Multi-Class SVM

What is multiclass classification?

- An input can belong to one of K classes
- Training data: Input associated with class label (one of K-classes)
- 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 object is it ?, from a set of categories
  - 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?

Multiclass Classification

- SVM is essentially for 2-class classification
- Many approaches to handle multiple classes
  - Combining binary classifiers
    - One-against-all (OAA-SVM)
    - One-against-one (OAO-SVM)
    - Decision Tree Structure (DAG-SVM)
    - Binary Tree structure (BT-SVM)
    - Error correcting codes (ECOC)
  - Training a single classifier
    - Multi-class optimization
    - Constraint classification

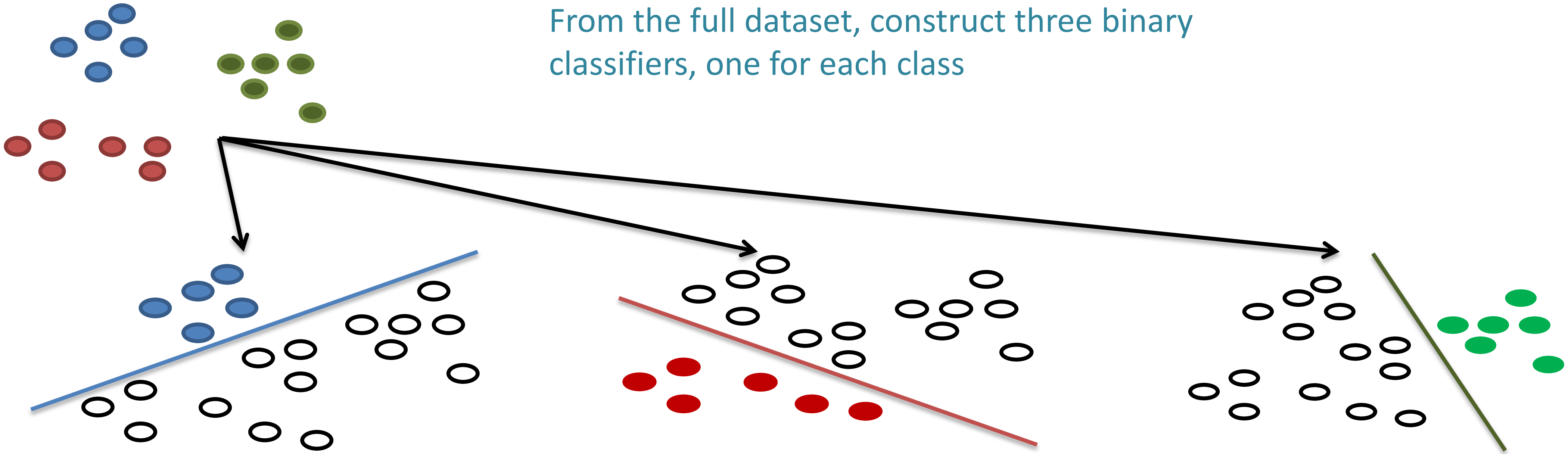
One-against-all classification

- Assumption: Each class individually separable from ***all*** the others
- **k** binary SVMs, each responsible to distinguish a class *i* from the remaining classes.
- The final prediction is usually given by the classifier with the highest output value.
- *Training*- Each classifier is trained with entire training set.
- *Testing*- Test sample is tested with k SVMs.
- Prediction: "*Winner Takes All*"

$$argmax_i \, \boldsymbol{w}_i^{\boldsymbol{T}} \boldsymbol{x} + \boldsymbol{b}_{\boldsymbol{i}}$$

Visualizing One-against-all

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



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

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

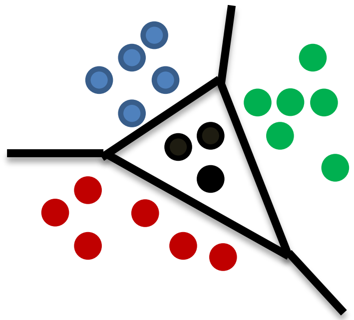
$\boldsymbol{w_{blue}}^T\boldsymbol{x} > 0$
for **green inputs**

Notation: Score for blue label

– *Winner Take All will predict the right answer. Only the correct label will have a positive score*

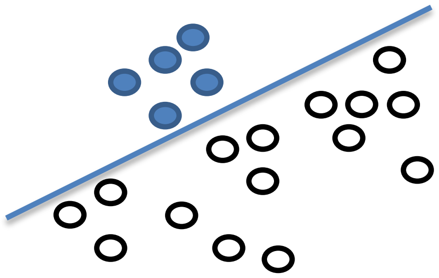
Winner take all: $argmax_i\, \boldsymbol{w}_i^T \boldsymbol{x} + \boldsymbol{b_i}$

One-against-all may not always work

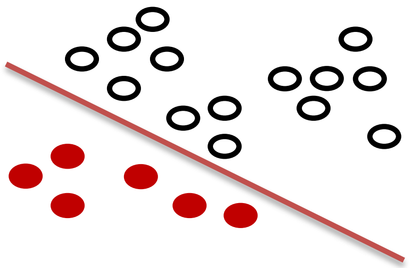


Black points are not separable with a single binary classifier

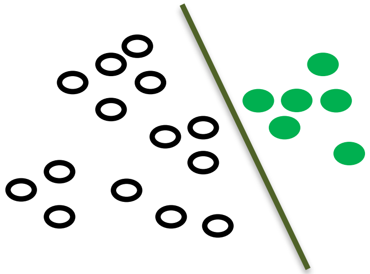
The decomposition will not work for these cases!



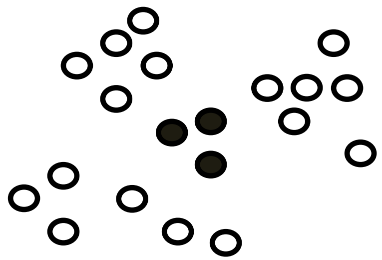
$\boldsymbol{w}_{blue}{}^{T}\boldsymbol{x} \ > \ 0$
for **blue inputs**



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



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



???

One-against-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

One-against-one classification

Sometimes called All-against-all

- Assumption: Every pair of classes is separable
- Learning: 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
- Prediction: $k(k - 1)/2$ binary SVMs- Each classifier gives one vote to its preferred class. The class with most of the votes is winner.

One-against-one classification

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

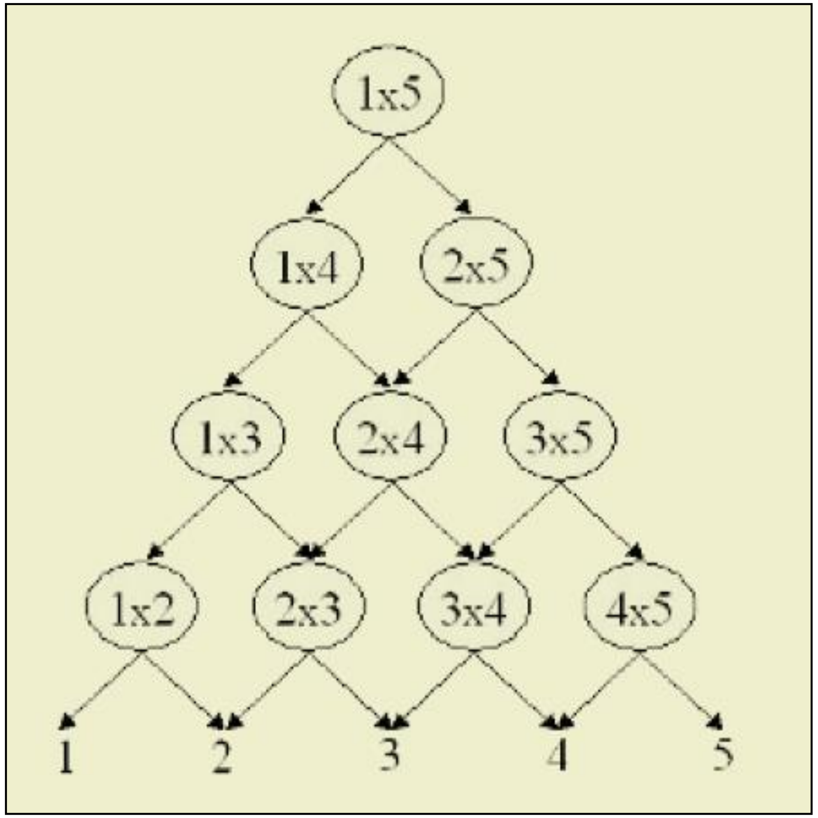
     Eg: What if two classes get the same number of votes?

DAG-SVM

- Training is same as OAO
- An advantage of using a DAG is that its testing time is less than the OAO methods.

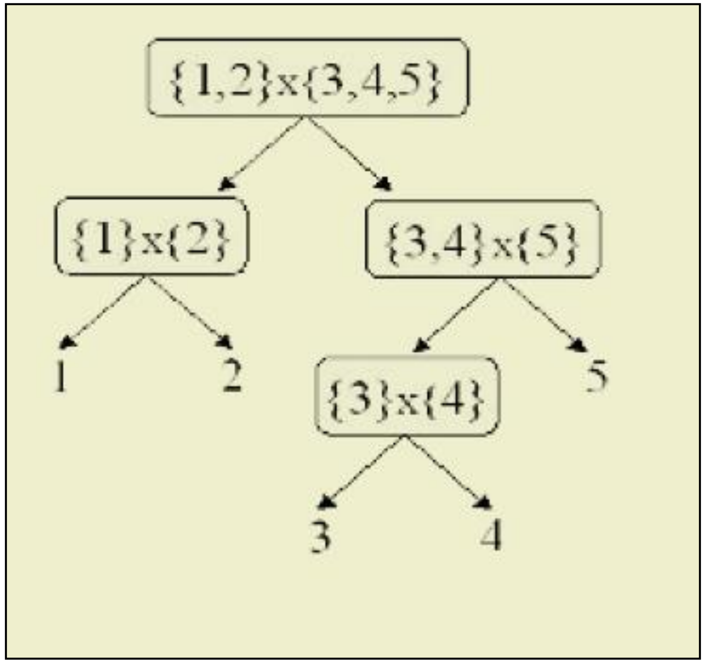
Testing:

- DAG first evaluates with the binary SVM that corresponds to the first and last element in list *L.*
- *If the* classifier prefers one of the two classes, the other one is eliminated from the list.
- Each time, a class label is excluded which results in k-1 binary evaluations.
- Individual binary SVM tend to over-fit the concerned 2-class training samples.



Tree-Structure SVM

- Binary-tree Structure:
- A binary SVM in the non leaf nodes.
- k-1 SVMs for an k class problem, Only $\log_2 k$ SVMs in average are required to classify a sample.
- Training - Each SVM is trained with small set of samples.
  - Dramatic improvement in training/test complexity
  - Performance of all models are more or less similar at this stage.



Binary Tree Structure

- Hierarchical Multiclass model Binary tree structure seems to be the best option.
- Critical decision- Binarization
- Prediction is dramatically fast.

Binarization

- Recursively dividing the classes into two disjoint groups in every node of the tree and training a SVM at this node
- Many ways to divide k classes into two groups
- To have proper grouping for the good performance
  - Divide based on class labels
  - Divide based on feature vectors
  - Hybrid

Challenge: Optimal Binarization

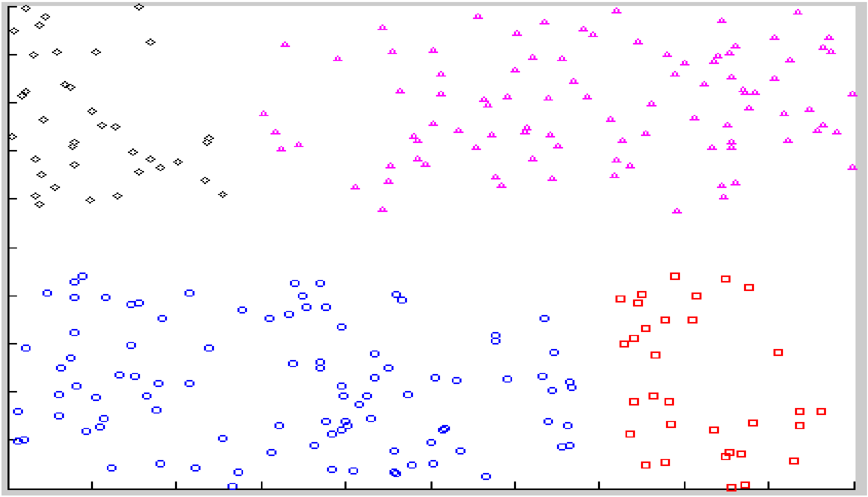
Strengths

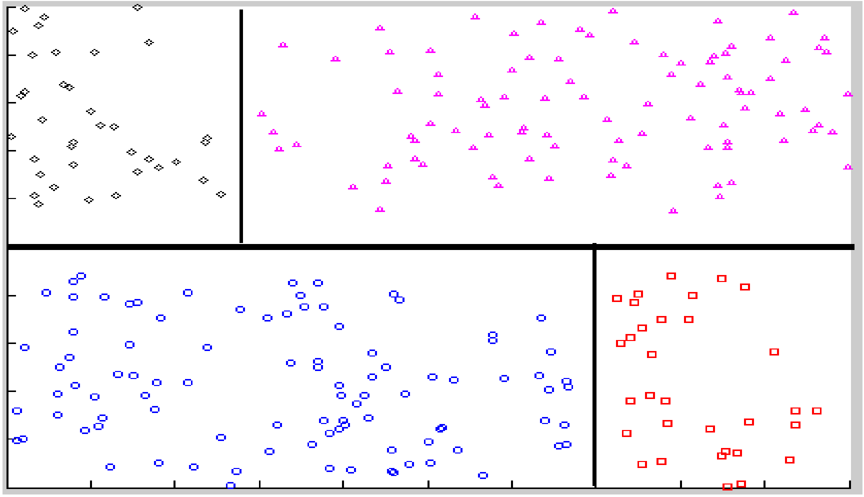
- Training and Testing complexity
- Early termination (even by just a few SVM evaluation)
- Discriminating order
- Stage-wise Linear separability

Weakness

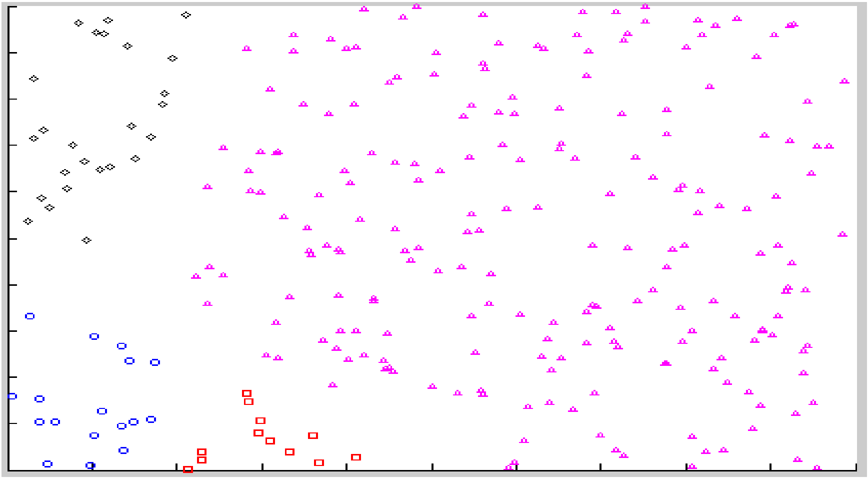
- Suboptimal 2-stage procedure
  - Tree construction is different from classifier learning
  - First use some measures about the separability of classes to partition
  - Then find a decision boundary to separate them
  - The measure of separability different from the separation margins achieved by decision boundaries
- Class-centers or mean-distance is used. A class is taken as single unit by default

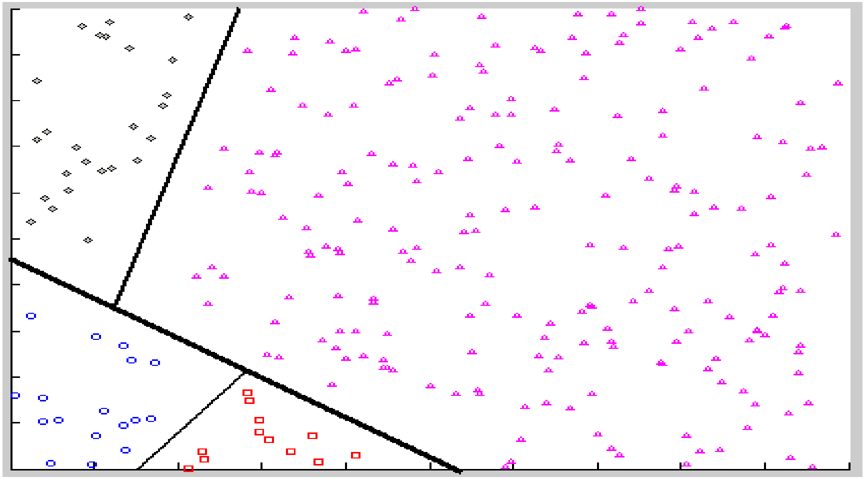
Toy Example 1





Toy Example 2





Error correcting output codes (ECOC)

- Each binary classifier provides one bit of information
- With K labels, we only need $\log_2 K$ bits
  - One-against-all uses K bits (one per classifier)
  - All-against-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

- Learning:
  - Represent each label by a bit string
  - We then build a binary learning problem for each column - in the problem for column j of the matrix, we label an instance as positive or negative depending on the value of the jth bit of the codeword corresponding to its original class
- 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!

#	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

Error correcting output code

Answer: Use redundancy

Learning:

- 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

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