

Classifier Ensembles for Changing Environments

Ludmila I. Kuncheva
School of Informatics
University of Wales, Bangor, UK

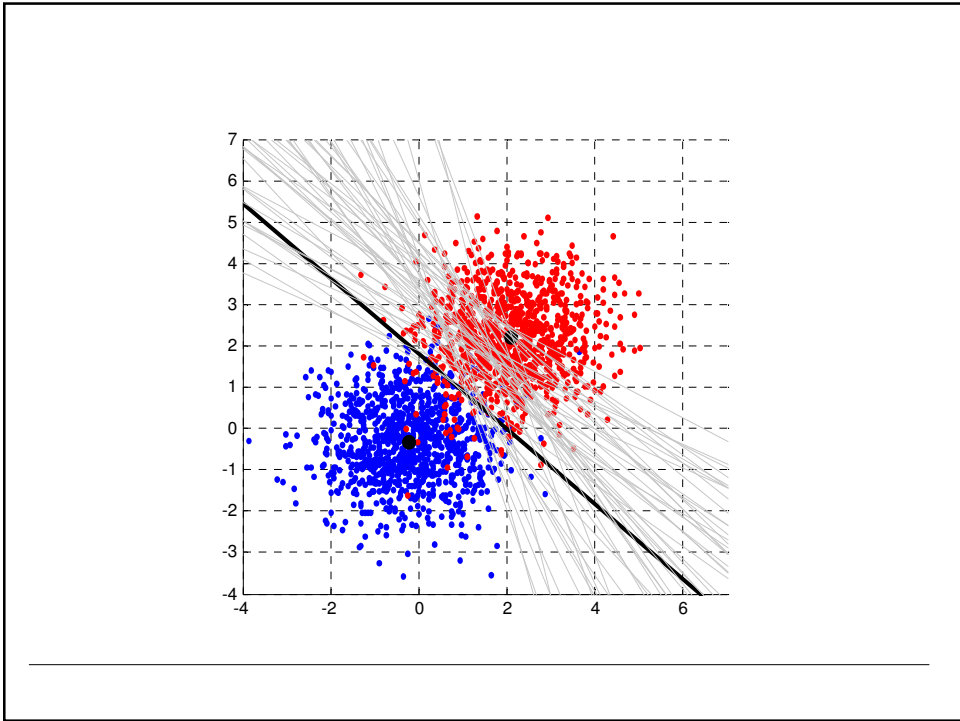


Good news: We can now design an almost perfect classifier for any pattern recognition problem, given a sufficient data set.

Bad news: The problem characteristics change with time and our classifier might quickly become outdated and inaccurate.

Example: Distinguish SPAM e-mail from legitimate e-mail

- user specific (preference change)
 - constantly varying class description (new tricks)
-



Concept drift and types of changes

Consider a probabilistic description of the problem

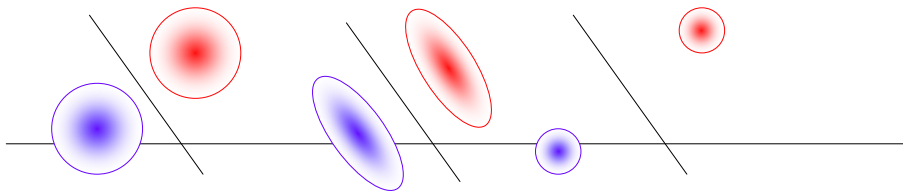
prior probabilities $P(\omega_1), \dots, P(\omega_c)$

class-conditional pdf-s $p(\mathbf{x} | \omega_1), \dots, p(\mathbf{x} | \omega_c)$

posterior probabilities $P(\omega_1 | \mathbf{x}), \dots, P(\omega_c | \mathbf{x})$

(population drift)

Not every change will invalidate the classifier.



Type of change (1)

gradual: seasonal, demographic, habitual

abrupt: "hidden context"

Type of change (2)

random noise

random trends (gradual changes)

random substitutions (abrupt changes)

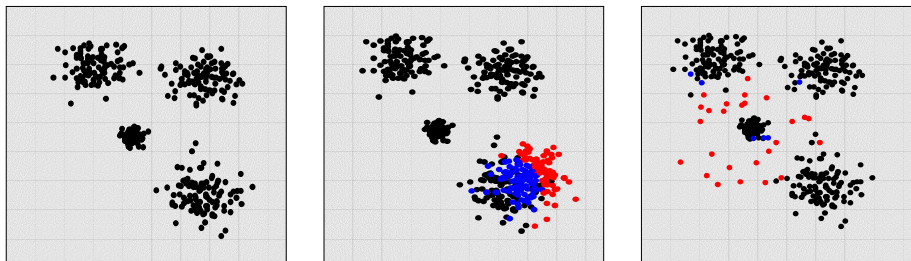
systematic trends ("recurring contexts")

Knowing the type of the change makes the problem much easier.

Detecting a change

Unlabeled data in on-line classification

- mammogram scanning (human expert needed for the verification)
- credit application (true label is available 2 years after the decision)
- spam e-mail filtering (user's confirmation is needed)



Unlabeled data

- (a) estimate $p(\mathbf{x})$ by a separate “classifier”
- (b) set θ = probability threshold for novelty
 t = proportion threshold for novelty
 W = window size
- (c) for any \mathbf{x} , calculate $p(\mathbf{x})$, If $p(\mathbf{x}) < \theta$, increment the novelty counter
- (d) If the proportion of novel objects in the current window exceeds t , report **novelty**

Labeled data

Monitor the changes in the classification accuracy. A sudden or gradual and steady drop may indicate **novelty**.

Learn to forget



The classifier is learning on-line

(there is a fresh supply of labeled data at any time)

If stopped at time t , this will be the best classifier for the distribution at time t , called “**any time learning**”.

Old (outdated) knowledge is forgotten within the training process.

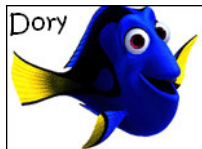
To design a suitable forgetting strategy we need to make assumptions about the type of changes.

Forgetting by **ageing at a constant rate**

(forget training objects and use the most recent "chunk" of data to retrain the classifier)

How large a window do we need?

- Window too small = responsive classifier but with little memory



- Window too large = sluggish, tardy, inert classifier

The famous **stability-plasticity dilemma**

The parameters for the forgetting have to be adjusted for individual problem

Forgetting by **ageing at a variable rate**

If a change is detected the window shrinks (past examples have to be forgotten).

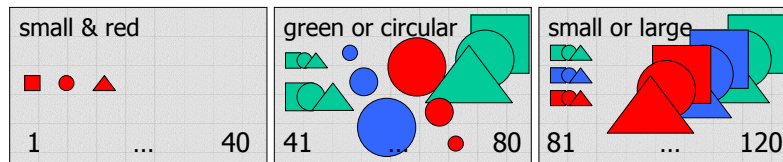
Example: Reference [G. Widmer and M. Kubat, Learning in the presence of concept drift and hidden contexts, *ML*, **23**, 1996, 69-101.]

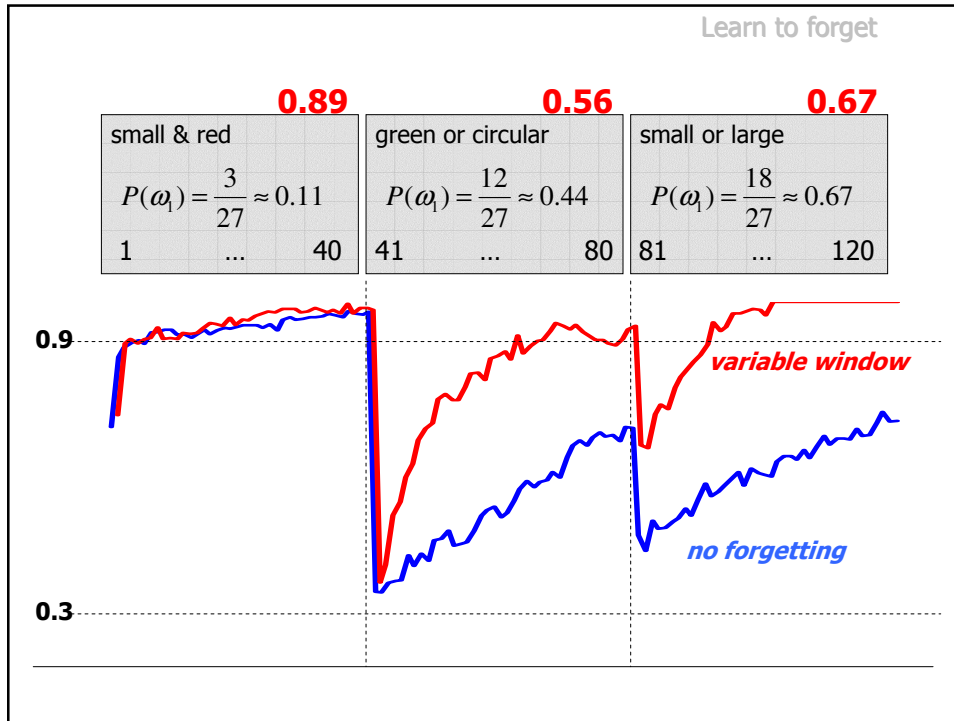
3 categorical features

size \in {small, medium, large}

color \in {red, green, blue}

shape \in {square, circular, triangular}





Learn to forget

Density-based forgetting

Sometimes older data points can be more useful than newer points.

Consider k-nn methods (**adaptive nearest neighbor** models)

We delete training points depending on how many times they have been referred to as the nearest neighbor.

Each point has weight attached to it. It may decay with time and also be updated by the frequency of calls to this point.

Online learning (incremental learning)

Large data streams

- telecommunications
- credit card transactions
- Internet searches

Ideally, the resultant classifier at step t should be equivalent to a classifier built on the whole training data prior to t .

Usually online learning assumes a **static environment**

A good online classifier

- Learns from **one pass** through the data set*
- Updates with every new data point within **limited memory and processing time***
- Is capable of **any-time-learning***

* valid for changing environments as well

Online learning

Online classifier models

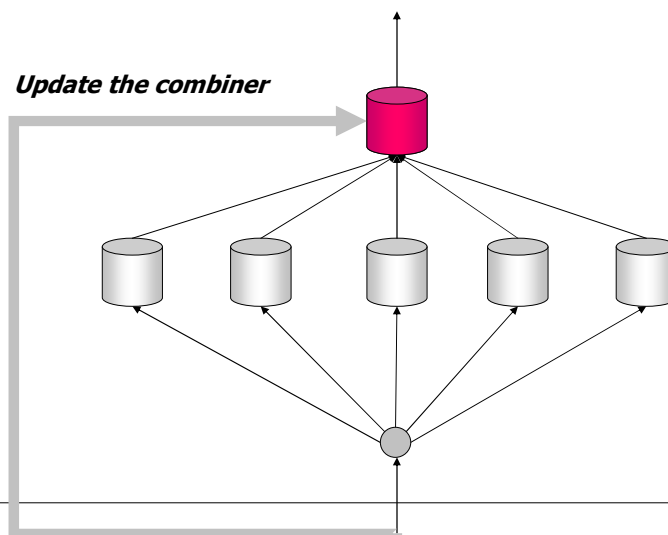
- **Rosenblatt's perceptron** (not with a single pass through the data and not any-time-learner)
- **LVQ** (Learning Vector Quantization)
- **Decision trees**
 - VFDT (Very Fast Decision Trees) [Domingos & Hulten, 2000] originally designed for static environments
 - Concept-adapting VFDT, to accommodate changing environments [Hulten et al., 2001]
- **Naïve Bayes** – updates the marginal probabilities with each new data point
- **Neural Networks** – capable of on-line learning (ARTMAP)
- **Nearest Neighbor** = Instance Based Learning (IB1-IB3 models [Aha et al., 1991])

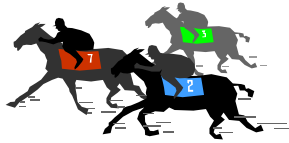
Ensemble strategies for changing environments

Approaches

1. Use dynamic combiners (horse racing): experts are trained in advance and only the combiner changes
2. Re-train the individual classifiers online
3. Structured changes in the ensemble (replace the loser)
4. Add new features

1. Dynamic combiners (Horse racing ensemble algorithms)



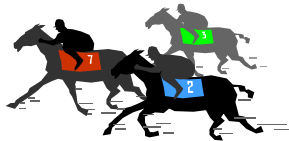


Not retrained!

Majority vote algorithm [Littlestone & Warmuth, 1994]

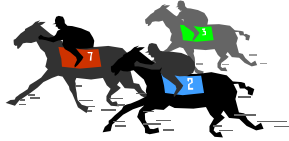
1. Train the classifiers in the ensemble D_1, \dots, D_L .
 (these are your friends, experts in horse racing)
 Set all weights to 1, $w_j = 1, i = 1, \dots, L$. Choose $\beta \in [0,1]$.
2. For a new x (race), calculate the support for each class as the sum of the weights for all classifiers that voted for that class. (Take all expert predictions for your chosen horse. Sum up the weights for those experts that predicted a win and compare with the sum of weights of those who predicted a loss.) Make a decision for the most supported class (bet or abstain).
3. Observe the true label of x (did your favourite win?) and update the weights of the classifiers that were wrong using $w_j \leftarrow \beta w_j$. Continue from 2.

COMBINER Retrained



Hedge β algorithm

1. Train the classifiers in the ensemble D_1, \dots, D_L .
 Set all weights to 1, $w_j = 1, i = 1, \dots, L$. Choose $\beta \in [0,1]$.
2. For a new x , *sample from the distribution on the classifiers and use the decision of the chosen classifier.*
3. Observe the true label of x and update the weights of the classifiers that were wrong. Continue from 2.



Winnow algorithm [Littlestone, 1988]

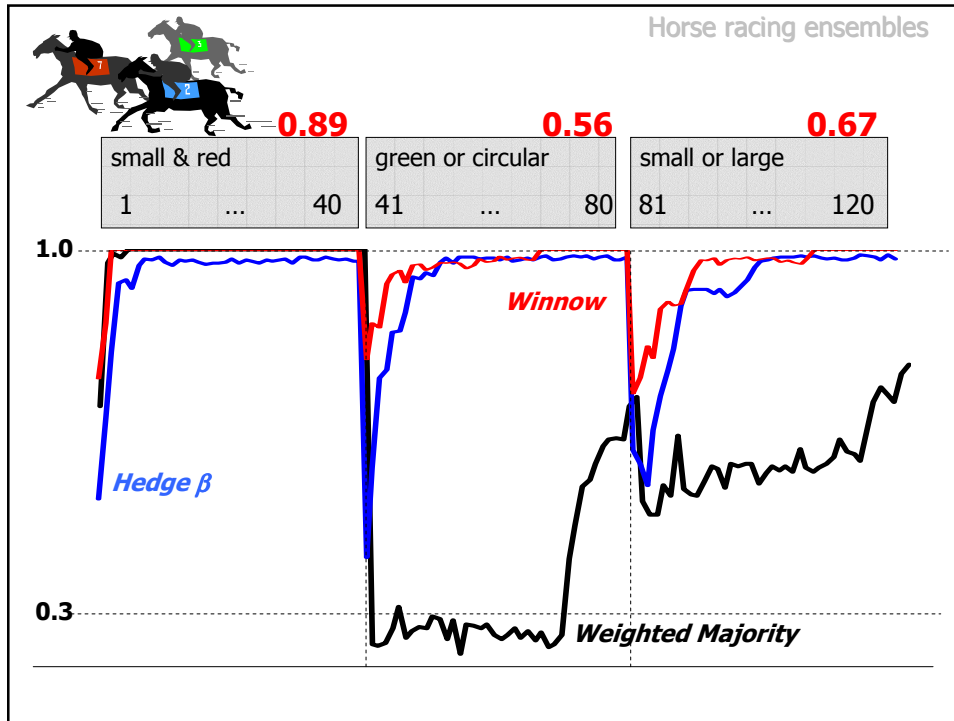
1. Train the classifiers in the ensemble D_1, \dots, D_L .
Set all weights to 1, $w_j = 1, i = 1, \dots, L$. Choose $\alpha > 1$.
 2. For a new x , calculate the support for each class as the sum of the weights for all classifiers that voted for that class. Make a decision for the most supported class.
 3. Observe the true label of x and update the weights of *all* the classifiers *if the ensemble prediction was wrong*.
If D_j was correct, then "promote" by $w_j \leftarrow \alpha w_j$
If D_j was wrong, then "demote" by $w_j \leftarrow w_j / \alpha$
Continue from 2.
-



0.89	0.56	0.67
small & red	green or circular	small or large
1 ... 40	41 ... 80	81 ... 120

Ensemble (***cheating!!!***):

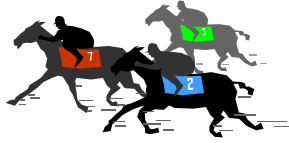
- 3 perfect classifiers, one for each stage
 - Hedge b, update only the weight for the chosen classifier
-



Horse racing ensembles

Blum [1995] for the **Winnow algorithm**

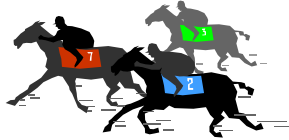
“learning simple things really well”



Mixture of experts (borrowing a neural network approach)

1. Initialize the individual classifiers D_1, \dots, D_L .
2. For each new \mathbf{x} , use a probability distribution over the set of classifiers $P(D_i | \mathbf{x}), i = 1, \dots, L$ to select one classifier for this \mathbf{x} , say D_k . Use D_k to label \mathbf{x} .
3. Observe the true label of \mathbf{x} and update

$$P(D_i | \mathbf{x}), i = 1, \dots, L \text{ and } D_k.$$

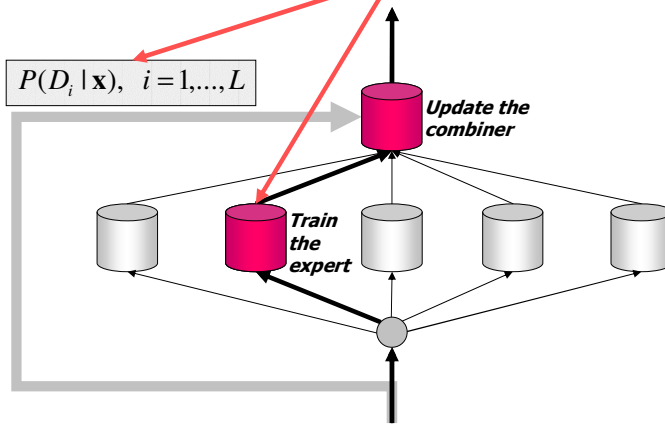


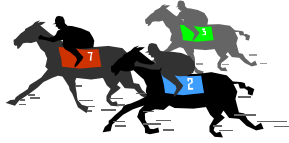
Mixture of experts



$P(D_i | \mathbf{x}), i = 1, \dots, L$

Correct or wrong?

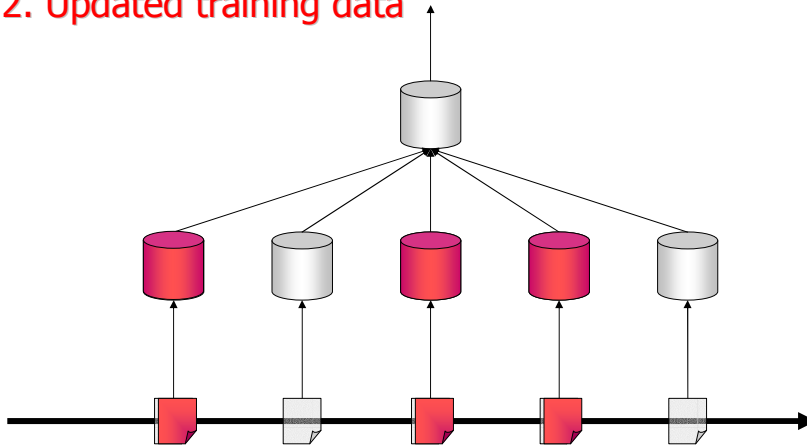




Update the combiner

1. Train the individual classifiers D_1, \dots, D_L .
2. For each new x , run the ensemble and label x .
3. Update the combination rule accordingly - **Naïve Bayes combiner, BKS, any online classifier model on the intermediate feature space**

2. Updated training data



Update the training sets (all or some) with each new x

Updated training data

Reusing the data points

Online bagging [Oza, 2001]

1. Initialize the individual classifiers D_1, \dots, D_L .
2. For each new x , for each classifier, $D_{k'}$ decide how many times x would have appeared in the training set of $D_{k'}$ if sampled with replacements from the training set acquired so far. Put that many copies of x in $T_k, k = 1, \dots, L$.
3. Update the ensemble and continue from 2.

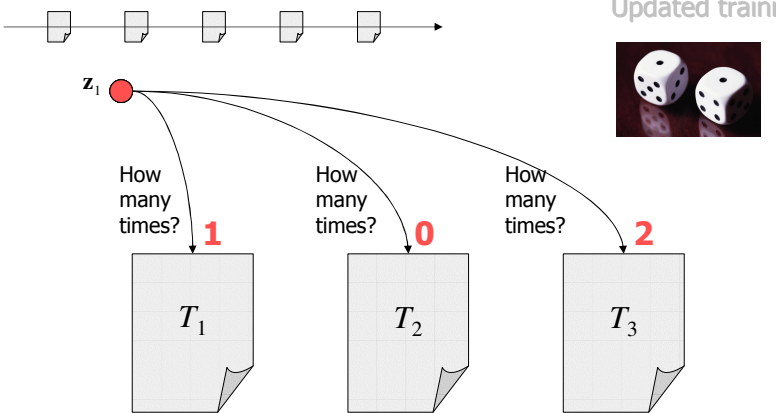
Online boosting [Oza, 2001]

...

2. The number of copies of x in T_k will depend on another parameter, λ . We start with $l = 1$ to update T_1 and D_1 . If D_1 misclassifies x , l increases so that x is more represented in T_2 . l is modified again for D_3 , etc., until the whole ensemble is retrained.

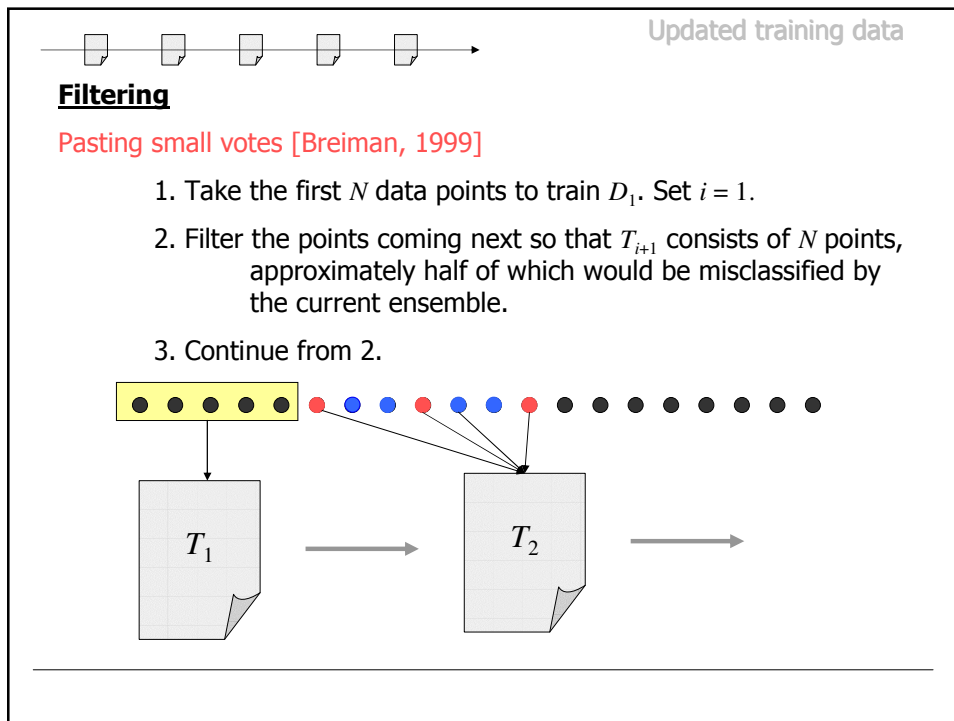
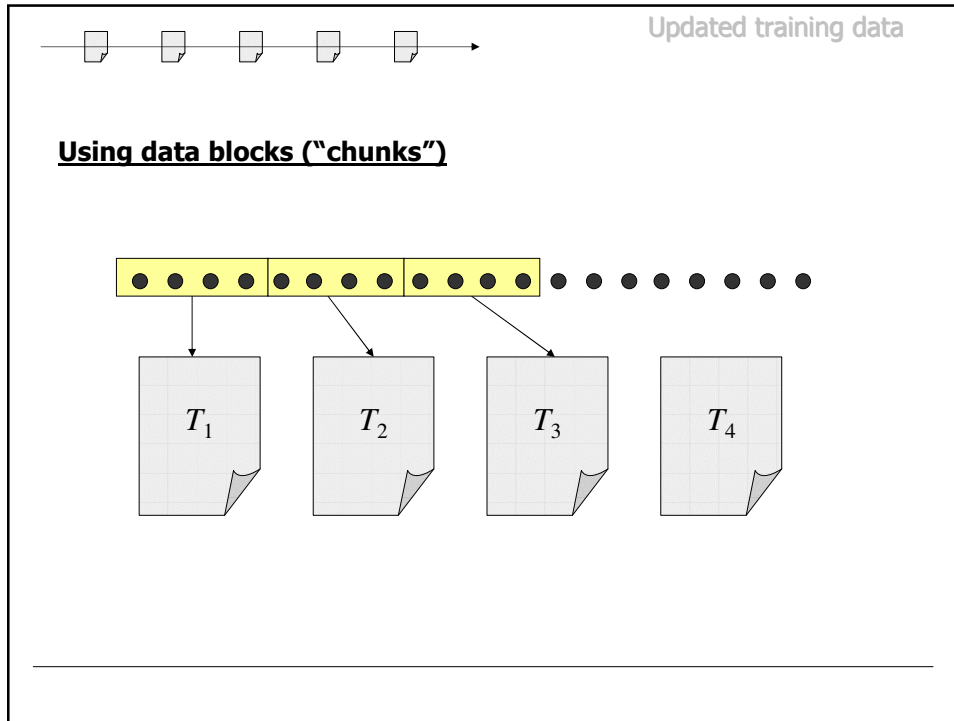
...

Updated training data



In a data set of size N , z_1 will appear K times, where K is a binomial random variable, $n=N, p=1/N$. For large N , we approximate with Poisson.

k	0	1	2	3	4
$P(K = k)$	0.3679	0.3679	0.1839	0.0613	0.0153





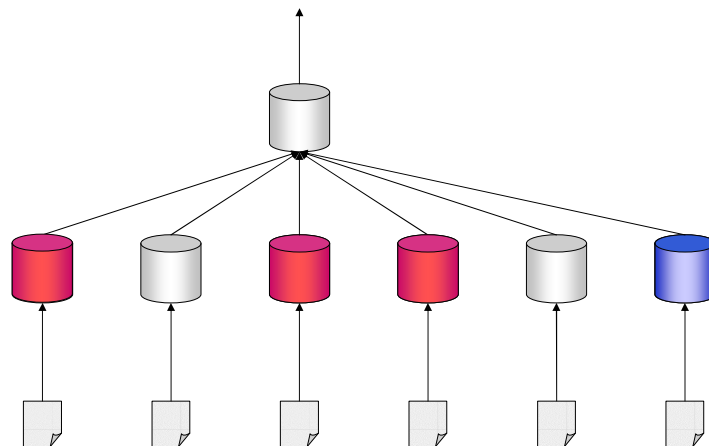
Updated training data

NOTE:

None of the methods in this subsection has special provisions for changing environments!

We need to take care of the forgetting mechanism explicitly.

3. Changing the ensemble structure





Replace the oldest [Wang et al, 2003]

Remove the "oldest" classifier and replace it by a newly trained one on the most recent "chunk" of data.

Replace the loser [Street & Kim, 2001]

Use a "quality score" for each classifier in the ensemble. Replace the classifier with the lowest score.

= A soft version of Winnow

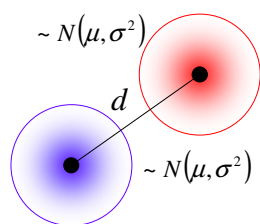
An experiment

2 classes

equal priors = 0.5

normal distributions with the same sigma = 0.5

theoretical error - available



$$\varepsilon = 1 - \Phi\left(\frac{d}{2\sigma}\right)$$

neat approximation (max error 0.005):

$$\Phi(z) = 0.5 + \frac{z(4.4 - z)}{10}, \quad 0 \leq z \leq 2.5$$

2 classes

equal priors = 0.5

normal distributions with the same sigma = 0.5

theoretical error - available

ensemble: random linear classifiers (could be less accurate than chance!)

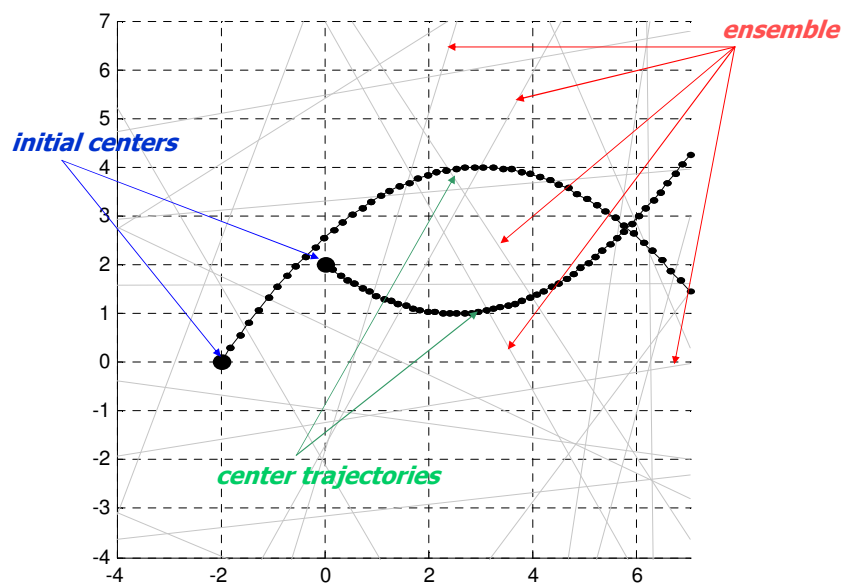
changing environment:

the class centers move on pre-specified trajectories

50 fixed positions on the trajectories

N points generated from each fixed position

100 points generated from each fixed position for testing



The 3 competitors

Single classifier

1. Start with a random classifier
2. Get x . If incorrectly labeled, retrain the classifier using the last 20 objects (when available)

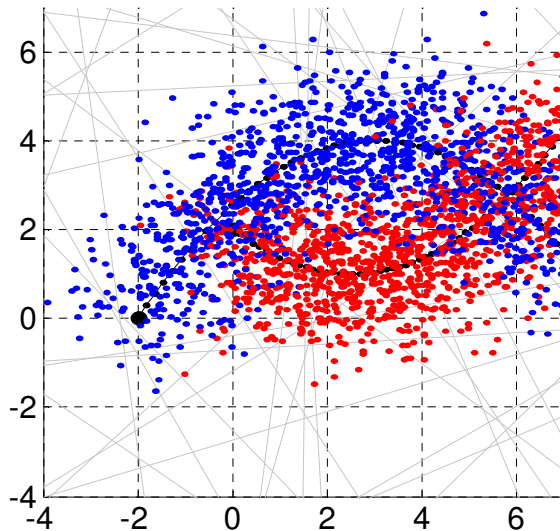
Winnow

1. Start with a random ensemble. Initialize the weights (equal)
2. Get x . If incorrectly labeled, update all weights

Winnow+ Replace the loser

1. Start with a random ensemble
2. Get x . If incorrectly labeled, update all weights as in **Winnow**
3. Replace the worst classifier by a newly trained one as in **Single classifier**

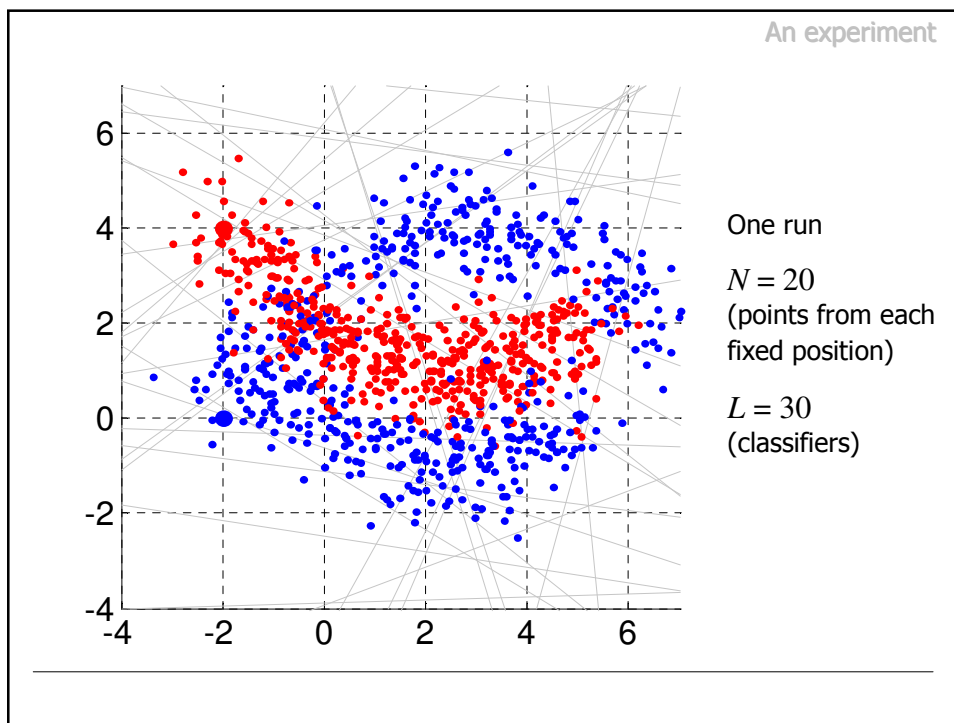
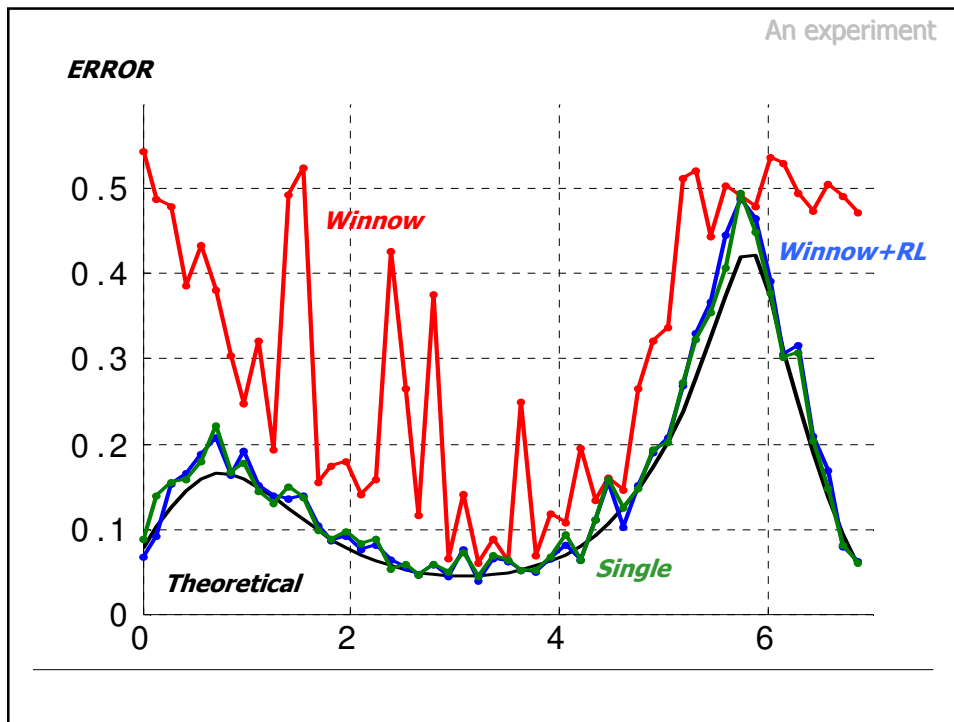
Run demo8/9

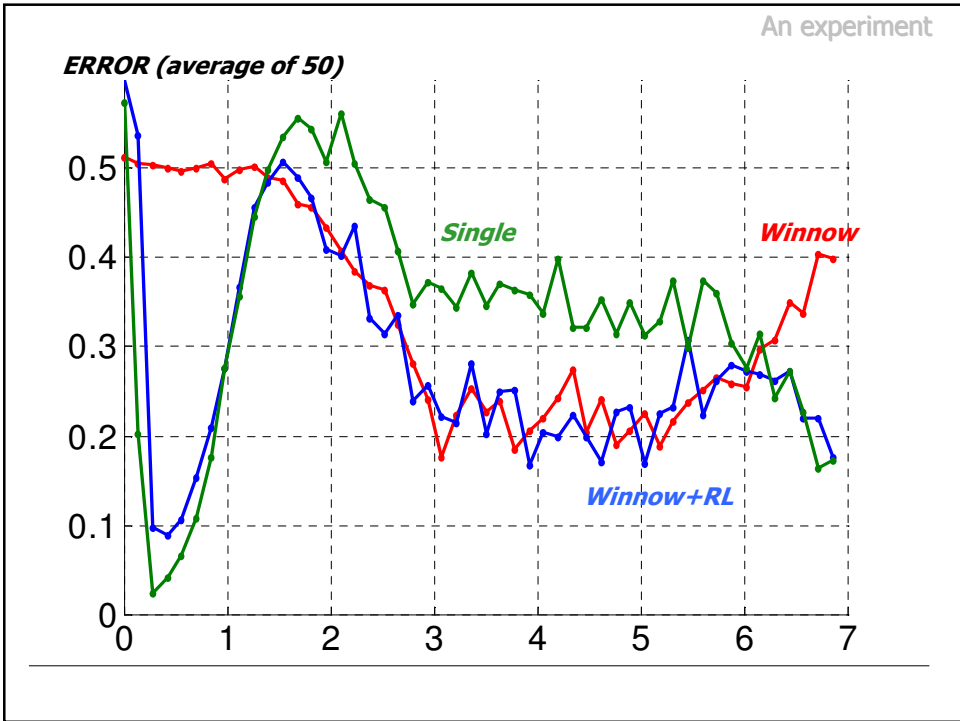
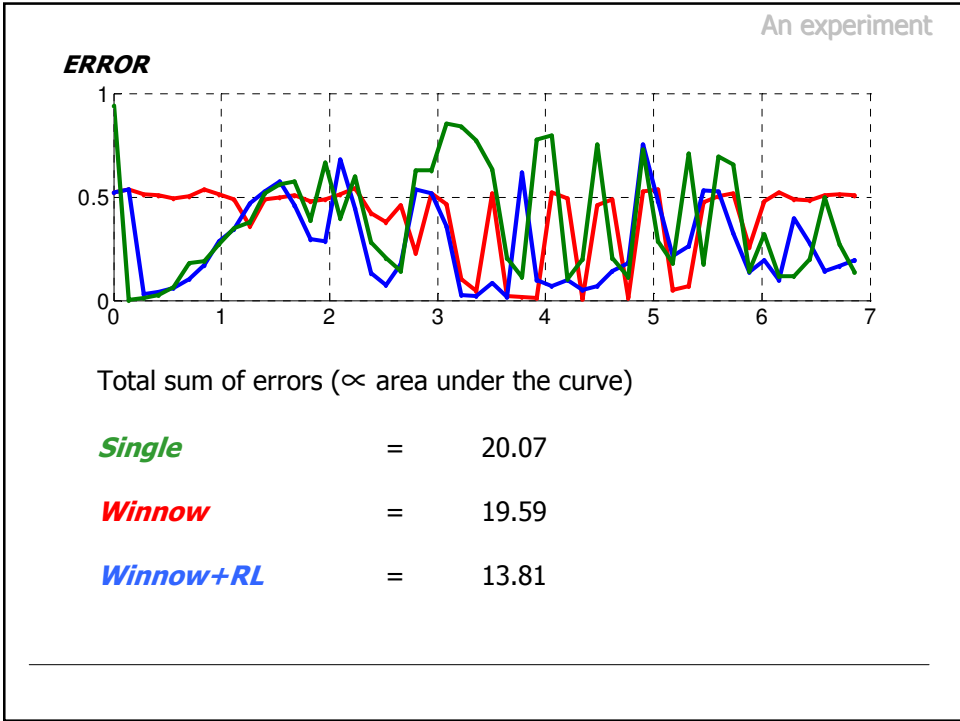


One run

$N = 50$
(points from each
fixed position)

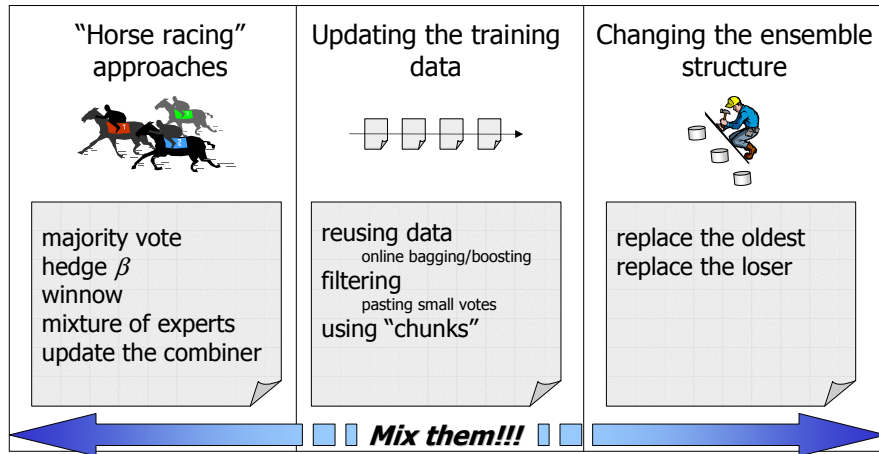
$L = 30$
(classifiers)





Conclusions (what did we learn?)

Strategies for building classifier ensembles for changing environments

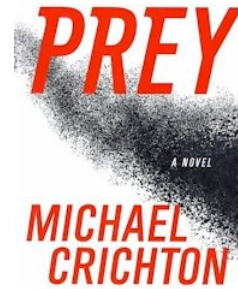


Conclusions (what does the future hold?)

- Make the classifiers "intelligent", aware of their competence and behaviour.
- Borrow ideas and tricks from agent technologies and artificial life.
- Theory (as usual) is going to fight for breath trying to catch the running ahead heuristics.
- And... watch out for emergent behaviour!!!

Conclusions (what does the future hold?)

“Within fifty to hundred years, a new class of organisms is likely to emerge...they will be `alive` under any reasonable definition of the word... The pace will be extremely rapid...”



**To be human ...
is to be hunted**

