CIS606 Machine Learning
Spring 2013 Lecture 15

Feature Selection

Wei Lee Woon and Zeyar Aung

Original Source:

http://www.macs.hw.ac.uk/~dwcorne/Teaching/DMML/DMML8_fese.ppt

Today

Feature Selection:

- What
- Why
- How

Feature Selection: What

You have some data, and you want to use it to build a classifier, so that you can predict something (e.g. likelihood of cancer)

Feature Selection: What

You have some data, and you want to use it to build a classifier, so that you can predict something (e.g. likelihood of cancer)

The data has 10,000 fields (features)

Feature Selection: What

You have some data, and you want to use it to build a classifier, so that you can predict something (e.g. likelihood of cancer)

The data has 10,000 fields (features)

you need to cut it down to 1,000 fields before you try machine learning. Which 1,000?

Feature Selection: What

You have some data, and you want to use it to build a classifier, so that you can predict something (e.g. likelihood of cancer)

The data has 10,000 fields (features)

you need to cut it down to 1,000 fields before you try machine learning. Which 1,000?

The process of choosing the 1,000 fields to use is called Feature Selection

Datasets with many features

Gene expression datasets (~10,000 features)

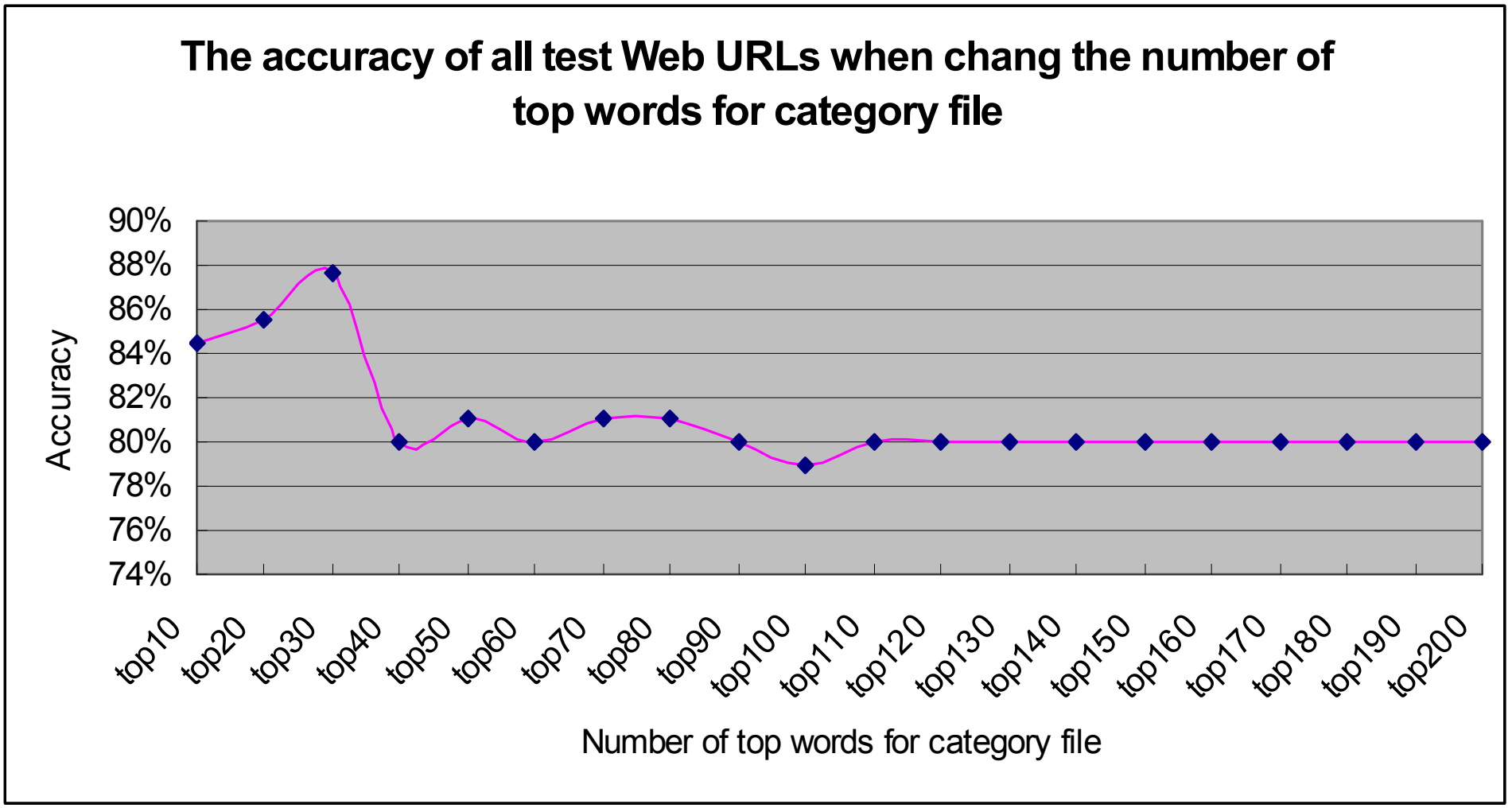
http://www.ncbi.nlm.nih.gov/sites/entrez?db=gds

Proteomics data (~20,000 features)

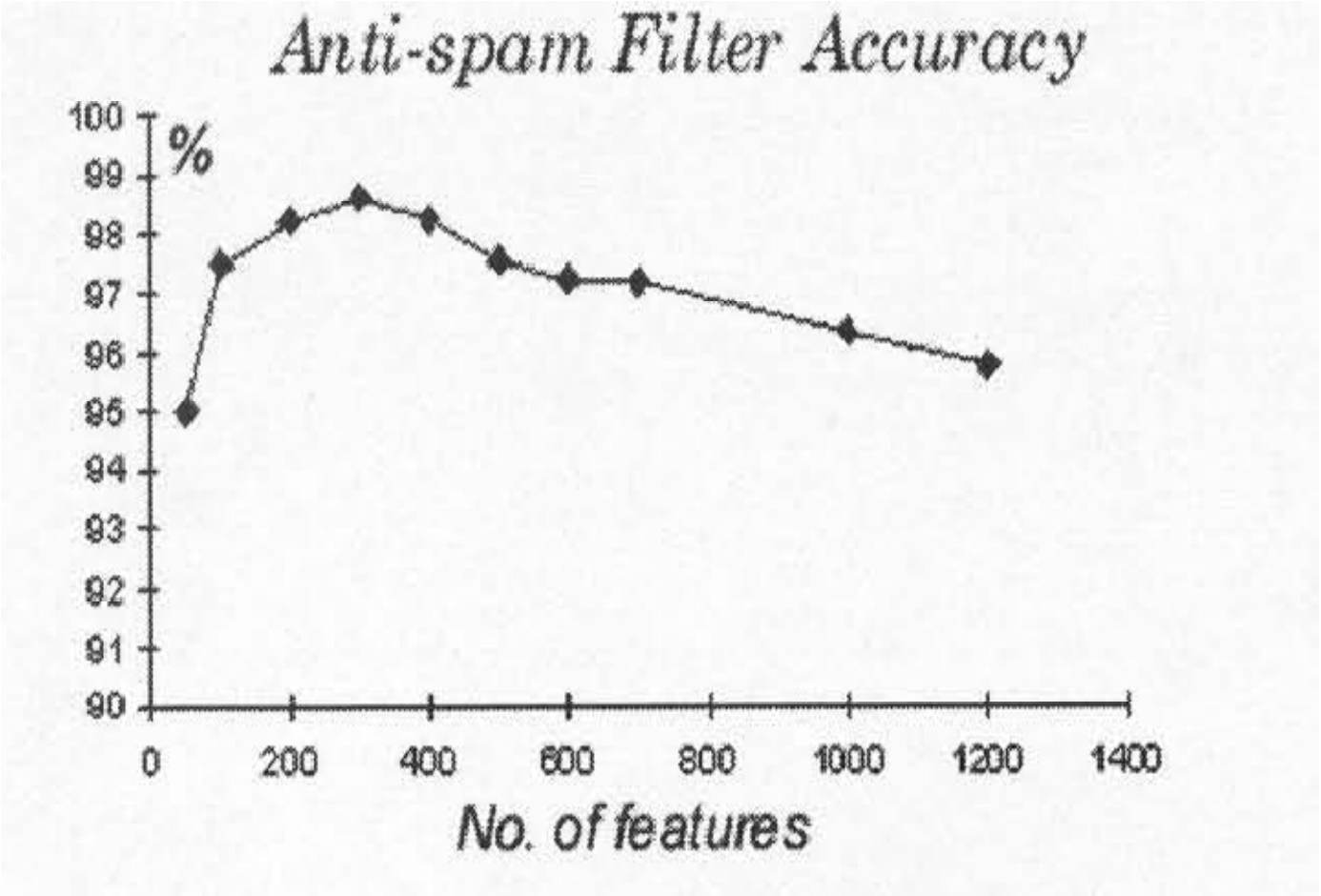
http://www.ebi.ac.uk/pride/

Feature Selection: Why?

Feature Selection: Why?



Feature Selection: Why?



From http://elpub.scix.net/data/works/att/02-28.content.pdf

Quite easy to find lots more cases from papers, where experiments show that accuracy reduces when you use more features

- Why does accuracy reduce with more features?
- How does it depend on the specific choice of features?
- What else changes if we use more features?
- So, how do we choose the right features?

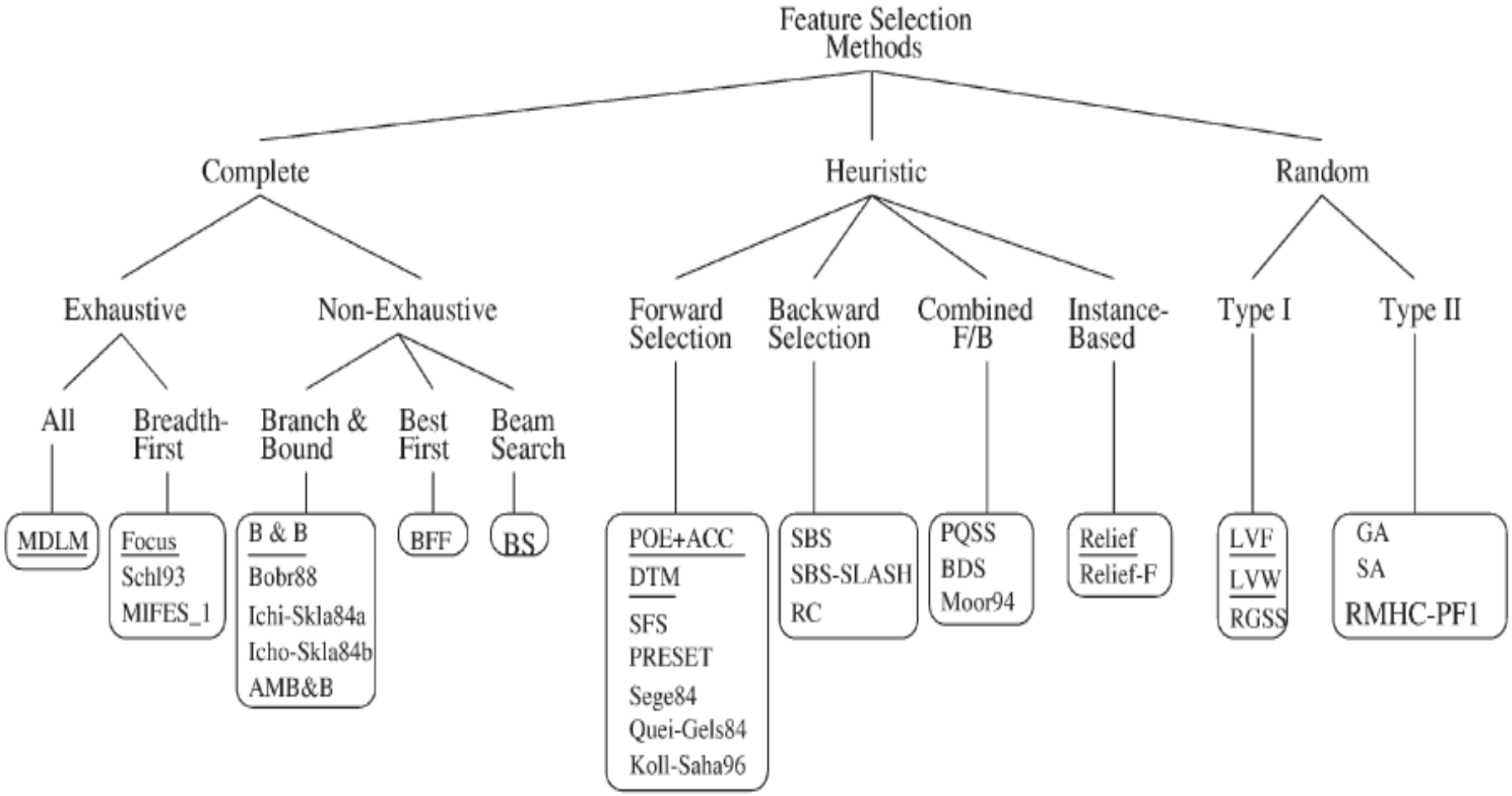
Why accuracy reduces:

- Note: suppose the best feature set has 20 features. If you *add* another 5 features, typically the accuracy of machine learning may reduce. But you still have the original 20 features!! Why does this happen???

Noise / Explosion

- The additional features typically add *noise*. Machine learning will pick up on spurious correlations, that might be true in the training set, but not in the test set.
- For some ML methods, more features means more *parameters* to learn (more NN weights, more decision tree nodes, etc…) – the increased space of possibilities is more difficult to search.

Feature selection methods



Dash and Liu, 1997

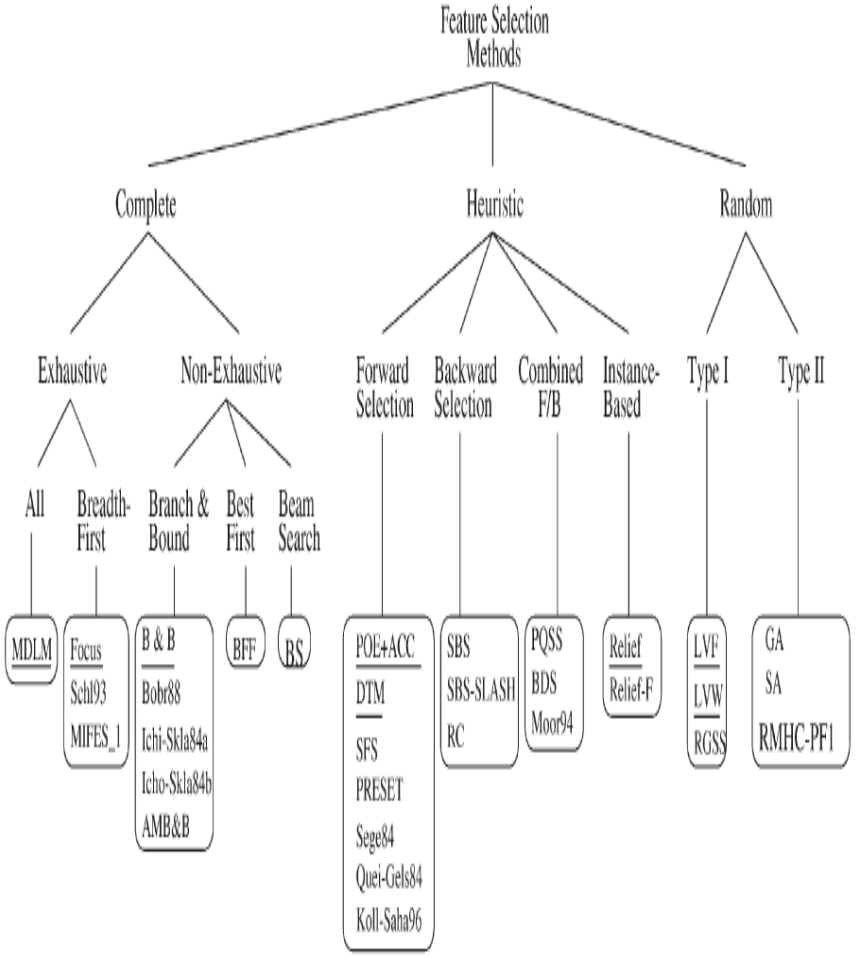
http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=25340E08625
21D41DE72656340CB7E24?doi=10.1.1.39.6038&rep=rep1&type=pdf

Feature selection methods

A big research area!

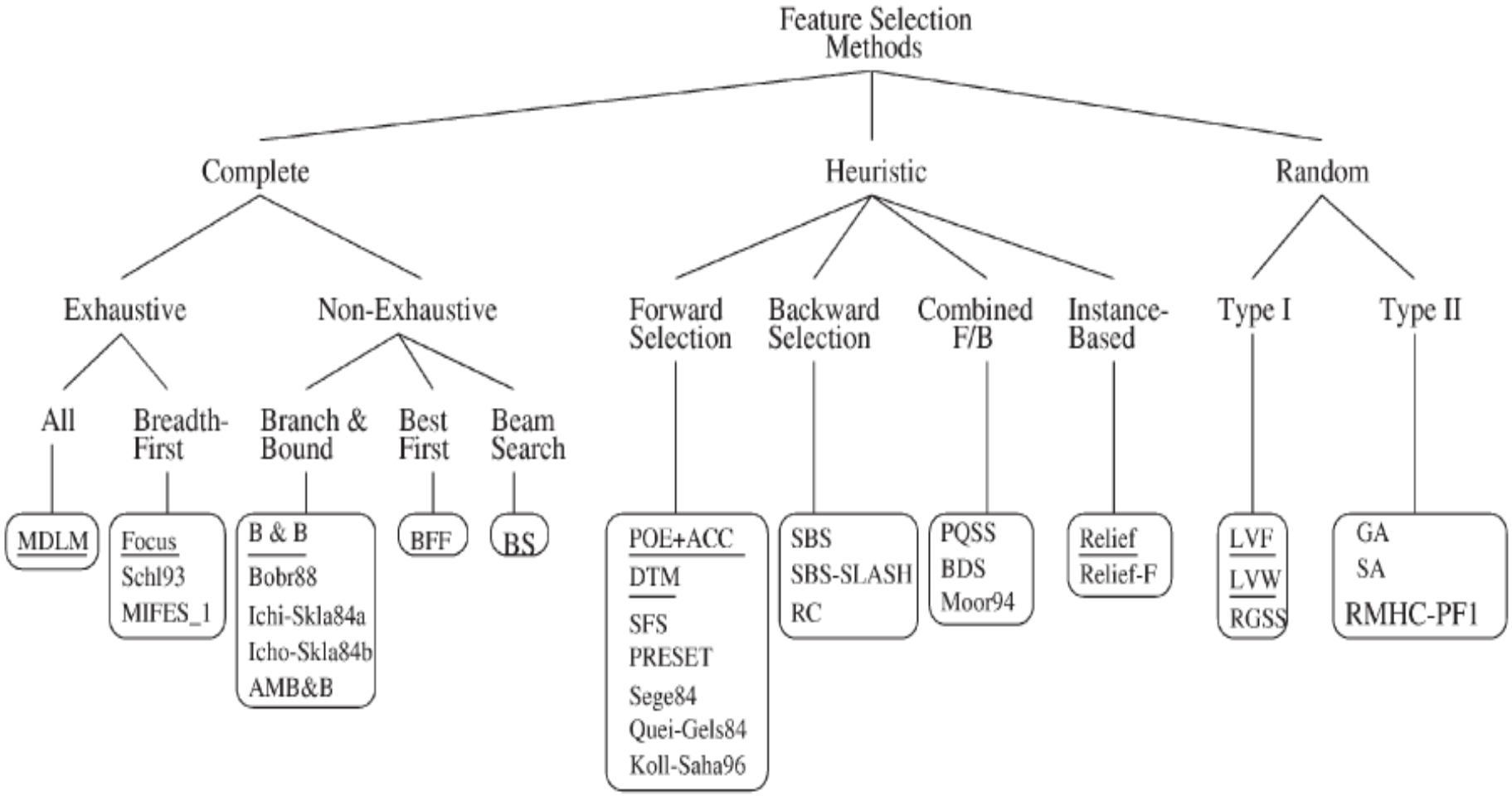
This diagram from (Dash & Liu, 1997)

We'll look briefly at parts of it

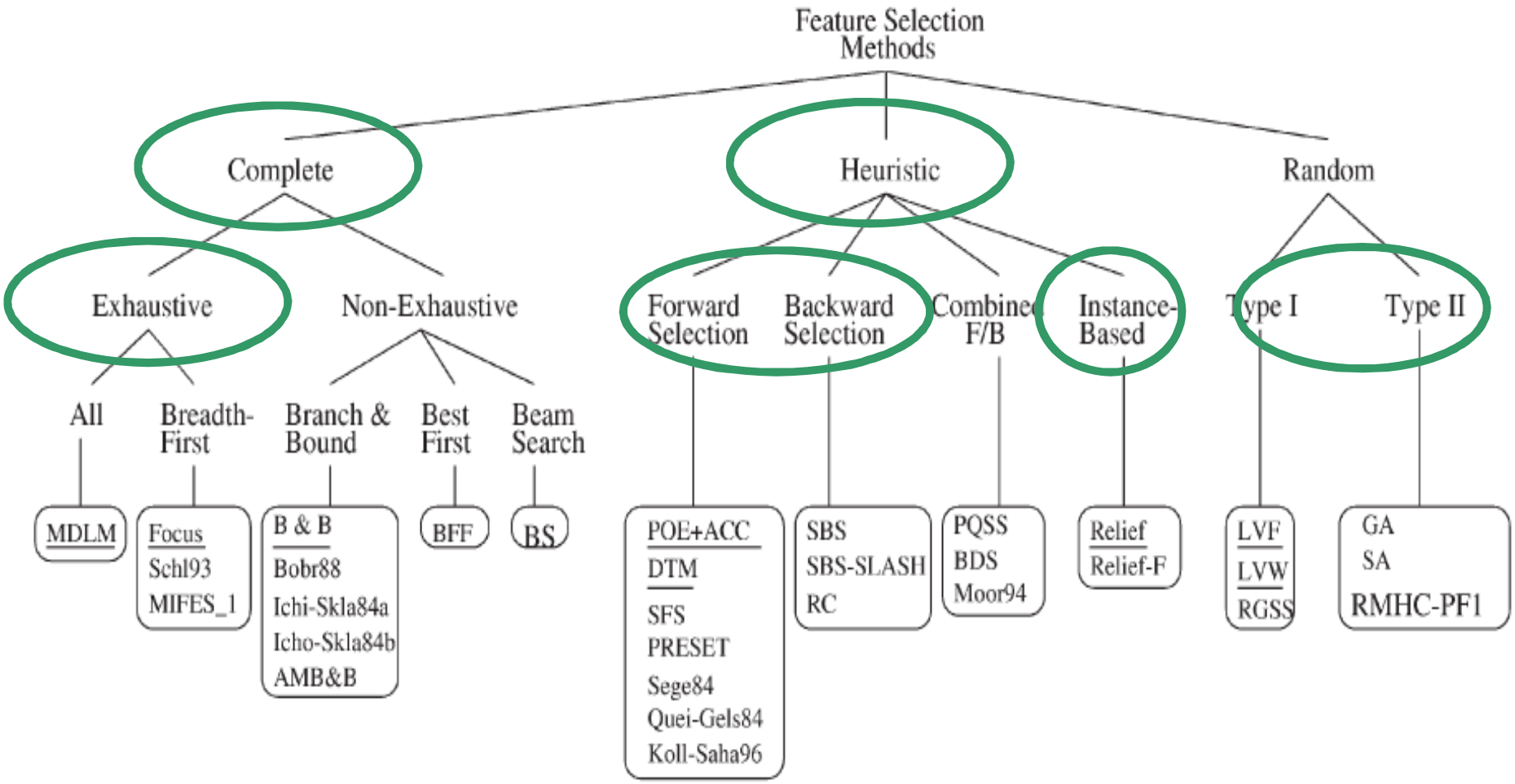


Dash and Liu, 1997
http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=25340E08625
21D41DE72656340CB7E24?doi=10.1.1.39.6038&rep=rep1&type=pdf

Feature selection methods



Feature selection methods



Correlation-based feature ranking

It is indeed used often, by practitioners (who perhaps don't understand the issues involved in FS)

It is actually fine for certain datasets.

It is not even considered in Dash & Liu's survey. Why?

A made-up dataset

f1	f2	f3	f4	…	class
0.4	0.6	0.4	0.6		1
0.2	0.4	1.6	-0.6		1
0.5	0.7	1.8	-0.8		1
0.7	0.8	0.2	0.9		2
0.9	0.8	1.8	-0.7		2
0.5	0.5	0.6	0.5		2

Correlated with the class

f1	f2	f3	f4	…	class
0.4	0.6	0.4	0.6		1
0.2	0.4	1.6	-0.6		1
0.5	0.7	1.8	-0.8		1
0.7	0.8	0.2	0.9		2
0.9	0.8	1.8	-0.7		2
0.5	0.5	0.6	0.5		2

uncorrelated with the class / seemingly random

f1	f2	f3	f4	…	class
0.4	0.6	0.4	0.6		1
0.2	0.4	1.6	-0.6		1
0.5	0.7	1.8	-0.8		1
0.7	0.8	0.2	0.9		2
0.9	0.8	1.8	-0.7		2
0.5	0.5	0.6	0.5		2

Correlation based FS reduces the dataset to this.

f1	f2			…	class
0.4	0.6				1
0.2	0.4				1
0.5	0.7				1
0.7	0.8				2
0.9	0.8				2
0.5	0.5				2

But, col 5 shows us f3 + f4 – which is perfectly correlated with the class!

f1	f2	f3	f4	…	class
0.4	0.6	0.4	0.6	1	1
0.2	0.4	1.6	-0.6	1	1
0.5	0.7	1.8	-0.8	1	1
0.7	0.8	0.2	0.9	1.1	2
0.9	0.8	1.8	-0.7	1.1	2
0.5	0.5	0.6	0.5	1.1	2

Good FS Methods therefore:

- Need to consider how well features work *together*
- As we have noted before, if you take 100 features that are each well correlated with the class, they may simply be correlated strongly with each other, so provide no more information than just one of them

`Complete' methods

Original dataset has N features

You want to use a subset of k features

A *complete* FS method means: try *every* subset of k features, and choose the best!

the number of subsets is $N! / k!(N-k)!$

what is this when N is 100 and k is 5?

`Complete' methods

Original dataset has N features

You want to use a subset of k features

A *complete* FS method means: try *every* subset of k features, and choose the best!

the number of subsets is $N! / k!(N-k)!$

what is this when N is 100 and k is 5?

75,287,520 -- almost nothing

`Complete' methods

Original dataset has N features

You want to use a subset of k features

A *complete* FS method means: try *every* subset of k features, and choose the best!

the number of subsets is $N! / k!(N-k)!$

what is this when N is 10,000 and k is 100?

`Complete' methods

Original dataset has N features

You want to use a subset of k features

A *complete* FS method means: try *every* subset of k features, and choose the best!

the number of subsets is $N! / k!(N-k)!$

what is this when N is 10,000 and k is 100?

5,000,000,000,000,000,000,000,000,000,

000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,

… continued for another 116 slides.

Actually it is around $5 \times 10^{35,101}$

(there are around 10^{80} atoms in the universe)

Can you see a problem with complete methods?

`forward' methods

These methods `grow' a set S of features –

- S starts empty
- Find the best feature to add (by checking which one gives best performance on a test set when combined with S).
- If overall performance has improved, return to step 2; else stop

`backward' methods

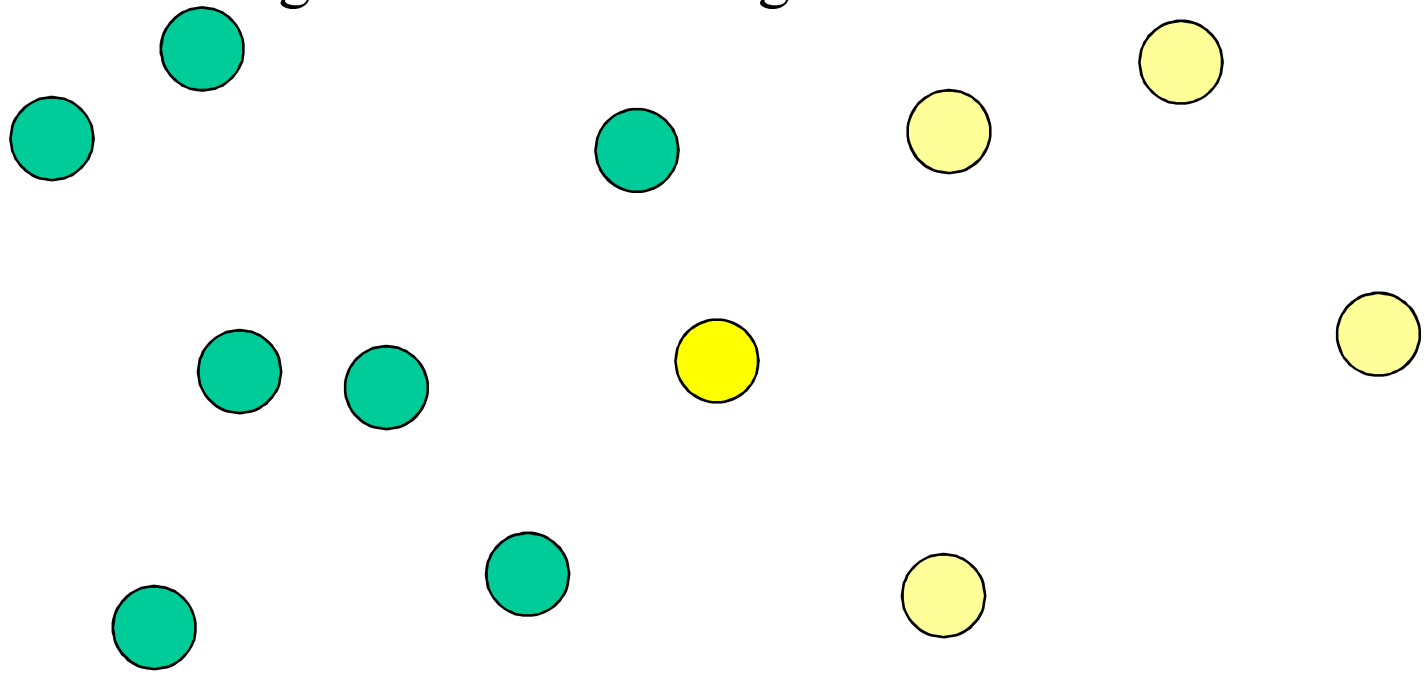
These methods remove features one by one.

- S starts with the full feature set
- Find the best feature to *remove* (by checking which removal from S gives best performance on a test set).
- If overall performance has improved, return to step 2; else stop

- When might you choose forward instead of backward?

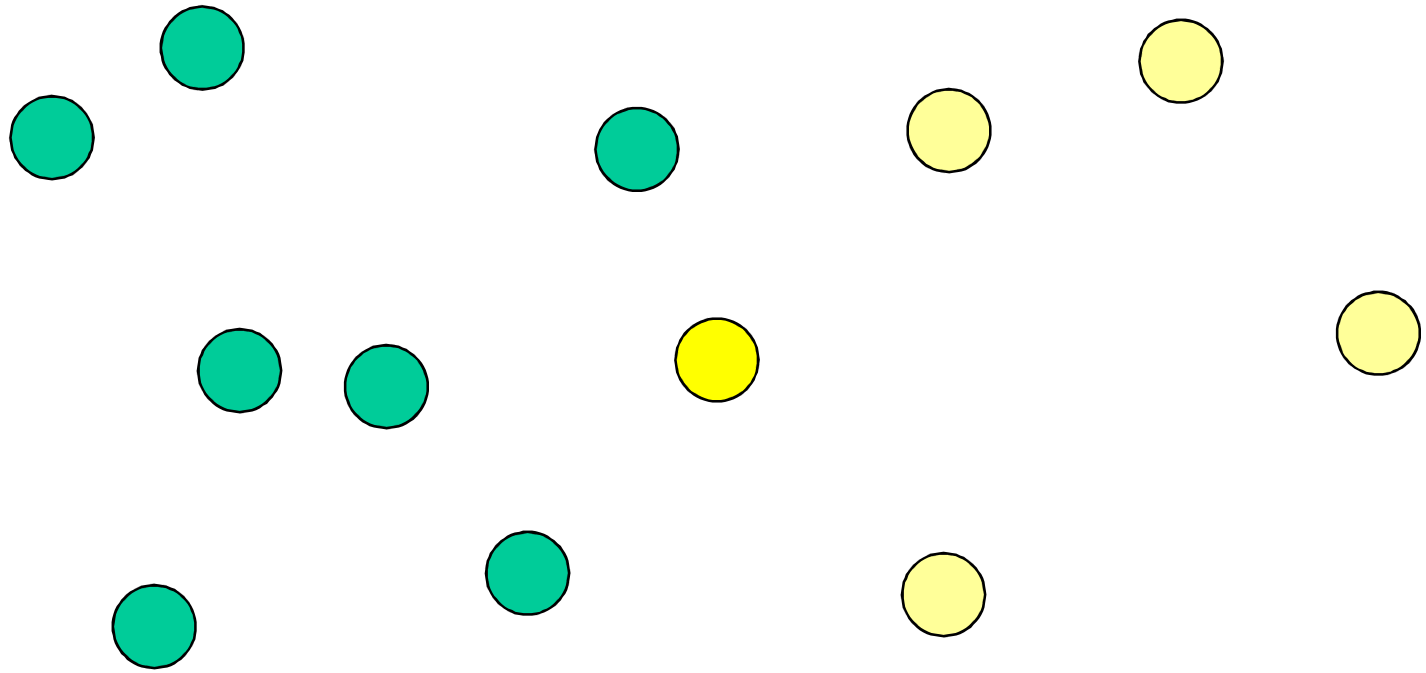
The Relief method

An *instance-based, heuristic* method – it works out weight values for Each feature, based on how important they seem to be in discriminating between near neighbours



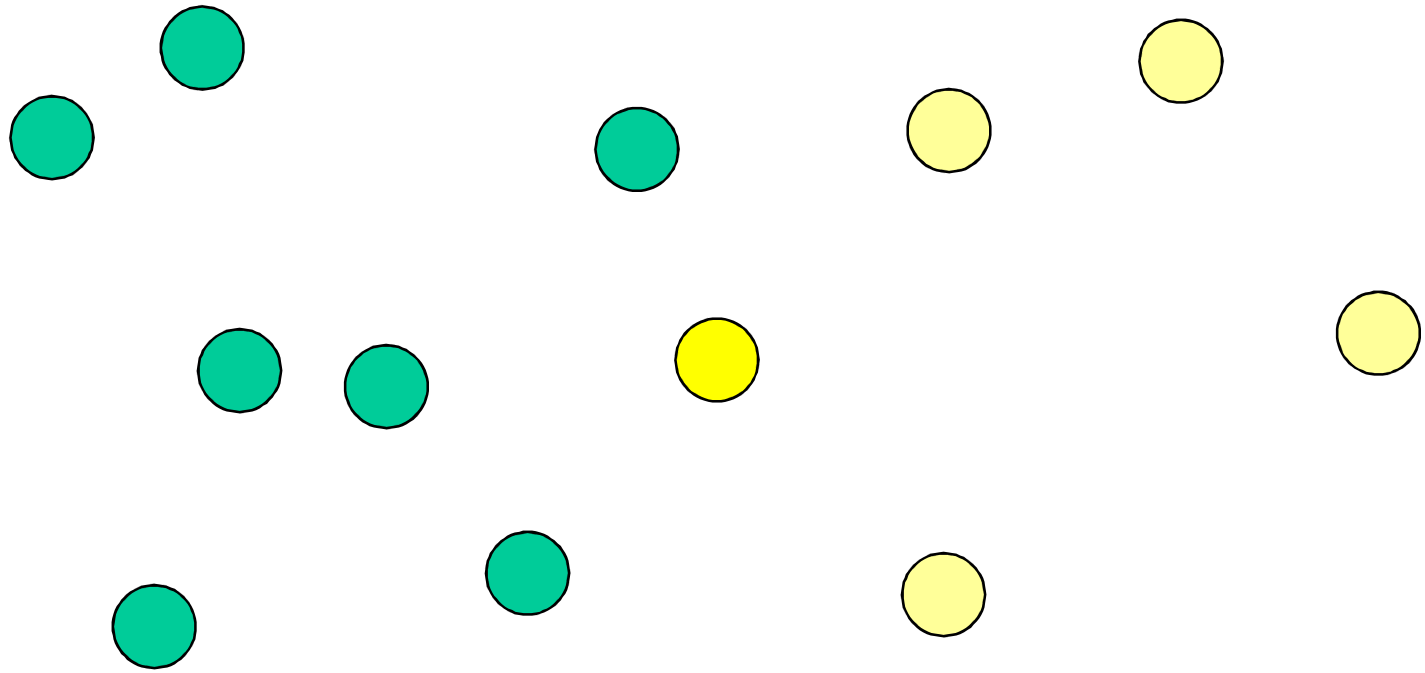
The Relief method

There are two features here – the x and the y co-ordinate
Initially they each have zero weight: wx = 0; wy = 0;



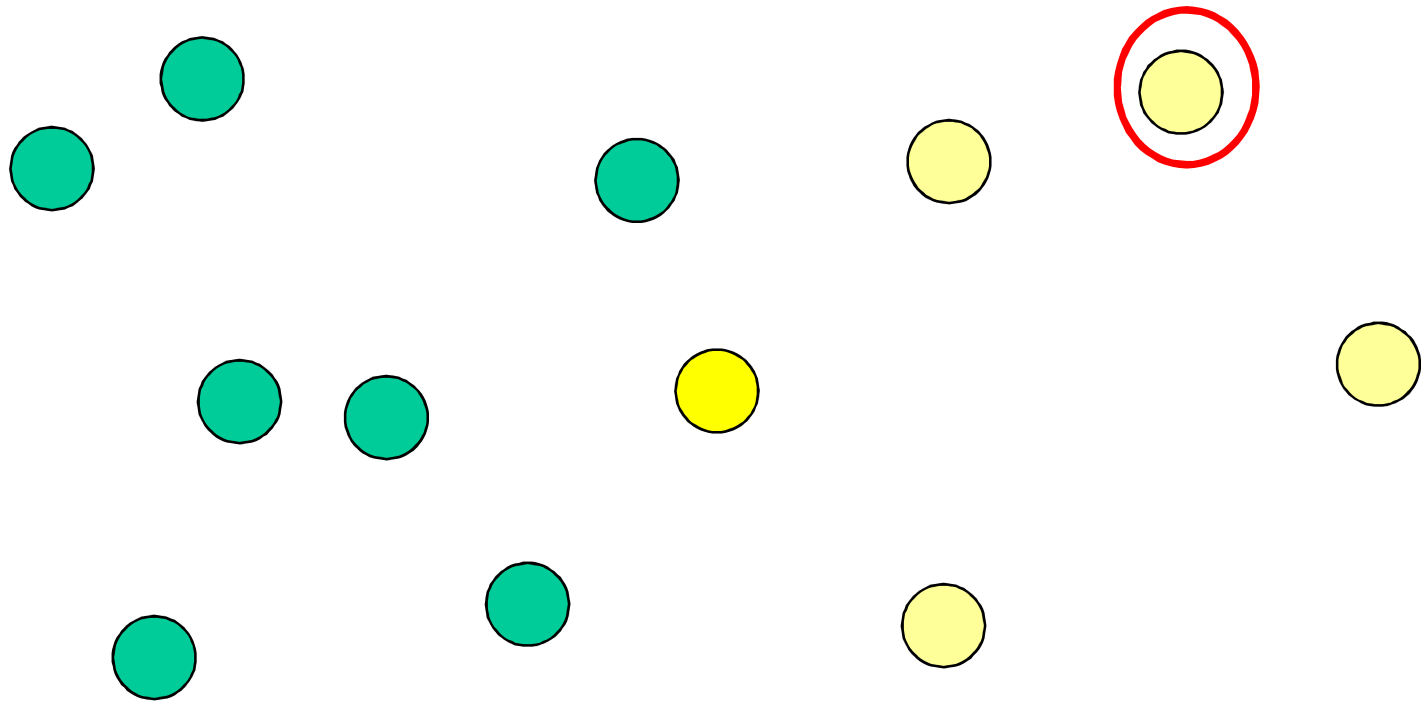
The Relief method

wx = 0; wy = 0; choose an instance at random



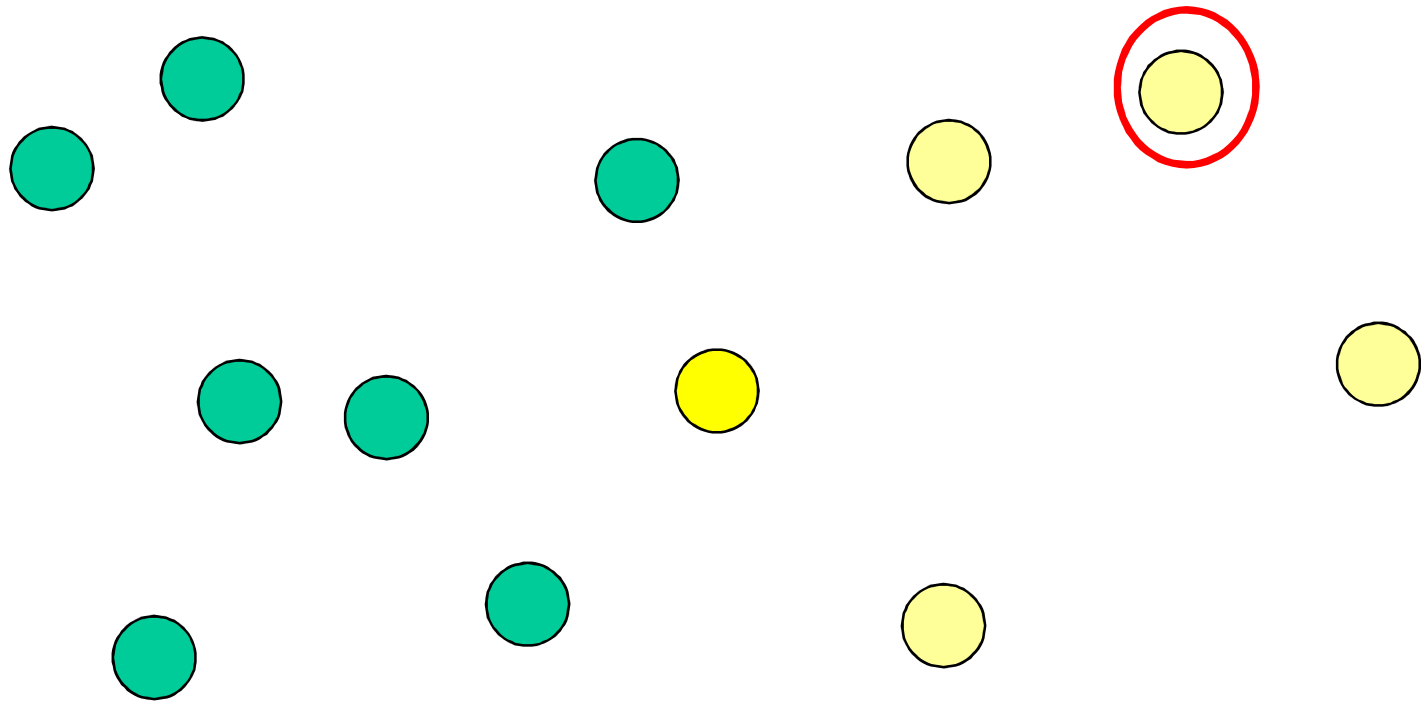
The Relief method

wx = 0; wy = 0; choose an instance at random, call it R



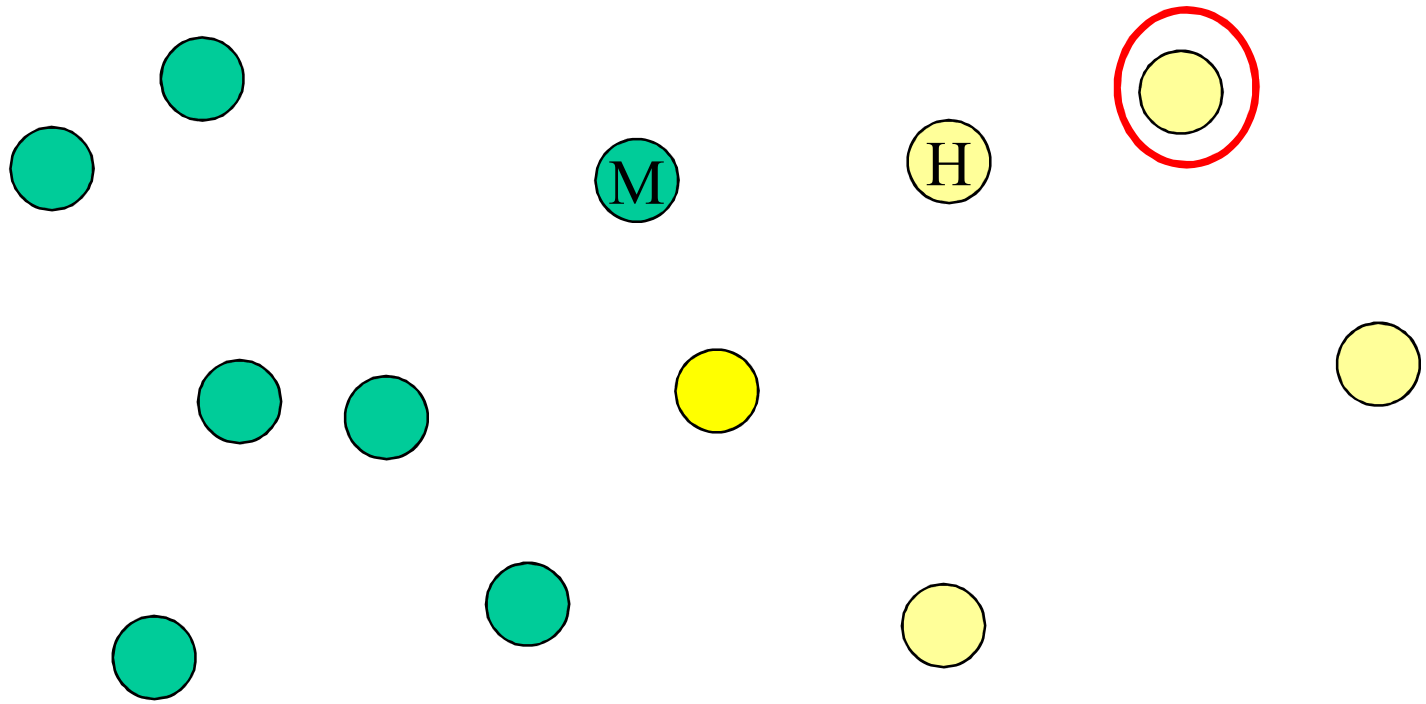
The Relief method

wx = 0; wy = 0; find H (hit: the nearest to R of the same class) and M (miss: the nearest to R of different class)



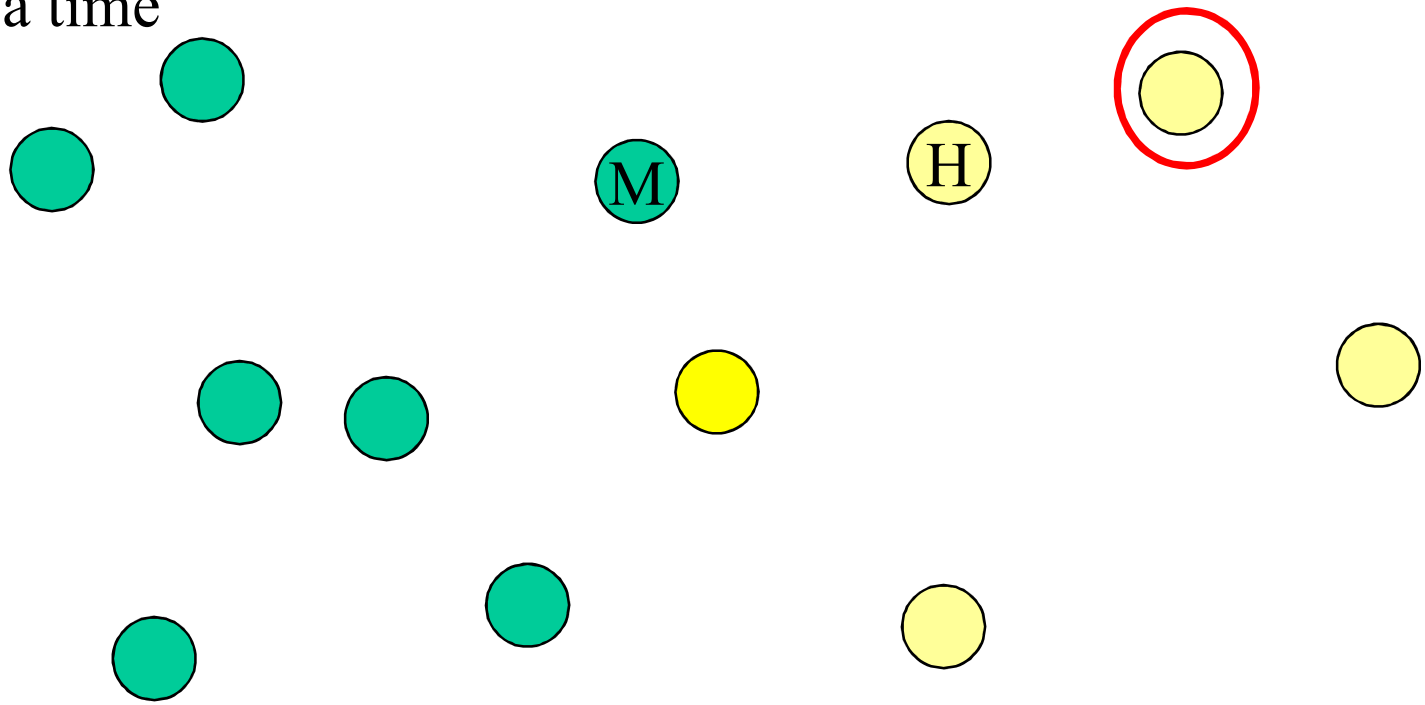
The Relief method

wx = 0; wy = 0; find H (hit: the nearest to R of the same class) and M (miss: the nearest to R of different class)



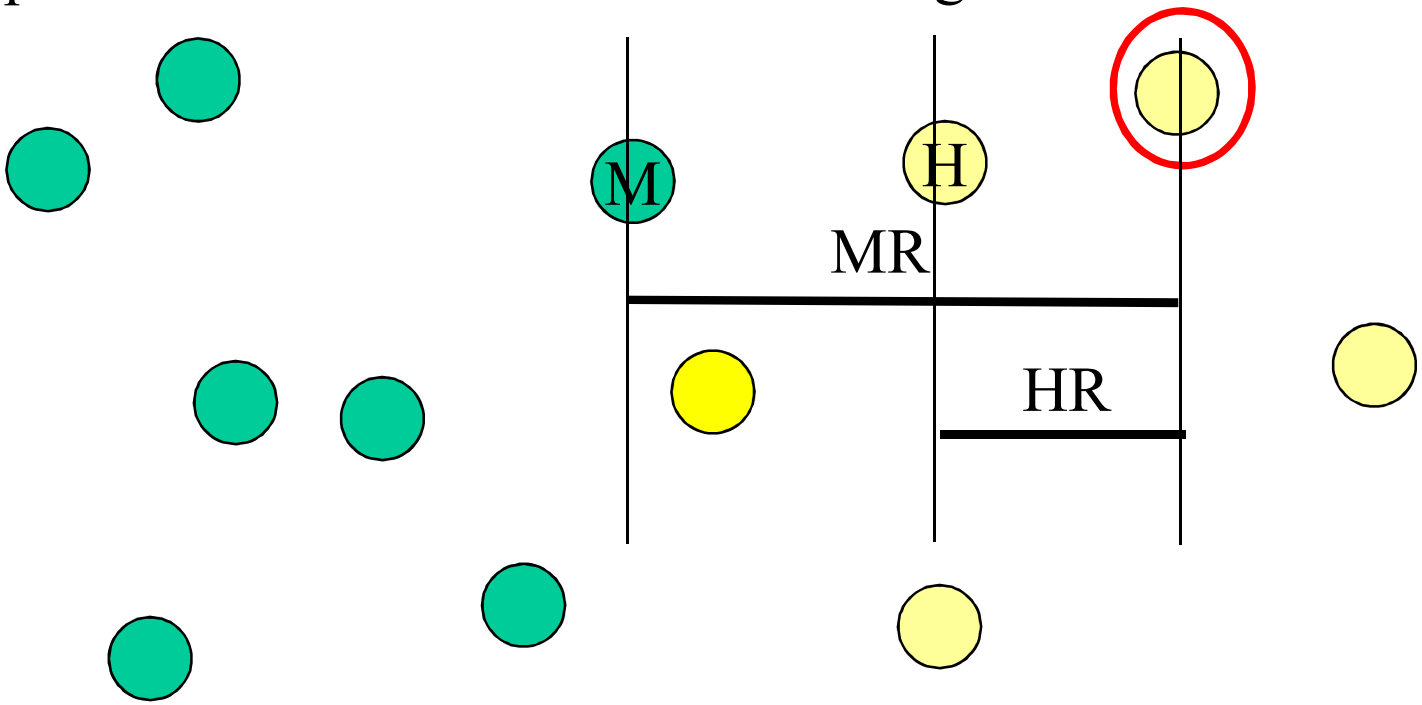
The Relief method

wx = 0; wy = 0; now we update the weights based on the distances between R and H and between R and M. This happens one feature at a time



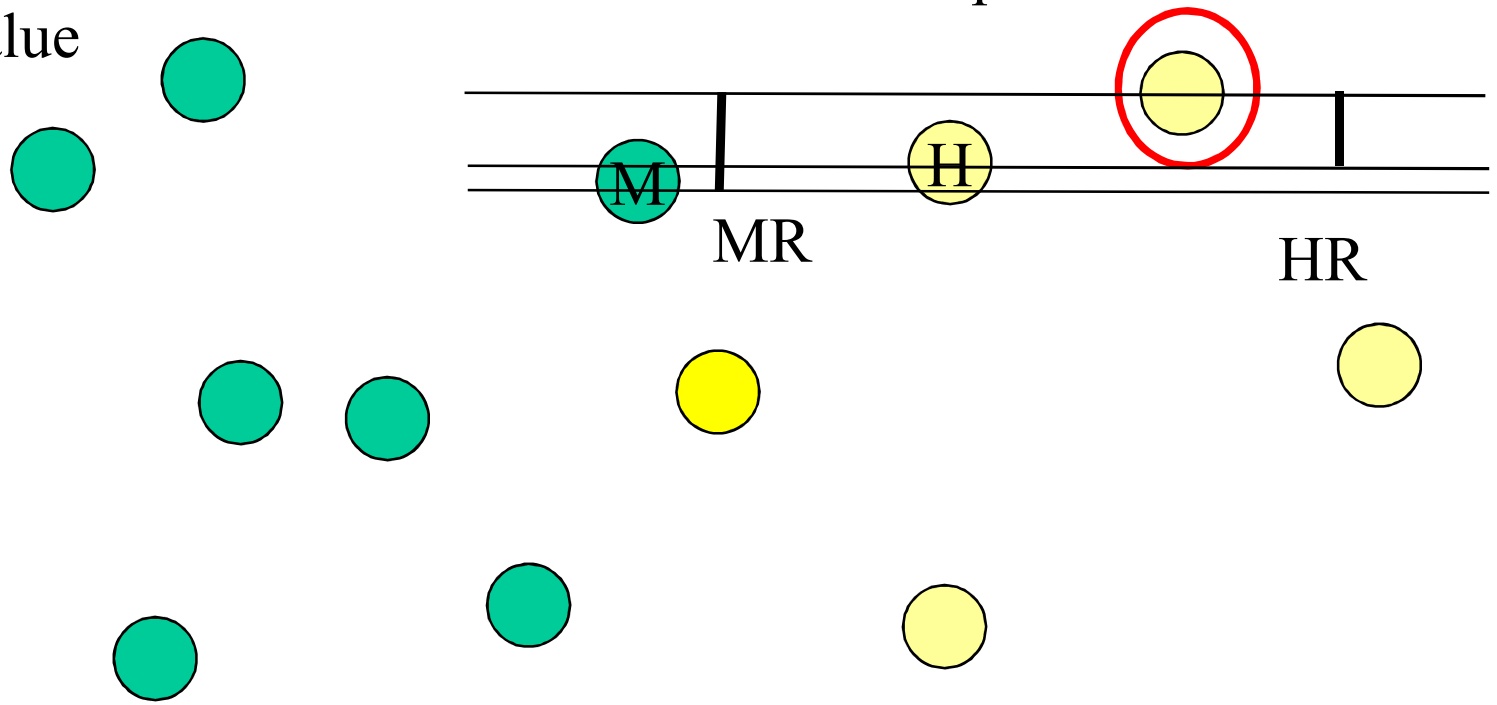
The Relief method

To change wx, we add to it: (MR − HR)/*n* ; so, the further the `miss' in the *x* direction, the higher the weight of *x* – the more important *x* is in terms of discriminating the classes



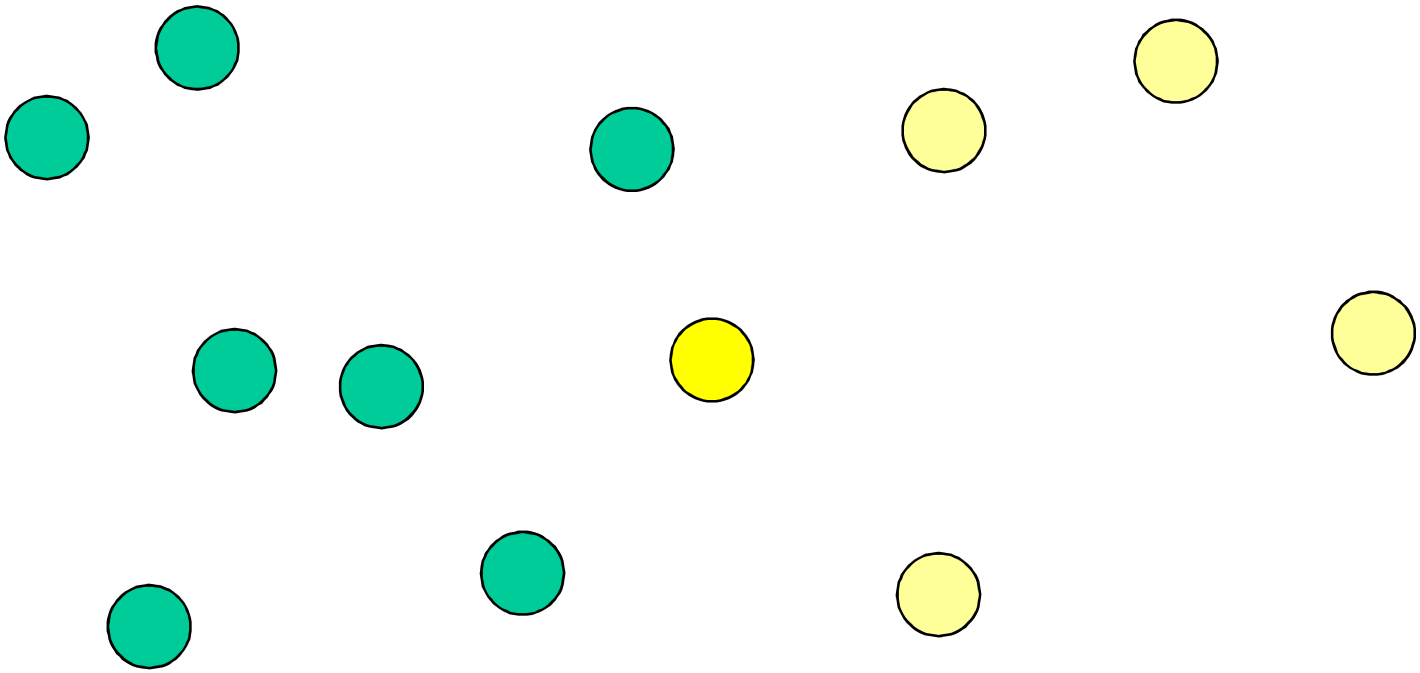
The Relief method

To change wy, we add to it: (MR − HR)/*n* again, but this time calculated in the *y* dimension; clearly the difference is smaller; differences in this feature don't seem important in terms of class value



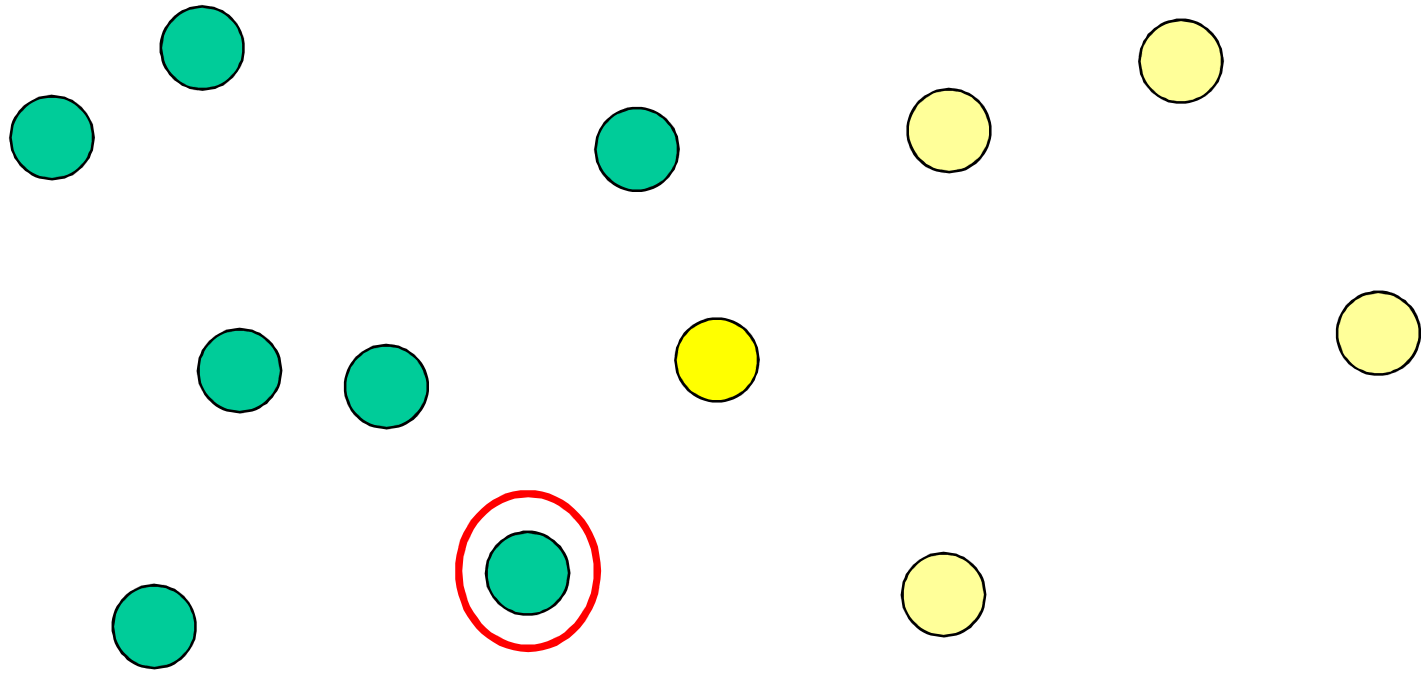
The Relief method

Maybe now we have wx = 0.07, wy = 0.002.



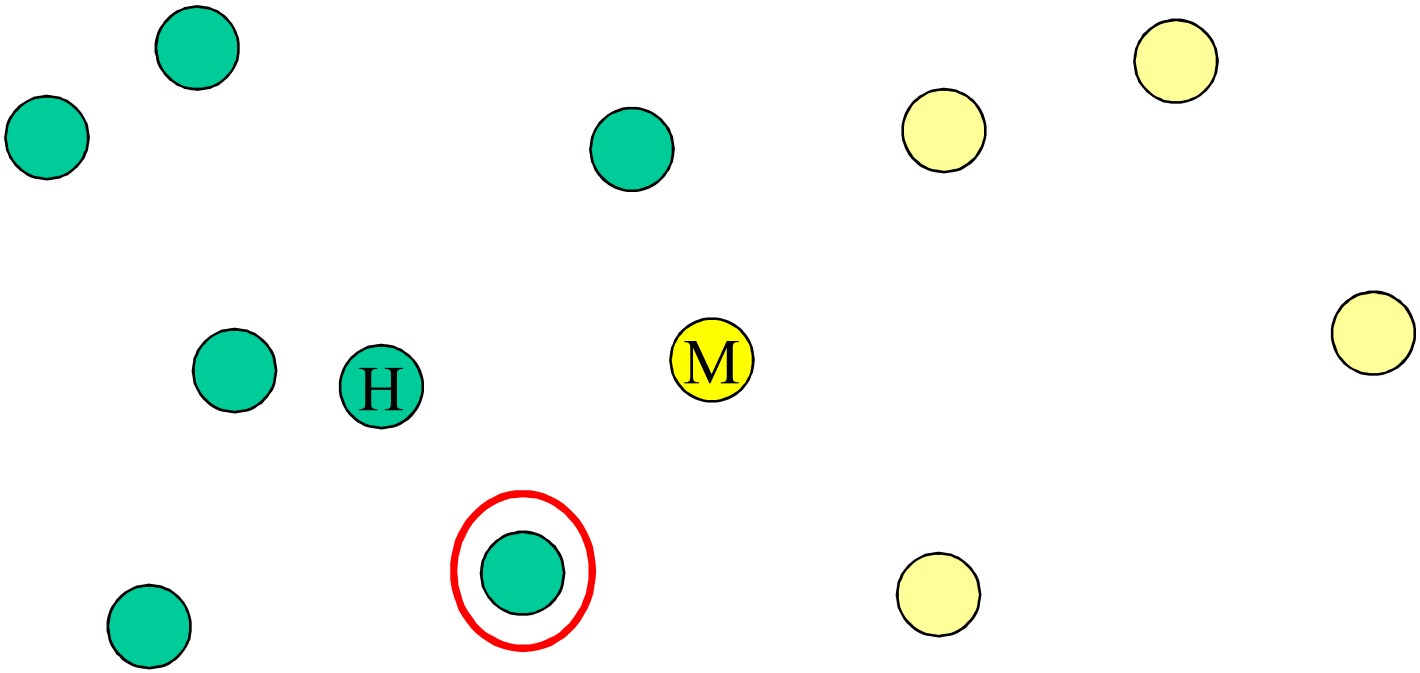
The Relief method

wx = 0.07, wy = 0.002;
Pick another instance at random, and do the same again.



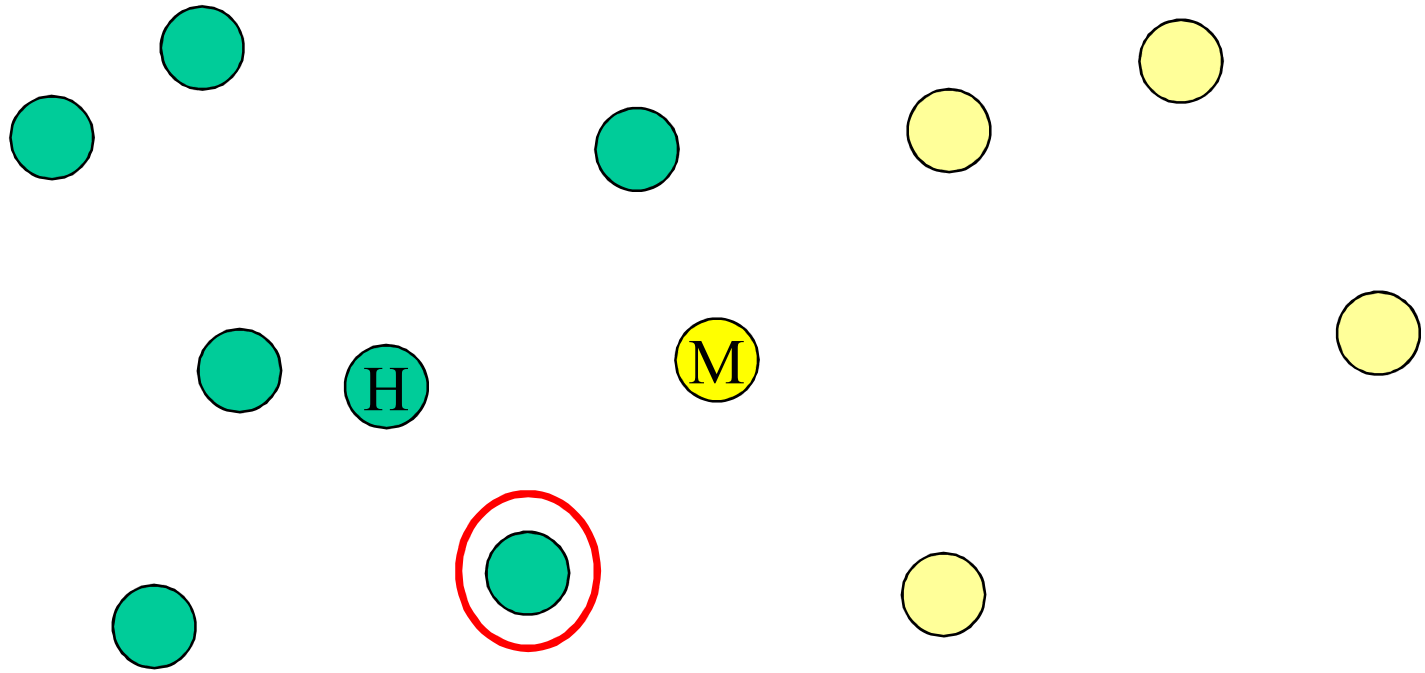
The Relief method

wx = 0.07, wy = 0.002;
Identify H and M



The Relief method

wx = 0.07, wy = 0.002;
Add the HR and MR differences divided by *n*, for each feature, again …



The Relief method

In the end, we have a weight value for each feature. The higher the value, the more relevant the feature.

We can use these weights for feature selection, simply by choosing the features with the *S* highest weights (if we want to use *S* features)

NOTE
-It is important to use Relief F only on min-max normalised data in [0,1]. However it is fine if category attibutes are involved, in which case use Hamming distance for those attributes,
-Why divide by *n*? Then, the weight values can be interpreted as a *difference in probabilities*.

The Relief method, plucked directly from the original paper (Kira and Rendell 1992)

```
Relief(S, m, τ)
    Separate S into S+ = {positive instances} and
        S- = {negative instances}
    W = (0, 0, . . . , 0)
    For i = 1 to m
        Pick at random an instance X ∈ S
        Pick at random one of the positive instances
            closest to X, Z+ ∈ S+
        Pick at random one of the negative instances
            closest to X, Z- ∈ S-
        if (X is a positive instance)
            then   Near-hit = Z+; Near-miss = Z-
            else   Near-hit = Z-; Near-miss = Z+
        update-weight(W, X, Near-hit, Near-miss)
    Relevance = (1/m)W
    For i = 1 to p
        if (relevance_i ≥ t)
            then   f_i is a relevant feature
            else   f_i is an irrelevant feature

update-weight(W, X, Near-hit, Near-miss)
    For i = 1 to p
        W_i = W_i - diff(x_i, near-hit_i)^2 + diff(x_i, near-miss_i)^2
```

Random(ised) methods
aka Stochastic methods

Suppose you have 1,000 features.

There are 2^{1000} possible subsets of features.

One way to try to find a good subset is to run a **stochastic search** algorithm

E.g. Hillclimbing, simulated annealing, genetic algorithm, particle swarm optimisation, …

One slide introduction to (most) stochastic search algorithms

A search algorithm:

BEGIN:

1. initialise a random population *P* of *N* candidate solutions (maybe just 1) (e.g. each solution is a random subset of features)
2. Evaluate each solution in *P* (e.g. accuracy of 3-NN using only the features in that solution)

ITERATE:

1. generate a set *C* of new solutions, using the good ones in *P* (e.g. choose a good one and mutate it, combine bits of two or more solutions, etc …)
2. evaluate each of the new solutions in *C*.
3. Update *P* – e.g. by choosing the best *N* from all of *P* and *C*
4. If we have iterated a certain number of times, or accuracy is good enough, stop

One slide introduction to (most) stochastic search algorithms

A search algorithm:

BEGIN:

1. initialise a random population P of N candidate solutions (maybe just 1) (e.g. each solution is a random subset of features)
2. Evaluate each solution in P (e.g. accuracy of 3-NN using only the features in that solution)

ITERATE:

1. generate a set C of new solutions, using the good ones in P (e.g. choose a good one and mutate it, combine bits of two or more solutions, etc …)
2. evaluate each of the new solutions in C.
3. Update P – e.g. by choosing the best N from all of P and C
4. If we have iterated a certain number of times, or accuracy is good enough, stop

GENERATE

TEST

GENERATE

TEST

UPDATE

Why randomised/search methods are good for FS

Usually you have a large number of features (e.g. 1,000)

You can give each feature a score (e.g. correlation with target, Relief weight, etc …), and choose the best-scoring features. This is very fast.

However this does not evaluate how well features work with other features. You could give *combinations* of features a score, but there are too many combinations of multiple features.

Search algorithms are the only suitable approach that get to grips with evaluating combinations of features.