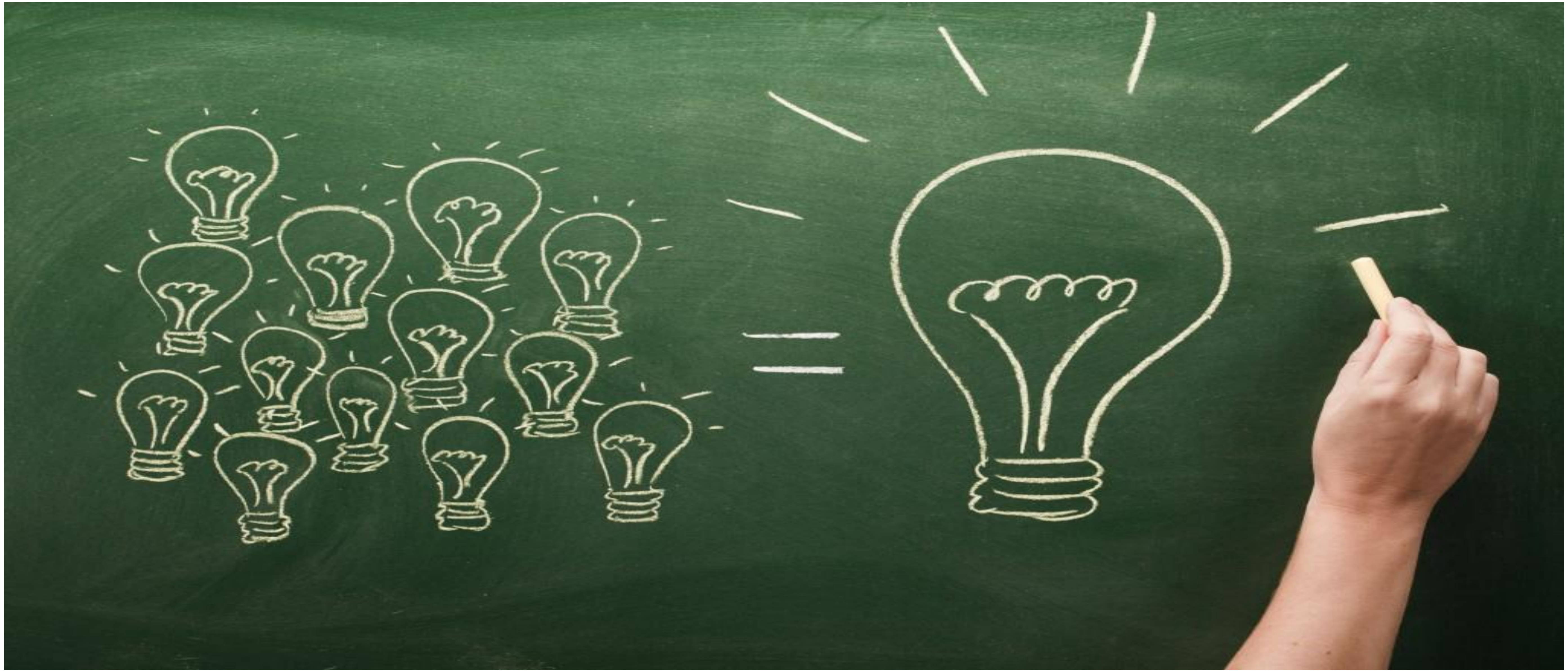
Bagging - Boosting

Dr. Cahit Karakuş

İçerik

- Bagging
- Random Forests
- Boosting

Power of the crowds



Torbalama

Verileri rastgele farklı şekillerde bölersek, karar ağaçları farklı sonuçlar, yüksek varyans verir.

Torbalama: Önyükleme toplama, düşük varyansla sonuçlanan bir yöntemdir.

Verilerin (veya birden çok örneğin) birden fazla gerçekleşmesine sahip olsaydık, tahminleri birden çok kez hesaplayabilir ve birden fazla zahmetli tahminin ortalamasını almanın daha az belirsiz sonuçlar ürettiği gerçeğinin ortalamasını alabilirdik.

Bagging

Say for each sample *b*, we calculate $f^b(x)$, then:

$$\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

How?

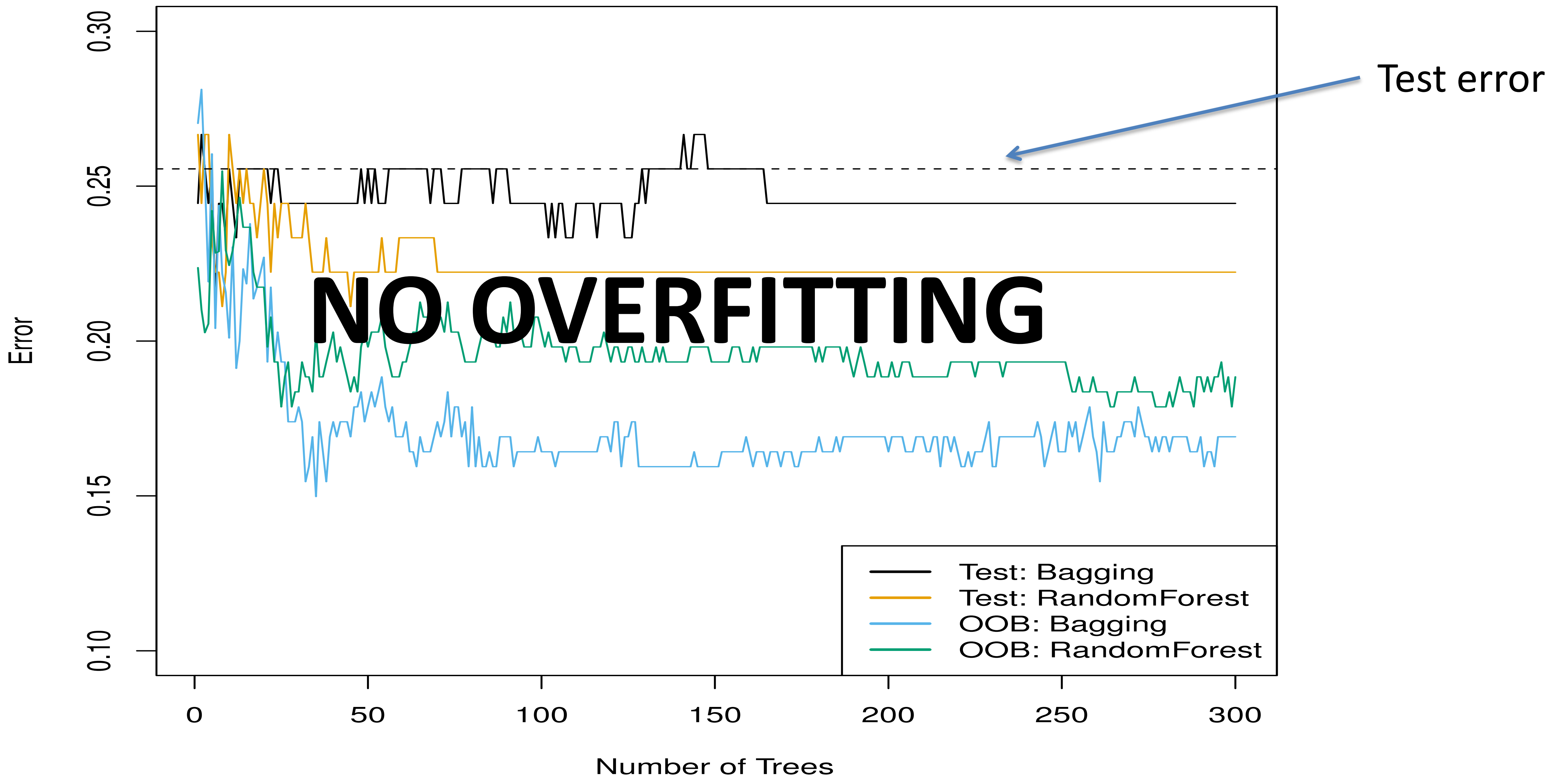
Bootstrap

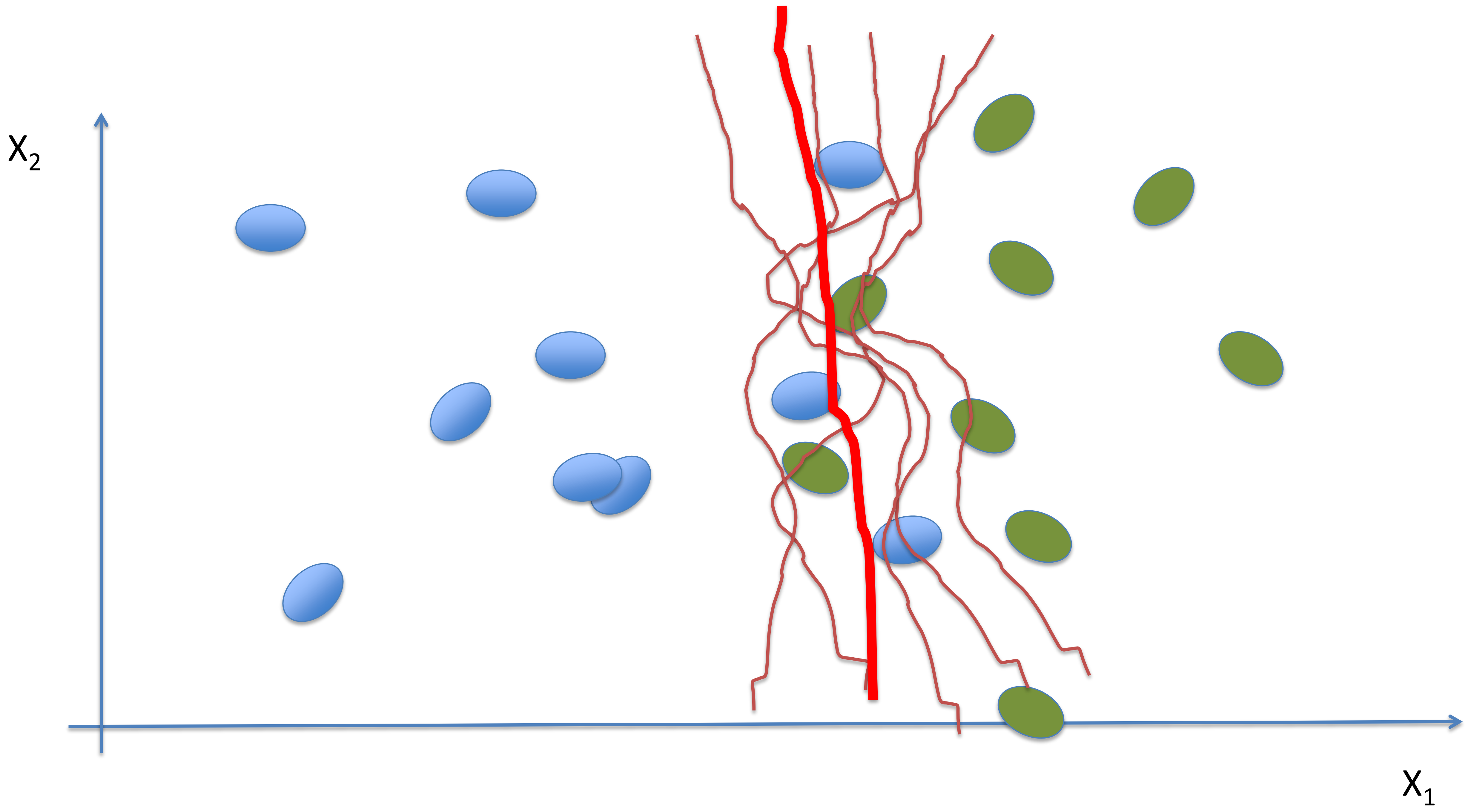
Construct B (hundreds) of trees (no pruning)

Learn a classifier for each bootstrap sample and average them

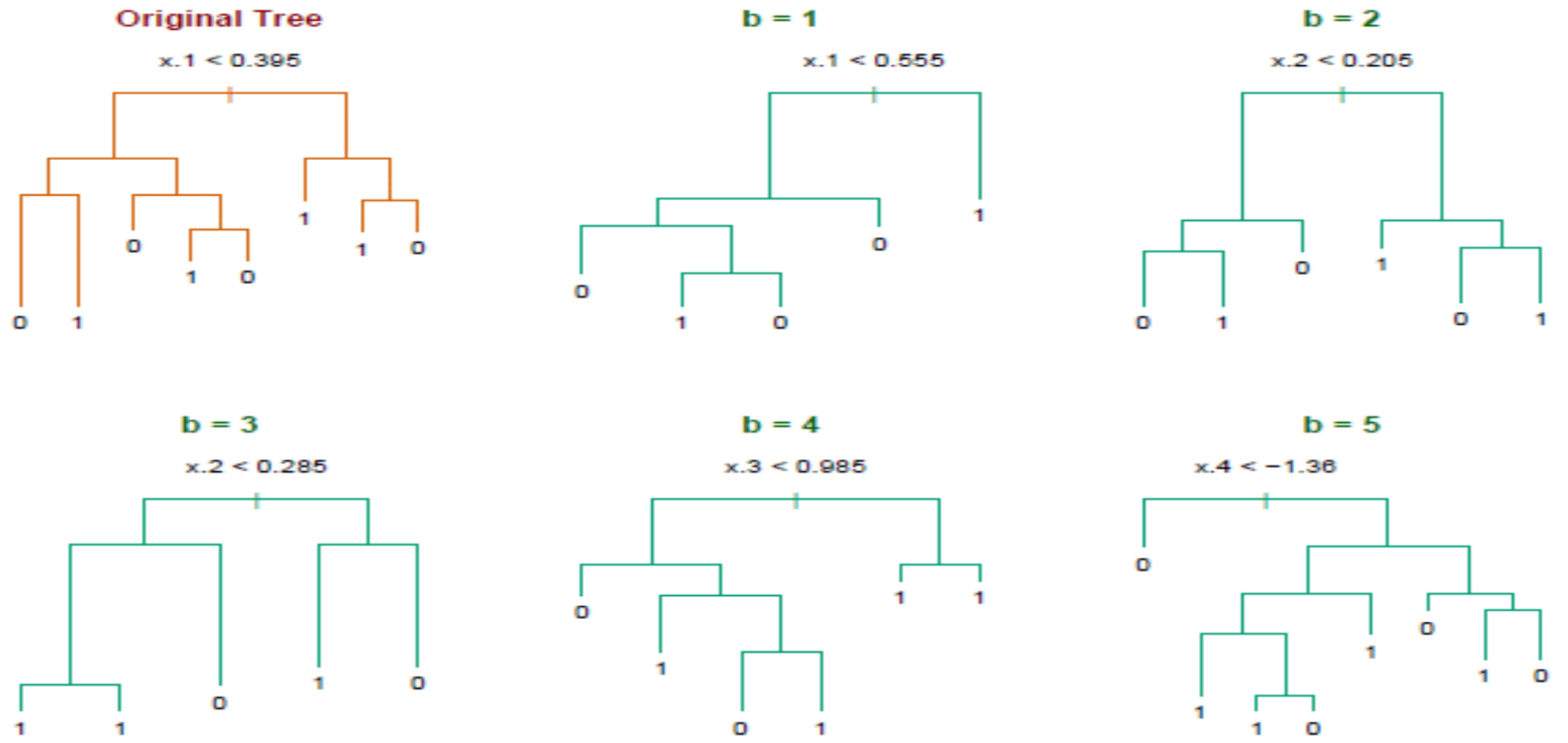
Very effective

Bagging for classification: Majority vote




X_2
X_1

Bagging decision trees



Hastie et al.,"The Elements of Statistical Learning: Data Mining, Inference, and Prediction", Springer (2009)

Out-of-Bag Error Estimation

- No cross validation?
- Remember, in bootstrapping we sample with replacement, and therefore **not all observations are used for each bootstrap sample**. On average 1/3 of them are not used!
- We call them out-of-bag samples (OOB)
- We can predict the response for the *i-th* observation using each of the trees in which that observation was OOB and do this for *n* observations
- Calculate overall OOB MSE or classification error

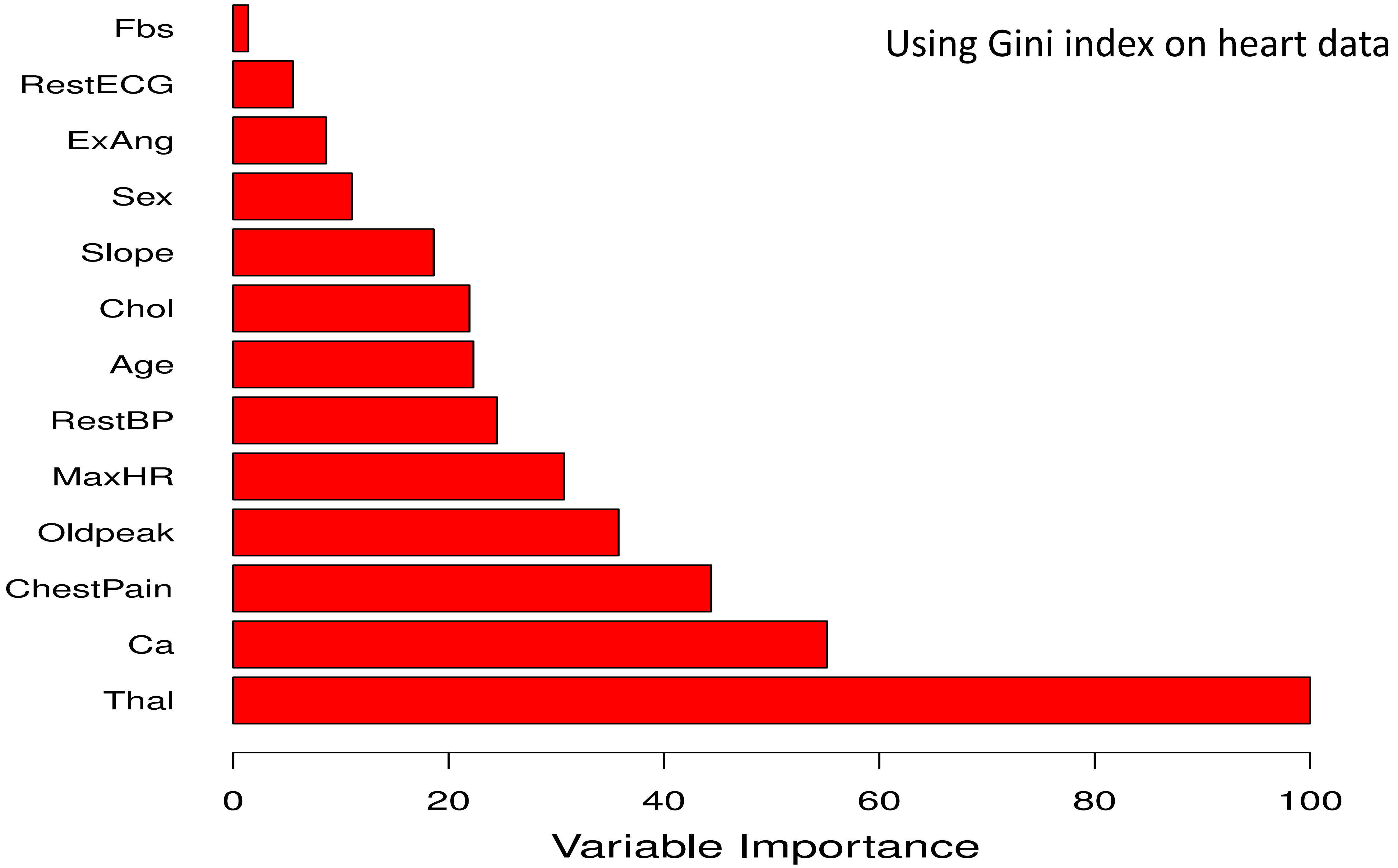
Bagging

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

Variable Importance Measures

- Bagging results in improved accuracy over prediction using a single tree
- Unfortunately, difficult to interpret the resulting model. Bagging improves prediction accuracy at the expense of interpretability.

Calculate the total amount that the RSS or Gini index is decreased due to splits over a given predictor, averaged over all B trees.

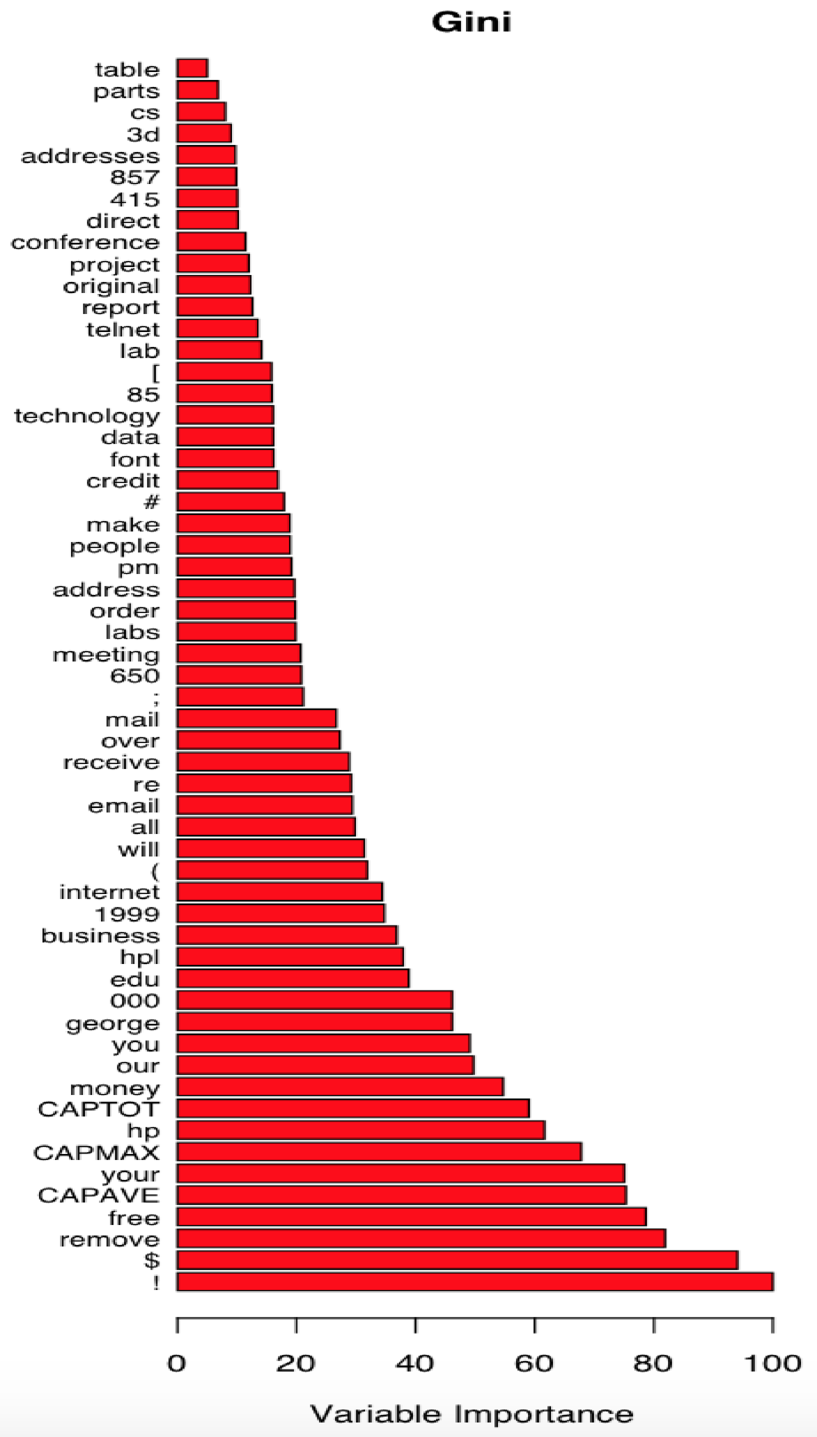

Using Gini index on heart data
Fbs
RestECG
ExAng
Sex
Slope
Chol
Age
RestBP
MaxHR
Oldpeak
ChestPain
Ca
Thal
0
20
40
60
80
100
Variable Importance

RF: Variable Importance Measures

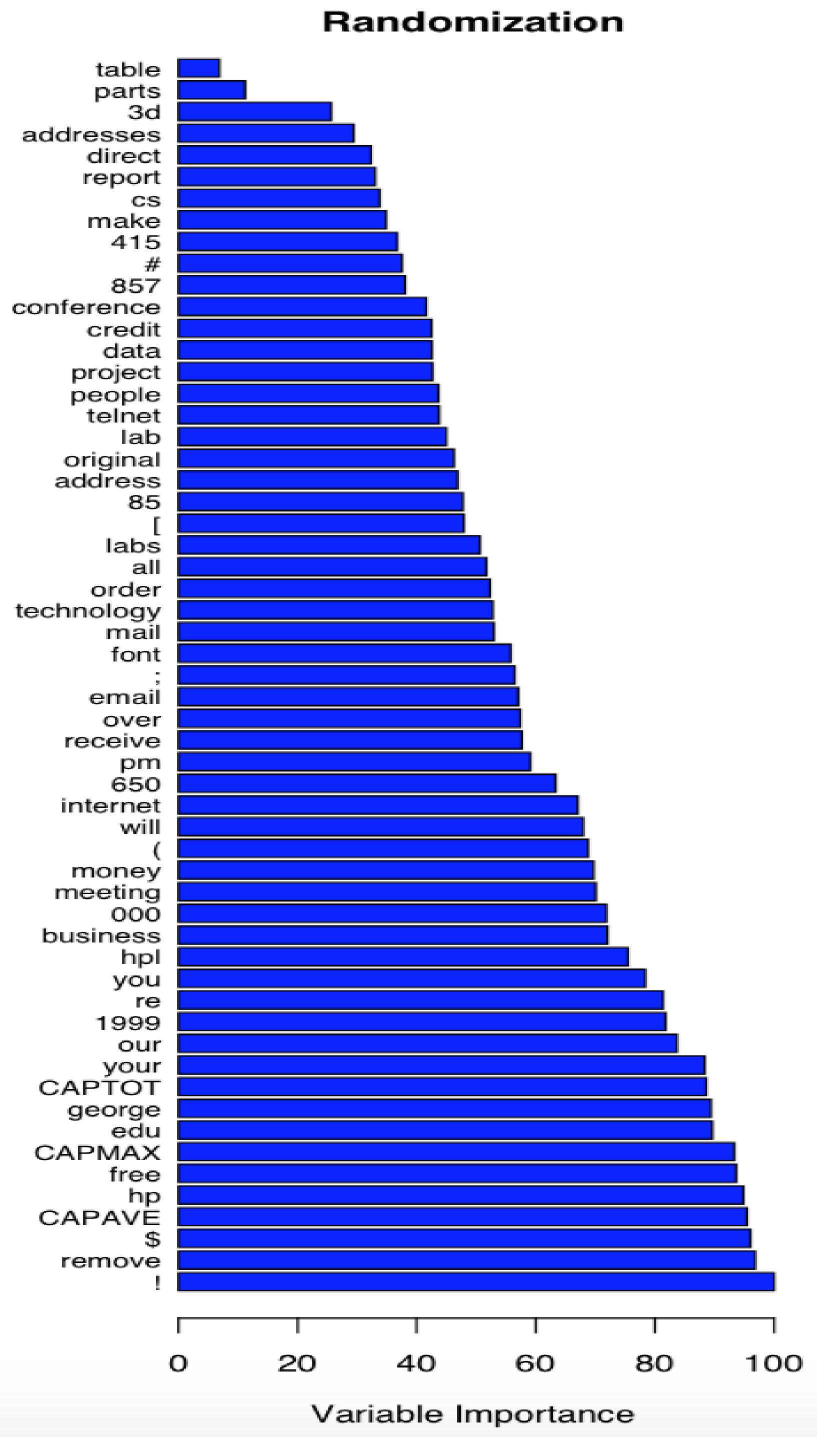
Record the prediction accuracy on the oob samples for each tree

Randomly permute the data for column *j* in the oob samples the record the accuracy again.

The decrease in accuracy as a result of this permuting is averaged over all trees, and is used as a measure of the importance of variable j in the random forest.



Gini
table
parts
cs
3d
addresses
857
415
direct
conference
project
original
report
telnet
lab
[
85
technology
data
font
credit
#
make
people
pm
address
order
labs
meeting
650
;
mail
over
receive
re
email
all
will
(
internet
1999
business
hpl
edu
000
george
you
our
money
CAPTOT
hp
CAPMAX
your
CAPAVE
free
remove
$
!
0
20
40
60
80
100
Variable Importance



Randomization
table
parts
3d
addresses
direct
report
cs
make
415
#
857
conference
credit
data
project
people
telnet
lab
original
address
85
[
labs
all
order
technology
mail
font
;
email
over
receive
pm
650
internet
will
(
money
meeting
000
business
hpl
you
re
1999
our
your
CAPTOT
george
edu
CAPMAX
free
hp
CAPAVE
$
remove
!
0
20
40
60
80
100
Variable Importance

Bagging - issues

Each tree is identically distributed (i.d.)

➔ the expectation of the average of *B* such trees is the same as the expectation of any one of them

➔the bias of bagged trees is the same as that of the individual trees

i.d. and not i.i.d

Bagging - issues

An average of B i.i.d. random variables, each with variance σ^2, has variance: σ^2/B

If i.d. (identical but not independent) and pair correlation ρ is present, then the variance is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

As B increases the second term disappears but the first term remains

Why does bagging generate correlated trees?

Bagging - issues

Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.

Then all bagged trees will select the strong predictor at the top of the tree and therefore all trees will look similar.

How do we avoid this?

Bagging - issues

We can penalize the splitting (like in pruning) with a penalty term that depends on the number of times a predictor is selected at a given length

We can restrict how many times a predictor can be used

We only allow a certain number of predictors

Bagging - issues

Remember we want i.i.d such as the bias to be the same and variance to be less?

Other ideas?

What if we consider only a subset of the predictors at each split?

We will still get correlated trees unless ….

we **randomly** select the subset !

Random Forests

Random Forests

As in bagging, we build a number of decision trees on bootstrapped training samples each time a split in a tree is considered, a random sample of *m* predictors is chosen as split candidates from the full set of p predictors.

Note that if $m = p$, then this is bagging.

Random Forests Algorithm

```
For b = 1 to B:
    (a) Draw a bootstrap sample Z∗ of size $N$ from the training data.
    (b) Grow a random-forest tree to the bootstrapped data, by recursively
repeating the following steps for each terminal node of the tree, until the minimum
node size $\boldsymbol{n_{min}}$ is reached.
        i. Select $\boldsymbol{m}$ variables at random from the $p$ variables.
        ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
Output the ensemble of trees.
```

To make a prediction at a new point *x* we do:

Random Forests Tuning

The inventors make the following recommendations:

- For classification, the default value for *m* is $\sqrt{p}$ and the minimum node size is one.
- For regression, the default value for m is *p/3* and the minimum node size is five.

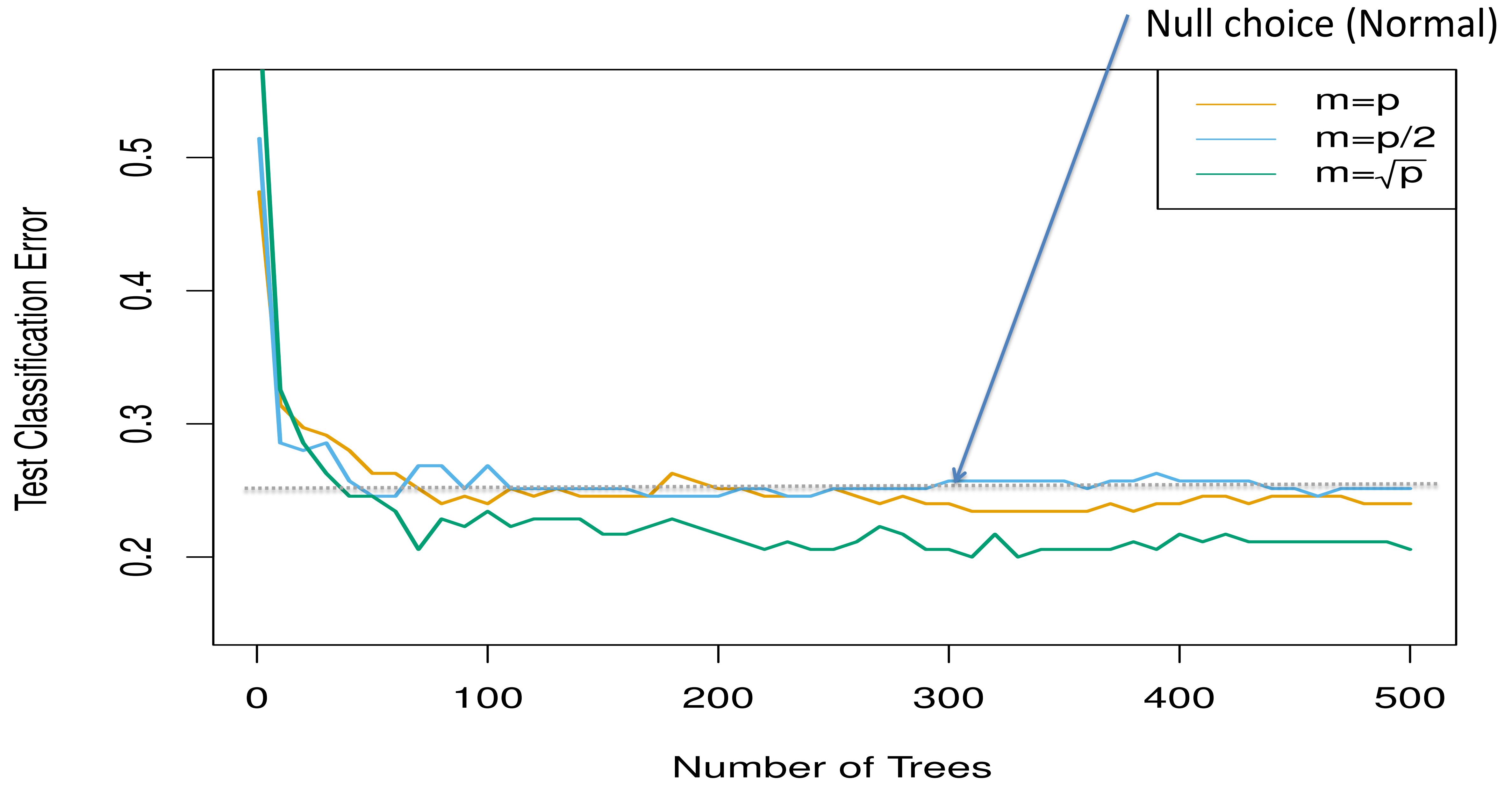
In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

Like with Bagging, we can use OOB and therefore RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

Example

- 4,718 genes measured on tissue samples from 349 patients.
- Each gene has different expression
- Each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer.

Use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.


Null choice (Normal)
m=p
m=p/2
m=√p
Test Classification Error
0.2
0.3
0.4
0.5
0
100
200
300
400
500
Number of Trees

Random Forests Issues

When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when *m* is small

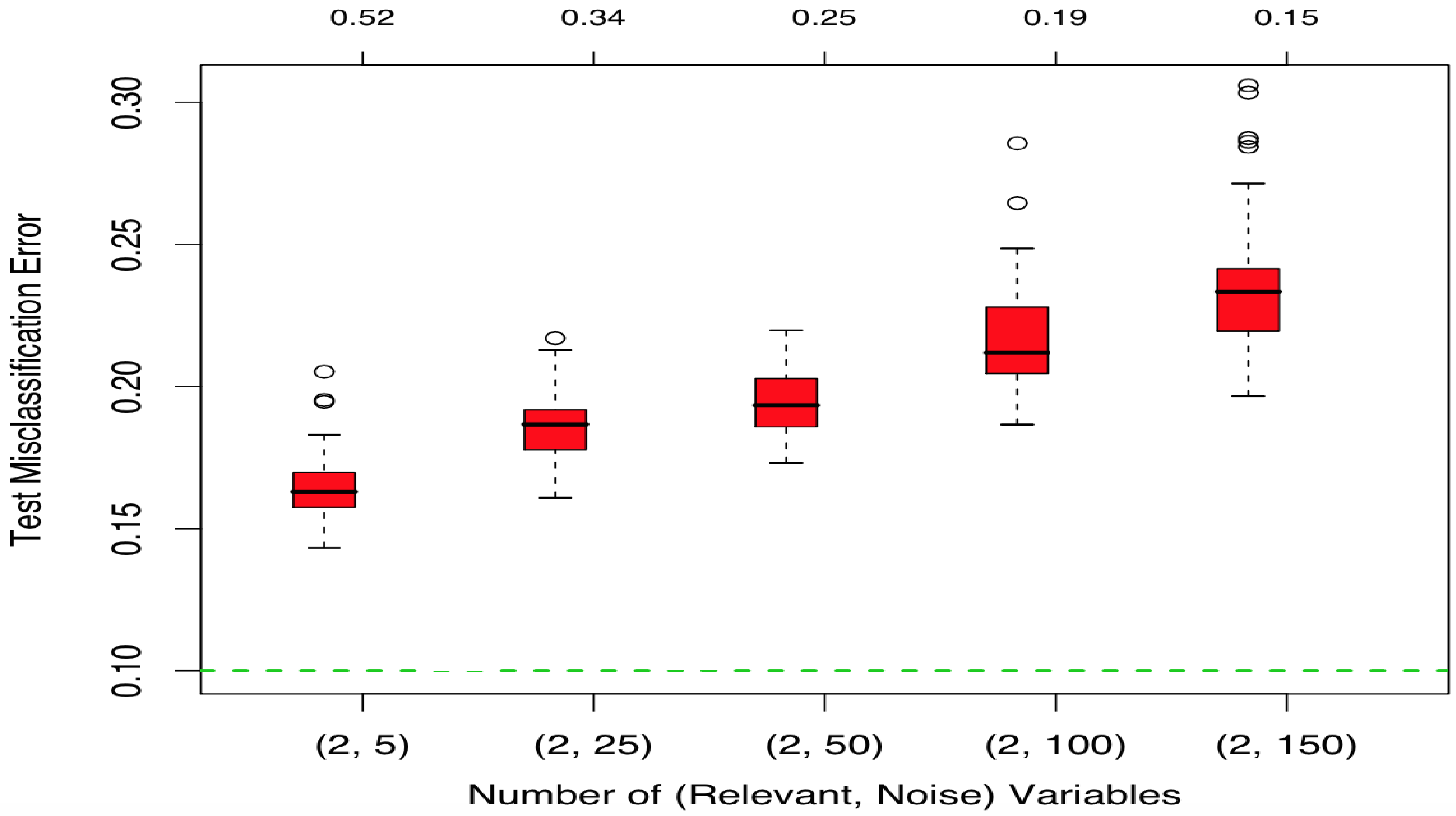
Why?

Because:

At each split the chance can be small that the relevant variables will be selected

For example, with 3 relevant and 100 not so relevant variables the probability of any of the relevant variables being selected at any split is ~0.25

Probability of being selected



Can RF overfit?

Random forests “cannot overfit” the data wrt to number of trees.

Why?

The number of trees, *B* does not mean increase in the flexibility of the model

I have seen discussion about gains in performance by controlling the depths of the individual trees grown in random forests. I usually use full-grown trees and seldom it costs much (in the classification error) and results in one less tuning parameter.

Boosting

Boosting

Boosting is a general approach that can be applied to many statistical learning methods for regression or classification.

Bagging: Generate multiple trees from bootstrapped data and average the trees.

Recall bagging results in i.d. trees and not i.i.d.

RF produces i.i.d (or more independent) trees by randomly selecting a subset of predictors at each step

Boosting

Boosting works very differently.

1. Boosting does not involve bootstrap sampling
2. Trees are grown sequentially: each tree is grown using information from previously grown trees

3. Like bagging, boosting involves combining a large number of decision trees, $f^1, \ldots, f^B$

Sequential fitting

Given the current model,

- we fit a decision tree to the **residuals** from the model. Response variable now is the residuals and not Y
- We then add this new decision tree into the fitted function in order to update the residuals
- The learning rate has to be controlled

Boosting for regression

1. Set *f(x)=0* and $r_i = y_i$ for all *i* in the training set.

2. For *b=1,2,...,B*, repeat:

   a. Fit a tree with d splits(+1 terminal nodes) to the training data (X, r).

   b. Update the tree by adding in a shrunken version of the new tree:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

   c. Update the residuals,

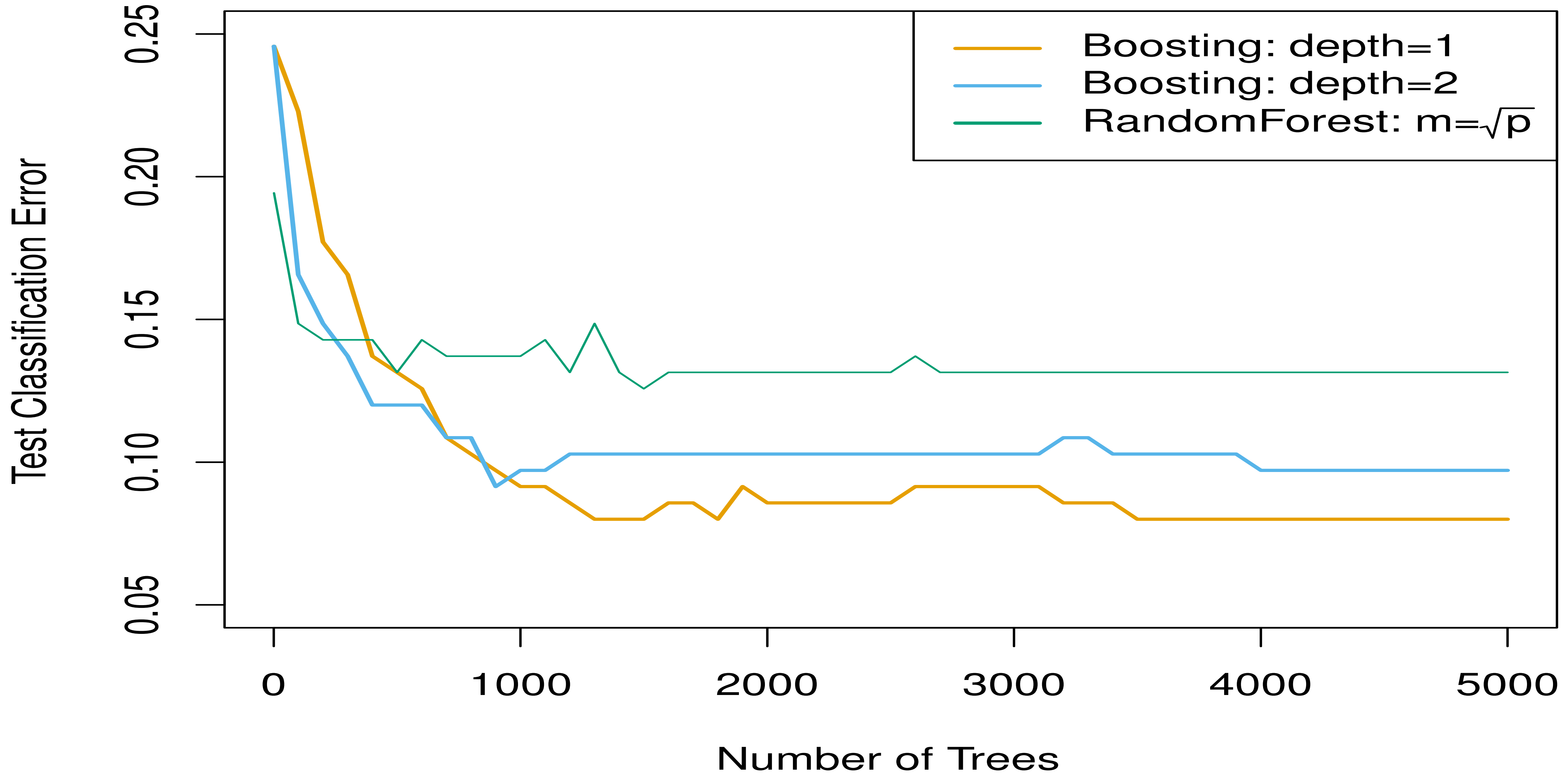
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

Boosting tuning parameters

- The number of trees B. RF and Bagging do not overfit as B increases. Boosting can overfit! **Cross Validation**
- The shrinkage parameter λ, a small positive number. Typical values are 0.01 or 0.001 but it depends on the problem. λ only controls the learning rate
- The number d of splits in each tree, which controls the complexity of the boosted ensemble. Stumpy trees, d = 1 works well.


Boosting: depth=1
Boosting: depth=2
RandomForest: $m=\sqrt{p}$
Test Classification Error
0.05
0.10
0.15
0.20
0.25
0
1000
2000
3000
4000
5000
Number of Trees

Different flavors

- C5.0, The most significant feature unique to C5.0 is a scheme for deriving rule sets. After a tree is grown, the splitting rules that define the terminal nodes can sometimes be simplified: that is, one or more condition can be dropped without changing the subset of observations that fall in the node.

- CART or Classification And Regression Trees is often used as a generic acronym for the term Decision Tree, though it apparently has a more specific meaning. In sum, the CART implementation is very similar to C4.5. **Used in sklearn**

Missing data

- What if we miss predictor values?
  - Remove those examples => depletion of the training set
  - Impute the values either with mean, knn, from the marginal or joint distributions
- Trees have a nicer way of doing this
  - Categorical

Where are we?

- Algorithms
  - DTs
  - Perceptron + Winnow
  - Gradient Descent
  - SVM
- 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?

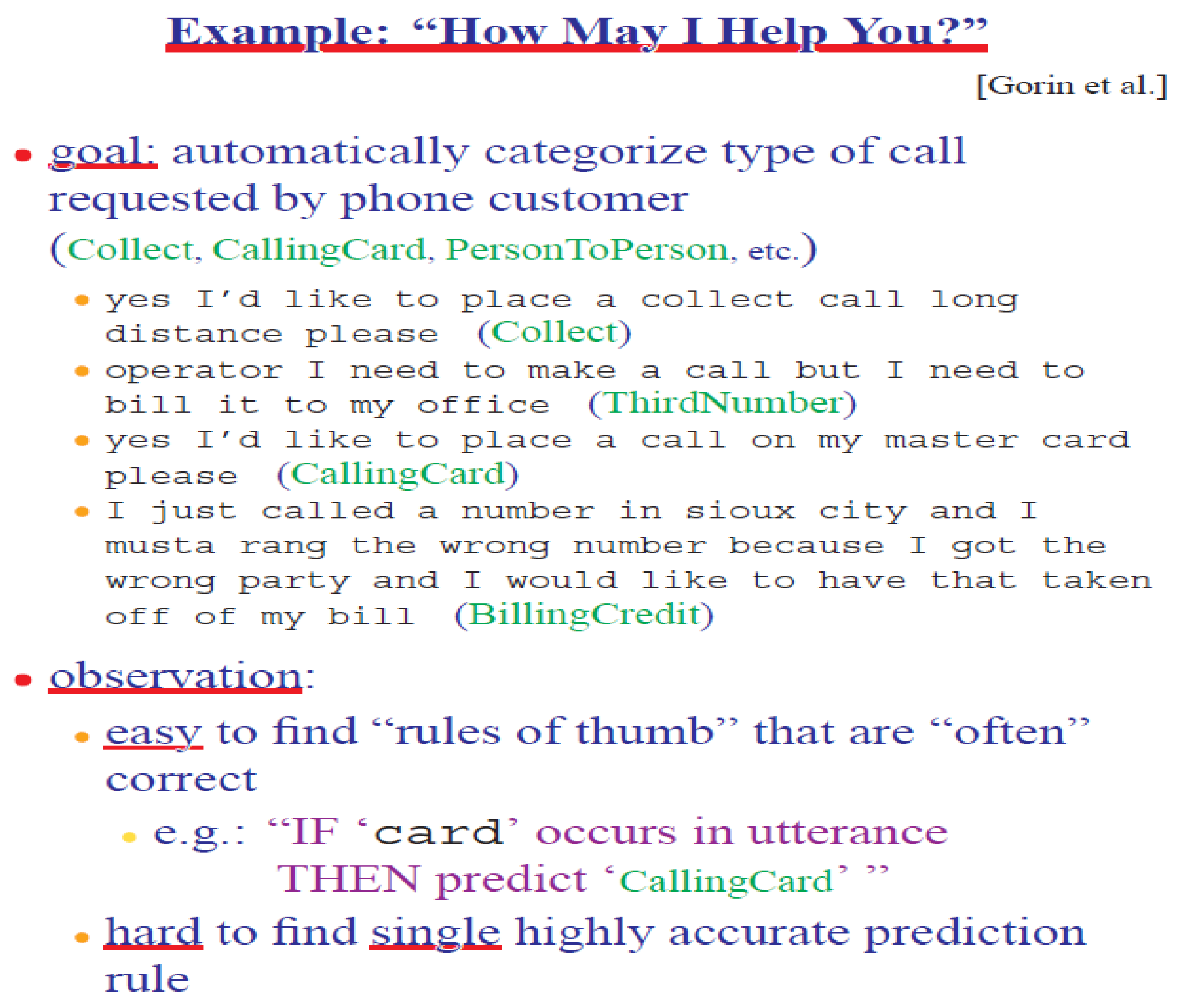
Boosting

- 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 $\log(\frac{1}{\varepsilon})$.
  - 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 $(\boldsymbol{x}_1, y_1), \dots\ (\boldsymbol{x}_m, y_m)$
- $y_i \in \{-1, +1\}$ is the correct label of instance $\boldsymbol{x}_i \in \boldsymbol{X}$
- For $t = 1, \dots, T$
  - Construct a distribution D_t on $\{1, \dots m\}$
  - Find weak hypothesis ("rule of thumb")

    $$h_t : \boldsymbol{X} \to \{-1, +1\}$$

    with small error ε_t on D_t:

    $$\varepsilon_t = \Pr_D[h_t(\boldsymbol{x}_i) \neq y_i]$$
- Output: final hypothesis H_{final}

Ad

Think about unwrapping it all the way to $1/m$

$$Z_t = \sum_i D_t(i)\exp(-\alpha_t\, y_i h_t(\boldsymbol{x}_i))$$

- Constructing D_t on $\{1, \ldots m\}$:
  - $D_1(i) = 1/m$
  - Given D_t and h_t :
    - $D_{t+1} = D_t(i)/z_t \times e^{-\alpha_t}$ if $y_i = h_t(\boldsymbol{x}_i)$
      $D_t(i)/z_t \times e^{+\alpha_t}$ if $y_i \neq h_t(\boldsymbol{x}_i)$
      $$= \frac{D_t(i)}{z_t} \times \exp(-\alpha_t\, y_i\, h_t(\boldsymbol{x}_i))$$

    where z_t = normalization constant
    and $\alpha_t = ½ \ln\{ (1 - \varepsilon_t)/\varepsilon_t \}$

$<$ 1; smaller weight

$>$ 1; larger weight

$$e^{+\alpha_t} = sqrt\left\{\frac{1-\varepsilon_t}{\varepsilon_t}\right\} > 1$$

Notes about α_t:

- ❑ 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: $H_{final}(\boldsymbol{x}) = sign\left(\sum_t \alpha_t\, h_t(\boldsymbol{x})\right)$

Administration (11

Are we recording? YES!

Available on the web site

- Remember that all the lectures are available on the website before the class
  - Go over it and be prepared
  - A new set of written notes will accompany most lectures, with some more details, examples and, (when relevant) some code.

- HW 3: Due on 11/16/20
  - You cannot solve all the problems yet.
  - Less time consuming; no programming

- Cheating
  - Several problems in HW1 and HW2

Projects

- CIS 519 students need to do a team project: Read the project descriptions
  - Teams will be of size 2-4
  - We will help grouping if needed

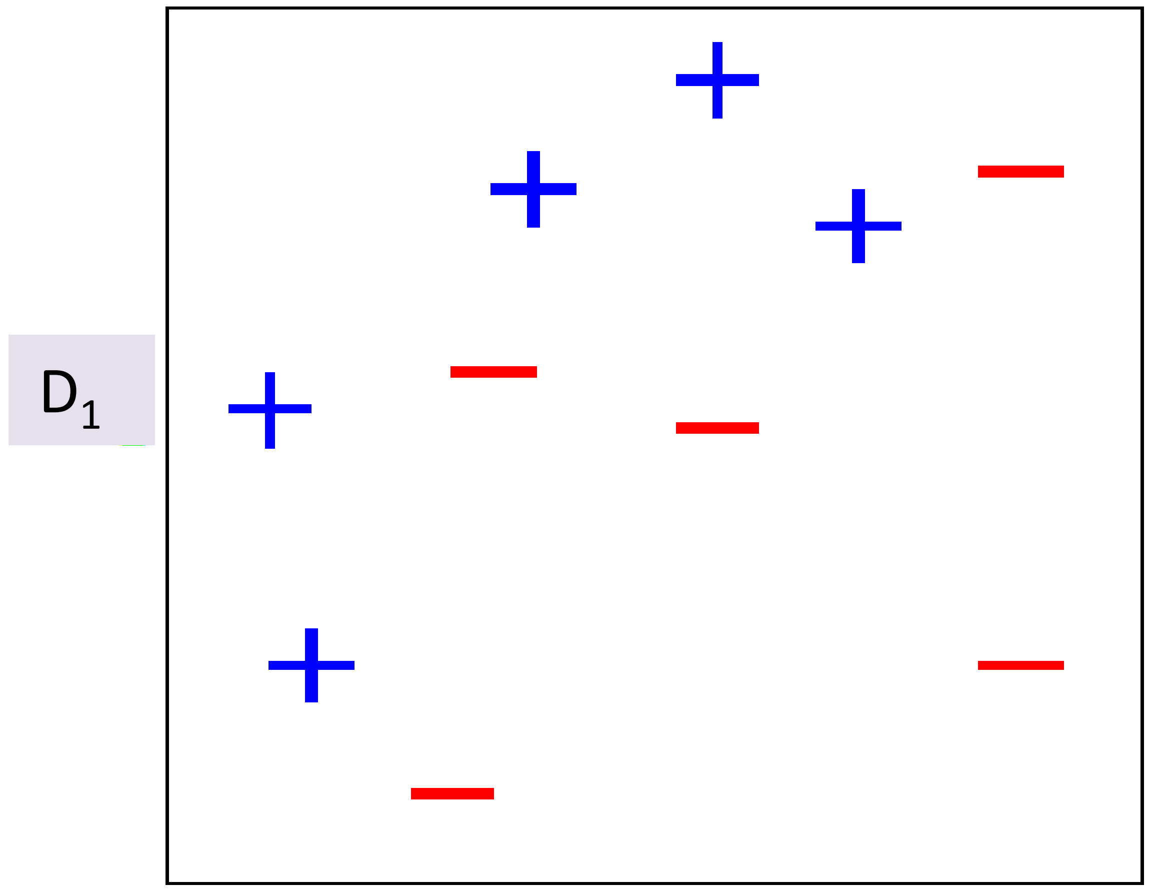
- There will be 3 projects.
  - Natural Language Processing (Text)
  - Computer Vision (Images)
  - Speech (Audio)

- In all cases, we will give you datasets and initial ideas
  - The problem will be multiclass classification problems
  - You will get annotated data only for some of the labels, but will also have to predict other labels
  - 0-zero shot learning; few-shot learning; transfer learning

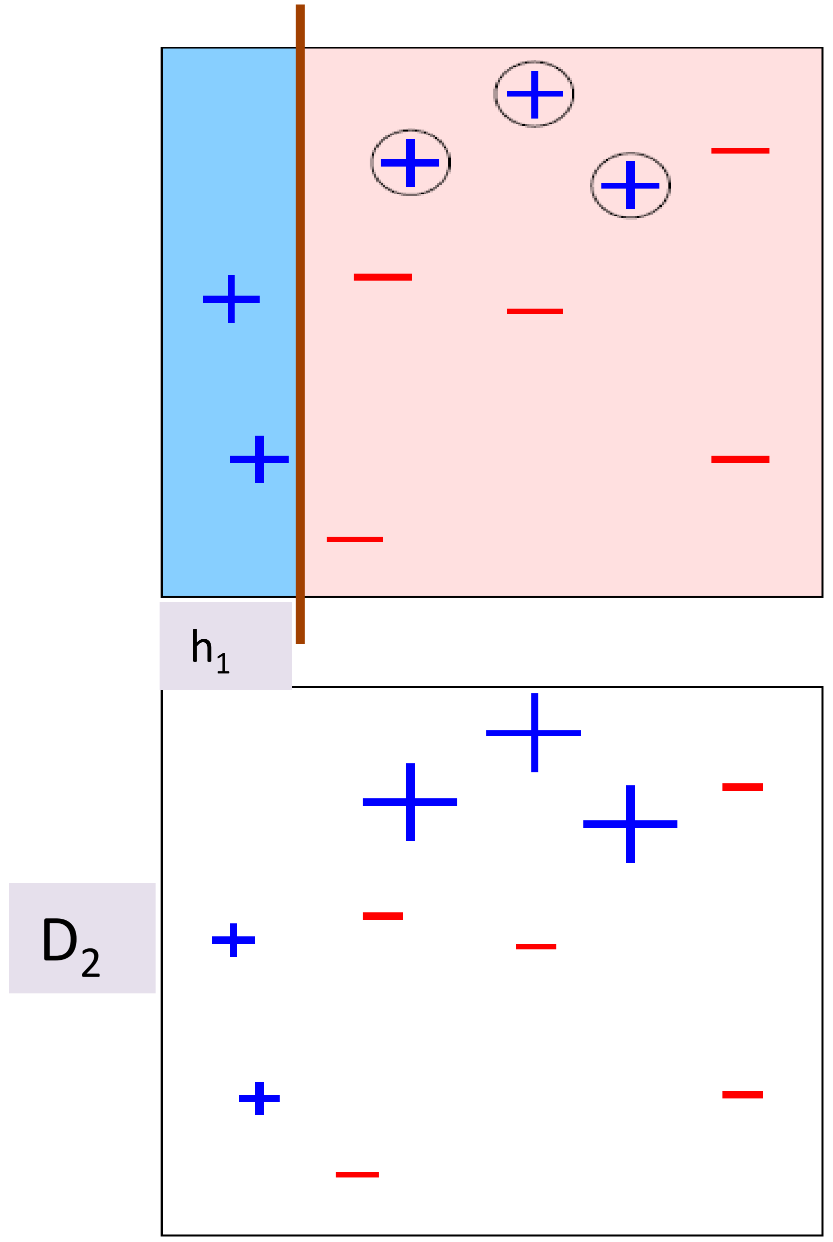
- A detailed note will come out today.

- Timeline:
  - 11/11 Choose a project and team up
  - 11/23 Initial proposal describing what your team plans to do
  - 12/2 Progress report
  - 12/15-20 (TBD) Final paper + short video
- Try to make it interesting!

A Toy Example



Round 1

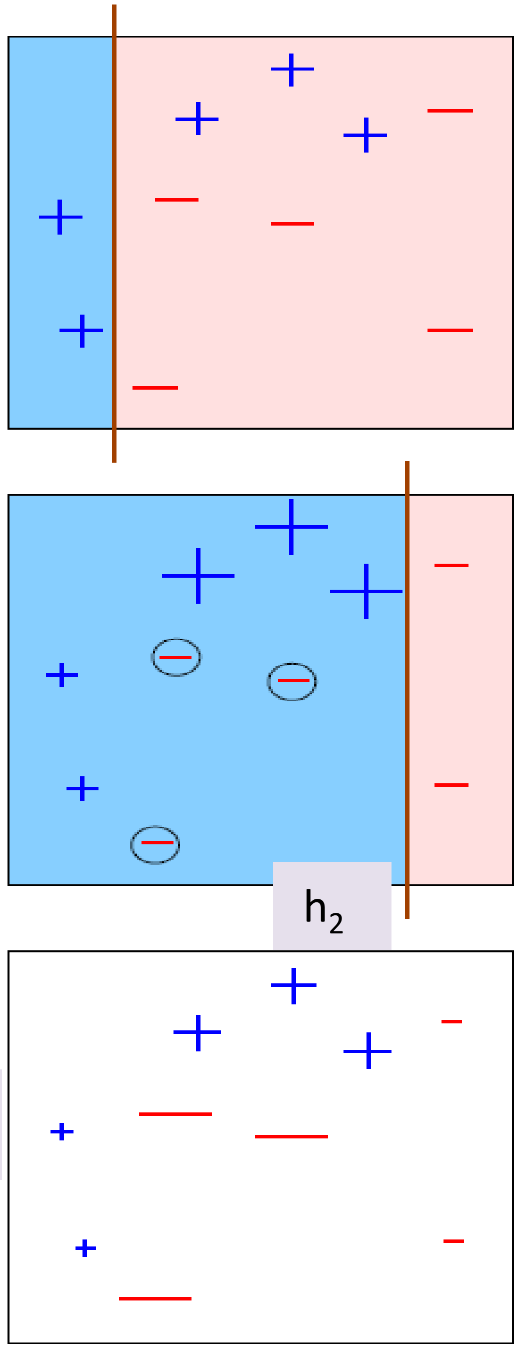


$\varepsilon_1 = 0.3$

$\alpha_1 = 0.42$

A

Round 2

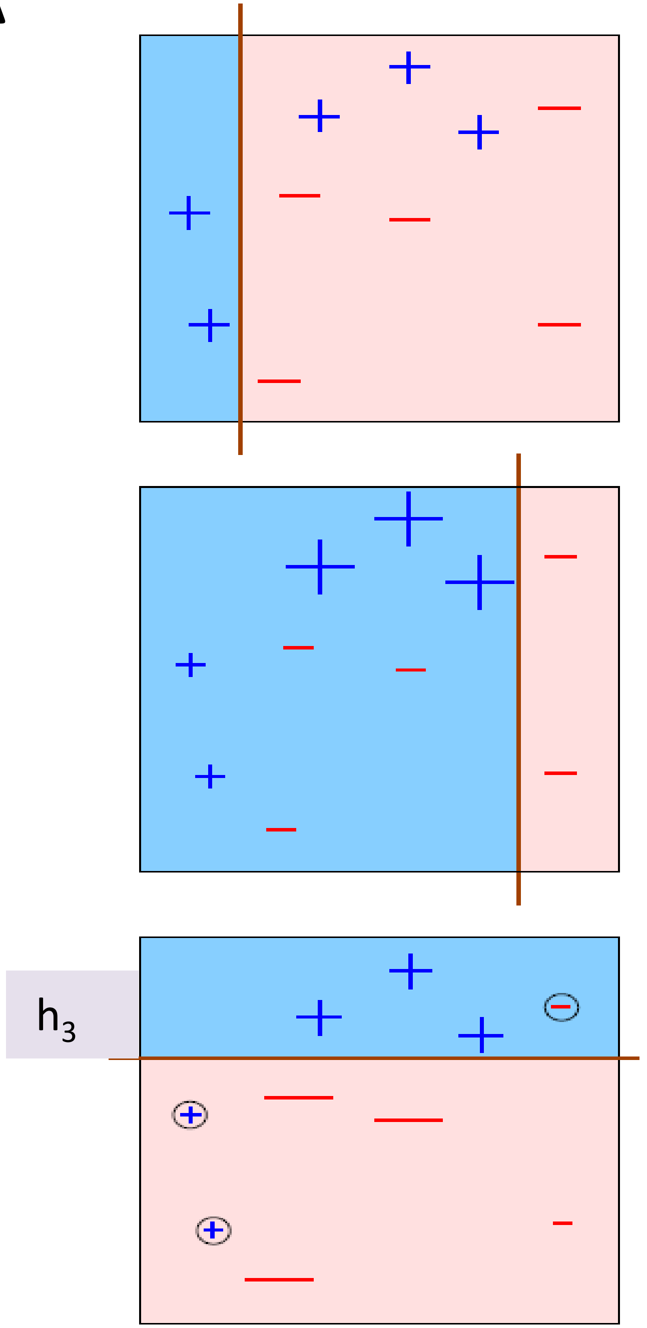


D$_3$

$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$

A ·

Round 3

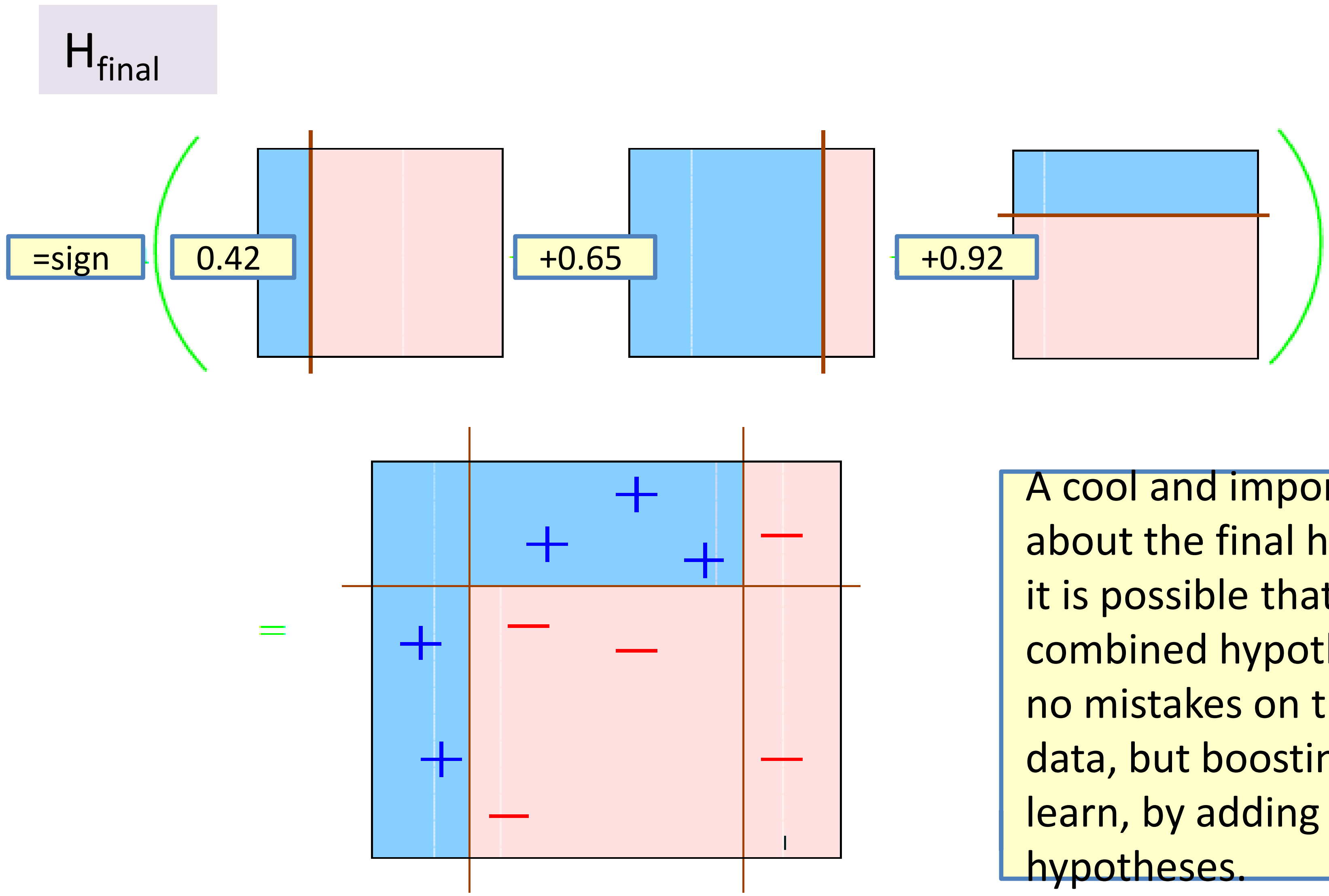


$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

A Toy Example

Final Hypothesis



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

$$\text{training error}(H_{\text{final}}) \leq \prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right]$$
$$= \prod_t \sqrt{1-4\gamma_t^2}$$
$$\leq \exp\left(-2\sum_t \gamma_t^2\right)$$

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

- 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 $\gamma_t \gg \gamma$

What does training error < exp{-2\gamma^2 T} mean? (here: \gamma--small constant; T number of boosting rounds)

Why is the Theorem stated in terms of training error? What guarantees do we really want? [we want guarantees on ..... error; but......]

Analyzing Adaboost

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

$$\text{training error}(H_{\text{final}}) \le \prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \le \exp\left(-2\sum_t \gamma_t^2\right)$$

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

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

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

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

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

AdaBoost Proof (1)

$$D_{t+1} = \begin{cases} D_t(i)/Z_t \times e^{-\alpha_t} & \text{if } y_i = h_t(\boldsymbol{x}_i) \\ D_t(i)/Z_t \times e^{+\alpha_t} & \text{if } y_i \neq h_t(\boldsymbol{x}_i) \end{cases}$$

$$= \frac{D_t(i)}{Z_t} \times \exp(-\alpha_t \, y_i \, h_t(\boldsymbol{x}_i))$$

- Let $f(\boldsymbol{x}) = \sum_t \alpha_t h_t(\boldsymbol{x}) \rightarrow H_{final}(\boldsymbol{x}) = sign(f(\boldsymbol{x}))$
- Step 1: the final weight of an example (via unwrapping recursion)

The final "weight" of the i-th example

$$D_{final}(i) = \frac{\exp(-y_i \sum_t \alpha_t h_t(\boldsymbol{x}_i))}{\prod_t Z_t} \cdot \frac{1}{m}$$

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

AdaBoost Proo

$$D_{final}(i) = \frac{\exp(-y_i \sum_t \alpha_t h_t(\boldsymbol{x}_i))}{\prod_t Z_t} \cdot \frac{1}{m} = \frac{e^{-y_i f(\boldsymbol{x}_i)}}{\prod_t Z_t} \cdot \frac{1}{m}$$

- Step 2: $training\ error(H_{final}) \leq \prod_t Z_t$
- Proof:
  - $H_{final}(\boldsymbol{x}) \neq y \rightarrow y f(\boldsymbol{x}) \leq 0 \rightarrow e^{-y f(\boldsymbol{x})} \geq 1$

  So:
  - $training\ error(H_{final})$

$$\left\{ \begin{array}{l} = \frac{1}{m} \sum_i 1 \quad if\ y_i \neq H_{final}(\boldsymbol{x}_i) \\ = \frac{1}{m} \sum_i 0 \quad else \end{array} \right.$$

$$\leq \frac{1}{m} \sum_i e^{-y_i f(\boldsymbol{x}_i)}$$

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

$$= \prod_t Z_t$$

The definition of training error

Always holds for mistakes (see above)

Using Step 1

D is a distribution over the m examples

AdaBoost Proof(3)

$$D_{t+1} = \begin{cases} D_t(i)/z_t \times e^{-\alpha_t} & \text{if } y_i = h_t(\boldsymbol{x}_i) \\ D_t(i)/z_t \times e^{+\alpha_t} & \text{if } y_i \neq h_t(\boldsymbol{x}_i) \end{cases}$$

$$= \frac{D_t(i)}{z_t} \times \exp(-\alpha_t\, y_i\, h_t(\boldsymbol{x}_i))$$

- Step 3: $Z_t = 2\,(\epsilon_t(1 - \epsilon_t))^{\frac{1}{2}}$
- Proof:

By definition of Z_t; it's a normalization term

$$Z_t = \sum_i D_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$$

Why does it work? The Weak Learning Hypothesis

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

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

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

The definition of ϵ_t

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

The definition of α_t

$$e^{+\alpha_t} = sqrt\left\{\frac{1 - \epsilon_t}{\epsilon_t}\right\} > 1$$

$$= 2\,(\epsilon_t(1 - \epsilon_t))^{\frac{1}{2}}$$

Step 2 $training\ error(H_{final}) \leq \prod_t Z_t$ and step 3 together prove the Theorem.

→ The error of the final hypothesis can be as low as you want.

Summary of Ensemble Methods

- Boosting
- Bagging
- Random Forests

Boosting

- Initialization:
  - Weigh all training samples equally
- Iteration Step:
  - Train model on (weighted) train set
    - Choose your favorite hypothesis space & learning algorithm
  - Compute error of model on train set
  - Update the distribution:
    - Increase/decrease weights on training cases model gets wrong/correct.
- 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”.

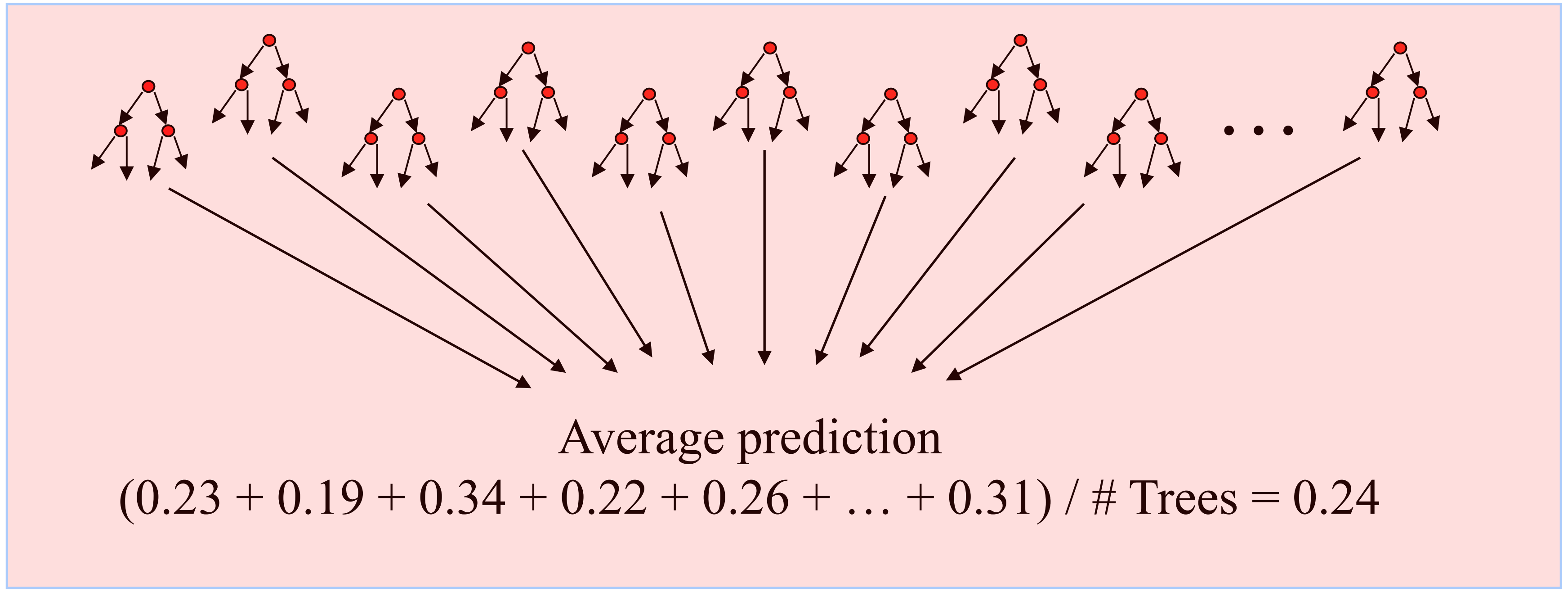
Bagging

- 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

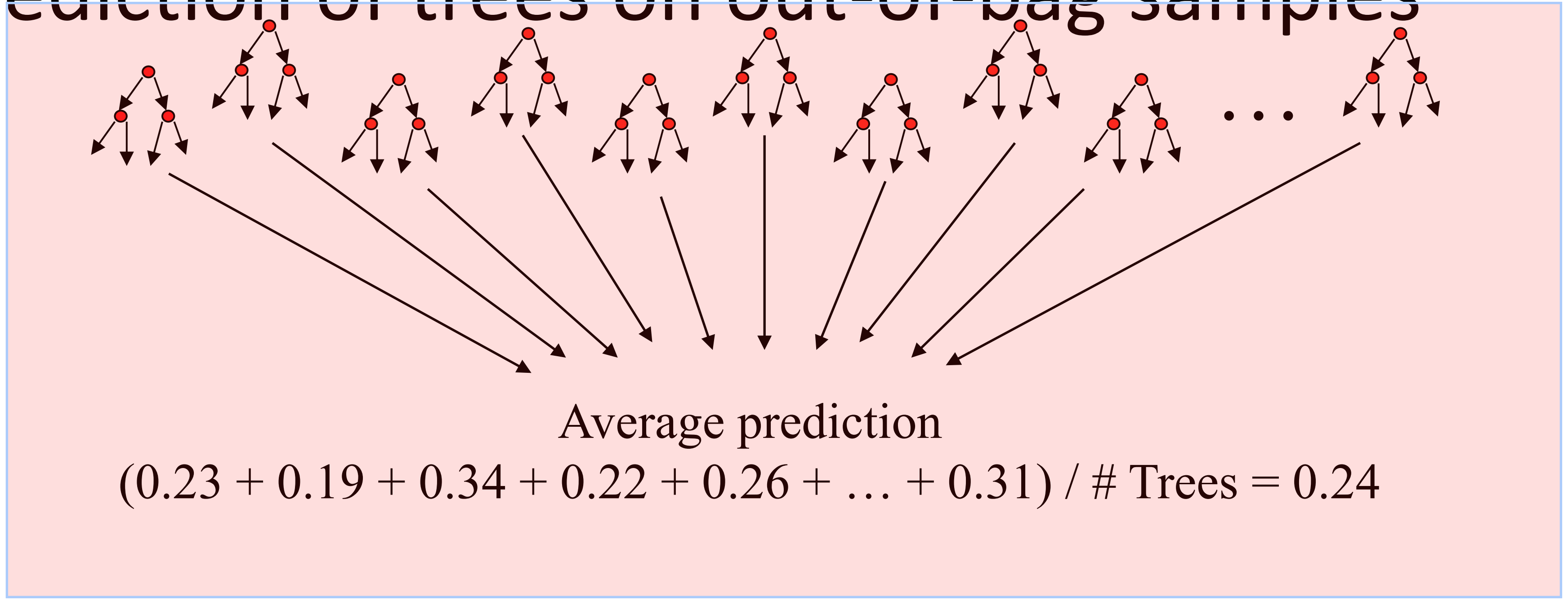
- Draw 100 bootstrap samples of data
- Train trees on each sample → 100 trees
- Average prediction of trees on out-of-bag samples



…
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


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

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 $\boldsymbol{x}$,
  - Prediction = $\operatorname{sgn}(\boldsymbol{w}^T \boldsymbol{x})$
  - Where $\boldsymbol{w}$ is the learned model
- The output   is a single bit

Multi-Categorical Output Tasks

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

Setting

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

You know how to train a binary classifier, say, an SVM. Now you have a 3-labels classification problem. What would you do?

Binary to Multiclass

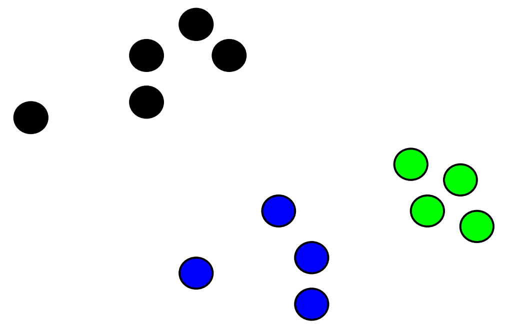
- Most schemes for multiclass classification work by reducing the problem to that of binary classification.
- There are multiple ways to decompose the multiclass prediction into multiple binary decisions
  - ✓ One-vs-all
  - ✓ All-vs-all
  - Error correcting codes
- We will then talk about a more general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

One-Vs-All

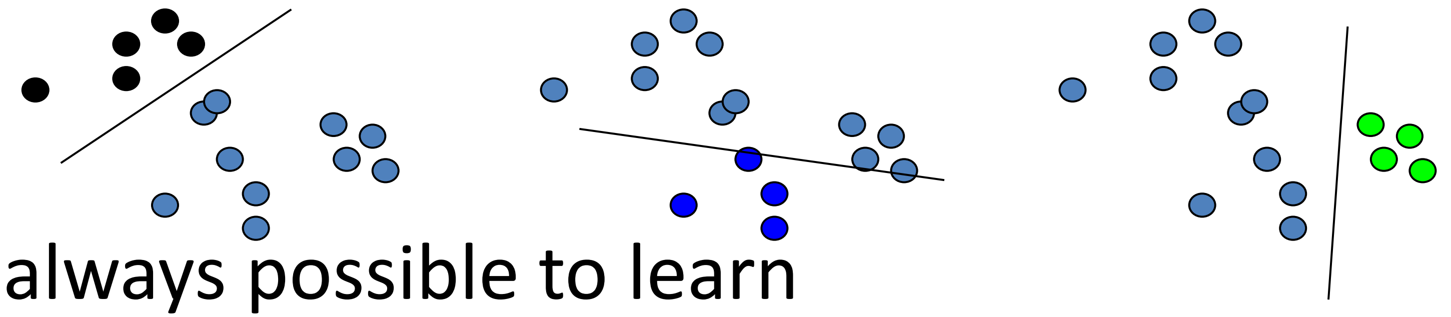
- Assumption: Each class can be separated from all the rest using a binary classifier in the hypothesis space.
- Learning: Decomposed to learning k independent binary classifiers, one for each class label.
- Learning:
  - Let D be the set of training examples.
  - $\forall$ label l, construct a binary 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 $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_k$
- Decision: Winner Takes All (WTA):
  - $f(x) = argmax_i \, \boldsymbol{v}_i^T \boldsymbol{x}$

Solving MultiClass with 1 vs All learning

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



- Decompose into binary problems



- Not always possible to learn
- No theoretical justification
  - Also: need to make sure the range of all classifiers is the same (for the argmax)
- Note: in high dimensional spaces, it's likely that things are separable

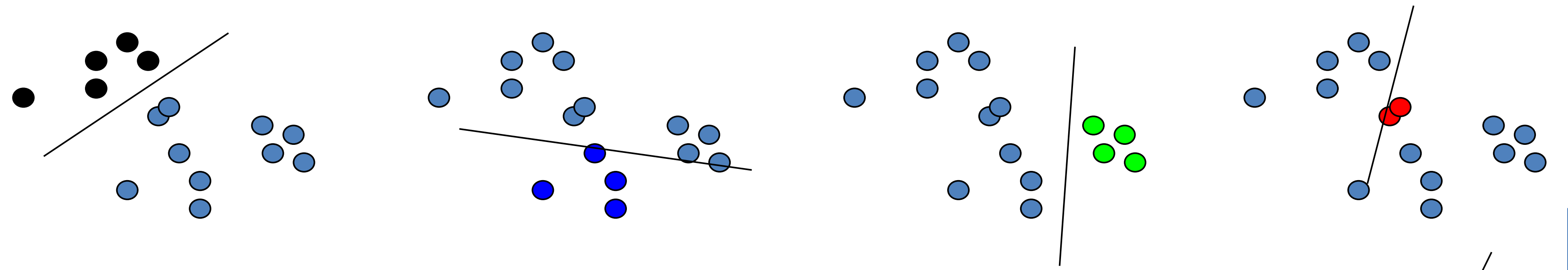
Learning via One-Versus-All (OvA) Assumption

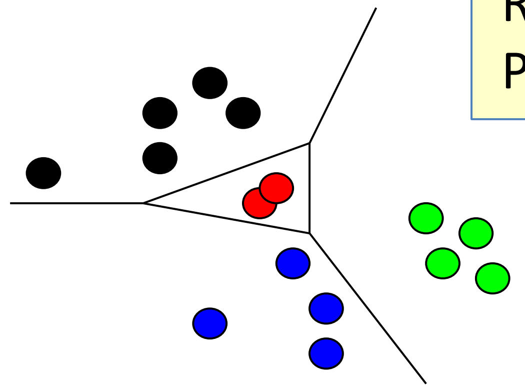
- Find $\boldsymbol{v}_r, \boldsymbol{v}_b, \boldsymbol{v}_g, \boldsymbol{v}_y \in \boldsymbol{R}^n$ such that

$$\boldsymbol{v}_r . \boldsymbol{x} > 0 \;\; iff \; y = red \qquad ⊗$$
$$\boldsymbol{v}_b . \boldsymbol{x} > 0 \;\; iff \; y = blue \qquad ✓$$
$$\boldsymbol{v}_g . \boldsymbol{x} > 0 \;\; iff \; y = green \qquad ✓$$
$$\boldsymbol{v}_y . \boldsymbol{x} > 0 \;\; iff \; y = yellow \qquad ✓$$

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

- **Classification:** $f(\boldsymbol{x}) = argmax_i \boldsymbol{v}_i \, \boldsymbol{x}$





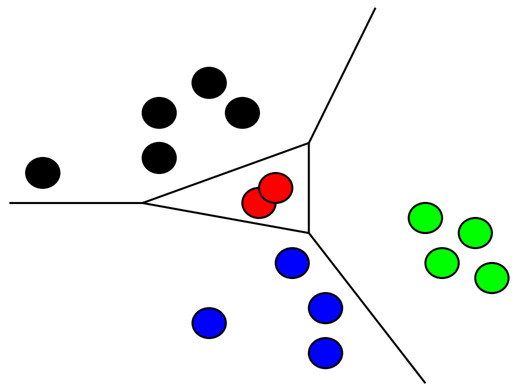
Real Problem

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 $[k\ choose\ 2] \sim k^2$ independent binary classifiers, one corresponding to each pair of class labels. For the pair (i, j):
  - Positive example: all examples 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 $\mathbf{x}$ to take label i if i wins on $\mathbf{x}$ more often than j $(j = 1, \dots k)$
  - A tournament: start with $\frac{n}{2}$ pairs; continue with winners .

Learning via All-Verses-All (AvA) Assumption

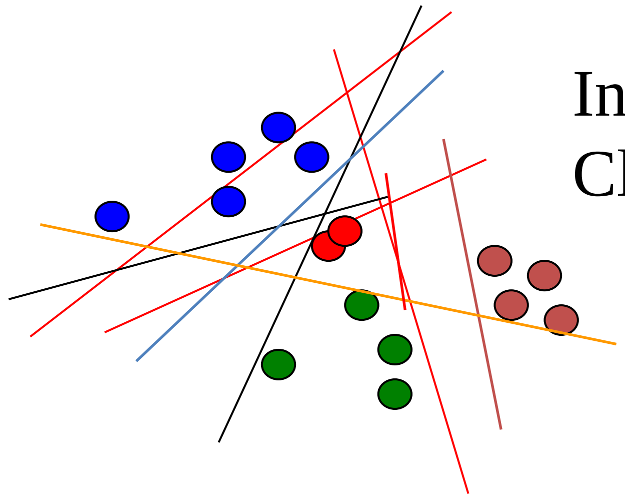
- Find $\boldsymbol{v}_{rb}, \boldsymbol{v}_{rg}, \boldsymbol{v}_{ry}, \boldsymbol{v}_{bg}, \boldsymbol{v}_{by}, \boldsymbol{v}_{gy} \in \boldsymbol{R}^d$ such that
  - $\boldsymbol{v}_{rb} \cdot \boldsymbol{x} > 0 \; if \; y = red$
    $< 0 \; if \; y = blue$
  - $\boldsymbol{v}_{rg} \cdot \boldsymbol{x}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



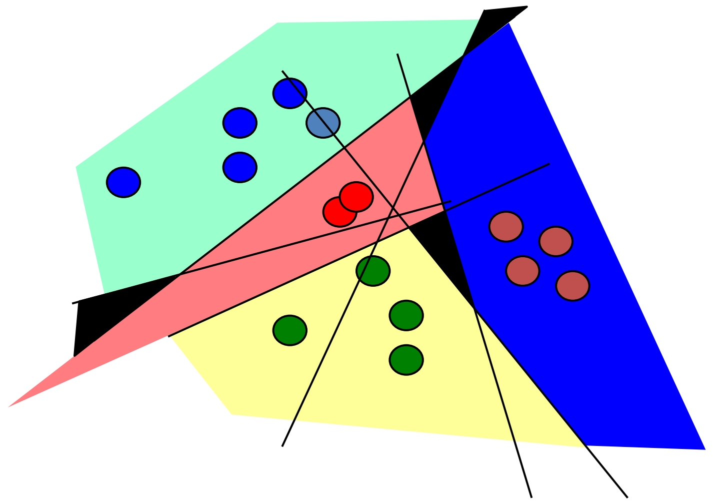
It is possible to separate all k classes with the $O(k^2)$ classifiers

$\boldsymbol{H} = \boldsymbol{R}^{kkn}$

How to classify?



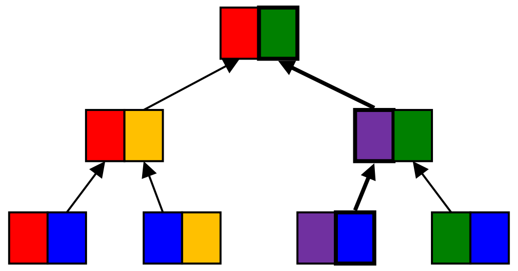
Individual Classifiers



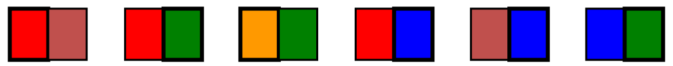
Decision Regions

Classifying with AvA

Tournament



Majority Vote



1 red, 2 blue, 2 green

➔ ?

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

One-vs-All vs. All vs. All

(Think about Dual/Primal)

- Assume m examples, k class labels.
  - For simplicity, say, $\frac{m}{k}$ in each.
- One vs. All:
  - Classifier f_i: $\frac{m}{k}$ (+) and $\frac{(k-1)m}{k}$ (-)
  - Decision:
  - Evaluate k linear classifiers and do Winner Takes All (WTA):
  - $$f(\boldsymbol{x}) = argmax_i f_i(\boldsymbol{x}) = argmax_i\, \boldsymbol{v}_i^T \boldsymbol{x}$$
- All vs. All:
  - Classifier f_{ij}: $\frac{m}{k}$ (+) and $\frac{m}{k}$ (-)
  - 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)?

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), +}

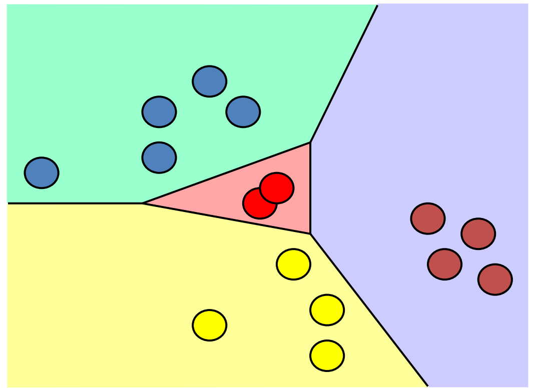
- To classify a new example x:
  - Evaluate hypothesis on the 4 binary problems {(x,P1) , (x,P2), (x,P3), (x,P4),}
  - Choose label that is most consistent with the results.
    - Use Hamming distance (bit-wise distance)
- Nice theoretical results as a function of the performance of the P_i s (depending on code & size)
- Potential Problems?

Label	P 1	P 2	P3	P4
1	-	+	-	+
2	-	+	+	-
3	+	-	-	+
4	+	-	+	+
k	-	+	-	-

Can you separate any dichotomy?

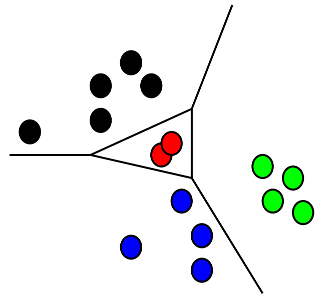
Problems with Decompositions

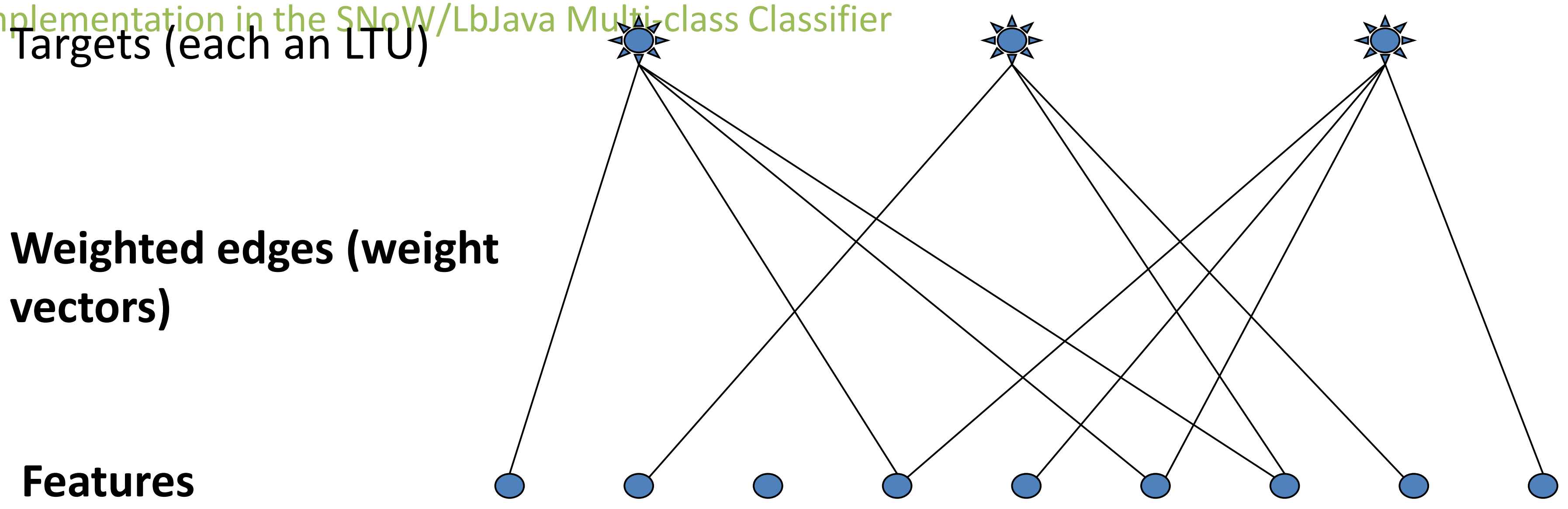
- Learning optimizes over local metrics
  - Does not guarantee good global performance
  - We don't care about the performance of the local classifiers
- Poor decomposition $\Rightarrow$ poor performance
  - Difficult local problems
  - Irrelevant local problems
- Especially true for Error Correcting 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.
- Nevertheless, the most dominant approach in applications is One Vs. All.
- Not clear how to generalize it well to problems with a very large # of labels.



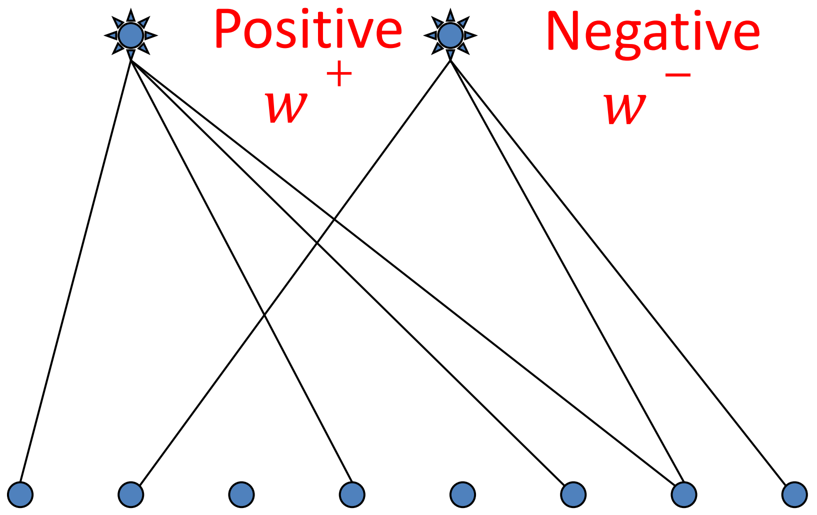
1 Vs All: Learning Architecture

- 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, including those the are "global"
  - e.g., see the implementation in the SNoW/LbJava Multi-class Classifier





Another View on Binary Classification



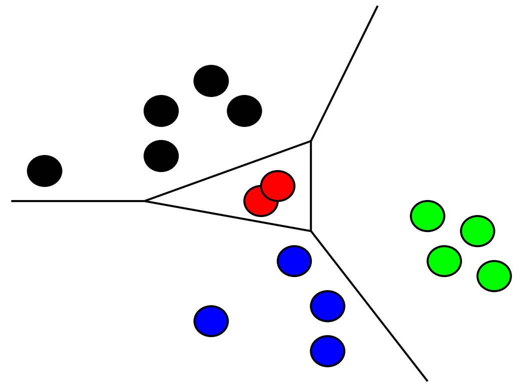
- Rather than a single binary variable at the output

- Represent 2 weights per feature;
  - Decision: using the "effective weight", the difference between $\boldsymbol{w}^+$ and $\boldsymbol{w}^-$
  - 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 suggest a "balanced" Update rule that takes into account how both sets of n weights predict on example x

$$\text{If } [(\boldsymbol{w}^+ - \boldsymbol{w}^-) \cdot \boldsymbol{x} \geq \theta] \neq y, \quad \boldsymbol{w}_i^+ \leftarrow \boldsymbol{w}_i^+ r^{yx_i}, \quad \boldsymbol{w}_i^- \leftarrow \boldsymbol{w}_i^- r^{-yx_i}$$

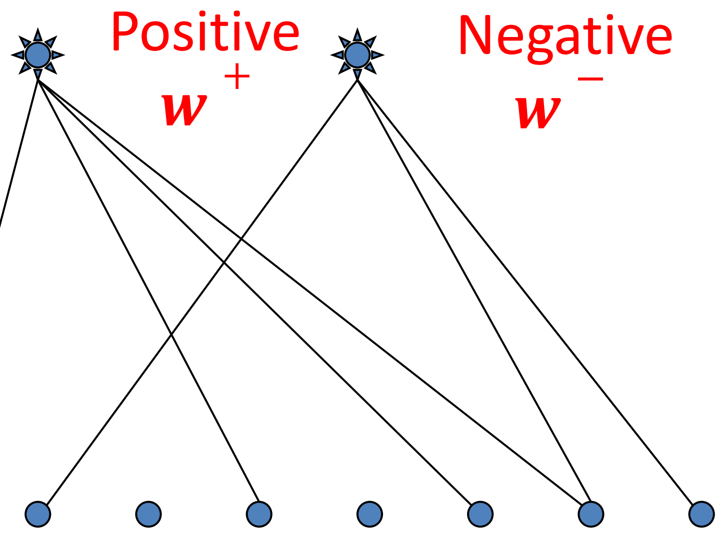
Can this be generalized to the case of k labels, $k > 2$?

We need a "global" learning approach



Recall: Winnow’s Extensions

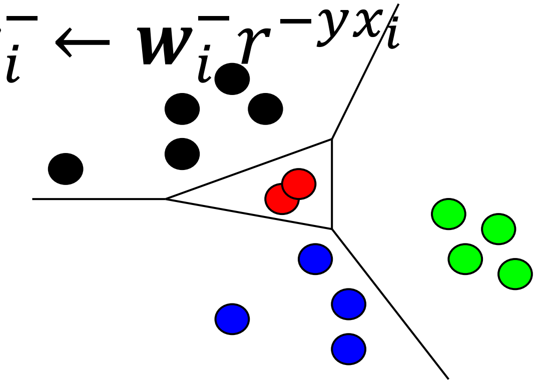
- Winnow learns monotone Boolean functions
- We extended to general Boolean functions via
- “Balanced Winnow”
  - 2 weights per variable;
  - Decision: using the “effective weight”, the difference between $\boldsymbol{w}^+$ and $\boldsymbol{w}^-$
  - This is equivalent to the Winner take all decision
  - 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



$$\text{If } [(\boldsymbol{w}^+ - \boldsymbol{w}^-) \cdot \boldsymbol{x} \geq \theta] \neq y, \quad \boldsymbol{w}_i^+ \leftarrow \boldsymbol{w}_i^+ r^{y x_i}, \quad \boldsymbol{w}_i^- \leftarrow \boldsymbol{w}_i^- r^{-y x_i}$$

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.
- We compared the activation of the different target nodes (classifiers) on a given example. (This example is more class + than class -)

$$\text{If } [(\boldsymbol{w}^+ - \boldsymbol{w}^-) \cdot \boldsymbol{x} \geq \theta] \neq y, \quad \boldsymbol{w}_i^+ \leftarrow \boldsymbol{w}_i^+ r^{y x_i}, \quad \boldsymbol{w}_i^- \leftarrow \boldsymbol{w}_i^- r^{-y x_i}$$

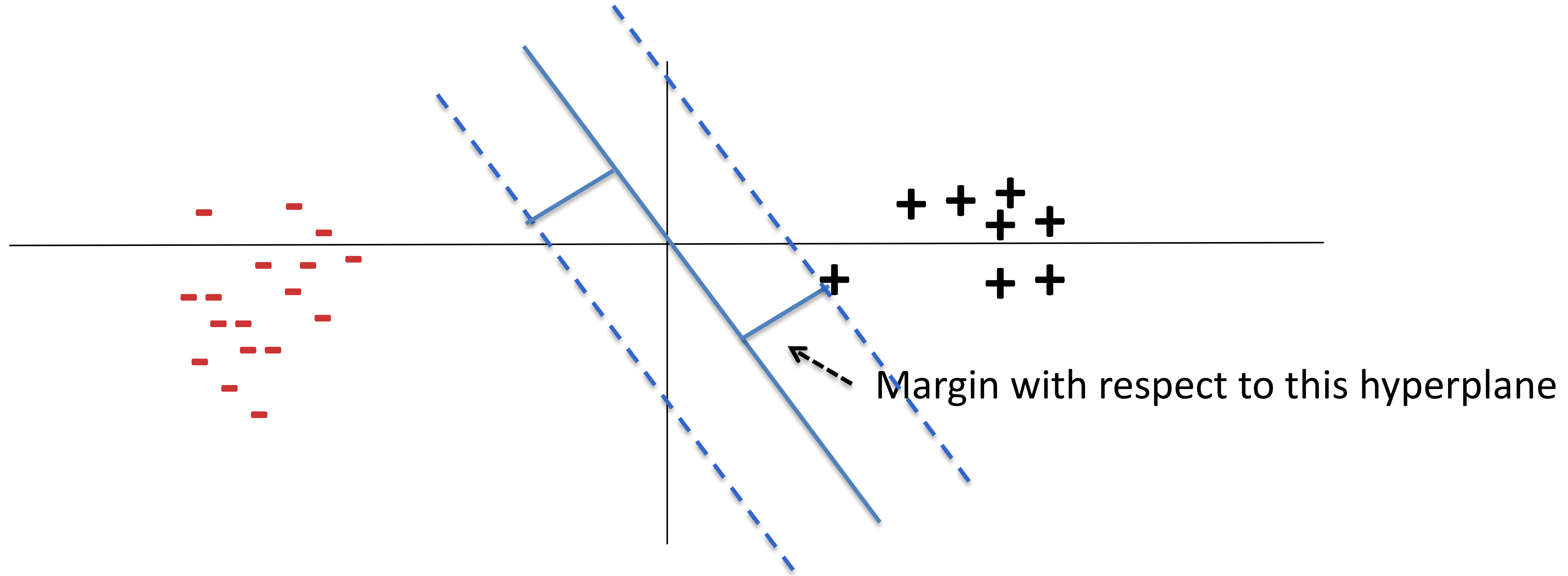
- Can this be generalized to the case of k labels, $k > 2$?

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

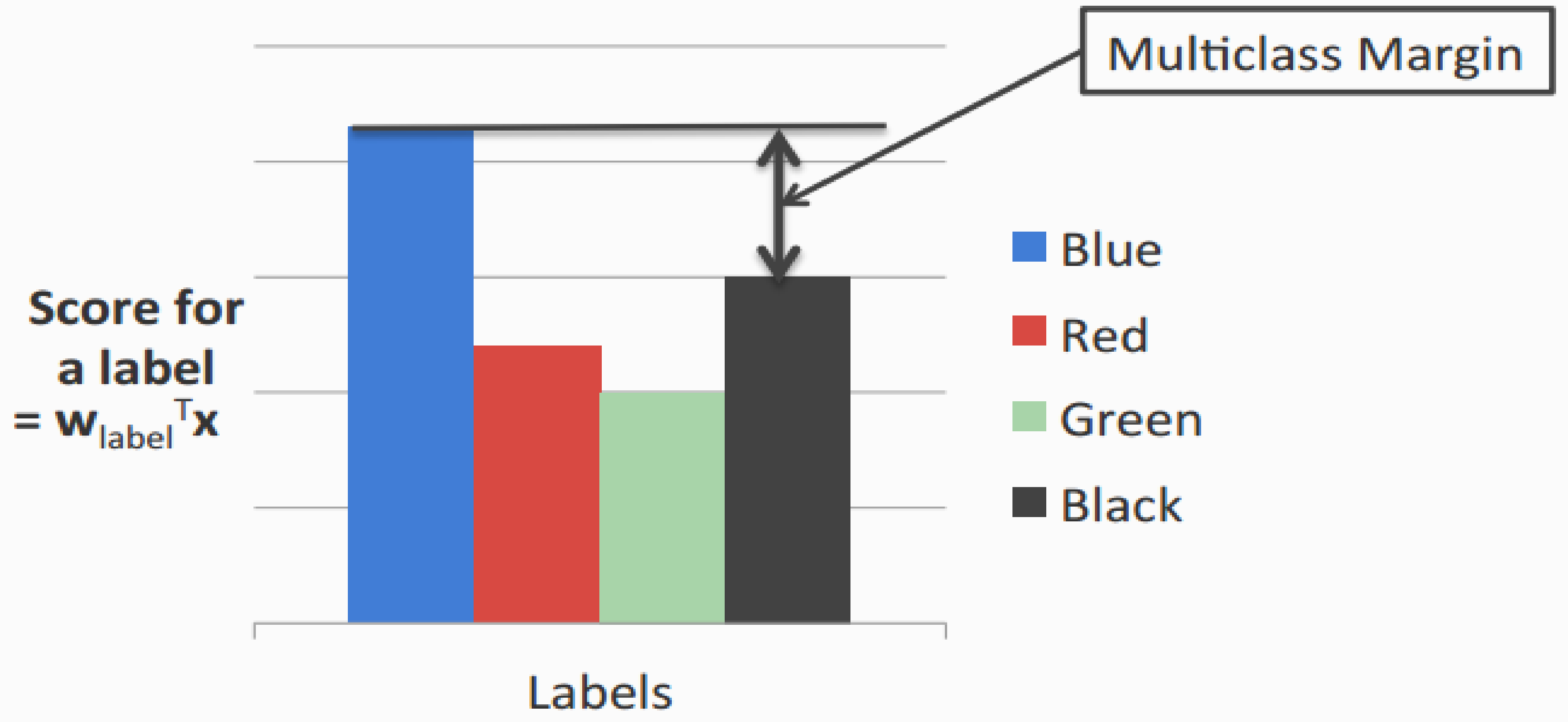
Recall: Margin for binary classifiers

- 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)
    - But, weight vectors are not learned independently
  - 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

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

Size of the weights. Effectively, regularizer

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

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

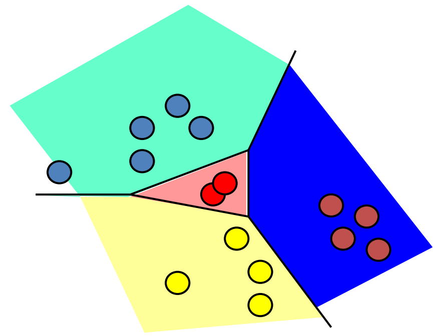
Slack variables. Not all examples need to satisfy the margin constraint.

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
    - $argmax_i\ \boldsymbol{w}_i^T \boldsymbol{x}$
- With K labels and inputs in $\mathbf{R}^n$, 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.
  - [Zimak et al. NeurIPS 2003]

Skip the rest of the notes

Constraint Classification

- The examples we give the learner are pairs $(\boldsymbol{x}, y), y \in \{1, \dots k\}$
- The "black box learner" (1 vs. all) we described might be thought of as a function of $\boldsymbol{x}$ only but, actually, we made use of the labels y
- How is y being used?
  - y decides what to do with the example $\boldsymbol{x}$; 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: $f_y(\boldsymbol{x}) = \boldsymbol{w}_y^T \boldsymbol{x}$
  - Then, we predict using: $y^* = argmax_{y=1,\dots,k} f_y(\boldsymbol{x})$
- Equivalently, we can say that we predict as follows:
  - Predict y iff $\forall y' \in \{1, \dots, k\}, y' \neq y \quad (\boldsymbol{w}_y^T - \boldsymbol{w}_{y'}^T)\, \boldsymbol{x} \geq 0 \quad (**)$
- So far, we did not say how we learn the k weight vectors $\boldsymbol{w}_y$ $(y = 1, \dots k)$
  - 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?

Linear Separability for Multiclass

- We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into
  - $\boldsymbol{w} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_k) \in \boldsymbol{R}^{nk}$

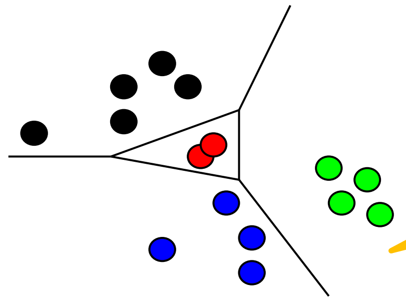
Notice: This is just a representational trick. We did not say how to learn the weight vectors.

- Key Construction: (Kesler Construction; Zimak's Constraint Classification)
  - We will represent each example $(\boldsymbol{x}, y)$, as an nk-dimensional vector, $\boldsymbol{x}_y$, with $\boldsymbol{x}$ embedded in the y-th part of it $(y = 1,2, \dots k)$ and the other coordinates are 0.
- E.g., $\boldsymbol{x}_y = (\boldsymbol{0}, \boldsymbol{x}, \boldsymbol{0}, \boldsymbol{0}) \in \boldsymbol{R}^{kn}$ $\quad (here\ k = 4, y = 2)$
- Now we can understand the n-dimensional decision rule:
- Predict y iff $\forall\ y' \in \{1, \dots k\}, y' \neq y$

$$\left(\boldsymbol{w}_y^T - \boldsymbol{w}_{y'}^T\right) \cdot \boldsymbol{x} \geq 0 \quad (**)$$

- Equivalently, in the nk-dimensional space
- Predict y iff $\forall\ y' \in \{1, \dots k\}, y' \neq y$

$$\boldsymbol{w}^T \left(\boldsymbol{x}_y - \boldsymbol{x}_{y'}\right) \equiv \boldsymbol{w}^T\ \boldsymbol{x}_{yy'} \geq 0$$



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

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

Constraint Classification

- Training:
  - We first explain via Kesler's construction; then show we don't need it
  - Given a data set $\{(\mathbf{x}, y)\}$, (m examples) with $\mathbf{x} \in \mathbf{R}^n, y \in \{1,2, \ldots k\}$

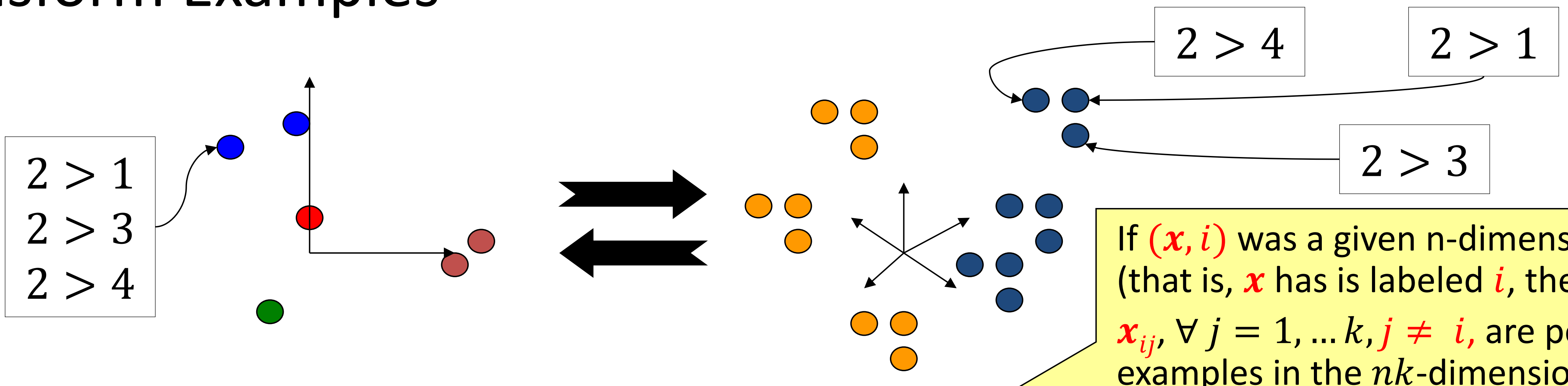
    create a binary classification task (in $\mathbf{R}^{kn}$):

    $(\mathbf{x}_y - \mathbf{x}_{y'}, +), (\mathbf{x}_{y'} - \mathbf{x}_y, -)$, for all $y' \neq y$ $[2m(k-1)$ examples]

    Here $\boldsymbol{x}_y \in \boldsymbol{R}^{kn}$
  - Use your favorite linear learning algorithm to train a binary classifier.
- Prediction:
  - Given an nk dimensional weight vector $\boldsymbol{w}$ and a new example $\boldsymbol{x}$, predict: $argmax_y \, \boldsymbol{w}^T \, \boldsymbol{x}_y$

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 $x_{ij}, \forall\, j = 1, \ldots k, j \neq i,$ are positive examples in the nk-dimensional space. $-x_{ij}$ are negative examples.

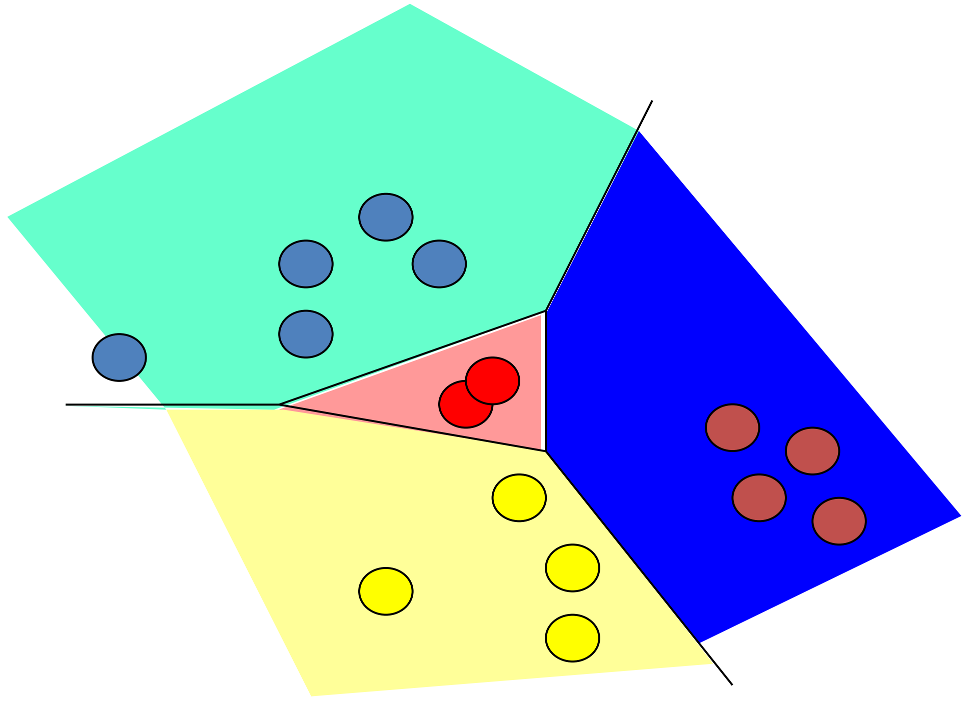
$$\begin{array}{lll} i > j & f_i(\boldsymbol{x}) - f_j(\boldsymbol{x}) & > 0 \\ & \boldsymbol{w}_i \cdot \boldsymbol{x} - \boldsymbol{w}_j \cdot \boldsymbol{x} & > 0 \\ & \boldsymbol{W} \cdot \boldsymbol{X}_i - \boldsymbol{W} \cdot \boldsymbol{X}_j & > 0 \\ & \boldsymbol{W} \cdot (\boldsymbol{X}_i - \boldsymbol{X}_j) & > 0 \\ & \boldsymbol{W} \cdot \boldsymbol{X}_{ij} & > 0 \end{array}$$

$$\begin{aligned} \boldsymbol{X}_i &= (\boldsymbol{0}, \boldsymbol{x}, \boldsymbol{0}, \boldsymbol{0}) \in \boldsymbol{R}^{kd} \\ \boldsymbol{X}_j &= (\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{x}) \in \boldsymbol{R}^{kd} \\ \boldsymbol{X}_{ij} &= \boldsymbol{X}_i - \boldsymbol{X}_j = (\boldsymbol{0}, \boldsymbol{x}, \boldsymbol{0}, -\boldsymbol{x}) \\ \boldsymbol{W} &= (\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3, \boldsymbol{w}_4) \in \boldsymbol{R}^{kd} \end{aligned}$$

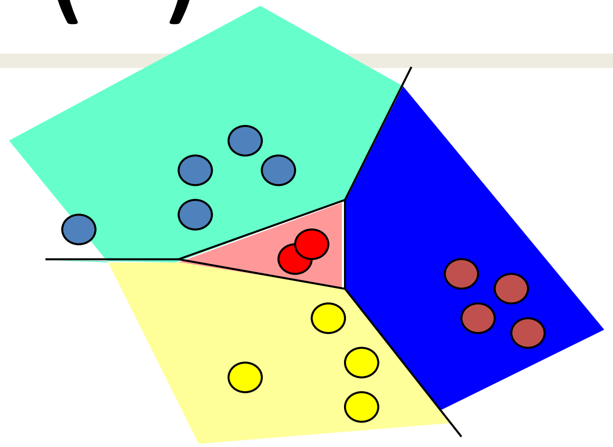
Kesler's Construction (1)

- $y = argmax_{i=(r,b,g,y)}\ \boldsymbol{w}_i . \boldsymbol{x}$
  - $\boldsymbol{w}_i, \boldsymbol{x} \in \boldsymbol{R}^n$
- Find $\boldsymbol{w}_r, \boldsymbol{w}_b, \boldsymbol{w}_g, \boldsymbol{w}_y \in \boldsymbol{R}^n$ such that
  - $\boldsymbol{w}_r . \boldsymbol{x} > \boldsymbol{w}_b . \boldsymbol{x}$
  - $\boldsymbol{w}_r . \boldsymbol{x} > \boldsymbol{w}_g . \boldsymbol{x}$
  - $\boldsymbol{w}_r . \boldsymbol{x} > \boldsymbol{w}_y . \boldsymbol{x}$

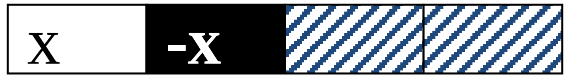
$\boldsymbol{H} = \boldsymbol{R}^{kn}$

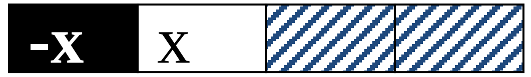


Kesler's Construction (2)



- Let $\boldsymbol{w} \;=\; (\boldsymbol{w}_{\color{red}r}, \boldsymbol{w}_{\color{blue}b}, \boldsymbol{w}_{\color{green}g}, \boldsymbol{w}_{\color{orange}y}) \in \boldsymbol{R}^{kn}$
- Let $\mathbf{0}^n$, be the n-dim zero vector



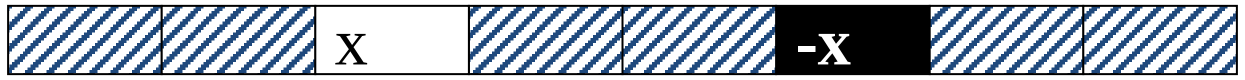


- $\boldsymbol{w}_{\color{red}r}.\boldsymbol{x} \;>\; \boldsymbol{w}_{\color{blue}b}.\boldsymbol{x} \Leftrightarrow \boldsymbol{w}.(x, -x, \mathbf{0}^n, \mathbf{0}^n) \;>\; 0 \Leftrightarrow \boldsymbol{w}.(-x, x, \mathbf{0}^n, \mathbf{0}^n) \;<\; 0$
- $\boldsymbol{w}_{\color{red}r}.\boldsymbol{x} \;>\; \boldsymbol{w}_{\color{green}g}.\boldsymbol{x} \Leftrightarrow \boldsymbol{w}.(x, \mathbf{0}^n, -x, \mathbf{0}^n) \;>\; 0 \Leftrightarrow \boldsymbol{w}.(-x, \mathbf{0}^n, x, \mathbf{0}^n) \;<\; 0$
- $\boldsymbol{w}_{\color{red}r}.\boldsymbol{x} \;>\; \boldsymbol{w}_{\color{orange}y}.\boldsymbol{x} \Leftrightarrow \boldsymbol{w}.\cancel{(x, \mathbf{0}^n, \mathbf{0}^n, -x)} \;>\; 0 \Leftrightarrow \boldsymbol{w}.(-x\underbrace{, \mathbf{0}^n, \mathbf{0}^n, x) <}\; 0$

$+$ $\qquad\qquad$ $-$

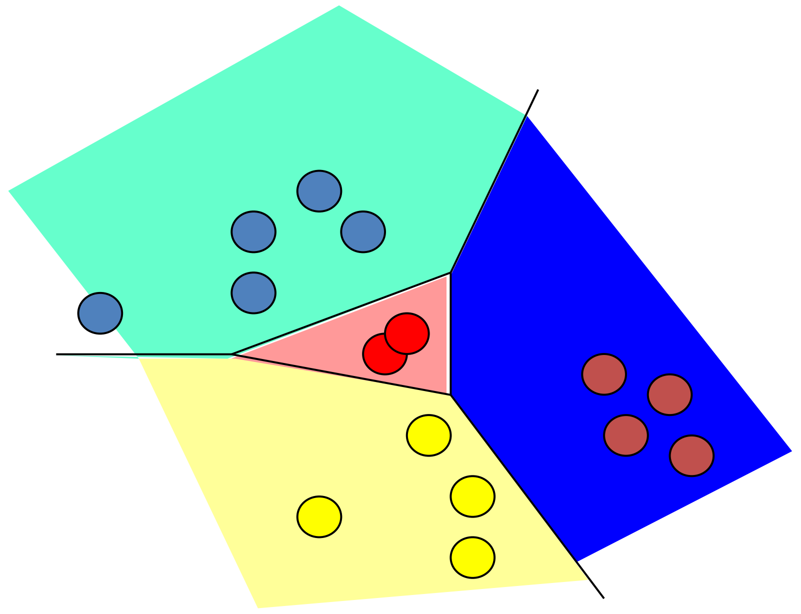
Kesler's Construction (3)

- Let
  - $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k) \in \boldsymbol{R}^n \times \cdots \times \boldsymbol{R}^n = \boldsymbol{R}^{kn}$
  - $\boldsymbol{x}_{ij} = \left(\boldsymbol{0}^{(i-1)n}, \boldsymbol{x}, \boldsymbol{0}^{(k-i)n}\right) - \left(\boldsymbol{0}^{(j-1)n}, -\boldsymbol{x}, \boldsymbol{0}^{(k-j)n}\right) \in \boldsymbol{R}^{kn}$

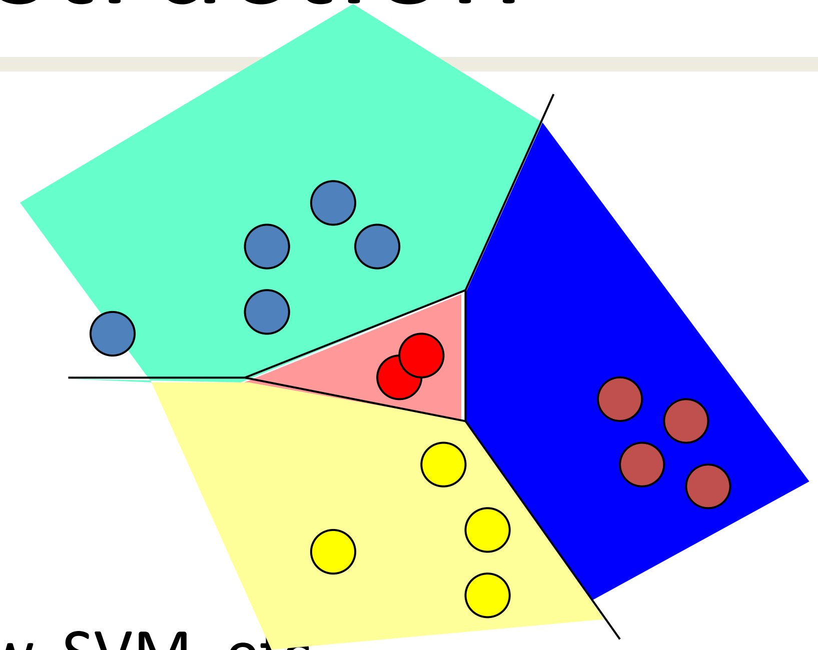


- Given $(\boldsymbol{x}, y) \in R^n \times \{1, \ldots, k\}$
  - For all $j \neq y$ (all other labels)
    - Add to $\boldsymbol{P}^+(x, y), (\boldsymbol{x}_{yj}, 1)$
    - Add to $\boldsymbol{P}^-(x, y), (-\boldsymbol{x}_{yj}, -1)$

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



Learning via Kesler's Construction



- Given $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N) \in \boldsymbol{R}^n \times \{1, \dots, k\}$
- Create
  - $\boldsymbol{P}^+ = \cup \, \boldsymbol{P}^+(\mathbf{x}_i, y_i)$
  - $\boldsymbol{P}^- = \cup \, \boldsymbol{P}^-(\mathbf{x}_i, y_i)$
- Find $\boldsymbol{w} = (\mathbf{w}_1, \dots, \mathbf{w}_k) \in \boldsymbol{R}^{kn}$, such that
  - $\boldsymbol{w}.\boldsymbol{x}$ separates $\boldsymbol{P}^+$ from $\boldsymbol{P}^-$
- 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

  $\mathbf{w}^T \ \mathbf{x}_{yy'} \ \geq \ 0$ (in the nk-dimensional space)
- is equivalent to:

  $(\mathbf{w}_y^T - \mathbf{w}_{y'}^{\mathrm{T}}) \, \mathbf{x} \ \geq \ 0$ (in the n-dimensional space)

Perceptron in Kesler Construction

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

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

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

- Note that we promote $\boldsymbol{w}_i$ and demote $k-1$ weight vectors $\boldsymbol{w}_j$

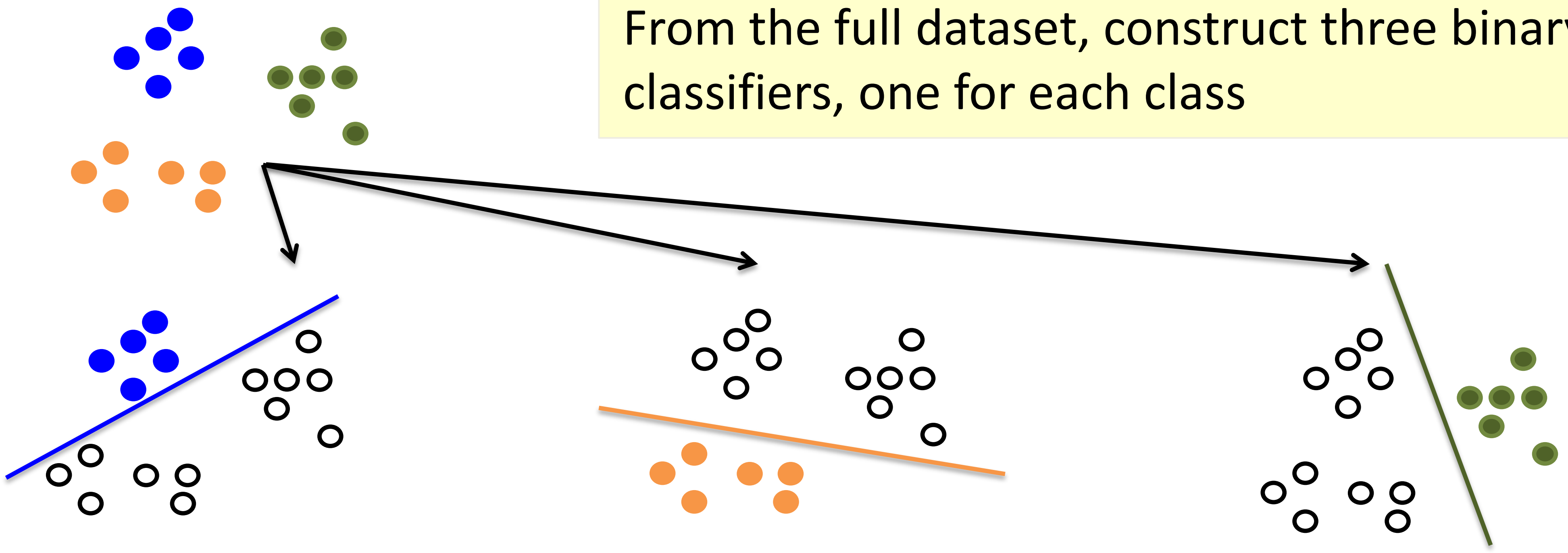
Conservative update

- The general scheme suggests:
- Given example $(\boldsymbol{x}, i)$ (example $\boldsymbol{x} \in \boldsymbol{R}^n$, labeled i)
  - $\forall (i,j), i,j = 1, \dots k,\ i \neq j$ (***)
  - If $(\boldsymbol{w}_i^T - \boldsymbol{w}_j^T)\, \boldsymbol{x} < 0$ (mistaken prediction; equivalent to $\boldsymbol{w}^T \boldsymbol{x}_{ij} < 0$)
  - $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \boldsymbol{x}$ (promotion) and $\boldsymbol{w}_j \leftarrow \boldsymbol{w}_j - \boldsymbol{x}$ (demotion)
- Promote $\boldsymbol{w}_i$ and demote $k-1$ weight vectors $\boldsymbol{w}_j$
- 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: $j^* = argmax_{j=1,\dots,k}\, \boldsymbol{w}_j^T \boldsymbol{x}$ (highest activation among competing labels)
  - If $(\boldsymbol{w}_i^T - \boldsymbol{w}_{j^*}^T)\, \boldsymbol{x} < 0$ (mistaken prediction)
    - $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \boldsymbol{x}$ (promotion) and $\boldsymbol{w}_{j^*} \leftarrow \boldsymbol{w}_{j^*} - \boldsymbol{x}$ (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



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

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

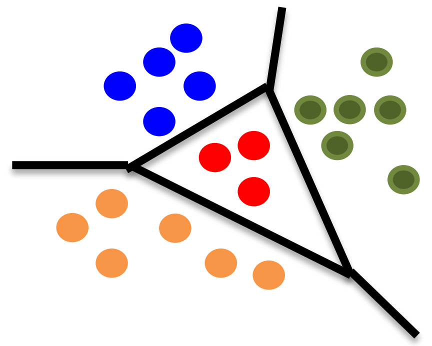
$\mathbf{w}_{black}{}^T\boldsymbol{x} > 0$ for black inputs

Notation: Score for blue label

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

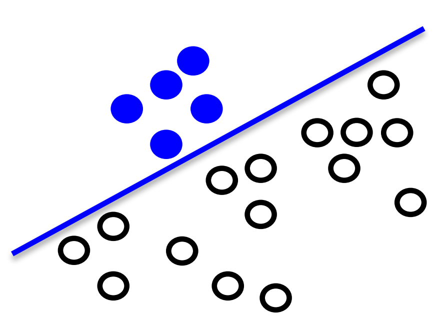
Multiclass Classification Summary 2:

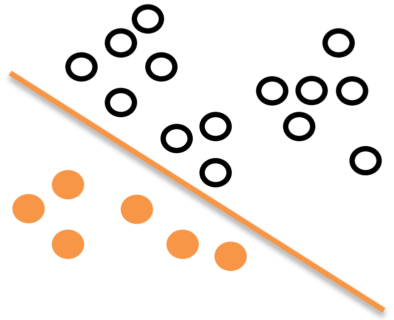
One-vs-all may not always work

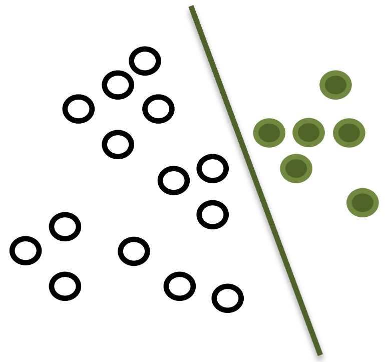


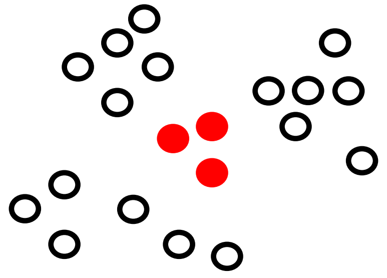
Red points are not separable with a single binary classifier

The decomposition is not expressive enough!









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

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

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

???

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: $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K$;
  - Predict with WTA: $argmax_i\ \boldsymbol{w}_i^T\boldsymbol{x}$
  - 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 $(\boldsymbol{x}_i, \boldsymbol{y}_i)$:

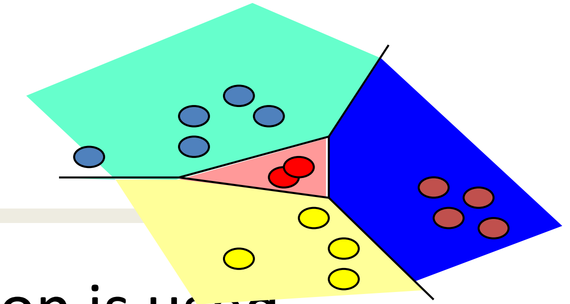
$$\hat{y} \leftarrow arg\max_j \boldsymbol{w}_j^T\phi(\boldsymbol{x}_i, y_i)$$

if $\hat{y} \neq y_i$

- $\boldsymbol{w}_{y_i} \leftarrow \boldsymbol{w}_{y_i} + \eta\boldsymbol{x}_i$ (promote)
- $\boldsymbol{w}_{\hat{y}} \leftarrow \boldsymbol{w}_{\hat{y}} - \eta\boldsymbol{x}_i$ (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 $\mathbf{x}_{ij} \in \mathbf{R}^{nk}$,

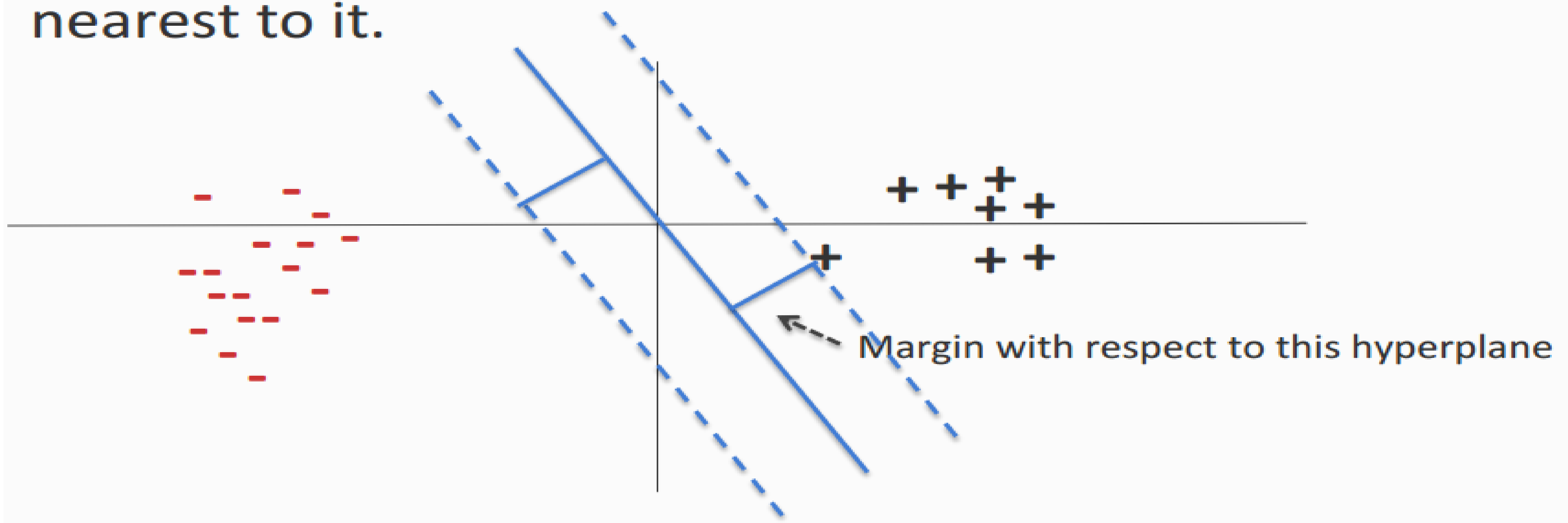
$$margin(\mathbf{x}_{ij}, \mathbf{w}) = \min_{ij} \boldsymbol{w}^T \boldsymbol{x}_{ij}$$

  - Consequently, given $\mathbf{x} \in \mathbf{R}^n$, labeled i

$$margin(\mathbf{x}, \mathbf{w}) = \min_{j} (\boldsymbol{w}_i^T - \boldsymbol{w}_j^T) \boldsymbol{x}$$

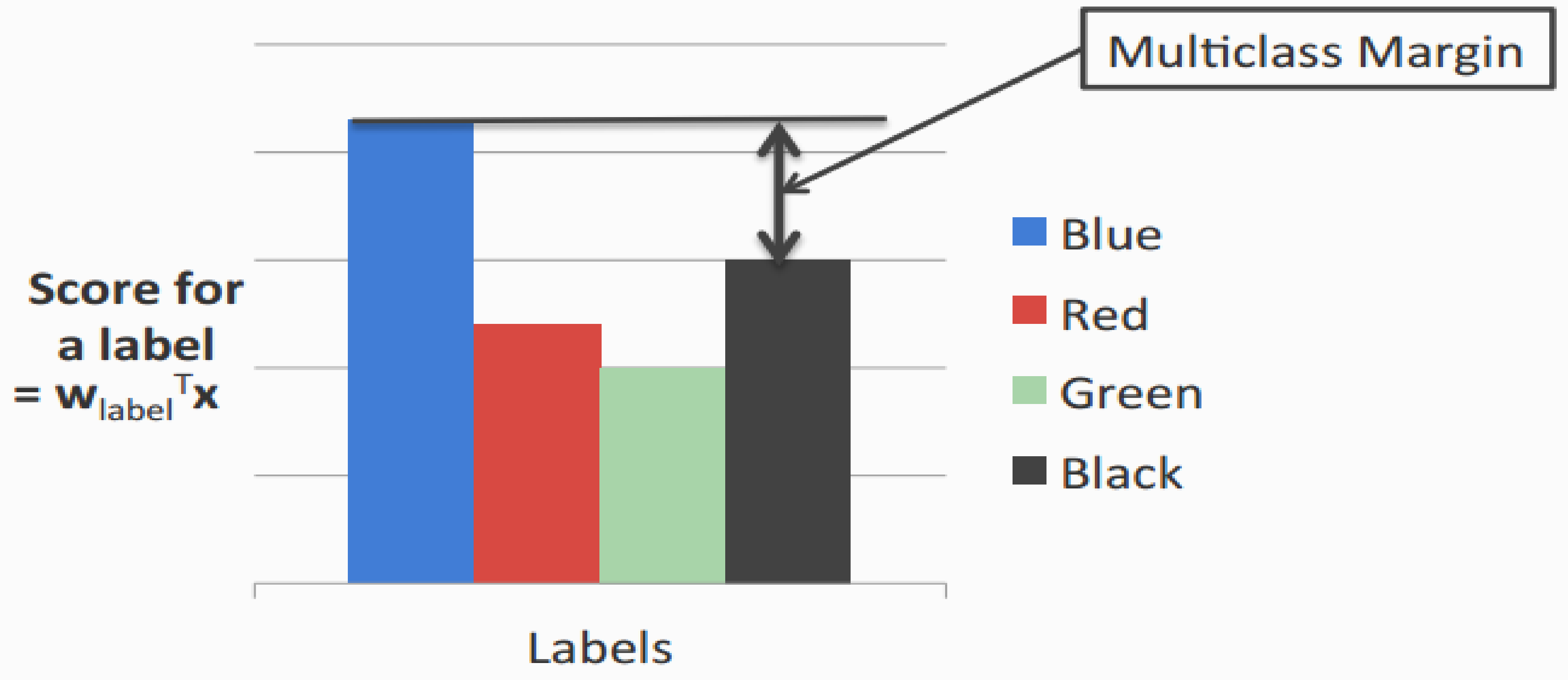
Margin

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
  - $(x, (5 > 4 > 3 > 2 > 1))$ $\Rightarrow (x, ((5 > 4, 4 > 3, 3 > 2, 2 > 1))$

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

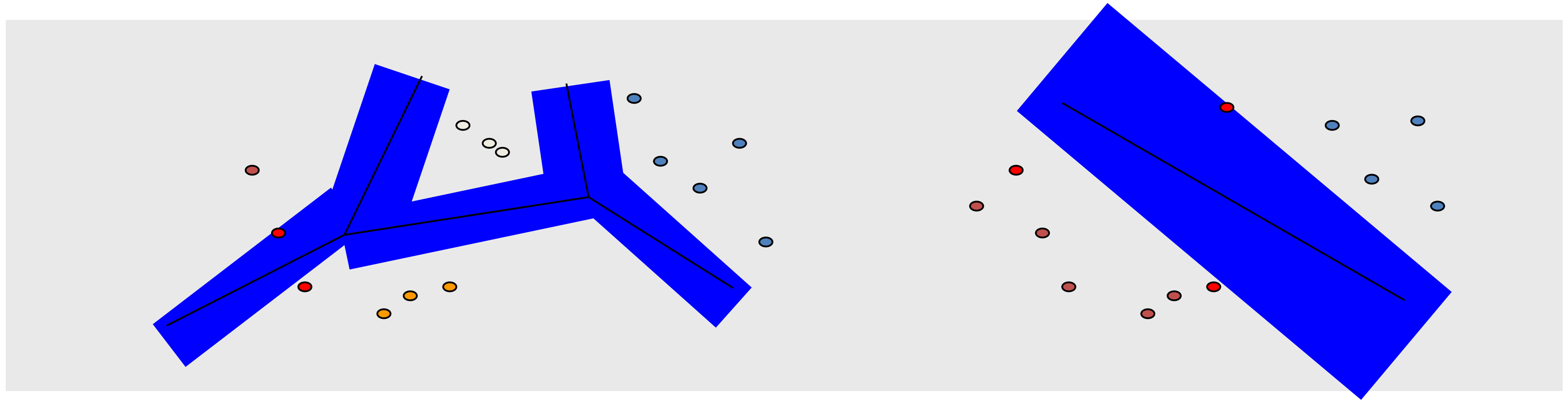
Just like in the multiclass we learn one $\boldsymbol{w}_i \in \boldsymbol{R}^n$ for each label, the same is done for multi-label and ranking. The weight vectors are updated according with the requirements from

$$y \in S_k$$

(Consult the Perceptron in Kesler construction slide)

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

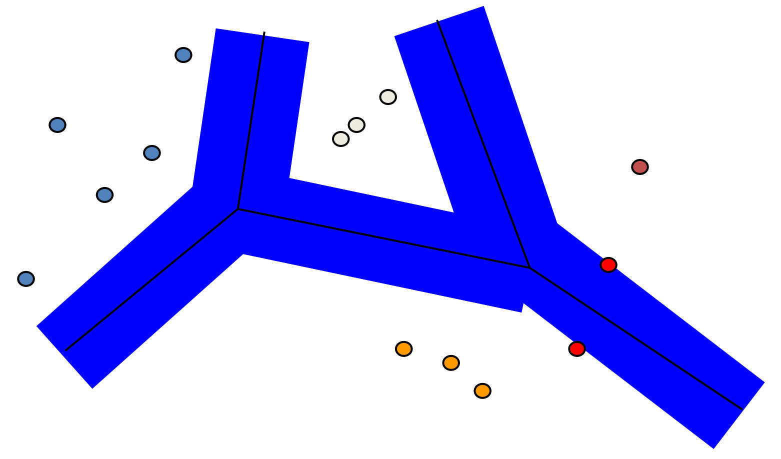
- Can learn any $argmax\ \boldsymbol{v}_i \cdot \boldsymbol{x}$ function (even when i isn't linearly separable from the union of the others)
- Can use any algorithm to find linear separation
  - Perceptron Algorithm
    - ultraconservative online algorithm [Crammer, Singer 2001]
  - Winnow Algorithm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



Margin Generalization Bounds

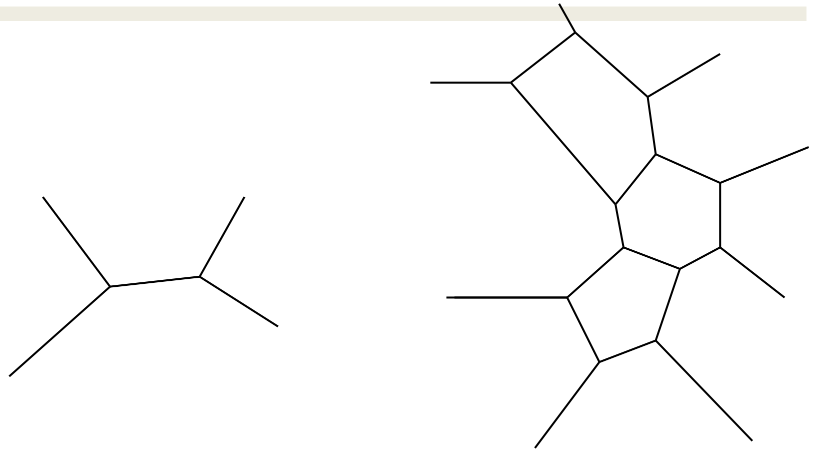
- Linear Hypothesis space:
  - $h(\boldsymbol{x}) = \ argsort\ \boldsymbol{v}_i \cdot \boldsymbol{x}$
    - $\boldsymbol{v}_i, \boldsymbol{x} \in \boldsymbol{R}^d$
    - $argsort$ returns permutation of $\{1, \dots, k\}$
- CC margin-based bound
  - $\gamma = \min_{(x,y) \in \boldsymbol{S}} \min_{(i<j) \in y} \boldsymbol{v}_i \cdot \boldsymbol{x} - \boldsymbol{v}_j \cdot \boldsymbol{x}$
  - $err_D(h) \leq \Theta\left(\frac{C}{m}\left(\frac{R^2}{\gamma^2} - \ln(\delta)\right)\right)$

- m - number of examples
- R - $max_x\ ||x||$
- δ - confidence
- C - average # constraints



VC-style Generalization Bounds

- Linear Hypothesis space:
  - $h(\mathbf{x}) = \ argsort\ \mathbf{v}_i \cdot \mathbf{x}$
    - $\mathbf{v}_i, \mathbf{x} \in \boldsymbol{R}^d$
    - $argsort$ returns permutation of $\{1, \dots, k\}$



- CC VC-based bound

$$err_D(h) \le err(S,h) + \theta \left\{ \frac{\left(kd log\left(\frac{mk}{d}\right) - ln\delta\right)}{m} \right\}^{\frac{1}{2}}$$

- m - number of examples
- d - dimension of input space
- δ - 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
  - $\boldsymbol{x}$ is both red and blue
- Complex relationships
  - $\boldsymbol{x}$ 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 ($y \in \{1, \ldots, K\}^+$)
  - e.g. POS tagging ('(NVNNA)')
    - "This is a sentence." $\Rightarrow$ D V D N
  - e.g. phrase identification
  - Many labels: K^L for length L sentence
- Structured Output Prediction ($y \in C(\{1, \ldots, K\}^+)$)
  - e.g. parse tree, multi-level phrase identification
  - e.g. sequential prediction
  - Constrained by:
    - domain, problem, data, background knowledge, etc...

Usage Notes

- A lot of slides are adopted from the presentations and documents published on internet by experts who know the subject very well.
- I would like to thank who prepared slides and documents.
- Also, these slides are made publicly available on the web for anyone to use
- If you choose to use them, I ask that you alert me of any mistakes which were made and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides.

Sincerely,

Dr. Cahit Karakuş

cahitkarakus@gmail.com