Data-driven Decision Making

Introduction to Machine Learning

João Pedro Pedroso

Kseniia Klimetnova

2025/2026

Last class

- Matching problems
- Kidney exchange problems

Today's class:

- Machine learning:
  - motivation
  - basics
- Fundamental algorithms
- Building Blocks of a Learning Algorithm:
  - loss function;
  - optimization criterion based on the loss function
  - optimization method to find a solution
  - basic practice

Introduction to Machine learning

Introduction to Machine learning

Bibliography

- Machine Learning: essential notions
  We'll follow closely
  The Hundred-Page Machine Learning Book
  Andriy Burkov
  `http://themlbook.com/wiki/doku.php`
- Machine Learning in action: we'll pick some topics from Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow
  Aurélien Géron O'Reilly Media, Inc
- Software:
  - Python
    `https://www.python.org`
  - Scikit-learn
    `https://scikit-learn.org`
  - Google colab notebooks
    `https://colab.research.google.com`

Plan for this course

The Hundred-Page Machine Learning Book: contents:

1. Introduction
2. Notation and Definitions
3. Fundamental Algorithms
4. Anatomy of a Learning Algorithm
5. Basic Practice
6. Neural Networks and Deep Learning
7. Problems and Solutions
8. Advanced Practice
9. Unsupervised Learning
10. Other Forms of Learning
11. Conclusion

What is Machine Learning

Machine Learning

- Algorithms that rely on a collection of examples of some phenomenon
  - nature
  - handcrafted by humans
  - generated by another algorithm
- Used for solving practical problems by
  1. gathering a dataset
  2. algorithmically building a statistical model based on that dataset
  3. using the model to solve the problem

Types of Machine Learning

- *Supervised learning*
  - based on a dataset with labeled examples
  - these are used to train (learn) the model
- *Semi-supervised learning*
  - dataset has labeled and unlabeled examples
- *Unsupervised learning*
  - dataset has no labels
  - somehow based on *similarity* among examples
  - learn patterns or data groupings from (unlabeled) data
- *Reinforcement learning*
  - data is acquired from the *environment*
  - *rewards* for *actions* in given *states*

Supervised Learning

Dataset: collection of labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$

- each element $\mathbf{x}_i \rightarrow$ feature vector
  - vector of dimension $j = 1, \ldots, D$
  - each element $\rightarrow$ value that somehow describes the example
  - feature $x^{(j)}$
- *e.g.*, if each example $\mathbf{x}$ in our collection represents a person:
  - feature $x^{(1)} \rightarrow$ height in cm
  - feature $x^{(2)} \rightarrow$ weight in kg
  - ...
  - (position (j) represents the same feature in all examples)
- each label y_i can be:
  - element belonging to a finite set of classes $\{1, 2, \ldots, C\}$
    - *e.g.*, mail message `spam` or `not_spam`
  - real number
    - *e.g.*, credit limit to a customer
  - more complex structure (vector, matrix, tree, graph)

Goal of a *Supervised Learning* algorithm

- use dataset to produce a model
  - input: feature vector **x**
  - output: its label y
- example:
  - input: vector with elements of a person's health
    - weight, BMI, age, ...
  - output: probability that the person has cancer

Unsupervised Learning

- Dataset: collection of unlabeled examples $\{\mathbf{x}_i\}_{i=1}^{N}$
- each element $x_i \rightarrow$ feature vector
- goal of unsupervised learning:
  - use dataset to produce a model
  - input: feature vector $\mathbf{x}$
  - output: a value/vector
- examples:
  - clustering:
    - output: ID of cluster to which each $\mathbf{x}_i$ belongs
  - dimensionality reduction:
    - output: selection of features from $\mathbf{x}$
  - outlier detection:
    - output: real number indicating how $\mathbf{x}$ is different from a "typical" example in the dataset

Semi-Supervised Learning

- Dataset: collection of labeled and unlabeled examples
- Much more *unlabeled* than *labeled*
- Goal:
  - same as the goal of the supervised learning algorithm
  - hope: using many unlabeled examples helps finding a better model
  - leverage on additional information given by larger sample

Reinforcement learning

- Machine "lives" in an environment
- State of environment: vector of features
- Machine can execute actions in every state
- Different actions $\rightarrow$ different rewards $\rightarrow$ may move machine to another state of the environment
- Goal: learn a policy
  - policy $\rightarrow$ function that
    - takes feature vector of a state as input
    - outputs an optimal action to execute in that state
    - action is *optimal* if it maximizes the *expected average reward*
- sequential decision making
- long-term goal:
  - game playing, robotics, resource management, logistics, ...

How Supervised Learning Works

1. Gather data
   - collection of pairs (input, output)
   - input: email messages, pictures, sensor measurements, ...
   - output:
     - (most common:) real numbers or labels (`spam/not_spam`, `cat/dog/mouse`, ...)
     - vectors (*e.g.*, four coordinates of rectangle person on a picture)
     - sequences (*e.g.*, ["adjective", "adjective", "noun"] for input "big beautiful car"
     - some other structure
2. Decide on a *learning algorithm*
3. Create a model

Supervised Learning At Work: example

- Our problem: spam detection
- Data: *e.g.*, 10,000 email messages, each with either label `spam` or `not_spam`
- Convert each email message into a feature vector
  e.g.: *bag of words*
  - to take dictionary of English words
    - say, 20,000 alphabetically sorted words
  - stipulate that each feature vector has 20,000 elements
  - element index j is
    - 1 if word index j in the dictionary is in message
    - 0 otherwise
  - compute this to each message $\rightarrow$
    - 10,000 feature vectors (each with dimension 20,000)
    - 10,000 labels `spam`/`not_spam` $\rightarrow 1/0$, or $+1/-1$
- Dataset now ready to use with a learning algorithm to compute a model
- *e.g.*, let the algorithm be *support vector machine* (SVM)

Supervised Learning At Work: example: SVM

- feature vector $\rightarrow$ point in a high-dimensional space
  - in our example: 20,000-dimensional space
  - algorithm determines 19,999-dimensional hyperplane
    - separate examples with positive labels from examples with negative labels
  - boundary separating examples of different classes $\rightarrow$ decision boundary
- equation of the hyperplane is given by two parameters:
  - real-valued vector $\mathbf{w}$ (same dimensionality as $\mathbf{x}$)
  - real number b; equation:

$$\mathbf{wx} - b = 0$$

    - $\mathbf{wx} \rightarrow w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \ldots + w^{(D)}x^{(D)}$
    - $D \rightarrow$ dimension of feature vector $\mathbf{x}$
- using our model: predict label of some input feature vector $\mathbf{x}$

$$y = \text{sign}(\mathbf{wx} - b)$$

- *learning* $\Leftrightarrow$ *determining* $\mathbf{w}$ *and* b

Goal of the learning algorithm (SVM)

- Objective of the SVM: find optimal values $\mathbf{w}^*$ and b^*
- SVM model:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^*\mathbf{x} - b^*)$$

- To predict whether an email message is spam or not:
  - take the text of the message
  - convert it into a feature vector
  - multiply this vector by $\mathbf{w}^*$, subtract b^*
  - take the sign of the result
    - $+1 \rightarrow$ `spam`
    - $-1 \rightarrow$ `not_spam`

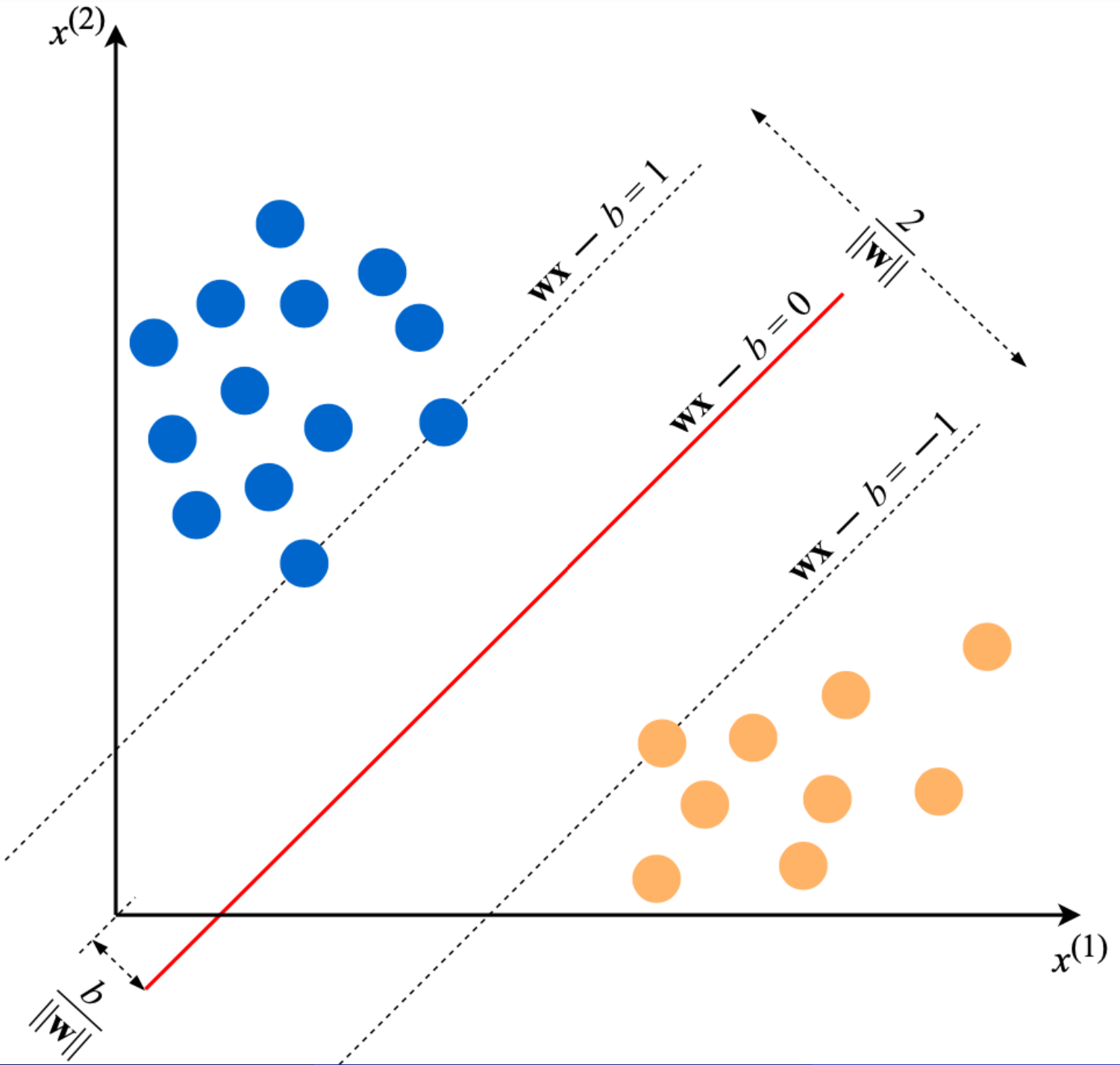
Goal of the learning algorithm (SVM)

- How to find $\mathbf{w}^*, b^*$?
  - solve an *optimization problem*
  - *constraints:* correctly predict labels of our $i = 1, \ldots, 10,000$ examples

$$\mathbf{w}\mathbf{x}_i - b \geq +1 \quad \text{if } y_i = +1$$
$$\mathbf{w}\mathbf{x}_i - b \leq -1 \quad \text{if } y_i = -1$$

  - *objective:* maximize *margin* $\Leftrightarrow$ minimize $||\mathbf{w}|| = \sqrt{\sum_{j=1}^{D} (w^{(j)})^2}$
- Optimization model (*training*):

$$\begin{aligned} \text{minimize} \quad & ||\mathbf{w}|| \\ \text{subject to} \quad & y_i(\mathbf{w}\mathbf{x}_i - b) \geq +1 \qquad \text{for } i = 1, \ldots \end{aligned}$$


$x^{(2)}$
$\mathbf{wx} - b = 1$
$\frac{2}{\|\mathbf{w}\|}$
$\mathbf{wx} - b = 0$
$\mathbf{wx} - b = -1$
$x^{(1)}$
$\frac{b}{\|\mathbf{w}\|}$

SVMs

- can also incorporate kernels
  - make the decision boundary arbitrarily non-linear
- usual version of SVM incorporates a *penalty hyperparameter*
  - what to pay in the objective for the misclassification of training examples

- hyperparameter:
  - property of a learning algorithm
  - often, a numerical value
  - influence the way the algorithm works
  - hyperparameter aren't learned by the algorithm
    - must be set by data analyst before running the algorithm

Summary

- Any classification learning algorithm that builds a model creates a decision boundary
  - implicitly or explicitly
- Can be straight, curved, have a complex form, ...
- Form of the decision boundary determines the accuracy of the model
  - ratio of examples whose labels are predicted correctly
- Different learning algorithms:
  - different decision boundaries
  - different speed of model building and prediction processing time

Notation

We'll follow notation used in
The Hundred-Page Machine Learning Book:

1. Data structures:
   - scalars: italic letters x, y, z
   - vectors: boldface letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$
     - $\mathbf{x}_i \rightarrow$ vector index i (on dataset)
     - elements of a vector $\mathbf{x}$: $(x^{(1)}, x^{(2)}, \ldots)$
   - matrices: bold capital letters $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$
   - sets: calligrafic letters $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$
2. Random variables
   - denoted with capital letter X, Y, Z
   - $f_X \rightarrow$ probability density function of X
     - "examples" $\rightarrow$ observations of X
     - "dataset" $\rightarrow$ collection of examples (*i.e.*, a *sample*) $S_X = \{x_i\}_{i=1}^N$

Parameters vs. Hyperparameters

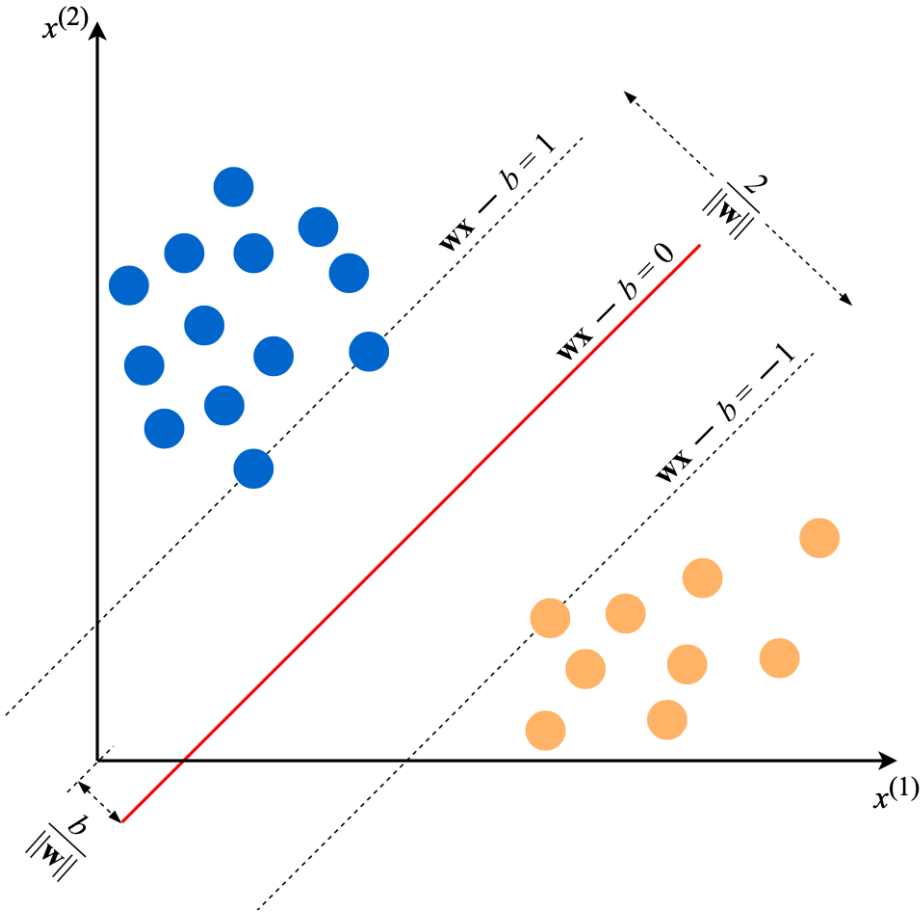
- *hyperparameters* $\rightarrow$ properties of a learning algorithm
  - influence the way the algorithm works
  - not learned by the algorithm from data
  - set by the data analyst before running the algorithm
- *parameters* $\rightarrow$ define the model learned
  - directly modified by the learning algorithm based on the training data
  - goal of learning: finding parameters that make the model "optimal"

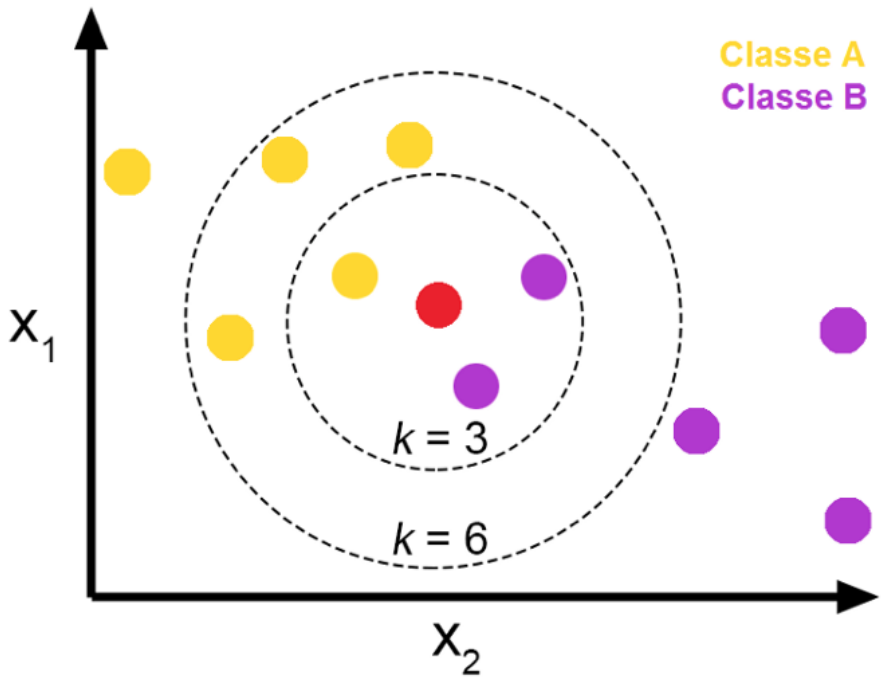
Classification vs. Regression

- Classification: problem of automatically assigning a label to an unlabeled example
  - *e.g.*, spam detection
  - learning algorithm:
    - takes a collection of labeled examples as inputs
    - produces a model:
      $\rightarrow$ take as input an unlabeled example
      $\rightarrow$ output a label (or a label identifier)
  - binary classification: two classes
  - multiclass classification: 3 or more classes
- Regression: problem of predicting a real-valued label for an unlabeled example
  - *e.g.*, estimating house price valuation based on house features
    - area, number of bedrooms, location, ...
  - learning algorithm:
    - takes a collection of labeled examples as inputs
    - produces a model:
      $\rightarrow$ take as input an unlabeled example
      $\rightarrow$ output a target

Model-Based vs. Instance-Based Learning

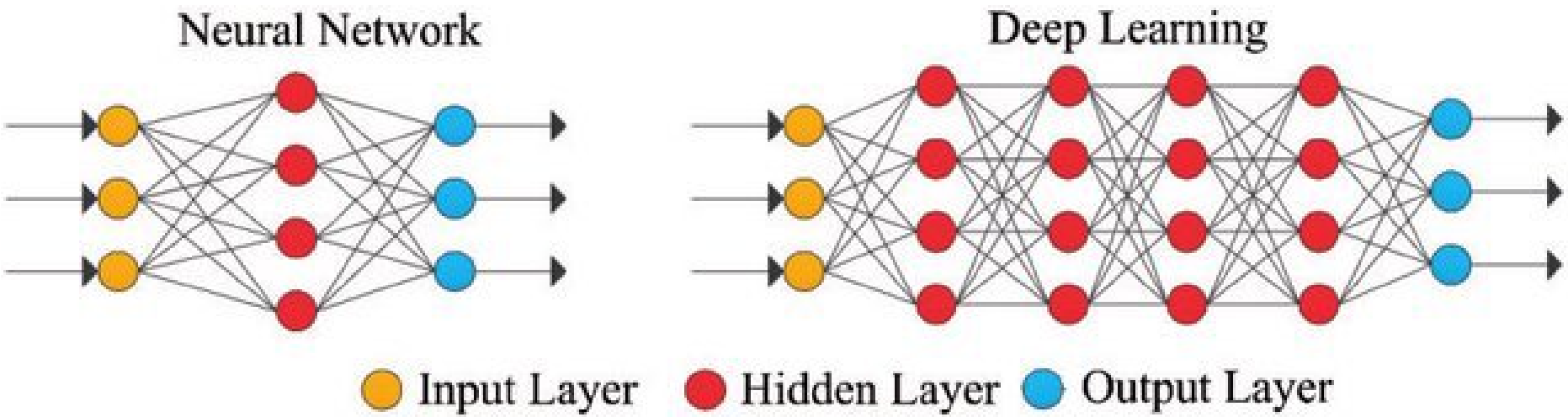
- Most supervised learning algorithms are model-based
  - *e.g.*, SVM
  - use the training data to create a model that has parameters learned from training data
    - in SVM: $\mathbf{w}^*, b^*$
  - after the model is built, training data can be discarded
- Instance-based learning algorithms:
  - use the whole dataset as the model
  - *e.g.*, k-Nearest Neighbors (kNN)
    - look at *neighborhood* of input example (in the space of feature vectors)
    - output:
      $\rightarrow$ classification: label most often seen
      $\rightarrow$ regression: average target of neighbors


$x^{(2)}$
$\mathbf{wx} - b = 1$
$\frac{2}{\|\mathbf{w}\|}$
$\mathbf{wx} - b = 0$
$\mathbf{wx} - b = -1$
$x^{(1)}$
$\frac{b}{\|\mathbf{w}\|}$


Classe A
Classe B
x_1
$k = 3$
$k = 6$
x_2

Shallow vs. Deep Learning

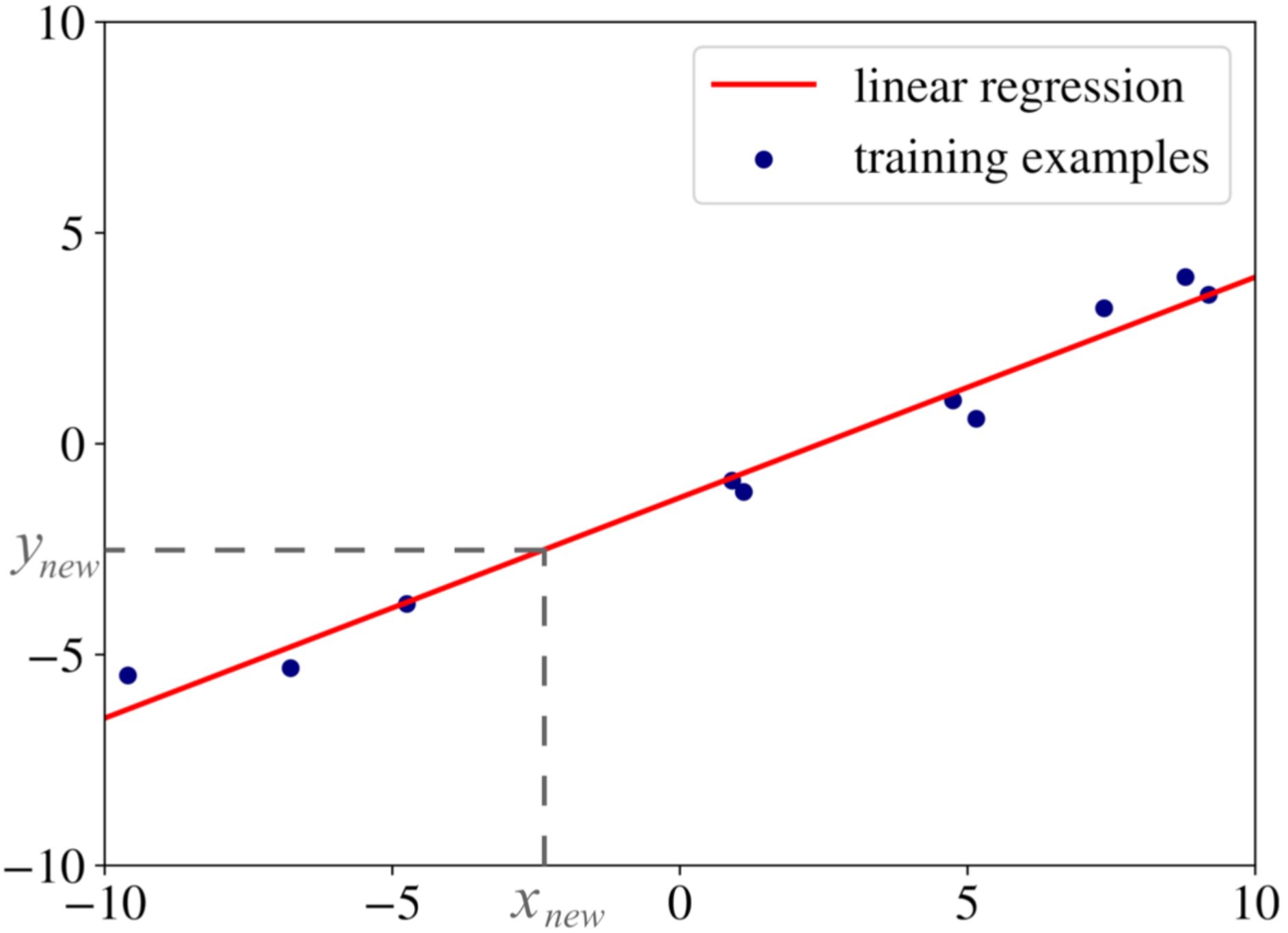
- Shallow learning algorithm:
  - learns the parameters of the model directly from the features of the training examples
  - most supervised learning algorithms are shallow
  - exceptions: are neural networks
- Deep learning algorithm:
  - deep neural networks
    - more than one layer between input and output
  - most model parameters:
    - not directly learned from the features of training examples
    - learned from the outputs of the preceding layers


Neural Network
Deep Learning
Input Layer
Hidden Layer
Output Layer

Linear Regression

Model is a linear combination of input features

- Problem statement:
  - input: labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
    - $N \rightarrow$ size of the collection
    - $\mathbf{x}_i \in \mathbb{R}^D \rightarrow D$ -dimensional vector i
    - $y_i \in \mathbb{R} \rightarrow$ target value
  - build model $f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}\mathbf{x} + b$
    - $\mathbf{w} \in \mathbb{R}^D \rightarrow D$ -dimensional vector of parameters
    - $b \in \mathbb{R}$
    - *f is parametrized by $\mathbf{w}$ and b*
  - model used to predict the unknown y for a given $\mathbf{x}$
    - $y \leftarrow f_{\mathbf{w},b}(\mathbf{x})$
  - find optimal values $\mathbf{w}^*, b^*$
    - *make the most accurate predictions on data*


linear regression
training examples
10
5
0
−5
−10
−10
−5
0
5
10
y_{new}
x_{new}

Linear Regression

- Minimize sum of squared errors:

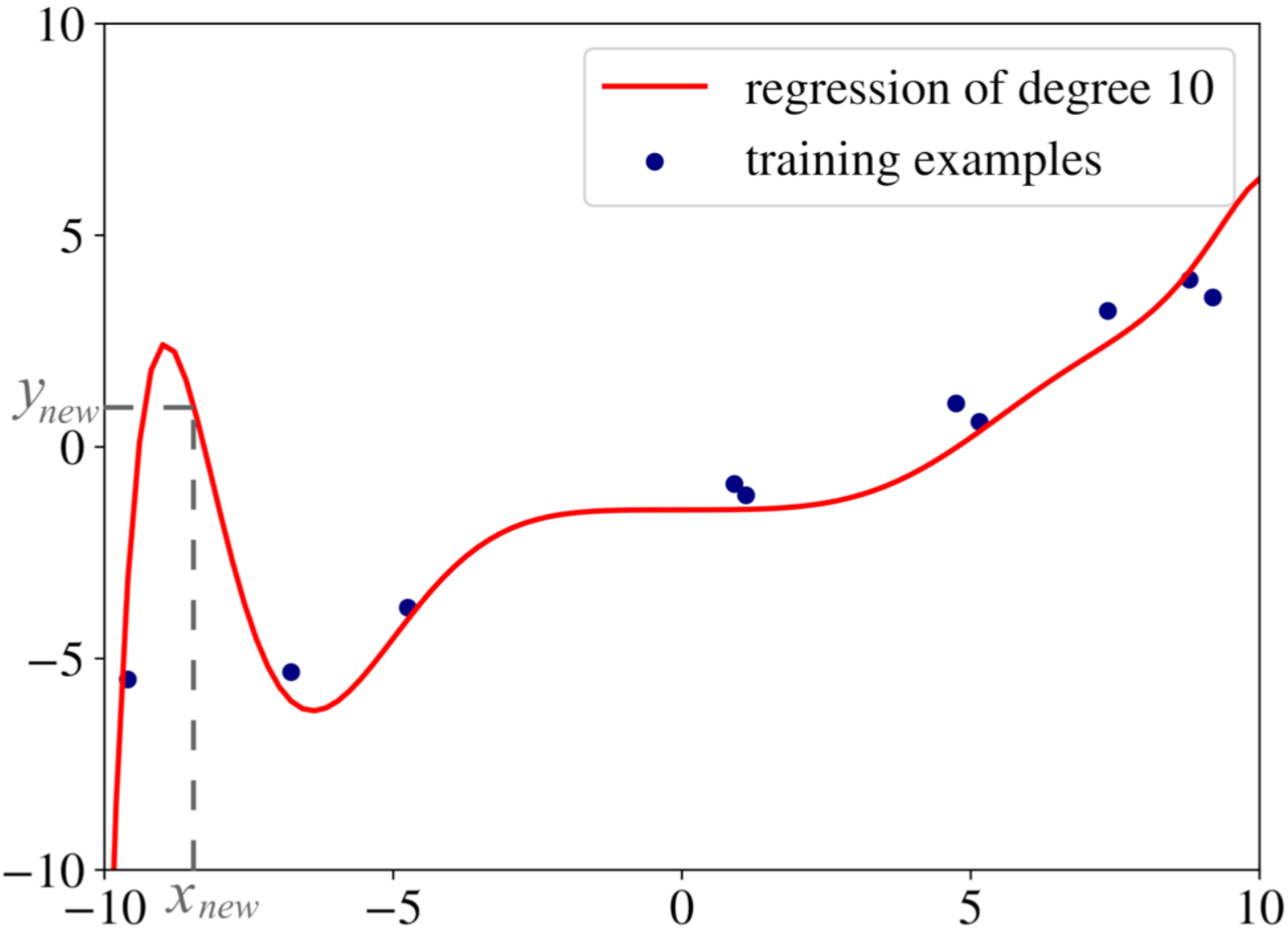
$$\frac{1}{N} \sum_{i=1,\ldots,N} (f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2$$

  - objective function/cost function
  - loss function: $(f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2$
    - *squared error loss/mean squared error (MSE)*
- Model-based learning: minimize loss function
- Linear regression: cost function is *average loss*
  - *empirical risk*
  - average of all penalties obtained by applying the model to the training data
  - quadratic vs absolute value $\rightarrow$ *different models*
    - quadratic is convenient: derivatives
    - optimum $\rightarrow$ gradient set to zero

Linear Regression

- Extensions:
  - use a higher degree polynomial instead of linear combination
  - overfitting:
    - model predicts very well labels of examples used for training
    - makes errors on new examples

Overfitting



- *more powerful models* $\Rightarrow$ *higher risk of overfitting*

Logistic Regression

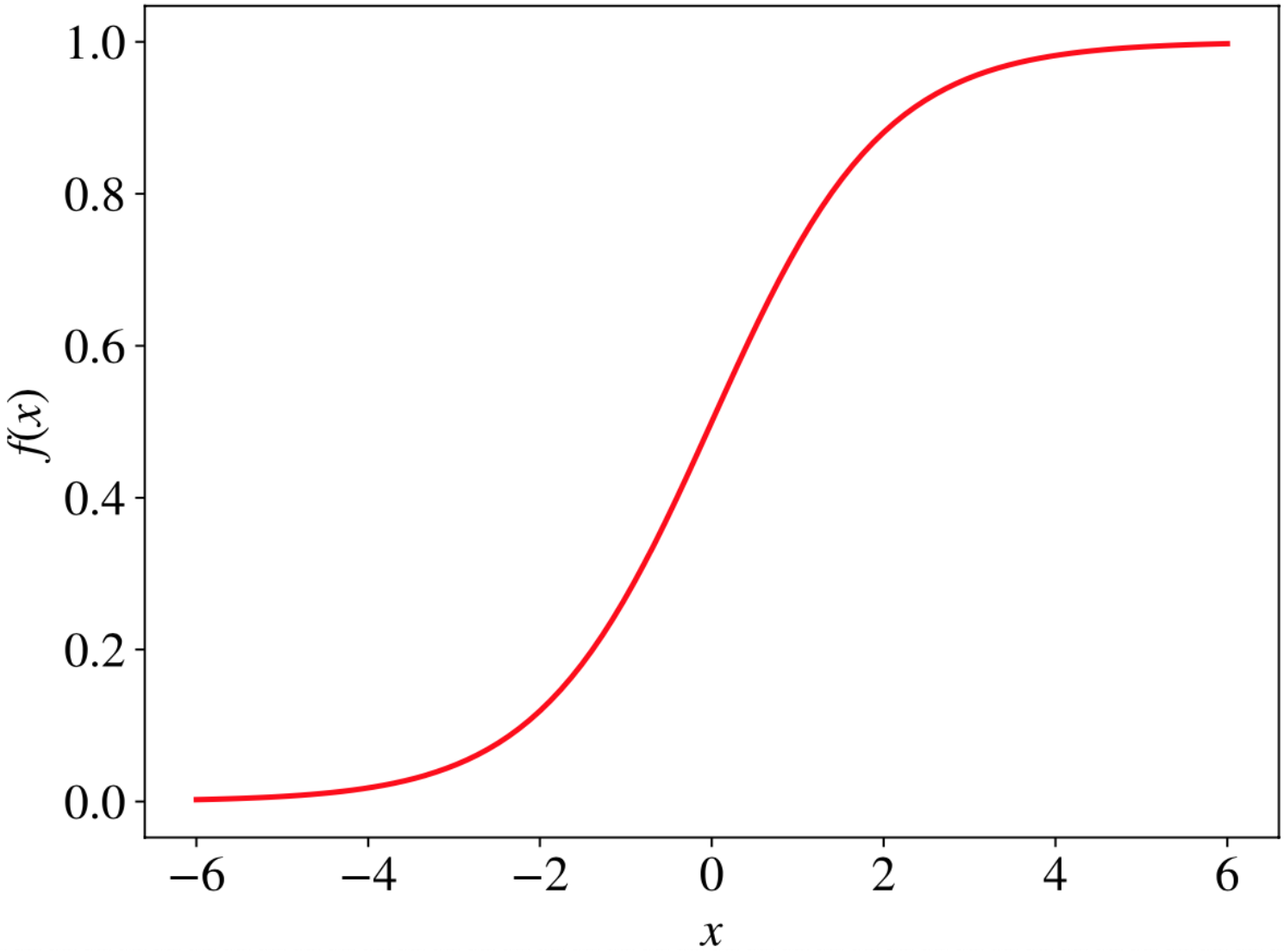
- Actually, logistic regression is a *classification learning algorithm*
- Name comes from statistics
  - mathematical formulation is similar to that of linear regression
- Sigmoid/Standard logistic function:

$$\frac{1}{1+e^{-x}}$$

- Logistic Regression model

$$f_{\mathbf{w},b}(\mathbf{x}) \stackrel{\text{def}}{=\!=} \frac{1}{1+e^{-(\mathbf{w}\mathbf{x}+b)}}$$

Standard logistic function



Logistic Regression

- Interpretation: $f(\mathbf{x})$ as the probability of y_i being positive
- How to find optimal $\mathbf{w}^*, b^*$?
  - maximize likelihood of our training set according to the model
    - how likely an observation (example) is
- For instance:
  - labeled example $(\mathbf{x}_i, y_i)$ in training data
  - model: arbitrary $\hat{\mathbf{w}}, \hat{b}$
  - applying to $\mathbf{x}_i$: $f_{\hat{\mathbf{w}},\hat{b}}(\mathbf{x}_i) \rightarrow$ output $0 < p < 1$
  - if y_i is the positive class:
    - *likelihood of y_i being the positive class given by p*
  - if y_i is the negative class:
    - *likelihood of y_i being the negative class given by $1 - p$*

Optimization criterion in logistic regression: maximum likelihood

- Likelihood of the training data:

$$L_{\mathbf{w},b} \stackrel{\text{def}}{=\!=} \prod_{i=1\ldots N} f_{\mathbf{w},b}(\mathbf{x}_i)^{y_i}(1 - f_{\mathbf{w},b}(\mathbf{x}_i))^{1-y_i}$$

  - each term in the product:
    - $f_{\mathbf{w},b}(\mathbf{x}_i)$ when $y_i = 1$
    - $1 - f_{\mathbf{w},b}(\mathbf{x}_i)$ otherwise
- in practice, to avoid overflow in the exponential, it's more convenient to maximize log-likelihood:

$$\begin{aligned} LogL_{\mathbf{w},b} &\stackrel{\text{def}}{=\!=} \log L_{\mathbf{w},b} \\ &\stackrel{\text{def}}{=\!=} \sum_{i=1\ldots N} [y_i \log f_{\mathbf{w},b}(\mathbf{x}_i) + (1 - y_i) \log (1 - f_{\mathbf{w},b}(\mathbf{x}_i))] \end{aligned}$$

- no closed form solution $\rightarrow$ use, *e.g.*, gradient descent to find optimum

Logistic Regression

- Classification learning algorithm
- Logistic Regression model

$$f_{\mathbf{w},b}(\mathbf{x}) \stackrel{\text{def}}{=\!=} \frac{1}{1+e^{-(\mathbf{w}\mathbf{x}+b)}}$$

  - $f_{\mathbf{w},b}(\mathbf{x}) \rightarrow$ *likelihood of* $\mathbf{x}$ *being on positive class*
  - $1 - f_{\mathbf{w},b}(\mathbf{x}) \rightarrow$ *likelihood of* $\mathbf{x}$ *being on negative class*

Decision Tree Learning

Decision Tree Learning

- Decision tree: acyclic graph used to make decisions
  - each branching node examines a specific feature j
    - if value below specific threshold $\Rightarrow$ follow left branch
    - otherwise $\Rightarrow$ follow right branch
  - leaf node $\rightarrow$ decision: class to which the example belongs
- Problem statement:
  - input: labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
    - labels $y_i \in \{0, 1\}$
- Solution: several possibilities, choosing ID3
  - average log-likelihood:

$$\frac{1}{N} \sum_{i=1 \ldots N} [y_i \log f_{\text{ID3}}(\mathbf{x}_i) + (1 - y_i) \log (1 - f_{\text{ID3}}(\mathbf{x}_i))]$$

  - $f_{\text{ID3}}(\mathbf{x}_i) \rightarrow$ decision tree model
  - ID3 constructs approximate nonparametric model

$$f_{\text{ID3}}(\mathbf{x}) \stackrel{\text{def}}{=\!=} \Pr(y = 1 | \mathbf{x})$$

ID3 steps

- Initially: decision tree only has a start node containing all examples

$$\mathcal{S} \stackrel{\text{def}}{=\!=} \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

- Initial (constant) model:

$$f_{\text{ID3}}^{\mathcal{S}} \stackrel{\text{def}}{=\!=} \frac{1}{|\mathcal{S}|} \sum_{(\mathbf{x},y)\in\mathcal{S}} y$$

  - same prediction $f_{\text{ID3}}^{\mathcal{S}}(\mathbf{x})$ for any input $\mathbf{x}$

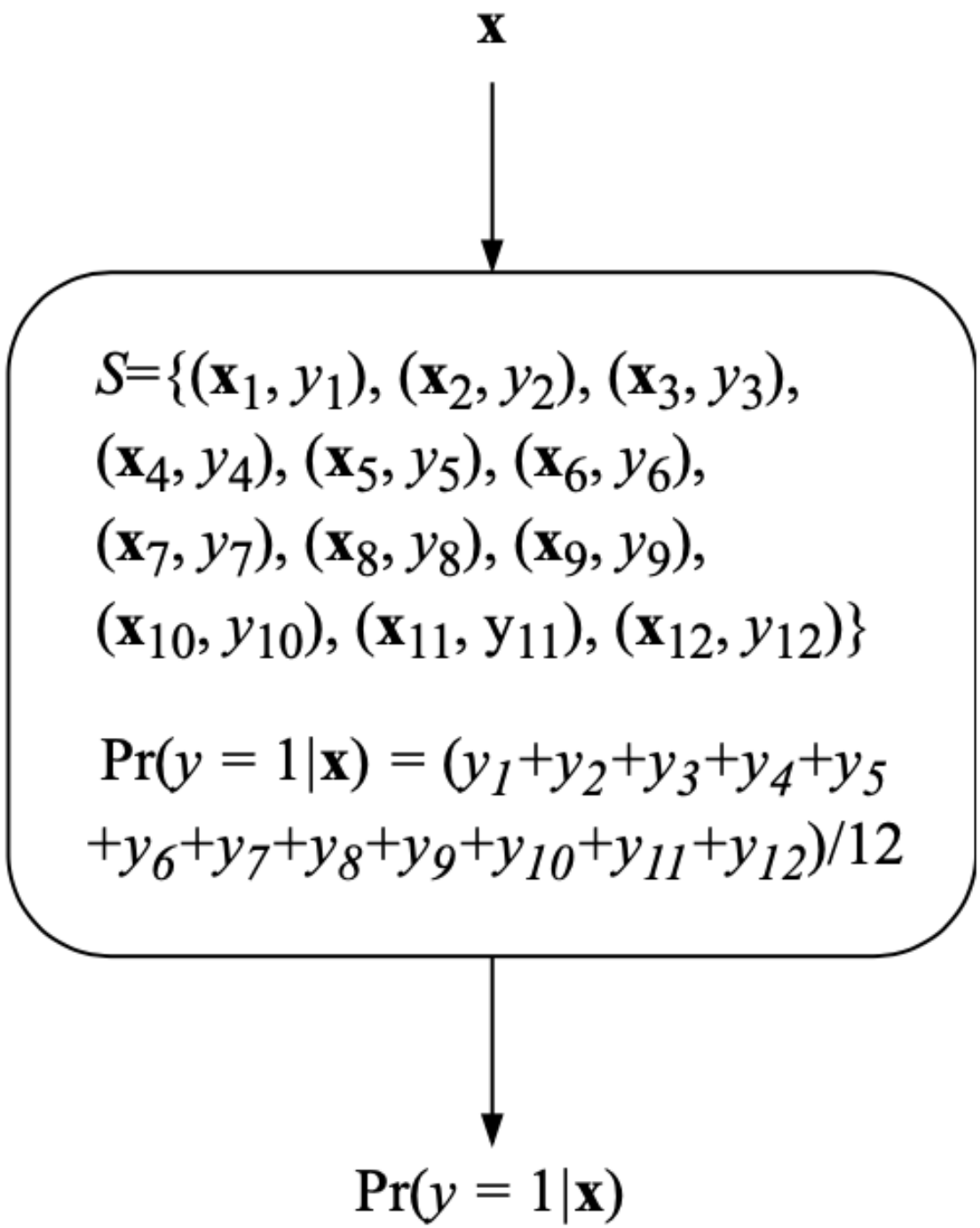
ID3 steps (cont.)

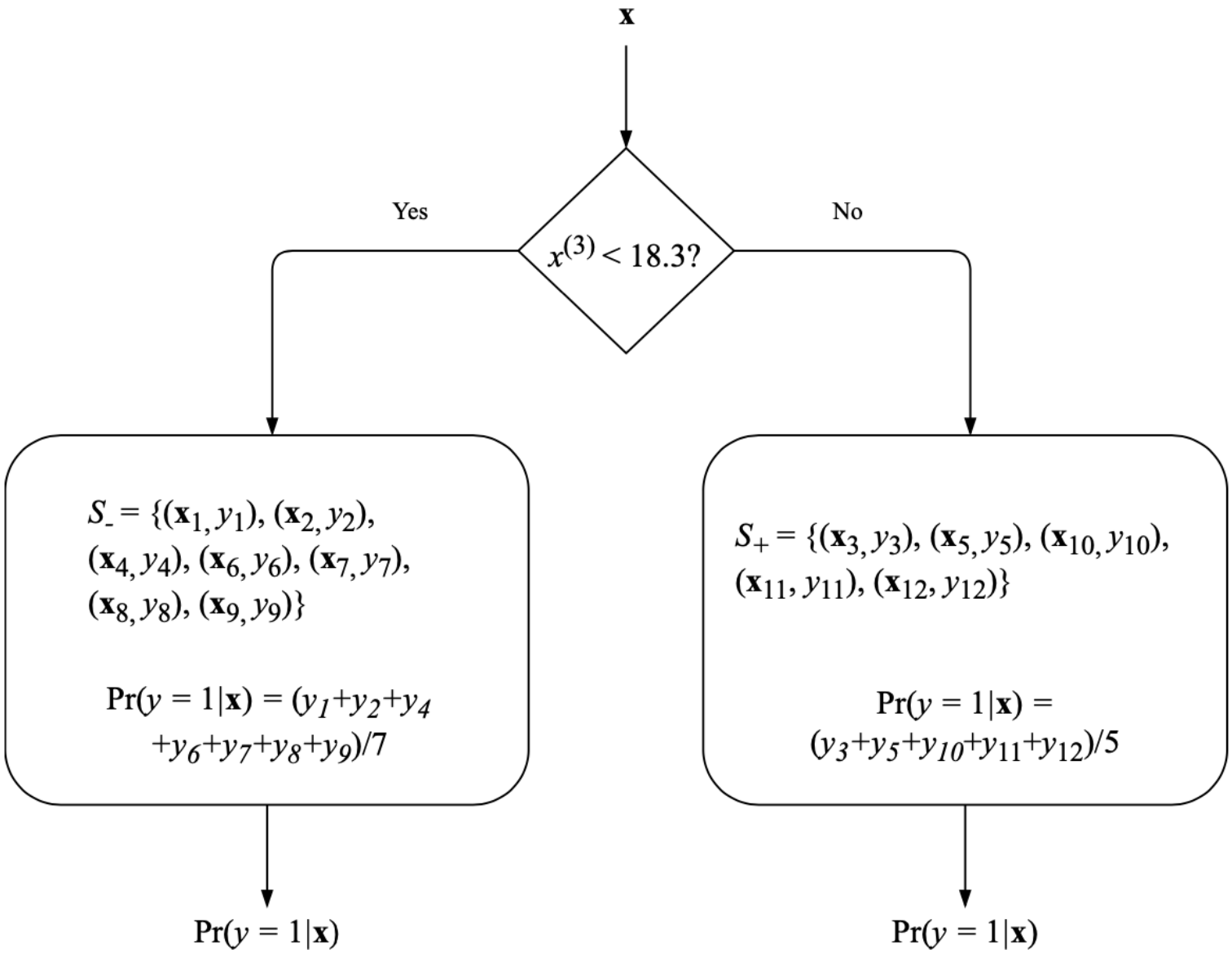
- Then, search through:
  - all features $j = 1, \ldots, D$
  - all thresholds t
  - and split the set S into two subsets

$$\mathcal{S}_- \stackrel{\text{def}}{=\!=} \{(\mathbf{x}, y) | (\mathbf{x}, y) \in \mathcal{S}, x^{(j)} < t\}$$
$$\mathcal{S}_+ \stackrel{\text{def}}{=\!=} \{(\mathbf{x}, y) | (\mathbf{x}, y) \in \mathcal{S}, x^{(j)} \geq t\}$$

  - new subsets $\rightarrow$ two new leaf nodes
  - evaluate for all possible pairs (j, t) "how good" split with pieces $\mathcal{S}_-$ and $\mathcal{S}_+$ is
  - finally:
    $\rightarrow$ pick the best such values (j, t)
    $\rightarrow$ split $\mathcal{S}$ into $\mathcal{S}_+$ and $\mathcal{S}_-$
    $\rightarrow$ form two new leaf nodes
    $\rightarrow$ continue recursively on $\mathcal{S}_+$ and $\mathcal{S}_-$

- Quit when no split produces a model that's sufficiently better than the current one


x
$S=\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3),$
$(\mathbf{x}_4, y_4), (\mathbf{x}_5, y_5), (\mathbf{x}_6, y_6),$
$(\mathbf{x}_7, y_7), (\mathbf{x}_8, y_8), (\mathbf{x}_9, y_9),$
$(\mathbf{x}_{10}, y_{10}), (\mathbf{x}_{11}, y_{11}), (\mathbf{x}_{12}, y_{12})\}$
$\Pr(y = 1|\mathbf{x}) = (y_1+y_2+y_3+y_4+y_5$
$+y_6+y_7+y_8+y_9+y_{10}+y_{11}+y_{12})/12$
$\Pr(y = 1|\mathbf{x})$


x
Yes
No
$x^{(3)} < 18.3?$
$S_- = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_4, y_4), (\mathbf{x}_6, y_6), (\mathbf{x}_7, y_7), (\mathbf{x}_8, y_8), (\mathbf{x}_9, y_9)\}$
$\Pr(y = 1|\mathbf{x}) = (y_1+y_2+y_4+y_6+y_7+y_8+y_9)/7$
$S_+ = \{(\mathbf{x}_3, y_3), (\mathbf{x}_5, y_5), (\mathbf{x}_{10}, y_{10}), (\mathbf{x}_{11}, y_{11}), (\mathbf{x}_{12}, y_{12})\}$
$\Pr(y = 1|\mathbf{x}) = (y_3+y_5+y_{10}+y_{11}+y_{12})/5$
$\Pr(y = 1|\mathbf{x})$
$\Pr(y = 1|\mathbf{x})$

ID3: "how good" is a split?

- Goodness of a split: estimated by using entropy criterion
  - measures uncertainty about a random variable
  - reaches maximum when all values of the random variables are equiprobable
  - reaches minimum when random variable can have only one value
- Entropy of a set of examples $\mathcal{S}$:

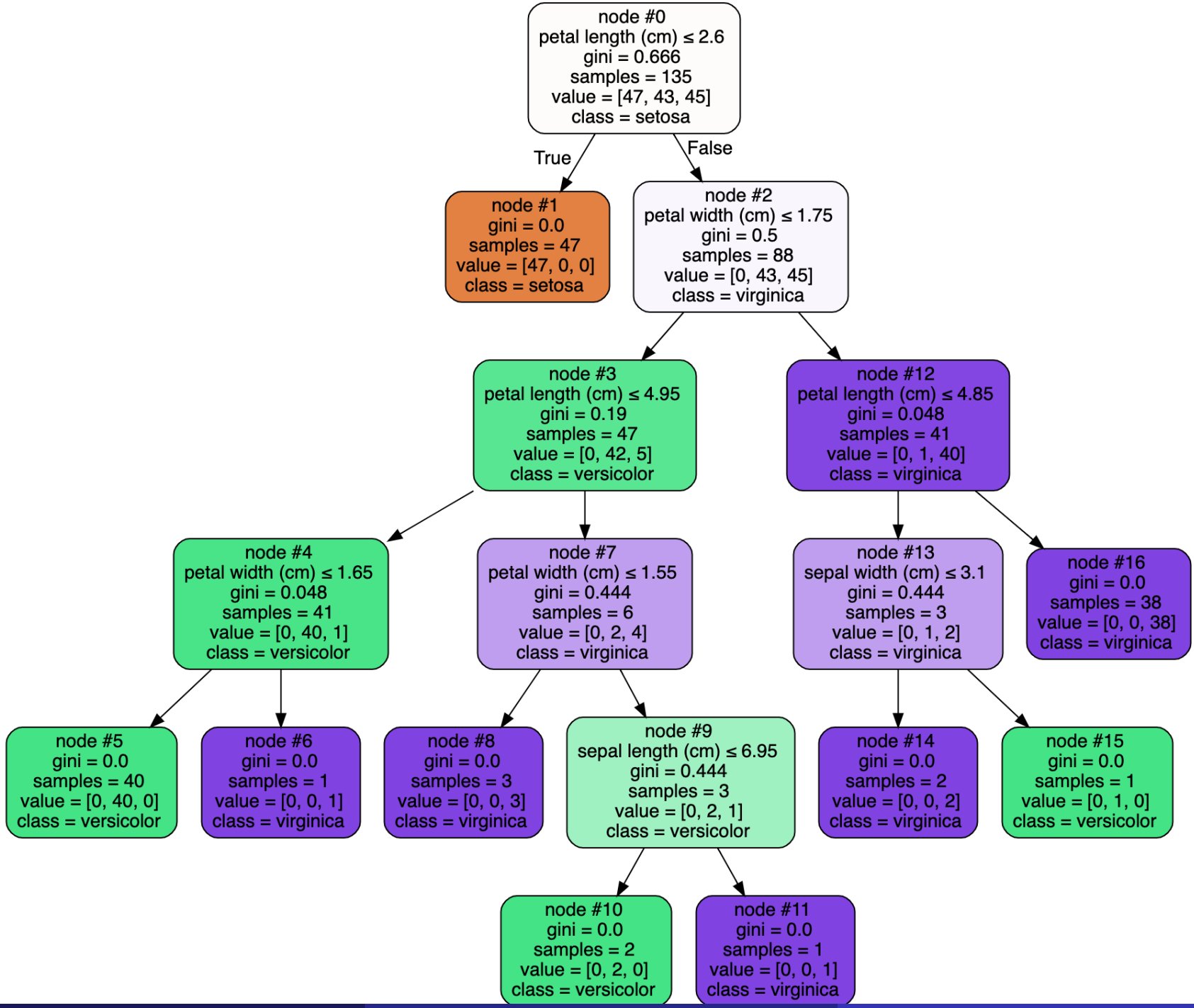
$$H(\mathcal{S}) \stackrel{\text{def}}{=\!=} -f^{\mathcal{S}}_{\text{ID3}} log(f^{\mathcal{S}}_{\text{ID3}}) - (1 - f^{\mathcal{S}}_{\text{ID3}}) \log(1 - f^{\mathcal{S}}_{\text{ID3}})$$

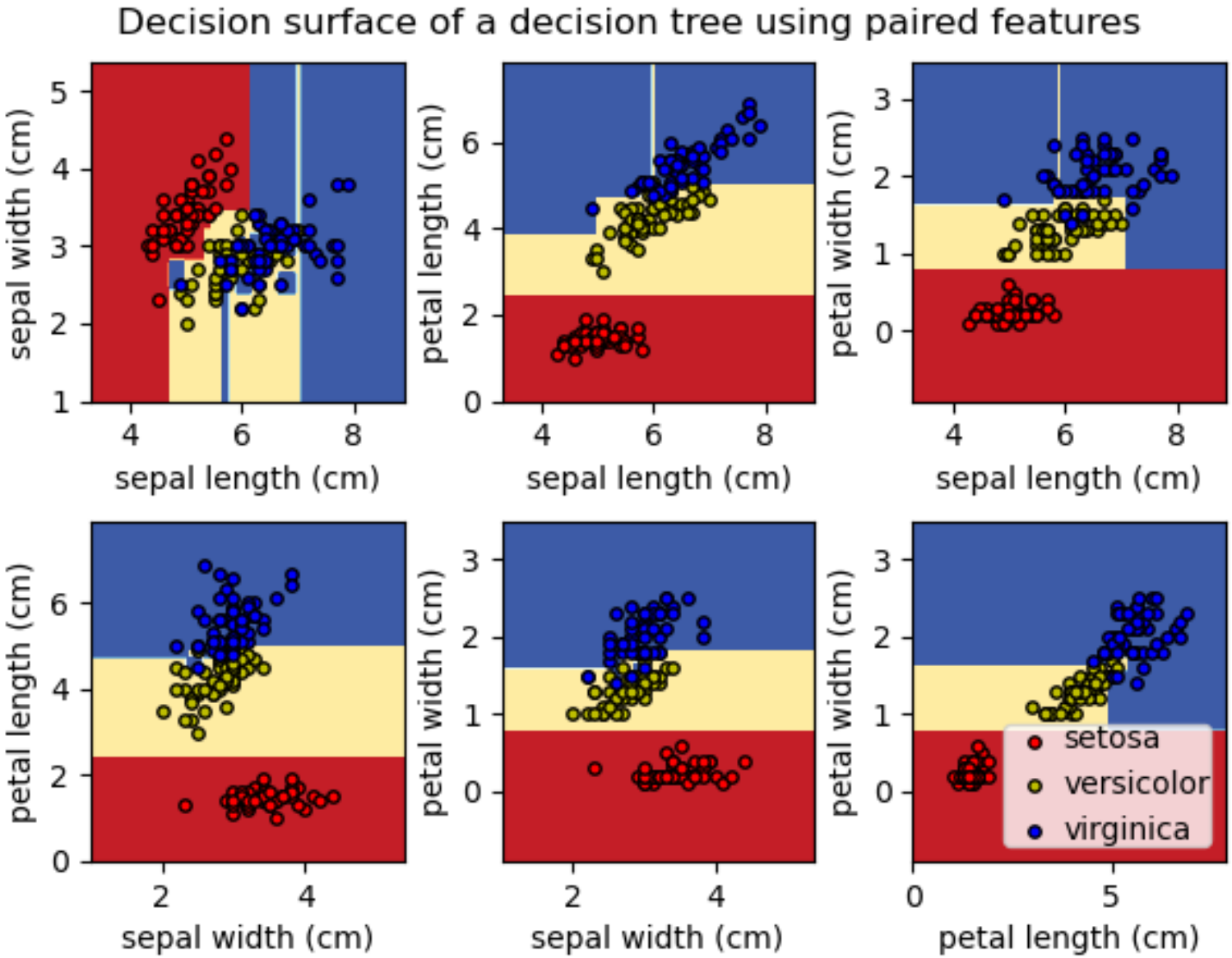
- Entropy of a split $H(\mathcal{S}_-, \mathcal{S}_+)$: weighted sum of two entropies:

$$H(\mathcal{S}_-, \mathcal{S}_+) \stackrel{\text{def}}{=\!=} \frac{|\mathcal{S}_-|}{|\mathcal{S}|} H(\mathcal{S}_-) + \frac{|\mathcal{S}_+|}{|\mathcal{S}|} H(\mathcal{S}_+)$$

ID3: stopping criteria

- At each step, at each leaf node:
  - find a split that minimizes entropy
  - or stop at this leaf node
- Criteria for stopping at a leaf node:
  - all examples there are correctly classified
  - we cannot find an attribute to split upon
  - best split reduces the entropy less than some given ϵ
  - tree reaches some given maximum depth d
- Hyperparameters ϵ, d to be found experimentally
- Widely used formulation improving ID3: C4.5
  - accepts both continuous and discrete features
  - handles incomplete examples
  - solves overfitting problem by using a bottom-up technique known as "pruning"
    - go through the tree once it's been created
    - branches that don't contribute significantly to the error reduction $\rightarrow$ replaced by leaf nodes

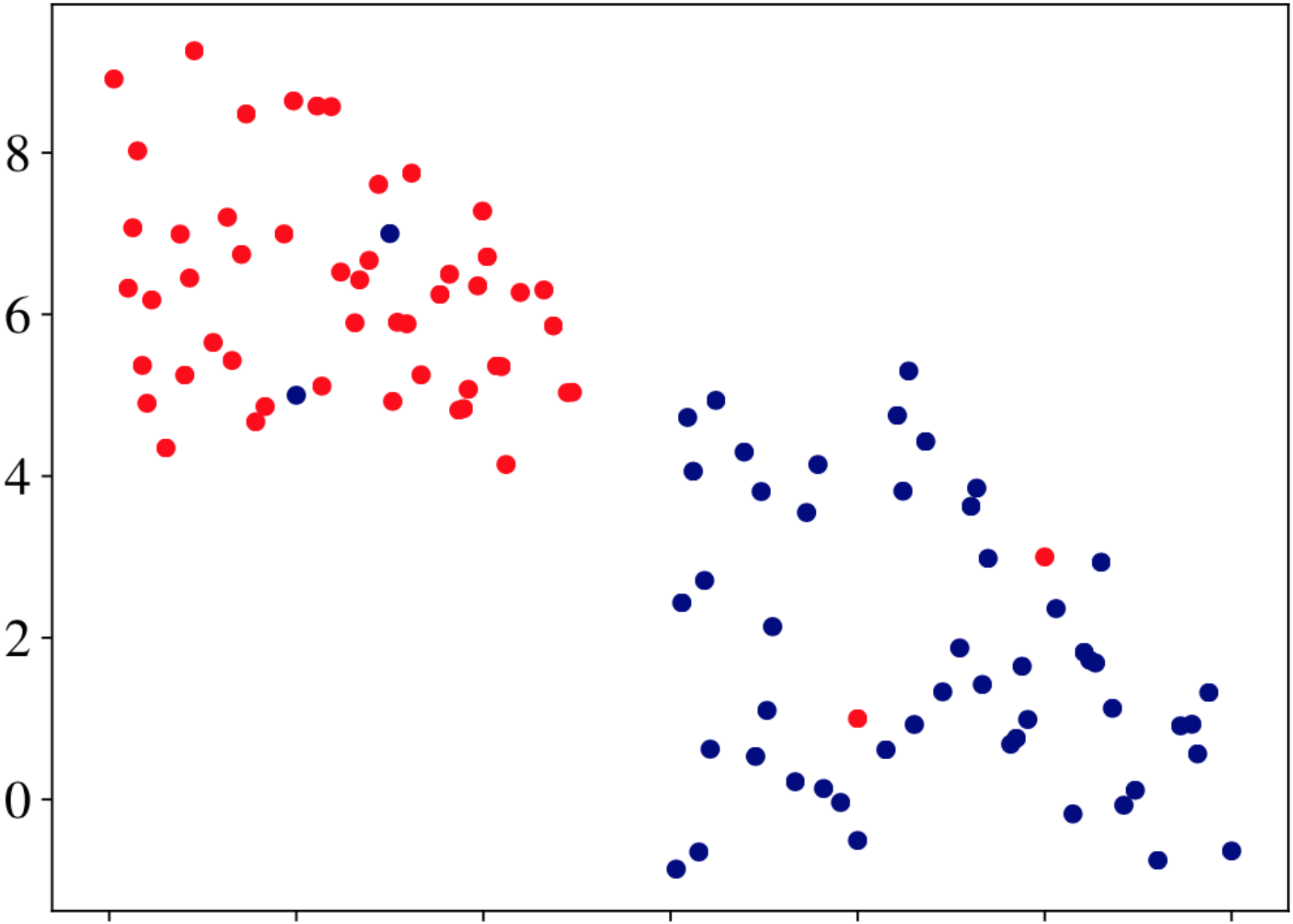

node #0
petal length (cm) ≤ 2.6
gini = 0.666
samples = 135
value = [47, 43, 45]
class = setosa
True
False
node #1
gini = 0.0
samples = 47
value = [47, 0, 0]
class = setosa
node #2
petal width (cm) ≤ 1.75
gini = 0.5
samples = 88
value = [0, 43, 45]
class = virginica
node #3
petal length (cm) ≤ 4.95
gini = 0.19
samples = 47
value = [0, 42, 5]
class = versicolor
node #12
petal length (cm) ≤ 4.85
gini = 0.048
samples = 41
value = [0, 1, 40]
class = virginica
node #4
petal width (cm) ≤ 1.65
gini = 0.048
samples = 41
value = [0, 40, 1]
class = versicolor
node #7
petal width (cm) ≤ 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
class = virginica
node #13
sepal width (cm) ≤ 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginica
node #16
gini = 0.0
samples = 38
value = [0, 0, 38]
class = virginica
node #5
gini = 0.0
samples = 40
value = [0, 40, 0]
class = versicolor
node #6
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
node #8
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
node #9
sepal length (cm) ≤ 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor
node #14
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
node #15
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
node #10
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
node #11
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

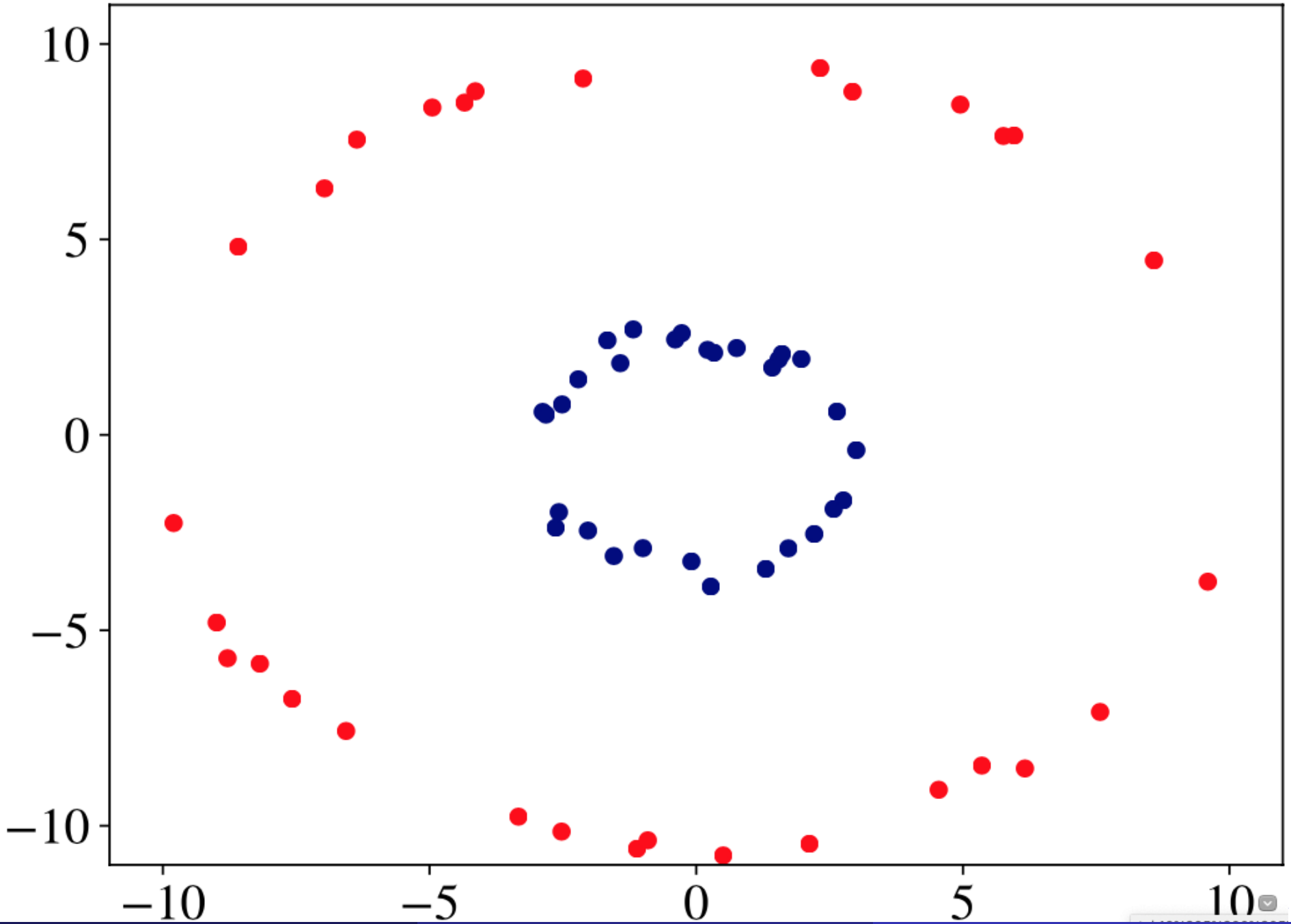

Decision surface of a decision tree using paired features
sepal width (cm)
sepal length (cm)
petal length (cm)
sepal length (cm)
petal width (cm)
sepal length (cm)
petal length (cm)
sepal width (cm)
petal width (cm)
sepal width (cm)
petal width (cm)
petal length (cm)
setosa
versicolor
virginica

Support Vector Machine

More about SVM:

- What if there's noise in the data and no hyperplane can perfectly separate positive examples from negative ones?
- What if the data cannot be separated using a plane, but could be separated by a higher-order polynomial?


8
6
4
2
0


10
5
0
−5
−10
−10
−5
0
5
10

SVM: problem formulation

For convenience, objective is squared:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \text{subject to:} \quad & \mathbf{w}\mathbf{x}_i - b \geq +1 \quad \text{if } y_i = +1 \\ & \mathbf{w}\mathbf{x}_i - b \leq -1 \quad \text{if } y_i = -1 i = 1, \ldots, N \end{aligned}$$

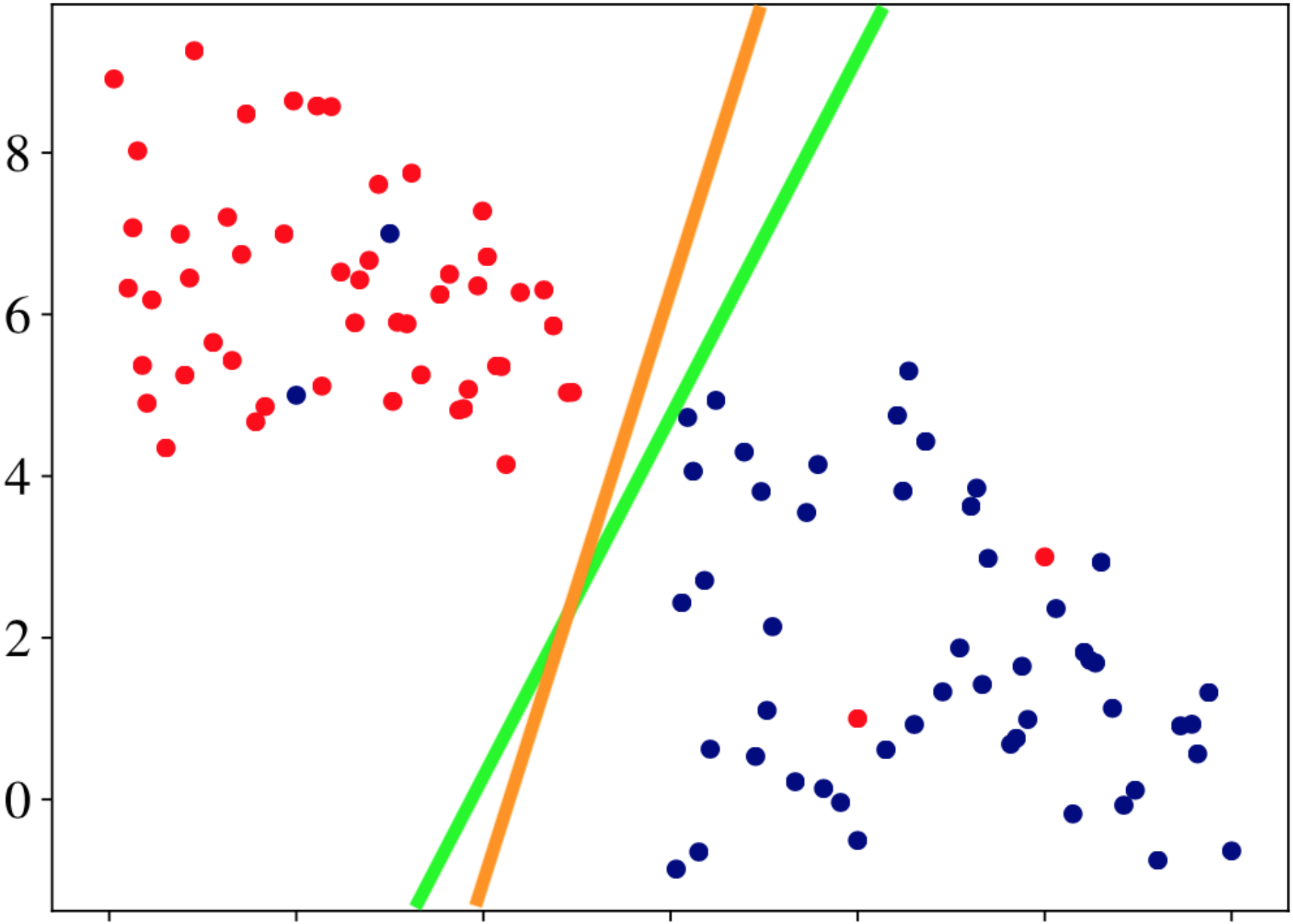
- Constraints can be written as $y_i(\mathbf{w}\mathbf{x}_i - b) \geq +1$
  - *hard margin SVM*
- We want to make them "soft": allow violations, at a cost in the objective
  - for dealing with noise
- Hinge loss function: $\max(0, 1 - y_i(\mathbf{w}\mathbf{x}_i - b))$
  - zero if constraints are satisfied
  - proportional to the distance to decision boundary if not

Dealing with noise: Soft-margin SVM

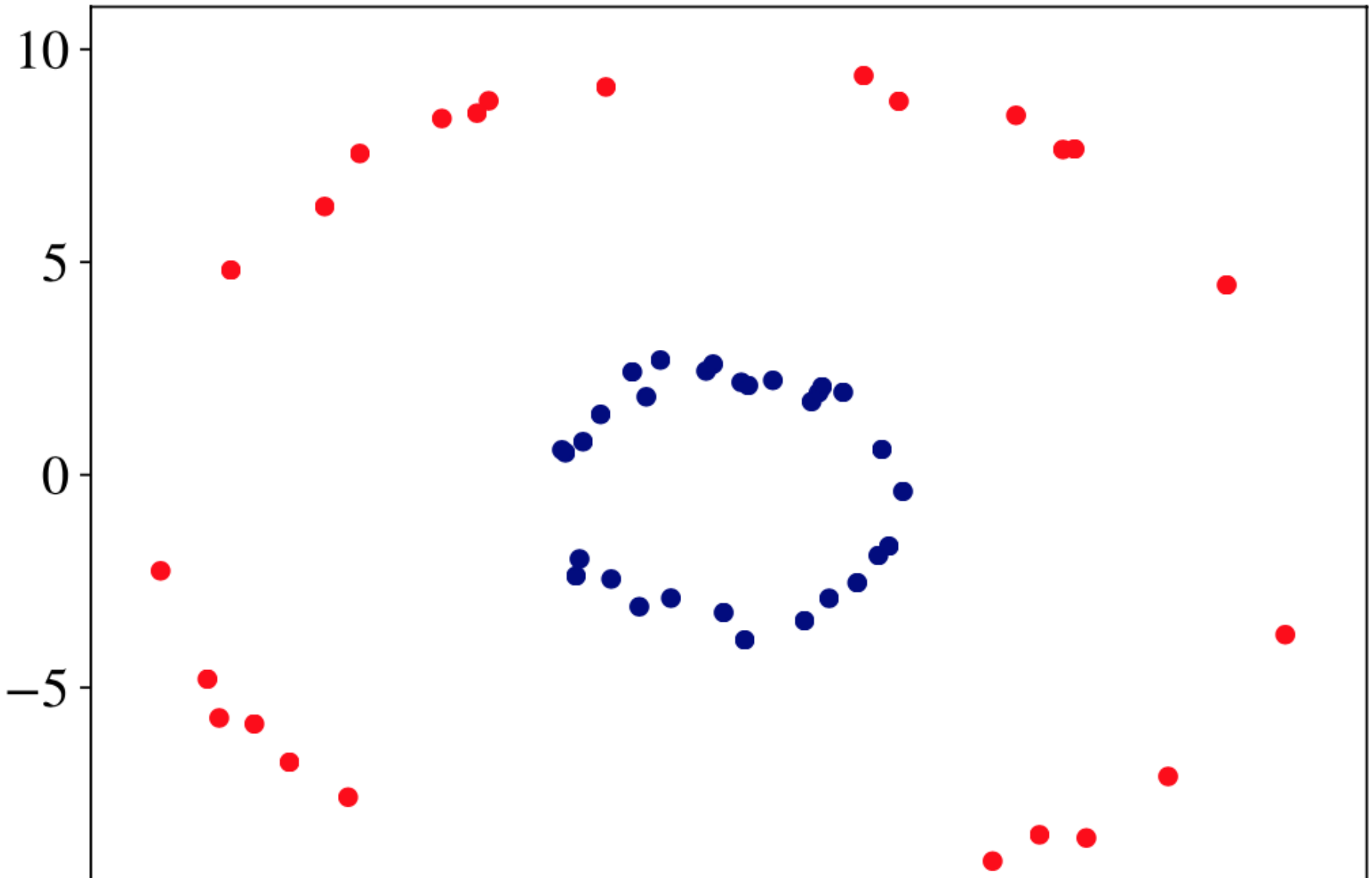
- New objective: *soft-margin SVM*

$$\text{minimize} \quad C||\mathbf{w}||^2 + \frac{1}{N}\sum_{i=1}^{N}\max(0, 1 - y_i(\mathbf{w}\mathbf{x}_i - b))$$

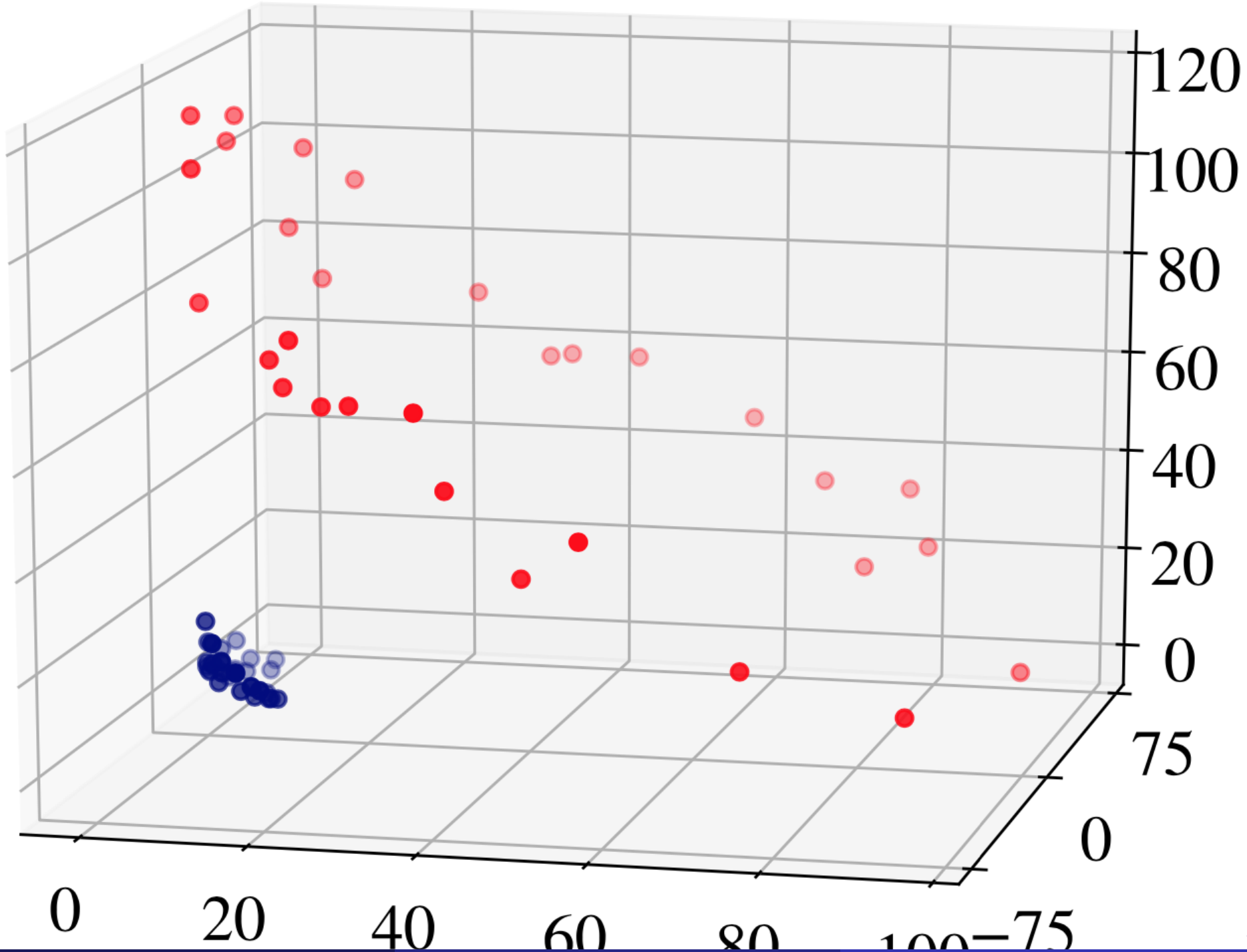
- Hyperparameter C: tradeoff between
  - increasing the size of the decision boundary
  - ensuring that each $\mathbf{x}_i$ lies on the correct side of the decision boundary
  - value for C:
    - high $\rightarrow$ ignore misclassification
    - low $\rightarrow$ classification errors more costly
    - must be chosen experimentally


8
6
4
2
0

SVM: dealing with inherent non-linearity



- transform the original space into a space of higher dimensionality
- hope: examples will become linearly separable in this space
- SVMs can use a function to implicitly doing this: kernel trick


120
100
80
60
40
20
0
75
0
−75
0
20
40
60
80
100

SVM: kernel trick

- Transform *feature space* into a higher dimentionality space: $\phi : \mathbf{x} \mapsto \phi(\mathbf{x})$
  - *e.g.*, $\phi([q, p]) \stackrel{\text{def}}{=\!=} (q^2, \sqrt{2}qp, p^2)$
- Problem: we don't know a priori which mapping would work for our data
- Kernel functions:
  - tool to efficiently work in higher-dimensional spaces
  - no need to do this transformation explicitly
- Application in SVMs:
  - using the method of *Lagrange multipliers*
  - $\alpha_i \rightarrow$ multiplier associated to example i
  - optimization problem becomes:

$$\max_{\alpha_1,\ldots,\alpha_N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} y_i \alpha_i (\mathbf{x}_i \mathbf{x}_k) y_k \alpha_k$$

$$\text{subject to:} \quad \sum_{i=1}^{N} \alpha_i y_i = 0, \alpha_i \geq 0 \qquad i = 1, \ldots, N$$

SVM: kernel trick

$$\max_{\alpha_1,\ldots,\alpha_N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} y_i \alpha_i (\mathbf{x}_i \mathbf{x}_k) y_k \alpha_k$$

$$\text{subject to:} \quad \sum_{i=1}^{N} \alpha_i y_i = 0, \alpha_i \geq 0 \qquad i = 1, \ldots, N$$

- convex quadratic optimization problem
  - efficiently solvable
- term $(\mathbf{x}_i \mathbf{x}_k)$
  - only place where the feature vectors are used
  - transform vector space into higher dimensional space:
    - transform $\mathbf{x}_i$ into $\phi(\mathbf{x}_i)$ and $\mathbf{x}_k$ into $\phi(\mathbf{x}_k)$
    - then multiply $\phi(\mathbf{x}_i)$ by $\phi(\mathbf{x}_k)$
    - $\rightarrow$ *would be very costly*
- we are only interested in the result of the dot-product $(\mathbf{x}_i \mathbf{x}_k)$
  - real number
  - replace that by operation on original feature vectors with the same result

SVM: kernel examples

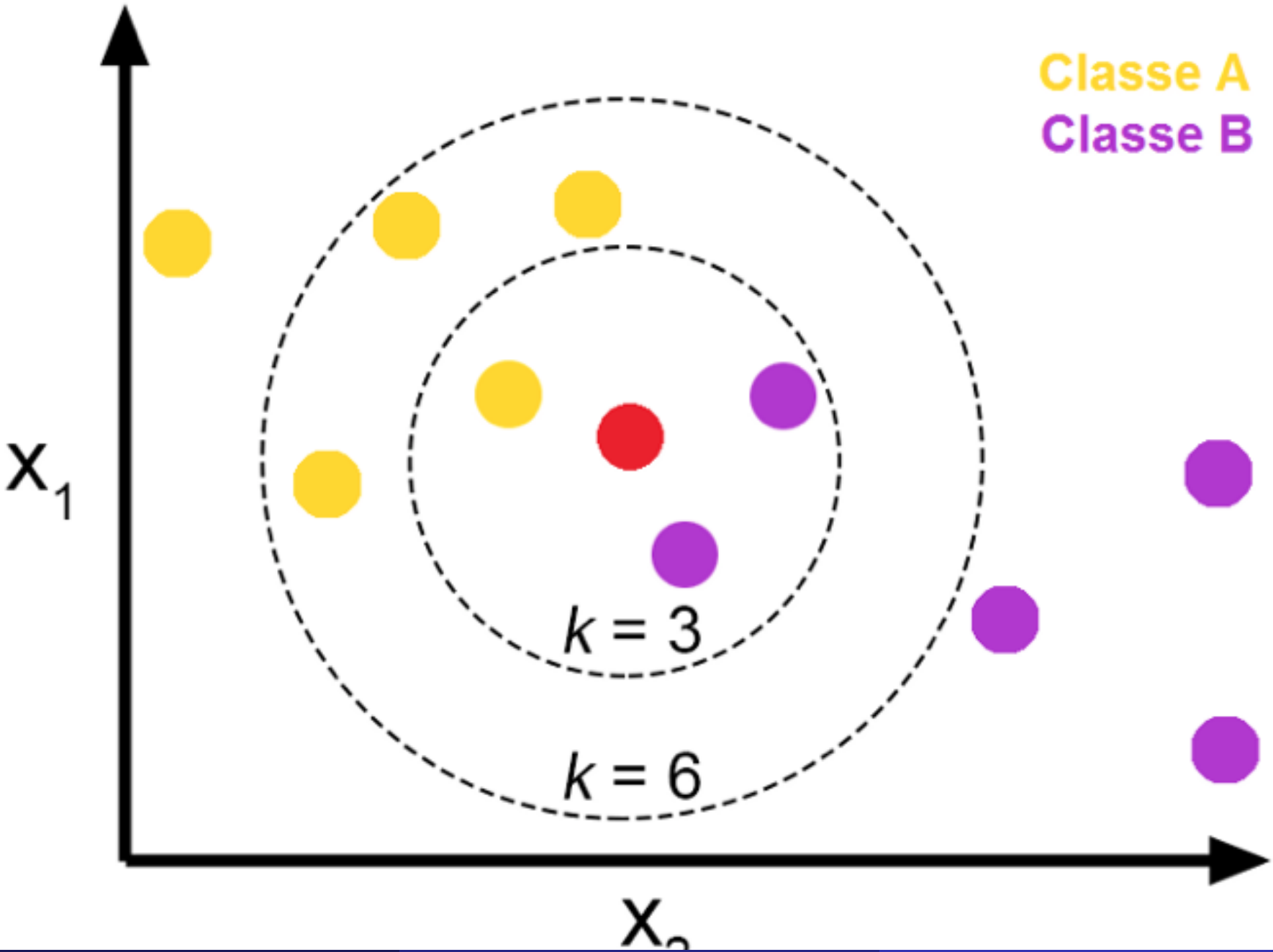
- previous example: quadratic kernel
  - transforming q_1, p_1 into $(q_1^2, \sqrt{2}q_1p_1, p_1^2)$
  - transforming q_2, p_2 into $(q_2^2, \sqrt{2}q_2p_2, p_2^2)$
  - computing dot-product $(q_1^2, \sqrt{2}q_1p_1, p_1^2) \cdot (q_2^2, \sqrt{2}q_2p_2, p_2^2)$
  - yields the same result as:
    - dot-product between (q_1, p_1) and $(q_2, p_2) \rightarrow (q_1q_2 + p_1p_2)$
    - then square it
    - same result $(q_1^2q_2^2 + 2q_1q_2p_1p_2 + p_1^2p_2^2)$
  - quadratic kernel: $k(\mathbf{x}_1, \mathbf{x}_j) \stackrel{\text{def}}{=\!=} (\mathbf{x}_i\mathbf{x}_k)^2$
- most widely used: *radial basis function*, RBF kernel

$$k(\mathbf{x}, \mathbf{x}') \stackrel{\text{def}}{=\!=} \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

  - $||\mathbf{x} - \mathbf{x}'||^2 \rightarrow$ *Euclidean distance*
  - infinite number of dimensions (series expansion of exp)
  - hyperparameter $\sigma \rightarrow$ controls how smooth or curvy decision boundary is in the original space

k-Nearest Neighbors

- Non-parametric learning algorithm
- kNN keeps all training examples in memory
- Once a previously unseen example **x** comes in:
  - algorithm finds k training examples closest to **x**
  - returns:
    - classification $\rightarrow$ majority label
    - regression $\rightarrow$ average label
- Closeness of two examples: given by a distance function
- Popular choices:
  - Euclidean distance
  - Negative cosine similarity
    - angle between two vectors
  - Chebychev distance
  - Hamming distance
  - ...
- Distance metric could also be *learned from data*


Classe A
Classe B
x_1
$k = 3$
$k = 6$
x_2

Summary: fundamental algorithms

5 basic algorithms:

- Linear Regression
- Logistic Regression
- Decision Tree Learning
- Support Vector Machine
- k-Nearest Neighbors