Multiclass Classification

Machine Learning

THE UNIVERSITY OF UTAH

So far: Binary Classification

- We have seen linear models
- Learning algorithms for linear models
  - Perceptron, Winnow, Adaboost, SVM
  - We will see more soon: Naïve Bayes, Logistic Regression
- In all cases, the prediction is simple
  - Given an example **x**, prediction = sgn($\mathbf{w}^T\mathbf{x}$)
  - Output is a single bit

What about decision trees and nearest neighbors? Is the output a single bit here too?

Multiclass classification

- Introduction: What is multiclass classification?

- Combining binary classifiers
  - One-vs-all
  - All-vs-all
  - Error correcting codes

At the end of the semester: Training a single classifier

- Multiclass SVM
- Constraint classification

Where are we?

- Introduction: What is multiclass classification?

- Combining binary classifiers
  - One-vs-all
  - All-vs-all
  - 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: Images

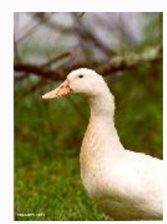
- *Input*: hand-written character; *Output*: which character?

all map to the letter A

- *Input*: a photograph of an object; *Output*: which of a set of categories of objects is it?
  - Eg: the Caltech 256 dataset

Car tire

Car tire



Duck

laptop

Example applications: Language

- *Input*: a news article
  Output: which section of the newspaper should it belong to?

- *Input*: an email
  Output: which folder should an email be placed into?

- *Input*: an audio command given to a car;
  Output: which of a set of actions should be executed?

Where are we?

- Introduction: What is multiclass classification?

- Combining binary classifiers
  - One-vs-all
  - All-vs-all
  - Error correcting codes

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: $\mathbf{x} \in \Re^n$
  - The inputs are represented by their feature vectors
- Output $\mathbf{y} \in \{1, 2, \cdots, K\}$
  - These classes represent domain-specific labels

- Learning: Given a dataset D = $\{\langle \mathbf{x}_i, \mathbf{y}_i \rangle\}$
  - 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**

1. One-vs-all classification

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

- Learning: Given a dataset D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>},

  Note: $\mathbf{x}_i \in \Re^n$, $\mathbf{y}_i \in \{1, 2, \cdots, K\}$
  - Decompose into K binary classification tasks
  - For class k, construct a binary classification task as:
    - Positive examples: Elements of D with label k
    - Negative examples: All other elements of D
  - Train K binary classifiers $\mathbf{w}_1$, $\mathbf{w}_2$, $\cdots$ $\mathbf{w}_K$ using any learning algorithm we have seen

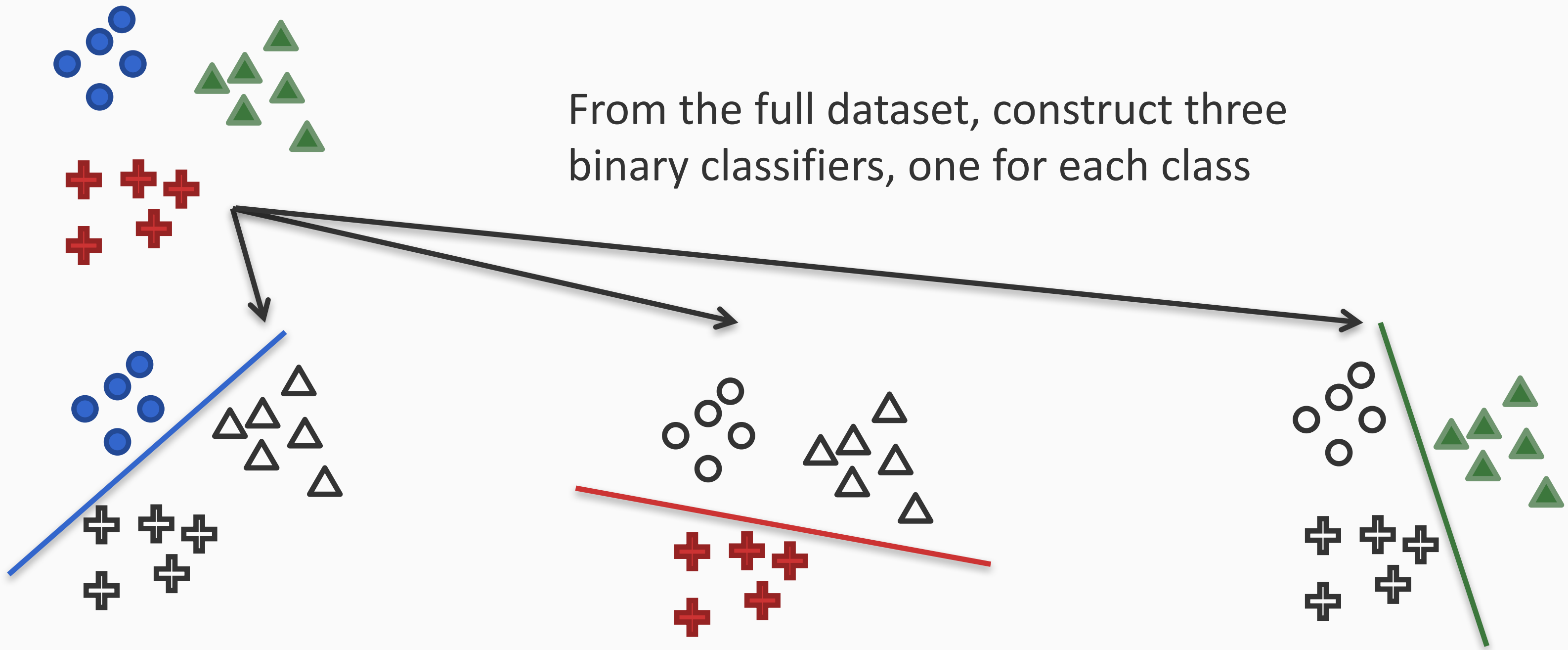
- Prediction: "*Winner Takes All*"

$$\text{argmax}_i \ \mathbf{w}_i^T \mathbf{x}$$

Question: What is the dimensionality of each $\mathbf{w}_i$?

Visualizing One-vs-all

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



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

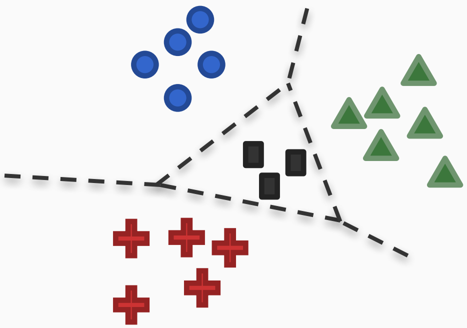
$\mathbf{w}_{\mathbf{red}}^{T}\mathbf{x} > 0$ for **red inputs**

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

Notation: Score for blue label

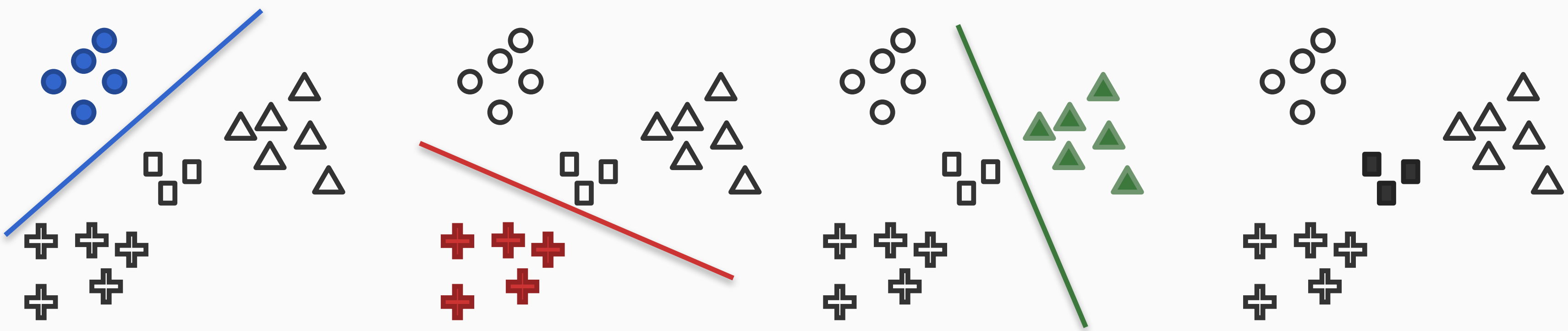
For this case, Winner Take All will predict the right answer. Only the correct label will have a positive score

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!



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

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

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

Side note about Winner Take All prediction

- If the final prediction is winner take all, is a bias feature useful?
  - Recall bias feature is a constant feature for all examples
  - Winner take all:

$$\text{argmax}_i \ \mathbf{w}_i^T \mathbf{x}$$

- Answer: No
  - The bias adds a constant to all the scores
  - Will not change the prediction

2. All-vs-all classification

Sometimes called one-vs-one

Assumption: *Every* pair of classes is separable

- Learning: Given a dataset D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>},

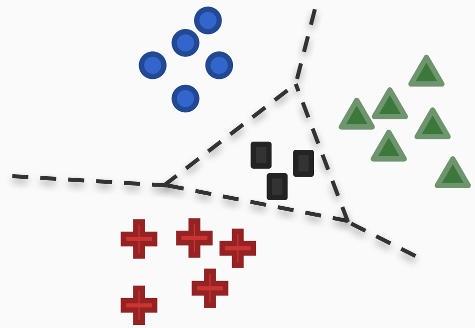
  Note: $\mathbf{x}_i \in \Re^n$, $\mathbf{y}_i \in \{1, 2, \cdots, K\}$

  – For every pair of labels (j, k), create a binary classifier with:
    - Positive examples: All examples with label j
    - Negative examples: All examples with label k

  – 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?
  1. $O(K^2)$ weight vectors to train and store
  2. Size of training set for a pair of labels could be very small, leading to overfitting
  3. 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?

3. 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 $\log_2 K$ 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
- Prediction:
  - 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

Answer: Use redundancy

- Assign a binary string with each label
  - Could be random
  - Length of the code word $L >= \log_2 K$ 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?

#	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

How to predict?

- 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

#	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

Error correcting codes: Discussion

- Assumes that columns are independent
  - Otherwise, ineffective encoding
- Strong theoretical results that depend on code length
  - If minimal Hamming distance between two rows is d, then the prediction can correct up to (d-1)/2 errors in the binary predictions
- Code assignment could be random, or designed for the dataset/task
- One-vs-all and all-vs-all are special cases
  - All-vs-all needs a ternary code (not binary)

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?

Questions?

Coming up later

- Decomposition methods
  - Do not account for how the final predictor will be used
  - Do not optimize any global measure of correctness
- Goal: To train a multiclass classifier that is “global”