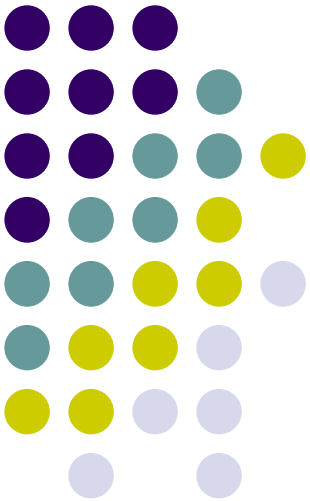
Ensemble Learning

J. Savoy
University of Neuchâtel



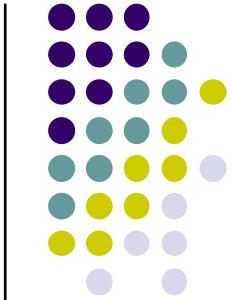
Ian H. Witten, Eibe Frank: *Data Mining. Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

T. Hastie, R. Tibshirani, J. Friedman: *The Elements of Statistical Learning*. Springer, New York, 2009.

Combining Methods

- Each method has its own advantages and drawbacks
- The same method with different training sets may produce different answers.

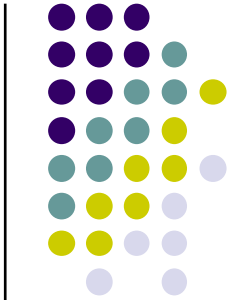
- Can we build different “experts” and let them vote
  - Pro: may improve the accuracy rate
  - Cons: output is hard to analyze / understand
- Ensemble learning
  - Bagging
  - Randomization
  - Boosting
  - …



Overview

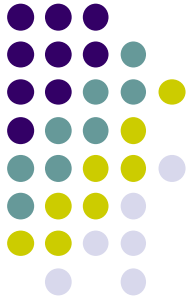
- **Bagging**
- Randomization
- Boosting

Bagging



- Combining predictions by voting/averaging
  - Simplest way
  - Each model receives equal weight
- “Idealized” version:
  - Sample several training sets of size *n* (instead of just having one training set of size n)
  - Build a classifier for each training set
  - Combine the classifiers’ predictions
- OK but we have only one training set of size *n*!

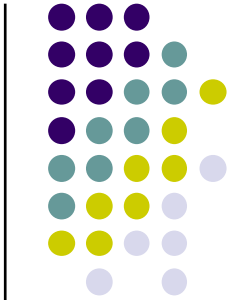
Bagging



- Bagging
  - bagging = bootstrap aggregation
  - usually apply with decision tree approach
  - generate B (e.g., B=100) training set of size *n* by drawing randomly with replacement
- Learning scheme is unstable

  almost always improves performance
  - (sometimes) Small change in training data can make big change
  - Because decision tree has a high variance
  - What is a variance in this context?

Bias - Variance Decomposition



Total expected error ≈ irreducible error + bias + variance

The Mean Squared Error (MSE) is a function of these three components.

$$MSE = E\left[f(x_0) - y_0\right]^2$$

We cannot reduce the irreducible error.

Decreasing the bias means increasing the variance and the reverse is also true.

What is the variance? The bias?

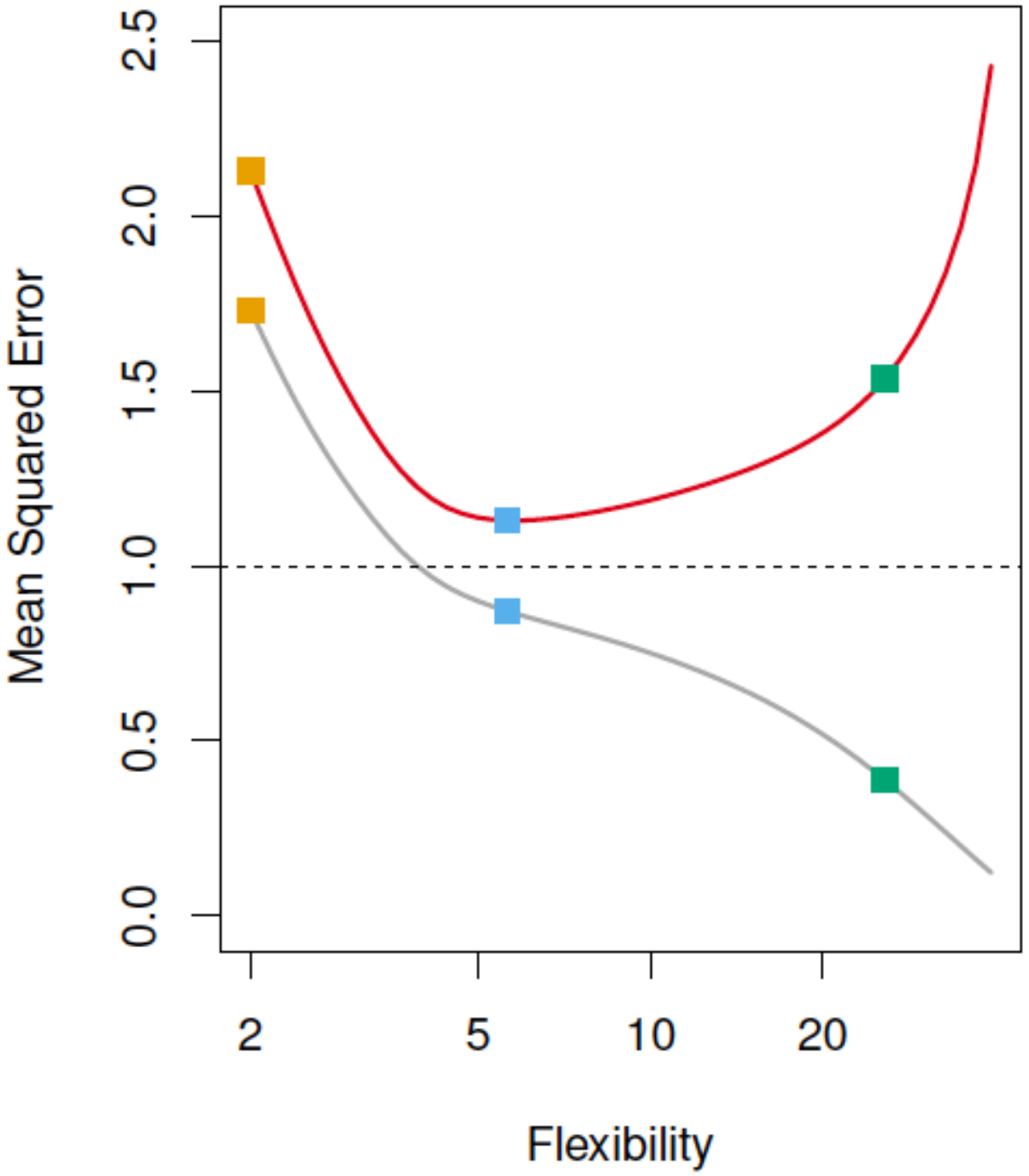
Bias - Varia

The error rate MSE

In grey in the training set with increasing complexity (df). More flexible, less error in the training set.

In red MSE in the test set.

dotted: min over all classifiers

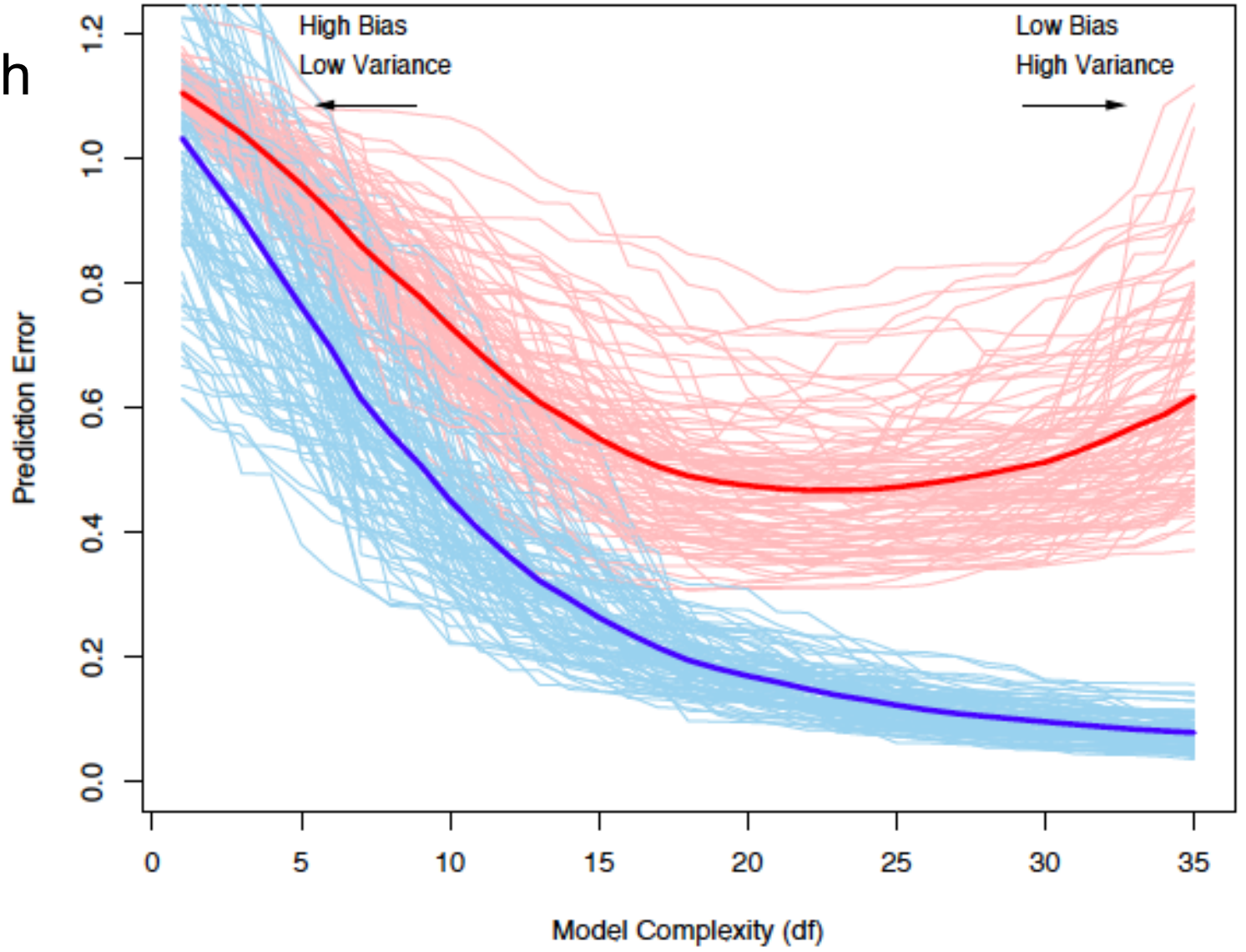


Bias - Variance

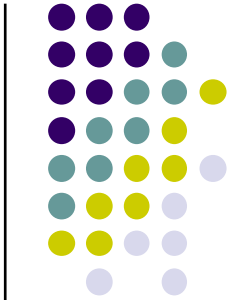
In general with more than a few observations.

Blue error estimation in the training set.

In red in the test set.



Bias - Variance Decomposition



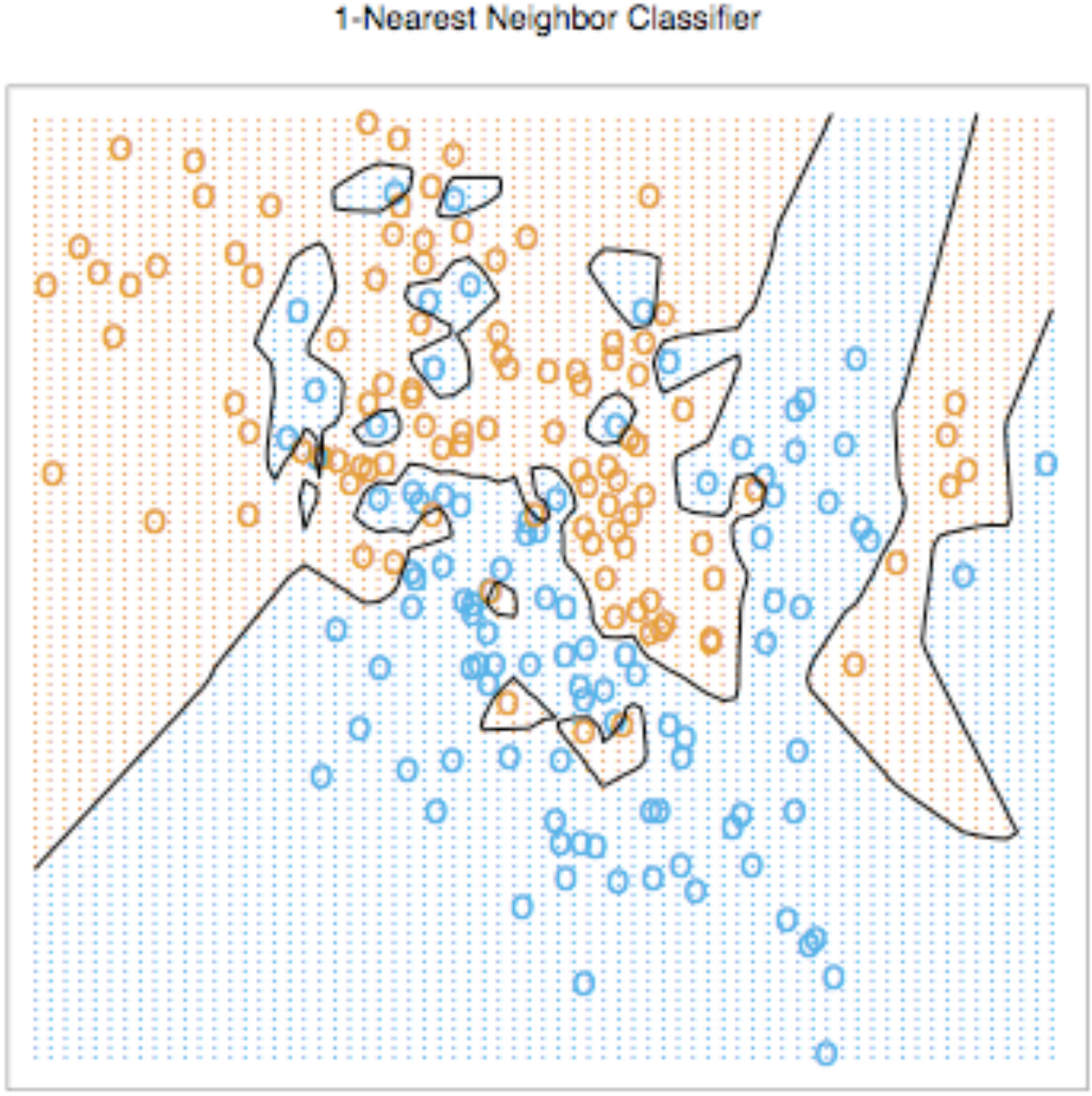
- Variance: variability due to the training set used. Changing the training set, we have another classifier, and another performance measure.
- Bias: error due to approximation of the real model by our classifier

- Example: *k*-NN
  if $k = 1$, a small variation is the data will generate a different classifier;
  if $k = 15$, a small variation is the data will generate a similar classifier
  This explains the fact that $df = 1/k$

Bias – Variance Decomposition

Binary Response

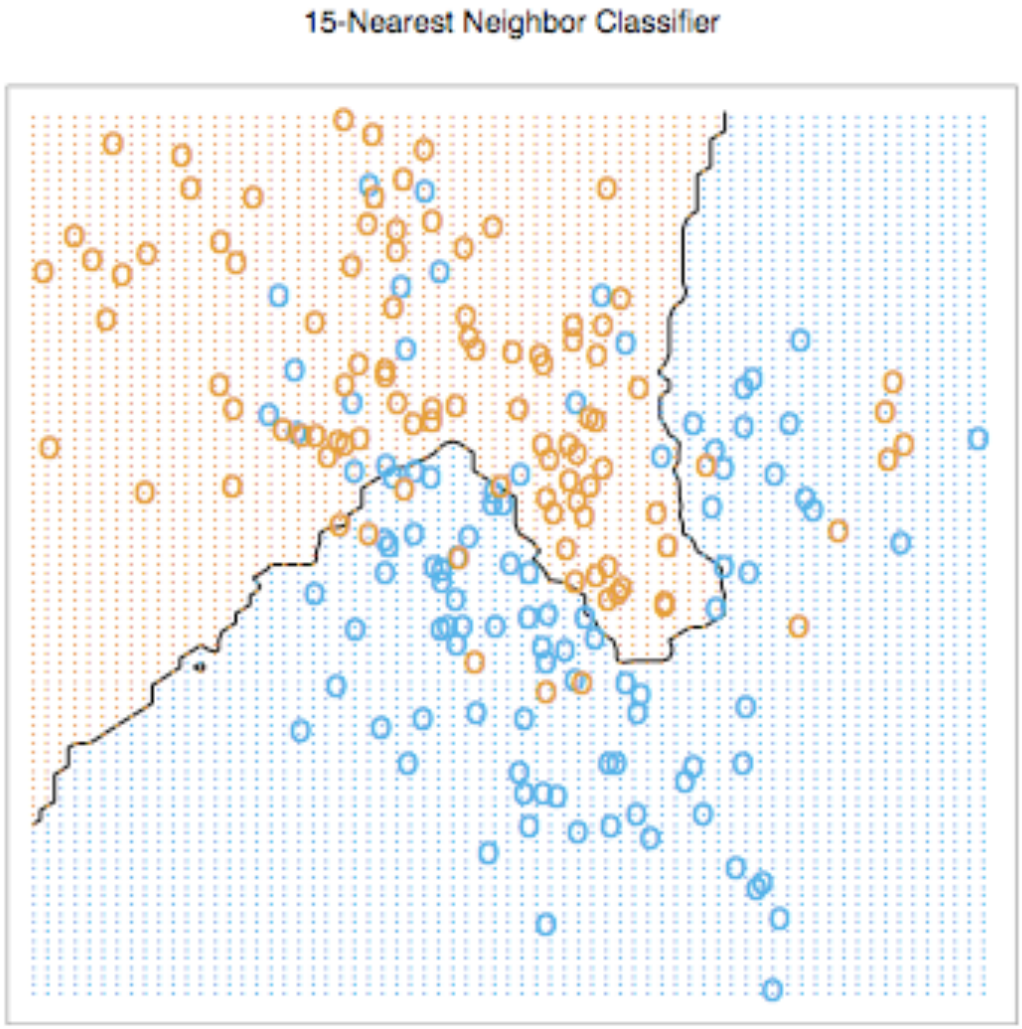
1-Nearest Neighbor



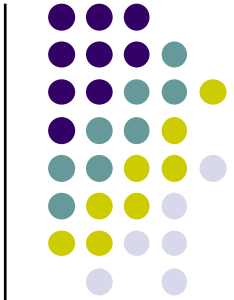
Bias – Variance Decomposition

Binary Response

15-Nearest Neighbors



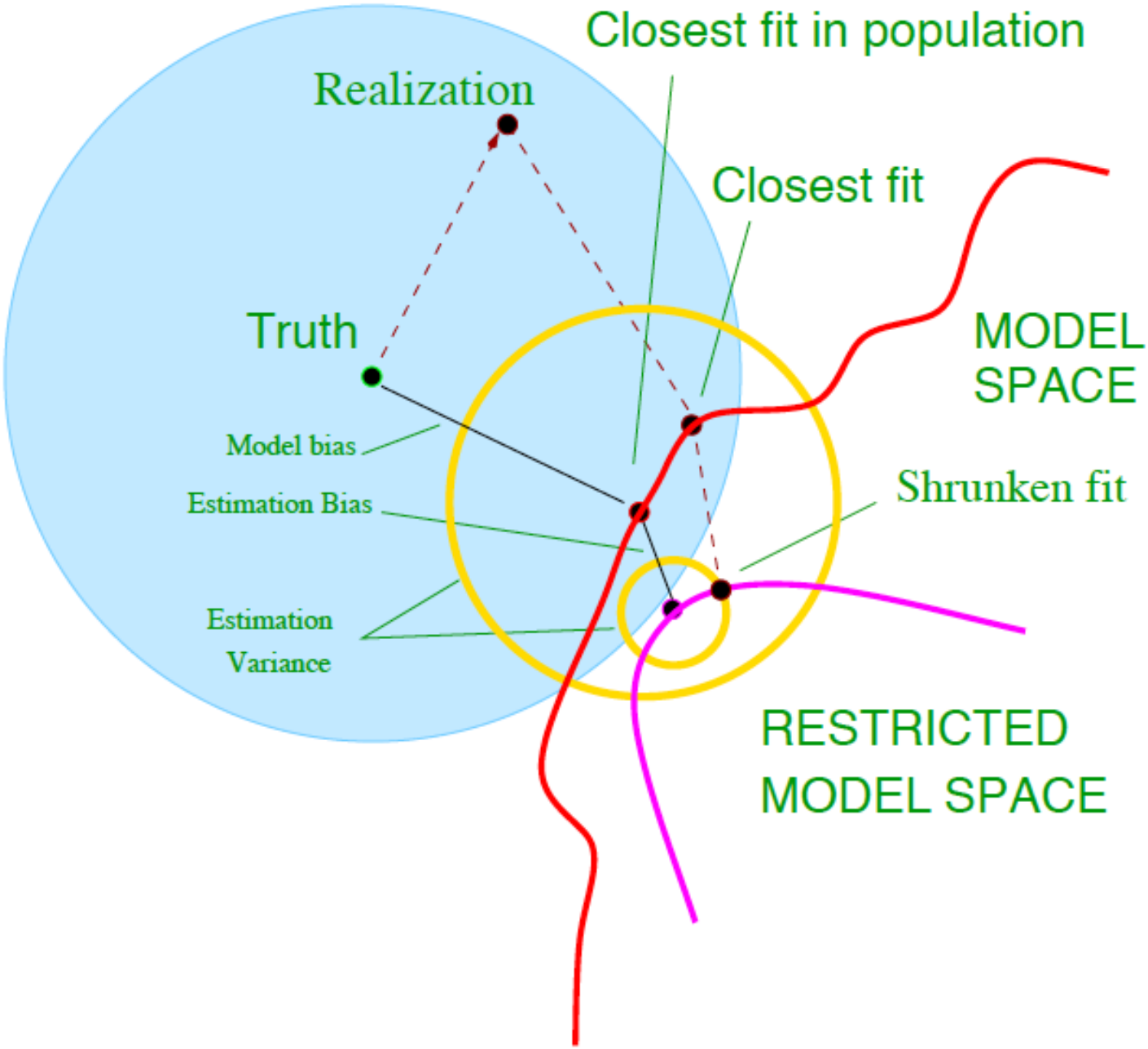
Bias - Variance Decomposition



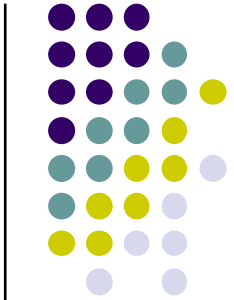
- When analyzing the error rate of a classifier, we can find:

1. an irreducible error representing the underlying variability of the system (we cannot reduce this part).
2. bias: the amount of difference between our model and the true real model (mean). The more complex the classifier, the lower this bias.
3. variance: the expected deviation around the mean. Variation coming from changing the training set

- Total expected error ≈ (irreducible error) + bias + variance

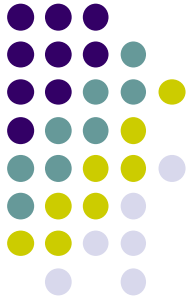

Closest fit in population
Realization
Closest fit
Truth
MODEL
SPACE
Model bias
Estimation Bias
Shrunken fit
Estimation
Variance
RESTRICTED
MODEL SPACE

Bias - Variance Decomposition



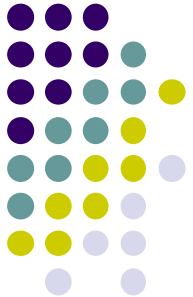
- Used to analyze how much selection of any *specific* training set affects performance
- Assume infinitely many classifiers, built from different training sets of size *n*
- For any learning scheme,
  - Bias = expected error of the combined classifier on new data (mainly a function of the classifier model selected).
  - Variance = expected error due to the particular training set used.

Bagging



- Bagging works because it reduces variance by voting / averaging
  - Note: in some pathological hypothetical situations the overall error might increase
  - Usually, the more classifiers the better
- Problem: we only have one dataset!
- Solution: generate new ones of size *n* by sampling from it *with replacement*
- Can help if data is noisy
- Can also be applied to numeric prediction

Bagging



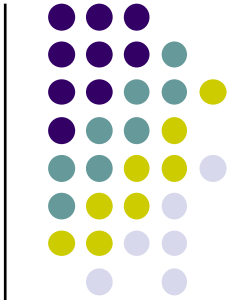
Model Generation

```
Let n be the number of instances in the training data.
For each of t iterations:
  Sample n instances with replacement from training data.
  Apply the learning algorithm to the sample.
  Store the resulting model.
```

Classification

```
For each of the t models:
  Predict class of instance using model.
Return class that has been predicted most often.
```

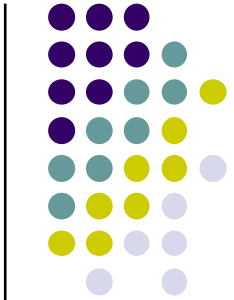
Bagging



- Bagging unpruned decision trees known to produce good probability estimates.
  - Where, instead of voting, the individual classifiers' probability estimates are averaged.
  - Note: this can also improve the success rate.

- Problem: not interpretable.
  - With a single decision tree we can interpret the decision
  - But with B=100 trees?

Combination Variants



- Generate many different classifier models (Bayesian, 1R, *k*-NN, decision tree, …)
  - Each classifier model has its own pros/cons
  - Meta-classifier usually performs better
  - But high development / update cost

- Using the same classifier model but with different text representations (words, letters, n-grams, POS, …)
  - Each representation has its own pros/cons

Example

- Verification problem (decision: yes/no)
- Overall evaluation (six datasets)
  The meta-classifier works better

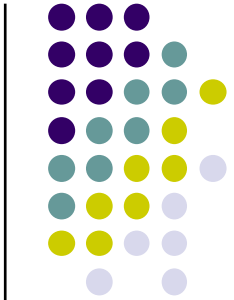
Rank	Run	Score
1	*Meta-classifier*	0.566
2	Spatium-L1	0.555
3	Khonji & Iraqi	0.490
4	Frery *et al.*	0.484
5	Castillo *et al.*	0.565

Example

- Verification problem (decision: yes/no)
- Only on Greek articles
  The meta is not the best system in all cases

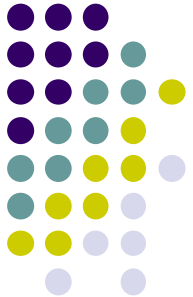
Rank	Run	Score
1	Khonji & Iraqi	0.720
2	Spatium-L1	0.666
3	*Meta-classifier*	0.635
4	Major *et al.*	0.621
5	Moreau *et al.*	0.461

Overview



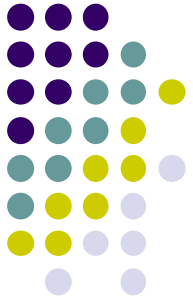
- Bagging
- **Randomization**
- Boosting

Randomization



- Can randomize learning algorithm instead of the training set
- Some algorithms already have a random component: e.g. initial weights in neural net
- Most algorithms can be randomized
  e.g., greedy algorithms:
  - Pick from the *n* best options at random instead of always picking the best option
  - E.g., attribute selection in decision trees
- More generally applicable than bagging:
  e.g. random subsets in nearest-neighbor scheme
- Can be combined with bagging

Random Forests



- Bagging creates ensembles of accurate classifiers with relatively low diversity.
  - Bootstrap sampling creates training sets with a distribution that resembles the original data.

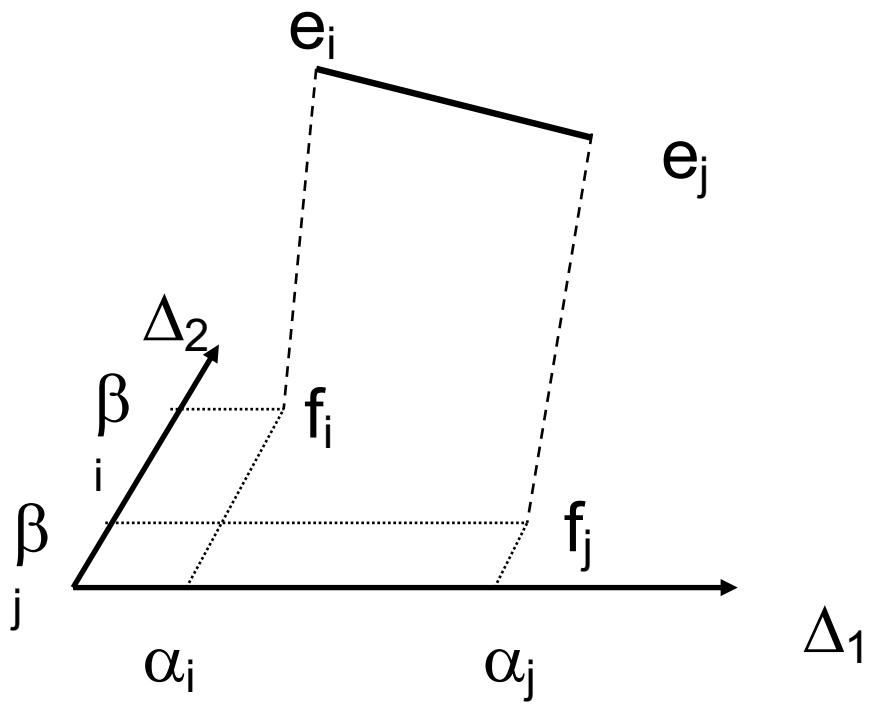
- Randomness in the learning algorithm increases diversity but sacrifices accuracy of individual ensemble members.
  From time to time, individual classifiers could present a poor performance.

Random Forests

- Combine random attribute sets, bagging and principal components (PCA) to generate an ensemble of decision trees
- An iteration involves
  - Randomly dividing the input attributes into *k* disjoint subsets
  - Applying PCA to each of the *k* subsets in turn
  - Learning a decision tree from the *k* sets of PCA directions
- Further increases in diversity can be achieved by creating a bootstrap sample in each iteration before applying PCA

Principal Component Analysis

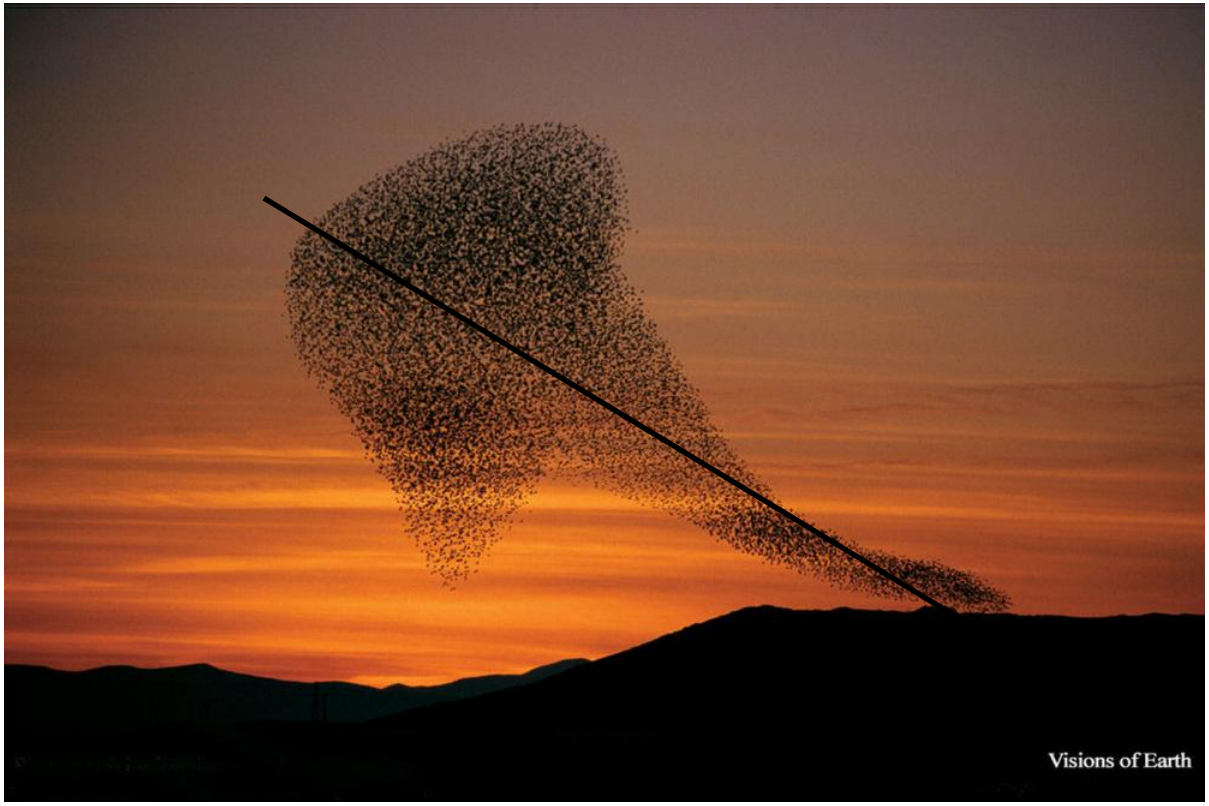
- PCA is a statistical method for arranging large arrays of data into interpretable patterning match
- "principal components" are computed by calculating the *correlations* between all the variables, then grouping them into sets that show the most correspondence



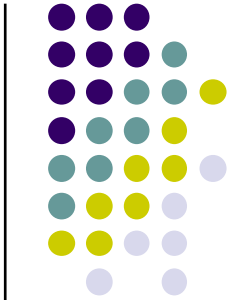
We will define a projection plane (defined by the lines Δ_1 and Δ_2, *perpendicular* (no correlation)) to represent the objects (e_i, e_j) and conserving the real distance $d(e_i, e_j)$.

PCA Representation

A cloud of birds in 3D → 2D (→ 1D)



Lexical Table (Small Example)

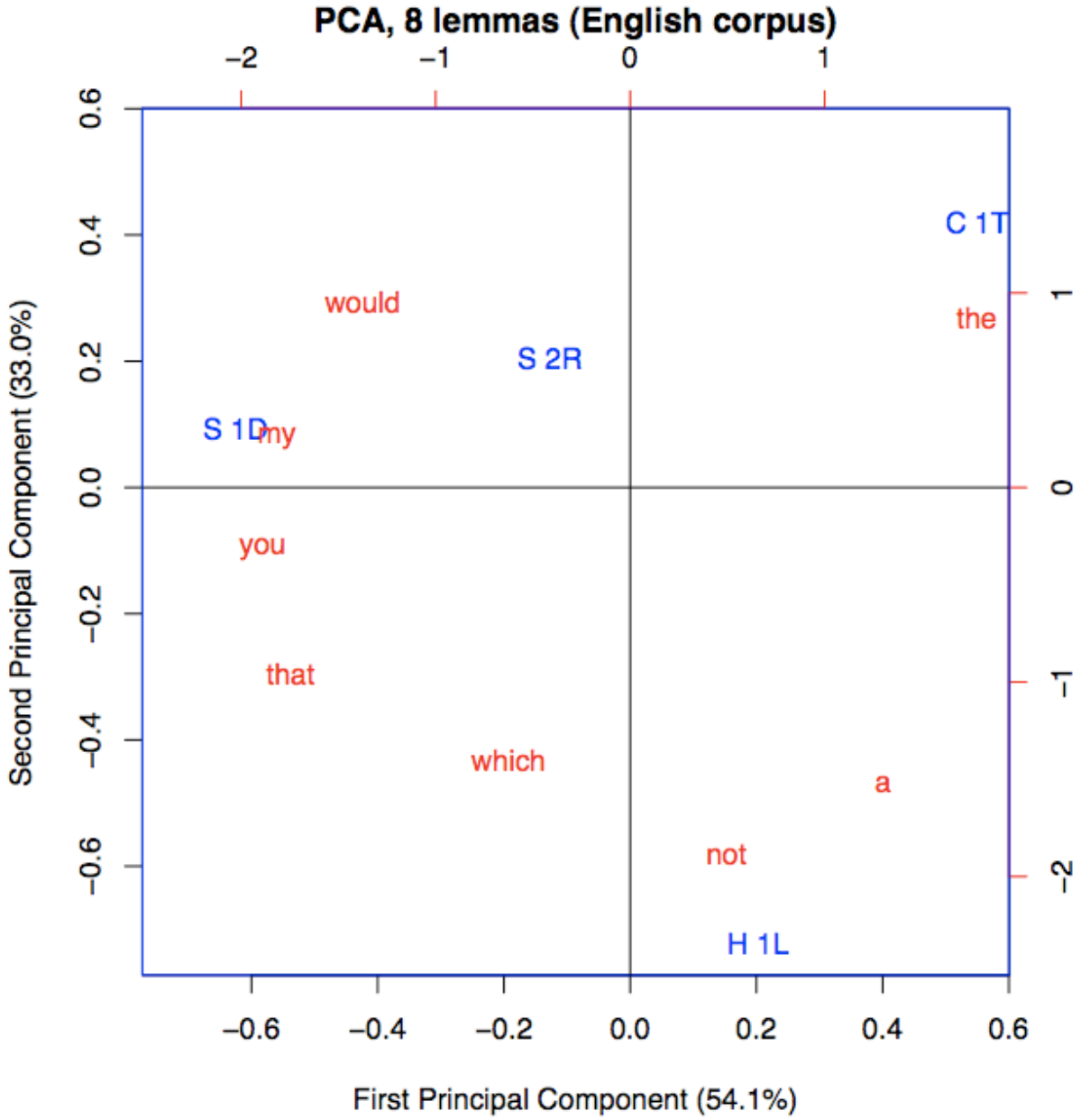


8 lemmas, English corpus,
Catriona (Stevenson), *Almayer* (Conrad), *Jude* (Hardy)

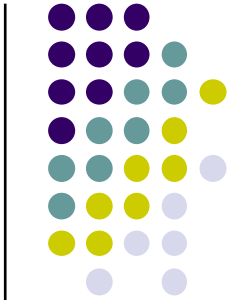
	1D *Catriona*	2R *Catriona*	1T *Almayer*	1L *Jude*
the	421	548	**838**	540
a	3	5	7	**12**
you	**280**	127	54	135
which	27	39	14	41
would	42	48	31	28
my	**18**	4	1	2
not	73	57	70	**135**
that	149	115	82	128

PCA

8 lemmas (English)
1D, 2R *Catriona* (Stevenson),
1T *Almayer's Folly* (Conrad),
1L *Jude the Obscure* (Hardy)

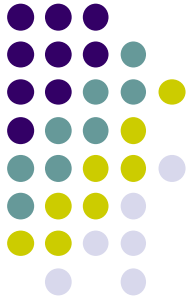


Overview



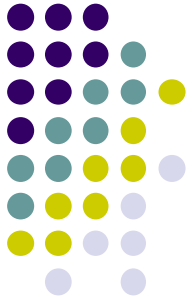
- Bagging
- Randomization
- **Boosting**

Boosting



- Idea:
  - select small subset of the training set
  - derive a simple rule of thumb for them
  - select a 2nd subset of the training set
  - derive 2nd rule of thumb for them
  - …
  - repeat this *t* times
- how to combine this *t* classifers?
- how can we select the *t* different subsets?

Boosting



- Also uses voting (classification)
  (or averaging for regression)

  but weight models according to performance
- Iterative: new models are influenced by performance of previously built ones
- View as a *sequence of classifiers* of the same type
  - Encourage new model to become an “expert” for instances misclassified by earlier models.
  - Intuitive justification: models should be experts that complement each other.
- Several variants

Boosting

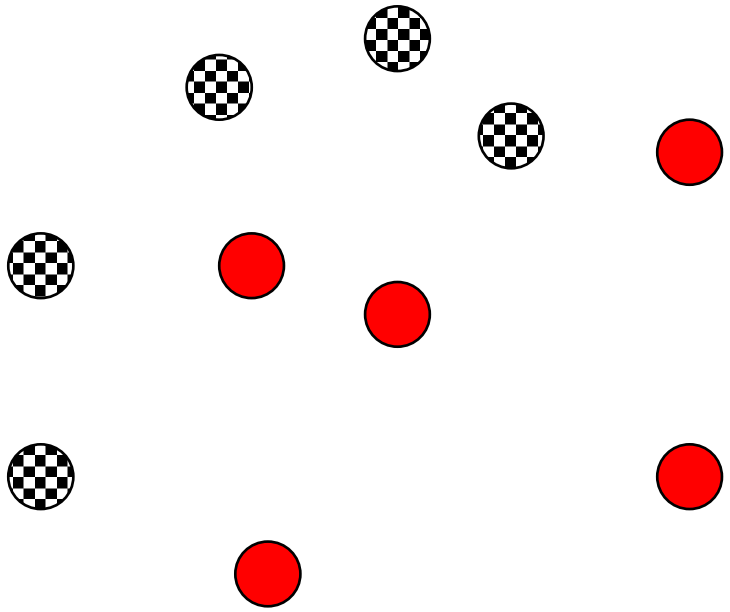
The problem:

Classification

5 red

5 BW

Use only linear classifiers



Boosting

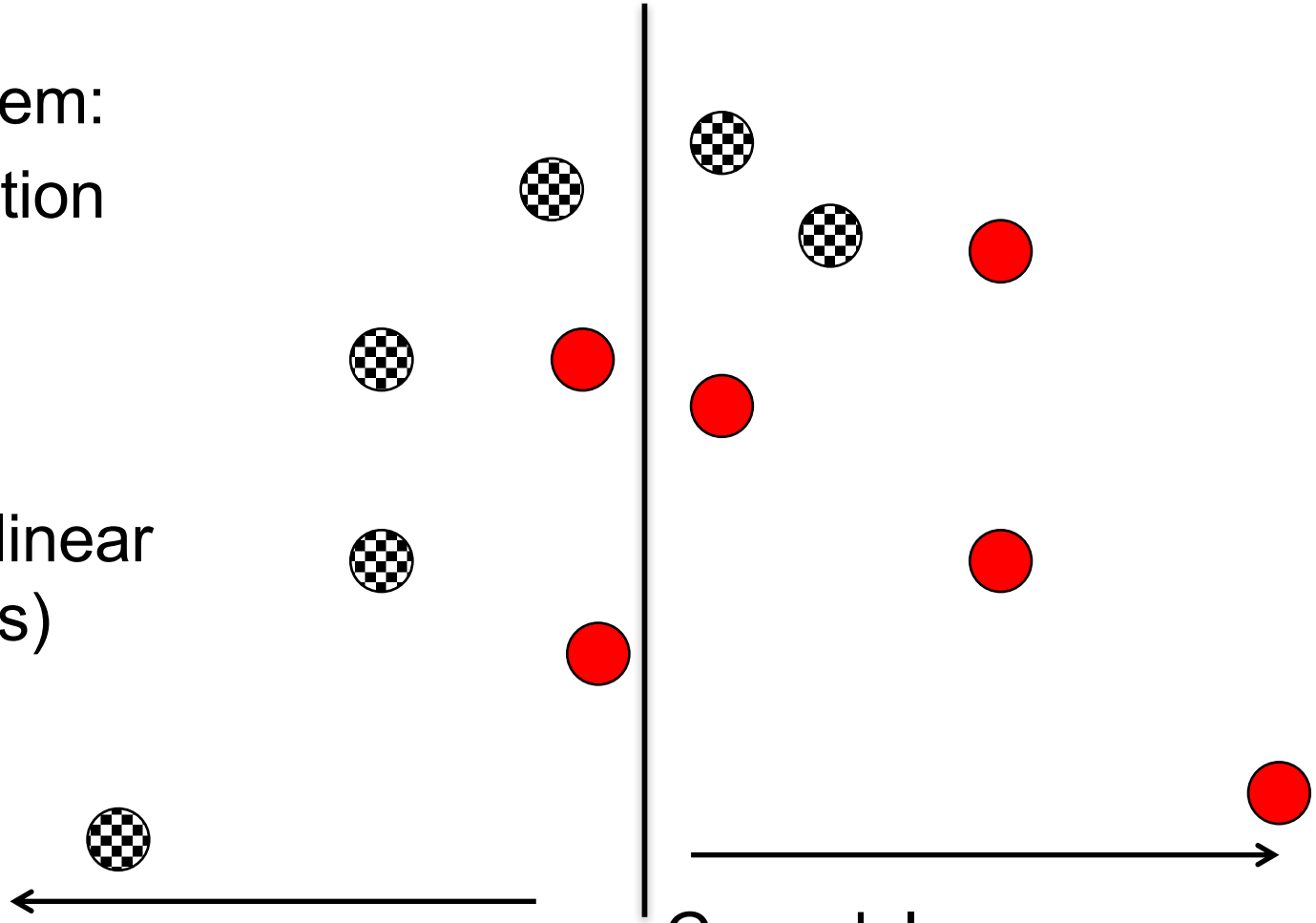
The problem:

Classification

5 red

5 BW

Use only linear classifier(s)



C_i weak learner

Error rate = 4/10 = 0.4

Boosting

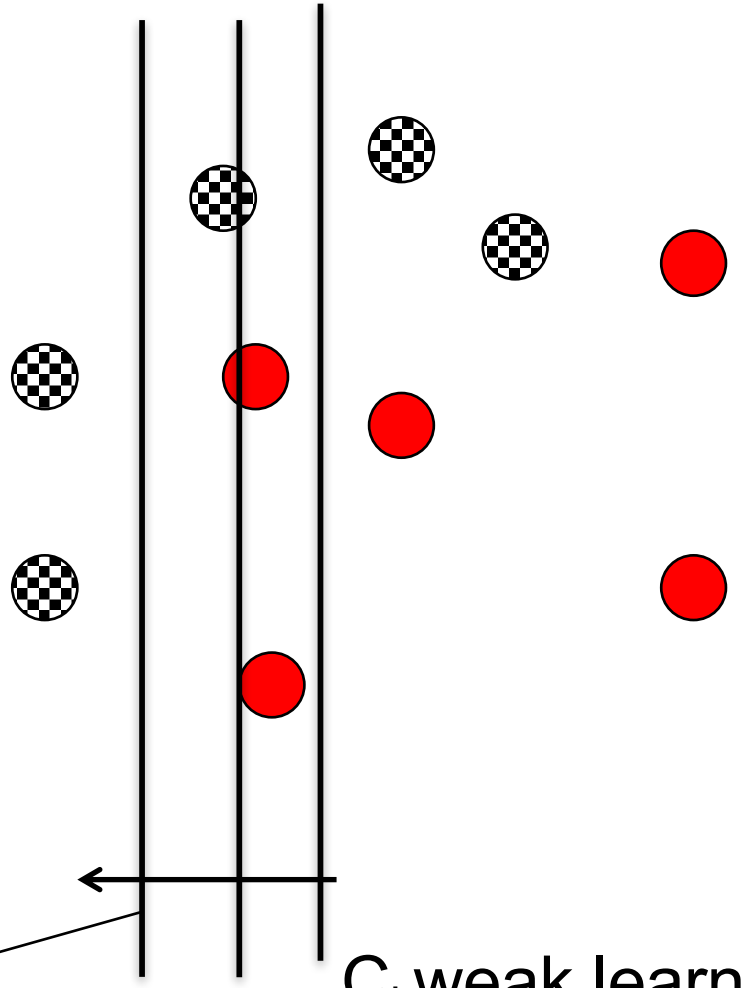
The problem:

Classification

5 red

5 BW

Use only linear classifiers



C_1 weak learner
Error rate = 3/10 = 0.3

C_i weak learner
Error rate = 4/10 = 0.4

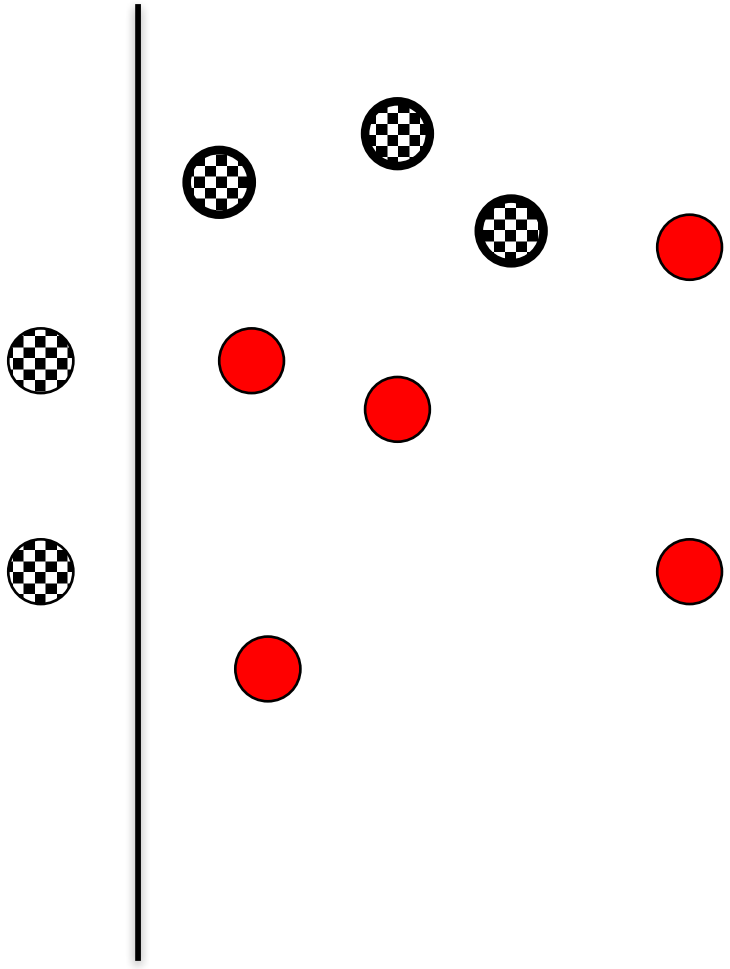
Boosting

Classification

5 red

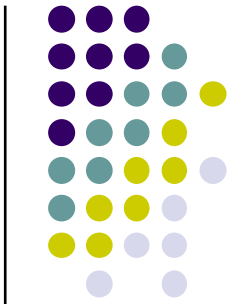
5 BW

Use only linear classifiers



C_1 weak learner

Boosting



- How do you compute the error rate?
- Trivial: number of errors / number of instances (=*n*).
- OK if all instances have the importance (weight).

- With weights?
- D_k the distribution of the weights over all instances at *k*th iteration.
- $D_k(i)$ weight for *i*th instance at *k*th iteration
- Normalization: $\sum_{i=1}^{n} D_k(i) = 1$
- and $D_1 = 1/n$

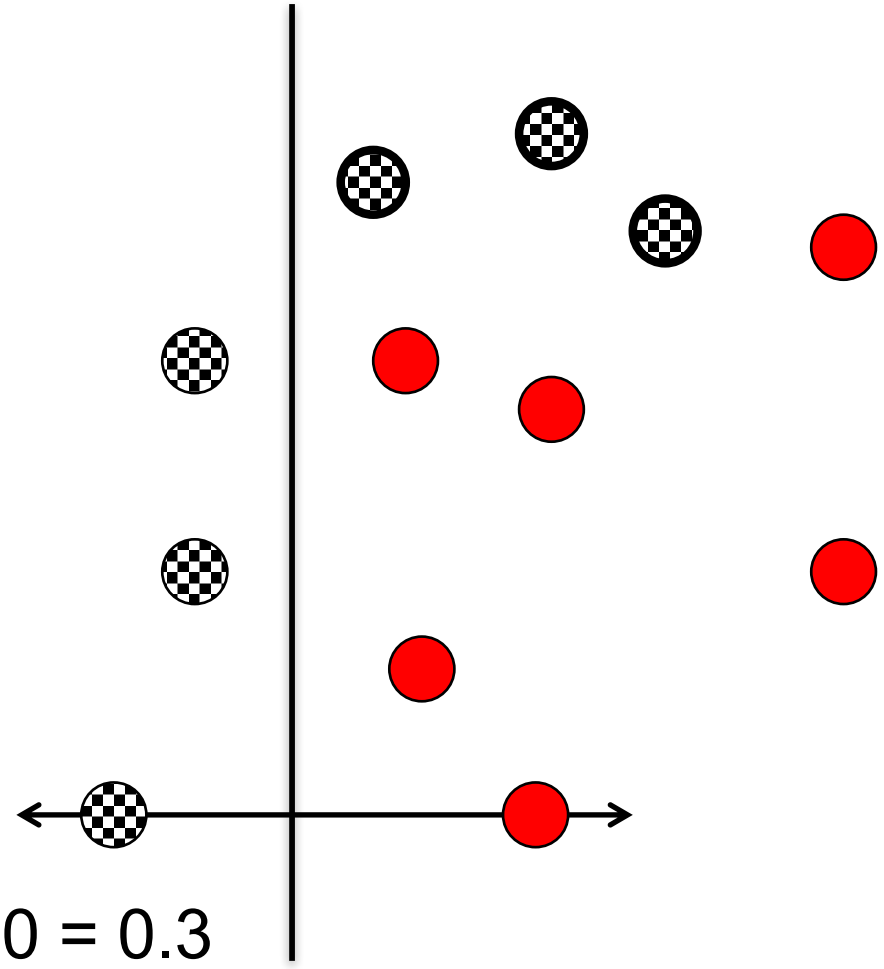
Boosting

Classification

5 red

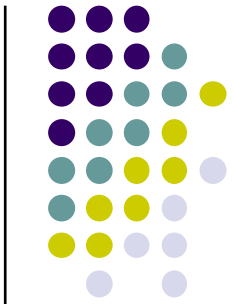
5 BW

Use only linear classifiers



C_1 weak learner
Error rate = 3 / 10 = 0.3
(without weights)

Boosting



- Update the weight assigned to the *i*th instance.
- If it is correctly classified, decrease its weight.
- Otherwise, increase its importance.

$$D_{k+1}(i) = \frac{D_k(i)}{Z_k} \cdot \begin{cases} \exp^{-\alpha_k} & if\ correct \\ \exp^{\alpha_k} & if\ incorrect \end{cases}$$

- Z_k, normalization factor (sum $D_k(i) = 1$)
- e_k overall error rate

$$\alpha_k = \frac{1}{2} \cdot \ln\left(\frac{1-e_k}{e_k}\right) > 0$$

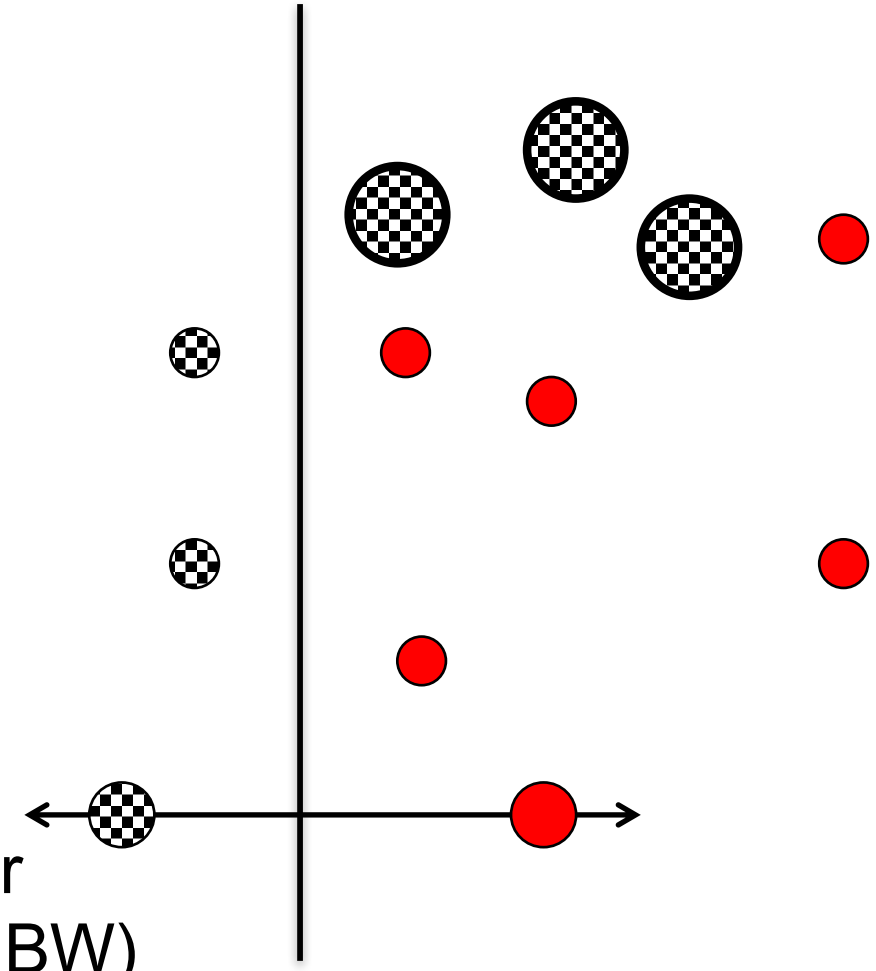
Boosting

Classification

5 red

5 BW

Use only linear classifiers



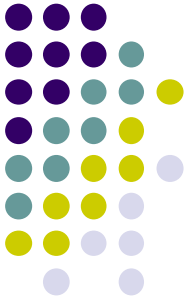
$k = 1$

C_1 weak learner

Error rate e_k (3 BW)

$e_k = 0.1 + 0.1 + 0.1 = 0.3$

Boosting



In the example,

$e_k = 0.3$

$\alpha_k = 0.5 * \ln(0.7/0.3) = 0.424$

Weight correct instance = 0.1 * exp(-0.42) = 0.065

Weight incorrect instance = 0.1 * exp(0.42) = 0.153

After normalization ($/Z_k$)

Weight correct instance = 0.071

Weight incorrect instance = 0.167

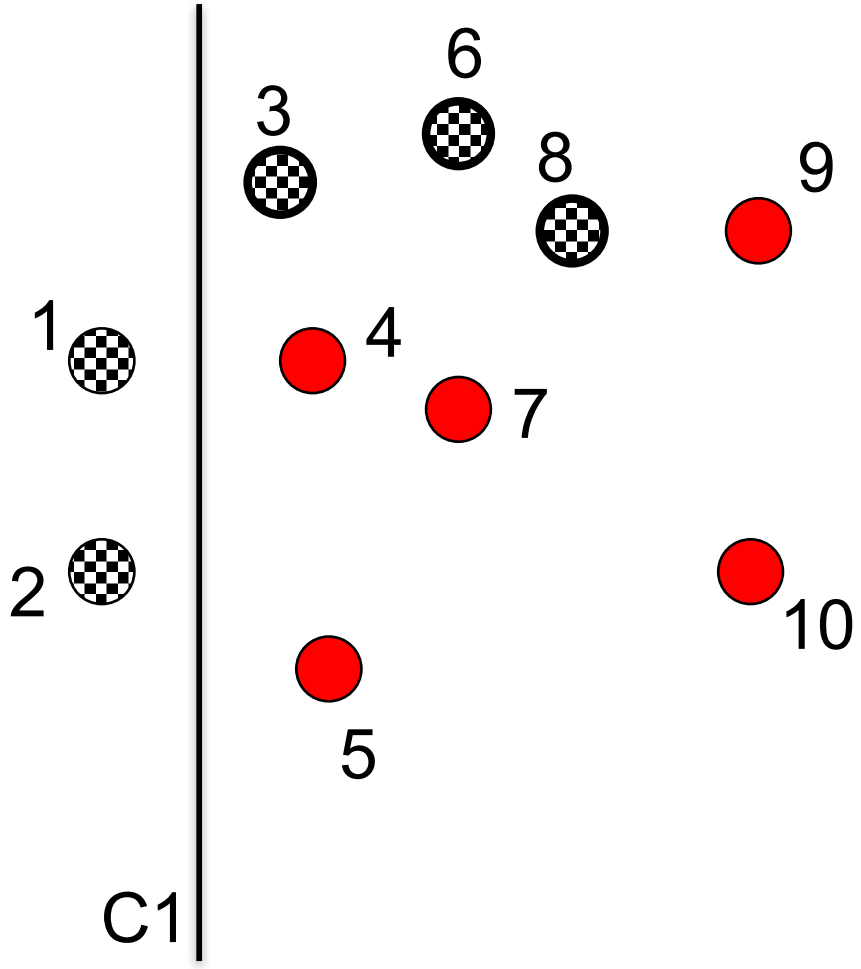
Example in XL

Classification

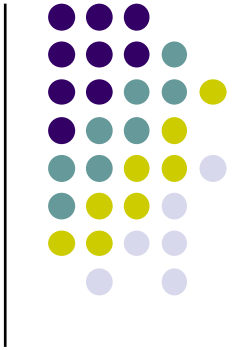
5 red

5 BW

Use only linear classifiers



Example in XL



ID	Do	Error?	Ek	D'1	D1
1	0.1	0	0	0.065	0.071
2	0.1	0	0	**0.065**	**0.071**
3	0.1	1	0.1	**0.153**	**0.167**
4	0.1	0	0	0.065	0.071
5	0.1	0	0	0.065	0.071
6	0.1	1	0.1	0.153	0.167
7	0.1	0	0	0.065	0.071
8	0.1	1	0.1	0.153	0.167
9	0.1	0	0	0.065	0.071
10	0.1	0	0	0.065	0.071

Boosting

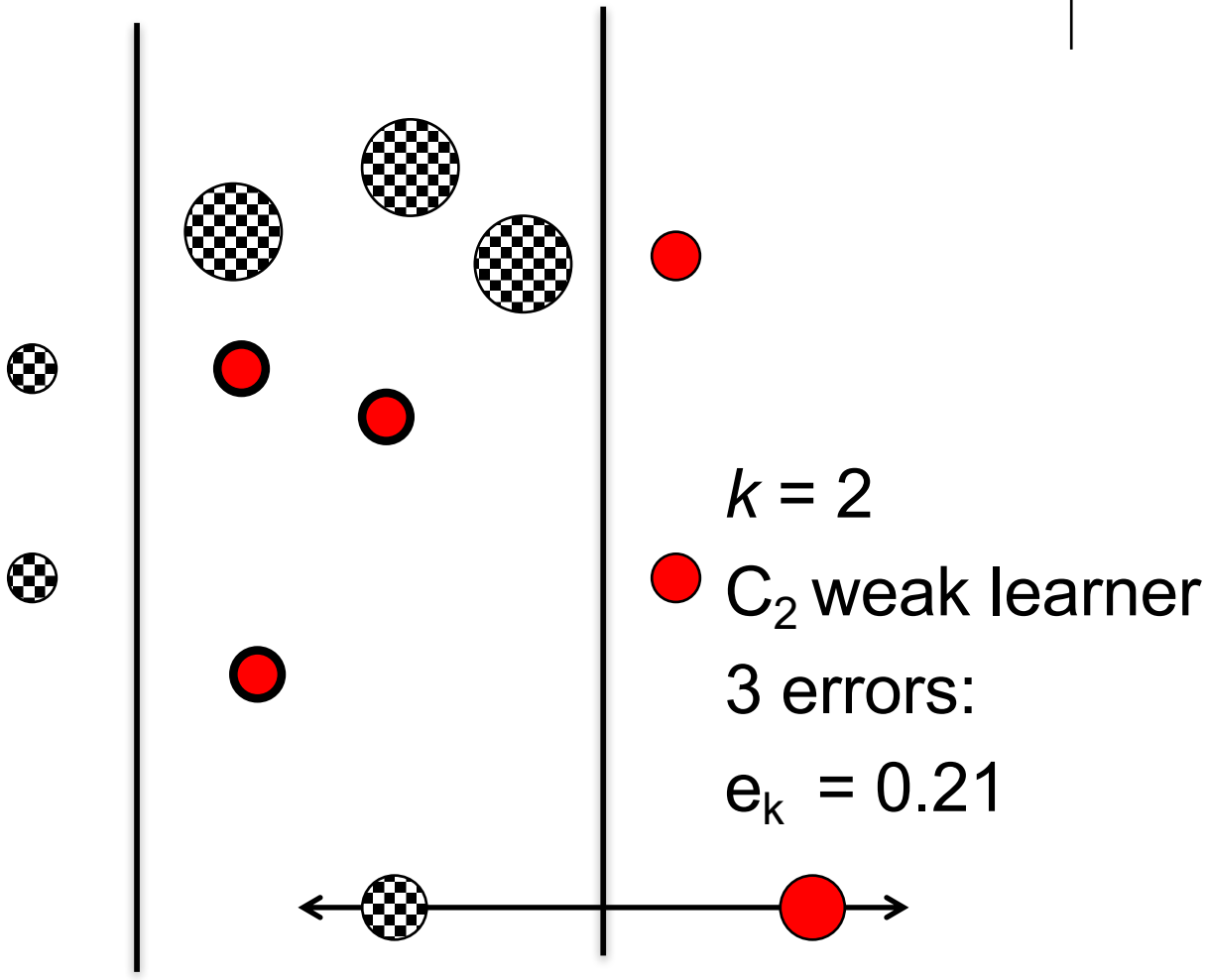
The problem:

Classification

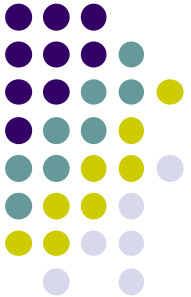
5 red

5 BW

Use only linear classifiers



Boosting



In the example, 3 small red dots are incorrect…

$e_k = 0.0655 + 0.0655 + 0.0655 = 0.214$

$\alpha_k = 0.5 * \ln(0.786 / 0.214) = 0.650$

Weight' correct instance = $0.1 * \exp(-0.65) = 0.037$

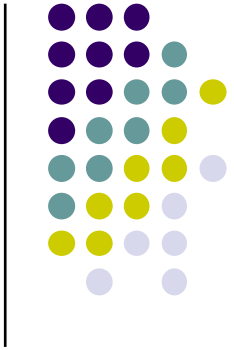
Weight' incorrect instance = $0.1 * \exp(0.65) = 0.137$

After normalization

Weight correct instance = 0.045

Weight incorrect instance = 0.167

Example in XL



ID	Do	Error?	Ek	D'1	D1
1	0.071	0	0	**0.037**	**0.045**
2	0.071	0	0	0.037	0.045
3	0.167	0	0	0.087	0.106
4	0.071	1	0.071	**0.137**	**0.167**
5	0.071	1	0.071	0.137	0.167
6	0.167	0	0	0.087	0.106
7	0.071	1	0.071	0.137	0.167
8	0.167	0	0	0.087	0.106
9	0.071	0	0	0.037	0.045
10	0.071	0	0	0.037	0.045

Boosting

The problem:

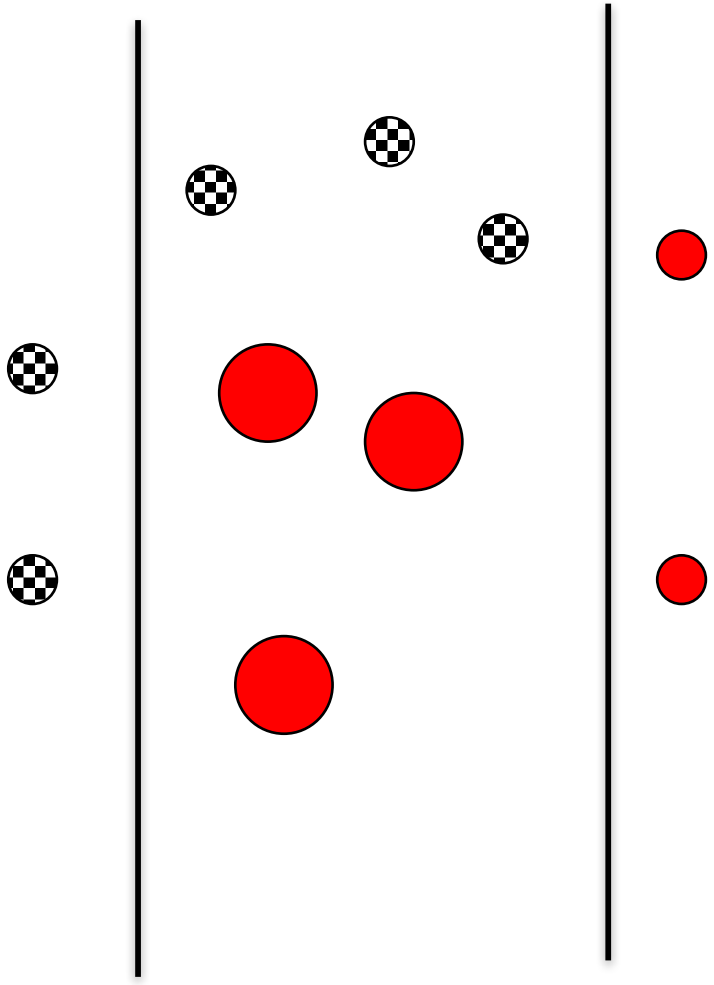
Classification

5 red

5 BW

Use only linear classifiers

$k = 3$



Boosting

The problem:

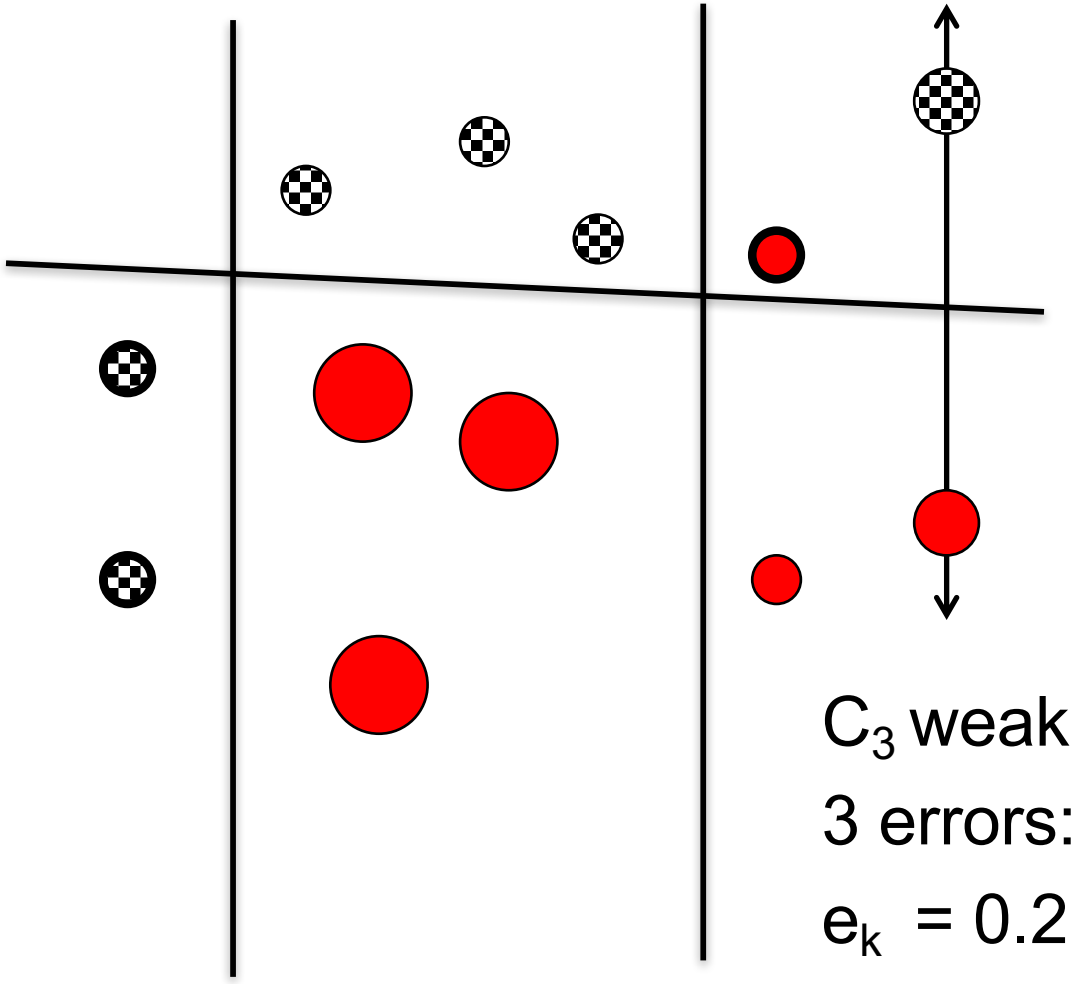
Classification

5 red

5 BW

Use only linear classifiers

$k = 3$



C_3 weak learner

3 errors:

$e_k = 0.21$

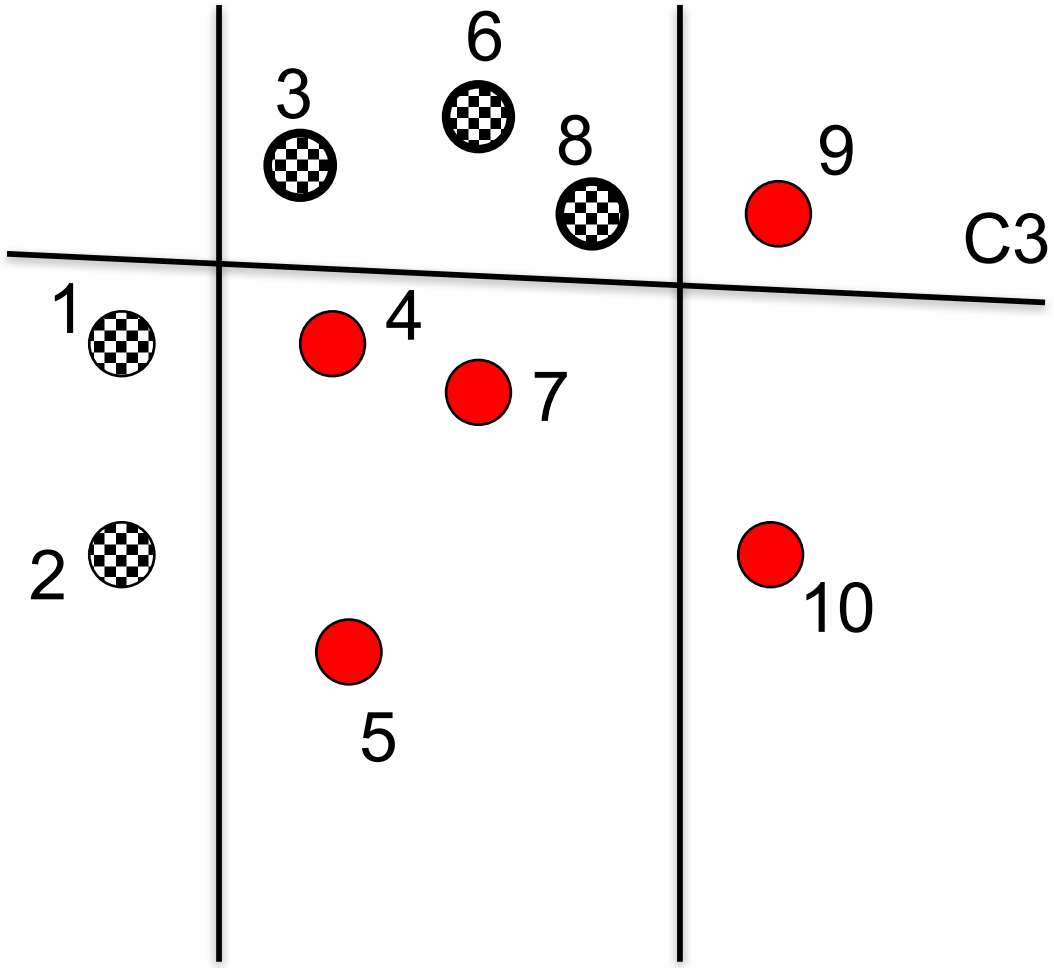
Example in XL

Classification

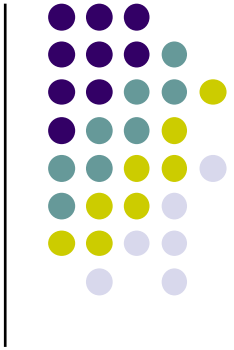
5 red

5 BW

Use only linear classifiers

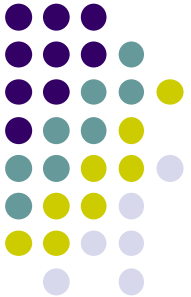


Example in XL



ID	D2	Error?	Ek	D'3	D3
1	0.071	1	0.045	**0.114**	**0.167**
2	0.071	1	0.045	0.114	0.167
3	0.167	0	0	**0.042**	**0.061**
4	0.071	0	0	0.066	0.096
5	0.071	0	0	0.066	0.096
6	0.167	0	0	0.042	0.061
7	0.071	0	0	0.066	0.096
8	0.167	0	0	0.042	0.061
9	0.071	1	0.045	0.114	0.167
10	0.071	0	0	0.018	0.026

Boosting



In the example, 3 small red dots are incorrect…

$e_k = 0.0167 + 0.0167 + 0.0167 = 0.136$

$\alpha_k = 0.5 * \ln(0.864 / 0.136) = 0.923$

Weight' correct instance = $0.1 * \exp(-0.923) = 0.042$

Weight' incorrect instance = $0.1 * \exp(0.923) = 0.114$

After normalization

Weight correct instance = 0.061

Weight incorrect instance = 0.167

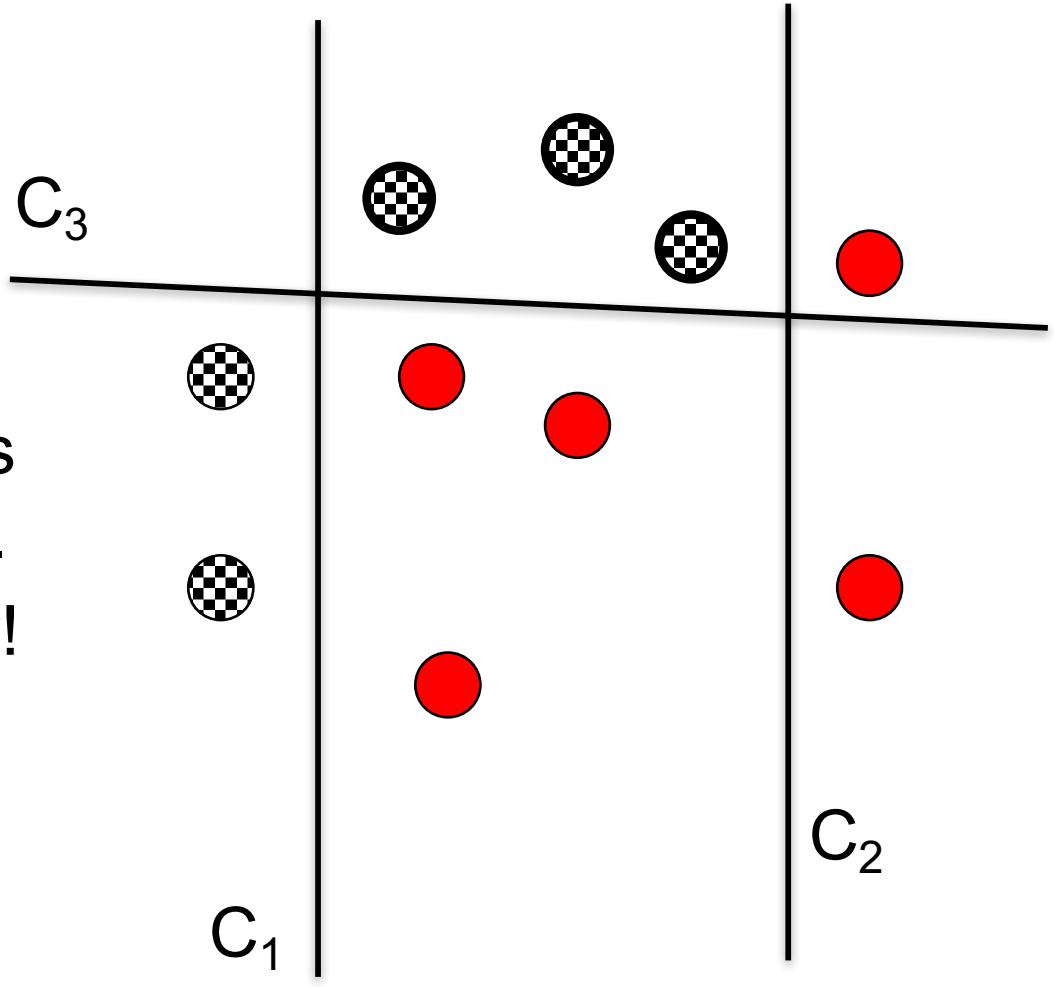
Boosting

5 red

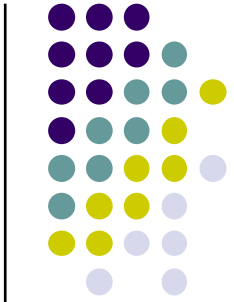
5 BW

Use only t=3 linear classifiers

to define a non-linear boundary!



AdaBoost.M1



- With this ensemble classifier, the new instance is classify as

$$H_{final}(x) = sign\left(\sum_{k=1}^{t} \alpha_k \cdot h_k(x)\right)$$

with each classifier

$$h_k(x) = X \mapsto \{-1, +1\}$$

Example in XL

1 for BW dot, -1 for red dot

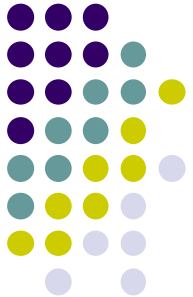
ID	C1	C1*alpha	C2	C2*alpha	C3	C3*alpha
1	1	0.424	1	0.650	-1	-0.923
2	1	0.424	1	0.650	-1	-0.923
3	-1	-0.424	1	0.650	1	0.923
4	-1	-0.424	1	0.650	-1	-0.923
5	-1	-0.424	1	0.650	-1	-0.923
6	-1	-0.424	1	0.650	1	0.923
7	-1	-0.424	1	0.650	-1	-0.923
8	-1	-0.424	1	0.650	1	0.923
9	-1	-0.424	-1	-0.650	1	0.923
10	-1	-0.424	-1	-0.650	-1	-0.923

Example in XL

1 for BW dot, -1 for red dot

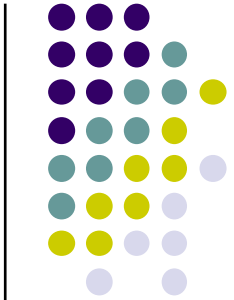
ID	Color	Sum	C3*alpha
1	BW	0.150	BW
2	BW	0.150	BW
3	BW	1.149	BW
4	red	-0.697	red
5	red	-0.697	red
6	BW	1.149	BW
7	red	-0.697	red
8	BW	1.149	BW
9	red	-0.150	red
10	red	0.150	BW

Boosting



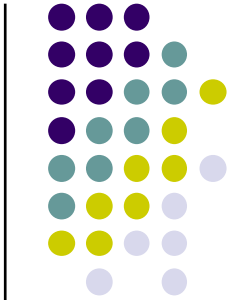
- Boosting needs weights … but
- Can adapt learning algorithm ... or
- Can apply boosting without weights
  - resample with probability determined by weights
  - disadvantage: not all instances are used
  - advantage: if error > 0.5, can resample again
- Stems from computational learning theory
- Theoretical result:
  - training error decreases exponentially
- Also:
  - works if base classifiers are not too complex, and
  - their error doesn't become too large too quickly

Boosting



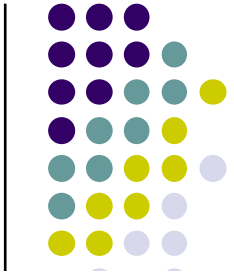
- Continue boosting after training error = 0?
- Puzzling fact:
  generalization error continues to decrease!
  - Seems to contradict Occam's Razor
- Explanation:
  consider margin (confidence), not error
  - Difference between estimated probability for true class and nearest other class (between –1 and 1)
- Boosting works with weak learners only condition: error doesn't exceed 0.5
- In practice, boosting sometimes overfits (in contrast to bagging)

Conclusion



- Bias-Variance decomposition is an important concept
- Ensemble learning is usually a good solution when the performance is the main criterion
- Difficult to explain the proposed solution
- Bagging works well with decision tree (high variance)
- Randomization: yes if we have a “random” choice in the learning algorithm
- In practice, boosting sometimes over-fits (in contrast to bagging)
- Working with different learning schemes is more complex than working with the same classifier (with different training sets or randomization)

AdaBoost.M1



Model Generation

```
Assign equal weight to each training instance.
For each of t iterations:
  Apply learning algorithm to weighted dataset and store resulting
    model.
  Compute error e of model on weighted dataset and store error.
  If e equal to zero, or e greater or equal to 0.5:
    Terminate model generation.
  For each instance in dataset:
    If instance classified correctly by model:
       Multiply weight of instance by e / (1 - e).
  Normalize weight of all instances.
```

Classification

```
Assign weight of zero to all classes.
For each of the t (or less) models:
  Add -log(e / (1 - e)) to weight of class predicted by model.
Return class with highest weight.
```