CIS 519/419
Applied Machine Learning

www.seas.upenn.edu/~cis519

Dan Roth

danroth@seas.upenn.edu

http://www.cis.upenn.edu/~danroth/

461C, 3401 Walnut

Slides were created by Dan Roth (for CIS519/419 at Penn or CS446 at UIUC), Eric Eaton for CIS519/419 at Penn, or from other authors who have made their ML slides available.

Exams

- 1. Overall:
- Mean: 62 (18.6 - 13.2 - 18.7 - 10.5)
- Std Dev: 13.8 (2.5 - 6.7 - 4.4 - 5.8)
- Max: 94, Min: 27.5
- 2. CIS 519 (91 students):
- Mean: 61.48 (18.4 - 12.8 - 18.5 - 10.75)
- Std Dev: 14.7 (2.6 - 7.1 - 4.5 - 5.9)
- Max: 94   Min: 27.5
- 3. CIS 419 (47 students):
- Mean: 63.6 (19 - 14 - 19 - 10)
- Std Dev: 12 (2.2 - 5.9 - 4.1 - 5.8)
- Max: 93, Min: 41

- Solutions are available.
- Midterms will be made available at the recitations, Tuesday and Wednesday.
- This will also be a good opportunity to ask the Tas questions about the grading.

Questions?

Projects

- Please start working!

- Come to my office hours at least once in the next 3 weeks to discuss the project.

Hard SVM Optimization

- We have shown that the sought after weight vector w is the solution of the following optimization problem:

SVM Optimization: (***)

- Minimize: ½ $||w||^2$

  Subject to: ∀ (x,y) ∈ S: $y\, w^T x \geq 1$

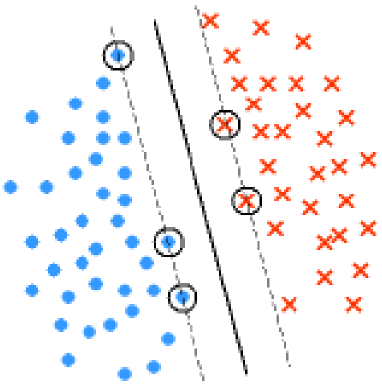
- This is a quadratic optimization problem in (n+1) variables, with |S|=m inequality constraints.
- It has a unique solution.

Duality

- This, and other properties of Support Vector Machines are shown by moving to the dual problem.

- Theorem: Let w^* be the minimizer of the SVM optimization problem (***) for $S = \{(x_i, y_i)\}$.
  Let $I = \{i: y_i (w^{*T} x_i + b) = 1\}$.
  Then there exists coefficients $\alpha_i > 0$ such that:

$$w^* = \sum_{i \in I} \alpha_i y_i x_i$$



Soft SVM

- Notice that the relaxation of the constraint:

$$y_i w^T x_i \geq 1$$

- Can be done by introducing a slack variable ξ_i (per example) and requiring:

$$y_i w^T x_i \geq 1 - \xi_i \,;\, \xi_i \geq 0$$

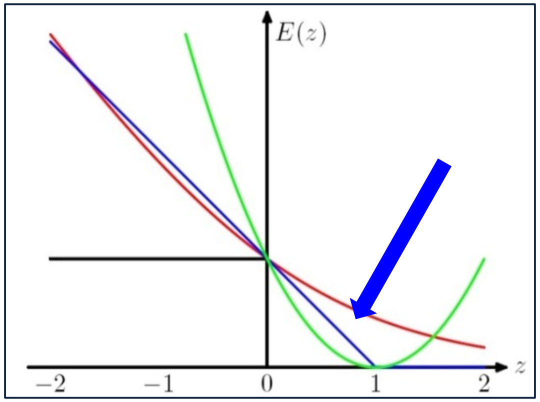
- Now, we want to solve:

$$\min_{w,\xi_i} \quad \frac{1}{2} w^T w + C \sum_i \xi_i$$

$$\text{s.t} \quad y_i w^T x_i \geq 1 - \xi_i \,;\, \xi_i \geq 0 \quad \forall i$$

A large value of C means that misclassifications are bad – we focus on a small training error (at the expense of margin). A small C results in more training error, but hopefully better true error.

Soft SVM (2)



- Now, we want to solve:

$$\min_{w,\xi_i} \ \frac{1}{2} w^T w + C \sum_i \xi_i$$

$$\text{s.t} \quad \xi_i \geq 1 - y_i w^T x_i; \ \xi_i \geq 0 \quad \forall i$$

In optimum, $\xi_i = \max(0, 1 - y_i w^T x_i)$

- Which can be written as:

$$\min_{w} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i w^T x_i) \, .$$

- What is the interpretation of this?

SVM Objective Function

- The problem we solved is:

$$\text{Min } ½ \,||w||^2 + c \sum \xi_i$$

- Where $\xi_i > 0$ is called a slack variable, and is defined by:
    - $\xi_i = \max(0, 1 - y_i\, w^t x_i)$
    - Equivalently, we can say that: $y_i\, w^t x_i$ ¸ $1 - \xi_i;\ \xi_i \geq 0$
- And this can be written as:

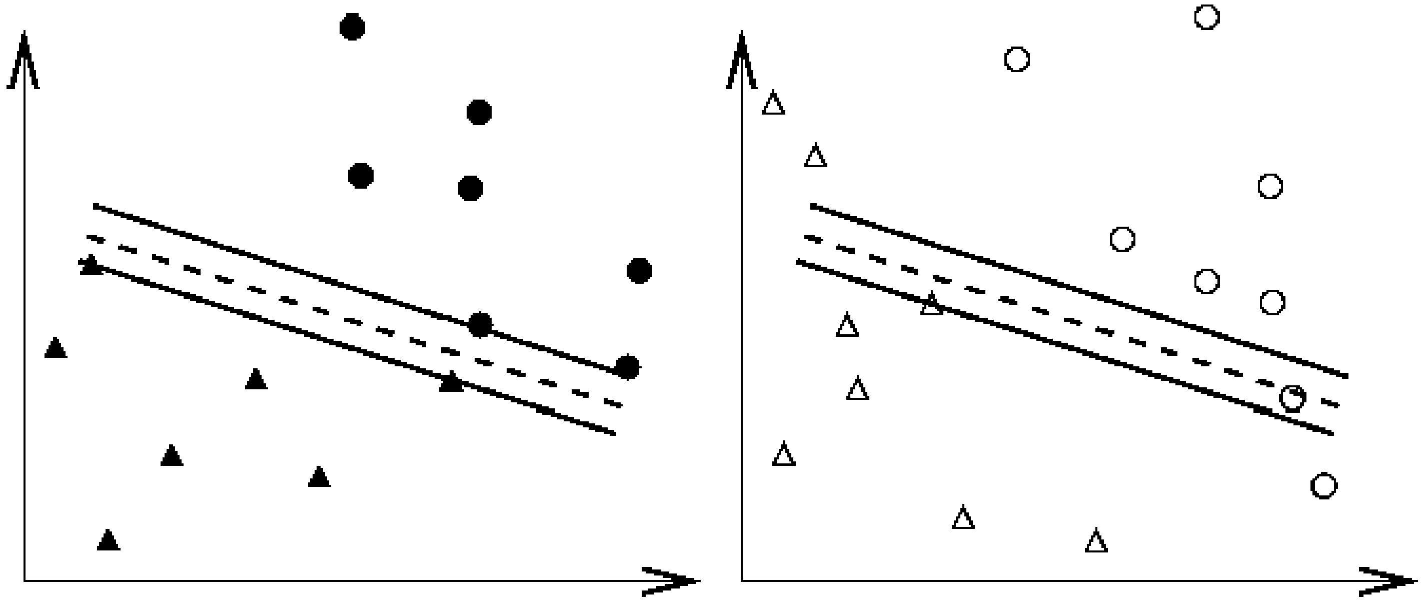
$$\text{Min } \underbrace{½ \,||w||^2}_{\text{Regularization term}} \quad + \quad \underbrace{c \sum \xi_i}_{\text{Empirical loss}}$$

Can be replaced by other **regularization functions**

Can be replaced by other **loss functions**

- General Form of a learning algorithm:
    - Minimize empirical loss, and Regularize (to avoid over fitting)
    - Theoretically motivated improvement over the original algorithm we've seen at the beginning of the semester.

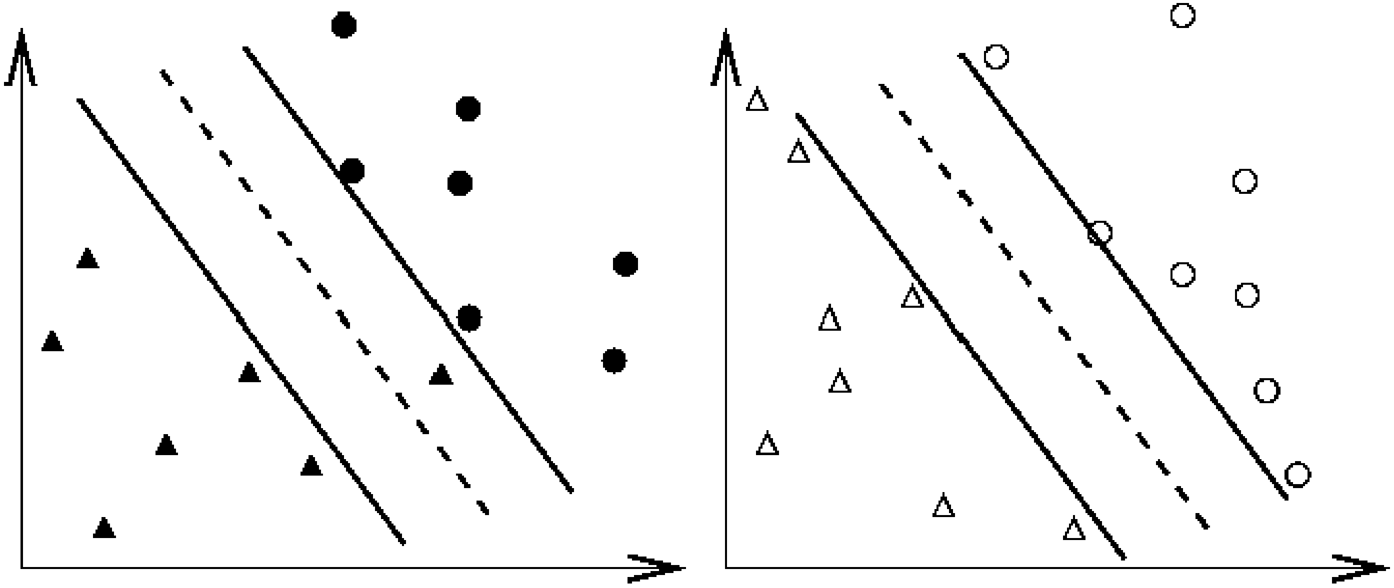
Balance between regularization and empirical loss



(a) Training data and an over-fitting classifier

(b) Testing data and an over-fitting classifier

Balance between regularization and empirical loss



(c) Training data and a better classifier

(d) Testing data and a better classifier

[(DEMO)]

Optimization: How to Solve

- 1. Earlier methods used Quadratic Programming. Very slow.
- 2. The soft SVM problem is an unconstrained optimization problems. It is possible to use the gradient descent algorithm.
- Many options within this category:
  - Iterative scaling; non-linear conjugate gradient; quasi-Newton methods; truncated Newton methods; trust-region newton method.
  - All methods are iterative methods, that generate a sequence w_k that converges to the optimal solution of the optimization problem above.
  - Currently: Limited memory BFGS is very popular
- 3. 3rd generation algorithms are based on Stochastic Gradient Decent
  - The runtime does not depend on n=#(examples); advantage when n is very large.
  - Stopping criteria is a problem: method tends to be too aggressive at the beginning and reaches a moderate accuracy quite fast, but it's convergence becomes slow if we are interested in more accurate solutions.
- 4. Dual Coordinated Descent (& Stochastic Version)

SGD for SVM

- Goal: $\min_{w} f(w) \equiv \frac{1}{2}w^Tw + \frac{C}{m}\sum_i \max(0, 1 - y_i w^T x_i)$. m: data size

m is here for mathematical correctness, it doesn't matter in the view of modeling.

- Compute sub-gradient of $f(w)$:

$\nabla f(w) = w - Cy_ix_i$ if $1 - y_iw^Tx_i \geq 0$; otherwise $\nabla f(w) = w$

1. Initialize $w = 0 \in R^n$
2. For every example $(\mathrm{x_i}, \mathrm{y_i}) \in D$

   If $y_iw^Tx_i \leq 1$ update the weight vector to

   $w \leftarrow (1-\gamma)w + \gamma C y_i x_i$ (γ - learning rate)

   Otherwise $w \leftarrow (1-\gamma)w$
3. Continue until convergence is achieved

Convergence can be proved for a slightly complicated version of SGD (e.g, Pegasos)

This algorithm should ring a bell…

Nonlinear SVM

- We can map data to a high dimensional space: $\mathrm{x} \rightarrow \phi(x)$ (DEMO)
- Then use Kernel trick: $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ (DEMO2)

Primal:

$$\min_{w,\xi_i} \frac{1}{2} w^T w + C \sum_i \xi_i$$

s.t $\mathrm{y_i w^T} \phi(x_i) \geq 1 - \xi_i$

$\xi_i \geq 0 \quad \forall i$

Dual:

$$\min_{\alpha} \frac{1}{2} \alpha^T \mathrm{Q} \alpha - e^T \alpha$$

s.t $0 \leq \alpha \leq C \;\; \forall i$

$\mathrm{Q}_{ij} = y_i \, y_j K(x_i, x_j)$

Theorem: Let w* be the minimizer of the primal problem, α^* be the minimizer of the dual problem.
Then $\mathrm{w}^* = \sum_i \alpha^* \mathrm{y_i x_i}$

Where are we?

- Algorithms
  - DTs
  - Perceptron + Winnow
  - Gradient Descent
  - [NN]
- Theory
  - Mistake Bound
  - PAC Learning
- We have a formal notion of "learnability"
  - We understand Generalization
    - How will your algorithm do on the next example?
  - How it depends on the hypothesis class (VC dim)
    - and other complexity parameters
- Algorithmic Implications of the theory?

Boosting

- Boosting is (today) a general learning paradigm for putting together a Strong Learner, given a collection (possibly infinite) of Weak Learners.
- The original Boosting Algorithm was proposed as an answer to a theoretical question in PAC learning. [The Strength of Weak Learnability; Schapire, 89]
- Consequently, Boosting has interesting theoretical implications, e.g., on the relations between PAC learnability and compression.
  - If a concept class is efficiently PAC learnable then it is efficiently PAC learnable by an algorithm whose required memory is bounded by a polynomial in n, size c and log(1/ε).
  - There is no concept class for which efficient PAC learnability requires that the entire sample be contained in memory at one time – there is always another algorithm that "forgets" most of the sample.

Boosting Notes

- However, the key contribution of Boosting has been practical, as a way to compose a good learner from many weak learners.
- It is a member of a family of Ensemble Algorithms, but has stronger guarantees than others.

- A Boosting demo is available at http://cseweb.ucsd.edu/~yfreund/adaboost/

- Example
- Theory of Boosting
  - Simple & insightful

Boosting Motivation

Example: "How May I Help You?"

[Gorin et al.]

- goal: automatically categorize type of call requested by phone customer (Collect, CallingCard, PersonToPerson, etc.)
  - `yes I'd like to place a collect call long distance please` (Collect)
  - `operator I need to make a call but I need to bill it to my office` (ThirdNumber)
  - `yes I'd like to place a call on my master card please` (CallingCard)
  - `I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill` (BillingCredit)
- observation:
  - easy to find "rules of thumb" that are "often" correct
    - e.g.: "IF '`card`' occurs in utterance THEN predict 'CallingCard' "
  - hard to find single highly accurate prediction rule

The Boosting Approach

- Algorithm
  - Select a small subset of examples
  - Derive a rough rule of thumb
  - Examine 2nd set of examples
  - Derive 2nd rule of thumb
  - Repeat T times
  - Combine the learned rules into a single hypothesis

- Questions:
  - How to choose subsets of examples to examine on each round?
  - How to combine all the rules of thumb into single prediction rule?
- Boosting
  - General method of converting rough rules of thumb into highly accurate prediction rule

Theoretical Motivation

- "Strong" PAC algorithm:
  - for any distribution
  - $\forall \delta, \varepsilon > 0$
  - Given polynomially many random examples
  - Finds hypothesis with error $\leq \varepsilon$ with probability $\geq (1- \delta)$

- "Weak" PAC algorithm
  - Same, but only for some $\varepsilon \leq ½ - \Upsilon$

- [Kearns & Valiant '88]:
  - Does weak learnability imply strong learnability?
  - Anecdote: the importance of the distribution free assumption
    - It does not hold if PAC is restricted to only the uniform distribution, say

History

- [Schapire '89]:
  - First provable boosting algorithm
  - Call weak learner three times on three modified distributions
  - Get slight boost in accuracy
  - apply recursively
- [Freund '90]:
  - "Optimal" algorithm that "boosts by majority"
- [Drucker, Schapire & Simard '92]:
  - First experiments using boosting
  - Limited by practical drawbacks
- [Freund & Schapire '95]:
  - Introduced "AdaBoost" algorithm
  - Strong practical advantages over previous boosting algorithms
- AdaBoost was followed by a huge number of papers and practical applications

Some lessons for Ph.D. students

A Formal View of Boosting

- Given training set $(x_1, y_1), \ldots (x_m, y_m)$
- $y_i \in \{-1, +1\}$ is the correct label of instance $x_i \in X$
- For $t = 1, \ldots, T$
  - Construct a distribution D_t on $\{1,\ldots m\}$
  - Find weak hypothesis ("rule of thumb")

    $$h_t : X \rightarrow \{-1, +1\}$$

    with small error ε_t on D_t:

    $$\varepsilon_t = Pr_D \left[h_t (x_i) \neq y_i\right]$$
- Output: final hypothesis H_{final}

Adaboost

- Constructing D_t on {1,…m}:
  - $D_1(i) = 1/m$
  - Given D_t and h_t :
  - $D_{t+1} = \quad D_t(i)/z_t \; e^{-\alpha_t} \quad y_i = h_t(x_i)$

    $\quad D_t(i)/z_t \; e^{+\alpha_t} \quad$ if $y_i \neq h_t(x_i)$

    $= \quad D_t(i)/z_t \; \exp(-\alpha_t \, y_i \, h_t(x_i))$

    where z_t = normalization constant

    and

    $\alpha_t = ½ \ln\{ (1 - \varepsilon_t)/\varepsilon_t \}$

Think about unwrapping it all the way to 1/m

$$Z_t = \sum_i D_t(i) \exp(-\alpha_t \, y_i \, h_t(x_i))$$

< 1; smaller weight

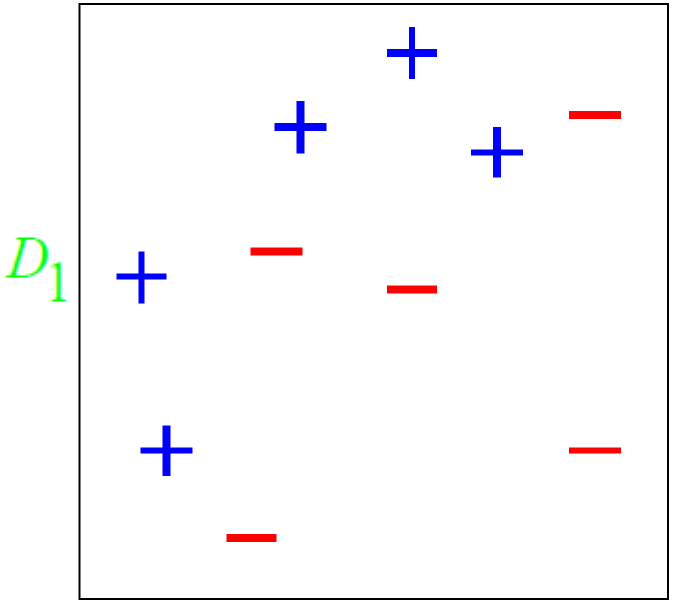
> 1; larger weight

Notes about α_t: $e^{+\alpha_t} = \mathrm{sqrt}\{(1-\varepsilon_t)/\varepsilon_t\} > 1$

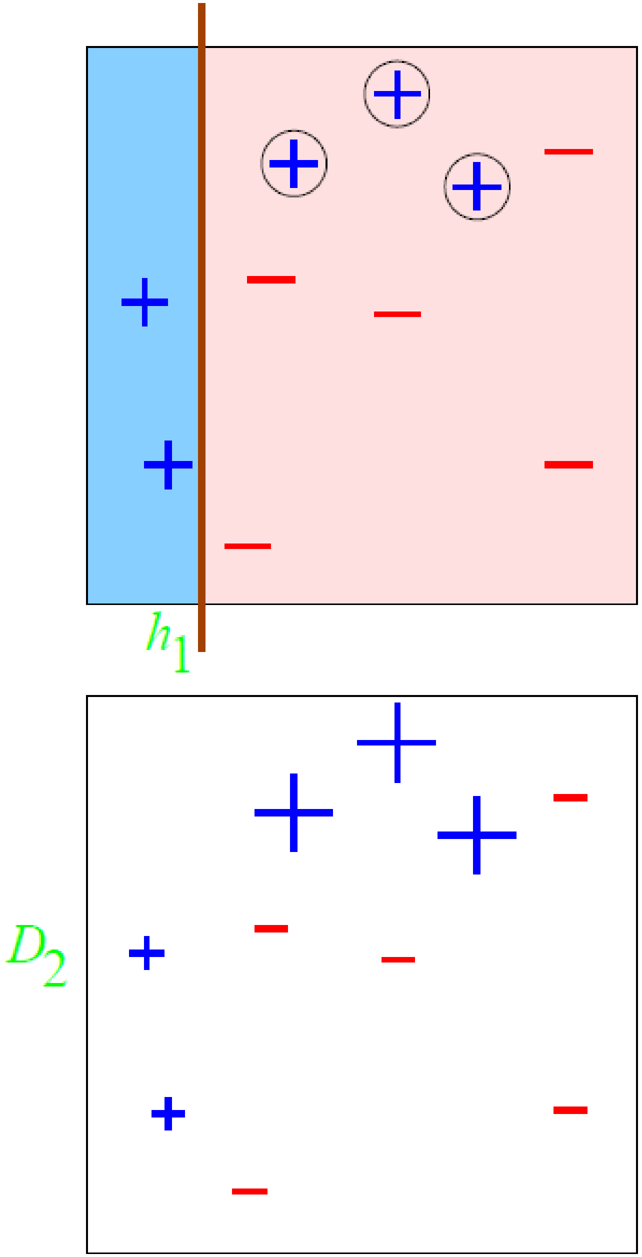
- Positive due to the weak learning assumption
- Examples that we predicted correctly are demoted, others promoted
- Sensible weighting scheme: better hypothesis (smaller error) → larger weight

- Final hypothesis: $H_{final}(x) = \mathrm{sign}(\sum_t \alpha_t h_t(x))$

A Toy Example



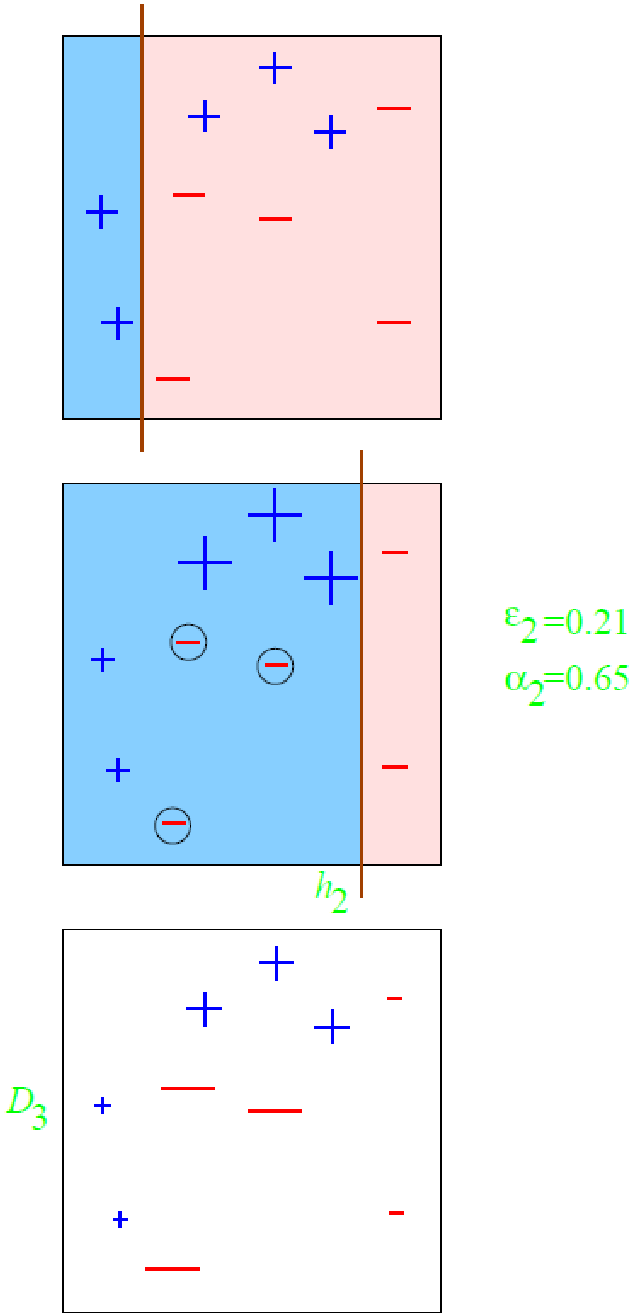
Round 1



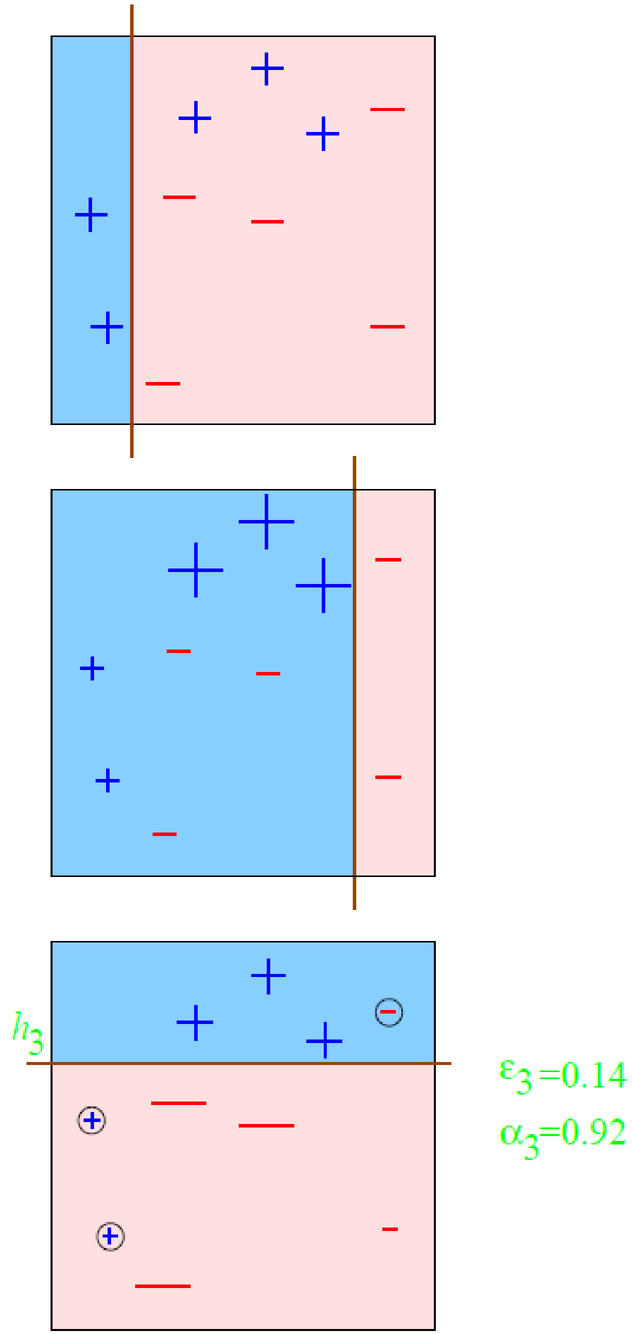
$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

Round 2

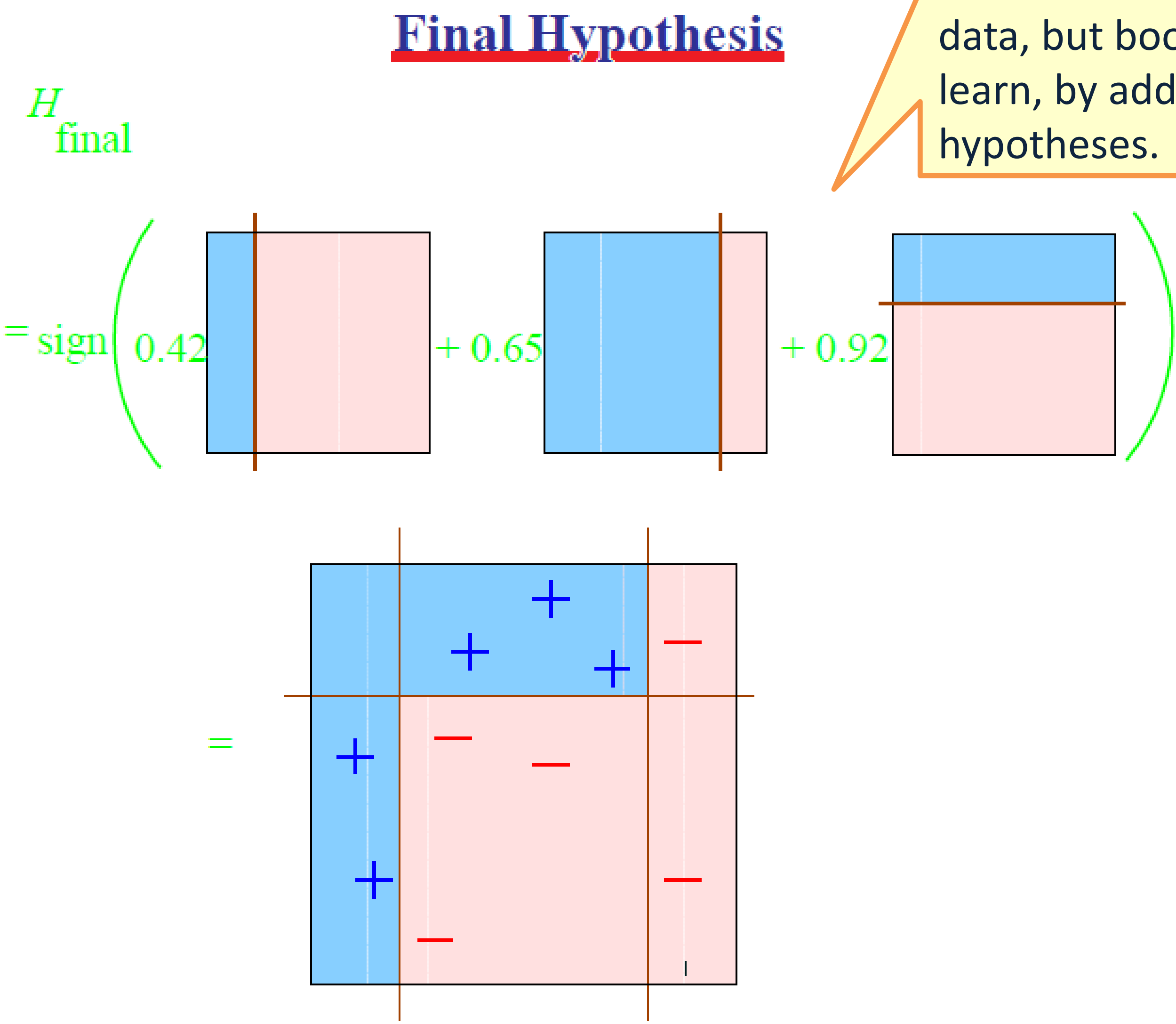


Round 3



A Toy Example

A cool and important note about the final hypothesis: it is possible that the combined hypothesis makes no mistakes on the training data, but boosting can still learn, by adding more weak hypotheses.



Analyzing Adaboost

- Theorem:
  - run AdaBoost
  - let $\epsilon_t = 1/2 - \gamma_t$
  - then

1. Why is the theorem stated in terms of minimizing training error? Is that what we want?
2. What does the bound mean?

$$\text{training error}(H_{\text{final}}) \leq \prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right]$$

Need to prove only the first inequality, the rest is algebra.

$\varepsilon_t\,(1-\ \varepsilon_t) = (1/2-\Upsilon_t)(1/2+\Upsilon_t)) = 1/4 - \Upsilon_t^2$

$1-(2\Upsilon_t)^2 \ \cdot\ \exp(-(2\Upsilon_t)^2)$

$$= \prod_t \sqrt{1-4\gamma_t^2}$$

$$\leq \exp\left(-2\sum_t \gamma_t^2\right)$$

- so: if $\forall t:\ \gamma_t \geq \gamma > 0$
  then training error$(H_{\text{final}}) \leq e^{-2\gamma^2 T}$
- adaptive:
  - does not need to know γ or T a priori
  - can exploit $\gamma_t \gg \gamma$

AdaBoost Proof (1)

Need to prove only the first inequality, the rest is algebra.

- let $f(x) = \sum_t \alpha_t h_t(x) \Rightarrow H_{\text{final}}(x) = \text{sign}(f(x))$
- *Step 1*: unwrapping recursion:

The final "weight" of the i-th example

$$D_{\text{final}}(i) = \frac{1}{m} \cdot \frac{\exp\left(-y_i \sum_t \alpha_t h_t(x_i)\right)}{\prod_t Z_t}$$

$$= \frac{1}{m} \cdot \frac{e^{-y_i f(x_i)}}{\prod_t Z_t}$$

AdaBoost Proof (2)

- *Step 2*: training error$(H_{\text{final}}) \le \prod_t Z_t$
- Proof:
  - $H_{\text{final}}(x) \neq y \Rightarrow yf(x) \le 0 \Rightarrow e^{-yf(x)} \ge 1$
  - so:

$$\text{training error}(H_{\text{final}}) = \frac{1}{m}\sum_i \begin{cases} 1 & \text{if } y_i \neq H_{\text{final}}(x_i) \\ 0 & \text{else} \end{cases}$$

$$\le \frac{1}{m}\sum_i e^{-y_i f(x_i)}$$

$$= \sum_i D_{\text{final}}(i) \prod_t Z_t$$

$$= \prod_t Z_t$$

The definition of training error

Always holds for mistakes (see above)

Using Step 1

D is a distribution over the m examples

Why does it work?
The Weak Learning Hypothesis

AdaBoost Proof(3)

A strong assumption due to the "for all distributions".
But – works well in practice

- *Step 3*: $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$
- Proof:

By definition of Z_t; it's a normalization term

$$Z_t = \sum_i D_t(i)\exp(-\alpha_t\, y_i\, h_t(x_i))$$

Splitting the sum to "mistakes" and no-mistakes"

$$= \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t}$$

The definition of ϵ_t

$$= \epsilon_t\, e^{\alpha_t} + (1-\epsilon_t)\, e^{-\alpha_t}$$

The definition of α_t

$e^{+\alpha_t} = \text{sqrt}\{(1 - \epsilon_t)/ \epsilon_t\}>1$

$$= 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

Steps 2 and 3 together prove the Theorem.
→ The error of the final hypothesis can be as low as you want.

Boosting The Confidence

- Unlike Boosting the accuracy (ε), Boosting the confidence (δ) is easy.

- Let's fix the accuracy parameter to ε.
- Suppose that we have a learning algorithm L such that for any target concept $c \in C$ and any distribution D, L outputs h s.t. $error(h) < \varepsilon$ with confidence at least $1-\delta_0$, where $\delta_0 = 1/q(n,size(c))$, for some polynomial q.

- Then, if we are willing to tolerate a slightly higher hypothesis error, $\varepsilon + \gamma$ ($\gamma > 0$, arbitrarily small) then we can achieve arbitrary high confidence $1-\delta$.

Boosting The Confidence(2)

- Idea: Given the algorithm L, we construct a new algorithm L' that simulates algorithm L k times (k will be determined later) on independent samples from the same distribution

- Let $h_1, \ldots h_k$ be the hypotheses produced. Then, since the simulations are independent, the probability that all of $h_1, \ldots h_k$ have error $>\varepsilon$ is as most $(1-\delta_0)^k$. Otherwise, at least one h_j is good.
- Solving $(1-\delta_0)^k < \delta/2$ yields that value of k we need,

$$k > (1/\delta_0) \ln(2/\delta)$$

- There is still a need to show how L' works. It would work by using the h_i that makes the fewest mistakes on the sample S; we need to compute how large S should be to guarantee that it does not make too many mistakes.

[Kearns and Vazirani's book]

Summary of Ensemble Methods

- Boosting
- Bagging
- Random Forests

Boosting

- Initialization:
  - Weigh all training samples equally
- Iteration Step:
  - Train model on (weighted) train set
  - Compute error of model on train set
  - Increase weights on training cases model gets wrong!!!
- Typically requires 100's to 1000's of iterations
- Return final model:
  - Carefully weighted prediction of each model

Boosting: Different Perspectives

- Boosting is a maximum-margin method
  (Schapire et al. 1998, Rosset et al. 2004)
  - Trades lower margin on easy cases for higher margin on harder cases
- Boosting is an additive logistic regression model (Friedman, Hastie and Tibshirani 2000)
  - Tries to fit the logit of the true conditional probabilities
- Boosting is an *equalizer*
  (Breiman 1998) (Friedman, Hastie, Tibshirani 2000)
  - Weighted proportion of times example is misclassified by base learners tends to be the same for all training cases
- Boosting is a linear classifier, over an incrementally acquired “feature space”.

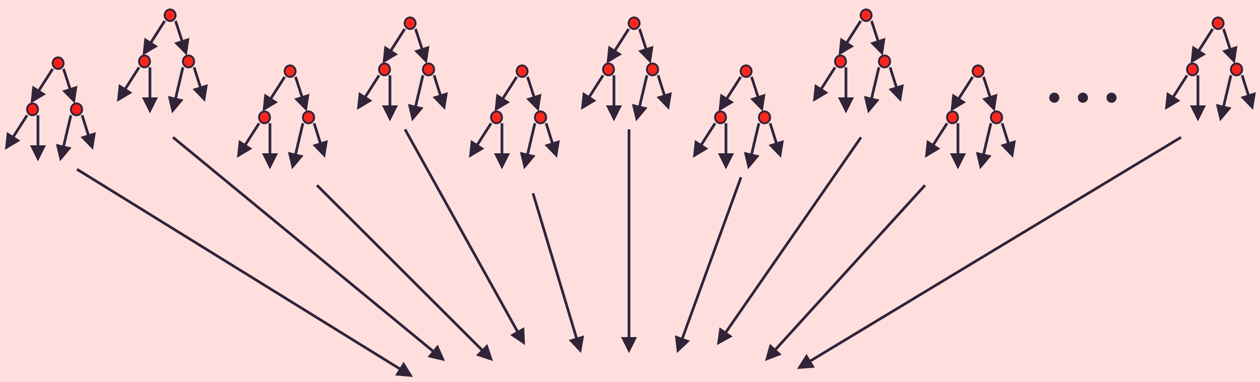
Bagging

- Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor.
- The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class.
- The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets.
  - That is, use samples of the data, with repetition

- Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy.
- The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed then bagging can improve accuracy.

Example: Bagged Decision Trees

- Draw 100 bootstrap samples of data
- Train trees on each sample → 100 trees
- Average prediction of trees on out-of-bag samples

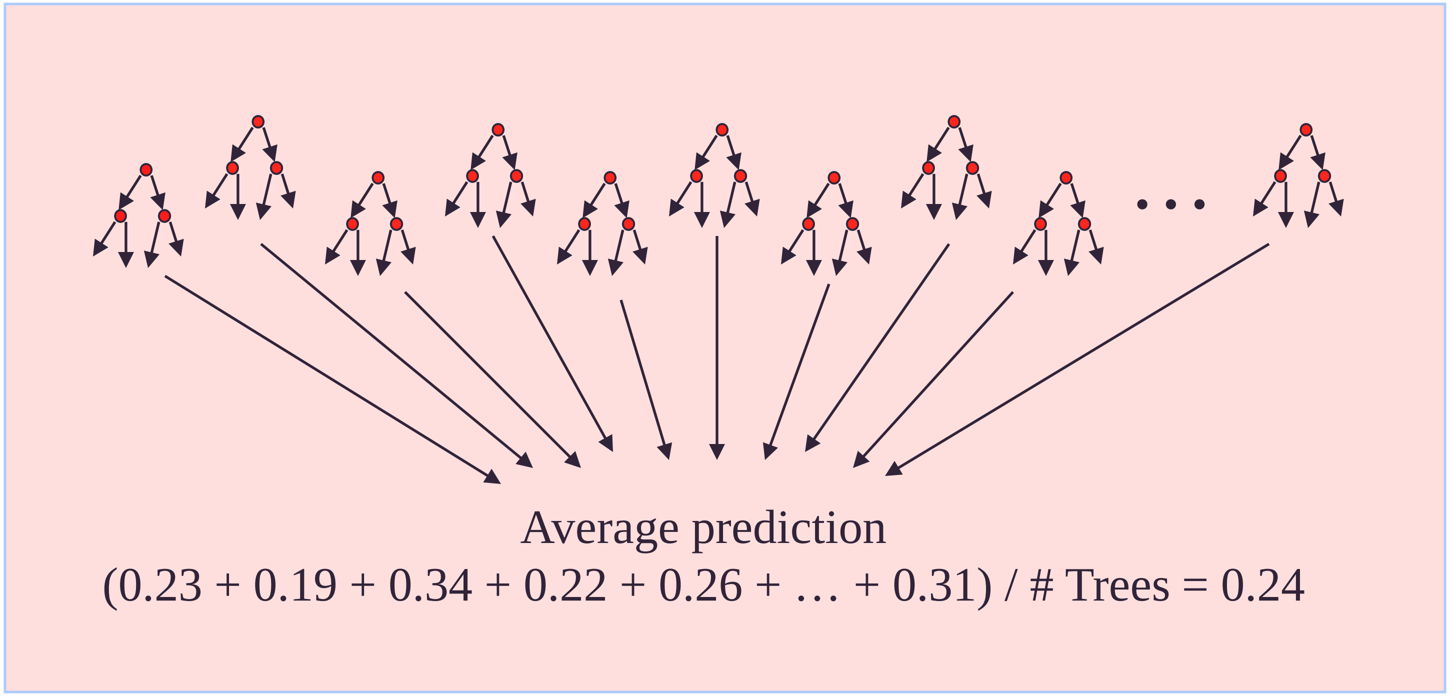


Average prediction

(0.23 + 0.19 + 0.34 + 0.22 + 0.26 + … + 0.31) / # Trees = 0.24

Random Forests (Bagged Trees++)

- Draw ***1000+*** bootstrap samples of data
- ***Draw sample of available attributes at each split***
- Train trees on each sample/attribute set → ***1000+*** trees
- Average prediction of trees on out-of-bag samples



So Far: Classification

- So far we focused on Binary Classification
- For linear models:
  - Perceptron, Winnow, SVM, GD, SGD
- The prediction is simple:
  - Given an example x,
  - Prediction = $\mathrm{sgn}(w^T x)$
  - Where w is the learned model
- The output is a single bit

Multi-Categorical Output Tasks

- Multi-class Classification ($y \in \{1,...,K\}$)
  - character recognition ('6')
  - document classification ('homepage')
- Multi-label Classification ($y \subseteq \{1,...,K\}$)
  - document classification ('(homepage,facultypage)')
- Category Ranking ($y \in \pi(K)$)
  - user preference ('(love > like > hate)')
  - document classification ('hompage > facultypage > sports')
- Hierarchical Classification ($y \subseteq \{1,..,K\}$)
  - cohere with class hierarchy
  - place document into index where 'soccer' is-a 'sport'

Setting

- Learning:
  - Given a data set $D = \{(x_i , y_i)\}_1^m$
  - Where $x_i \in R^n$, $y_i \in \{1,2,\dots,k\}$.
- Prediction (inference):
  - Given an example x, and a learned function (model),
  - Output a single class labels y.

Binary to Multiclass

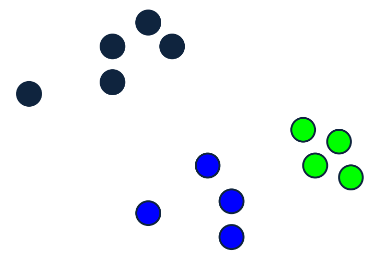
- Most schemes for multiclass classification work by reducing the problem to that of binary classification.
- There are multiple ways to decompose the multiclass prediction into multiple binary decisions
  - One-vs-all
  - All-vs-all
  - Error correcting codes
- We will then talk about a more general scheme:
  - Constraint Classification
- It can be used to model other non-binary classification schemes and leads to Structured Prediction.

One-Vs-All

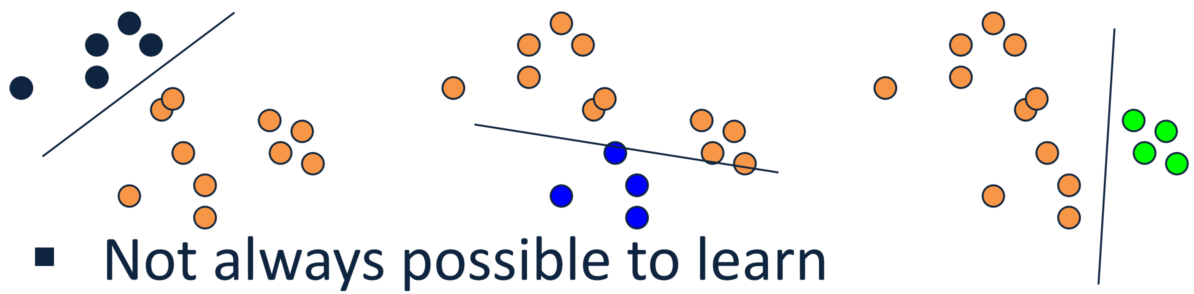
- Assumption: Each class can be separated from all the rest using a binary classifier in the hypothesis space.
- Learning: Decomposed to learning k independent binary classifiers, one for each class label.
- Learning:
  - Let D be the set of training examples.
  - $\forall$ label l, construct a binary classification problem as follows:
    - Positive examples: Elements of D with label l
    - Negative examples: All other elements of D
  - This is a binary learning problem that we can solve, producing k binary classifiers $w_1, w_2, \ldots w_k$
- Decision: Winner Takes All (WTA):
  - $f(x) = \text{argmax}_i \; w_i^T x$

Solving MultiClass with 1vs All learning

- MultiClass classifier
  - Function f : $R^n \rightarrow \{1,2,3,...,k\}$



- Decompose into binary problems

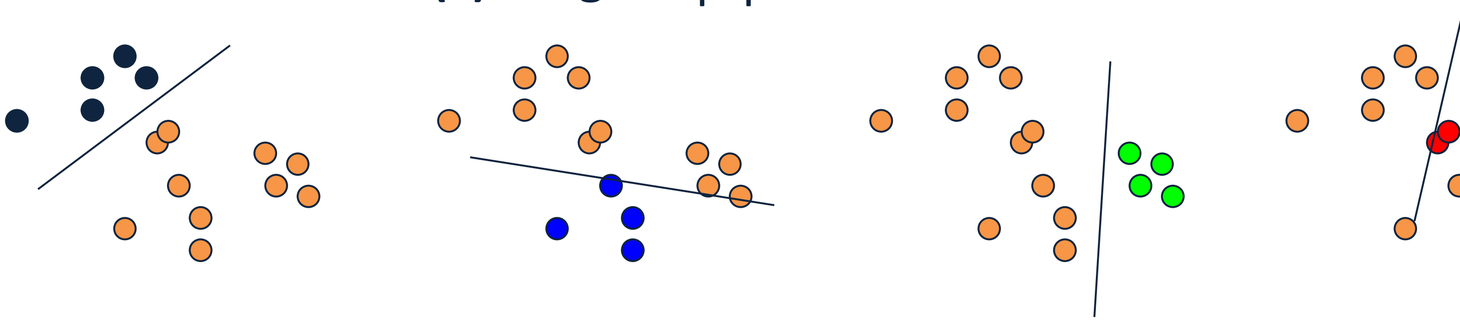


- Not always possible to learn
- No theoretical justification
  - Need to make sure the range of all classifiers is the same
- (unless the problem is easy)

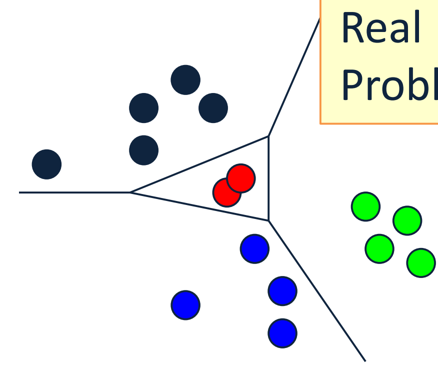
Learning via One-Versus-All (OvA) Assumption

- Find $v_r, v_b, v_g, v_y \in \mathbf{R}^n$ such that
  - $v_r.x > 0$ iff y = red ⊗
  - $v_b.x > 0$ iff y = blue √
  - $v_g.x > 0$ iff y = green √
  - $v_y.x > 0$ iff y = yellow √
- **Classification: $f(x) = \text{argmax}_i\ v_i\ x$**

$\mathbf{H} = \mathbf{R}^{nk}$



Real Problem

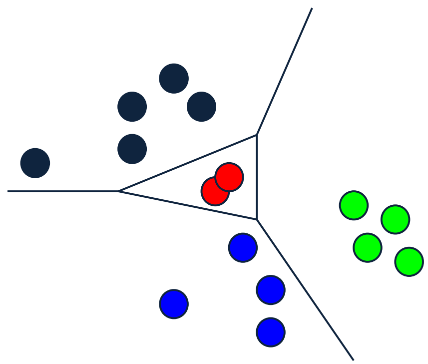


All-Vs-All

- Assumption: There is a separation between every pair of classes using a binary classifier in the hypothesis space.
- Learning: Decomposed to learning [k choose 2] ~ k^2 independent binary classifiers, one corresponding to each pair of class labels. For the pair (i, j):
  - Positive example: all exampels with label i
  - Negative examples: all examples with label j
- Decision: More involved, since output of binary classifier may not cohere. Each label gets k-1 votes.
- Decision Options:
  - Majority: classify example x to take label i if i wins on x more often than j (j=1,…k)
  - A tournament: start with n/2 pairs; continue with winners .

Learning via All-Verses-All (AvA) Assumption

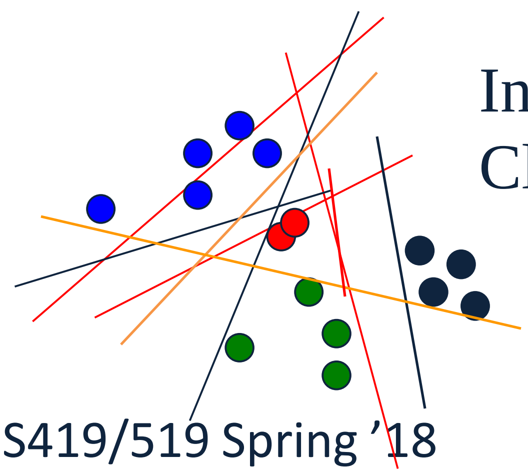
- Find $v_{rb}, v_{rg}, v_{ry}, v_{bg}, v_{by}, v_{gy} \in \mathbf{R}^d$ such that
  - $v_{rb}.x > 0$ if y = red
    < 0 if y = blue
  - $v_{rg}.x > 0$ if y = red
    < 0 if y = green
  - ... (for all pairs)



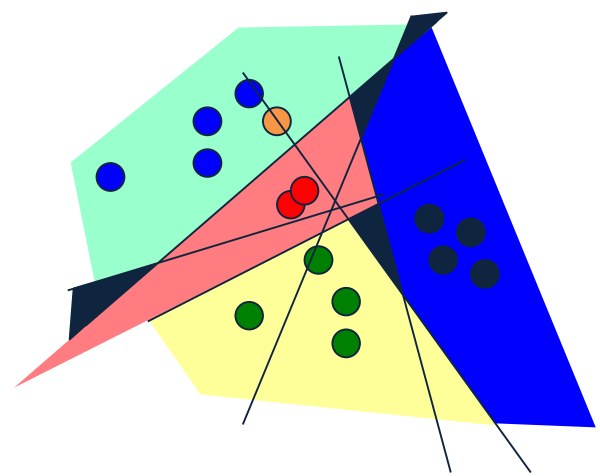
It is possible to separate all k classes with the $O(k^2)$ classifiers

$H = R^{kkn}$

How to classify?



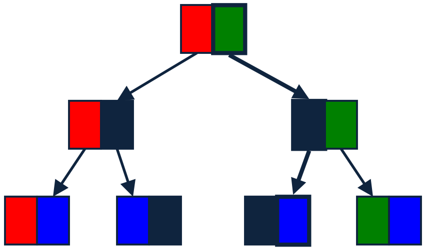
Individual Classifiers



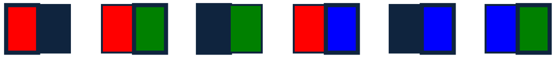
Decision Regions

Classifying with AvA

Tournament



Majority Vote



1 red, 2 yellow, 2 green

➔ ?

All are post-learning and *might* cause weird stuff

One-vs-All vs. All vs. All

- Assume m examples, k class labels.
  - For simplicity, say, m/k in each.
- One vs. All:
  - classifier f_i: m/k (+) and $(k-1)m/k$ (-)
  - Decision:
  - Evaluate k linear classifiers and do Winner Takes All (WTA):
  - $f(x) = \text{argmax}_i\, f_i(x) = \text{argmax}_i\, w_i^T x$
- All vs. All:
  - Classifier f_{ij}: m/k (+) and m/k (-)
  - More expressivity, but less examples to learn from.
  - Decision:
  - Evaluate k^2 linear classifiers; decision sometimes unstable.
- What type of learning methods would prefer All vs. All (efficiency-wise)?

(Think about Dual/Primal)

Error Correcting Codes Decomposition

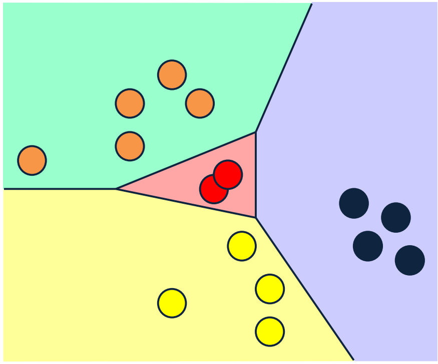
- 1-vs-all uses k classifiers for k labels; can you use only $\log_2 k$?
- Reduce the multi-class classification to random binary problems.
  - Choose a "code word" for each label.
  - K=8: all we need is 3 bits, three classifiers
- Rows: An encoding of each class (k rows)
- Columns: L dichotomies of the data, each corresponds to a new classification problem
- Extreme cases:
  - 1-vs-all: k rows, k columns
  - k rows $\log_2 k$ columns
- Each training example is mapped to one example per column
  - (x,3) → {(x,P1), +; (x,P2), -; (x,P3), -; (x,P4), +}
- To classify a new example x:
  - Evaluate hypothesis on the 4 binary problems {(x,P1) , (x,P2), (x,P3), (x,P4),}
  - Choose label that is most consistent with the results.
    - Use Hamming distance (bit-wise distance)
- Nice theoretical results as a function of the performance of the P_i s (depending on code & size)
- Potential Problems?

Label	P1	P2	P3	P4
1	-	+	-	+
2	-	+	+	-
3	+	-	-	+
4	+	-	+	+
k	-	+	-	-

Can you separate any dichotomy?

Problems with Decompositions

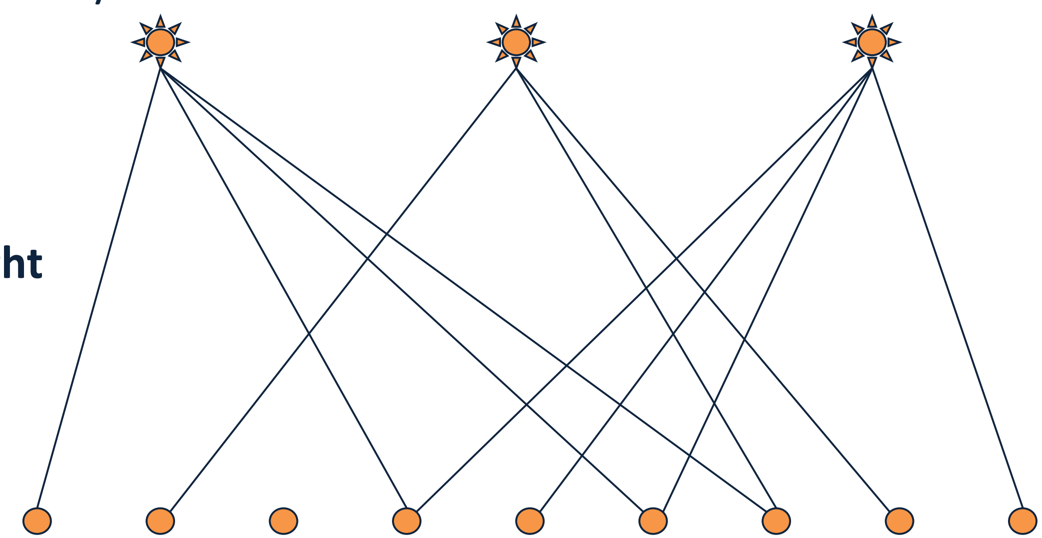
- Learning optimizes over *local* metrics
  - Does not guarantee good *global* performance
  - We don't care about the performance of the *local* classifiers
- Poor decomposition $\Rightarrow$ poor performance
  - Difficult local problems
  - Irrelevant local problems



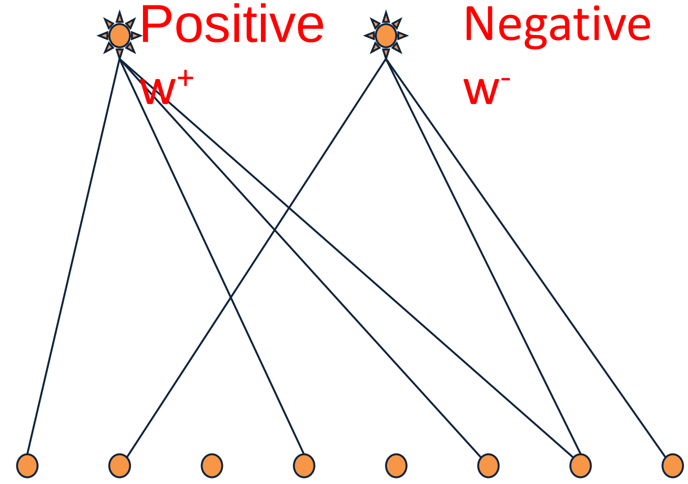
- Especially true for Error Correcting Output Codes
  - Another (class of) decomposition
  - Difficulty: how to make sure that the resulting problems are separable.
- Efficiency: e.g., All vs. All vs. One vs. All
- Former has advantage when working with the dual space.
- Not clear how to generalize multi-class to problems with a very large # of output variables.

1 Vs All: Learning Architecture

- k label nodes; n input features, nk weights.
- Evaluation: Winner Take All
- Training: Each set of n weights, corresponding to the i-th label, is trained
  - Independently, given its performance on example x, and
  - Independently of the performance of label j on x.
- Hence: Local learning; only the final decision is global, (Winner Takes All (WTA)).
- However, this architecture allows multiple learning algorithms; e.g., see the implementation in the SNoW/LbJava Multi-class Classifier



Recall: Winnow's Extensions

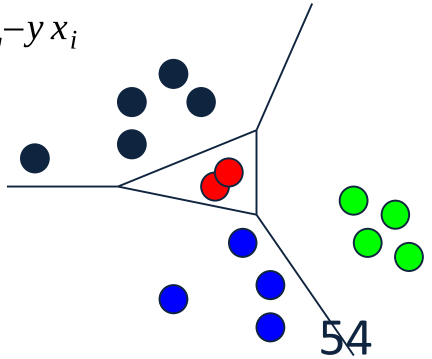


- Winnow learns monotone Boolean functions
- We extended to general Boolean functions via
- "Balanced Winnow"
  - 2 weights per variable;
  - Decision: using the "effective weight",
    the difference between w^+ and w^-
  - This is equivalent to the Winner take all decision
  - Learning: In principle, it is possible to use the 1-vs-all rule and update each set of n weights separately, but we suggested the "balanced" Update rule that takes into account how both sets of n weights predict on example x

$$If \quad [(\mathbf{w}^+ - \mathbf{w}^-) \bullet \mathbf{x} \geq \theta] \neq y, \quad w_i^+ \leftarrow w_i^+ r^{y x_i}, \quad w_i^- \leftarrow w_i^- r^{-y x_i}$$

Can this be generalized to the case of k labels, k >2?

We need a "global" learning approach



Extending Balanced

- In a 1-vs-all training you have a target node that represents positive examples and target node that represents negative examples.
- Typically, we train each node separately (mine/not-mine example).
- Rather, given an example we could say: this is more a + example than a – example.

$$If \quad [(\mathbf{w}^{+} - \mathbf{w}^{-}) \bullet \mathbf{x} \geq \theta] \neq y, \quad w_i^{+} \leftarrow w_i^{+} r^{y x_i}, \quad w_i^{-} \leftarrow w_i^{-} r^{-y x_i}$$

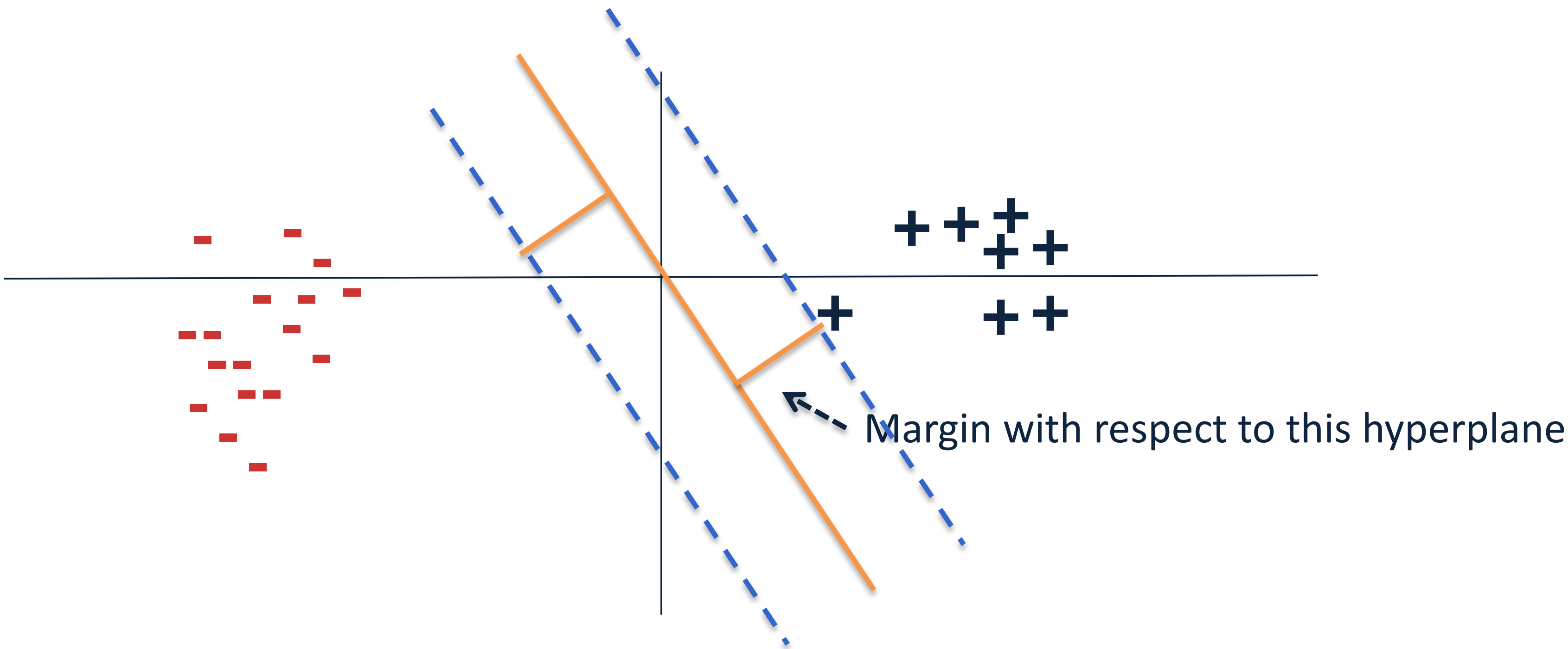
- We compared the activation of the different target nodes (classifiers) on a given example. (This example is more class + than class -)
- Can this be generalized to the case of k labels, k >2?

Where are we?

- Introduction
- Combining binary classifiers
  - One-vs-all
  - All-vs-all
  - Error correcting codes
- Training a single (global) classifier
  - Multiclass SVM
  - Constraint classification

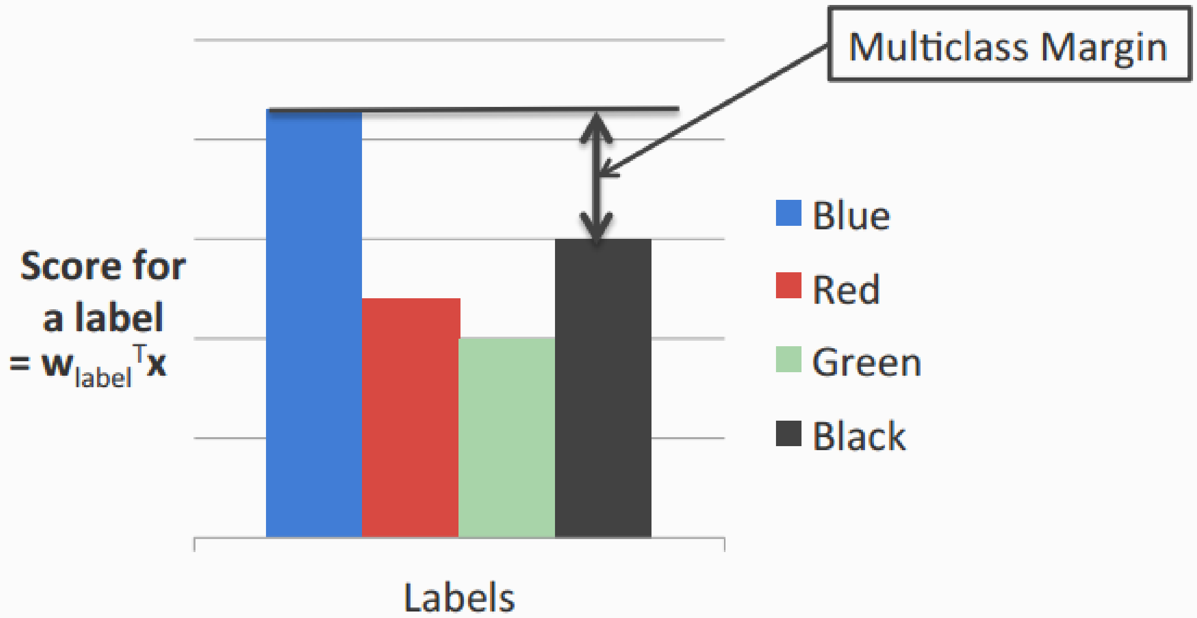
Recall: Margin for binary classifiers

- The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Multiclass Margin

Defined as the score difference between the highest scoring label and the second one



Multiclass SVM (Intuition)

- Recall: Binary SVM
  - Maximize margin
  - Equivalently,

    Minimize norm of weight vector, while keeping the closest points to the hyperplane with a score § 1

- Multiclass SVM
  - Each label has a different weight vector (like one-vs-all)
  - Maximize multiclass margin
  - Equivalently,

    Minimize total norm of the weight vectors while making sure that the true label scores at least 1 more than the second best one.

Multiclass SVM in the separable case

Recall hard binary SVM

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{s.t.} \forall i, \quad & y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{aligned}$$

Size of the weights.
Effectively, regularizer

$$\begin{aligned} \min_{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_K} \quad & \frac{1}{2}\sum_k \mathbf{w}_k^T \mathbf{w}_k \\ \text{s.t.} \quad & \mathbf{w}_{\mathbf{y}_i}^T \mathbf{x} - \mathbf{w}_k^T \mathbf{x} \geq 1 \qquad \forall (\mathbf{x}_i, \mathbf{y}_i) \in D, \\ & k \in \{1, 2, \cdots, K\}, k \neq \mathbf{y}_i, \end{aligned}$$

The score for the true label is higher than the score for ***any*** other label by 1

Multiclass SVM: General case

Size of the weights. Effectively, regularizer

Total slack. Effectively, don't allow too many examples to violate the margin constraint

$$\min_{\mathbf{w}_1,\mathbf{w}_2,\cdots,\mathbf{w}_K} \quad \frac{1}{2}\sum_k \mathbf{w}_k^T\mathbf{w}_k$$

$$\text{s.t.} \quad \mathbf{w}_{\mathbf{y}_i}^T\mathbf{x} - \mathbf{w}_k^T\mathbf{x} \geq 1 \qquad \forall(\mathbf{x}_i,\mathbf{y}_i)\in D,$$

$$k \in \{1,2,\cdots,K\}, k \neq \mathbf{y}_i,$$

The score for the true label is higher than the score for ***any*** other label by 1

Slack variables. Not all examples need to satisfy the margin constraint.

Slack variables can only be positive

Multiclass SVM: General case

Size of the weights. Effectively, regularizer

Total slack. Effectively, don't allow too many examples to violate the margin constraint

$$\min_{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_K, \xi} \quad \frac{1}{2}\sum_k \mathbf{w}_k^T \mathbf{w}_k + C \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in D} \xi_i$$

$$\text{s.t.} \quad \mathbf{w}_{\mathbf{y}_i}^T \mathbf{x} - \mathbf{w}_k^T \mathbf{x} \geq 1 - \xi_i, \quad \forall (\mathbf{x}_i, \mathbf{y}_i) \in D, \; k \in \{1, 2, \cdots, K\}, k \neq \mathbf{y}_i,$$

$$\xi_i \geq 0, \quad \forall i.$$

The score for the true label is higher than the score for ***any*** other label by 1 - ξ_i

Slack variables. Not all examples need to satisfy the margin constraint.

Slack variables can only be positive

Multiclass SVM

- Generalizes binary SVM algorithm
  - If we have only two classes, this reduces to the binary (up to scale)
- Comes with similar generalization guarantees as the binary SVM
- Can be trained using different optimization methods
  - Stochastic sub-gradient descent can be generalized
    - Try as exercise

Multiclass SVM: Summary

- Training:
  - Optimize the "global" SVM objective
- Prediction:
  - Winner takes all

    $\text{argmax}_i\ \mathbf{w}_i^T \mathbf{x}$
- With K labels and inputs in R^n, we have nK weights in all
  - Same as one-vs-all
- Why does it work?
  - Why is this the "right" definition of multiclass margin?
- A theoretical justification, along with extensions to other algorithms beyond SVM is given by "Constraint Classification"
  - Applies also to multi-label problems, ranking problems, etc.
  - [Dav Zimak; with D. Roth and S. Har-Peled]

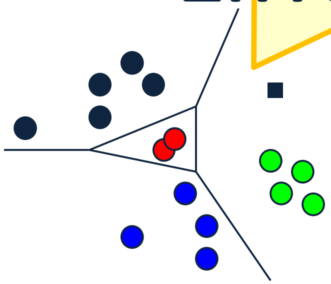
Constraint Classification

- The examples we give the learner are pairs (x,y), $y \in \{1,\ldots k\}$
- The "black box learner" (1 vs. all) we described might be thought of as a function of x only but, actually, we made use of the labels y
- How is y being used?
  - y decides what to do with the example x; that is, which of the k classifiers should take the example as a positive example (making it a negative to all the others).
- How do we predict?
  - Let: $f_y(x) = w_y^T x$
  - Then, we predict using: $y^* = \text{argmax}_{y=1,\ldots k} f_y(x)$
- Equivalently, we can say that we predict as follows:
  - Predict y iff
  - $\forall y' \in \{1,\ldots k\}, y' \neq y \quad (w_y^T - w_{y'}^T)\, x \geq 0 \quad (**)$
- So far, we did not say how we learn the k weight vectors w_y $(y = 1,\ldots k)$
  - Can we train in a way that better fits the way we predict?
  - What does it mean?

Is it better in any well defined way?

We showed: if pairs of labels are separable (a reasonable assumption) than in some higher dimensional space, the problem is linearly separable.

Linear Separability for Multiclass



- We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into

  $w = (w_1, w_2, \ldots w_k)$ 2

Notice: This is just a representational trick. We did not say how to learn the weight vectors.

- Key Construction: (Kesler Construction; Zimak's Constraint Classification)
  - We will represent each example (x,y), as an nk-dimensional vector, x_y, with x embedded in the y-th part of it (y=1,2,…k) and the other coordinates are 0.
- **E.g.,** $\mathbf{x}_y = (\mathbf{0}, x, \mathbf{0}, \mathbf{0}) \in \mathbf{R}^{kn}$ (here k=4, y=2)
- Now we can understand the n-dimensional decision rule:
- Predict y iff ∀ y' 2 {1,…k}, y': =y $(w_y^T - w_{y'}^T)$ ¢x ¸ 0 (**)
- Equivalently, in the nk-dimensional space.
- Predict y iff ∀ y' 2 {1,…k}, y'≠y $w^T(x_y - x_{y'}) \equiv w^T x_{yy'} \geq 0$

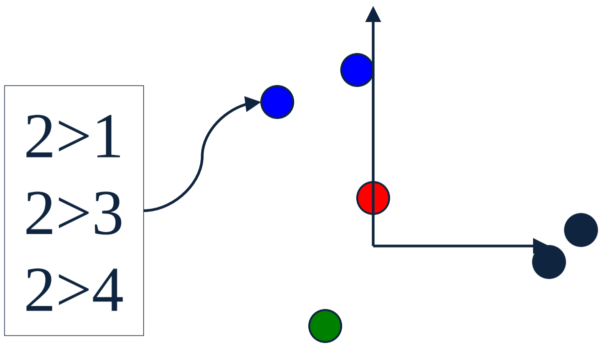
- Conclusion: The set $(x_{yy'}, +) \equiv (x_y - x_{y'}, +)$ is linearly separable from the set $(-x_{yy'}, -)$ using the linear separator $w \in R^{kn}$,
- We solved the voronoi diagram challenge.

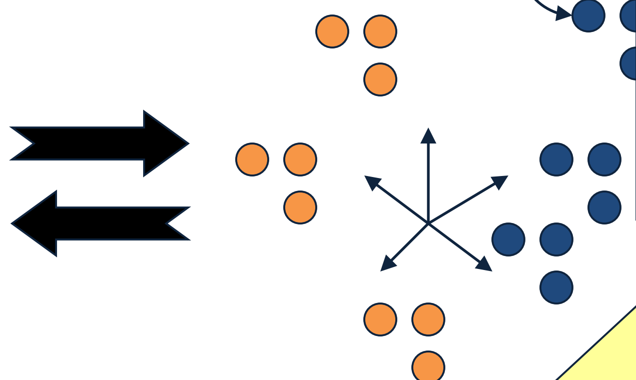
Constraint Classification

- Training:
  - [We first explain via Kesler's construction; then show we don't need it]
  - Given a data set $\{(x,y)\}$, (m examples) with $x \in R^n$, $y \in \{1,2,\ldots k\}$ create a binary classification task (in R^{kn}):
    $(x_y - x_{y'}, +), (x_{y'} - x_y, -)$, for all $y' \neq y$ ($2m(k-1)$ examples)
    Here $x_y \in R^{kn}$
  - Use your favorite linear learning algorithm to train a binary classifier.
- Prediction:
  - Given an nk dimensional weight vector w and a new example x, predict: $\arg\max_y w^T x_y$

Details: Kesler Construction & Multi-Class Separability

- Transform Examples





If (x,i) was a given n-dimensional example (that is, x has is labeled i, then x_{ij}, $\forall$ j=1,…k, j≠ i, are positive examples in the nk-dimensional space. $-x_{ij}$ are negative examples.

i>j	$f_i(x) - f_j(x)$	> 0
	$w_i \cdot x - w_j \cdot x$	> 0
	$\mathbf{W} \cdot \mathbf{X}_i - \mathbf{W} \cdot \mathbf{X}_j$	> 0
	$\mathbf{W} \cdot (\mathbf{X}_i - \mathbf{X}_j)$	> 0
	$\mathbf{W} \cdot \mathbf{X}_{ij}$	> 0

$$\mathbf{X}_i = (\mathbf{0},x,\mathbf{0},\mathbf{0}) \in \mathbf{R}^{kd}$$

$$\mathbf{X}_j = (\mathbf{0},\mathbf{0},\mathbf{0},x) \in \mathbf{R}^{kd}$$

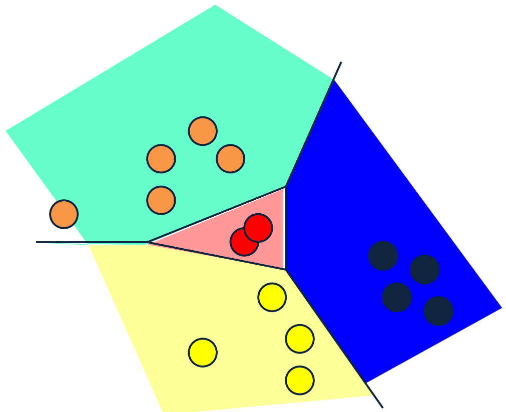
$$\mathbf{X}_{ij} = \mathbf{X}_i - \mathbf{X}_j = (\mathbf{0},x,\mathbf{0},-x)$$

$$\mathbf{W} = (w_1,w_2,w_3,w_4) \in \mathbf{R}^{kd}$$

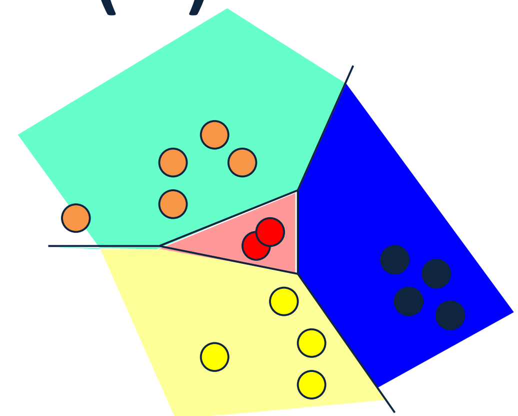
Kesler's Construction (1)

- $y = \text{argmax}_{i=(r,b,g,y)}\ w_i.x$
  - w_i , $x \in \mathbf{R}^n$
- Find $w_r, w_b, w_g, w_y \in \mathbf{R}^n$ such that
  - $w_r.x > w_b.x$
  - $w_r.x > w_g.x$
  - $w_r.x > w_y.x$

$\mathbf{H} = \mathbf{R}^{kn}$

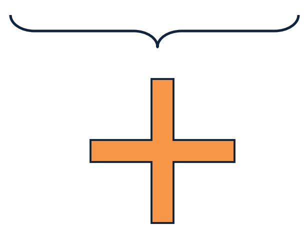


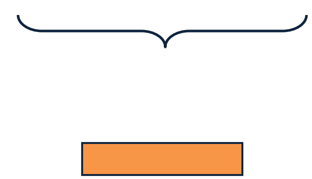
Kesler's Construction (2)



- Let $\mathbf{w} = (w_r, w_b, w_g, w_y) \in \mathbf{R}^{kn}$
- Let $\mathbf{0}^n$, be the n-dim zero vector

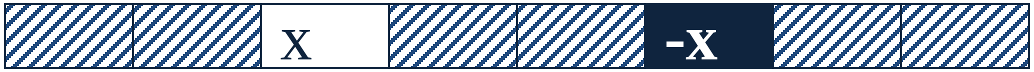
- $w_r.x > w_b.x \Leftrightarrow \mathbf{w}.(x,-x,\mathbf{0}^n,\mathbf{0}^n) > 0 \Leftrightarrow \mathbf{w}.(-x,x,\mathbf{0}^n,\mathbf{0}^n) < 0$
- $w_r.x > w_g.x \Leftrightarrow \mathbf{w}.(x,\mathbf{0}^n,-x,\mathbf{0}^n) > 0 \Leftrightarrow \mathbf{w}.(-x,\mathbf{0}^n,x,\mathbf{0}^n) < 0$
- $w_r.x > w_y.x \Leftrightarrow \mathbf{w}.(x,\mathbf{0}^n,\mathbf{0}^n,-x) > 0 \Leftrightarrow \mathbf{w}.(-x,\mathbf{0}^n,\mathbf{0}^n,x) < 0$





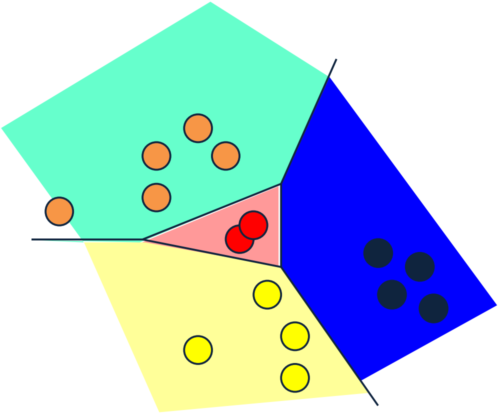
Kesler's Construction (3)

- Let
  - $\mathbf{w} = (w_1, \ldots, w_k) \in \mathbf{R}^n \times \ldots \times \mathbf{R}^n = \mathbf{R}^{kn}$
  - $\mathbf{x}_{ij} = (\mathbf{0}^{(i-1)n}, x, \mathbf{0}^{(k-i)n}) - (\mathbf{0}^{(j-1)n}, -x, \mathbf{0}^{(k-j)n}) \in \mathbf{R}^{kn}$

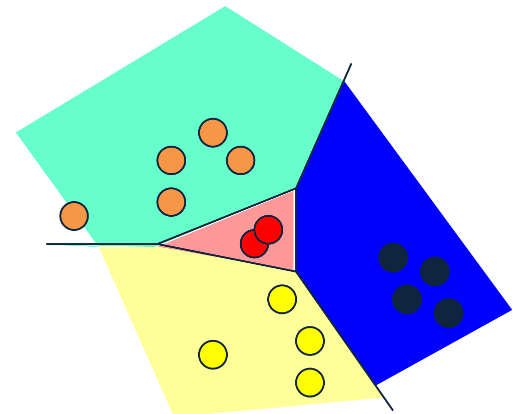


- Given $(x, y) \in \mathbf{R}^n \times \{1,\ldots,k\}$
  - For all $j \neq y$ (all other labels)
    - Add to $\mathbf{P}^+(x,y)$, $(\mathbf{x}_{yj}, 1)$
    - Add to $\mathbf{P}^-(x,y)$, $(-\mathbf{x}_{yj}, -1)$

- $\mathbf{P}^+(x,y)$ has k-1 positive examples ($\in \mathbf{R}^{kn}$)
- $\mathbf{P}^-(x,y)$ has k-1 negative examples ($\in \mathbf{R}^{kn}$)



Learning via Kesler's Construction



- Given $(x_1, y_1), \dots, (x_N, y_N) \in \mathbf{R}^n \times \{1,\dots,k\}$
- Create
  - $\mathbf{P}^+ = \cup\, \mathbf{P}^+(x_i, y_i)$
  - $\mathbf{P}^- = \cup\, \mathbf{P}^-(x_i, y_i)$
- Find $\mathbf{w} = (w_1, \dots, w_k) \in \mathbf{R}^{kn}$, such that
  - $\mathbf{w.x}$ separates $\mathbf{P}^+$ from $\mathbf{P}^-$
- One can use any algorithm in this space: Perceptron, Winnow, SVM, etc.
- To understand how to update the weight vector in the n-dimensional space, we note that
- $w^T\ x_{yy'} \geq 0$ (in the nk-dimensional space)
- is equivalent to:
- $(w_y^T - w_{y'}^T)\ x \geq 0$ (in the n-dimensional space)

Perceptron in Kesler Construction

- A perceptron update rule applied in the nk-dimensional space due to a mistake in $w^T x_{ij} \geq 0$
- Or, equivalently to $(w_i^T - w_j^T)x \geq 0$ (in the n-dimensional space)
- Implies the following update:

- Given example (x,i) (example $x \in R^n$, labeled i)
  - $\forall$ (i,j), i,j = 1,…k, $i \neq j$ (***)
  - If $(w_i^T - w_j^T) x < 0$ (mistaken prediction; equivalent to $w^T x_{ij} < 0$)
  - $w_i \leftarrow w_i + x$ (promotion) and $w_j \leftarrow w_j - x$ (demotion)

- Note that this is a generalization of balanced Winnow rule.

- Note that we promote w_i and demote k-1 weight vectors w_j

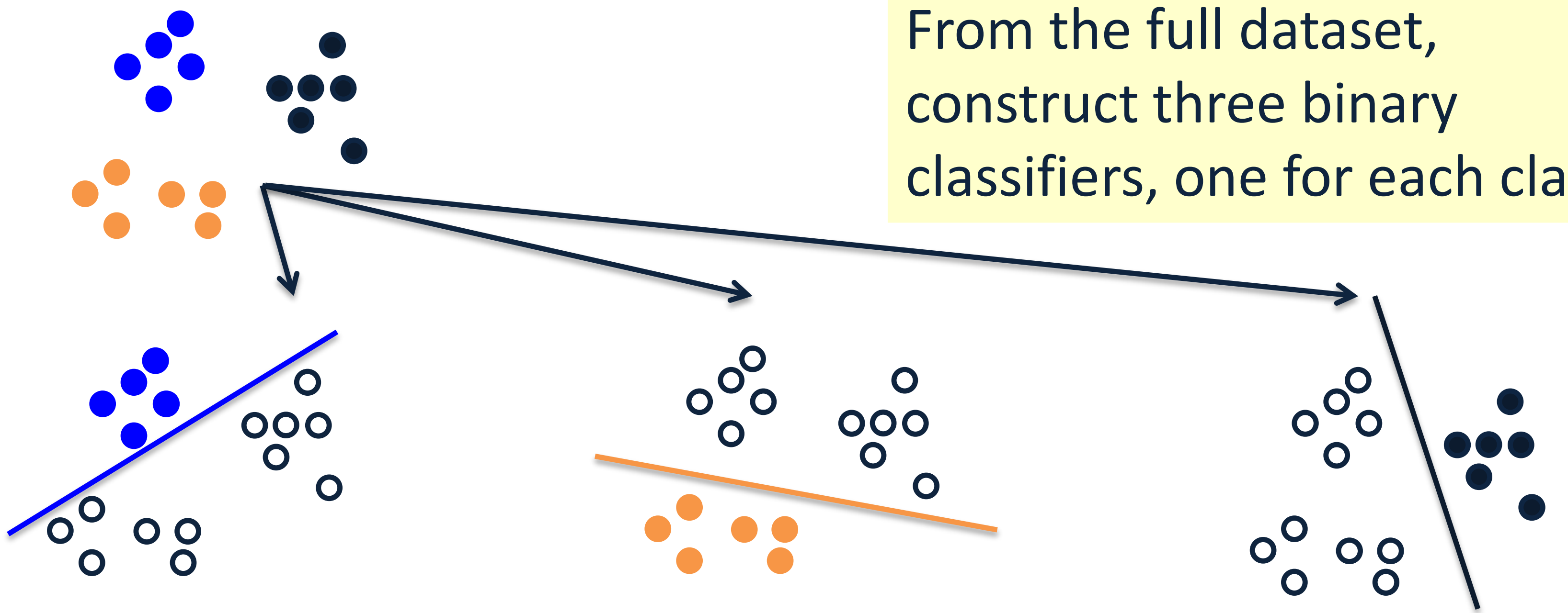
Conservative update

- The general scheme suggests:
- Given example (x,i) (example $x \in R^n$, labeled i)
  - $\forall$ (i,j), i,j = 1,…k, i : = j (***)
  - If $(w_i^T - w_j^T)\, x < 0$ (mistaken prediction; equivalent to $w^T x_{ij} < 0$)
  - $w_i \leftarrow w_i + x$ (promotion) and $w_j \leftarrow w_j - x$ (demotion)
- Promote w_i and demote $k-1$ weight vectors w_j
- A conservative update: (SNoW and LBJava's implementation):
  - In case of a mistake: only the weights corresponding to the target node i and that closest node j are updated.
  - Let: $j^* = \text{argmax}_{j=1,\dots k}\, w_j^T x$ (highest activation among competing labels)
  - If $(w_i^T - w_{j^*}^T)\, x < 0$ (mistaken prediction)
  - $w_i \leftarrow w_i + x$ (promotion) and $w_{j^*} \leftarrow w_{j^*} - x$ (demotion)
  - Other weight vectors are not being updated.

Multiclass Classification Summary 1:

Multiclass Classification

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



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

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

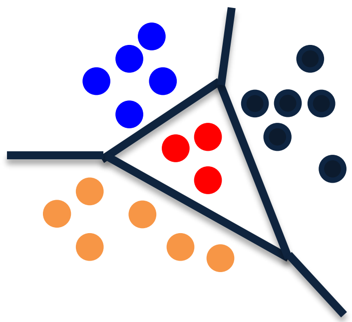
$\mathbf{w}_{black}^T\mathbf{x}$ > 0 for **black inputs**

Notation: Score for blue label

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

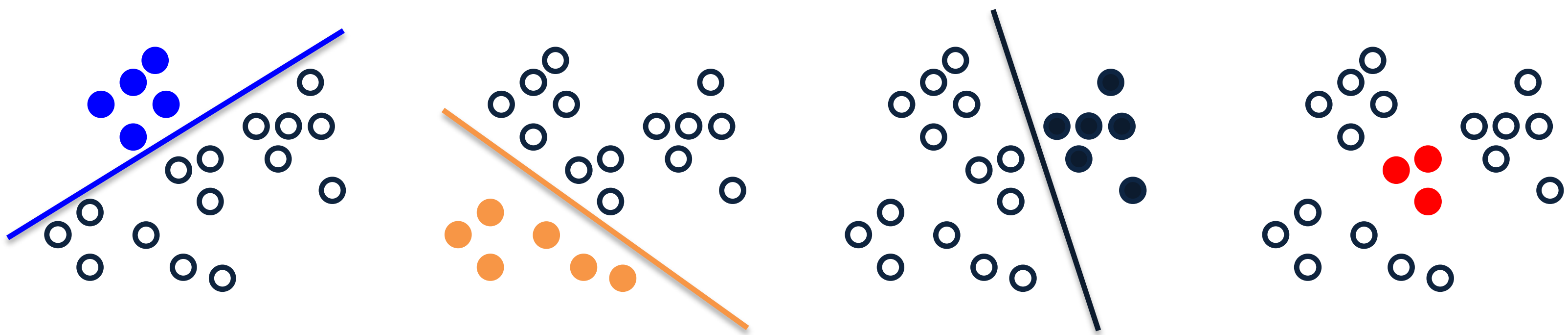
Multiclass Classification Summary 2:

One-vs-all may not always work



Red points are not separable with a single binary classifier

The decomposition is not expressive enough!



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

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

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

???

Summary 3:

Local Learning: One-vs-all classification

- Easy to learn
  - Use any binary classifier learning algorithm
- Potential Problems
  - Calibration issues
    - We are comparing scores produced by K classifiers trained independently. No reason for the scores to be in the same numerical range!
  - Train vs. Train
    - Does not account for how the final predictor will be used
    - Does not optimize any **global** measure of correctness
  - Yet, works fairly well
    - In most cases, especially in high dimensional problems (everything is already linearly separable).

Summary 4:
Global Multiclass Approach [Constraint Classification, Har-Peled et. al '02]

- Create K classifiers $\mathbf{w}_1$, $\mathbf{w}_2$, …, $\mathbf{w}_{K.}$;
- Predict with WTA: $\mathrm{argmax}_i\ \mathbf{w}_i^T\mathbf{x}$

- **But**, train differently:
  - For examples with label i, we want

    $\boldsymbol{w}_i^T\boldsymbol{x} > \boldsymbol{w}_j^T\boldsymbol{x}$ *for all j*

- **Training:** For each training example $(\boldsymbol{x_i}, \boldsymbol{y_i})$:

$$\hat{y} \leftarrow arg \max_{\boldsymbol{j}} \boldsymbol{w}_{\boldsymbol{j}}^{T} \phi(\boldsymbol{x}_i, y_i)$$

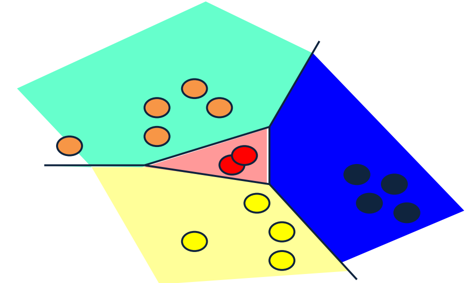
if $\hat{y} \neq y_i$

$$\boldsymbol{w}_{y_i} \leftarrow \boldsymbol{w}_{y_i} + \eta \boldsymbol{x}_i \qquad \text{(promote)}$$

$$\boldsymbol{w}_{\hat{y}} \leftarrow \boldsymbol{w}_{\hat{y}} - \eta \boldsymbol{x}_i \qquad \text{(demote)}$$

η: learning rate

Significance



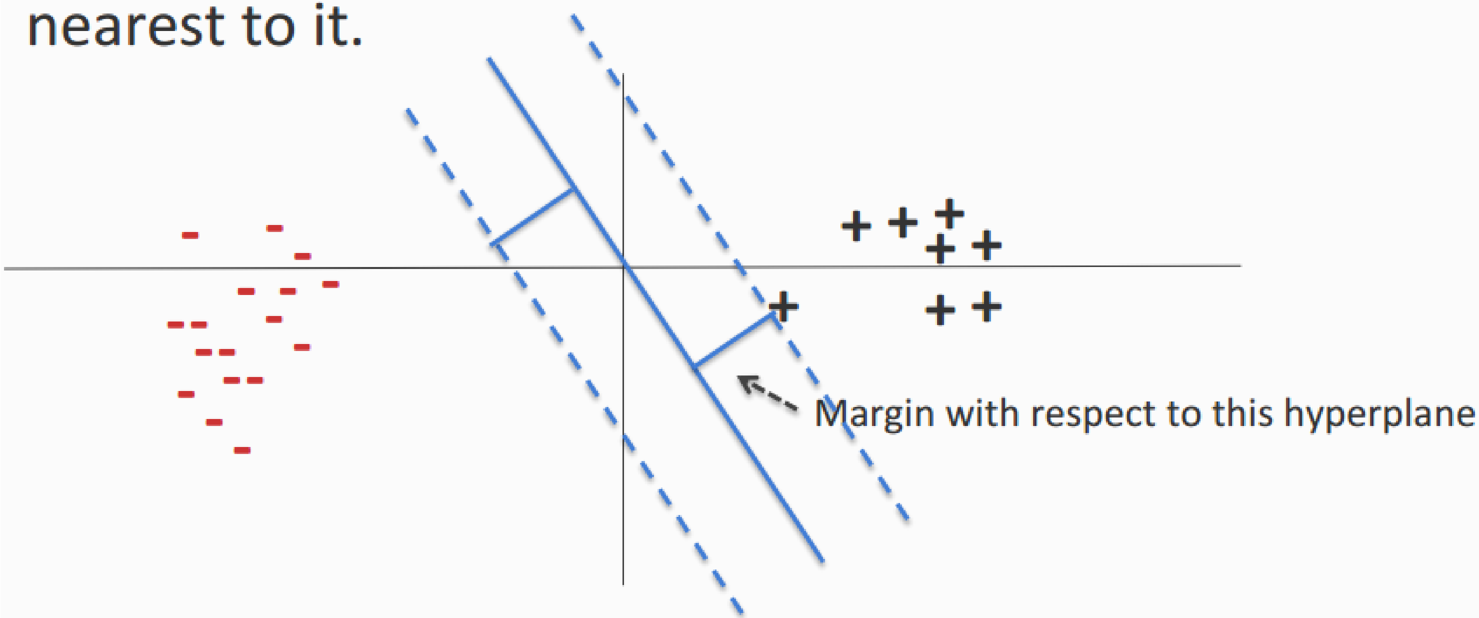
- The hypothesis learned above is more expressive than when the OvA assumption is used.
- Any linear learning algorithm can be used, and algorithmic-specific properties are maintained (e.g., attribute efficiency if using winnow.)
- E.g., the multiclass support vector machine can be implemented by learning a hyperplane to separate P(S) with maximal margin.

- As a byproduct of the linear separability observation, we get a natural notion of a margin in the multi-class case, inherited from the binary separability in the nk-dimensional space.
  - Given example $x_{ij} \in R^{nk}$, margin$(x_{ij},w) = \min_{ij} w^T x_{ij}$
  - Consequently, given $x \in R^n$, labeled i

margin$(x,w) = \min_j (w_i^T - w_j^T)\, x$

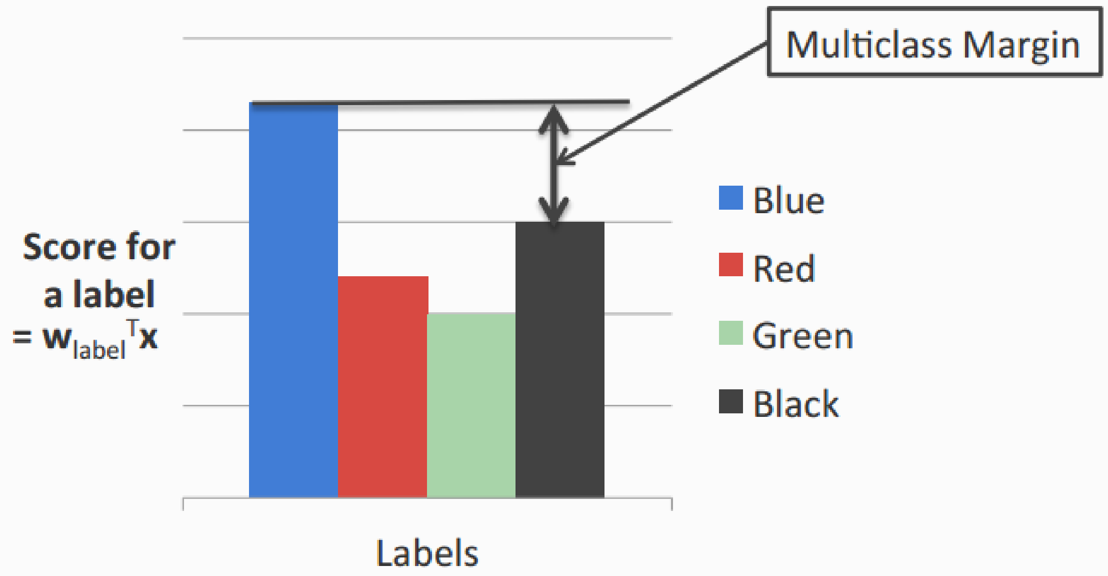
Margin

The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Multiclass Margin

Defined as the score difference between the highest scoring label and the second one



Constraint Classification

- The scheme presented can be generalized to provide a uniform view for multiple types of problems: multi-class, multi-label, category-ranking
- Reduces learning to a *single* binary learning task
- Captures theoretical properties of binary algorithm
- Experimentally verified
- Naturally extends Perceptron, SVM, etc...

- *It is called "constraint classification" since it does it all by representing labels as a set of constraints or preferences among output labels.*

Multi-category to Constraint Classification

- The unified formulation is clear from the following examples:
- Multiclass
  - (x, A) $\Rightarrow$ (x, (A>B, A>C, A>D))
- Multilabel
  - (x, (A, B)) $\Rightarrow$ (x, ((A>C, A>D, B>C, B>D))
- Label Ranking
  - (x, (5>4>3>2>1)) $\Rightarrow$ (x, ((5>4, 4>3, 3>2, 2>1))

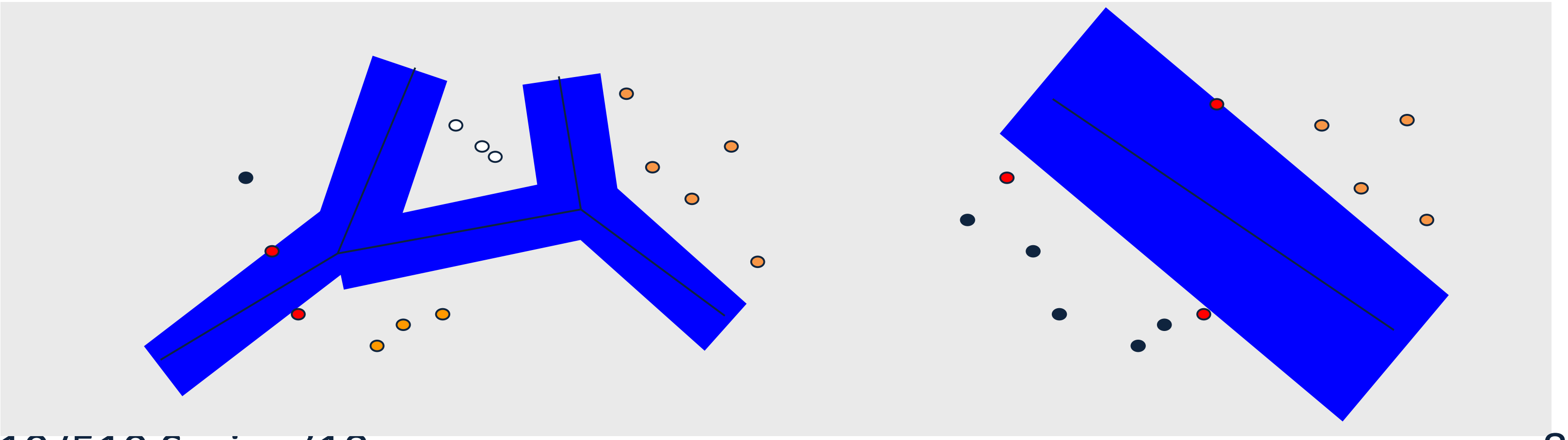
Just like in the multiclass we learn one $w_i \in R^n$ for each label, the same is done for multi-label and ranking. The weight vectors are updated according with the requirements from $y \in S_k$

(Consult the Perceptron in Kesler construction slide)

- In all cases, we have examples (x,y) with $y \in \mathbf{S_k}$
- Where $\mathbf{S_k}$: partial order over class labels {1,...,k}
  - defines "*preference*" relation (>) for class labeling
- Consequently, the Constraint Classifier is: h: $\mathbf{X} \longrightarrow \mathbf{S_k}$
  - h(x) is a partial order
  - h(x) is *consistent* with y if $(i<j) \in y \rightarrow (i<j) \in h(x)$

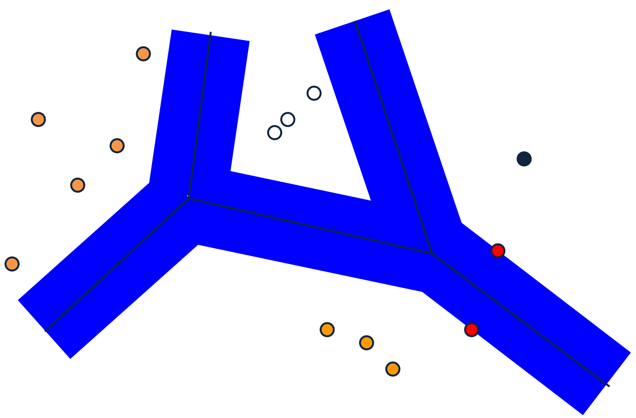
Properties of Construction (Zimak et. al 2002, 2003)

- Can learn *any* argmax $v_i.x$ function (even when i isn't linearly separable from the union of the others)
- Can use *any* algorithm to find linear separation
  - Perceptron Algorithm
    - *ultraconservative online algorithm* [Crammer, Singer 2001]
  - Winnow Algorithm
    - *multiclass winnow* [Masterharm 2000]
- Defines a *multiclass margin*
  - by binary margin in R^{kd}
  - multiclass SVM [Crammer, Singer 2001]



Margin Generalization Bounds

- Linear Hypothesis space:
  - $h(x) = \text{argsort } v_i.x$
    - $v_i, x \in \mathbf{R}^d$
    - argsort returns *permutation* of {1,…,k}
- CC margin-based bound
  - $\gamma = \min_{(x,y)\in \mathbf{S}} \min_{(i<j)\in \mathbf{y}} v_i.x - v_j.x$



$$err_D(h) \le \Theta\left(\frac{C}{m}\left(\frac{R^2}{\gamma^2} - \ln(\delta)\right)\right)$$

m - number of examples

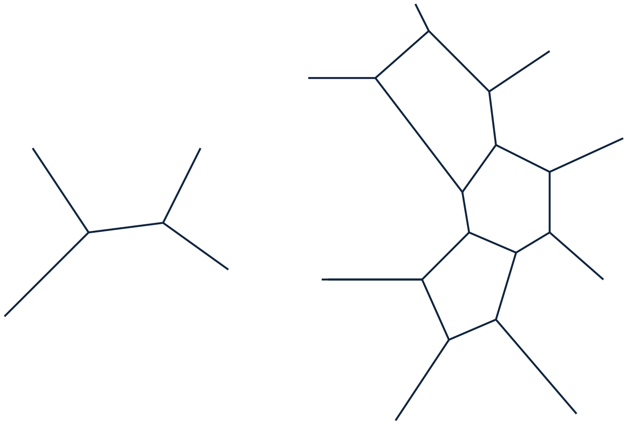
R - $\max_x ||x||$

δ - confidence

C - average # constraints

VC-style Generalization Bounds

- Linear Hypothesis space:
  - h(x) = argsort $v_i.x$
    - v_i, $x \in \mathbf{R}^d$
    - argsort returns *permutation* of {1,...,k}



- CC VC-based bound

$$err_D(h) \le err(S,h) + \theta\left(\sqrt{\frac{kd\log(mk/d) - \ln\delta}{m}}\right)$$

m - number of examples

d - dimension of input space

delta - confidence

k - number of classes

Performance: even though this is the right thing to do, and differences can be observed in low dimensional cases, in high dimensional cases, the impact is not always significant.

Beyond MultiClass Classification

- Ranking
  - category ranking (over classes)
  - ordinal regression (over examples)
- Multilabel
  - **x** is both red and blue
- Complex relationships
  - **x** is more red than blue, but not green
- Millions of classes
  - sequence labeling (e.g. POS tagging)
  - The same algorithms can be applied to these problems, namely, to Structured Prediction
  - This observation is the starting point for CS546.

(more) Multi-Categorical Output Tasks

- Sequential Prediction ($y \in \{1,...,K\}^+$)

  e.g. POS tagging ('(NVNNA)')

  "This is a sentence." $\Longrightarrow$ D V D N

  e.g. phrase identification

  Many labels: K^L for length L sentence

- Structured Output Prediction ($y \in C(\{1,...,K\}^+)$)

  e.g. parse tree, multi-level phrase identification

  e.g. sequential prediction

  Constrained by

  domain, problem, data, background knowledge, etc...