Multiclass Classification

Motivation

- We have seen linear models for binary classification
- Learning algorithms for linear models
  - Perceptron, Pocket, Adaline, Logistic Regression
- In all cases, the prediction is simple
  - Given an example $x, y = sgn(wTx)$
  - Output is a single bit (0 or 1), (-1 or 1)
- What if our data contains k $> \mathbf{2}$ classes ($\boldsymbol{y} \in \{\mathbf{1}, \mathbf{1}, \mathbf{2}, \ldots \boldsymbol{k}\}$)?
  - Combining binary classifiers
    1. One-vs-all
    2. one-vs-one
    3. Error correcting codes

What is multiclass classification?

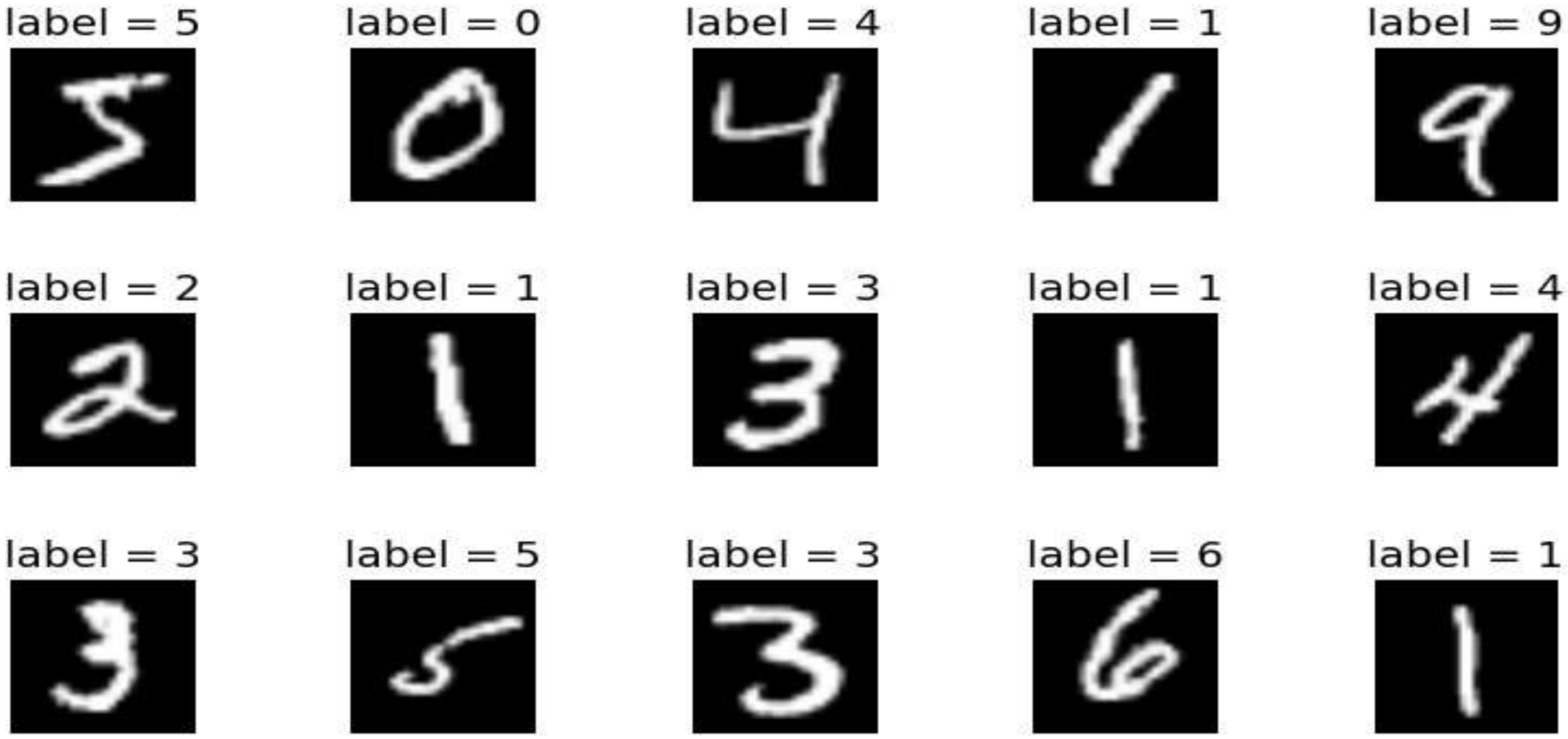
- An instance can belong to one of k classes
- Training data: Instance with class label (a number from 1 to k)
- Prediction: Given a new input, predict the class label

Each input belongs to exactly one class. Not more, not less.

- Otherwise, the problem is not multiclass classification
- If an input can be assigned multiple labels (think tags for emails rather than folders), it is called *multi-label classification*

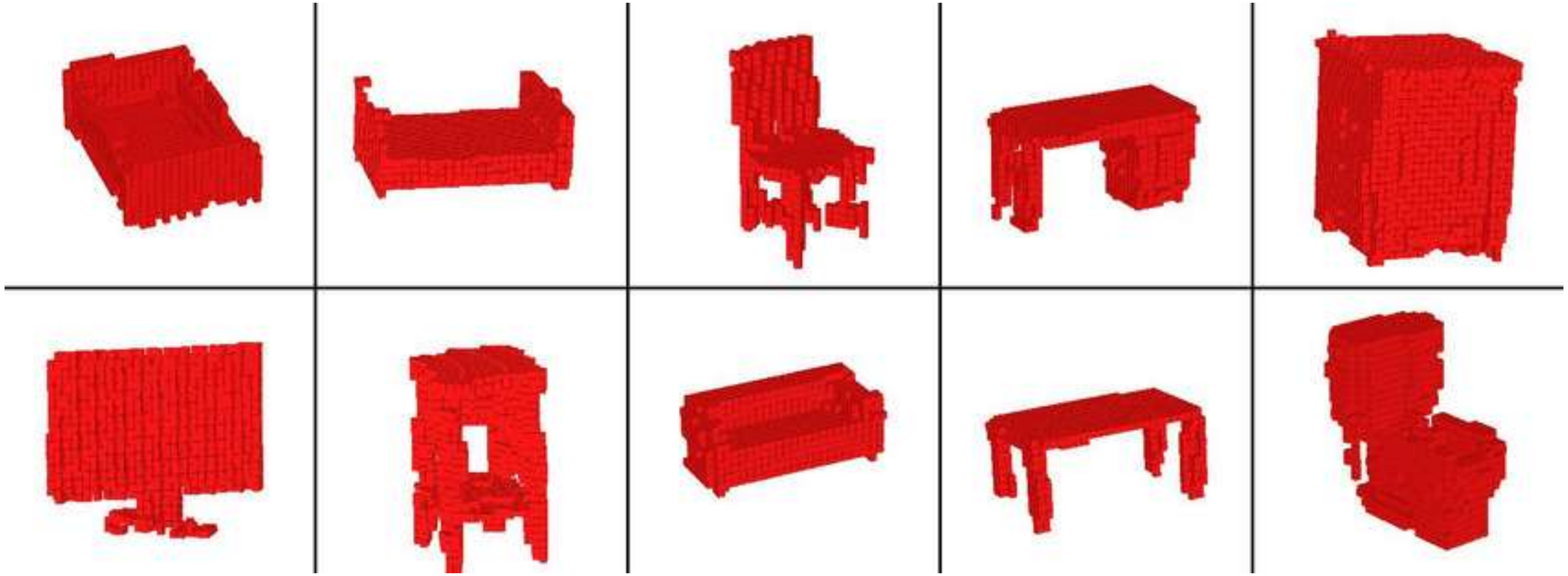
Example applications: computer vision

- *Input*: 2D hand-written character; *Output*: which character?



Example applications: computer vision

- *Input*: 3D object; *Output*: which of a set of categories of objects is it?



Binary to multiclass

- Can we use a binary classifier to construct a multiclass classifier?
  – Decompose the prediction into multiple binary decisions

- How to decompose?
  – One-vs-all
  – All-vs-all
  – Error correcting codes

General setting

- Instances: $x \in \mathbb{R}^n$
  - The inputs are represented by their feature vectors
- Output $y \in \{1, 2, \ldots, k\}$
  - These classes represent domain-specific labels
- Learning: Given a dataset $S = \{(x_i, y_i)\}_{i=1}^{m}$
  - Need to specify a learning algorithm that takes uses D to construct a function that can predict y given x
  - Goal: find a predictor that does well on the training data and has low generalization error
- Prediction: Given an example x and the learned hypothesis
  - Compute the class label for x

One-vs-all classification

Assumption: Each class individually separable from ***all*** the others

- Learning: Given a dataset $S = \{(x_i, y_i)\}_{i=1}^{m}$

  Note: $x_i \in \mathbb{R}^n$, $y_i \in \{1, 2, \dots, k\}$

  – Decompose into k binary classification tasks

  – For class $c \in \{1, 2, \dots, k\}$, construct a binary classification task as:

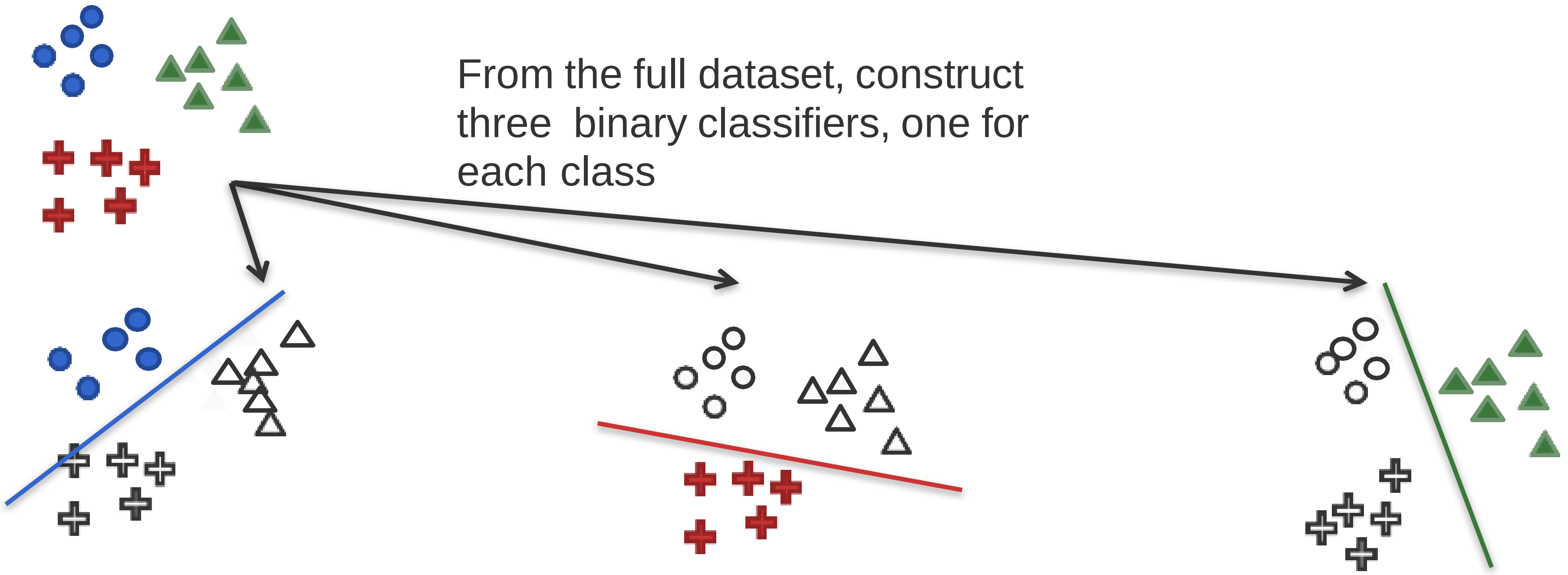
    - Positive examples: Elements of S with label c
    - Negative examples: All other elements of S

  – Train k binary classifiers $w_1, w_2, \dots w_k$ using any learning algorithm we have seen

- Prediction: Winner Takes All $y = \operatorname*{argmax}_{i=1,\dots,k} w_i^T x$

Visualizing One-vs-all

From the full dataset, construct three binary classifiers, one for each class



$w_{blue}^T x > 0$ for **blue inputs**

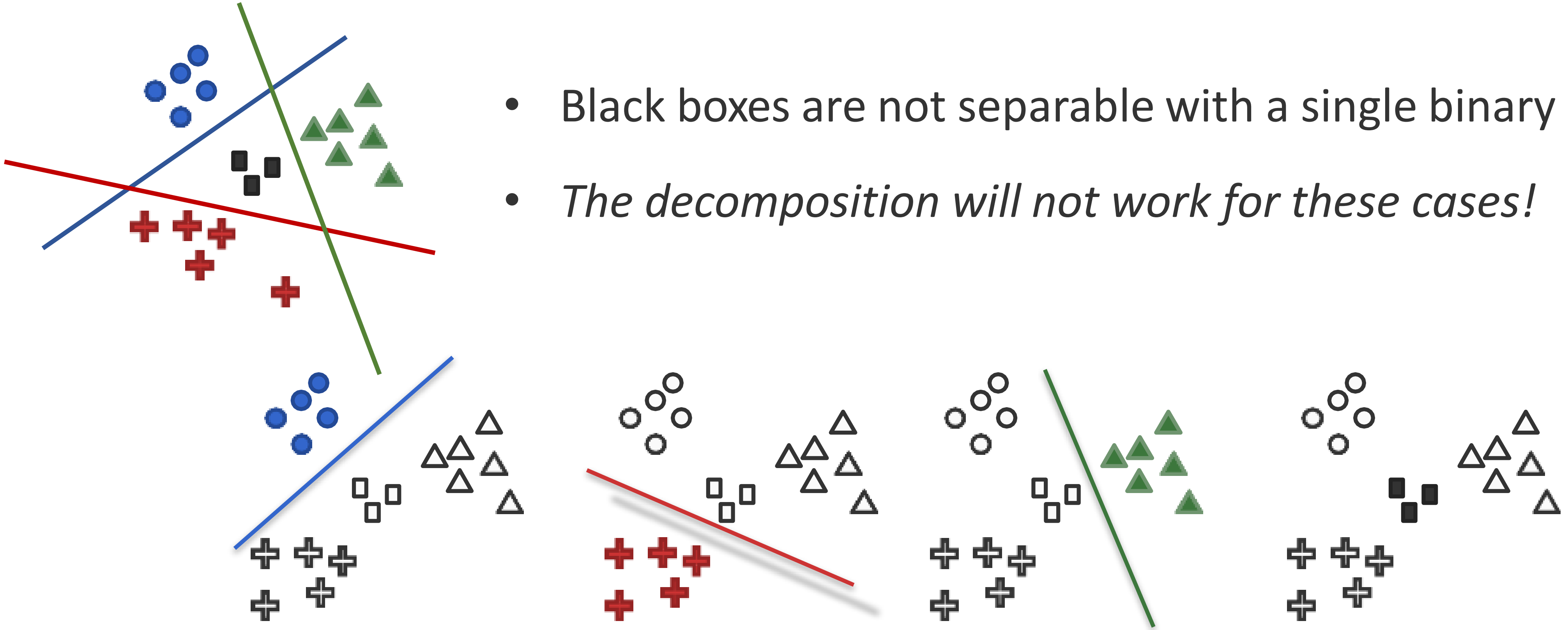
$w_{red}^T x > 0$ for **red inputs**

$w_{green}^T x > 0$ for **green inputs**

Notation: Score for blue label

For this case, Winner Take All will predict the right answer. Only the correct label will have a positive score

One-vs-all may not always work



- Black boxes are not separable with a single binary classifier
- *The decomposition will not work for these cases!*

$\mathbf{w}_{blue}^T\mathbf{x} > 0$
for
blue
inputs

$\mathbf{w}_{re}{}^T\mathbf{x} > 0$
for
red
input
s

$\mathbf{w}_{gree}{}^T\mathbf{x} > 0$
for
green
inputs

??
?

One-vs-all classification: Summary

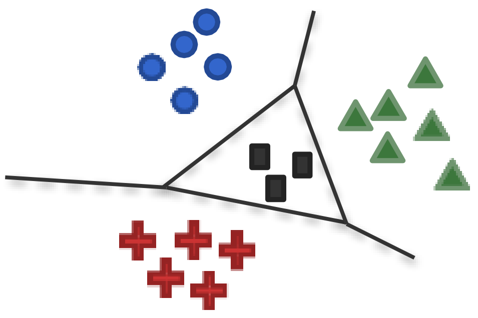
- Easy to learn
  - Use any binary classifier learning algorithm
- Problems
  - No theoretical justification
  - Calibration issues
    - We are comparing scores produced by K classifiers trained independently. No reason for the scores to be in the same numerical range!
  - Might not always work
    - Yet, works fairly well in many cases, especially if the underlying binary classifiers are well tuned

One-vs-One classification

- Assumption: Every pair of classes is separable
- Learning: Given a dataset $S = \{(x_i, y_i)\}_{i=1}^{m}$,
  - Note: $x_i \in \mathbb{R}^n, \quad y_i \in \{1, 2, \ldots, k\}$
  - For every pair of labels (i, j), create a binary classifier with:
    - Positive examples: All examples with label i
    - Negative examples: All examples with label j
  - Train $\binom{k}{2} = \frac{k(k-1)}{2}$ classifiers in all

- Prediction: More complex, each label get $k - 1$ votes
  - How to combine the votes? Many methods
    - Majority: Pick the label with maximum votes
    - Organize a tournament between the labels

All-vs-all classification



- Every pair of labels is linearly separable here
  - When a pair of labels is considered, all others are ignored

- Problems with this approach?
  - $O(k^2)$ weight vectors to train and store
  - Size of training set for a pair of labels could be very small, leading to overfitting
  - Prediction is often ad-hoc and might be unstable

    Eg: What if two classes get the same number of votes?
    For a tournament, what is the sequence in which the labels compete?

Error correcting output codes (ECOC)

- Each binary classifier provides one bit of information
- With K labels, we only need $\log_2 K$ bits
  - One-vs-all uses K bits (one per classifier)
  - All-vs-all uses $O(K^2)$ bits
- Can we get by with O(log K) classifiers?
  - Yes! Encode each label as a binary string
  - Or alternatively, if we do train more than O(log K) classifiers, can we use the redundancy to improve classification accuracy?

Using $\boldsymbol{log_2}$K classifiers

#	Code		
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

8 classes, code-length = 3

- Learning:
  - Represent each label by a bit string
  - Train one binary classifier for each bit
- Learning:
  - Use the predictions from all the classifiers to create a $\log_2$N bit string that uniquely decides the output

- What could go wrong here?
  - Even if one of the classifiers makes a mistake, final prediction is wrong!
  - How do we fix this problem?

Error correcting output code

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

Answer: Use redundancy

- Assign a binary string with each label
  - Could be random
  - Length of the code word $L \geq \log_2 K$ is a parameter
- Train one binary classifier for each bit
  - Effectively, split the data into random dichotomies
  - We need only $\log_2 K$ bits
    - Additional bits act as an error correcting code
- One-vs-all is a special case.
  - How?

How to predict?

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

- Prediction
  - Run all L binary classifiers on the example
  - Gives us a predicted bit string of length L
  - Output = label whose code word is "closest" to the prediction
  - Closest defined using Hamming distance
    - Longer code length is better, better error-correction
- Example
  - Suppose the binary classifiers here predict 11010
  - The closest label to this is 6, with code word 11000

Summary: Decomposition for multiclass classification methods

- General idea
  - Decompose the multiclass problem into many binary problems
  - We know how to train binary classifiers
  - Prediction depends on the decomposition
    - Constructs the multiclass label from the output of the binary classifiers
- Learning optimizes *local correctness*
  - Each binary classifier does not need to be globally correct
    - That is, the classifiers do not need to agree with each other
  - The learning algorithm is not even aware of the prediction procedure!
- Poor decomposition gives poor performance
  - Difficult local problems, can be "unnatural"
    - Eg. For ECOC, why should the binary problems be separable?