CS5489
Lecture 1.2: K-Nearest Neighbors and Bayes Optimal Classifier

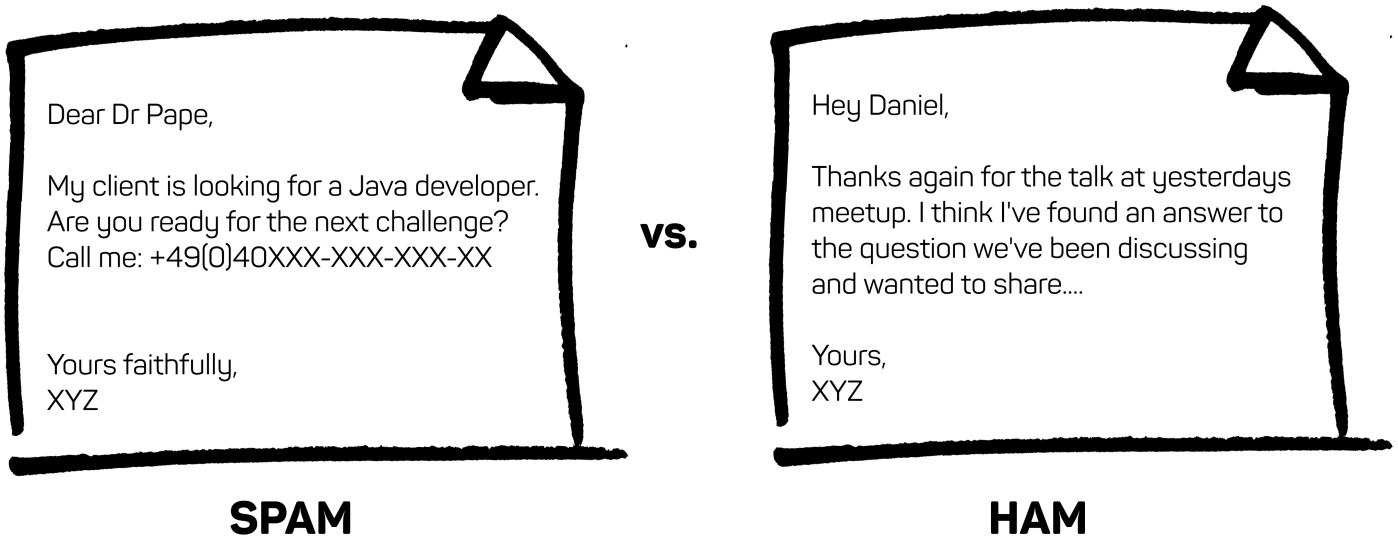
Kede Ma

City University of Hong Kong (Dongguan)

Slide template by courtesy of Benjamin M. Marlin

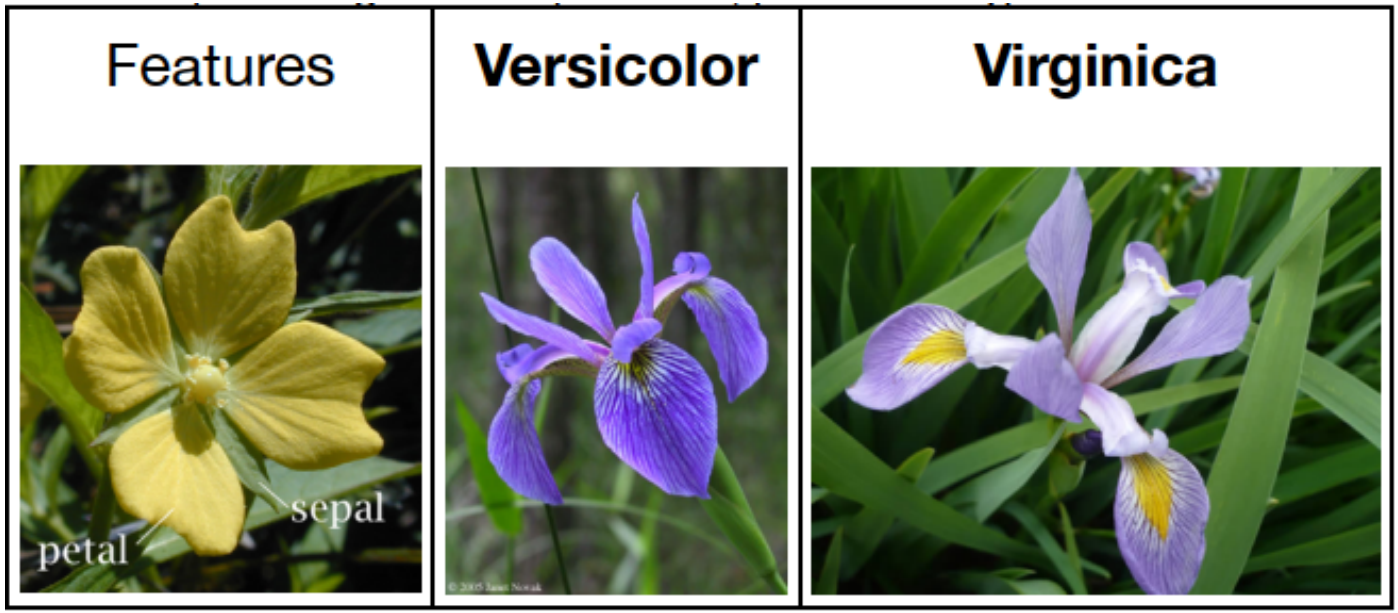
Example: Texts

- Given an email, predict whether it is spam or not spam

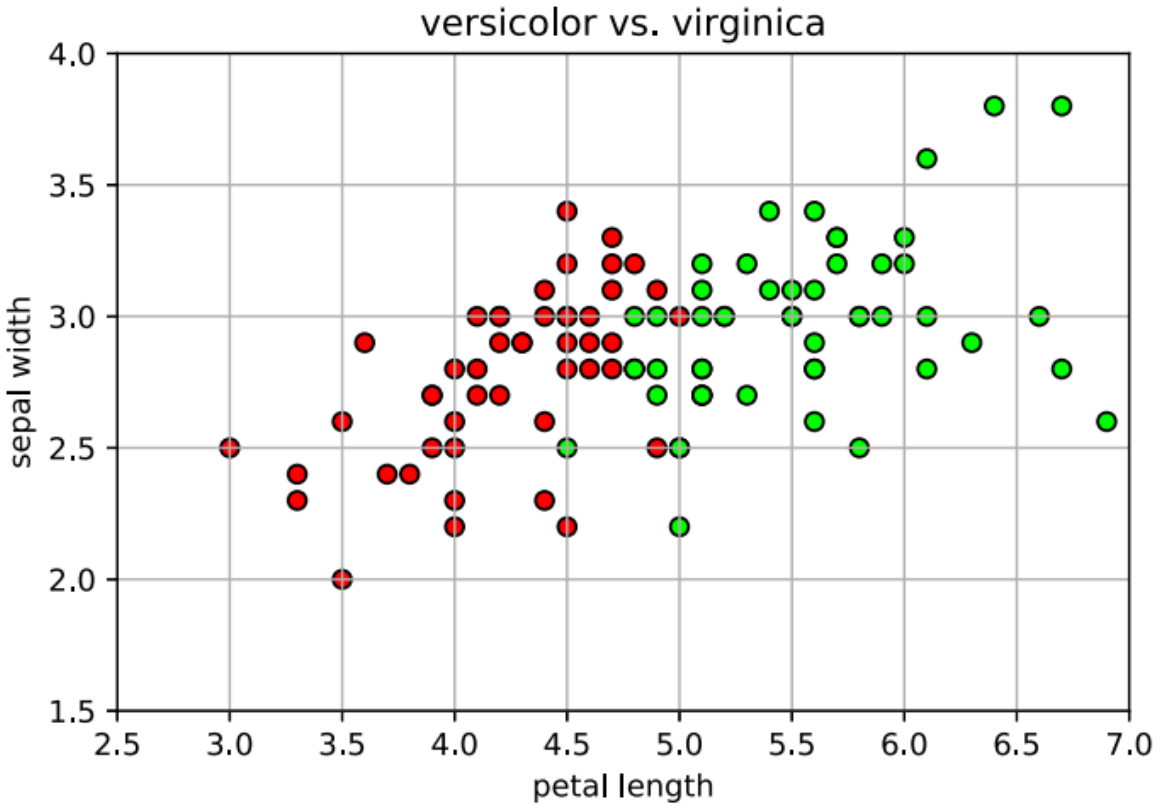


Example: Images

- Given the petal length and sepal width, predict the type of iris flower



Example: Images



The Classification Task

Definition: The Classification Task

Given a feature vector $\mathbf{x} \in \mathbb{R}^N$ that describes an object that belongs to one of C classes from the set $\mathcal{Y}$, predict which class the object belongs to

- In the previous example of iris classification
  - $\mathbf{x} = \begin{bmatrix} \text{petal length} \\ \text{sepal width} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$, where $N = 2$
  - $\mathcal{Y} = \{\text{"versicolor"}, \text{"virginica"}\}$, where $|\mathcal{Y}| = C = 2$
  - Or equivalently as numbers $\mathcal{Y} = \{1, 2\}$

The Classifier Learning Problem

Definition: Classifier Learning

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, M\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^N$ is a feature vector and $y^{(i)} \in \mathcal{Y} = \{1, \ldots, C\}$ is a class label, learn a function $f : \mathbb{R}^N \mapsto \mathcal{Y}$ that accurately predicts the class label y for any feature vector $\mathbf{x}$

Definition: Indicator Function

$$\mathbb{I}[A] = \begin{cases} 1, & \text{if event } A \text{ occurs} \\ 0, & \text{otherwise} \end{cases}$$

Classification Error and Accuracy

Definition: Classification Error Rate

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, M\}$ and a function $f : \mathbb{R}^N \mapsto \mathcal{Y}$, the classification error rate of f on $\mathcal{D}$ is

$$\text{Err}(f, \mathcal{D}) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}[y^{(i)} \neq f(\mathbf{x}^{(i)})]$$

Definition: Classification Accuracy Rate

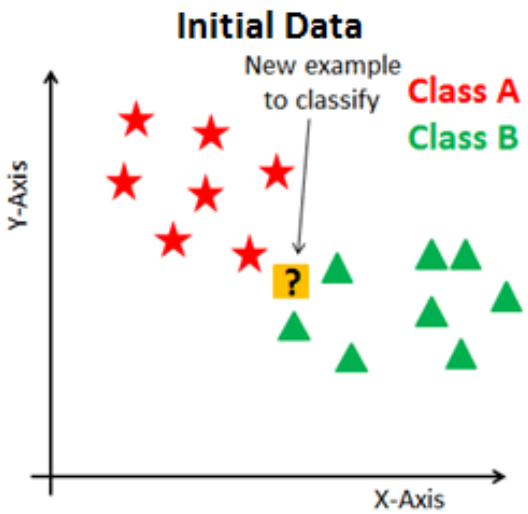
Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, M\}$ and a function $f : \mathbb{R}^N \mapsto \mathcal{Y}$, the classification accuracy rate of f on $\mathcal{D}$ is

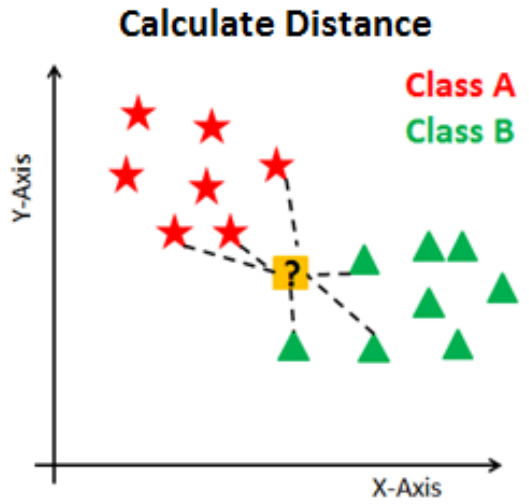
$$\text{Acc}(f, \mathcal{D}) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}[y^{(i)} = f(\mathbf{x}^{(i)})]$$

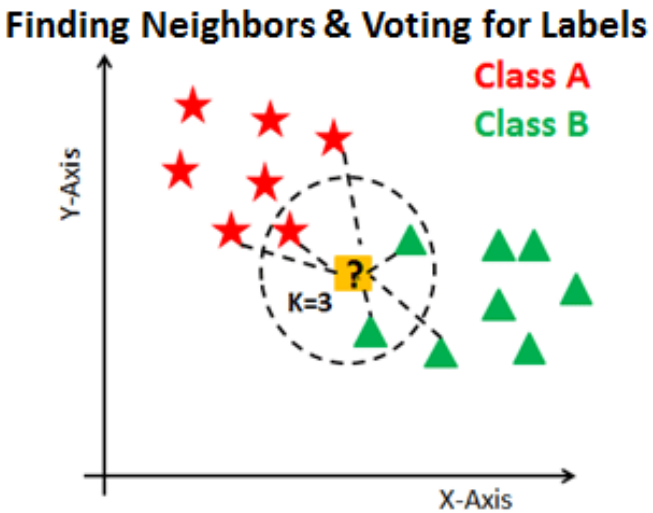
K-Nearest Neighbors (KNN) Classification

- The KNN classifier is one of the most basic yet essential classifier in machine learning
- It stores the training set $\mathcal{D}$, and classifies each **new** instance $\mathbf{x}$ using a majority vote over its K nearest neighbors $\mathcal{N}_K(\mathbf{x}) \subset \{1, \ldots, M\}$
- Use of KNN requires choosing the distance function $d : \mathbb{R}^N \times \mathbb{R}^N \mapsto \mathbb{R}$ and the number of neighbors K

Example: KNN


Initial Data
New example
to classify
Class A
Class B
Y-Axis
X-Axis
?


Calculate Distance
Class A
Class B
Y-Axis
X-Axis
?


Finding Neighbors & Voting for Labels
Class A
Class B
Y-Axis
X-Axis
?
K=3

Max vs. Arg Max

Max Operator

The max operator of a function $f(\mathbf{x})$ defined over $\mathbf{x} \in \mathcal{D}$ returns the maximum value of the function:

$$\max_{\mathbf{x}} f(\mathbf{x}) = \{f(\mathbf{x}) | \mathbf{x} \in \mathcal{D} \wedge \forall \mathbf{y} \in \mathcal{D}, f(\mathbf{y}) \leq f(\mathbf{x})\}$$

Arg Max Operator

The arg max operator of a function $f(\mathbf{x})$ defined over $\mathbf{x} \in \mathcal{D}$ gives the set of points for which $f(\mathbf{x})$ reaches the maximum value:

$$\arg\max_{\mathbf{x}} f(\mathbf{x}) = \{\mathbf{x} | \mathbf{x} \in \mathcal{D} \wedge \forall \mathbf{y} \in \mathcal{D}, f(\mathbf{y}) \leq f(\mathbf{x})\}$$

KNN Classification

KNN Classification Function

$$f_{\text{KNN}}(\mathbf{x}) = \arg\max_{c \in \{1,\ldots,C\}} \sum_{i \in \mathcal{N}_K(\mathbf{x})} \mathbb{I}[y^{(i)} = c]$$

Distance Function

- In general, KNN can work with any distance function d satisfying non-negativity $d(\mathbf{x}, \mathbf{x}') \geq 0$ and identity of indiscernibles $d(\mathbf{x}, \mathbf{x}) = 0$
- Generally, the more structure the distance function has (symmetry, triangle inequality), the more structure you can exploit when designing algorithms

Distance Metrics

Definition: Minkowski Distance (ℓ_p-norm)

Given two data vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$, the Minkowski distance with parameter p (the ℓ_p-norm) is a proper metric defined as follows:

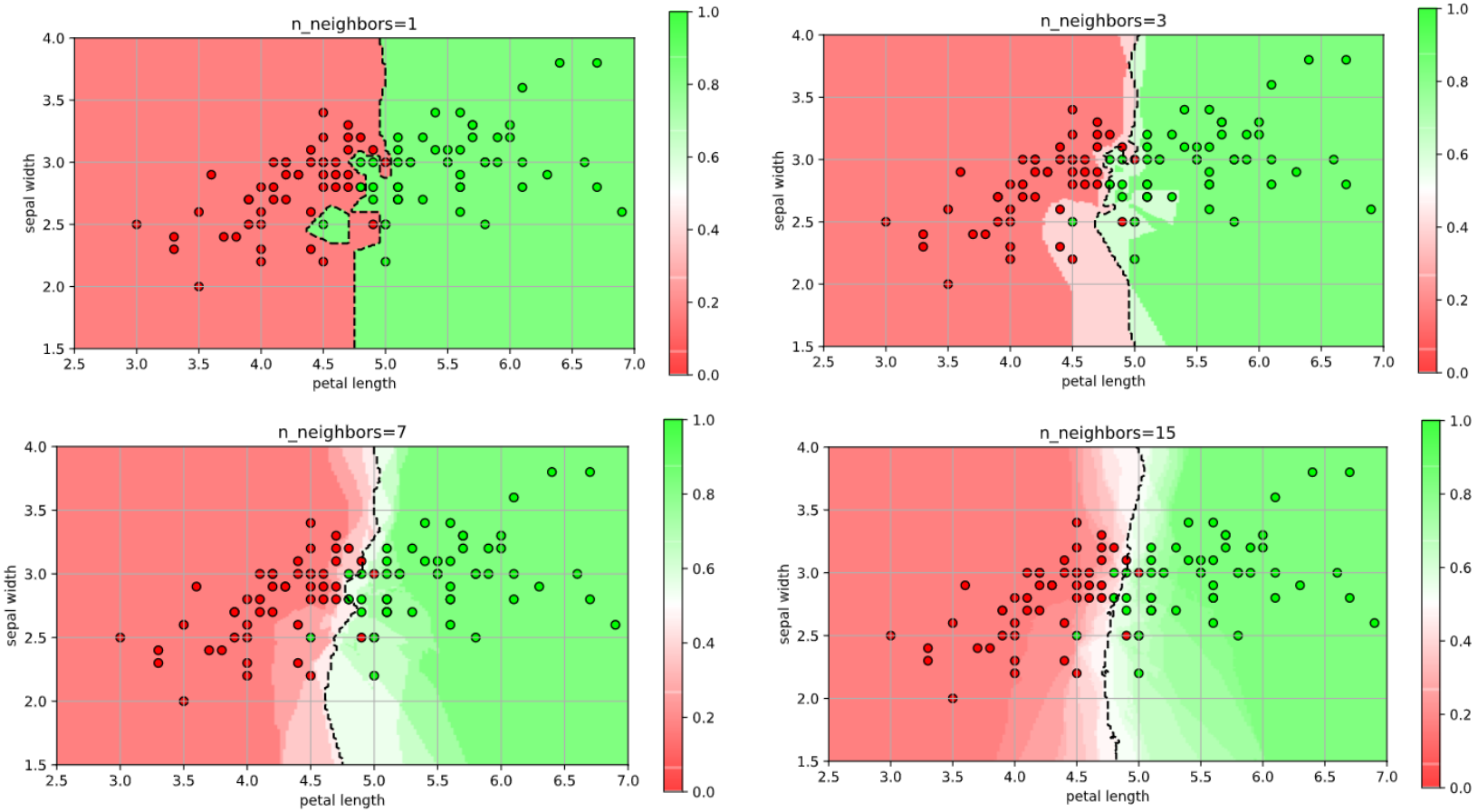
$$\begin{aligned} d_p(\mathbf{x}, \mathbf{x}') &= \|\mathbf{x} - \mathbf{x}'\|_p \\ &= \left(\sum_{j=1}^{N} |x_j - x'_j|^p \right)^{1/p} \end{aligned}$$

Special cases include Euclidean distance ($p = 2$), Manhattan distance ($p = 1$) and Chebyshev distance ($p = \infty$)

Brute Force KNN

- Given any distance function d, brute force KNN works by computing the distance $d_i = d(\mathbf{x}^{(i)}, \mathbf{x})$ from a target point $\mathbf{x}$ to all of the training points $\mathbf{x}^{(i)}$
- Sort the distances $\{d_i, i = 1, \ldots, M\}$ and choose the data cases with the K smallest distances to form the neighbor **index** set $\mathcal{N}_K(\mathbf{x})$
- Once the K neighbors are selected, applying the classification rule is easy

KNN on Iris Dataset



KNN Variants

- Instead of giving all of the K neighbors equal weight in the majority vote, a distance-weighted majority can be used:

$$f_{\text{KNN}}(\mathbf{x}) = \arg\max_{c \in \{1,\ldots,C\}} \frac{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i \mathbb{I}[y^{(i)} = c]}{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i},$$
$$w_i = \exp(-\alpha d_i)$$

- Instead of a brute force nearest neighbor search, data structures like K-d trees can be constructed over the training data that support nearest neighbor search with lower computational complexity

KNN Trade-Offs

- **Advantages**:
  - No training period is involved (i.e., lazy learning), and new data can be added seamlessly without re-training the model
  - Converges to the correct decision surface as data goes to infinity
- **Disadvantages**:
  - Does not work well with large datasets. Since KNN needs to store all training data, performing neighbor search requires a lot of memory and takes a lot of time
  - Does not work well with high dimensions. It becomes difficult for KNN to calculate the distance in each dimension. Moreover, everything is far from everything else in high dimensions (the so-called “curse of dimensionality”)

Probabilistic Classifiers

- One type of classifier to model the data
- Model how the data is generated using probability distributions (i.e., **generative** models)
- Defining generative models:
  - The world has objects /patterns of various classes
  - The observer measures features/observations from the objects/patterns
  - Each class of objects/patterns has a particular (and possibly different) distribution of features

Class Model

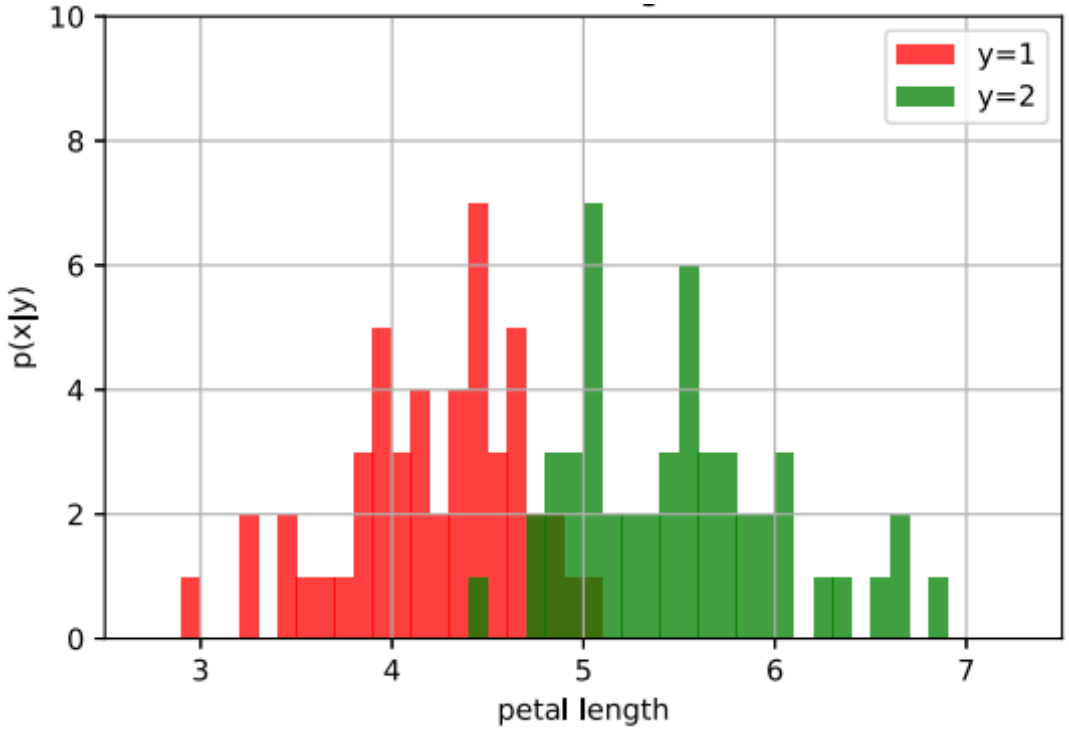
- Possible classes are $\mathcal{Y}$
  - For example, in the iris dataset, $\mathcal{Y} = \{\text{“versicolor”}, \text{“virginica”}\}$, or more generally, $\mathcal{Y} = \{1, 2\}$
- In the real world, the frequency that Class y occurs is given by the probability distribution $p(y)$
  - $p(y)$ is called the **prior distribution**
- Example:
  - $p(y = 1) = 0.4$ and $p(y = 2) = 0.6$
  - This means in the world of iris flowers, there are 40% that are Class 1 (versicolor) and 60% that are Class 2 (virginica)
- Learn from our data:

$$p(y = c) = \frac{\sum_{i=1}^{M} \mathbb{I}[y^{(i)} = c]}{M}, \quad c \in \{1, 2\}$$

Observation Model

- We measure/observe feature vector $\mathbf{x}$
  - The values of the features **depend** on the class
- The observation is drawn according to the distribution $p(\mathbf{x}|y)$
  - $p(\mathbf{x}|y)$ is called the **class conditional distribution**
  - It indicates the probability of observing a particular feature vector $\mathbf{x}$ given the object is Class y
- Learn from our data:
  - In the iris dataset, we draw histograms of feature "petal length" for each class

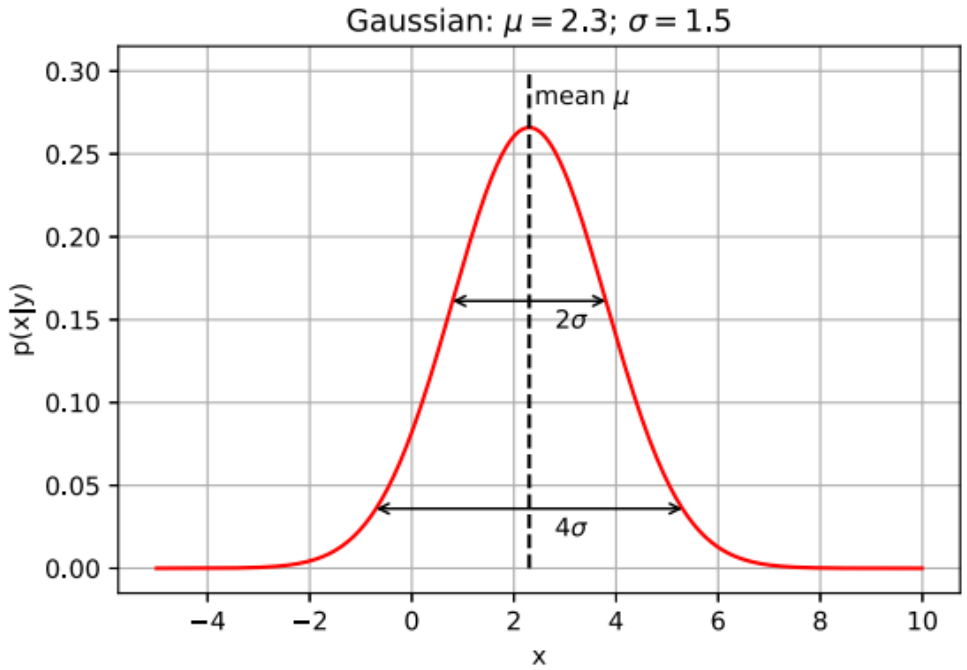
Feature Histogram



- Problem: looks a little bit noisy
- Solution: assume a probability model for the class conditional $p(\mathbf{x}|y)$

Gaussian Distribution

- Each class is modeled as a separate Gaussian distribution of the feature value
  - $p(x|y=c;\mu_c,\sigma_c^2) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{1}{2\sigma_c^2}(x-\mu_c)^2}$
  - Each class has its own mean and variance parameters (μ_c, σ_c^2)



Learn the Parameters from Data

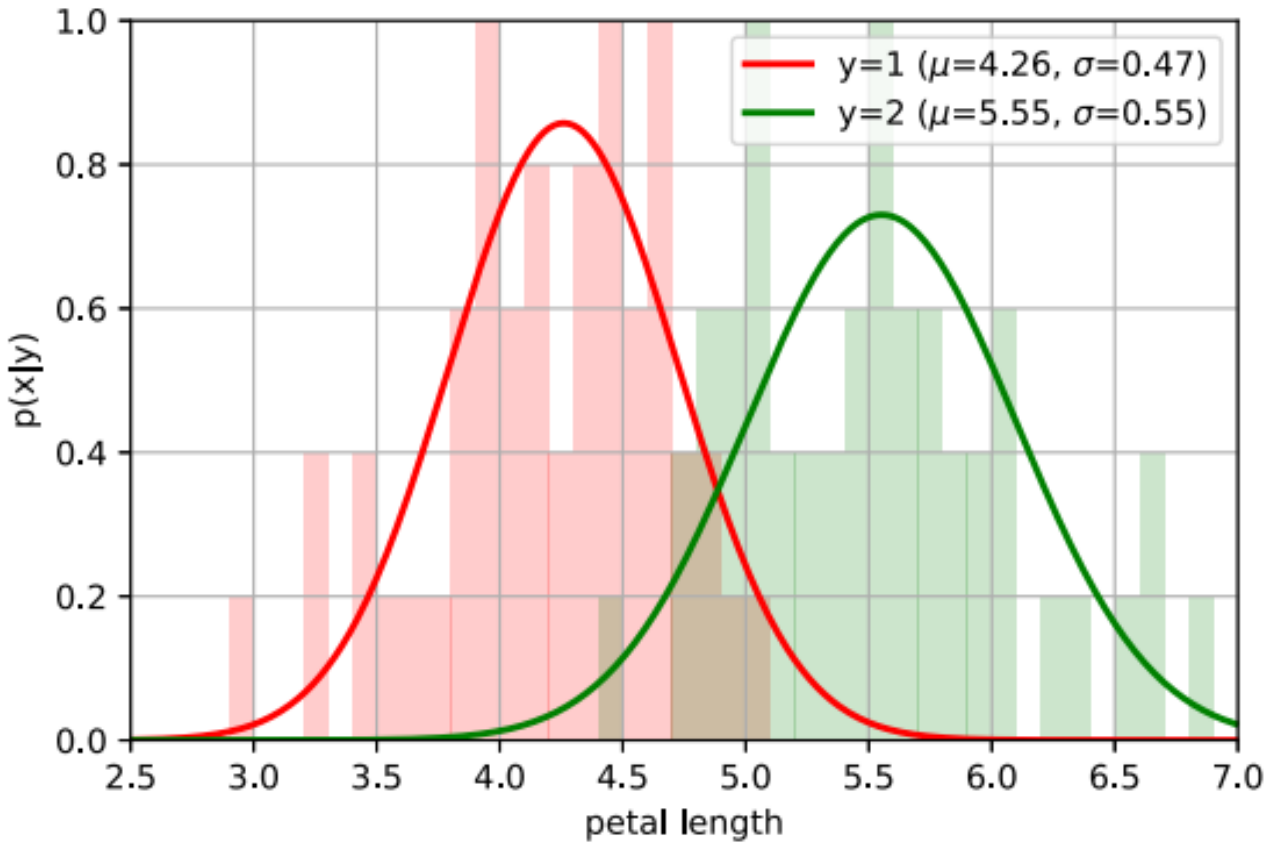
- Maximum likelihood estimation (MLE)
  - Set the parameters (μ_c, σ_c^2) to maximize the likelihood (probability) of the samples for Class c
  - Let $\mathcal{D}_c = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{M_c}$ be the data for Class c (i.e., $y^{(i)} = c$):

$$(\hat{\mu}_c, \hat{\sigma}_c^2) = \arg\max_{\mu_c, \sigma_c^2} \sum_{i=1}^{M_c} \log p(x^{(i)} | y^{(i)}; \mu_c, \sigma_c^2)$$

  - When we view the above objective as a function of the parameters $\{\mu_c, \sigma_c^2\}$, we instead call it the likelihood function of the data

- Solution:
  - Sample mean: $\hat{\mu}_c = \frac{1}{M_c} \sum_{i=1}^{M_c} x^{(i)}$
  - Sample variance: $\hat{\sigma}_c^2 = \frac{1}{M_c} \sum_{i=1}^{M_c} (x^{(i)} - \hat{\mu}_c)^2$

Gaussian Class Conditionals



Bayesian Decision Rule

- The Bayesian decision rule (BDR) makes the **optimal** decisions on problems involving probability (uncertainty)
  - Minimizes the probability of making a prediction error
- Bayes optimal classifier
  - Given observation $\mathbf{x}$, pick the class c with the **largest posterior probability** $p(y = c|\mathbf{x})$:

$$f_{\mathrm{B}}(\mathbf{x}) = \underset{c \in \{1,\ldots,C\}}{\arg\max}\, p(y = c|\mathbf{x})$$

- Example:
  - If $p(y = 1|\mathbf{x}) > p(y = 2|\mathbf{x})$, then choose Class 1
  - If $p(y = 1|\mathbf{x}) < p(y = 2|\mathbf{x})$, then choose Class 2
- Problem: we don't have $p(y|\mathbf{x})$
  - We only have $p(y)$ and $p(\mathbf{x}|y)$

Bayes' Rule

- The posterior probability can be calculated using Bayes' rule:

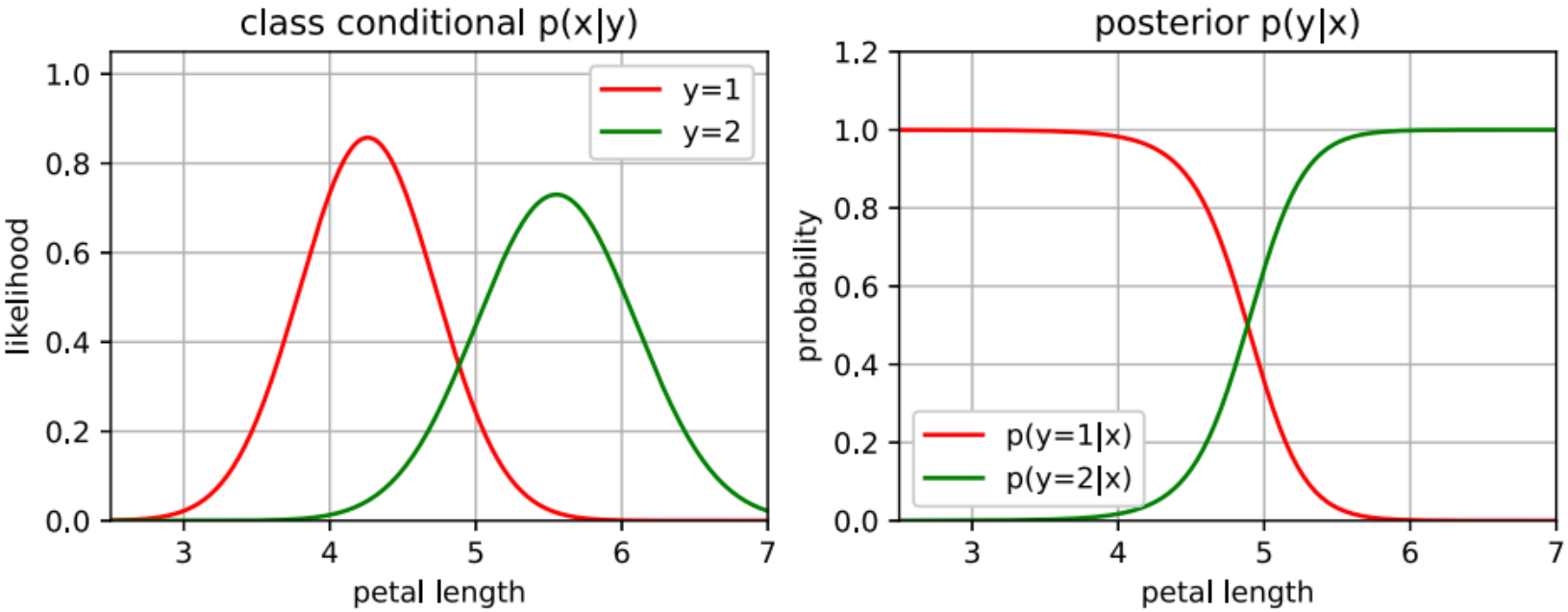
$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

  - The denominator is the probability of $\mathbf{x}$:
    - $p(\mathbf{x}) = \sum_{y \in \mathcal{Y}} p(\mathbf{x}, y) = \sum_{y \in \mathcal{Y}} p(\mathbf{x}|y)p(y)$
  - The denominator makes $p(y|\mathbf{x})$ sum to 1

- Bayes' rule:

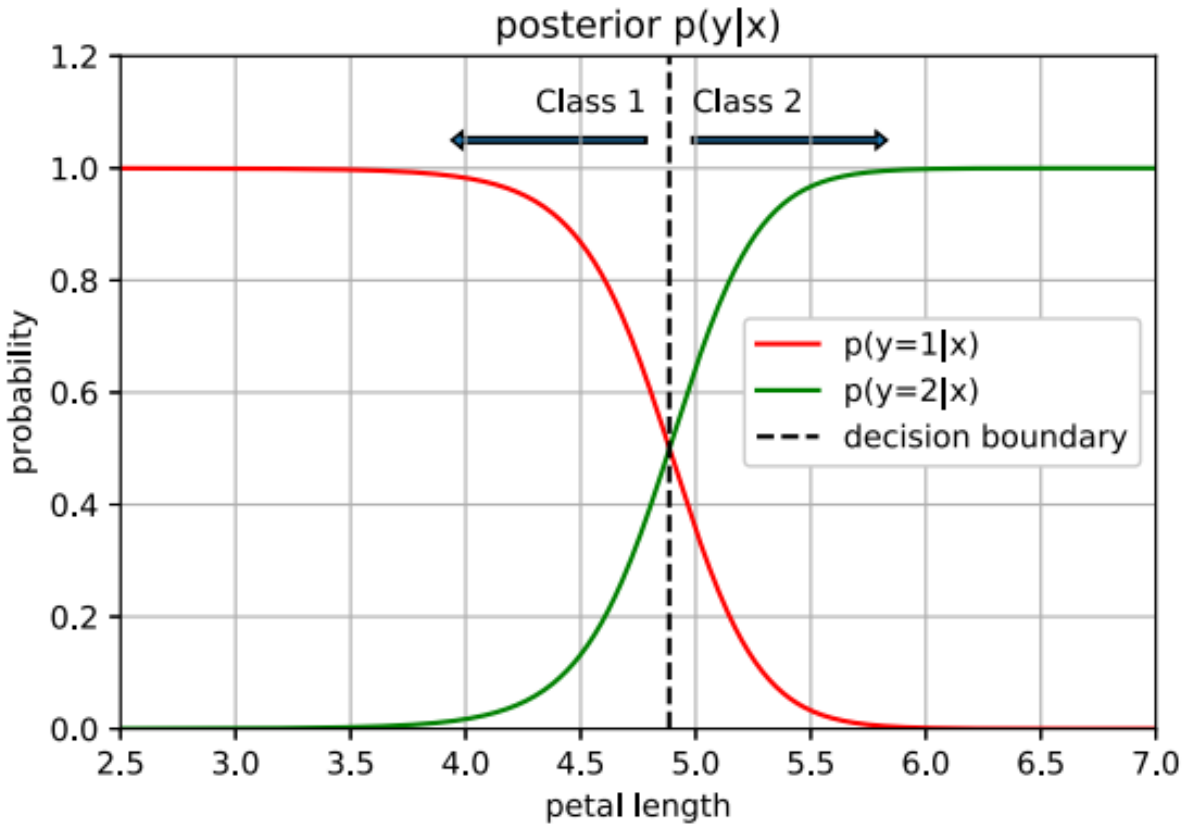
$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x}|y=1)p(y=1) + p(\mathbf{x}|y=2)p(y=2)}$$

Posterior Probability



Decision Boundary

- The decision boundary is where the two posterior probabilities are equal
  - $p(y = 1|\mathbf{x}) = p(y = 2|\mathbf{x})$



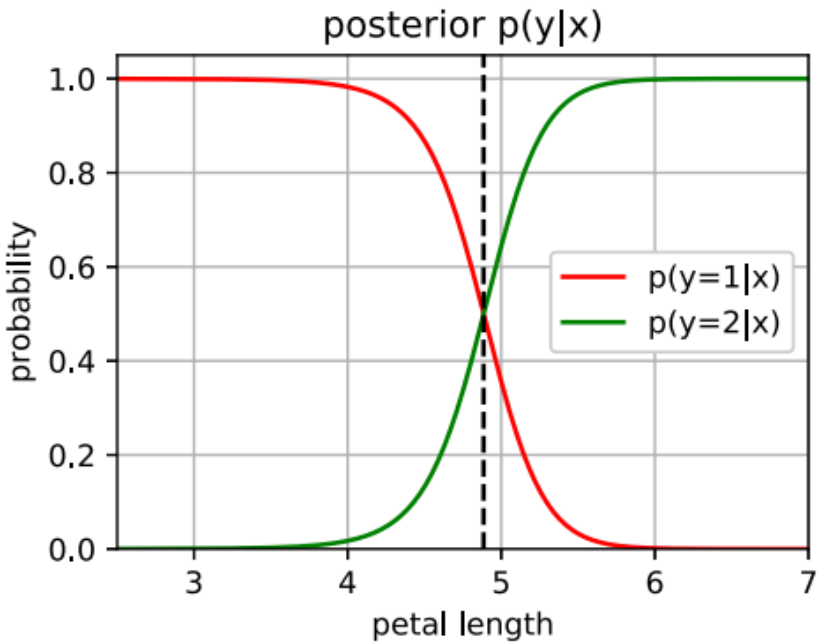
Bayes' Rule Revisited

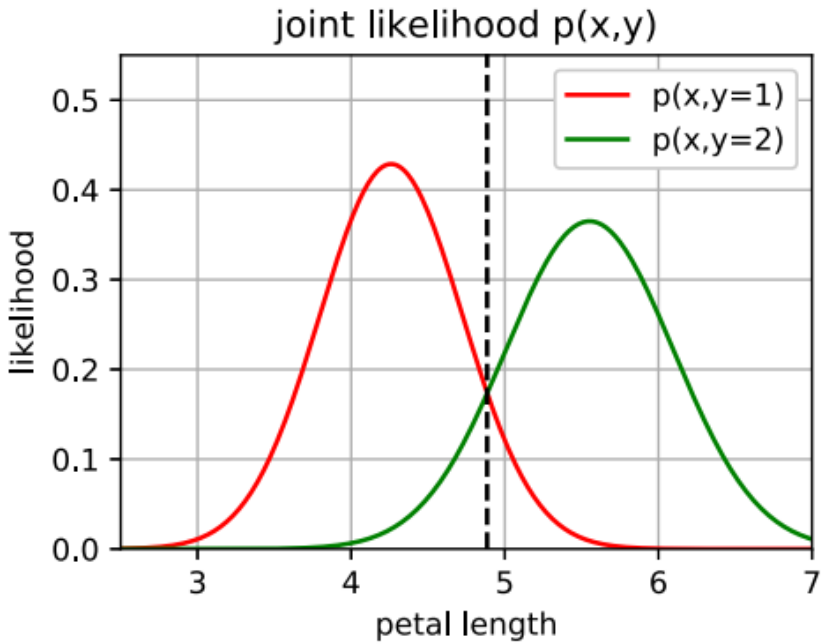
- Bayes' rule:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

- Note that the denominator is the same for each class y
  - Hence, we can compare just the numerator $p(\mathbf{x}|y)p(y)$
  - This also called the **joint likelihood** of the observation and class
    - $p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y)$
- Example:
  - BDR using joint likelihoods:
    - If $p(\mathbf{x}|y=1)p(y=1) > p(\mathbf{x}|y=2)p(y=2)$, then choose Class 1
    - Otherwise, choose Class 2

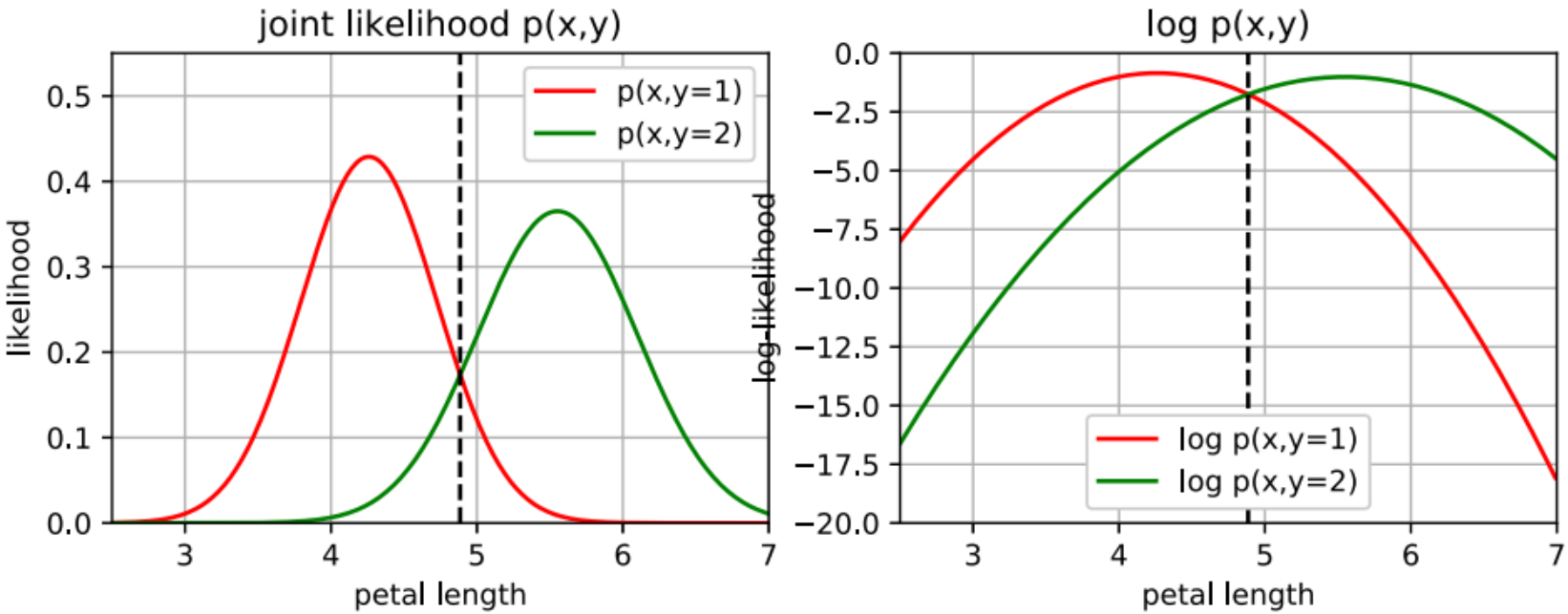
BDR Using Joint Likelihoods





Bayes' Rule Revisited

- Can also apply a monotonic increasing function (like log) and do the comparison
  - Using log likelihoods:
    - $\log p(\mathbf{x}|y=1) + \log p(y=1) > \log p(\mathbf{x}|y=2) + \log p(y=2)$
  - This is more numerically stable when the likelihoods are small



Bayes Classifier Summary

- **Training**:
  1. Collect training data from each class
  2. For each class c, estimate the class conditional densities $p(\mathbf{x}|y=c)$:
     1. Select a form of the distribution (e.g., Gaussian)
     2. Estimate its parameters with MLE
  3. Estimate the class priors $p(y)$ using MLE
- **Classification**:
  1. Given a new sample $\mathbf{x}^*$, calculate the likelihood $p(\mathbf{x}^*|y=c)$ for each class c
  2. Pick the class c with the largest posterior probability $p(y=c|\mathbf{x}^*)$
     - Equivalently, use $p(\mathbf{x}^*|y=c)p(y=c)$ or $\log p(\mathbf{x}^*|y=c) + \log p(y=c)$