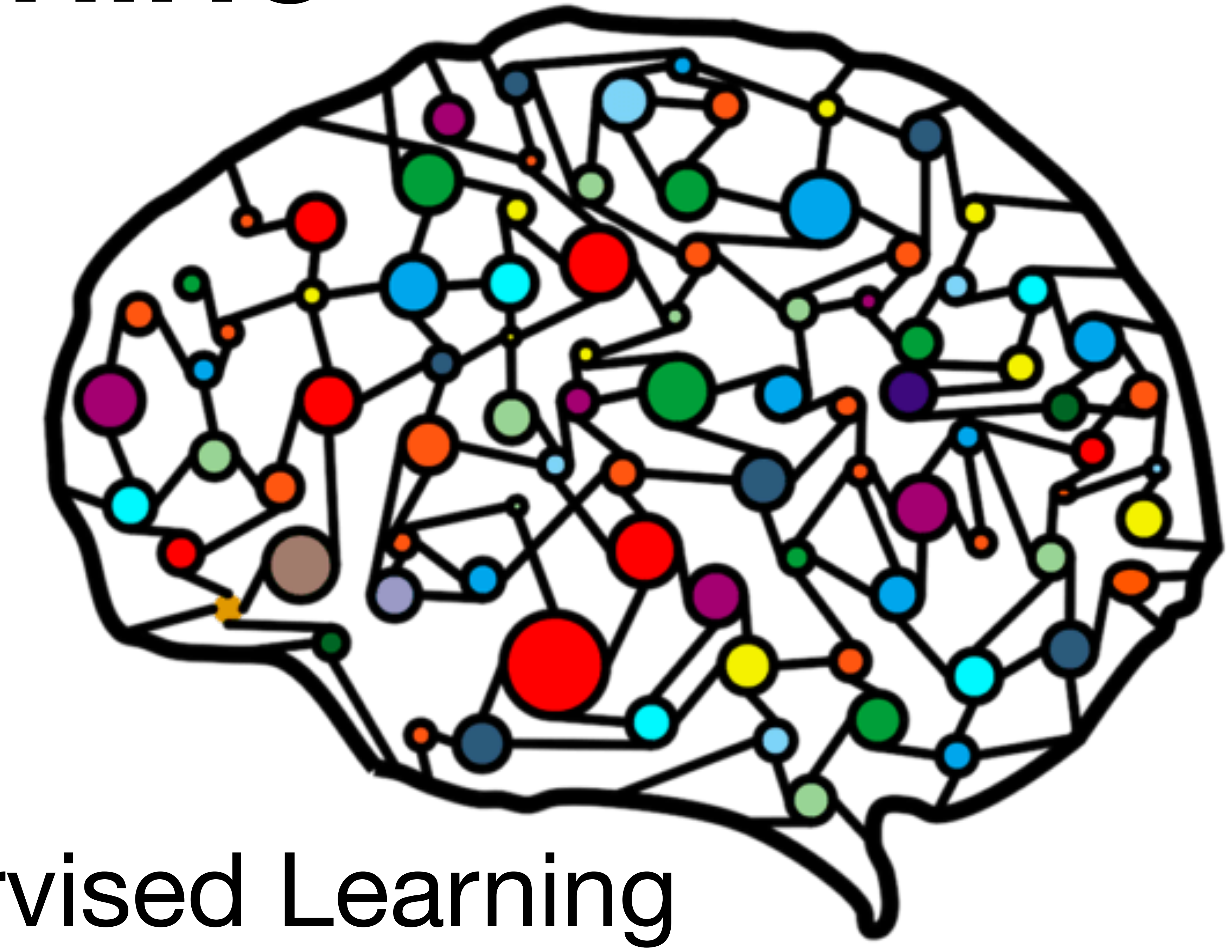
General Principles of Human and Machine Learning



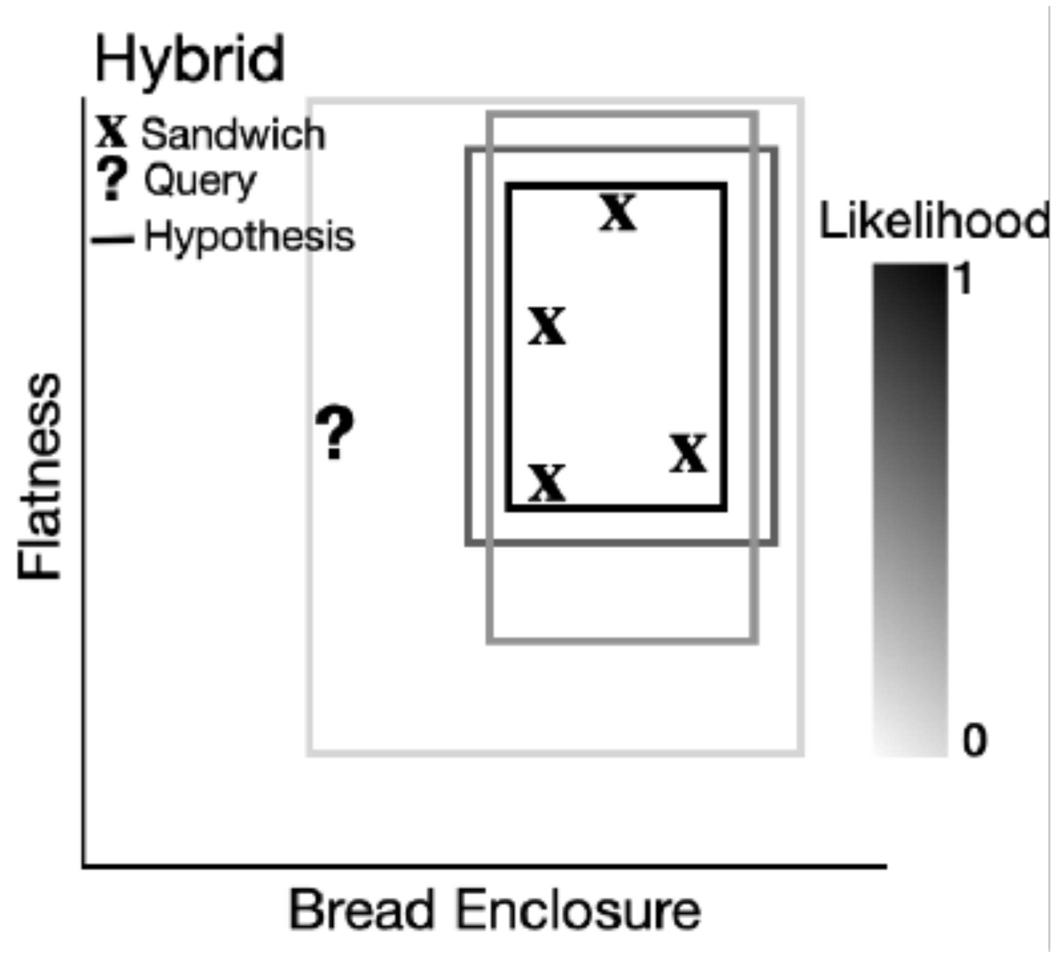
Lecture 7: Supervised and Unsupervised Learning

Dr. Charley Wu

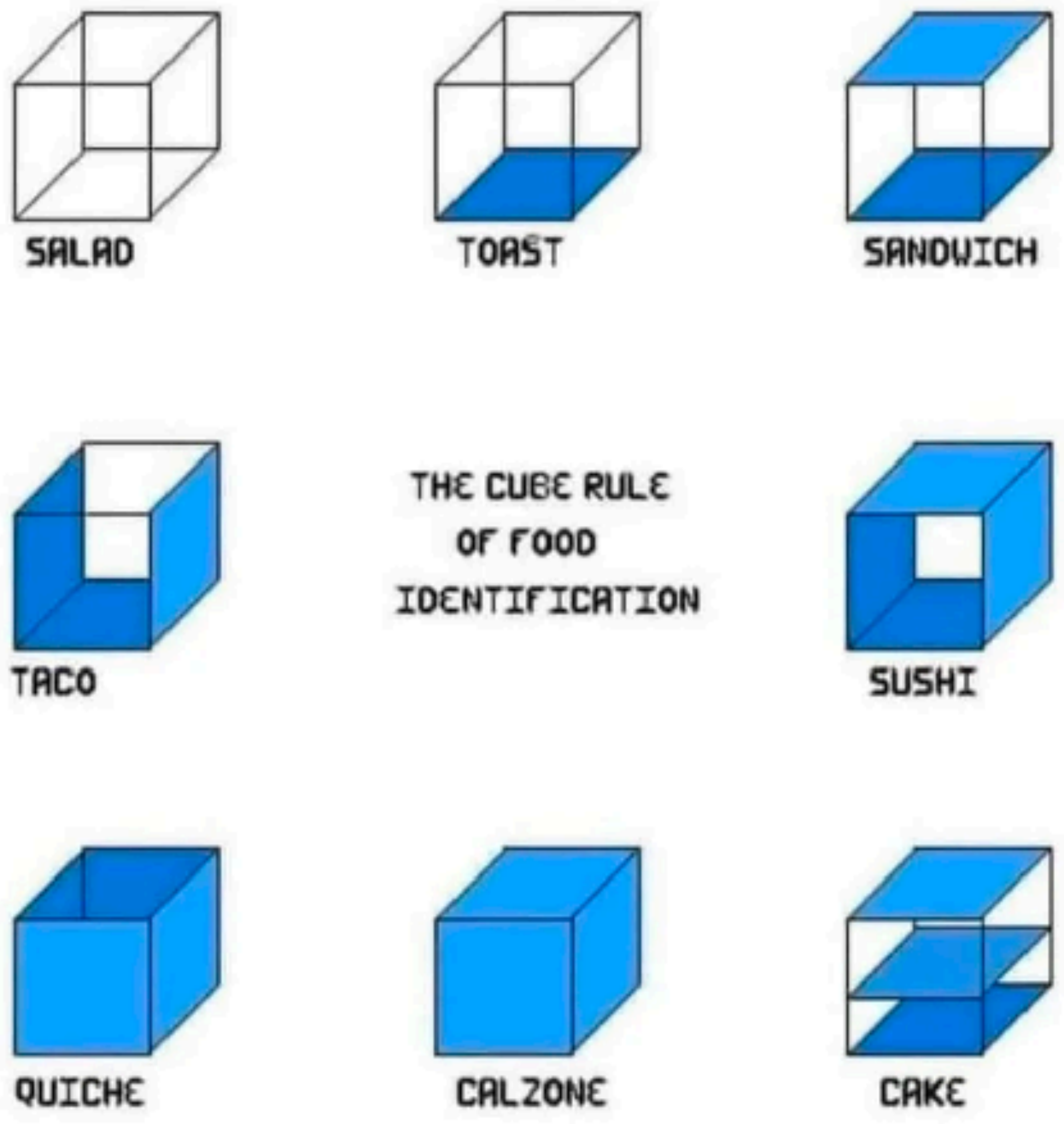
https://hmc-lab.com/GPHML.html

Last week …

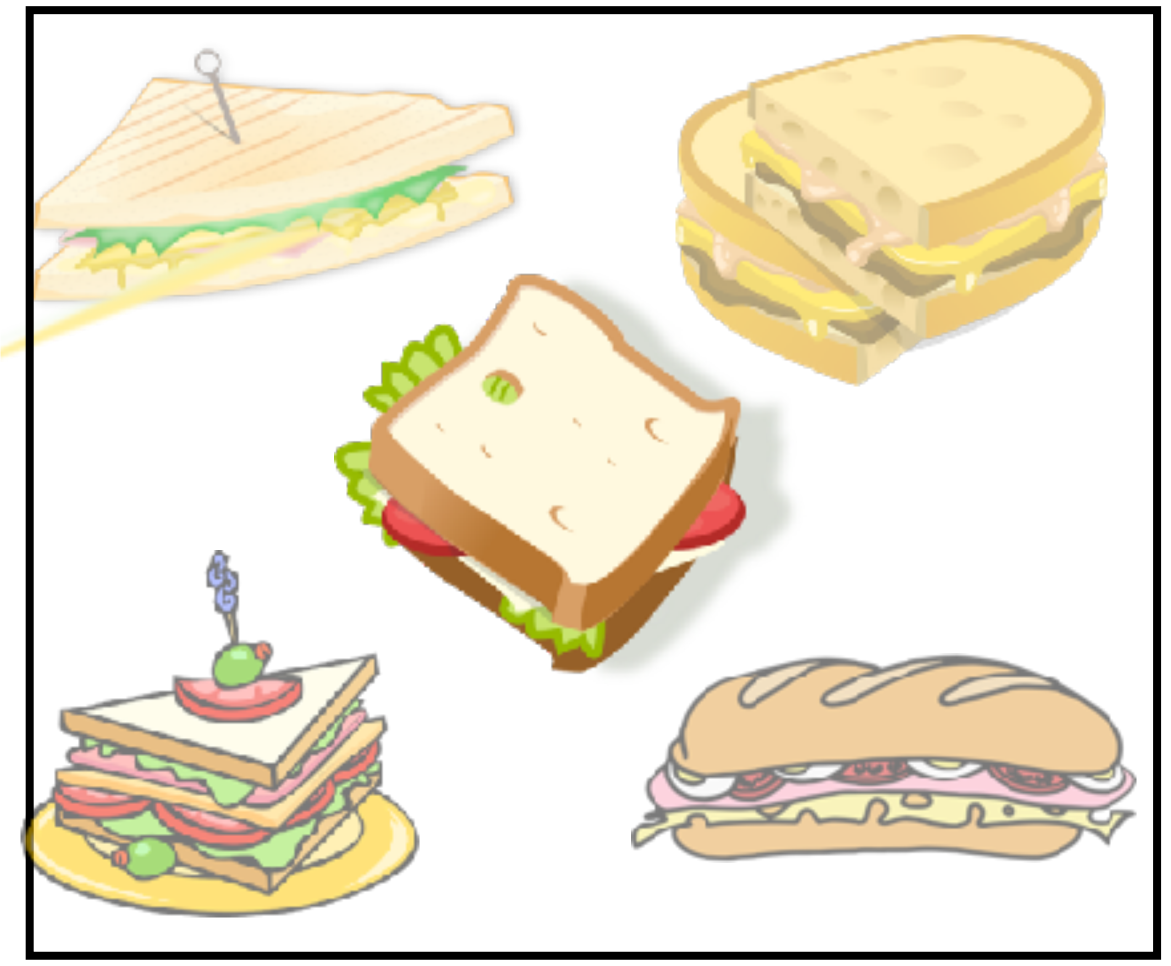
- Concepts are mental representations of categories in the world
- Classical view used **rules** to describe the necessary and sufficient conditions for category membership
- More psychological approaches used **similarity**, compared to a learned *prototypes* or past *exemplars*
- Bayesian concept learning is a **hybrid** approach, that uses distributions over rules, and recreating patterns consistent with similarity-based approaches



Rules



Similarity



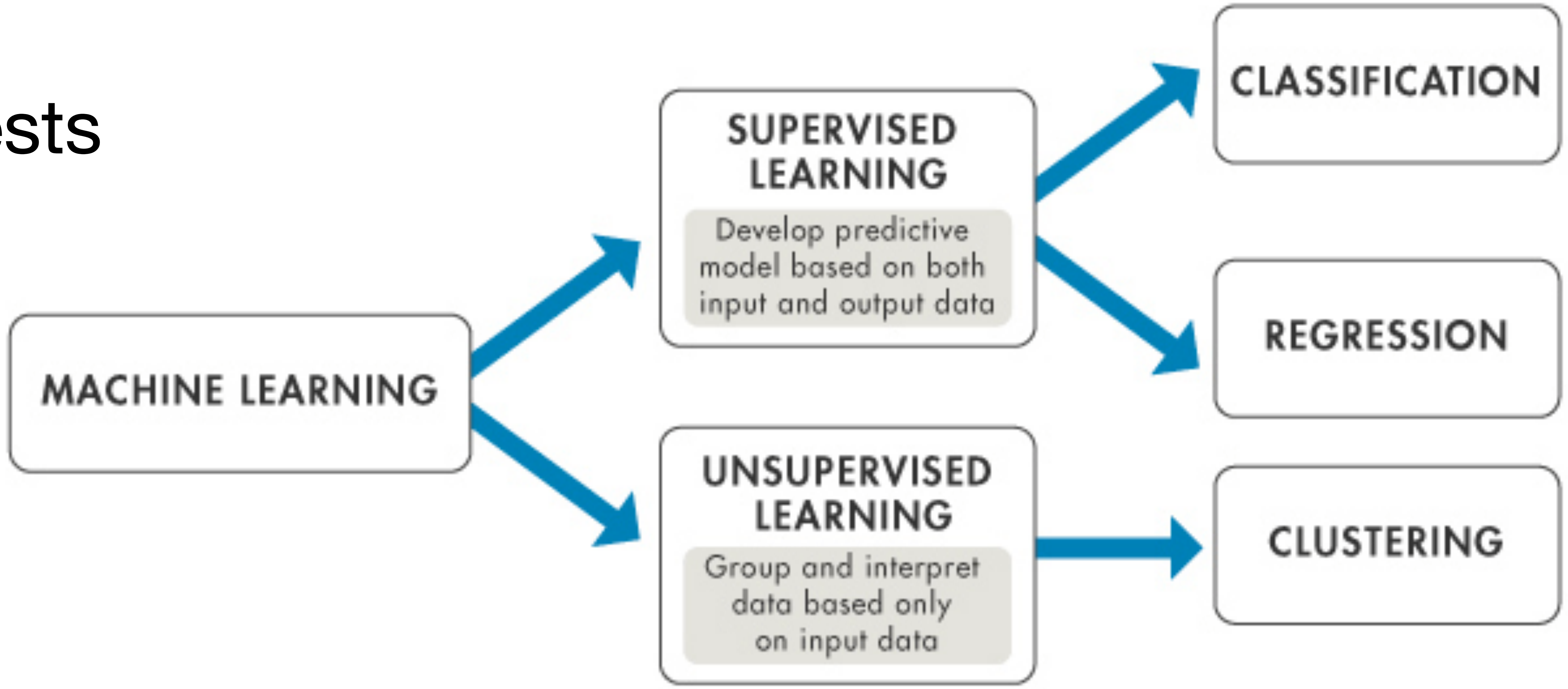
Today's agenda

Supervised learning (for classification)

- Multilayer Perceptrons
- Decision trees and random forests
- Support vector machines
- Naïve Bayes

Unsupervised Learning

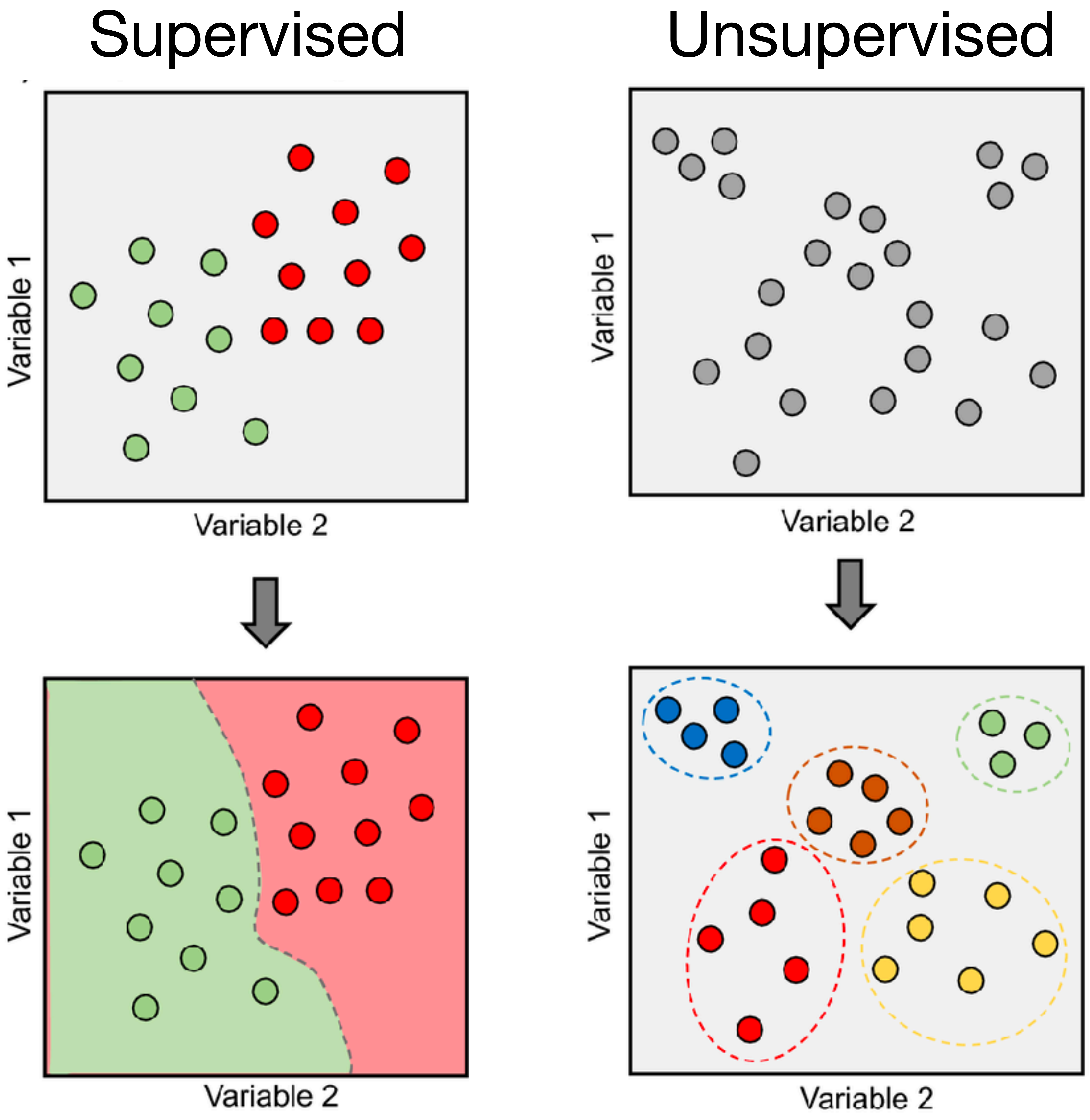
- k-Means
- Gaussian Mixture models



Supervised vs. unsupervised learning

- Classification problems*: classify data points into one of *n* different categories
- **Supervised** learning:
  - Training data provides category labels
  - Classifiers usually try to learn a decision-boundary
- **Unsupervised** learning:
  - Training data lacks category labels
  - Classifiers usually try to learn clusters

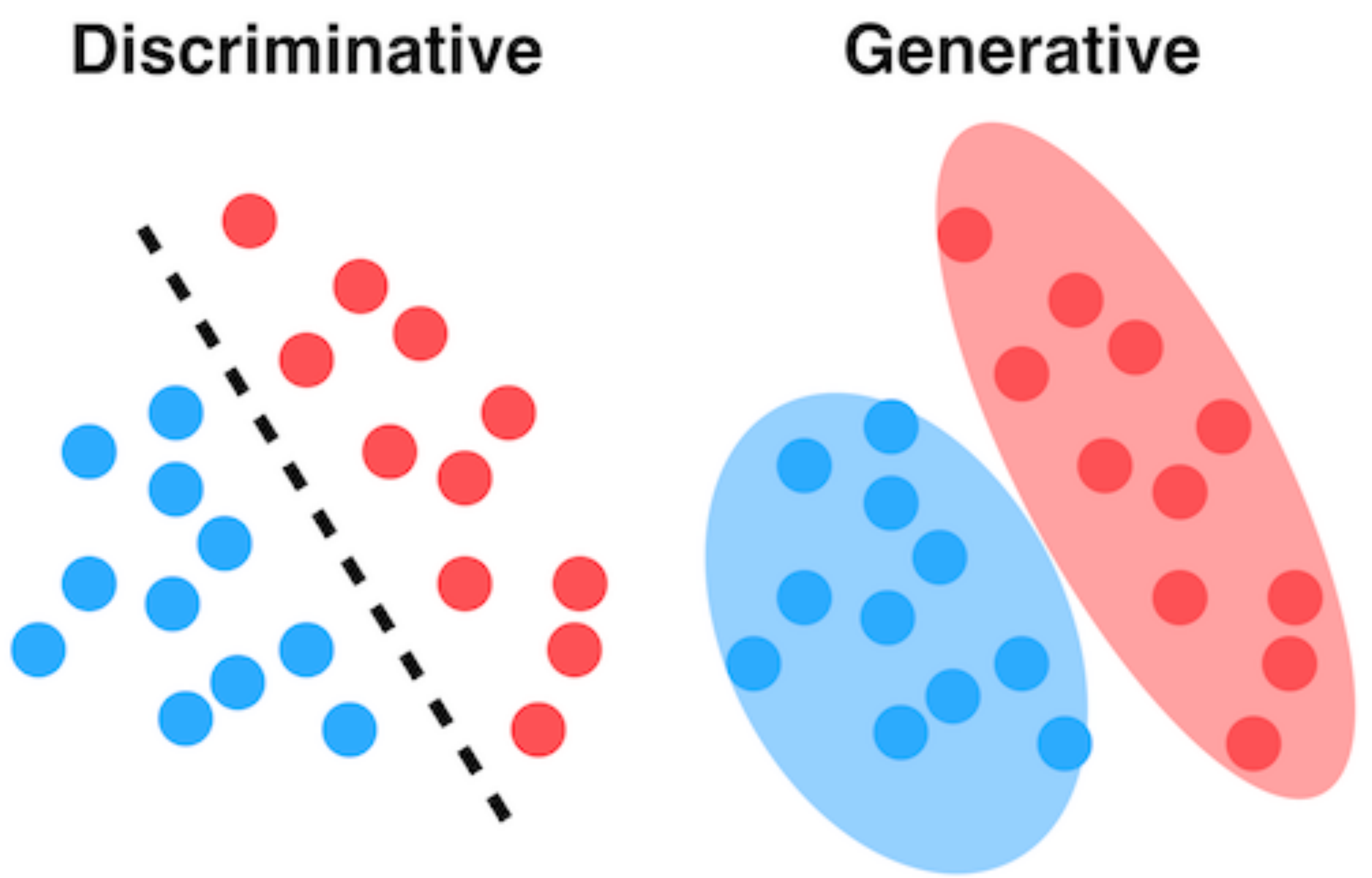
*Note that *regression* is another class of ML problems, which we will discuss next week

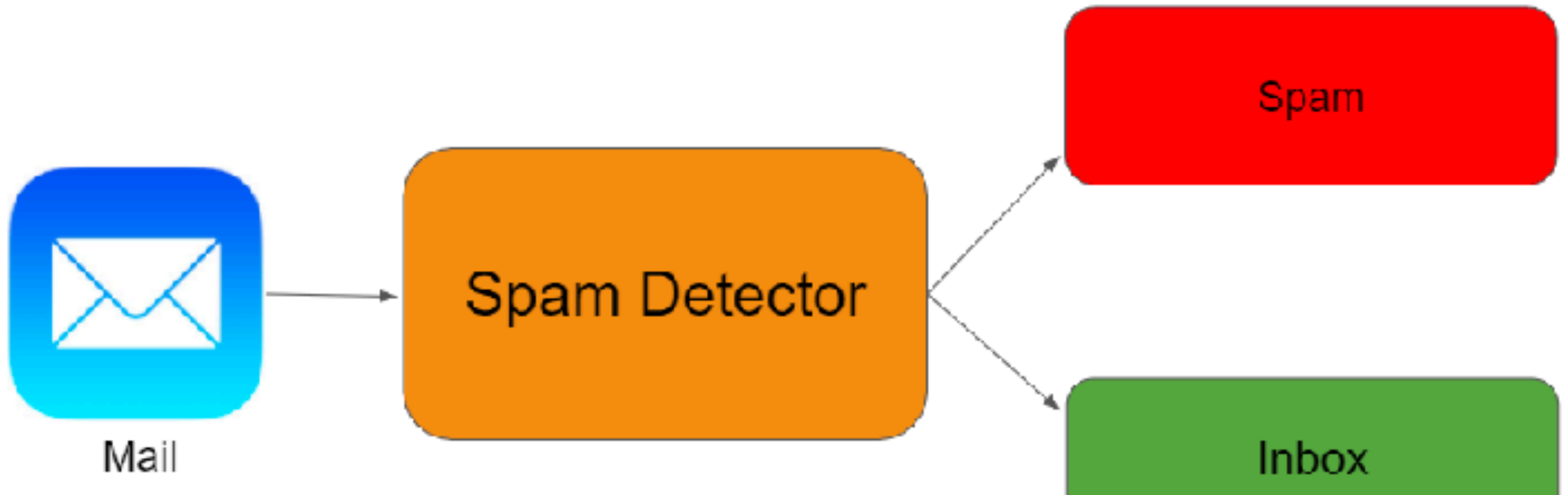


Supervised learning

Notation:
a scalar $\mathbf{a}$ vector $\mathscr{A}$ set
A constant $\mathbf{A}$ Matrix

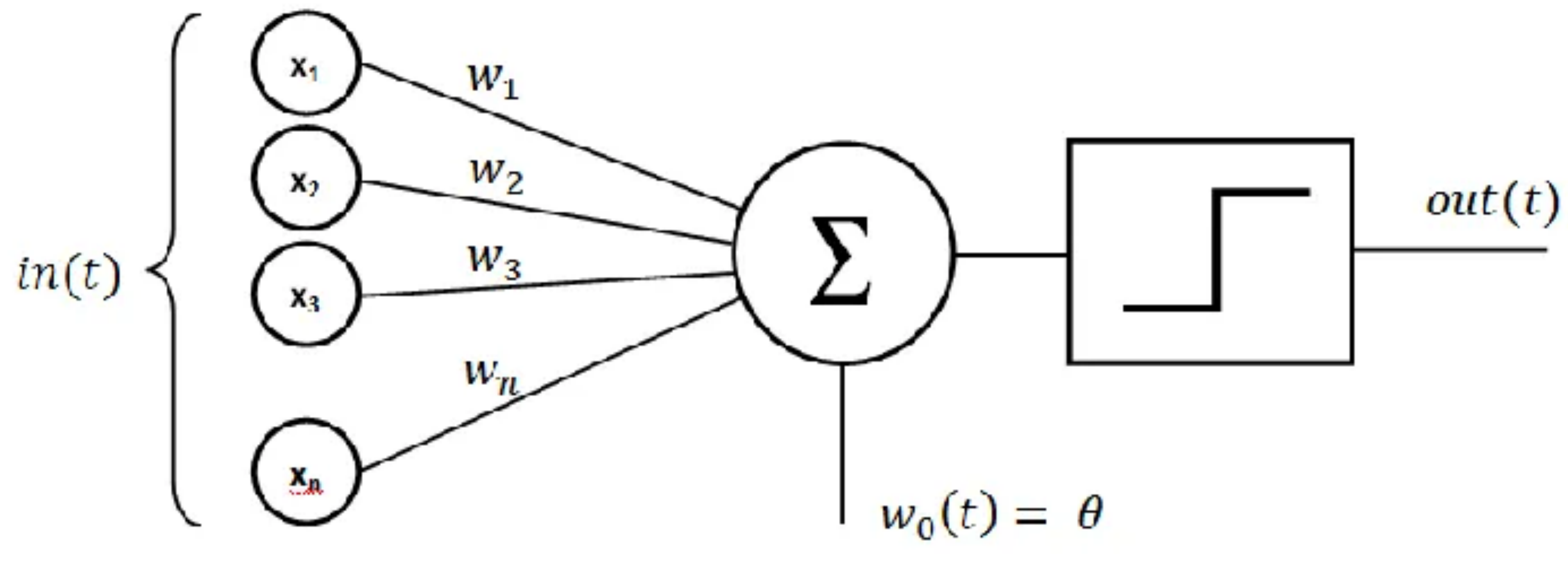
- Two general classes:
  - **Discriminitive** directly map features to class labels, often by learning a decision-boundary (rule-like)
  - **Generative** approaches learn the probability distribution of the data (similarity-like)
- Example problem: Spam detector
  - Data $\mathscr{D} = \{\mathbf{X}, \mathbf{y}\}$
  - each $\mathbf{x} \in \mathbf{X}$ are the features of an email (e.g., length, date, sender, content, etc…)
  - each $y \in \mathbf{y}$ is the label (1 if spam, 0 otherwise)
  - multiple labels are also possible:
    - Some classifiers inherently handle *n* classes
    - Some train multiple classifiers for each one vs all setting

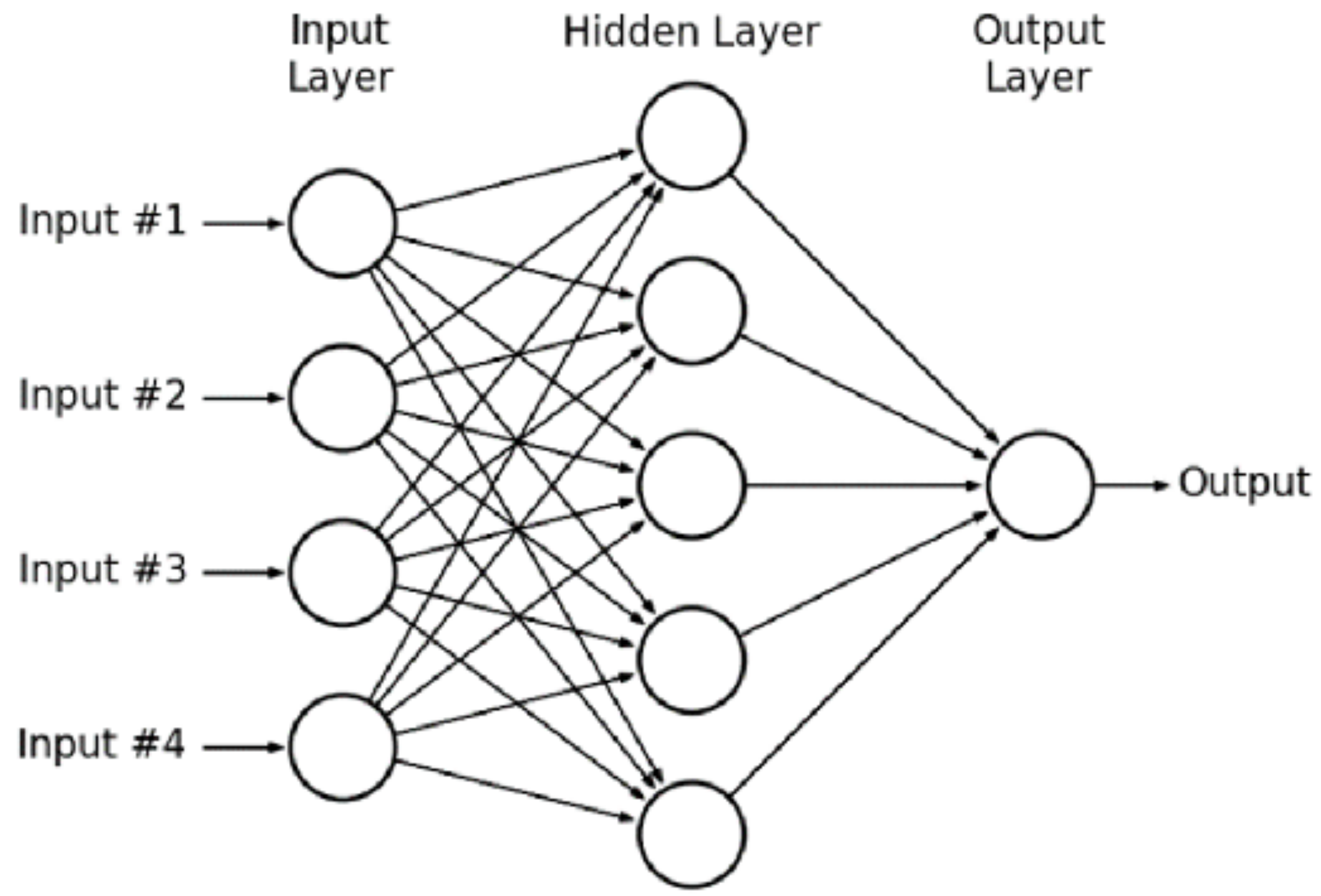




Perceptrons and Neural Networks

- Perceptrons were the first ML classifiers
- More generally, Multilayer Perceptrons (MLPs) can learn any abitrary decision boundary (i.e., non-linear) by adding more hidden layers
- Training via backpropogation





MSE Loss

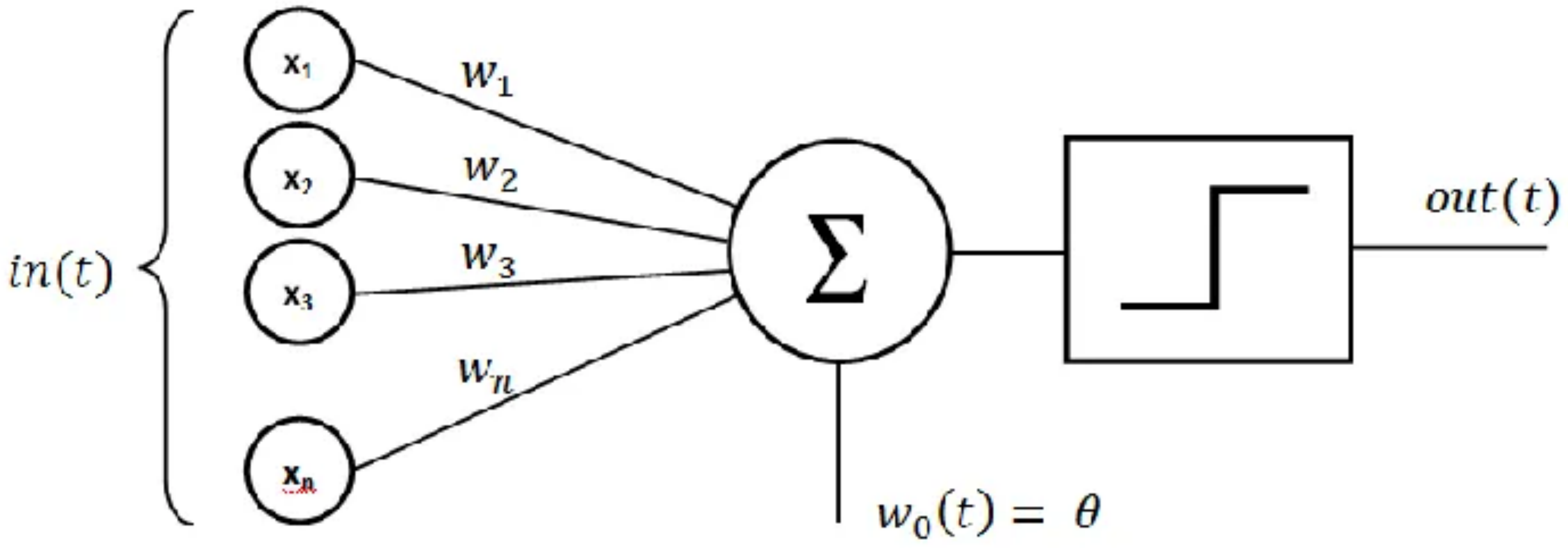
$$\mathscr{L} = \frac{1}{2}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

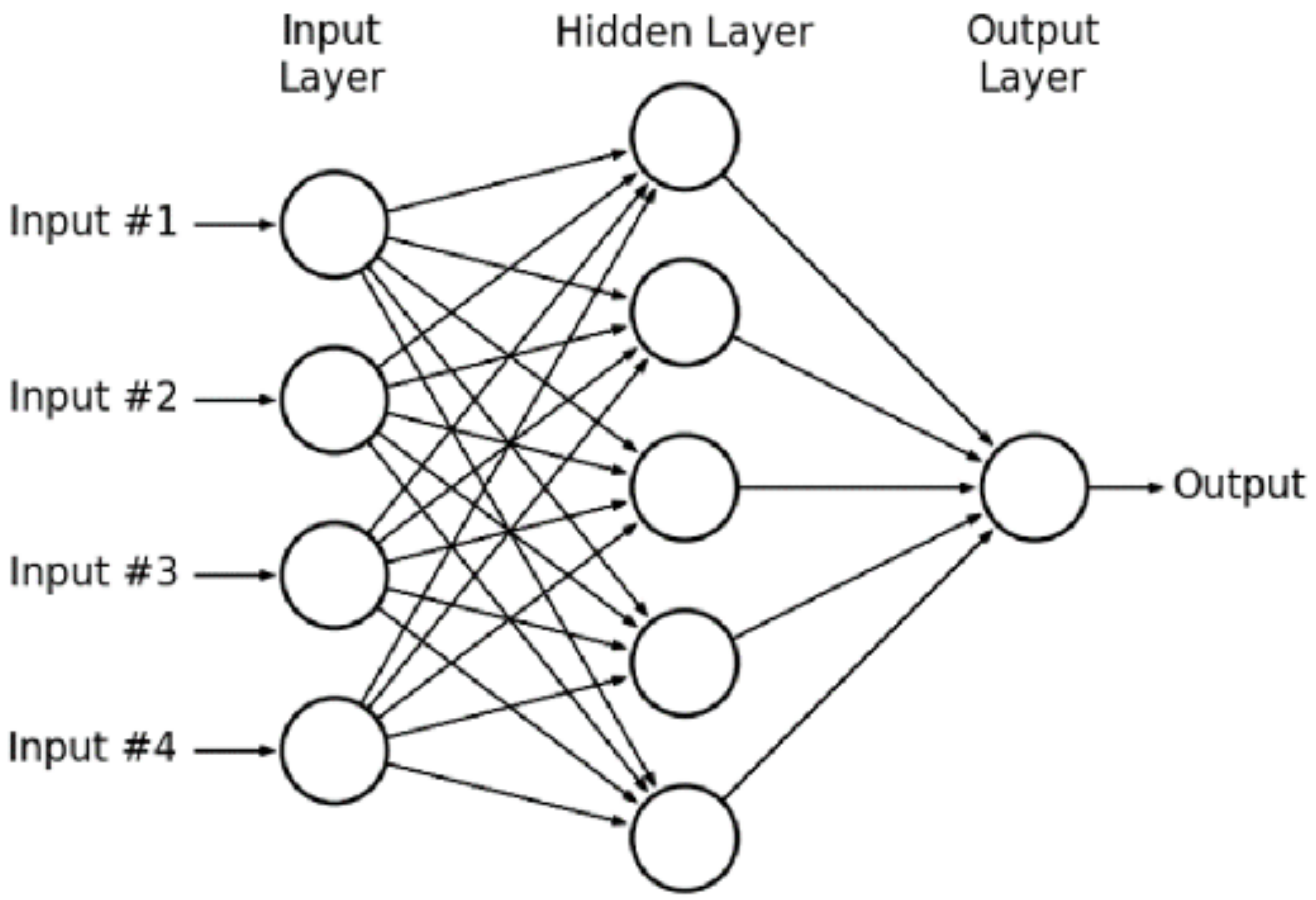
Weight updates

$$w \leftarrow w - \alpha\frac{\partial\mathscr{L}}{\partial w} \quad \text{where} \quad \frac{\partial\mathscr{L}}{\partial\mathbf{w}} = \frac{\partial\mathscr{L}}{\partial\mathbf{u}}\frac{\partial\mathbf{u}}{\partial\mathbf{w}}$$

Perceptrons and Neural Networks

- Perceptrons were the first ML classifiers
- More generally, Multilayer Perceptrons (MLPs) can learn any abitrary decision boundary (i.e., non-linear) by adding more hidden layers
- Training via backpropogation





MSE Loss

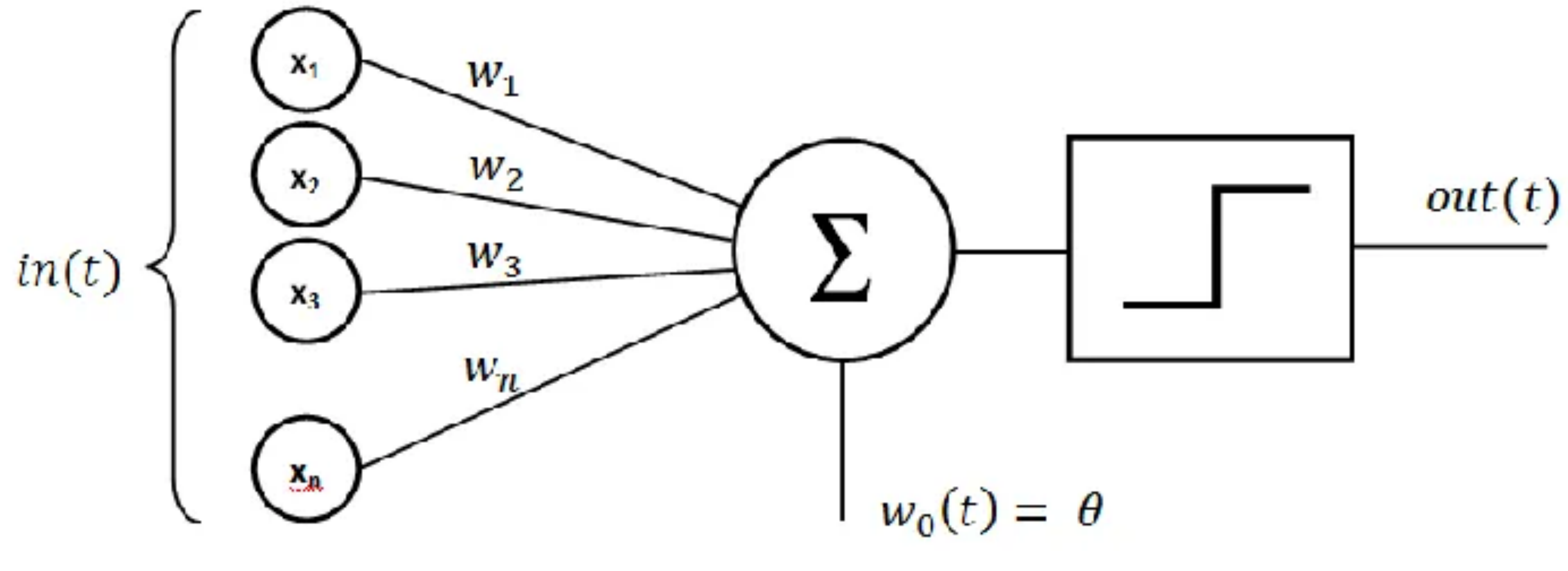
$$\mathscr{L} = \frac{1}{2}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

prediction

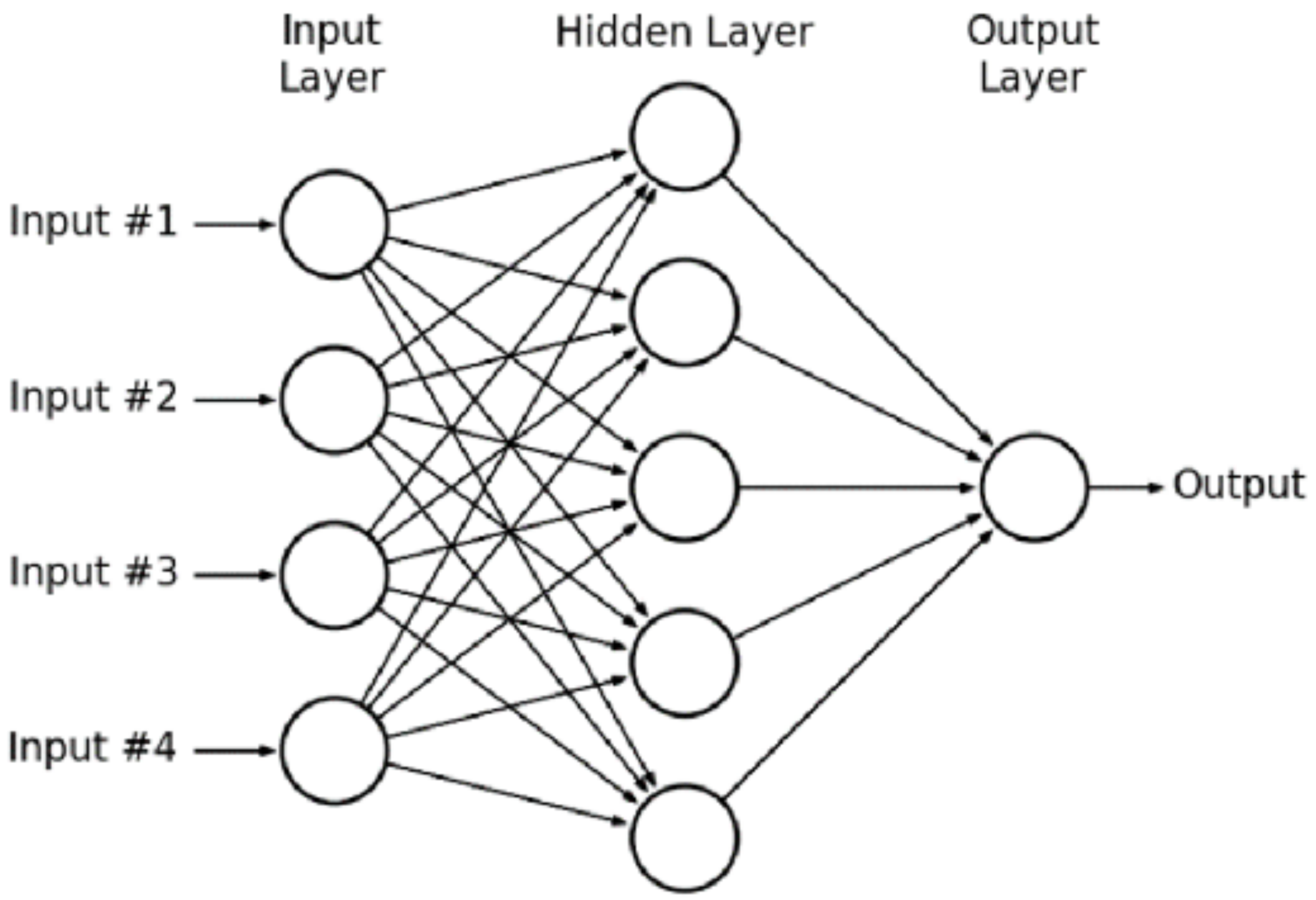
Weight updates

$$w \leftarrow w - \alpha \frac{\partial \mathscr{L}}{\partial w} \quad \text{where} \quad \frac{\partial \mathscr{L}}{\partial \mathbf{w}} = \frac{\partial \mathscr{L}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{w}}$$

Perceptrons and Neural Networks



- Perceptrons were the first ML classifiers
- More generally, Multilayer Perceptrons (MLPs) can learn any abitrary decision boundary (i.e., non-linear) by adding more hidden layers
- Training via backpropogation



MSE Loss

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{m}(y_i - \underset{\uparrow}{\hat{y}_i})^2$$

prediction

Weight updates

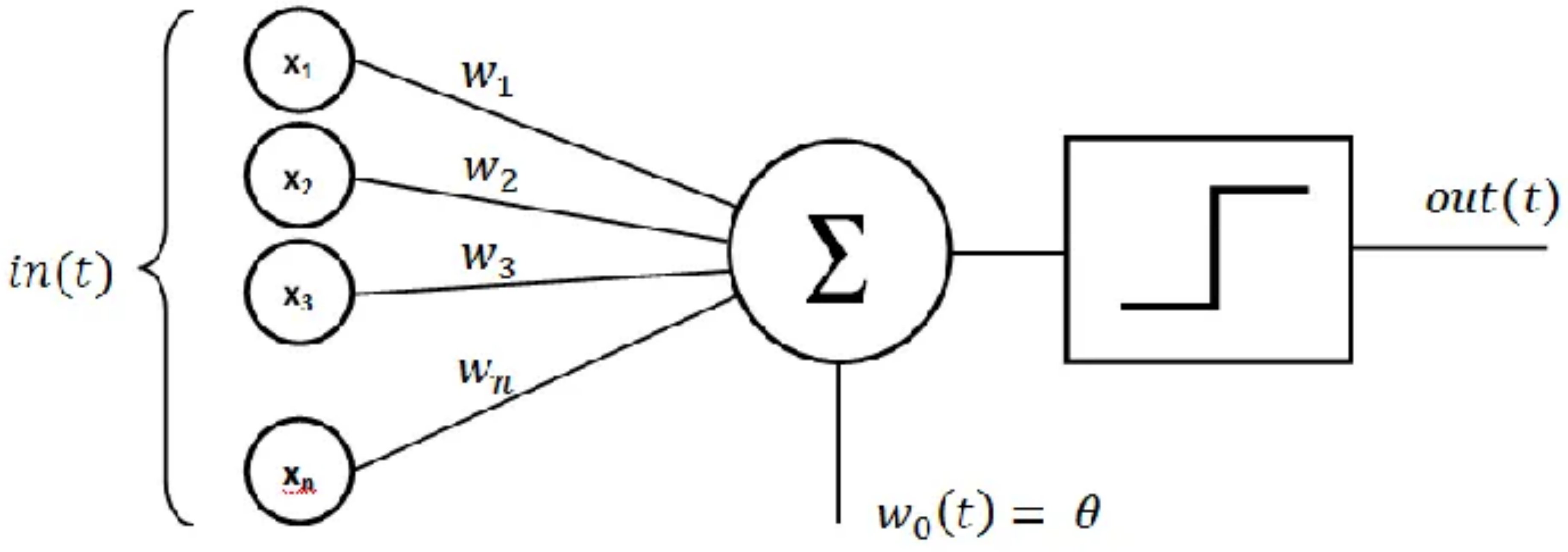
$$w \leftarrow w - \underset{\uparrow}{\alpha}\frac{\partial \mathcal{L}}{\partial w}$$

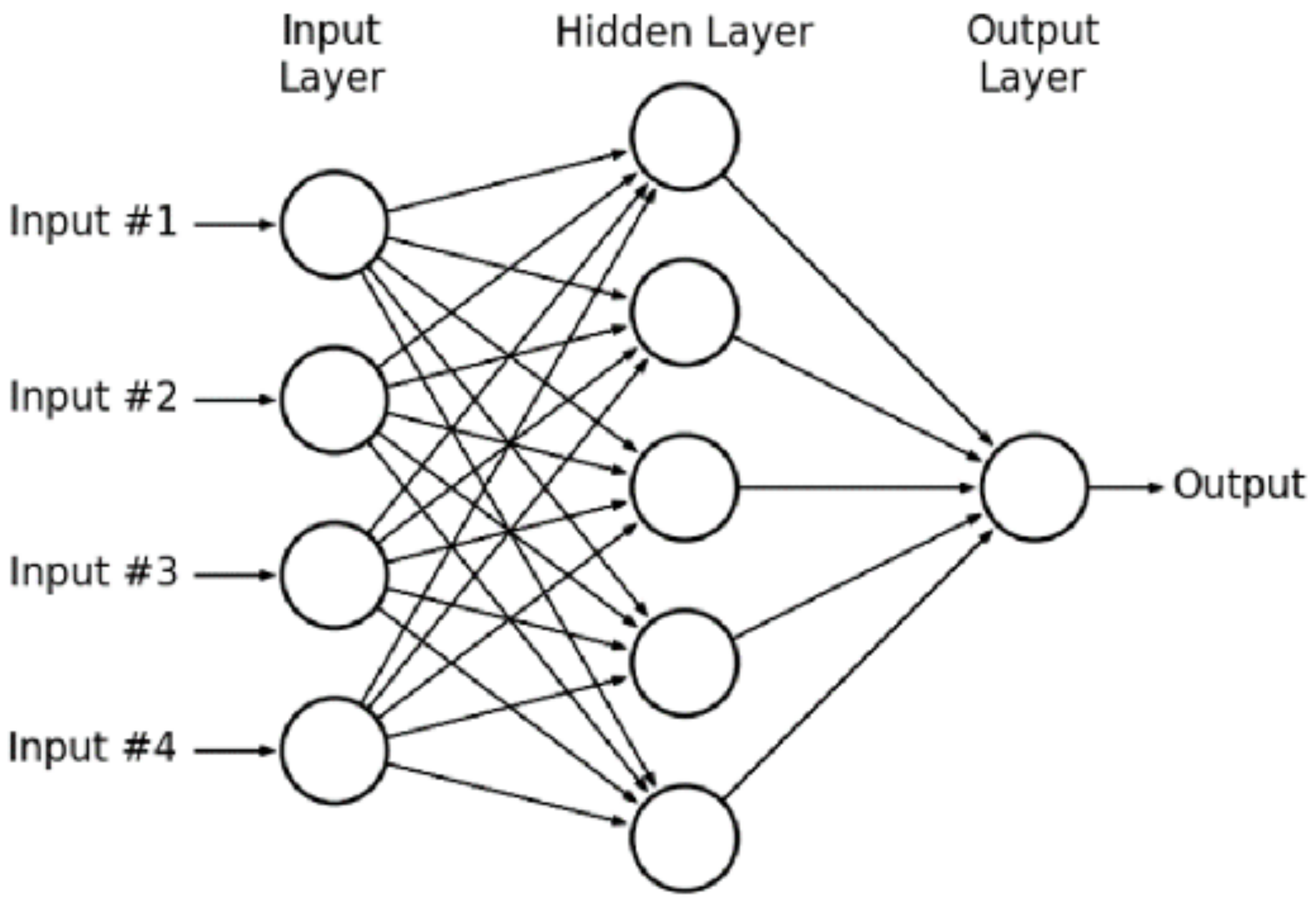
learning rate

where $\dfrac{\partial \mathcal{L}}{\partial \mathbf{w}} = \dfrac{\partial \mathcal{L}}{\partial \mathbf{u}}\dfrac{\partial \mathbf{u}}{\partial \mathbf{w}}$

Perceptrons and Neural Networks

- Perceptrons were the first ML classifiers
- More generally, Multilayer Perceptrons (MLPs) can learn any abitrary decision boundary (i.e., non-linear) by adding more hidden layers
- Training via backpropogation





MSE Loss

$$\mathscr{L} = \frac{1}{2}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

prediction

Weight updates

$$w \leftarrow w - \alpha \frac{\partial \mathscr{L}}{\partial w}$$

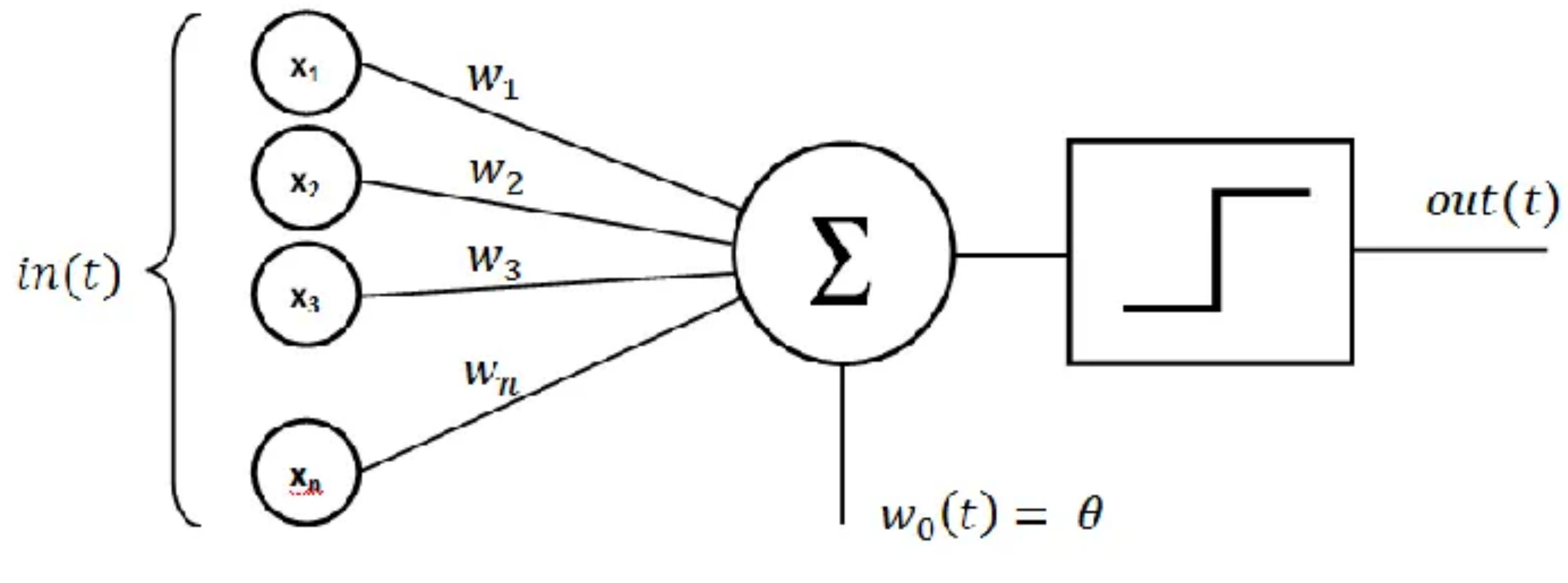
learning rate

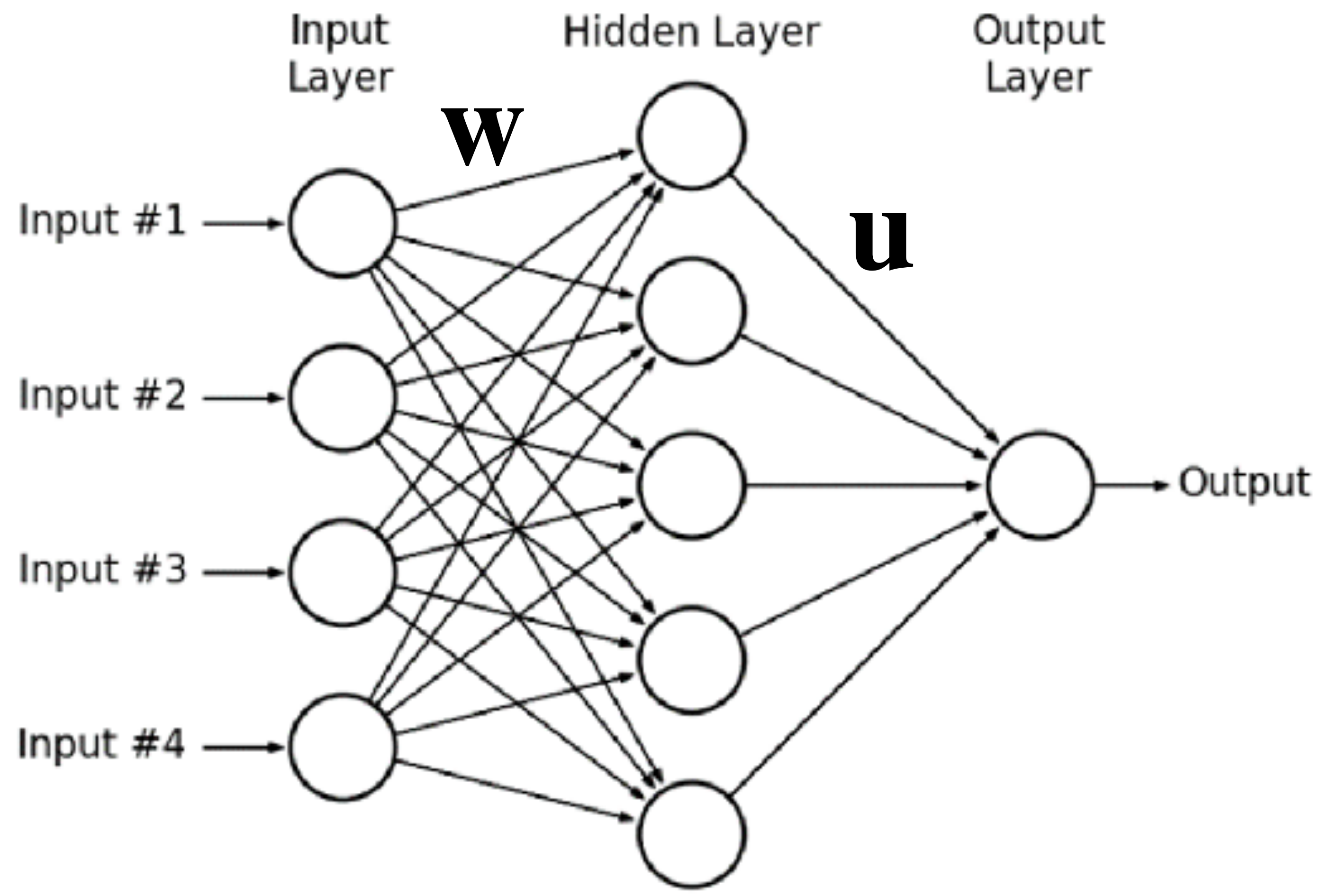
where $\frac{\partial \mathscr{L}}{\partial \mathbf{w}} = \frac{\partial \mathscr{L}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{w}}$

Chain rule is used to pass derivatives over layers

Perceptrons and Neural Networks

- Perceptrons were the first ML classifiers
- More generally, Multilayer Perceptrons (MLPs) can learn any abitrary decision boundary (i.e., non-linear) by adding more hidden layers
- Training via backpropogation





MSE Loss

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

prediction

Weight updates

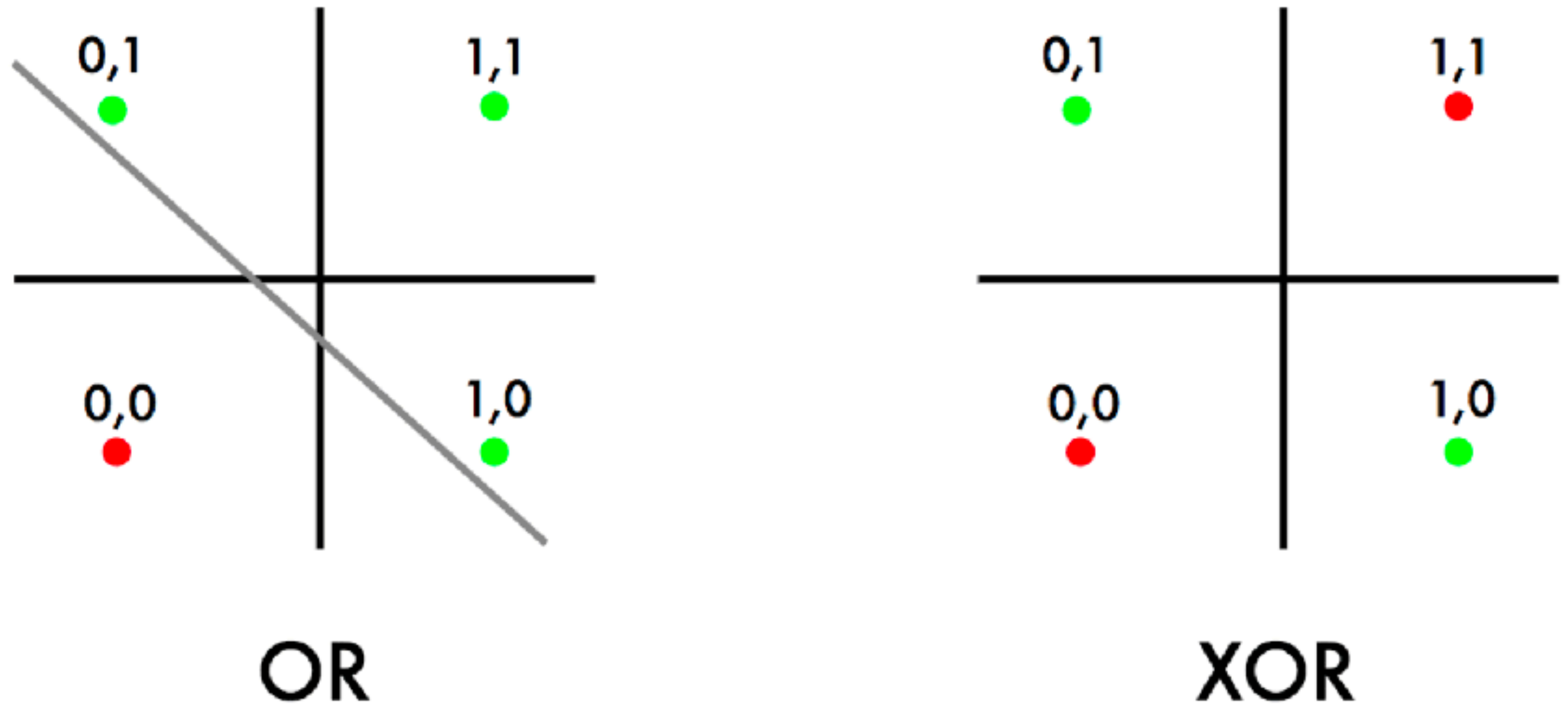
$$w \leftarrow w - \alpha\frac{\partial \mathcal{L}}{\partial w} \quad \text{where} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{w}}$$

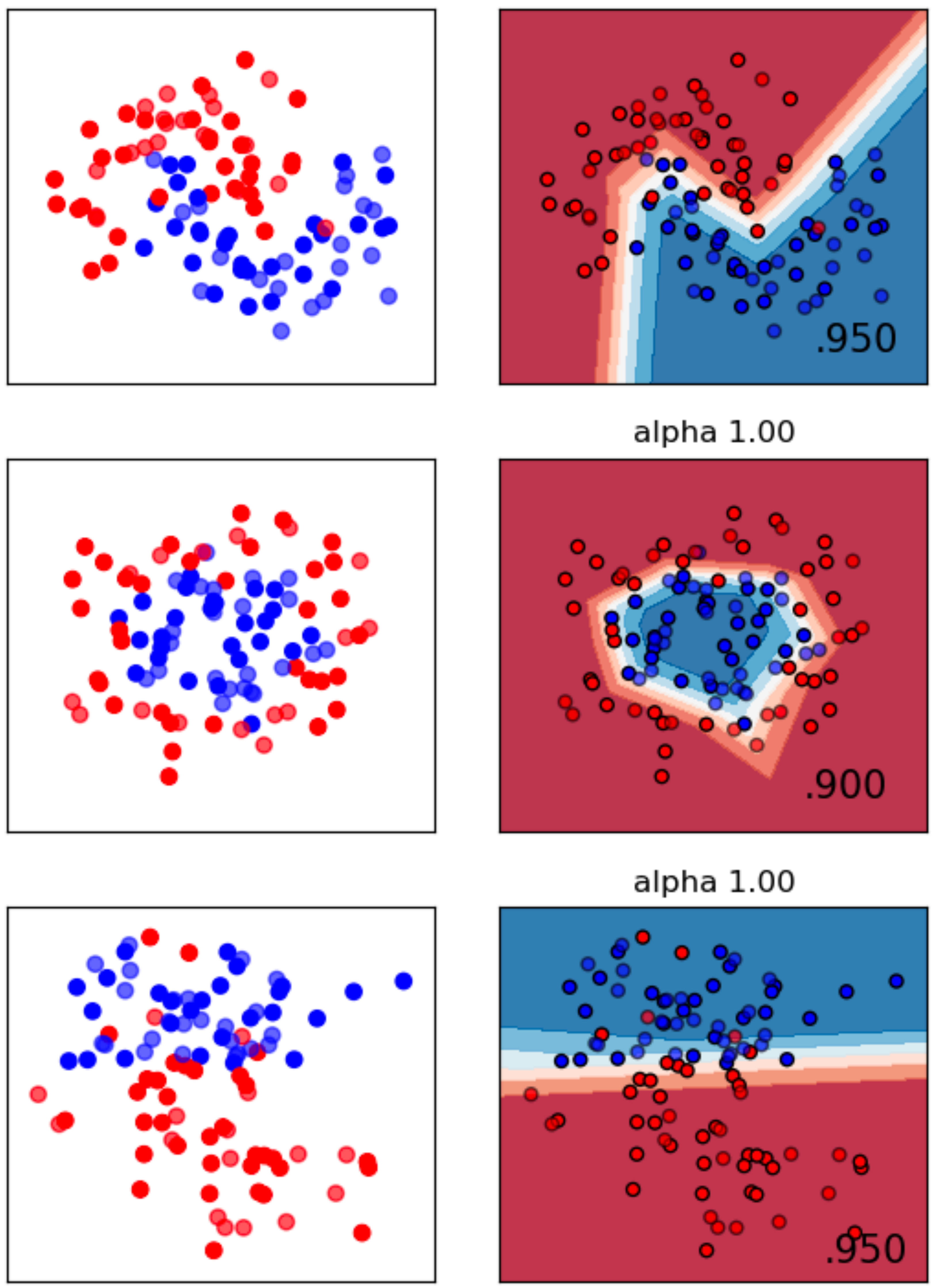
learning rate

Chain rule is used to pass derivatives over layers

Decision boundaries

- Decision boundaries can correspond to logical rules, even when non-linear
- But most decision-boundaries are certainly not easily explainable using symbolic operations
- Rather, the feature space is carved up based on similarity to trained exemplars

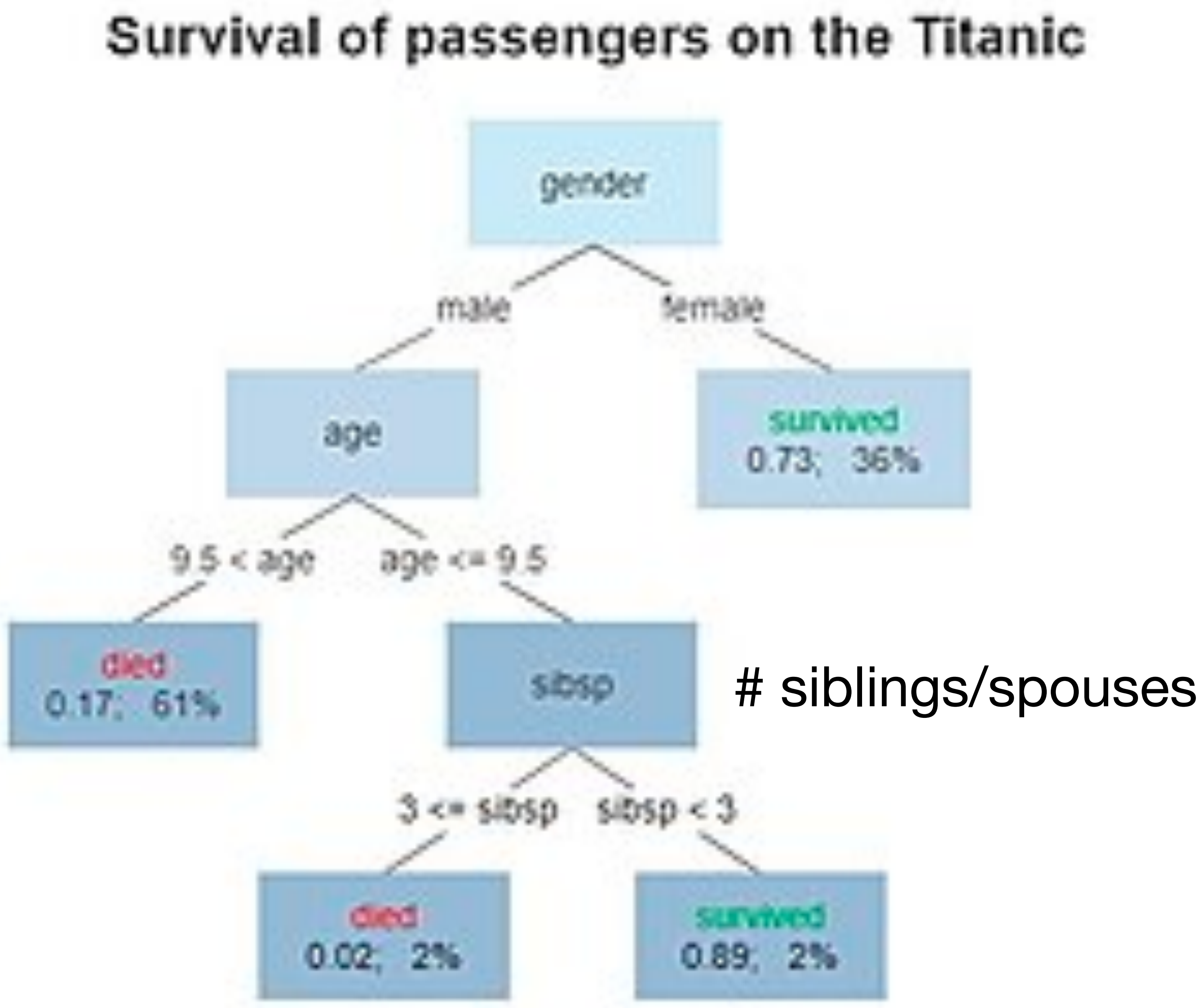


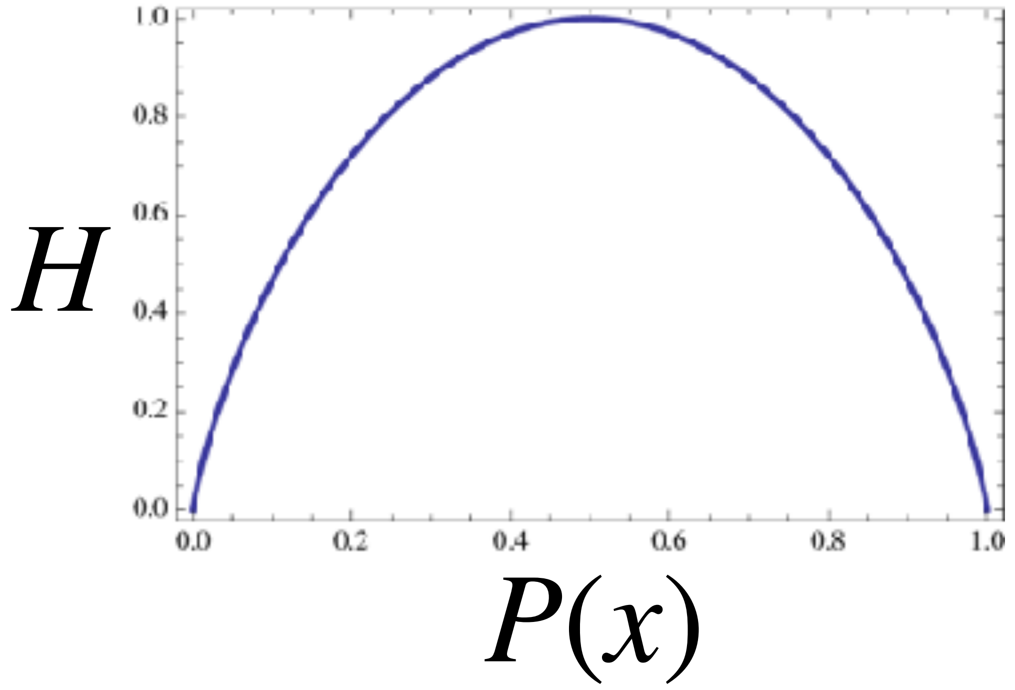


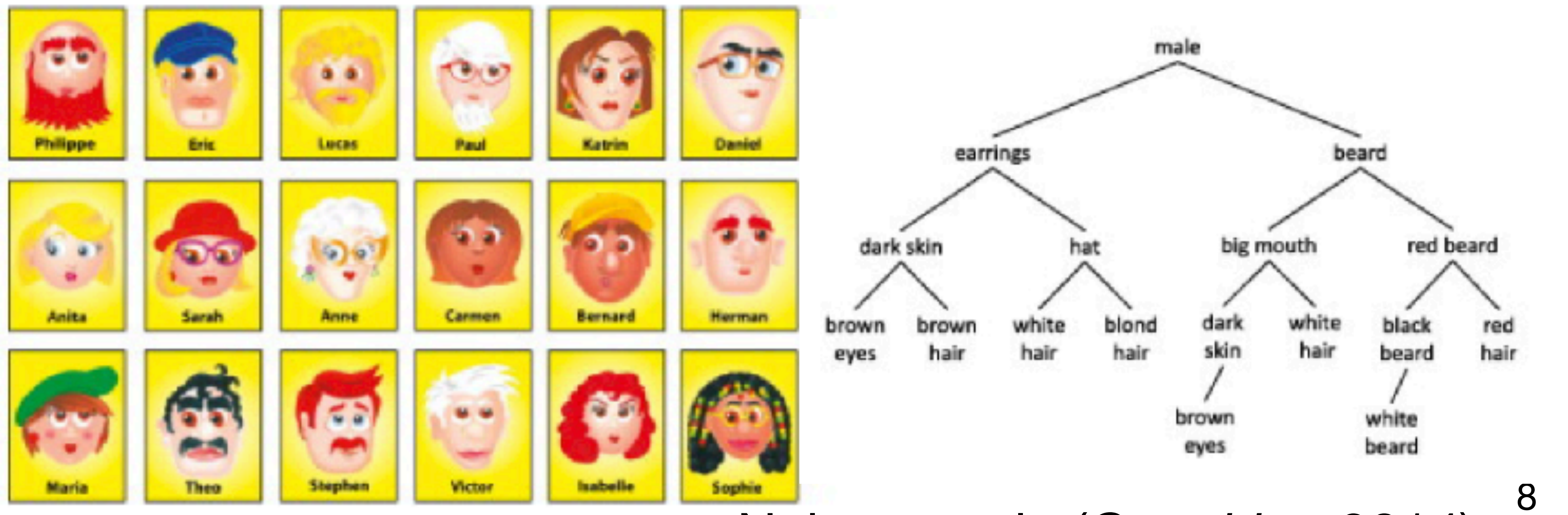
Decision-trees

- **Decision Trees** are the quintessential rule-based clasifier
  - Easy to interpret, but can be prone to bias and overfitting
- ID3 algorithm:
  - Calculate the Information gain (IG) of each feature
    - Shannon (1948) Entropy: $H(\mathbf{X}) = -\sum P(x)\log P(x)$
    - $IG(\mathbf{X}, f) = H(\mathbf{X}) - H(\mathbf{X}\,|\,f)$

      How much does feature f reduce entropy? The more the feature can even split the data (across labels), the greater the reduction
      - If not naturally a binary feature, define a threshold that maximizes Entropy (i.e., split half)
  - Make a decision node using the feature with max(IG)
  - Repeat until we run out of features



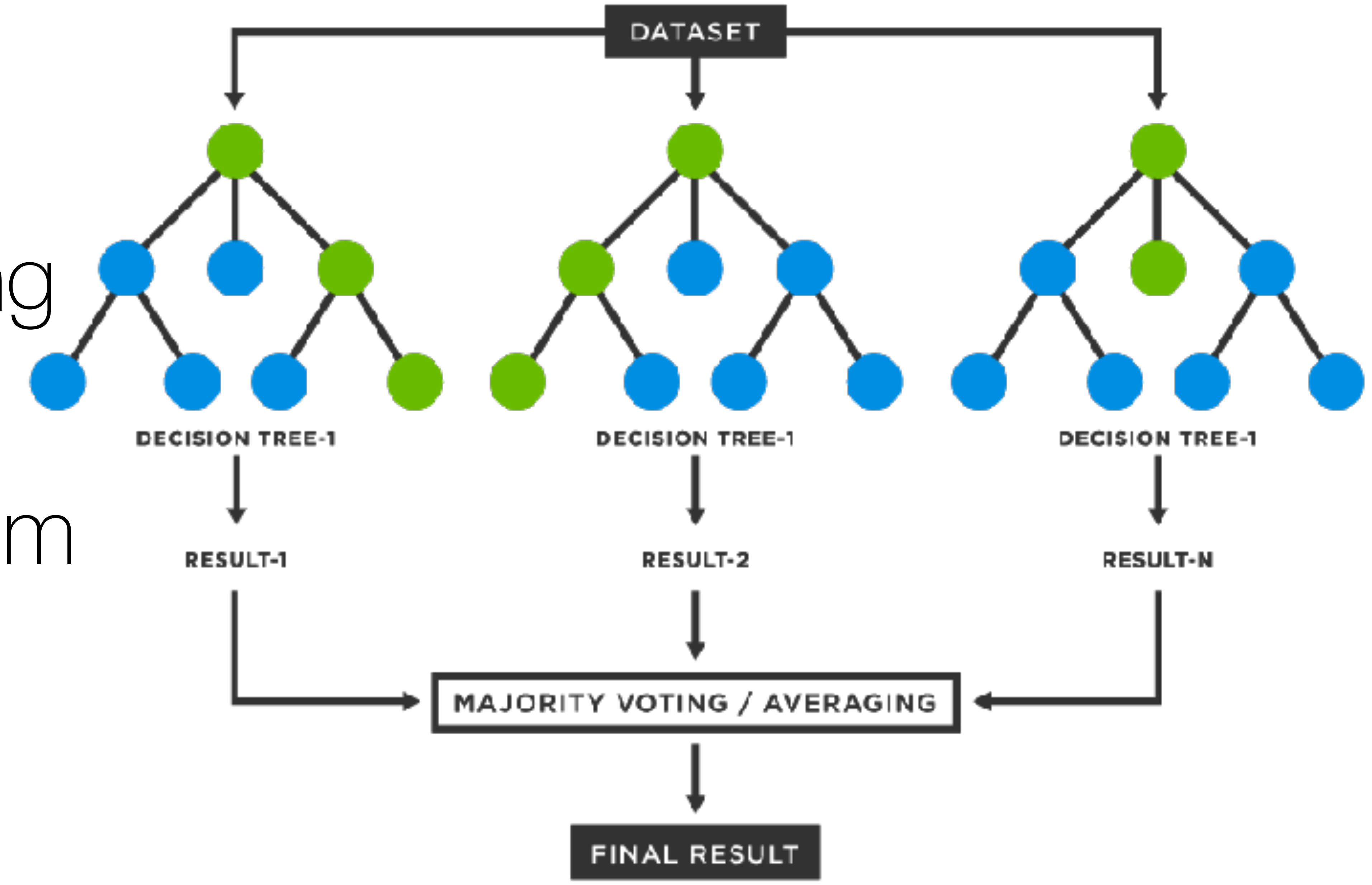


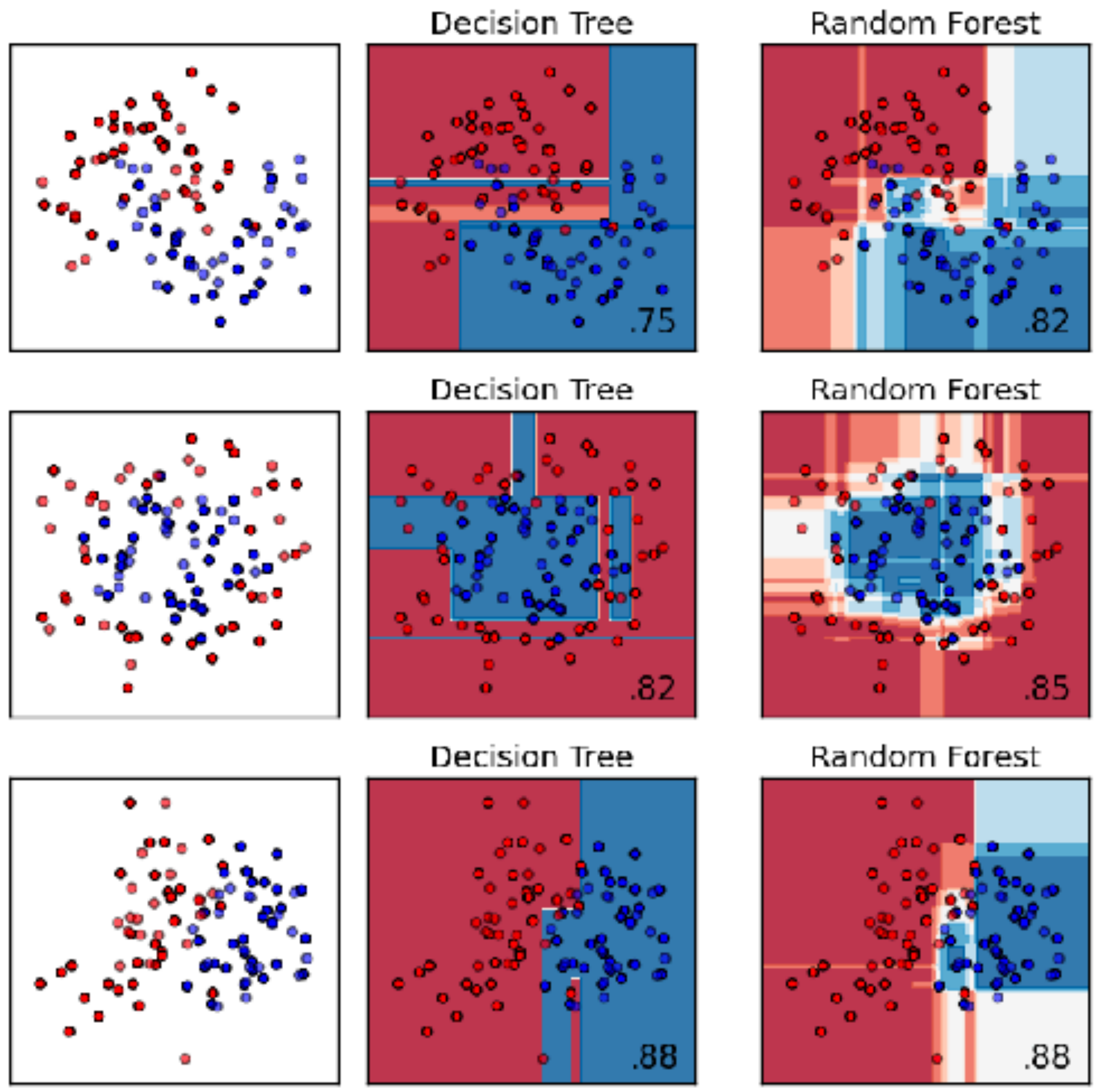


Nelson et al., (*Cognition* 2014)

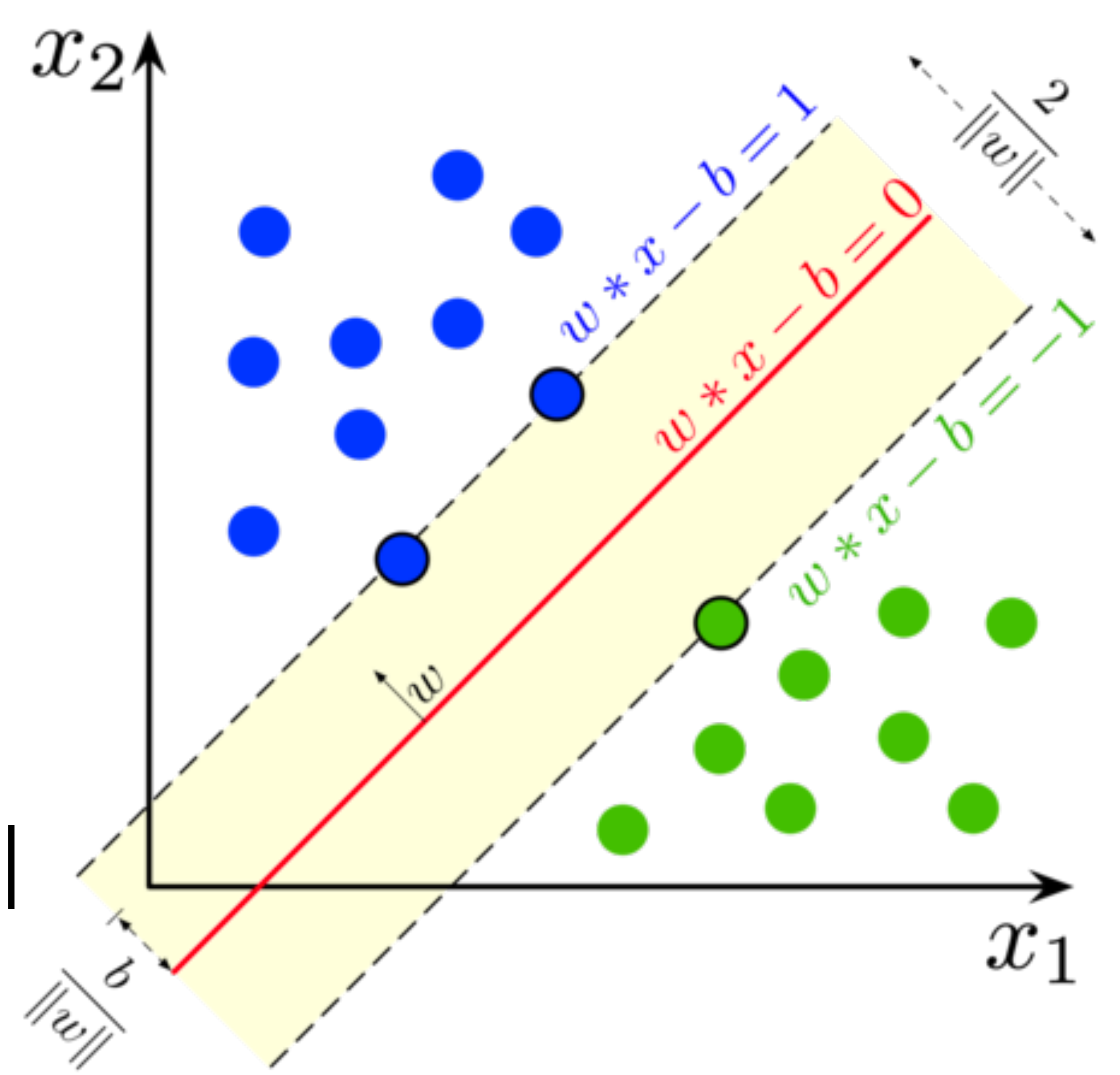
Random forests

- **Random forests** are an ensemble method combining random, uncorrelated decision trees
  - Each tree uses "feature bagging" to sample a random subset of features, ensuring low correlation among trees
  - Voting or averaging to make the final decision
- Ensemble methods are common in ML
  - Do brains also combine "opinions" from multiple decision-making systems?
- Aggregation over multiple trees is similar to how Bayesian concept learning operates over a distribution of rules, producing generalization patterns consistent with similarity-based theories

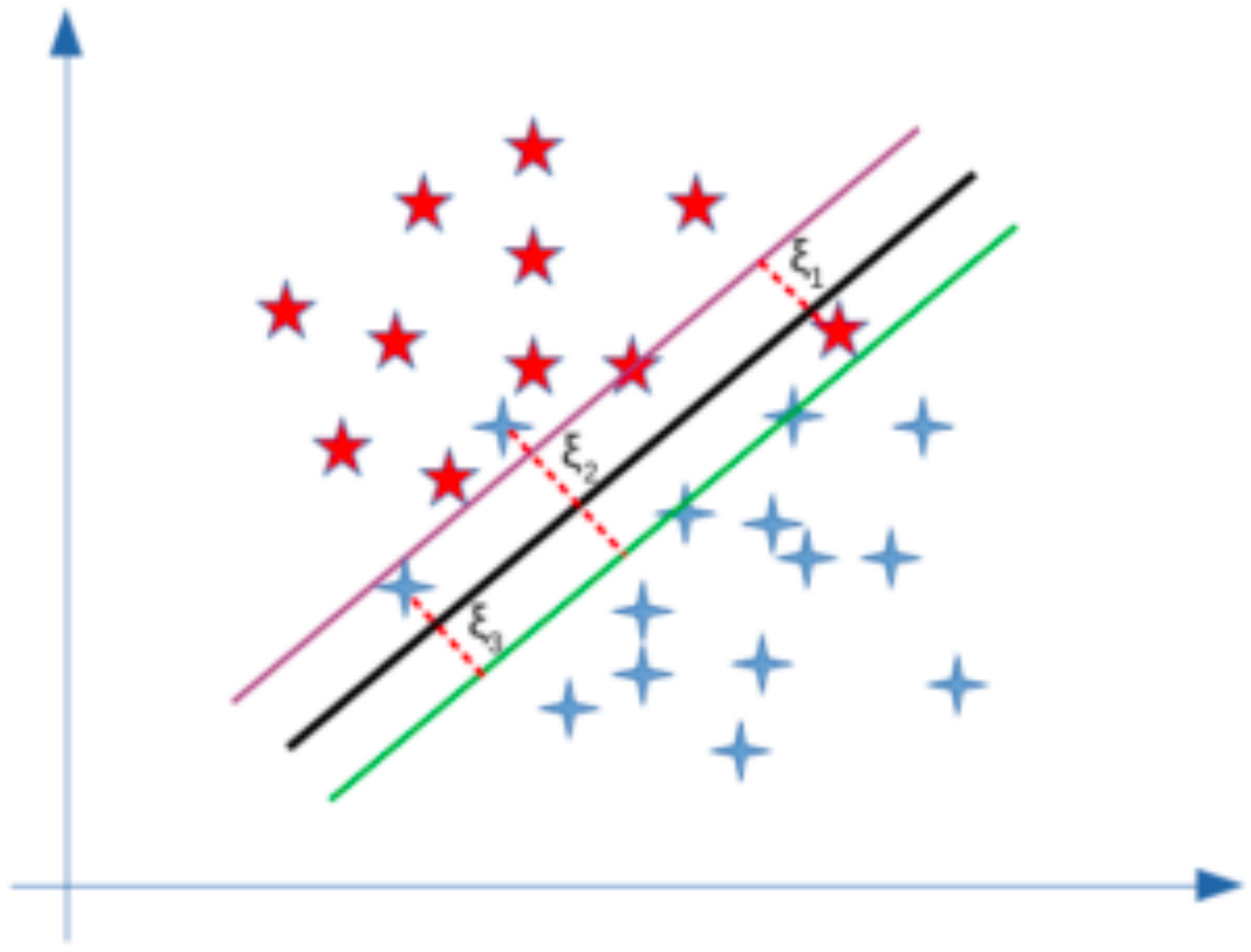




Support Vector Machines

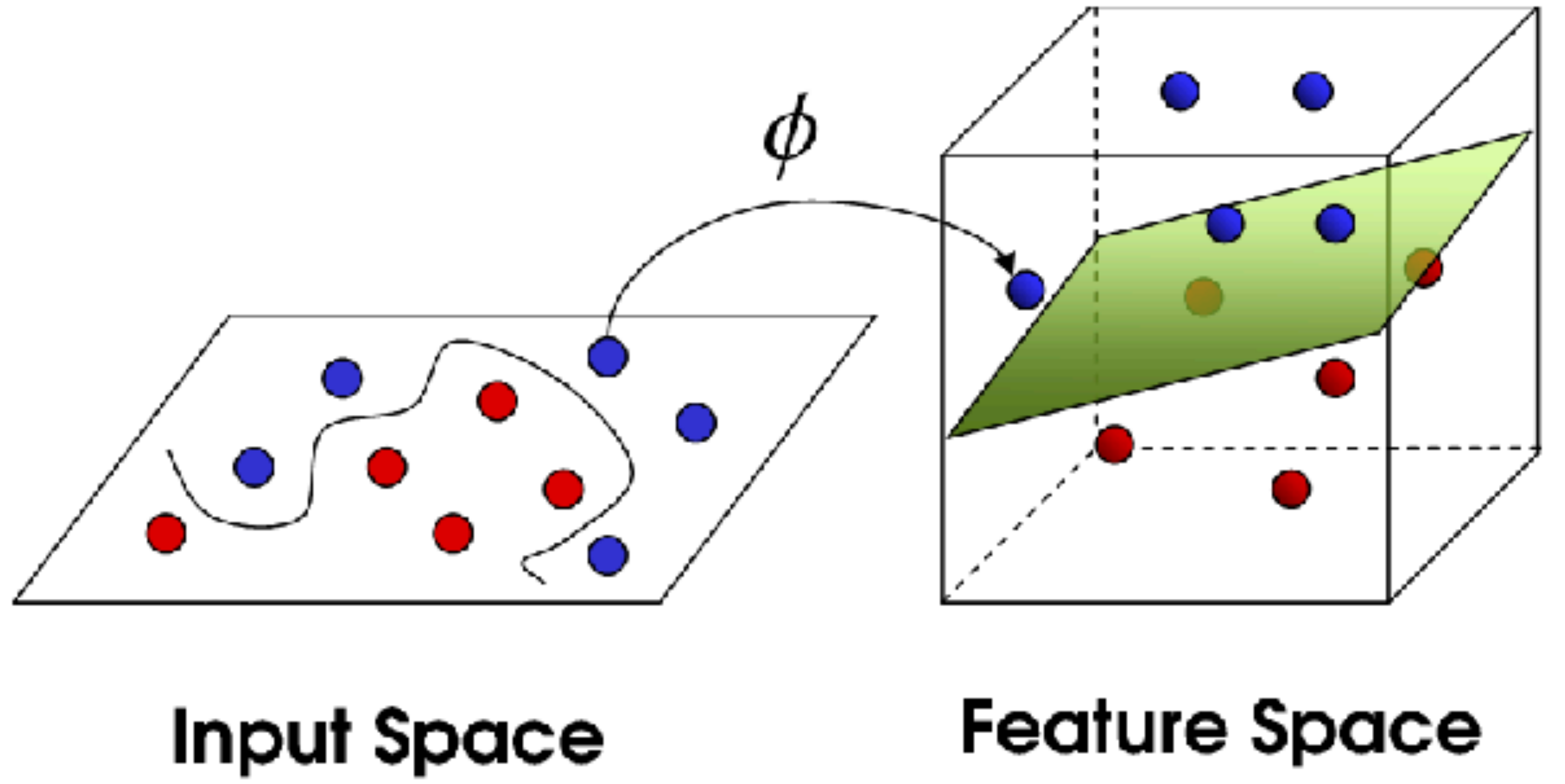


- Learn a decision boundary that best separates the data $\mathbf{w}^\top \mathbf{x} - b = 0$
- **Hard-margin**: with $y_i \in [-1,1]$, we want
  - $y_i(\mathbf{w}^\top \mathbf{x} - b) \geq 1$ (i.e., all data classified correctly)
  - And to maximize the margin between classes $\frac{2}{\|w\|}$, which we do by minimizing $\|w\|$
  - This gives us a constrained optimization problem:
    $\mathcal{L}(w, b) = \|w\|$ subject to $y_i(\mathbf{w}^\top \mathbf{x} - b) \geq 1$
  - The solution is completely determined by the $\mathbf{x}_i$ closest to the decision-boundary (i.e., support vectors)
- **Soft-margin**: Since data might not be linearly separable, use a soft-constraint to weight how much we care about the margin vs. errors:
  - C is a penalty term definining how much we care about errors
  - ζ_i is the distance of a datapoint from the decision-boundary (i.e., slack variable)

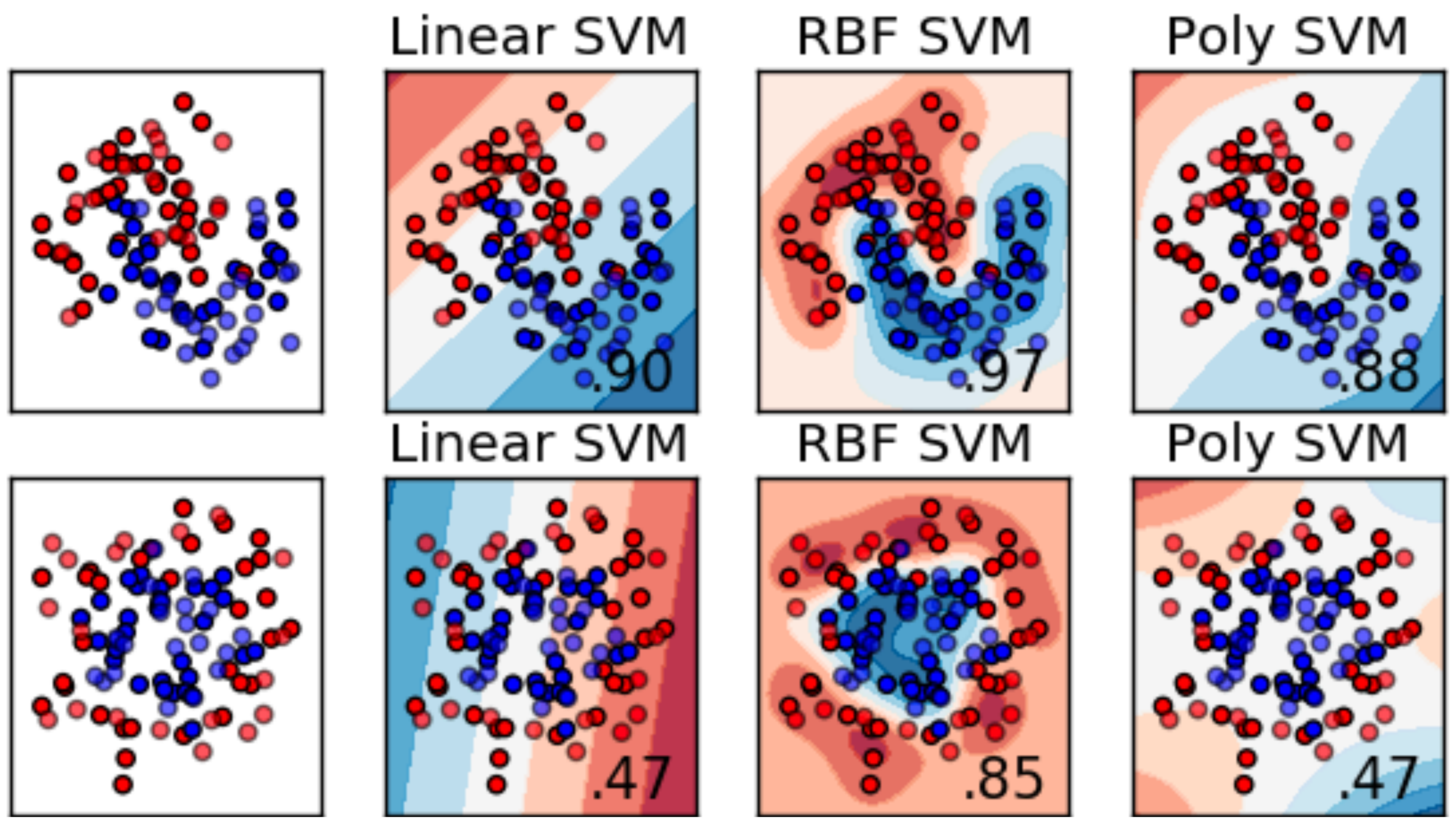


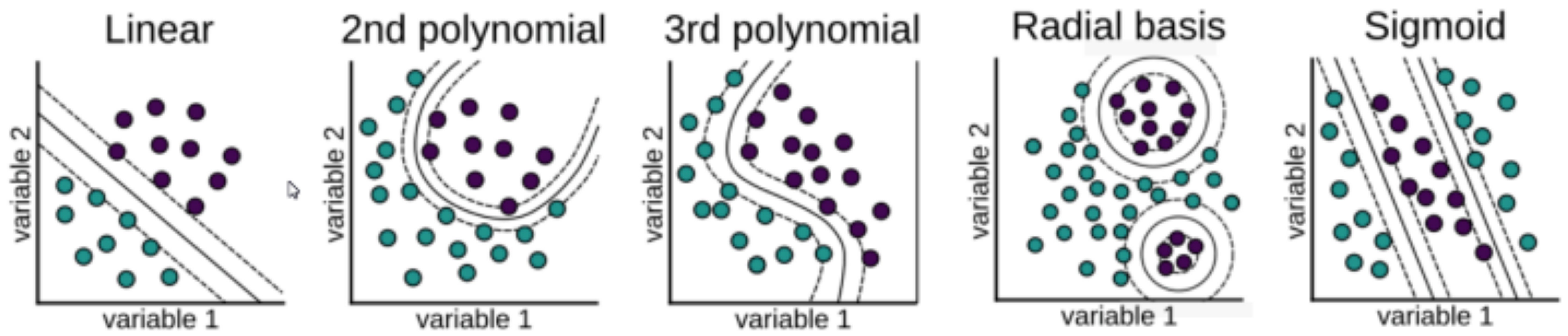
$$\mathcal{L}(w, b) = \|w\| - C \sum_i^N \zeta_i \qquad y_i(\mathbf{w}^\top \mathbf{x} - b) \geq 1 - \zeta_i, \quad \zeta_i \geq 0 \ \forall_i \in \{i, N\}$$

Kernel SVMs



- What about problems with non-linear decision boundaries?
- Kernel trick "projects" the data to a higher dimension, such that we can still learn a linear decision boundary
  - Rather than learning $\mathbf{w}^\top \mathbf{x} - b = 0$, we use a kernel to map $\mathbf{X}$ onto a feature space $\Phi = \phi(\mathbf{X})$ e.g., polynomial kernel $\phi(\mathbf{x}) = (1, \mathbf{x}, \mathbf{x}^2, \mathbf{x}^3, \ldots)$
  - We then substitute $\phi(\mathbf{x})$ for $\mathbf{x}$ and use all the same equations, e.g., decision boundary becomes $\mathbf{w}^\top \phi(\mathbf{x}) - b = 0$
- There are many types of kernels, in fact, every neural network learned by gradient descent is approximately a kernel machine (i.e., $y = f\left(\phi(\mathbf{x})\right)$; Domingos, 2020)





Naïve Bayes classifier

- First generative model: rather than learning a decision boundary, learns the distribution of the each category (which can be used to generate new data)
- Called naïve because we assume all features are independent
  - Easy and fast to learn
  - Can generalize to new feature values outside the data, although naïve assumption may be unrealistic
- We use Bayes' theorem to compute the posterior probability of an datapoint belonging to some class c_k given it's features $\mathbf{x}$:

$$P(c_k \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c_k)P(c_k)}{P(\mathbf{x})} \qquad \text{posterior} = \frac{\text{likelihood} * \text{class prior}}{\text{evidence}}$$

$$= \frac{P(x_1 \mid c_k)P(x_2 \mid c_k)\ldots P(x_n \mid c_k)P(c_k)}{P(x_1)P(x_2)\ldots P(x_n)} \propto P(c_k)\prod_j^n P(x_j \mid c_k)$$

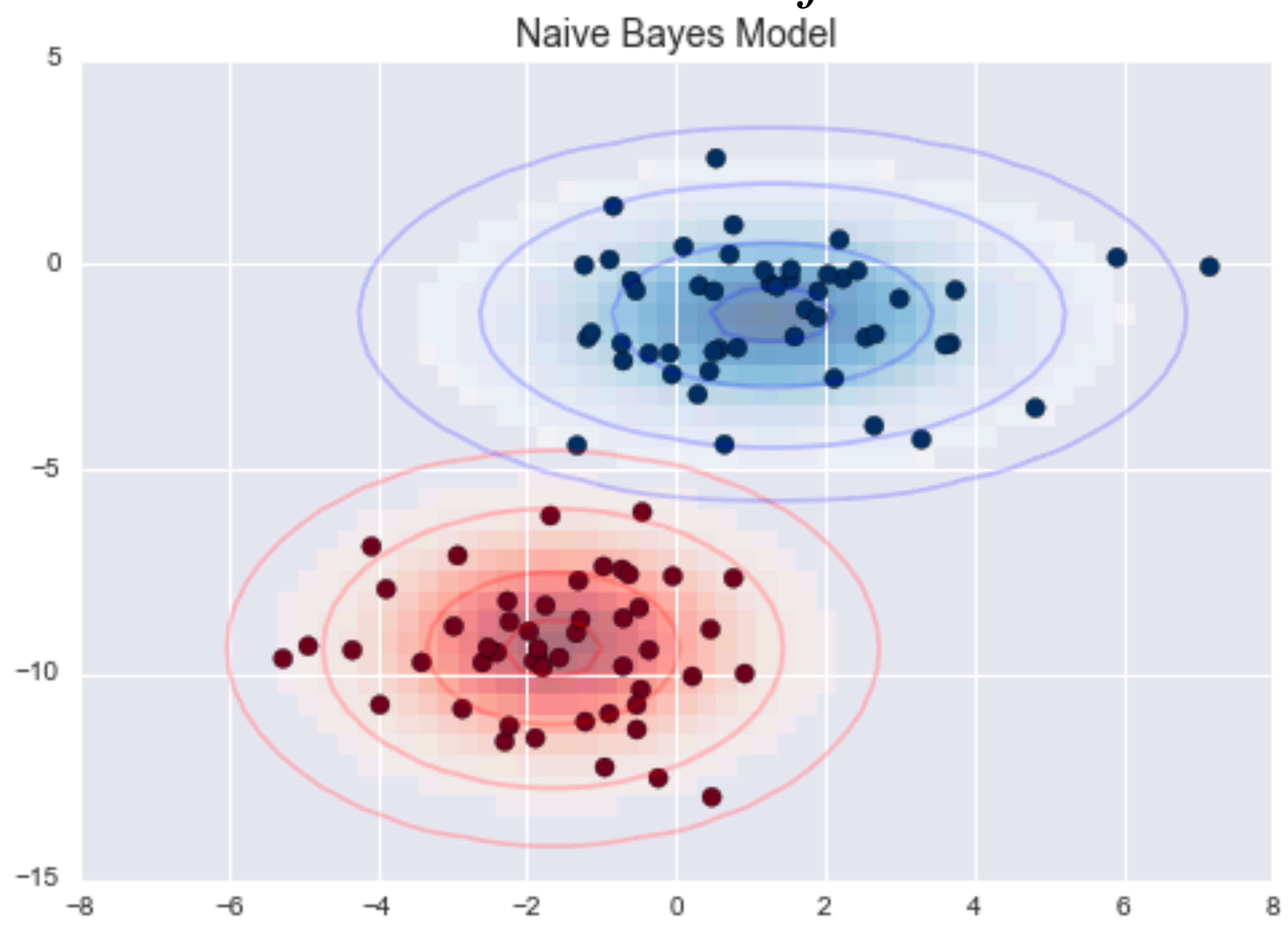
denominator removed, because it is the same for all data

Decision $$y = \arg\max_k P(c_k)\prod_i^n P(x_i \mid c_k)$$

although we can use the posterior to make probabilistic predictions

$$P(c_k|\mathbf{x}) \propto P(c_k)\prod_{j}^{n} P(x_j|c_k)$$

Naïve Bayes classifier



- Computing the prior and likelihood
- **Prior** is just how frequent the category is in the data $P(c_k) = \text{count}(c_k)/N$
- **Likelihood**:
  - When the **data is continous,** we can assume a Gaussian distribution*

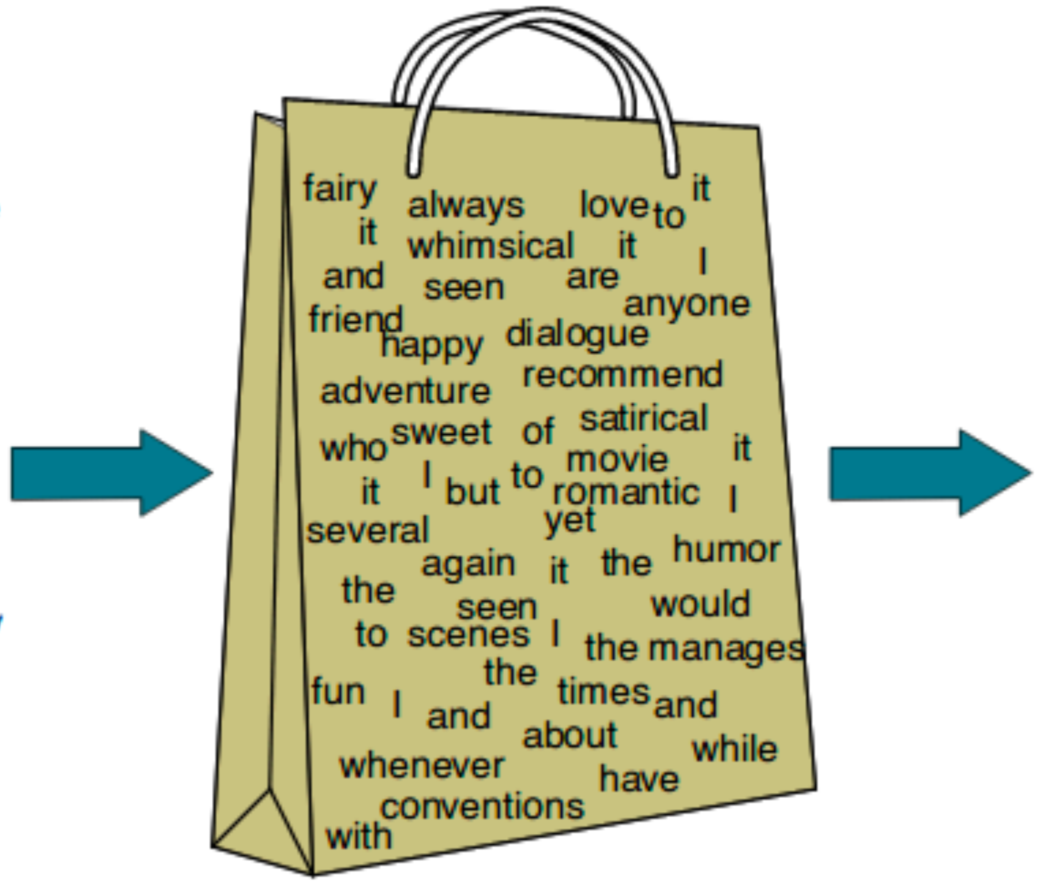
$$P(\mathbf{x}_j|c_k) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_k|}}\exp\left(-\frac{1}{2}(\mathbf{x}_j-\mu_k)^\top\Sigma_k^{-1}(\mathbf{x}_j-\mu_k)\right)$$

where μ_k and Σ_k are the mean vector and covariance matrix of the k-th category and d is the dimensionality of the data

  - For **text classification** (e.g., spam filters), use a "bag of words" representation
    - Vocabulary V, where each $\mathbf{x}_i = (x_1, x_2, \ldots x_V)$ represents the counts for each possible word
    - Likelihood with Laplacian (add 1) smoothing:

$$P(x_j|c_k) = \frac{\text{count}(x_j, c_k) + 1}{\sum_{x\in V}\text{count}(x, c_k) + |V|}$$

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

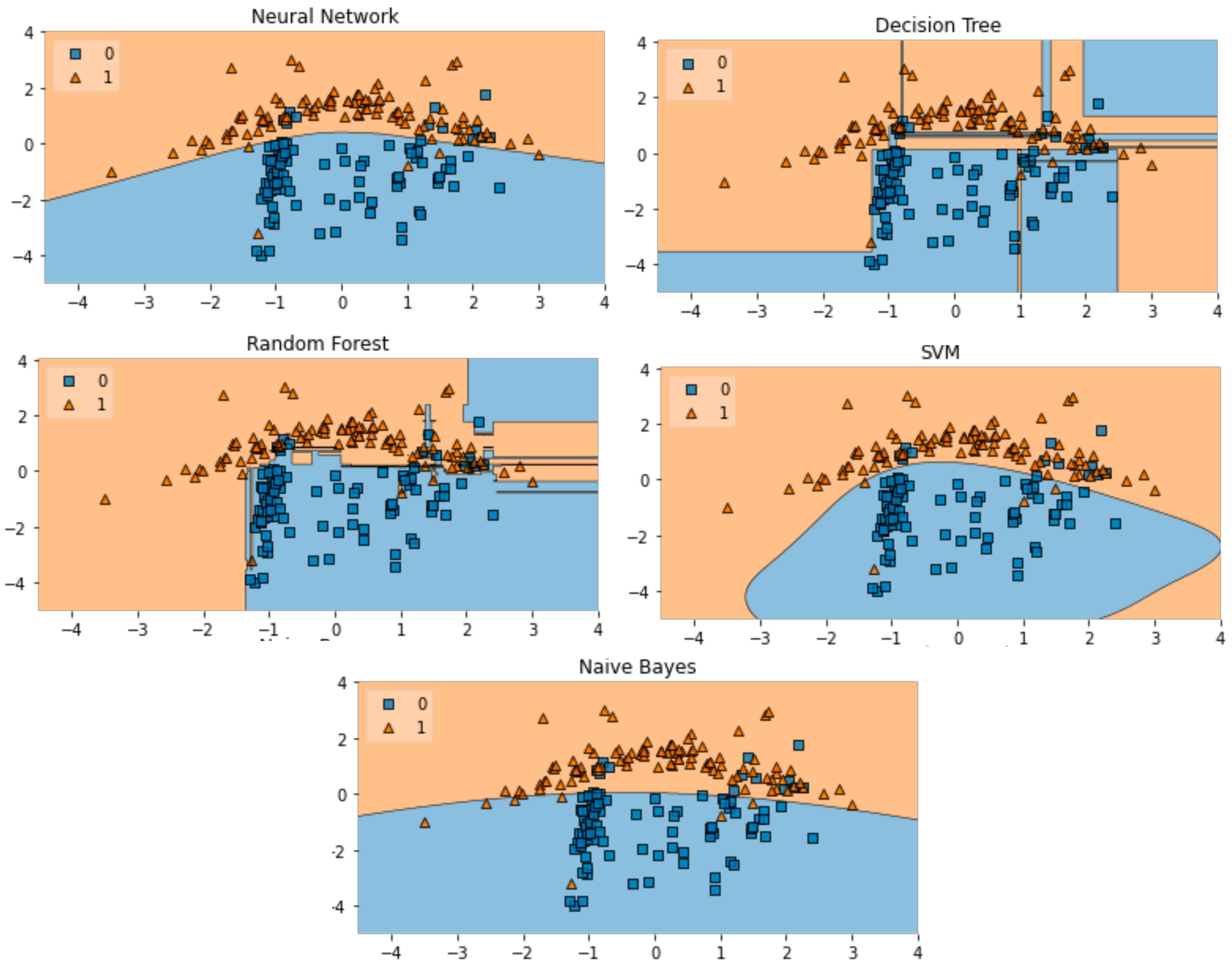


it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

* Don't need to memorize this for quizzes/exam

Supervised learning summary

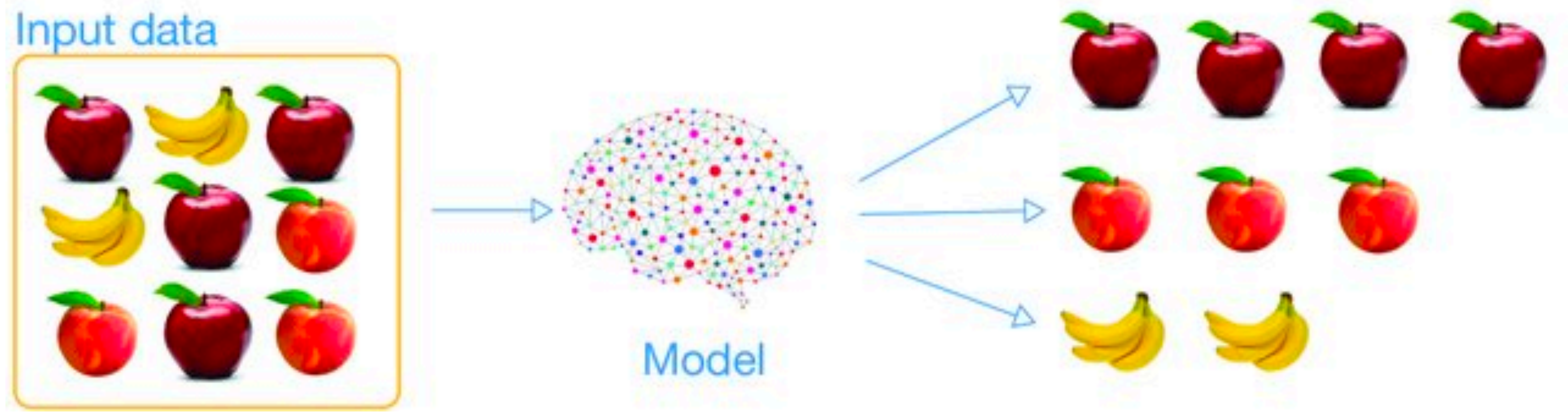
- Supervised learning is a **classification** problem
- Each method yields a corresponding **decision boundary** (rule-based interpretation)
- However, only decision trees operate on explicit rules
- Most **discriminative** approaches use similarity-based mechanisms (e.g., NNs and SVMs) to arrive at a decision-boundaries based on carving up self-similar regions based on labeled exemplars
- However, **generative** methods learn implicit representations of each category
    - Naïve Bayes learns distributions for each category (prototype interpretation)



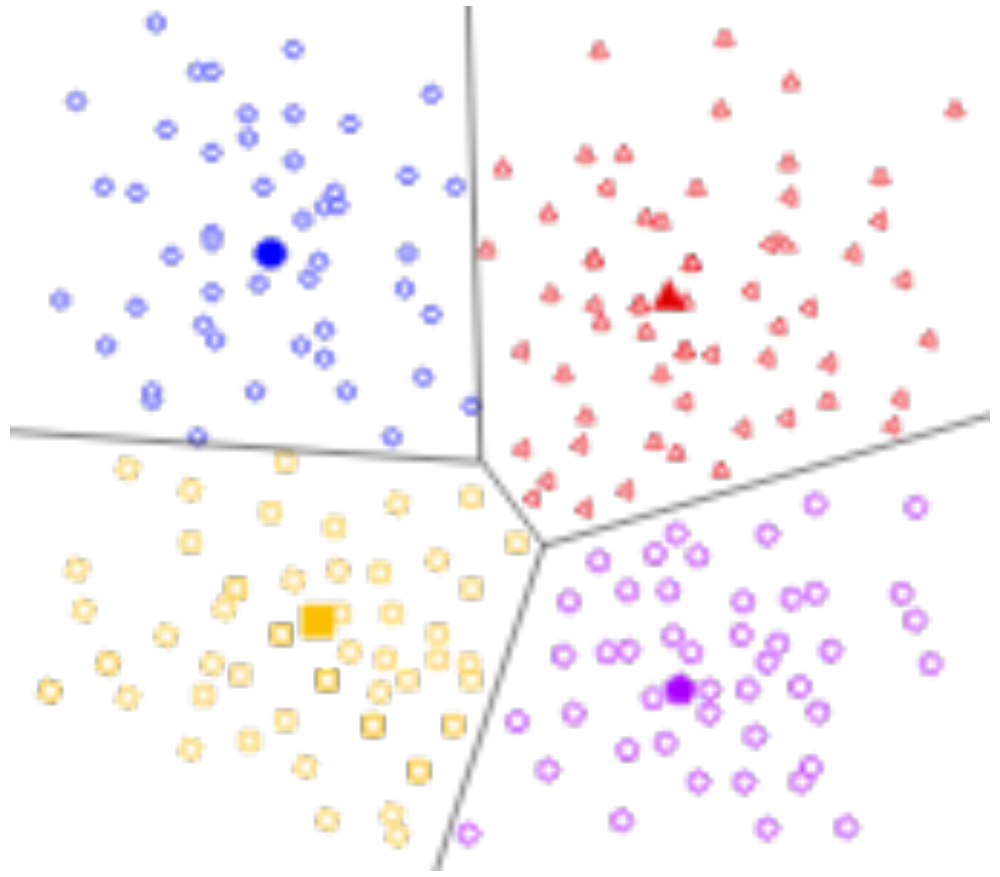
5 min break

Unsupervised learning

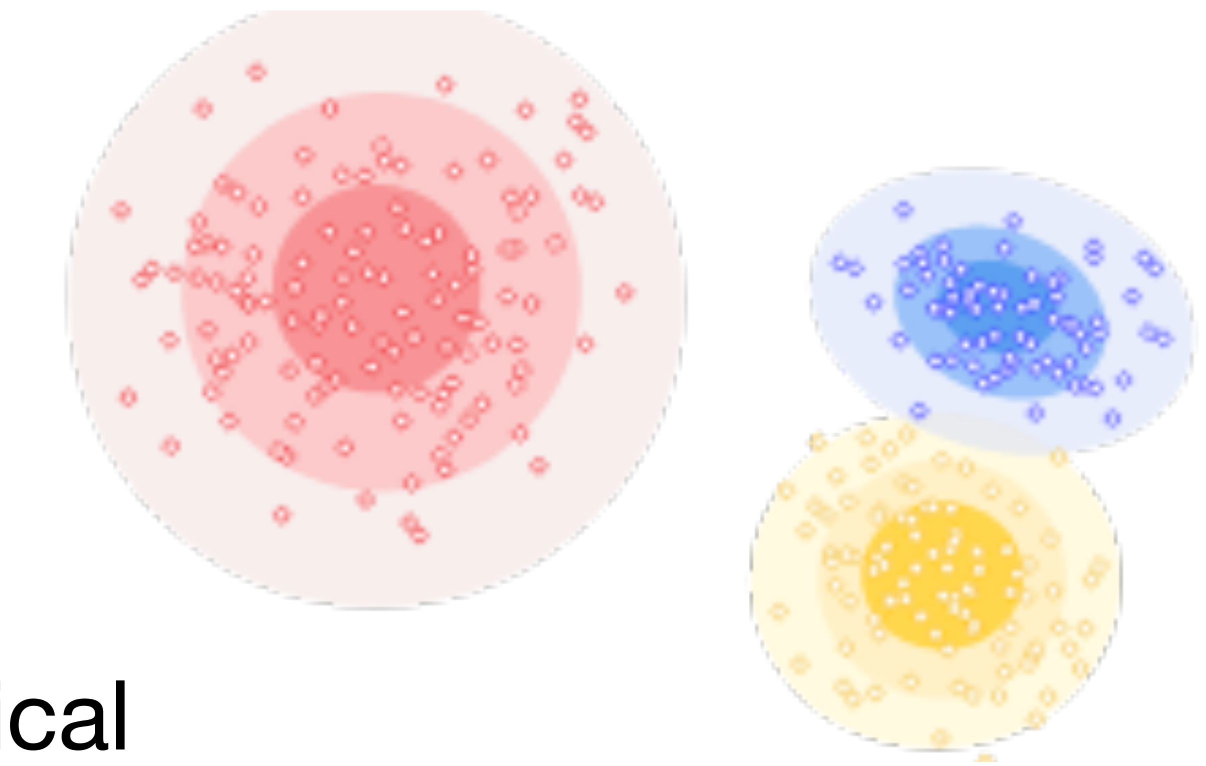
- Without supervised labels, the goal is to learn clusters based on similarity
- Types of clustering algorithms:
  - Centroid-based clustering
    - e.g., k-means
  - Distribution-based clustering
    - e.g., Gaussian mixture models
  - Hierarchical clustering
    - e.g., Agglomerative



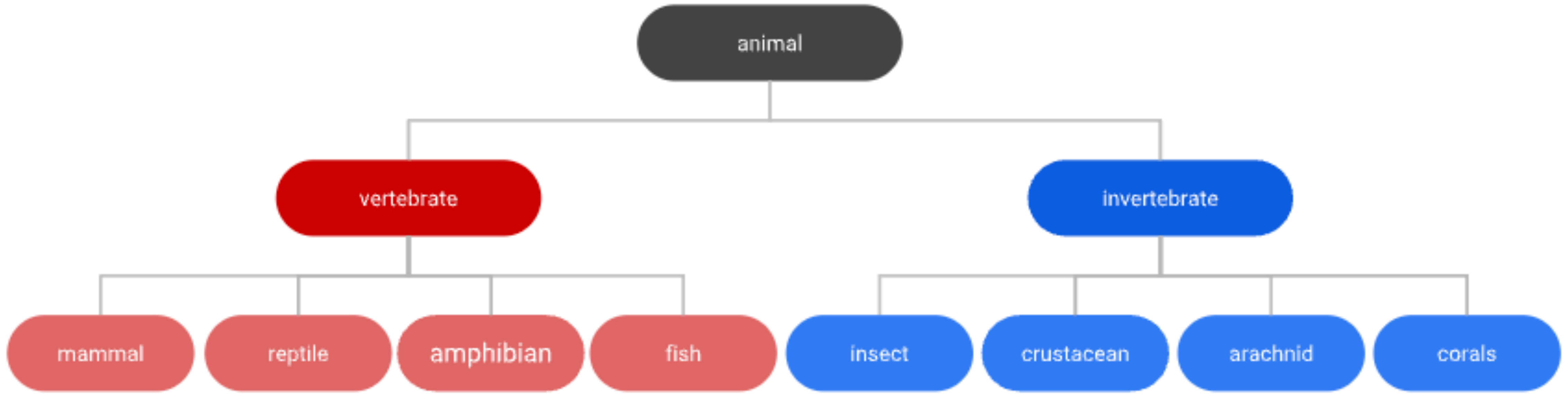
Centroid



Distribution



Hierarchical



k-means clustering

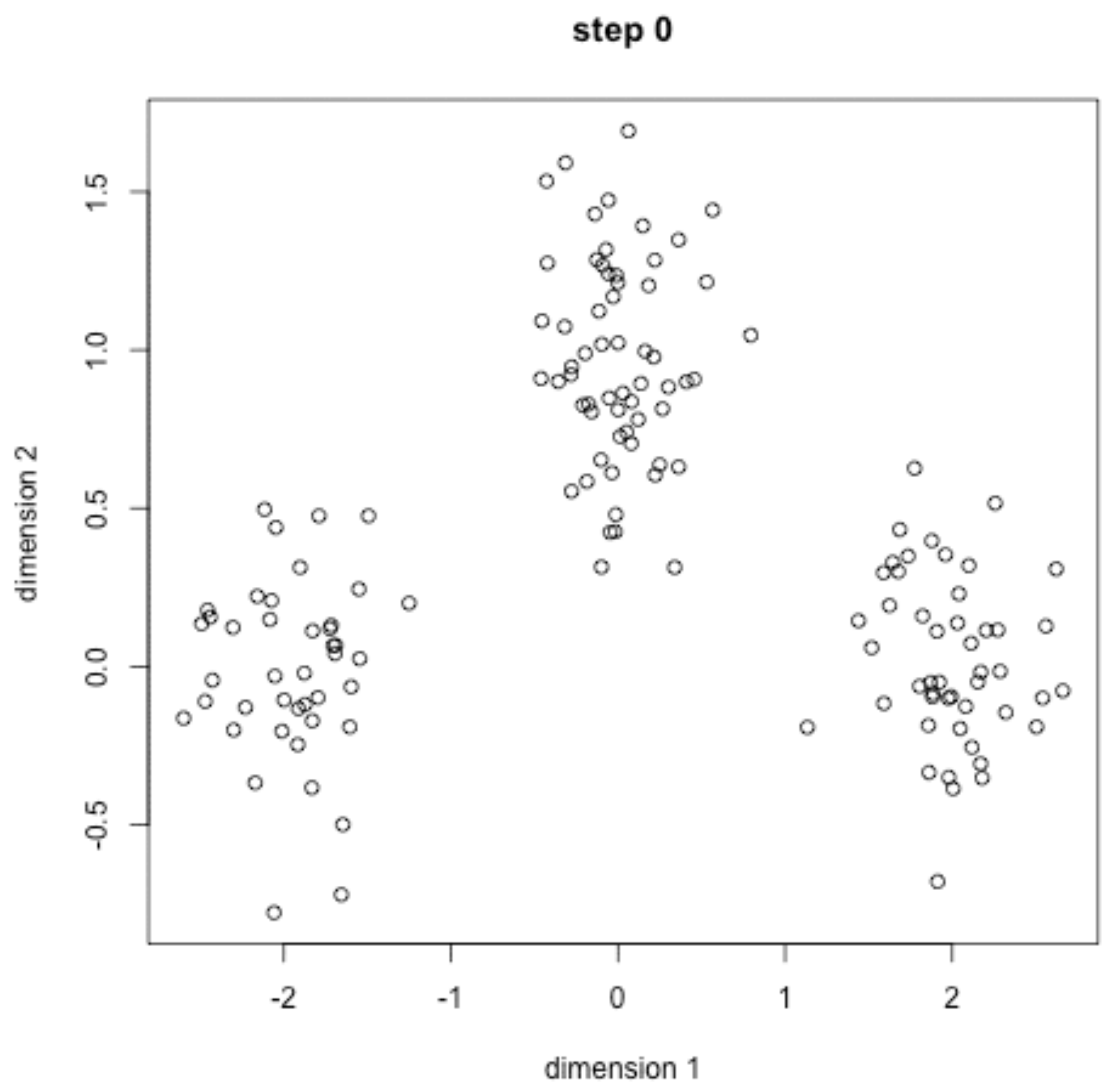
- Learn k centroids that minimize within-cluster variance

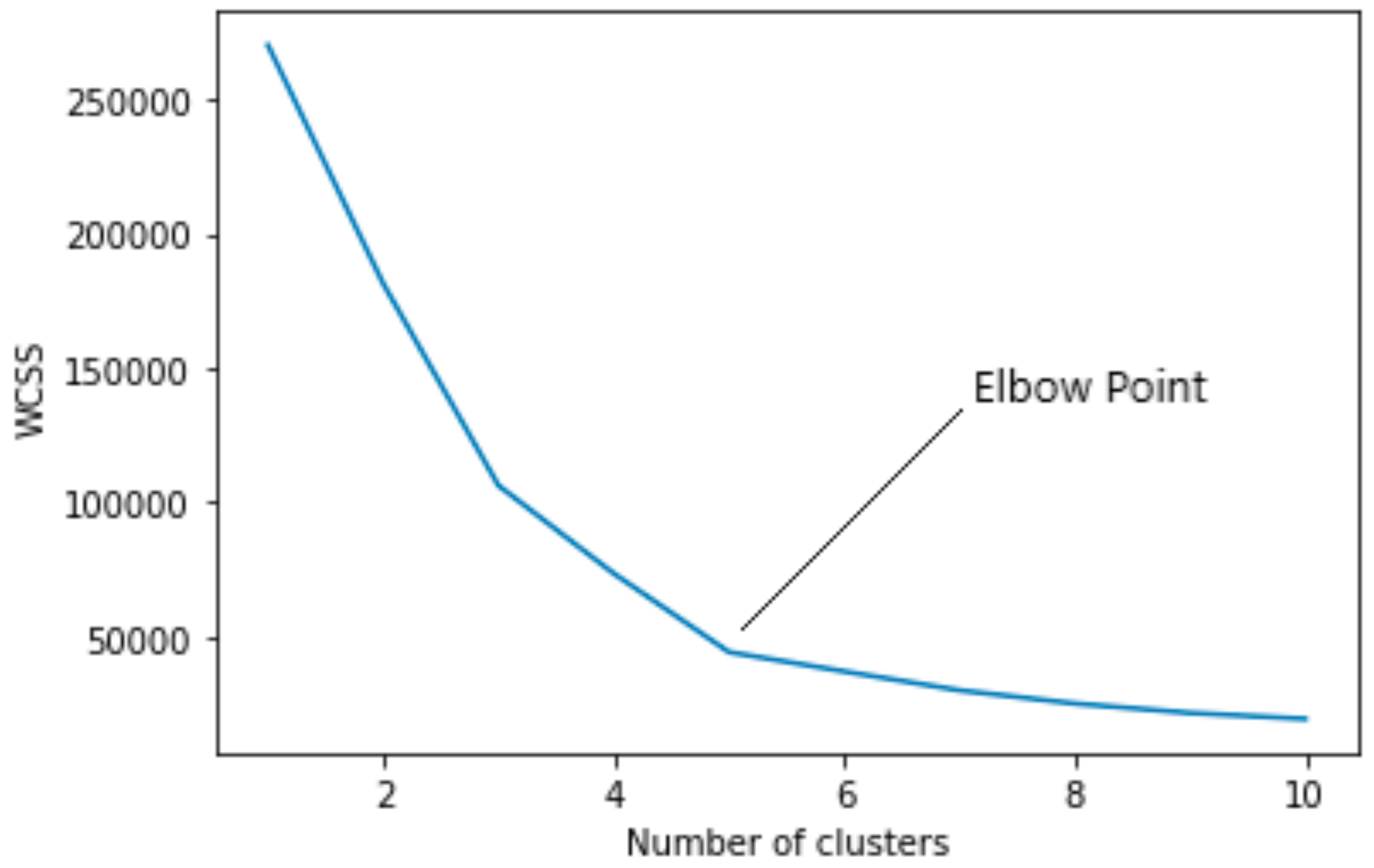
1. Pick the number of clusters k
2. Randomly select the centroid for each cluster
3. Assign all points to the closest centroid
4. Recompute centroid based on assigned points (i.e., mean)
5. Repeat until centroids do not change or max number of iterations reached

- How do we pick the number of clusters?
  Elbow method:
  - Within-cluster sum of squares (WCSS):
    for each cluster $c \in 1, \ldots, k$ compute the squared distance from each assigned datapoint x_i to the centroid μ_c

$$WCSS = \sum_{c}^{k} \sum_{i}^{m} (x_i - \mu_c)^2$$

  - Pick the number of clusters where WCSS begins to level off

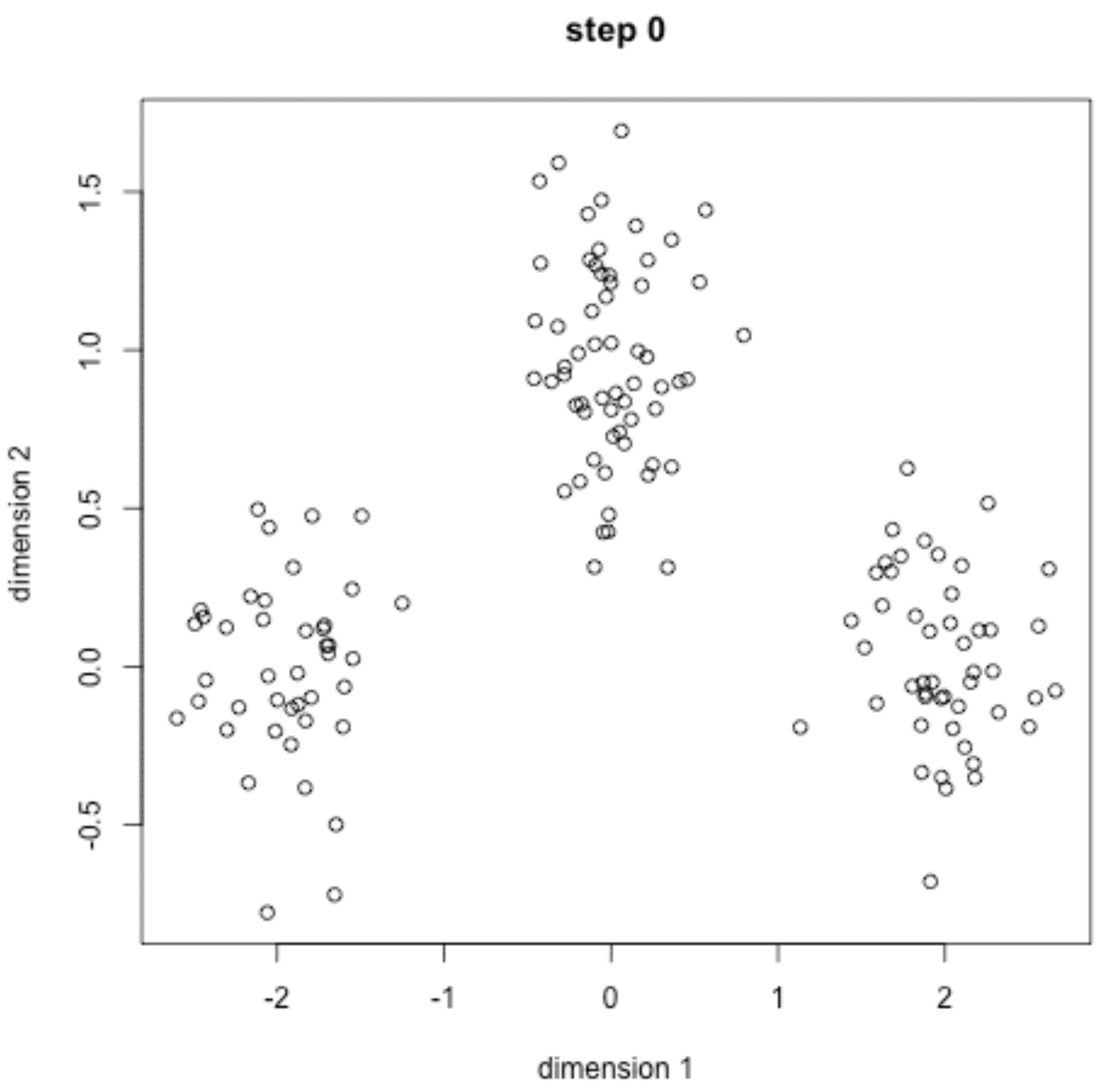




k-means clustering

- Learn k centroids that minimize within-cluster variance

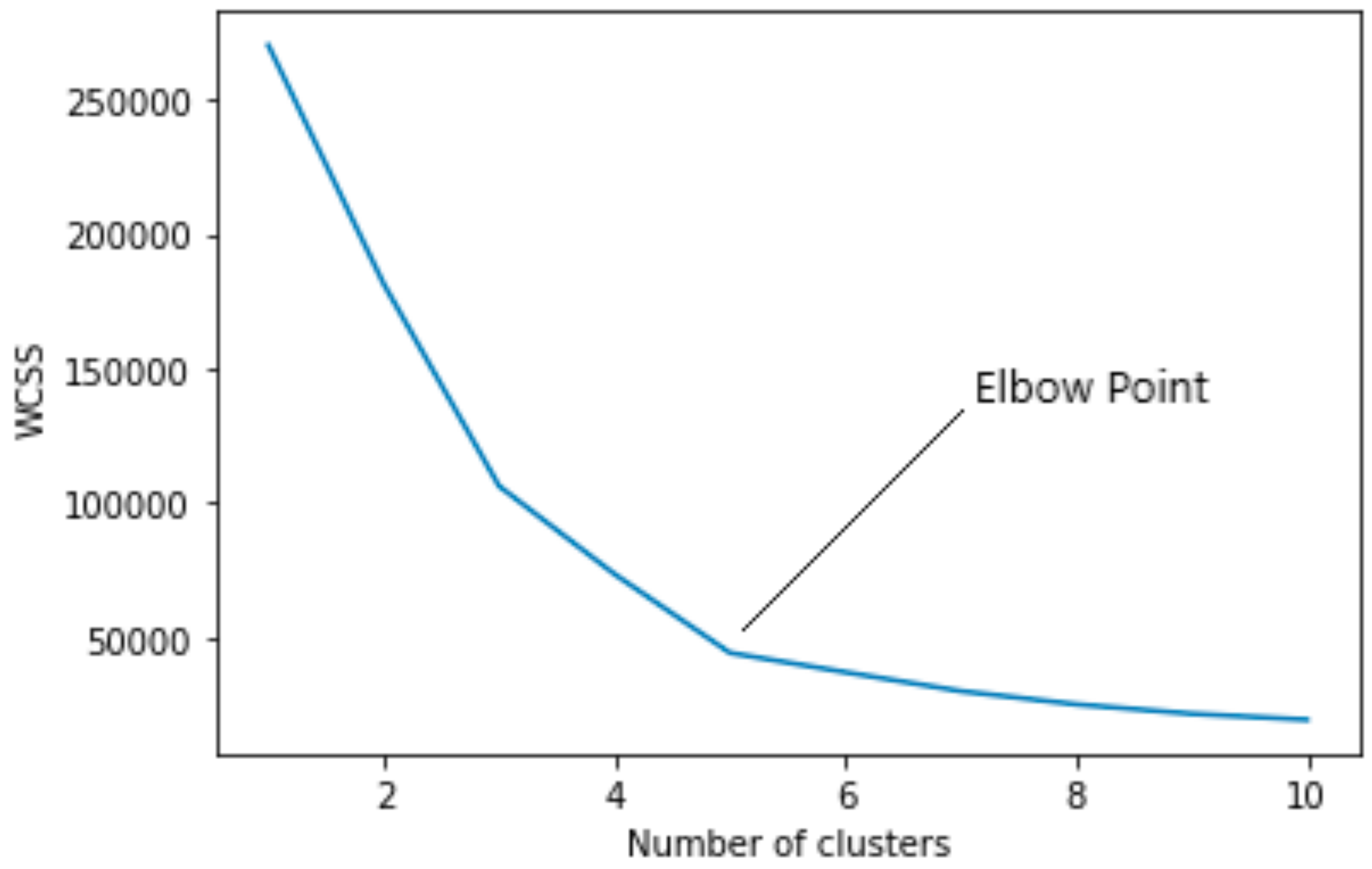
1. Pick the number of clusters k
2. Randomly select the centroid for each cluster
3. Assign all points to the closest centroid
4. Recompute centroid based on assigned points (i.e., mean)
5. Repeat until centroids do not change or max number of iterations reached



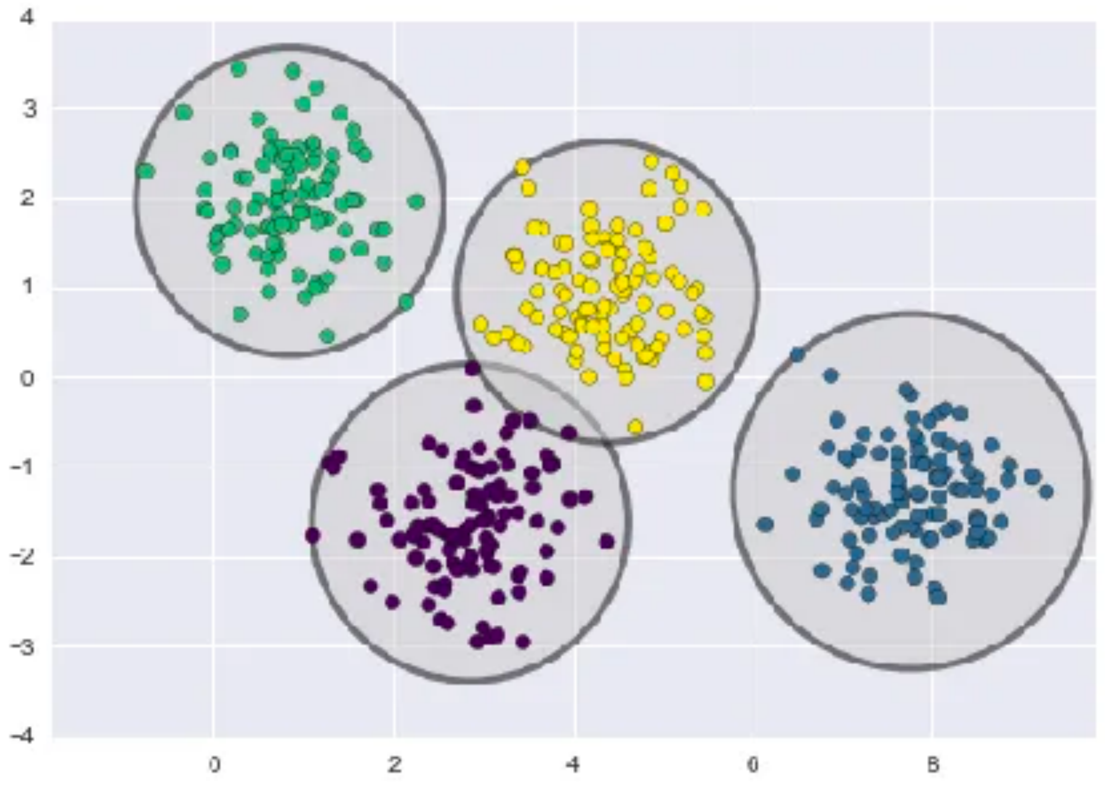
- How do we pick the number of clusters?
  Elbow method:
  - Within-cluster sum of squares (WCSS):
    for each cluster $c \in 1, \ldots, k$ compute the squared distance from each assigned datapoint x_i to the centroid μ_c

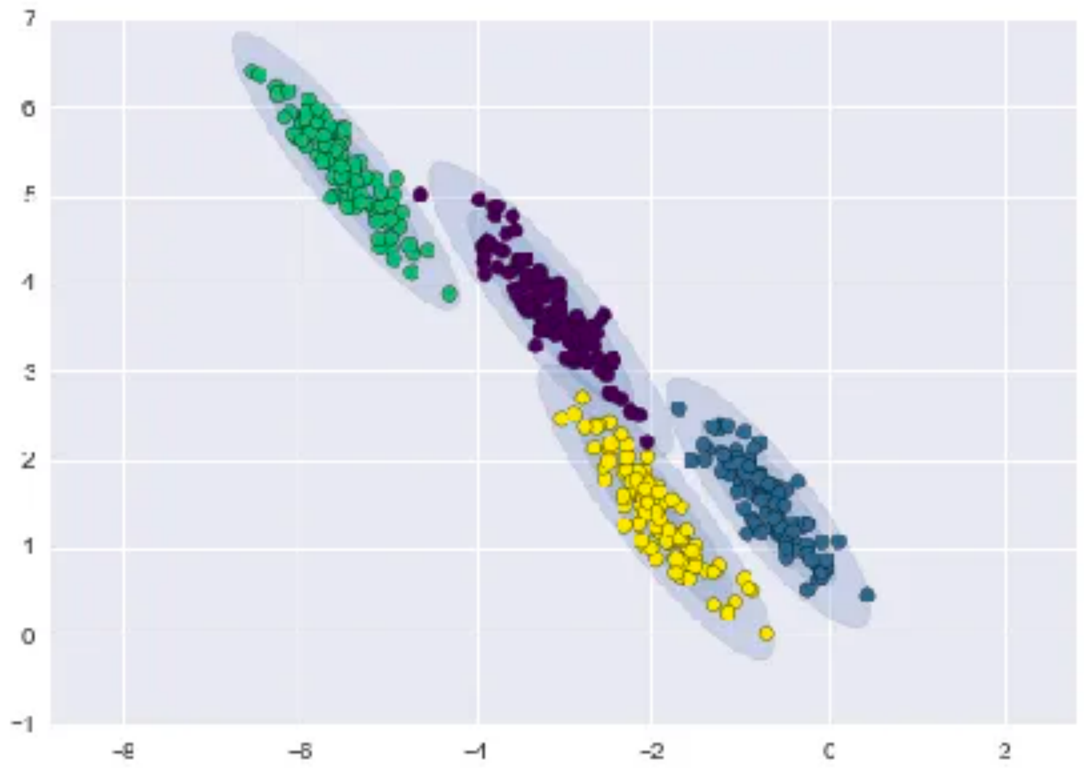
$$WCSS = \sum_{c}^{k} \sum_{i}^{m} (x_i - \mu_c)^2$$

  - Pick the number of clusters where WCSS begins to level off



Gaussian mixture models (GMMs)





- Instead of learning a centroid (prototype), where similarity is equivalent across feature dimensions…
- … learn a distribution for each cluster, where each feature dimension can have a different variance
- Assume data $\mathbf{x_i}$ is generated by a latent variable z_i in the form of a Gaussian distribution with unknown means μ_k and covariance Σ_k:

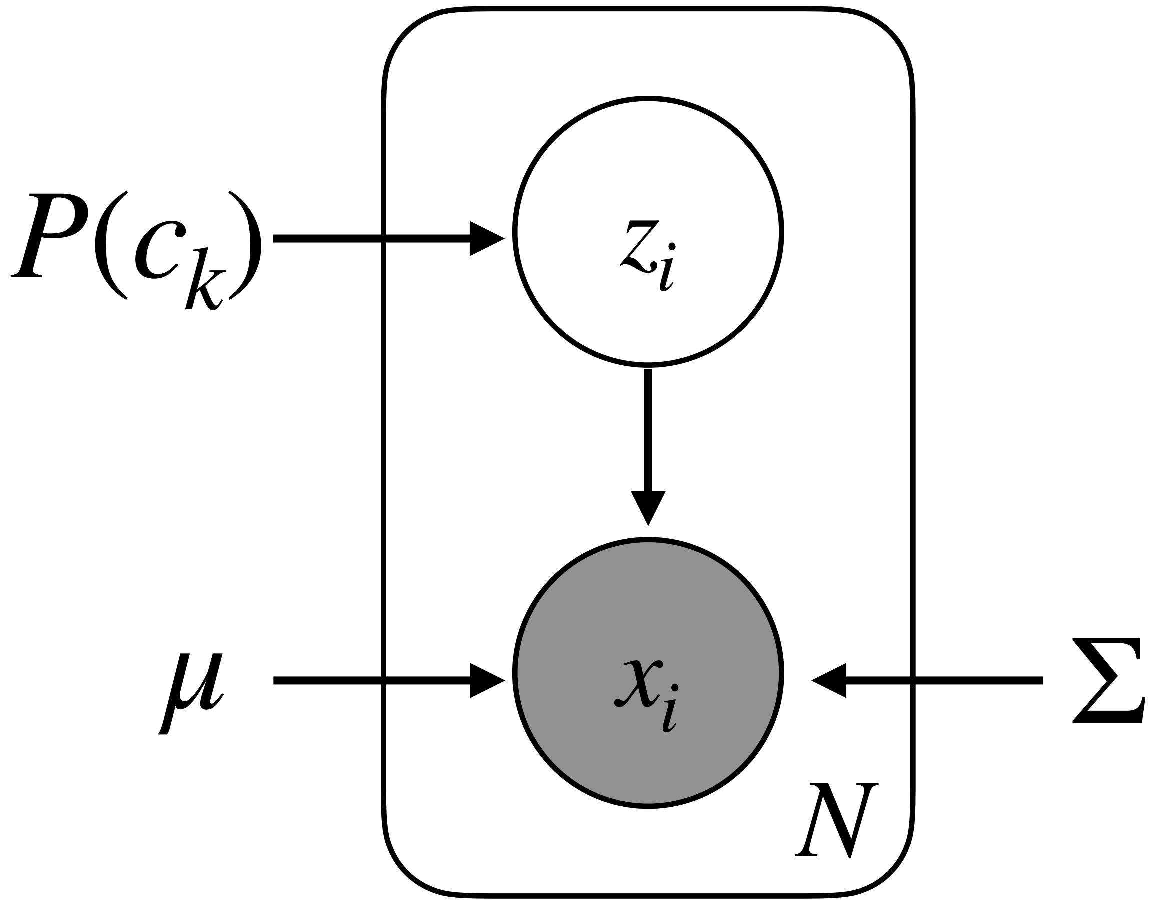
Gaussian likelihood Prior

$$p(\mathbf{x}_i) = \sum_k P(\mathbf{x}_i \mid z_i = k) P(z_i = k)$$

$$= \sum_k \mathcal{N}(\mathbf{x}_i \mid \mu_k, \Sigma_k) P(c_k)$$

$$\mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^d |\sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}_j - \mu_k)^\top \Sigma_k^{-1} (\mathbf{x}_j - \mu_k)\right)$$

Graphical representation



Expectation-Maximization (EM) algorithm

- Iterative method to compute a maximum likelihood when the data depends on latent variables
- **Expectation**: Compute "expected" classes for all data points, given current parameter values

$$P(\mathbf{x}_i = c_k) = \frac{\mathcal{N}(\mathbf{x}_i \mid \mu_k, \Sigma_k) P(c_k)}{\sum_j^K \mathcal{N}(\mathbf{x}_i \mid \mu_j, \Sigma_j) P(c_j)}$$

"Responsibility" of c_k for generating $\mathbf{x}_i$

- **Maximization**: Re-estimate parameters given current class assignments

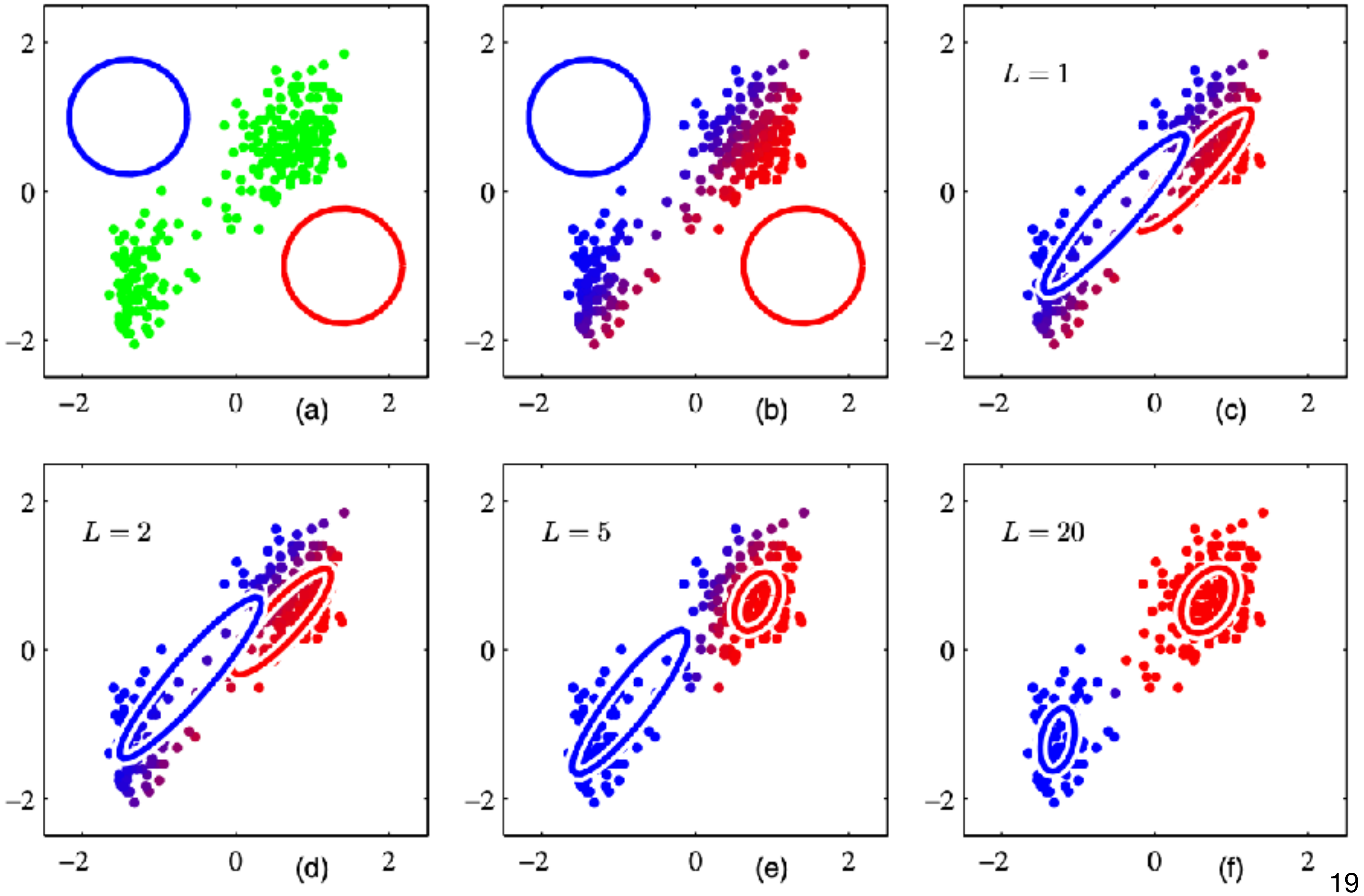
$$\mu_k^{\text{new}} = \frac{1}{N_k} \sum_i^N P(\mathbf{x}_i = c_k)\mathbf{x}_i$$ Centroid

Covariance

$$\Sigma_k^{\text{new}} = \frac{1}{N_k} \sum_i^N P(\mathbf{x}_i = c_k)(\mathbf{x}_i - \mu_k^{\text{new}})(\mathbf{x}_i - \mu_k^{\text{new}})^{\mathsf{T}}$$

$$P(c_k)^{\text{new}} = N_k / N$$ Prior on classes

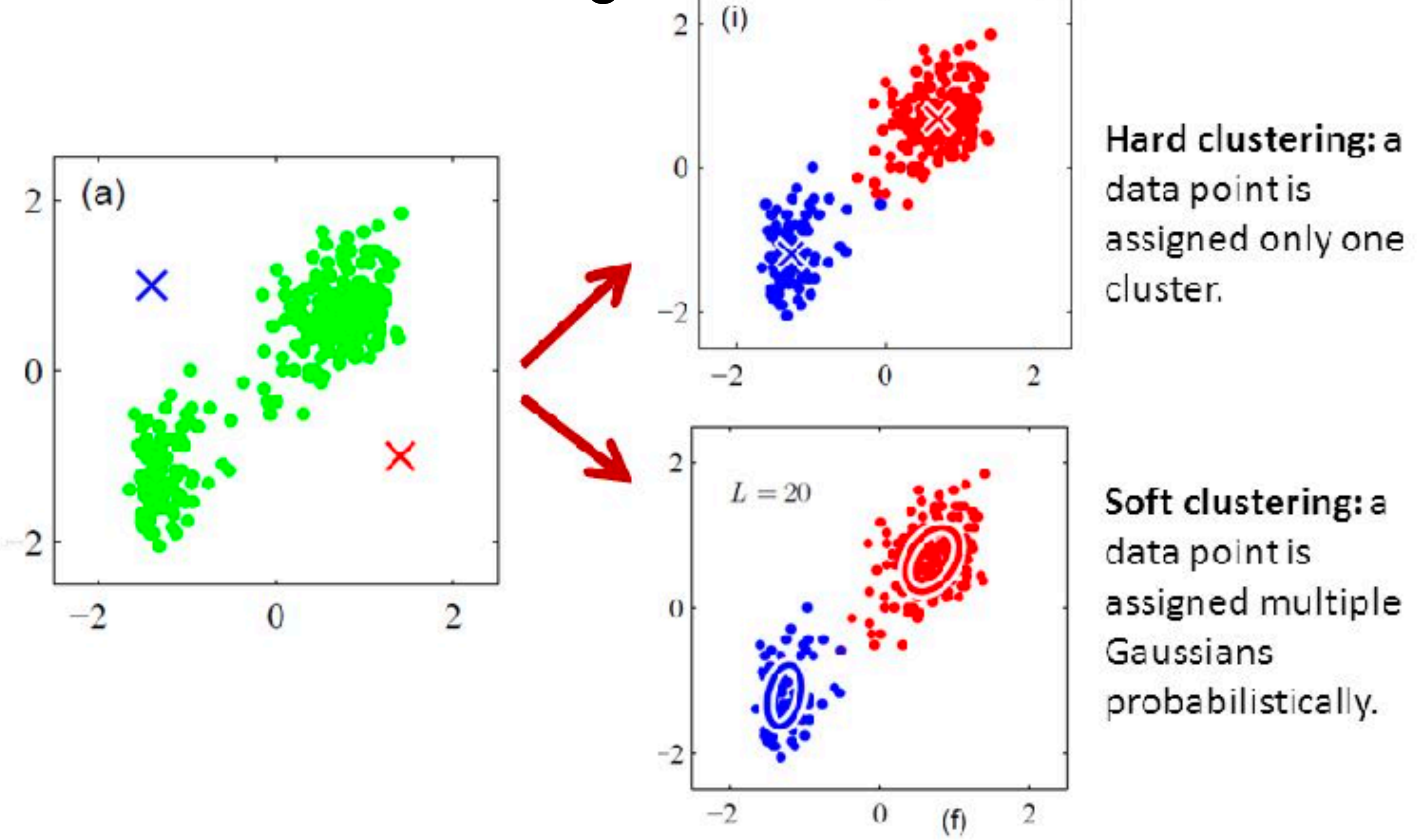
Bishop (2006)



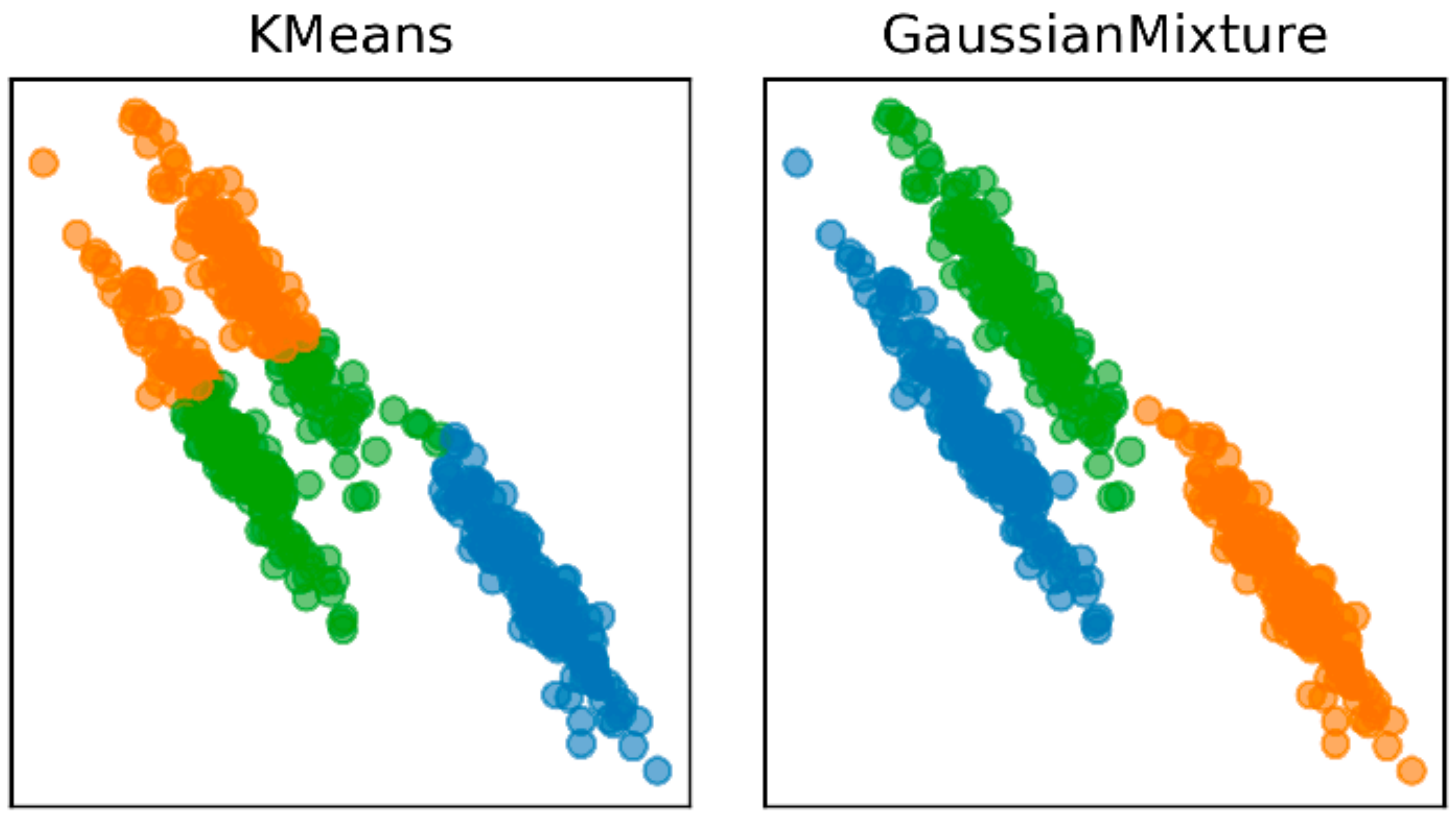
Summary of unsupervised learning

- Unsupervised learning is a clustering problem
- **k-means** performs “hard” assignment of data to clusters, with equivariant similarity across dimensions, and clusters defined by a centroid (prototype)
- **Gaussian mixture models** perform “soft” assignment, where learned covariance allows for skewed clusters, which are defined as a mixture of Gaussian densities (not exactly prototype or exemplar)

Hard vs. soft clustering



Skewed data



Data wrangling

- Beyond only implementing models, a big part of making ML work is data wrangling
  - **Feature scaling**
    - Min-Max normalization so feature values are in the range [0,1]

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

    - Standarization so feature values have mean = 0 and stdev = 1

$$X' = \frac{X - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation

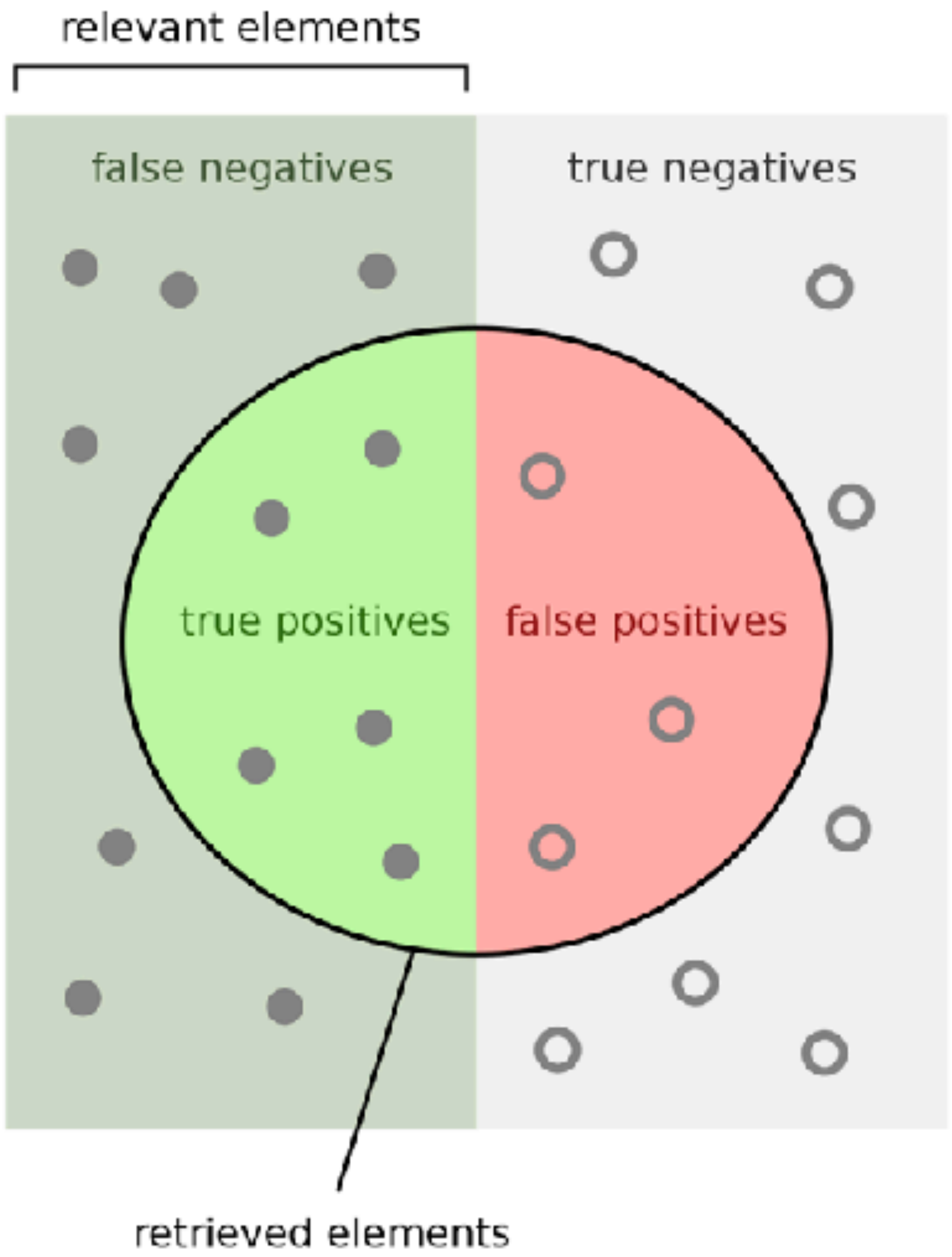
    - Normalization vs. standardization?
      - Normalization is useful when the distribution of the data is unknown or not Gaussian, since it retains the shape of the original distribution. However, it is sensitive to outliers
      - Standardization is useful when the data is Gaussian (but with enough data, everything becomes Gaussian) and is less sensitive to outliers. But may change the shape of the original distribution
  - **Feature engineering** by crafting new features
    - e.g., # of siblings/spouses in the titanic dataset combines two separate features
    - Requires some domain understanding

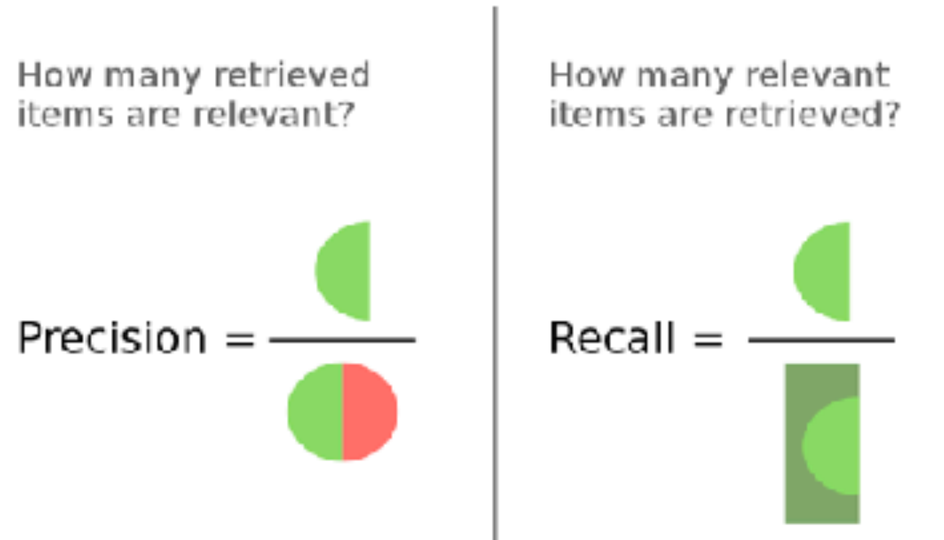
Assessing performance

- How to assess model performance?
- We need to balance both **precision** and **recall**

True Positives (TPs): 1	False Positives (FPs): 1
False Negatives (FNs): 8	True Negatives (TNs): 90

  - Precision is the proportion of items predicted TRUE that were actually true
    1 / 1+1 = 50%
  - Recall (also known as sensitivity) is the proportion of positives that were identified correctly
    1 / 1 + 8 = 11%
  - Precision and recall can be a tug-of-war based on how liberal or conservative your classification algorithm is
- **F1 score** is the harmonic mean of precision and recall
  F1 = (2 x .5 x .11)/(.5+.11) = 18%



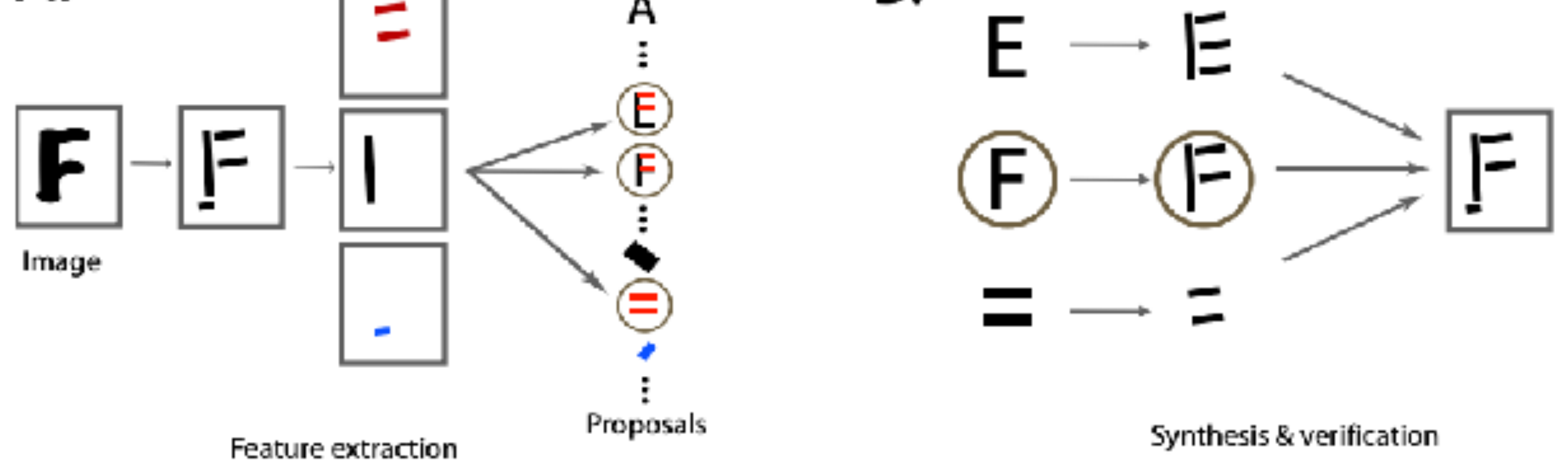


$$= \frac{TP}{TP + FP} \qquad = \frac{TP}{TP + FN}$$

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

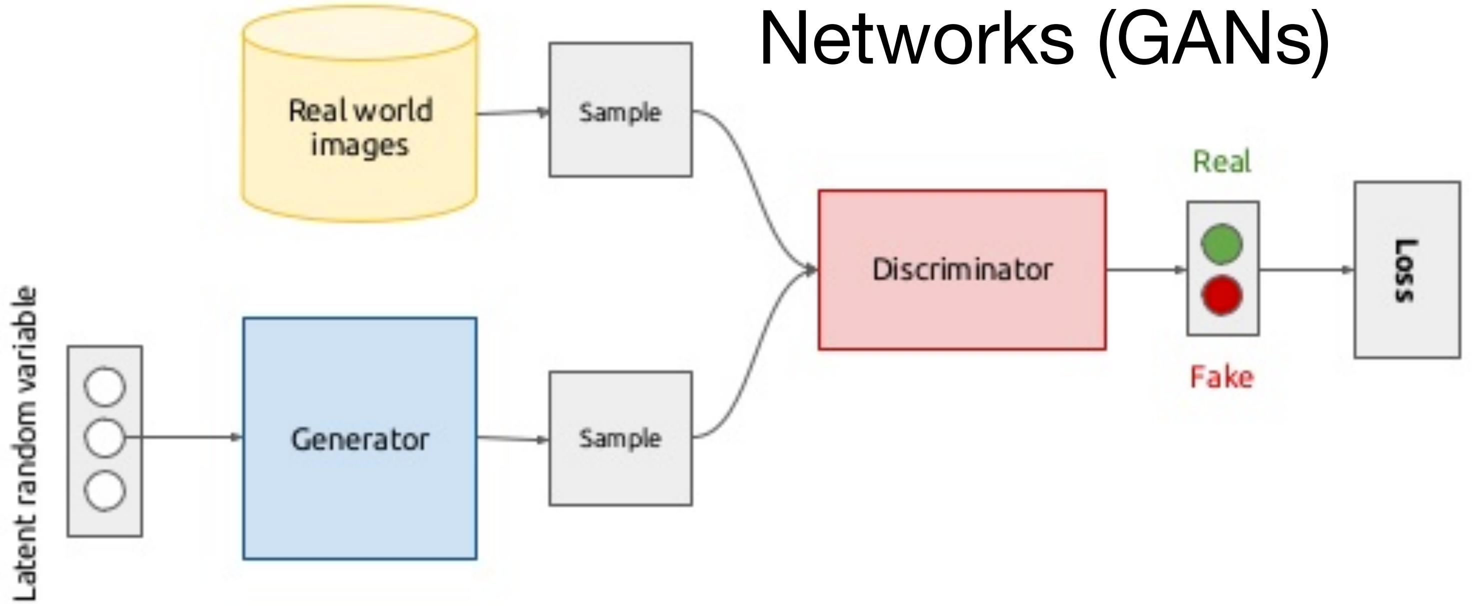
Discussion

- Both supervised and unsupervised learning methods provide tools for classifying data, with learned categories corresponding to:
    - Explicit category boundaries (decision trees, SVMs)
    - Implicit boundaries based on similarity of examples (MLPs)
    - Summary statistics of the data, based on a centroid (k-means) or a generative distribution (Naïve Bayes, GMM)
- **Discriminative** models simply learn to recognize the category labels, whereas **generative** models (Naïve Bayes, GMM) learn the data distribution and can be used to generate new datapoints consistent with each category
    - Many modern ML methods combine both (e.g., GANs)
        - Discriminative models are cheap to learn (e.g., deep neural networks), but require a lot of data
        - Generative models are more computationally costly, but can generate additional training data
        - Discriminative model provides an additional training signal to generative model
    - "Analysis by synthesis" (Yuille & Kersten, 2006) suggests humans do something similar
        - Interaction between top-down generative processes and bottom-up recognition



Yuille & Kersten (2006)

Generative Adversarial Networks (GANs)

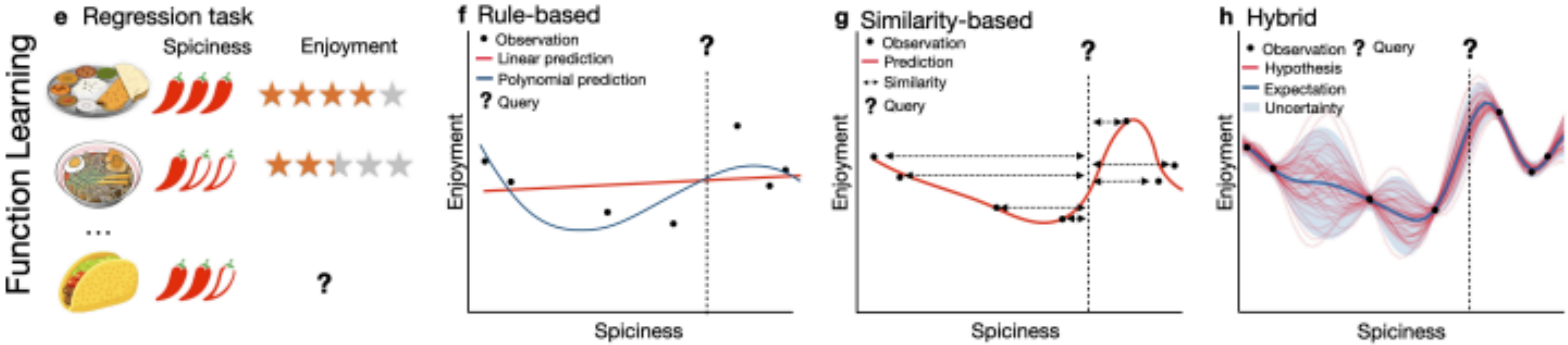


Bring your laptops for the tutorial on Friday

- We will provide 1 supervised and 1 unsupervised classification dataset
- Given the training data, implement one model of your choice
- We will provide code examples in Python and R for each model covered today
- Then, test your models on the test set. Best test performance on each dataset wins a small prize!
- We will use F1 score as the performance metric

Next week

*Function learning



*Note the change in topic and assigned reading. This is an in prep manuscript and I will send it via email