Visual Pattern Recognition: pattern classification and multiple classifier systems

Image Processing — scc0251

www.icmc.usp.br/∼moacir — moacir@icmc.usp.br

ICMC/USP — São Carlos, SP, Brazil

2011

Agenda

1. A classification task
2. Classification: basics
3. How to build a classifier
4. Logistic Regression Classifier
   - Cost Function
5. k-Nearest Neighbor Classifier
6. Multiple Classifier Systems
   - Creating ensembles of classifiers
   - Combining multiple classifiers
   - Diversity and Ensemble Performance

Agenda

A classification task

A problem — given two classes of images:

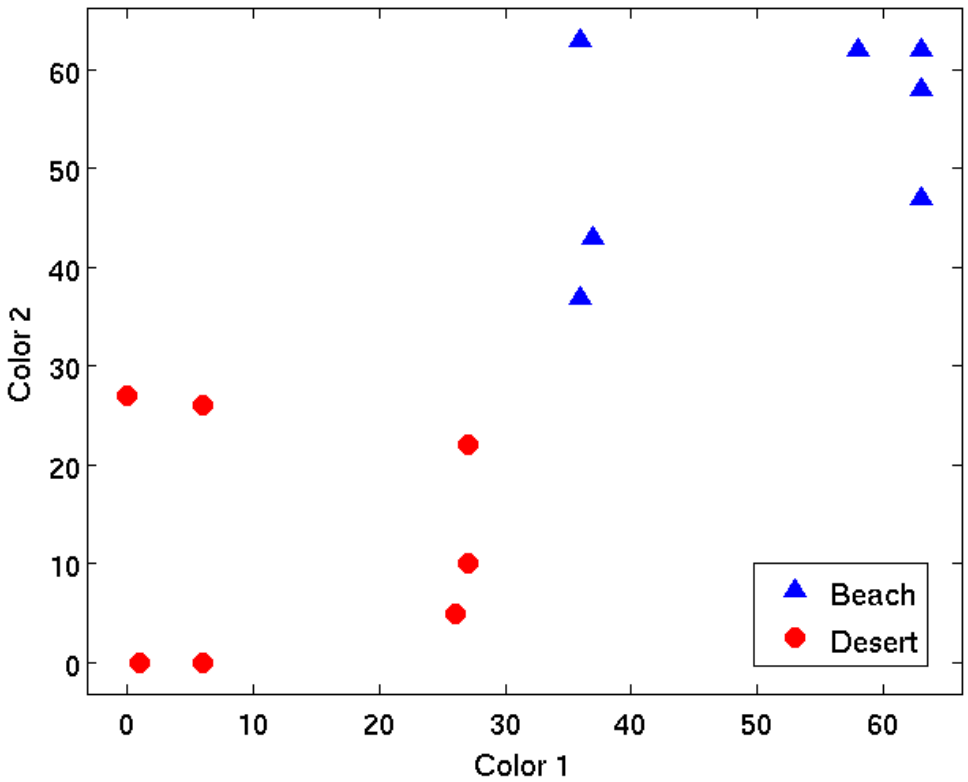
- class 1: pictures taken from **deserts**, and
- class 2: images taken from **beaches**,

develop a method able to classify a new, unknown image, into one of these two classes.

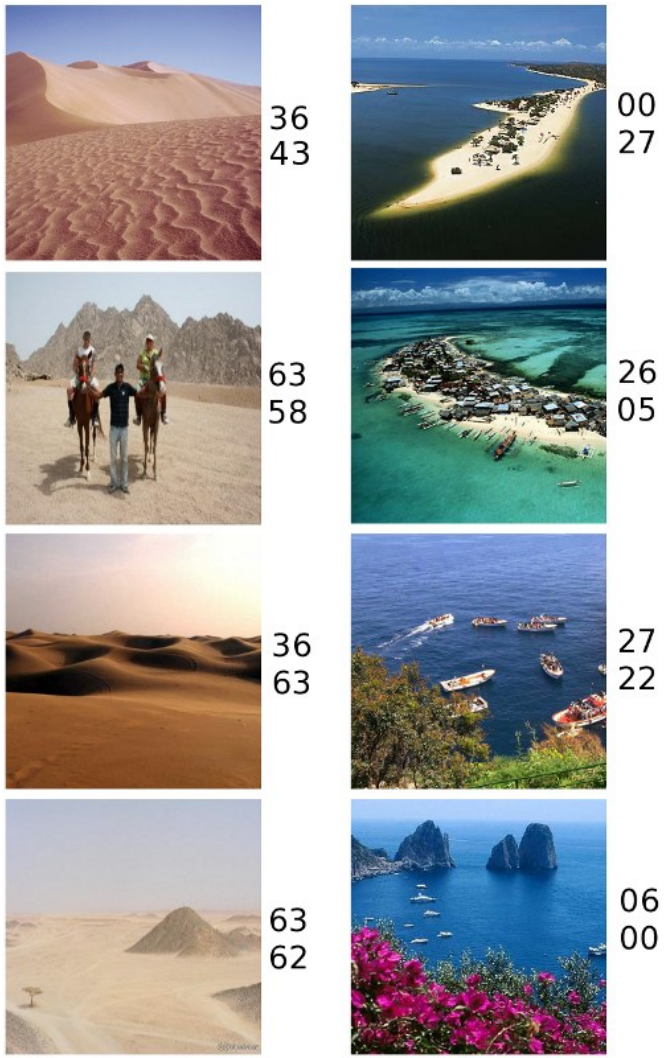
- **Object**: each image.

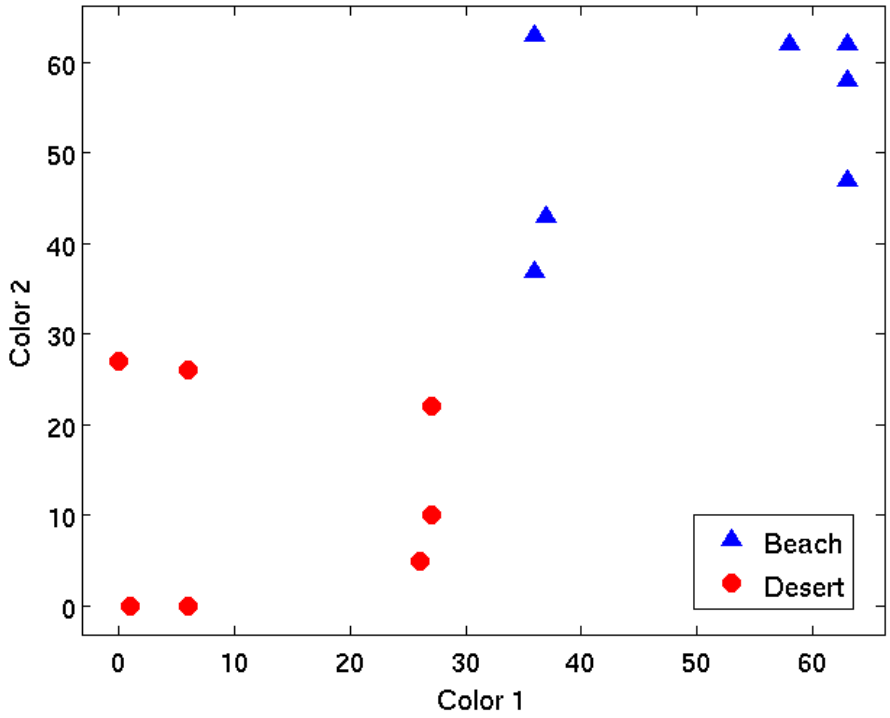
A classification task

- **Feature**: values extracted from images that can be used to compute a (dis)similarity between two images. **Suggestions?**
  - The two most dominant colors after reducing the colors to 64. Each color is considered a feature. In this case the feature space is 2D.



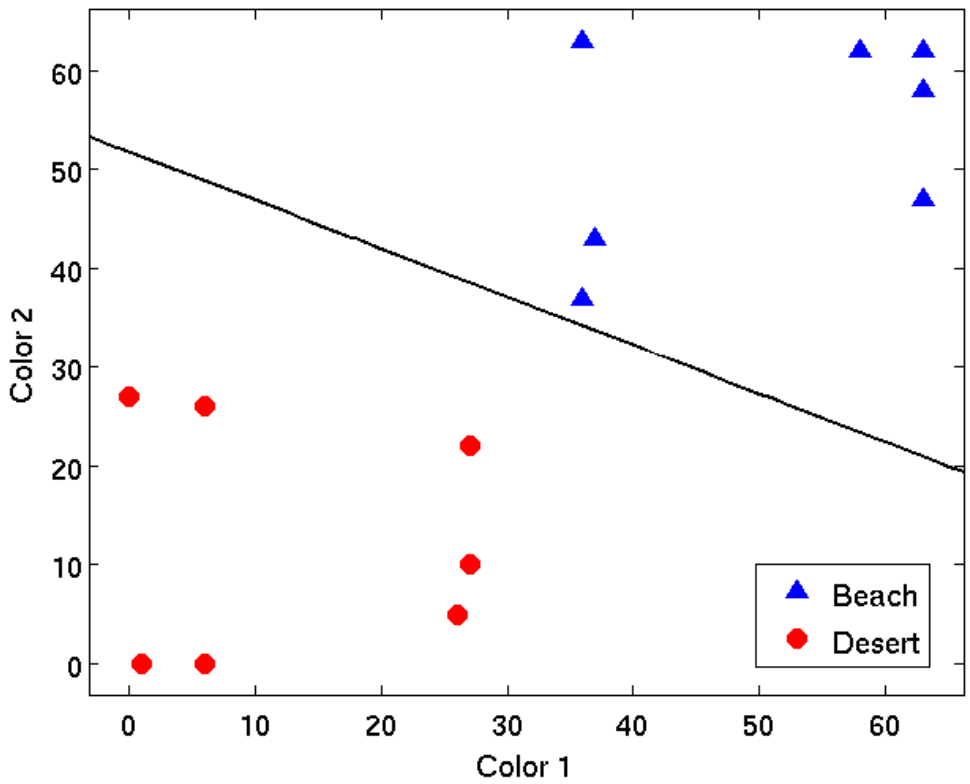
A classification task





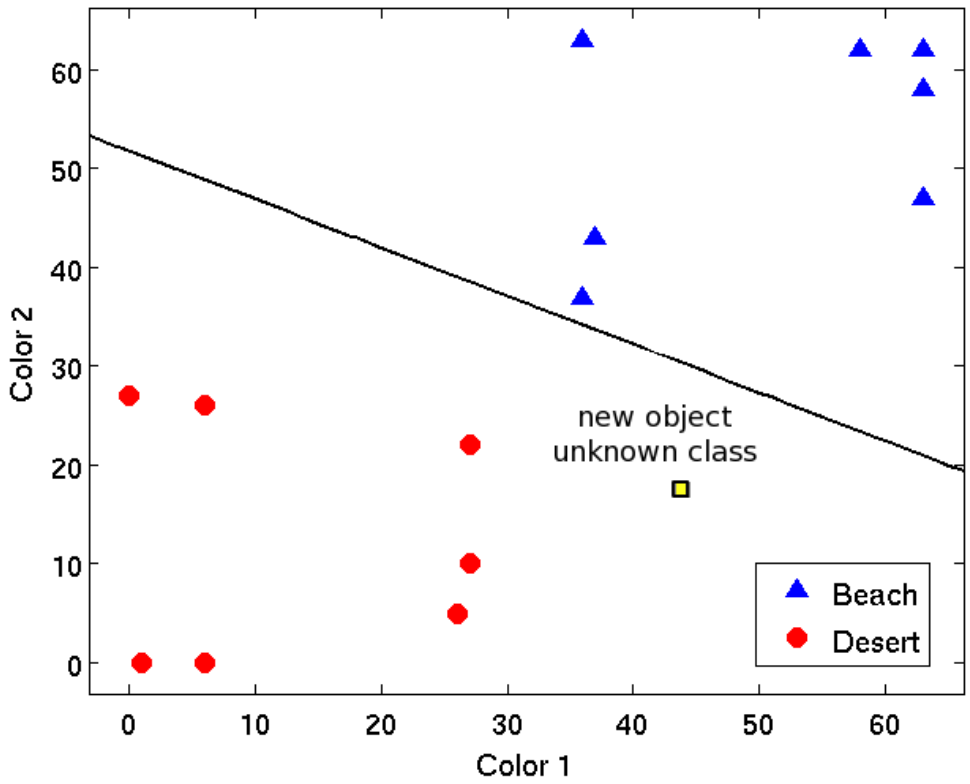
A classification task

- **Classifier**: a model build using a set of features extracted from images, for which the class is known. This model is capable of "predict" the class of a new, unknown image. **Suggestions?**
  - Estimate a decision boundary based on the distribution of the objects.



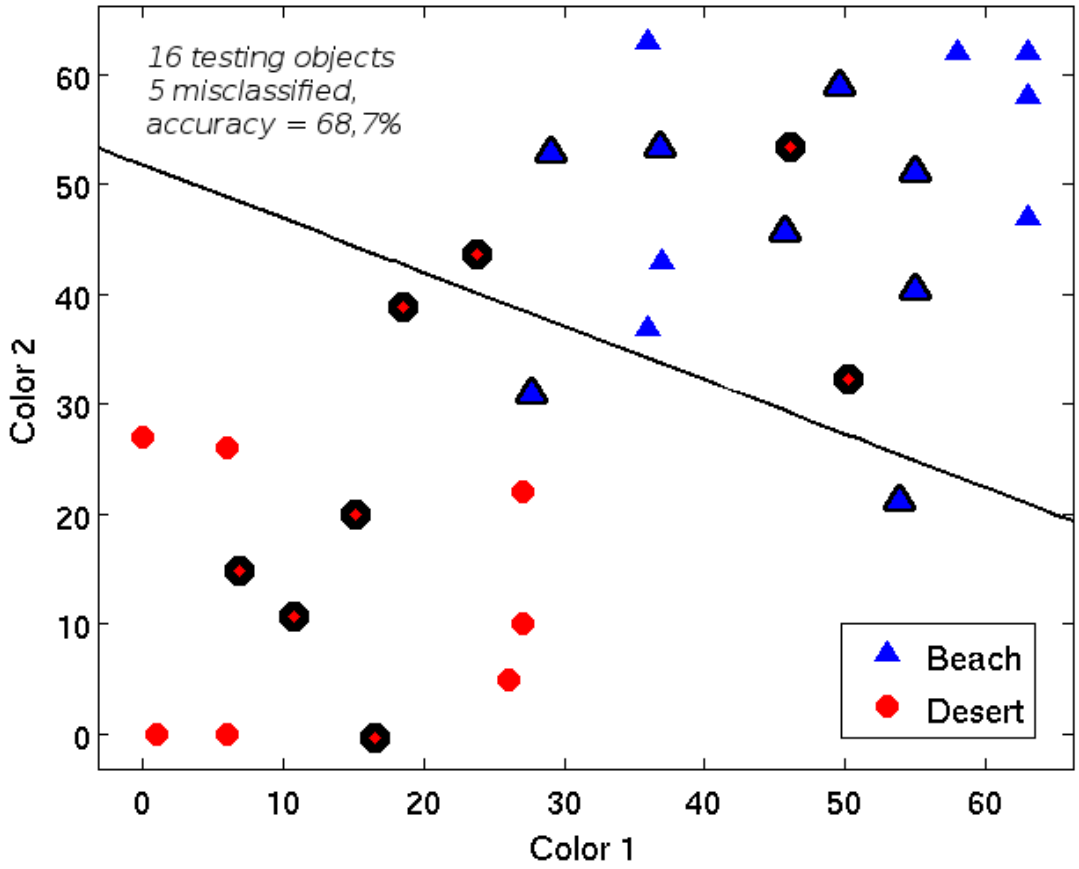
A classification task

- The set of objects used to build the classifier is called **training set**.
- The idea is to use the **trained classifier** in order to classify any **new object**. This is related to the concept of generalization.



A classification task

- Once the classifier is ready, a set of additional objects (not used for training) is used to test the accuracy of the classifier.
- This set of objects, for which the classes are known, is called **test set**.



Agenda

Basics: notation

Classes: groups of similar objects, $\Omega = \{\omega_1, \omega_2, ..., \omega_c\}$

Dataset: $\mathbf{X} = \{x_1, x_2, ..., x_N\}$, for $x_i \in \mathbb{R}^M$

$\mathbf{x}_i \in \mathbb{R}^M$ an **object** in the feature space, represented by a *feature vector*

$l(\mathbf{x}_i) = y_i \in \Omega$ the **labels** assigned to the object

the matrix N objects $\times$ M features:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,M} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,M} \\ \cdots & \cdots & & \cdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,M} \end{bmatrix}, \text{ labels} = \begin{bmatrix} l(\mathbf{s}_1) = y_1 \\ l(\mathbf{s}_2) = y_2 \\ \cdots \\ l(\mathbf{s}_N) = y_N \end{bmatrix}$$

Agenda

How to build a classifier

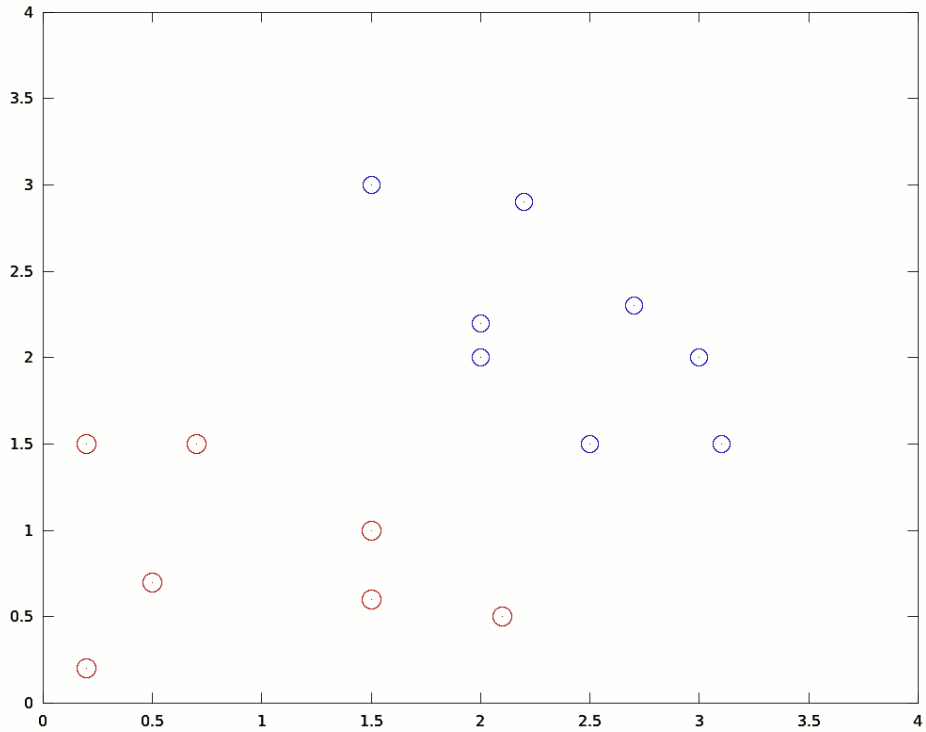
Classifier: is any function: $H : \mathbb{R}^N \rightarrow \Omega$, for $\Omega = \{\omega_1, \omega_2, ..., \omega_c\}$

Parameterized Hypotesis: $h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$.

- uses parameters $\theta_0, \theta_1, ...\theta_M$, where M is the number of features.
- example using 2 features: $h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$.
- by finding θ we can develop a function that can separate the classes in the feature space, so that we can classify $\mathbf{x}$ (considering 2 classes):

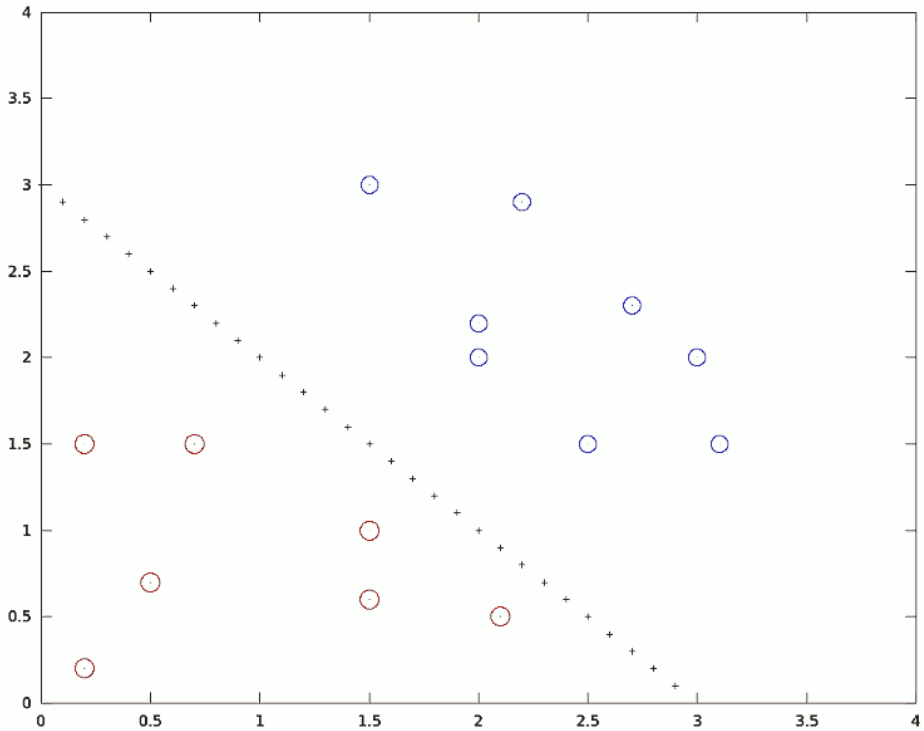
$$y_i = I(x_i) = \begin{cases} 1 & \text{if } h_\theta(\mathbf{x}) \geq 0 \\ 0 & \text{if } h_\theta(\mathbf{x}) < 0 \end{cases}$$

How to build a classifier



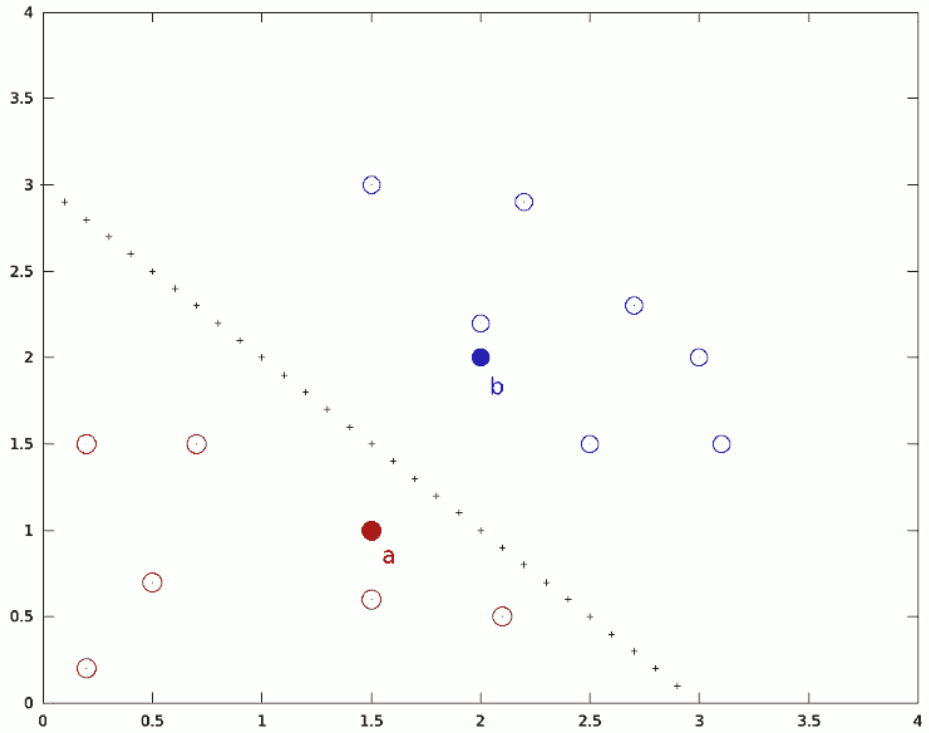
Which parameters should we use to separate the two classes?

How to build a classifier



$$\theta = [-3, 1, 1], \quad h_\theta(\mathbf{x}) = -3 + 1x_1 + 1x_2.$$

How to build a classifier

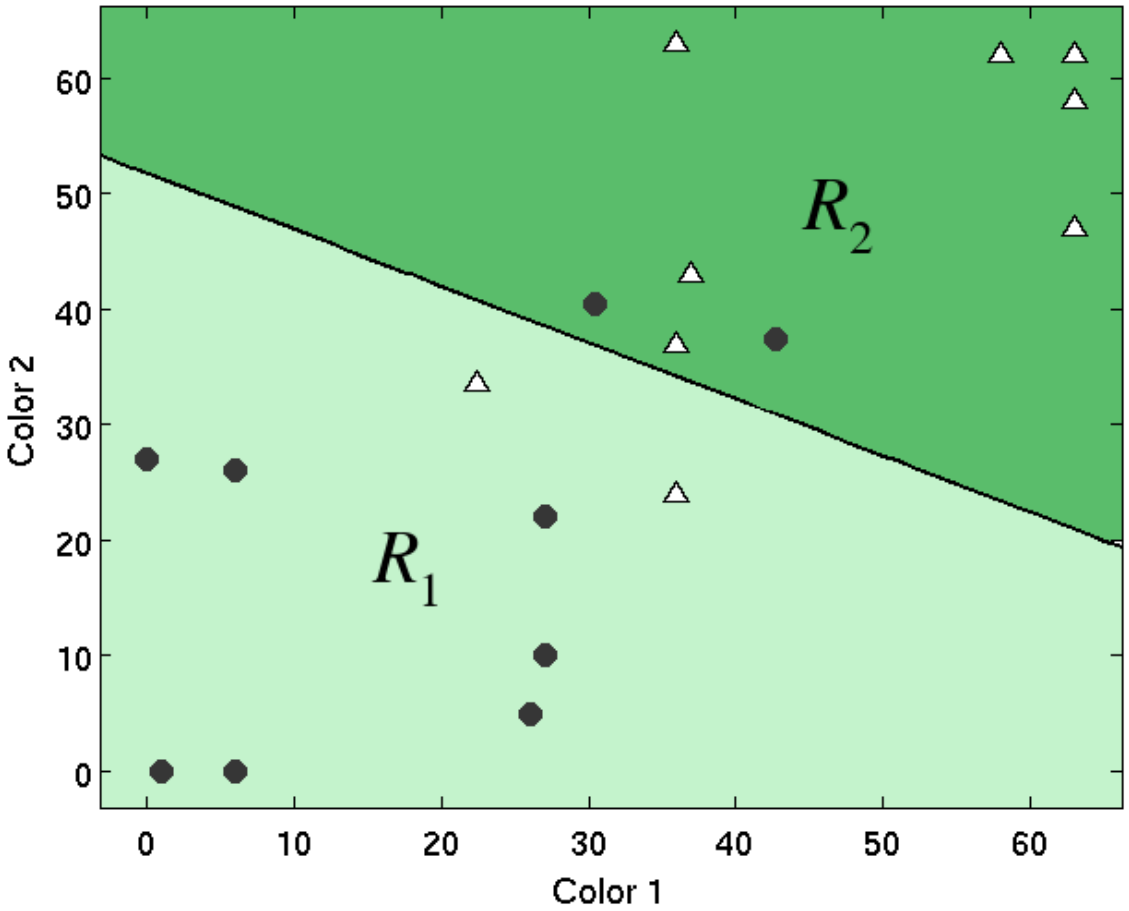


$a = (1.5, 1)$
$h_\theta(a) = -3 + 1.5 + 1 = -0.5$

$b = (2, 2.2)$
$h_\theta(b) = -3 + 2 + 2.2 = 1.2$

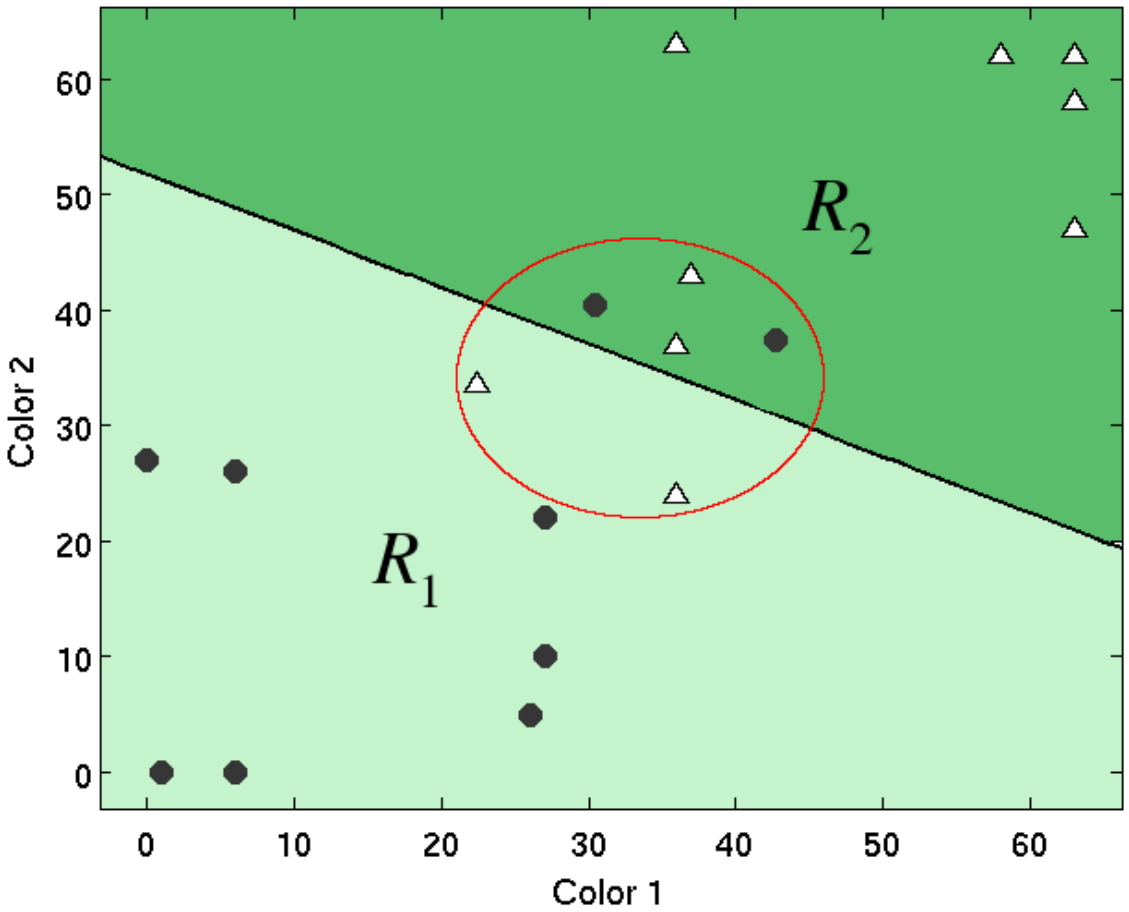
How to build a classifier

The discriminant functions partition the feature space into decision regions: $R_1, R_2, ..., R_c$, forming a **decision boundary**.



How to build a classifier

Often there is an error on the classifier:



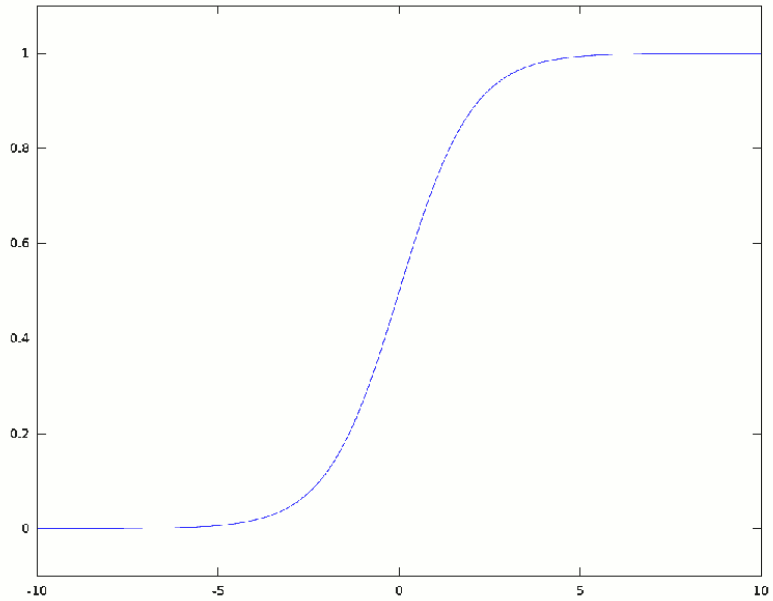
Agenda

The Logistic Regression Classifier

In order to compute values in the range $[0, 1]$ and achieve a better control of the boundaries, a logistic (or sigmoid) function can be used to evaluate the hypotesis:

$$h_\theta(\mathbf{x}) = g(\theta^T \mathbf{x})$$

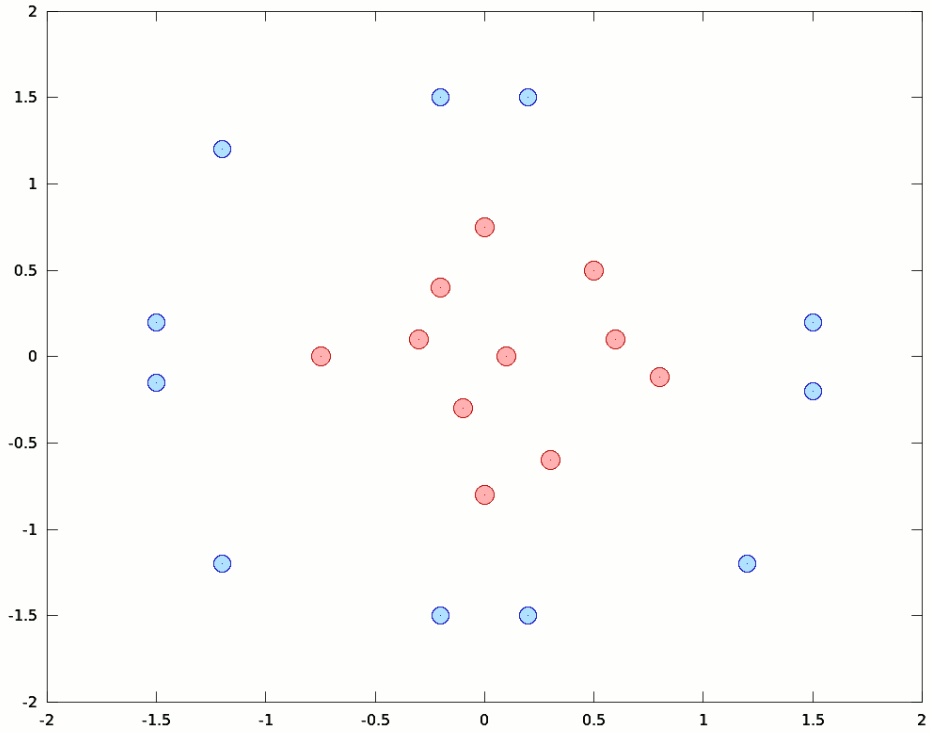
$$g(z) = \frac{1}{1 + e^{-z}}$$



$$y_i = I(x_i) = \begin{cases} 1 & \text{if } h_\theta(\mathbf{x}) \geq 0.5 \\ 0 & \text{if } h_\theta(\mathbf{x}) < 0.5 \end{cases}$$

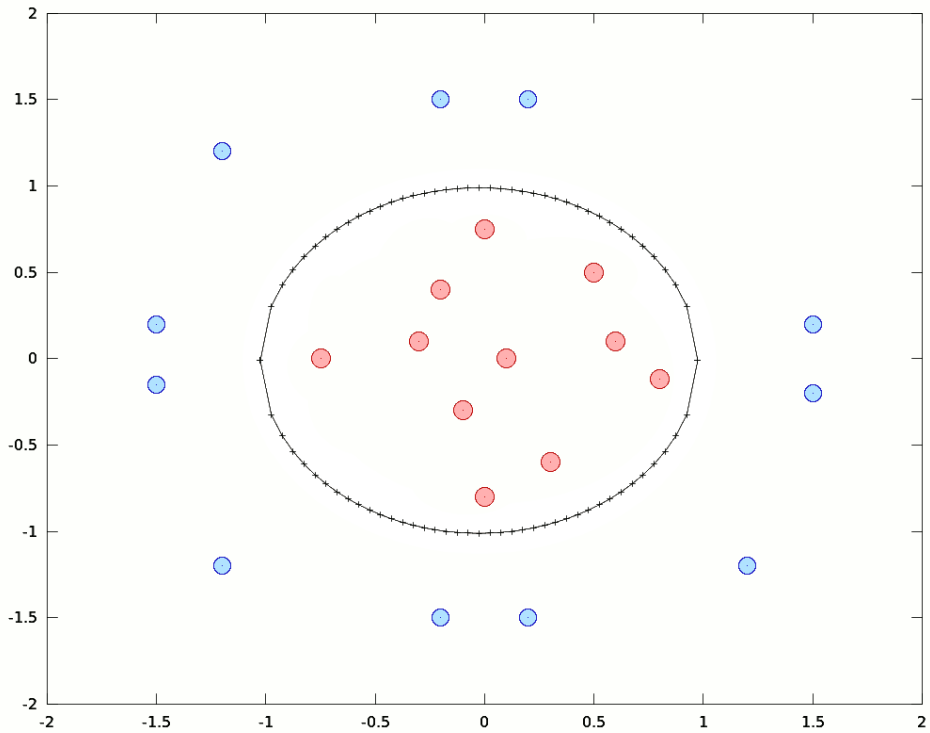
Now is is easier to understand the **confidence** about a prediction: an output around 0.5 means the sample is over the decision boundary, and not very separated from the other class

How to build a classifier: non-linear



How is is possible to separate such classes? By adding quadratic terms to the features Example: $h_\theta(\mathbf{x}) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$.

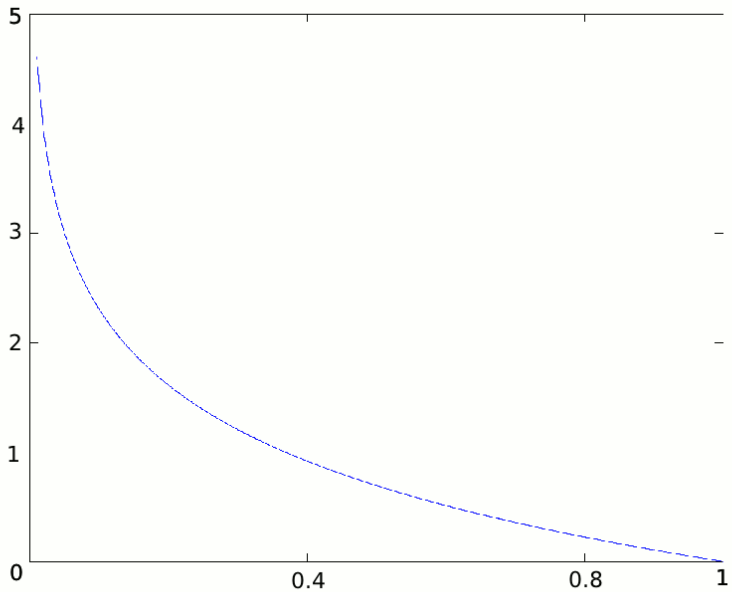
How to build a classifier: non-linear



By setting: $\theta = [-1, 0, 0, 1, 1]$, $\quad h_\theta(\mathbf{x}) = g(-1 + x_1^2 + x_2^2)$.

Logistic Regression Cost Function

$$Cost(h_\theta(\mathbf{x}, y) = \begin{cases} -\log(h_\theta(\mathbf{x})) & \text{if } y = 1 \\ -\log(1 - h_\theta(\mathbf{x})) & \text{if } y = 0 \end{cases}$$

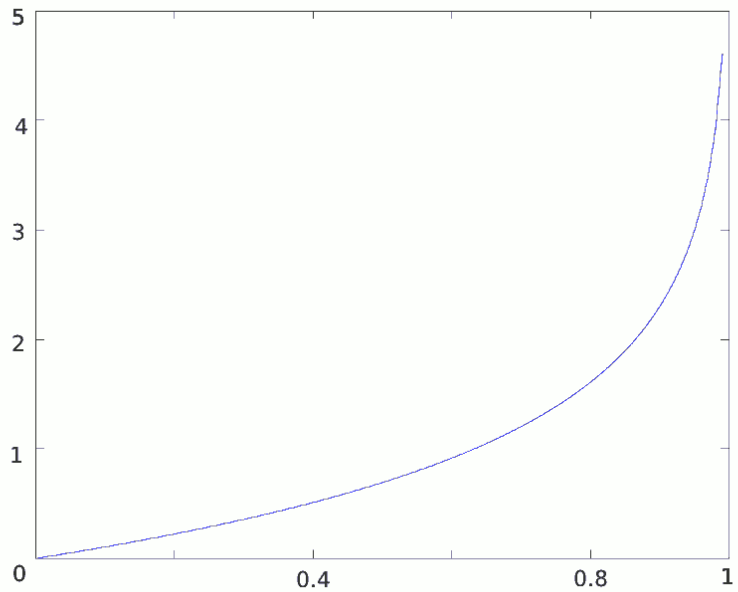


$Cost = 0$ if $y = 1$ and $h_\theta(\mathbf{x}) = 1$

Idea: penalize the learning algorithm in a wrong prediction.
Ex: $Cost = 2.3$ if $y = 1$ and $h_\theta(\mathbf{x}) = 0.1$

Logistic Regression Cost Function

$$Cost(h_\theta(\mathbf{x}, y) = \begin{cases} -\log(h_\theta(\mathbf{x})) & \text{if } y = 1 \\ -\log(1 - h_\theta(\mathbf{x})) & \text{if } y = 0 \end{cases}$$



$Cost = 0$ if $y = 0$ and $h_\theta(\mathbf{x}) = 0$

Ex: $Cost = 4.6$ if $y = 0$ and $h_\theta(\mathbf{x}) = 0.99$

Logistic Regression Cost Function

The overall cost function is:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} Cost(h_\theta(\mathbf{x_i}, y_i)$$

- The Cost Function can be rewritten in a more convenient way:

$$Cost(h_\theta(\mathbf{x}), y) = -y \log(h_\theta(\mathbf{x})) - (1 - y) \log(1 - h_\theta(\mathbf{x}))$$

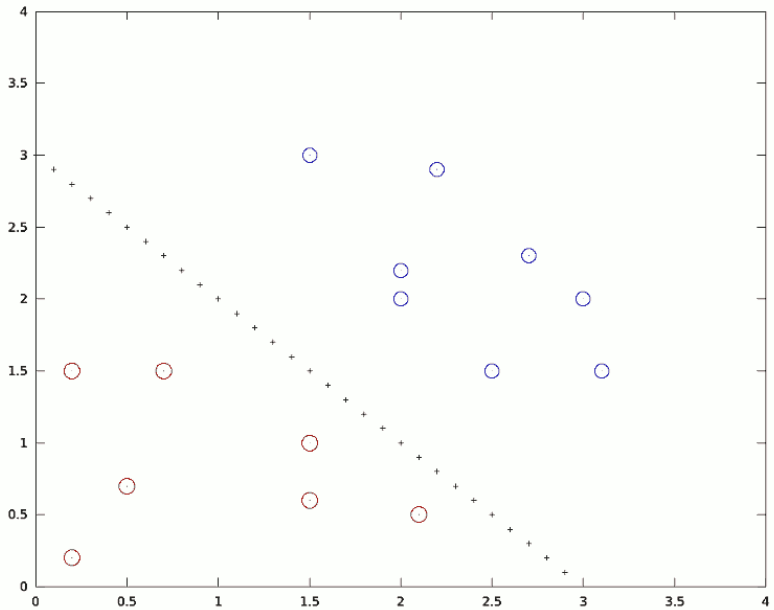
- The Overall Logistic Cost Function is:

$$J(\theta) = \frac{1}{N} \left[\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i)) \right]$$

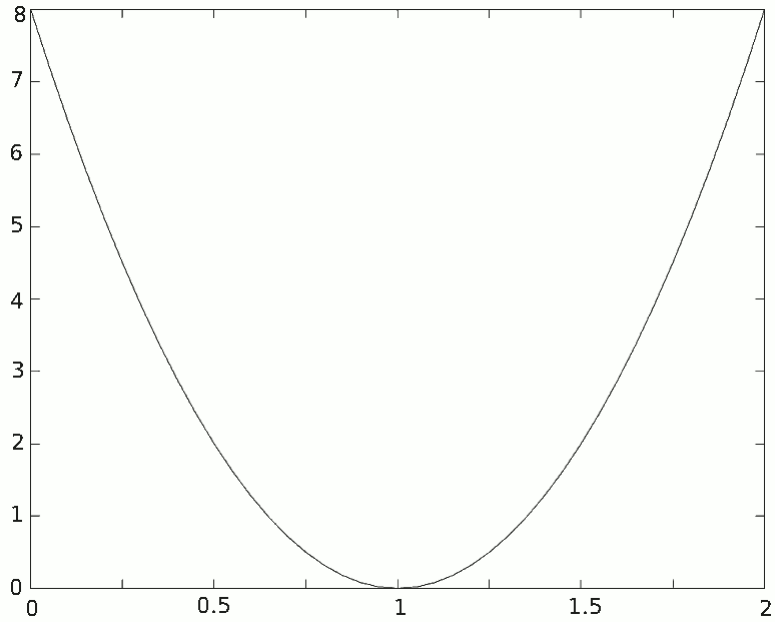
Logistic Regression Cost Function

Now the problem is to use $J(\theta)$ in order to minimize it in function of θ:

$$\min_{\theta} [J(\theta)]$$



$\theta = [-3, 1, 1]$
$h_\theta(\mathbf{x}) = -3 + x_1 + x_2$.



$J(\theta)$ function of the θ_1.

Logistic Regression Cost Function

$$\min_{\theta} [J(\theta)]$$

- There are many good optimization algorithms such as the Conjugate Gradient.
- A simpler algorithm (but not very fast) is the Gradient Descent, that looks for the descent direction of the derivative and updates the parameters in order to reduce the cost on each iteration:
  - start with some value for θ
  - changes the θ to reduce $J(\theta)$
  - hopefully end up at the global minimum
- The α parameter controls how "big" is each step of the descent. Too big can make the algorithm to diverge, too small can make the algorithm to be slow.

Logistic Regression Cost Function

- The algorithm repeats until convergence:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta), \ \forall j$$

- The algorithm perform simultaneous updates:

```
repeat until convergence (or a fixed number of iterations) {
  for each theta (j) {
      temp(j) = theta(j) - alpha (d / d theta(j)) J(theta)
  }
  for each theta (j) {
      theta(j) = temp(j)
  }
}
```

- For the logistic regression cost function the derivative was derived:

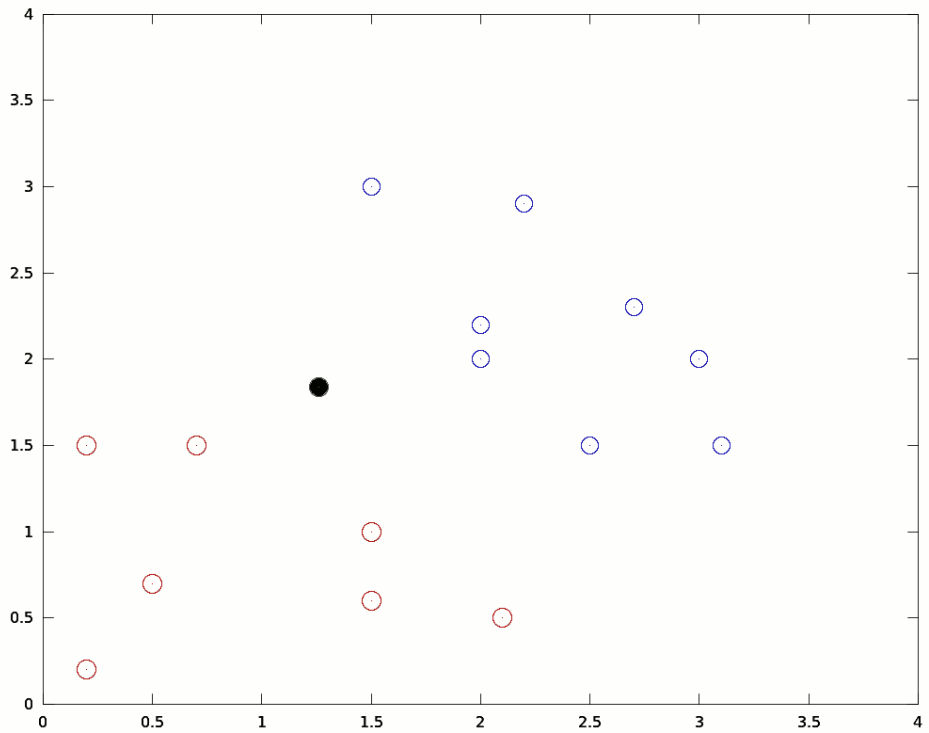
$$\theta_j = \theta_j - \alpha \sum_{i=1}^{N} (h_\theta(x_i) - y_i) x_{i,j}$$

Agenda

k-Nearest Neighbor Classifier

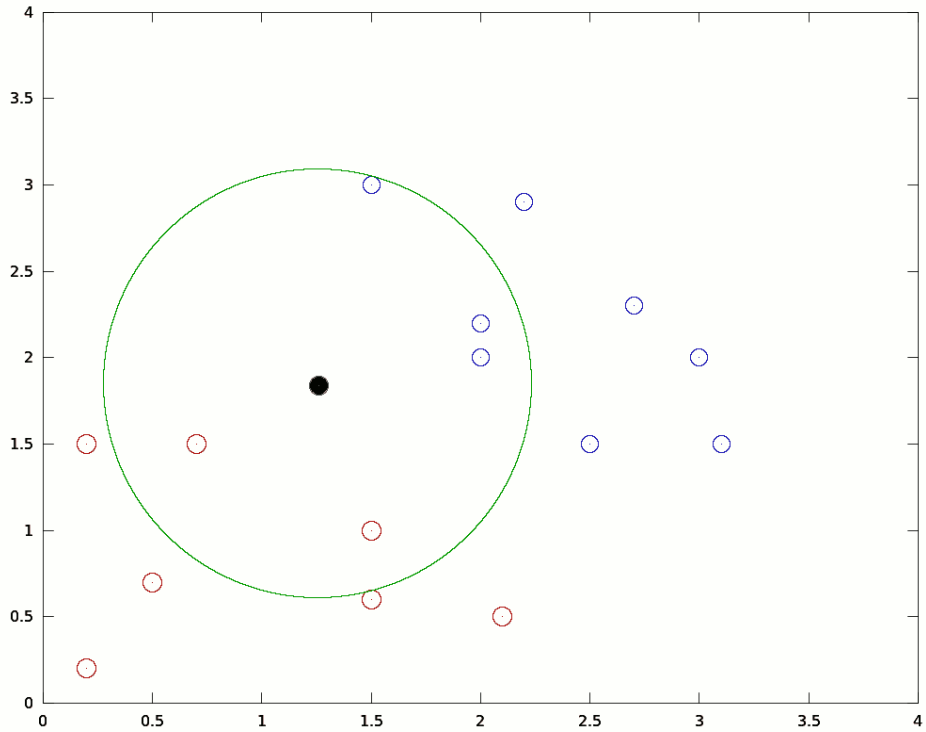
It is called a lazy learning algorithm, since it is just based on the instances.

- Very simple decision rule: a sample is assigned to the class that is more present in the k nearest neigbours considering the training set.



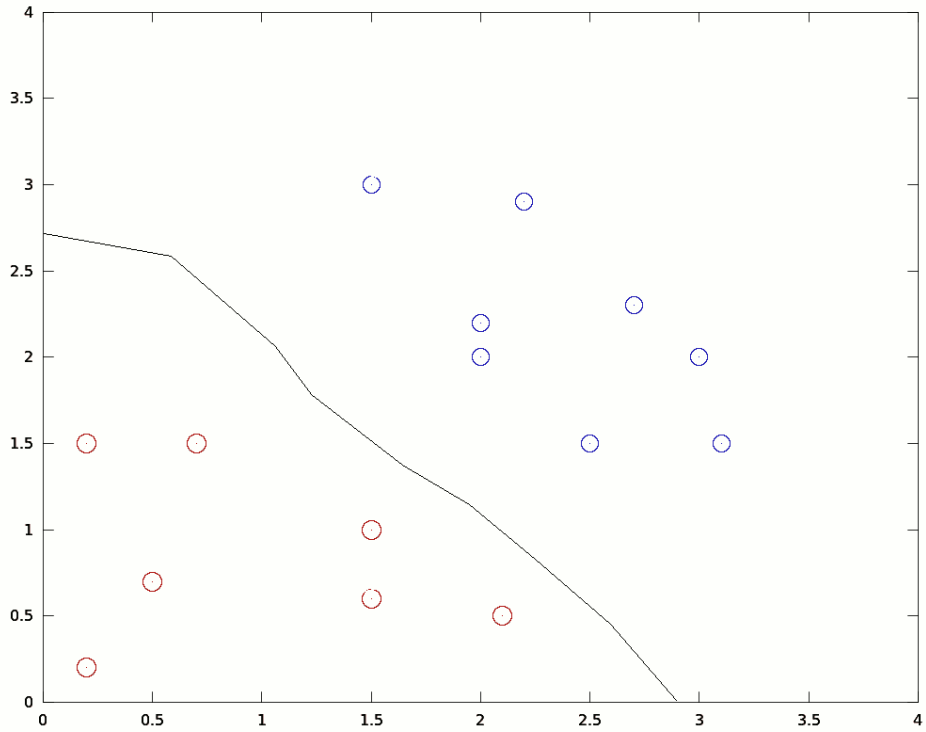
k-Nearest Neighbor Classifier

For $k > 1$, it is a classifier based on local densities. For instance, $k = 3$.



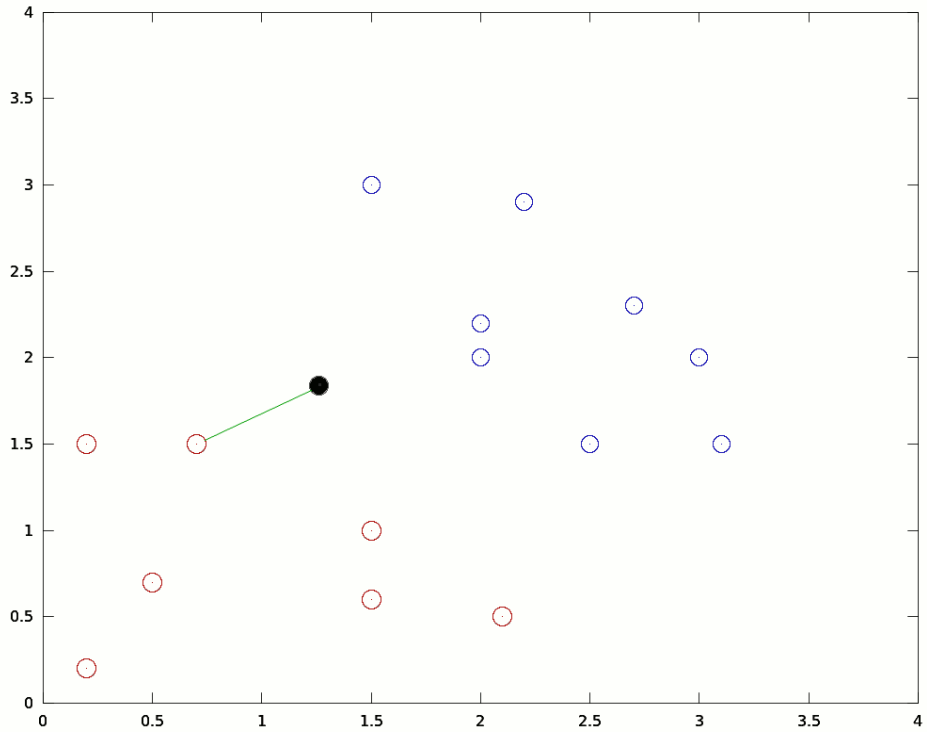
k-Nearest Neighbor Classifier

The decision boundary is non-linear and can be very complex in some cases.



k-Nearest Neighbor Classifier

For $k = 1$, it is a classifier based on distances.



k-Nearest Neighbor Classifier

The KNN classifier is simple but cannot handle complex spaces.

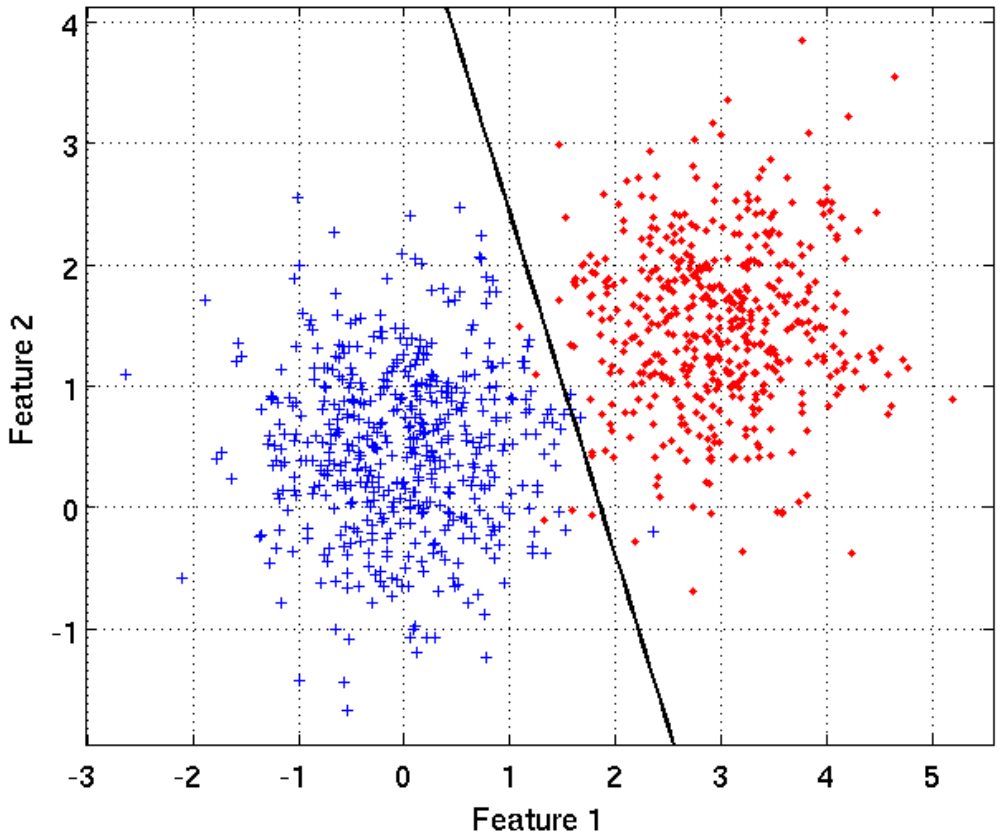
- A high value of k can overfit the classifier.
- Its behavior is sometimes difficult to predict.

Agenda

1. A classification task
2. Classification: basics
3. How to build a classifier
4. Logistic Regression Classifier
   - Cost Function
5. k-Nearest Neighbor Classifier
6. Multiple Classifier Systems
   - Creating ensembles of classifiers
   - Combining multiple classifiers
   - Diversity and Ensemble Performance

Classifier combination motivation

There is no single classifier that can be considered optimal for all problems. Depending on the problem, a simple linear classifier is sufficient:



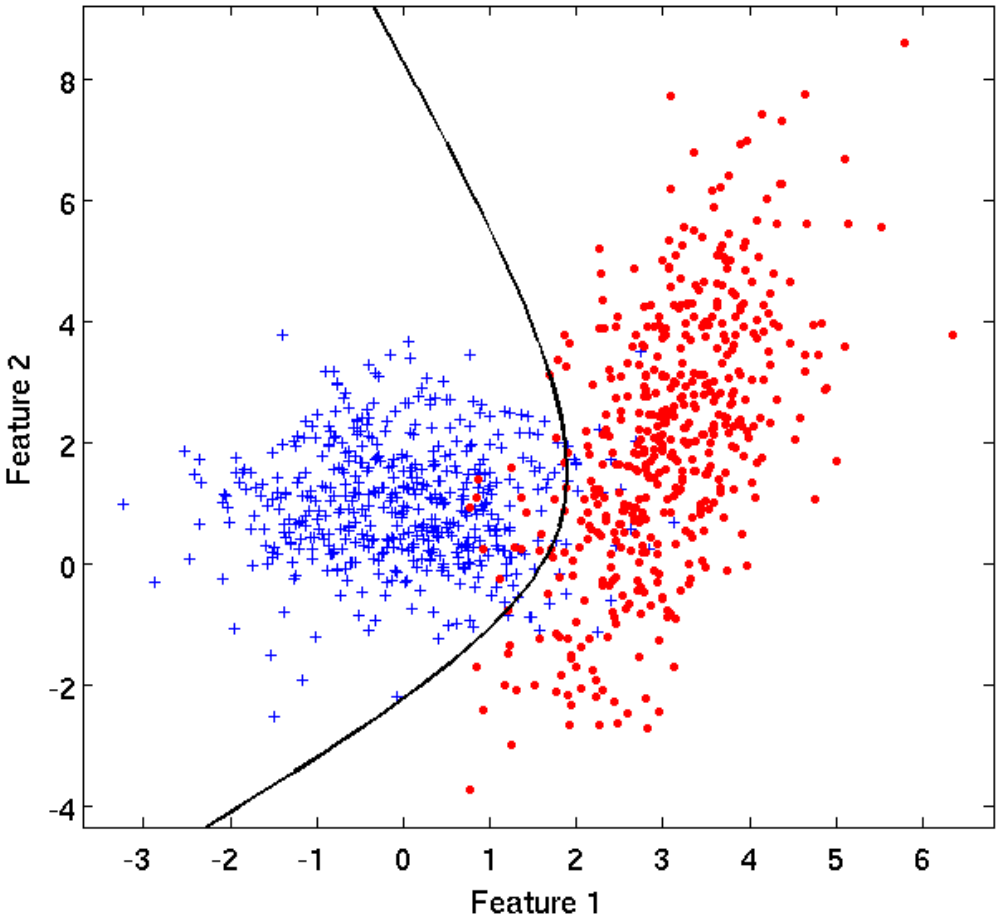
Two normally (Gaussian) distributed classes:

$$\mu_1 = [0, 0.5]\,, \mu_2 = [3, 1.5]$$

$$\Sigma_1 = \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The linear discriminant classifier (LDC) is optimal when all classes are Gaussians with equal covariance matrices.

Classifier combination motivation



Two normally (Gaussian) distributed classes:

$$\mu_1 = [0, 1], \mu_2 = [3, 2]$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}$$

The quadratic discriminant classifier (QDC) is optimal when all classes are Gaussians.

Classifier combination motivation

Finding a good classifier

- There is no clear guideline to choose a set of good learning methods
- It is rare when one has a complete knowledge about data distribution

Limited number of object to train the classifier

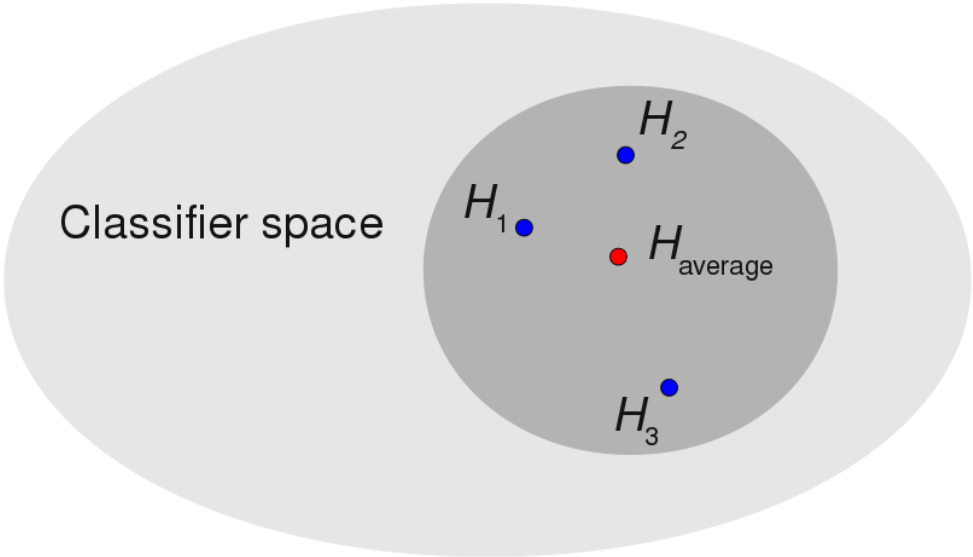
- Selecting the **best current classifier** can led to the choice of the **worst classifier for future data**.
- The test set provides just **apparent** errors $\hat{E}$ that differ from true errors E, in a generalization error: $\hat{E} = E \pm \Delta$

Classifier combination motivation

According to Dietterich (2000), there are three main motivations to combine classifiers:

(1) Statistical (or worst case) motivation

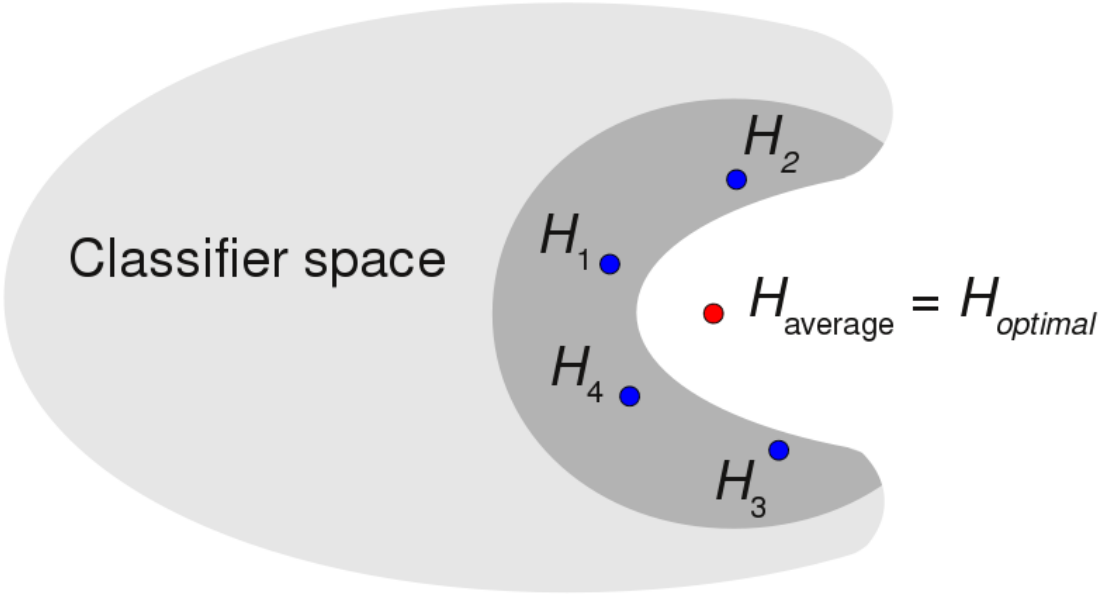
- **Avoid the worst classifier** by averaging several classifiers
- Confirmed theoretically by Fumera and Roli (2005)
- No guarantee that it will perform better than the best classifier



Classifier combination motivation

(2) Representational (or best case) motivation

- Under particular situations, it is possible to **improve the performance of the best individual classifier**
- It is true when the optimal classifier in a problem is outside the possible "classifier space"



Classifier combination motivation

(3) Computational motivation

- Some algorithms performs optimization to perform training and are subject to **local minima**
- Others, such as some neural-networks uses **random initialization**
- In both cases it is difficult to find a single best classifier, often choosen after hundreds of experiments
- Combination of such classifiers stabilize and can improve the best single classifier result (BREVE; PONTI-JR; MASCARENHAS, 2007)

Classifier combination motivation: the bias-variance decomposition

The **bias-variance decomposition** (GEMAN, 1992) is used in attempts to understand theoretically the ensemble methods.

Tumer and Ghosh (1996), based on manipulation used in bias-variance decomposition, built a framework and analyzed linear combination:

- the ensemble error will be equal to the average error of the individuals if the classifiers are correlated (with respect to their errors)
- the error will be smaller than the error of the individuals if the classifiers are statistically independent
- the combination of low biased classifiers with high variance can reduce the resulting variance.

Classifier combination motivation: natural multiple classifier applications

Other motivations are:

- applications that can **naturally use a set of independent sources** such as in sensor fusion, multimodal biometrics, and others,
- the difficulty on the design of a single pattern classifier by **tuning parameters**,
- set of classifiers are available and they **exhibit different competences in different feature subspaces**.

Agenda

Creating ensembles of classifiers

Methods to create classifiers that are good to combine:
diverse and **accurate**

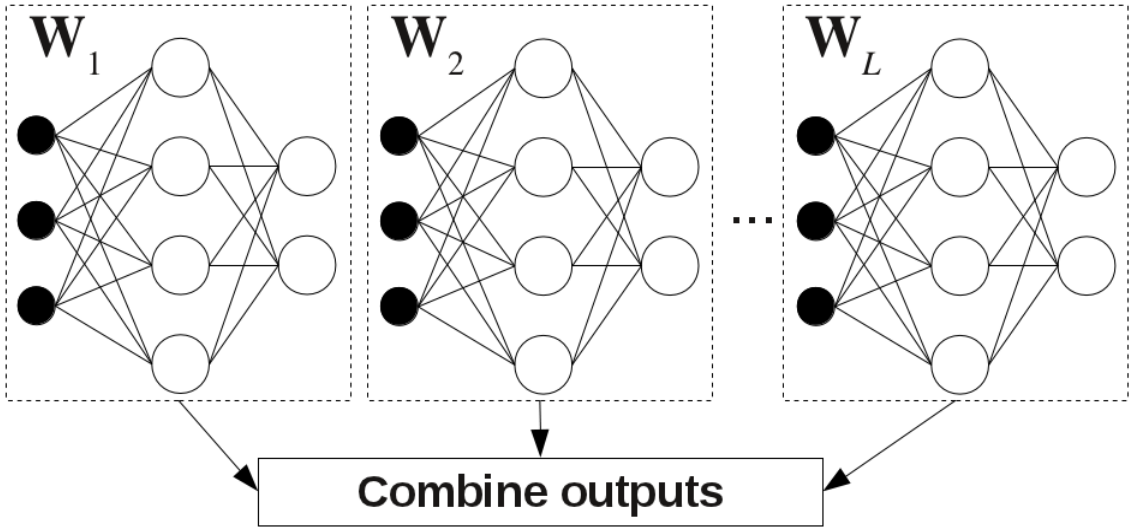
(1) Using the knowledge about the problem

- Build a tuned system regarding architecture and parameters to create diverse classifiers
- Examples: natural multiple classifier systems such as multiple sensors, when different representations of patterns are possible (statistical and structural).

Creating ensembles of classifiers — methods (2)

(2) Randomization

- Several classifiers trained using different random instances
- When use: classifiers that depends on some random choices
  - decision trees on the test of each internal node when random selection on n best tests are performed
  - neural-network with random weight initialization
- Example: Random Trees (BREIMAN,2001)



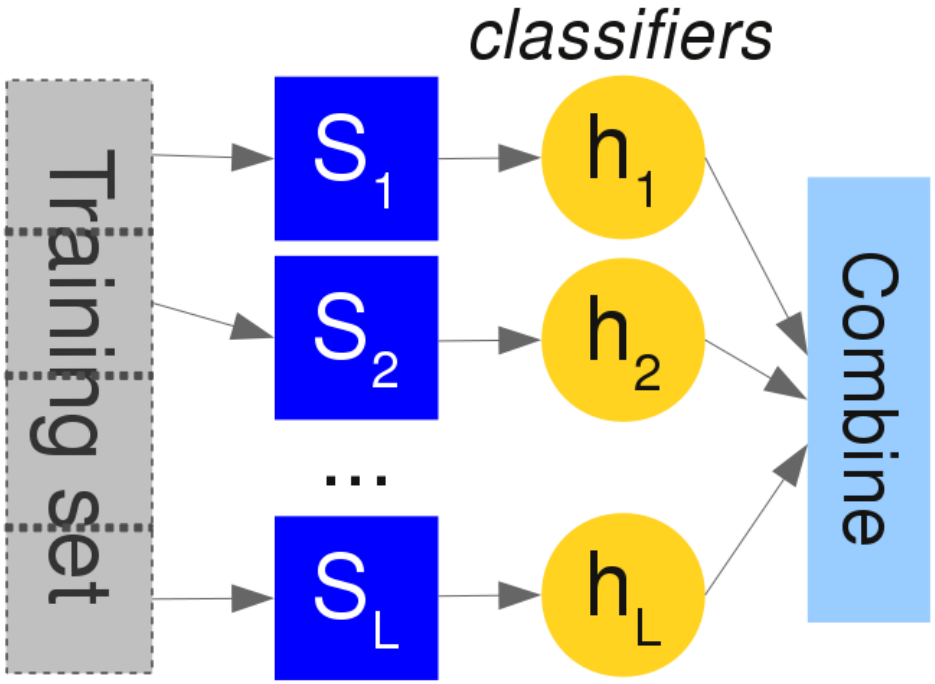
Creating ensembles of classifiers — methods (3)

(3) Training data manipulation

- Training an ensemble of classifiers using different training sets by:
  - splitting the set (PONTI-JR; PAPA, 2011)
  - using cross-validation ensembles — when few objects are available
  - bootstrap aggregation (BREIMAN, 1996)
  - boosting (FREUND; SCHAPIRE, 1996)

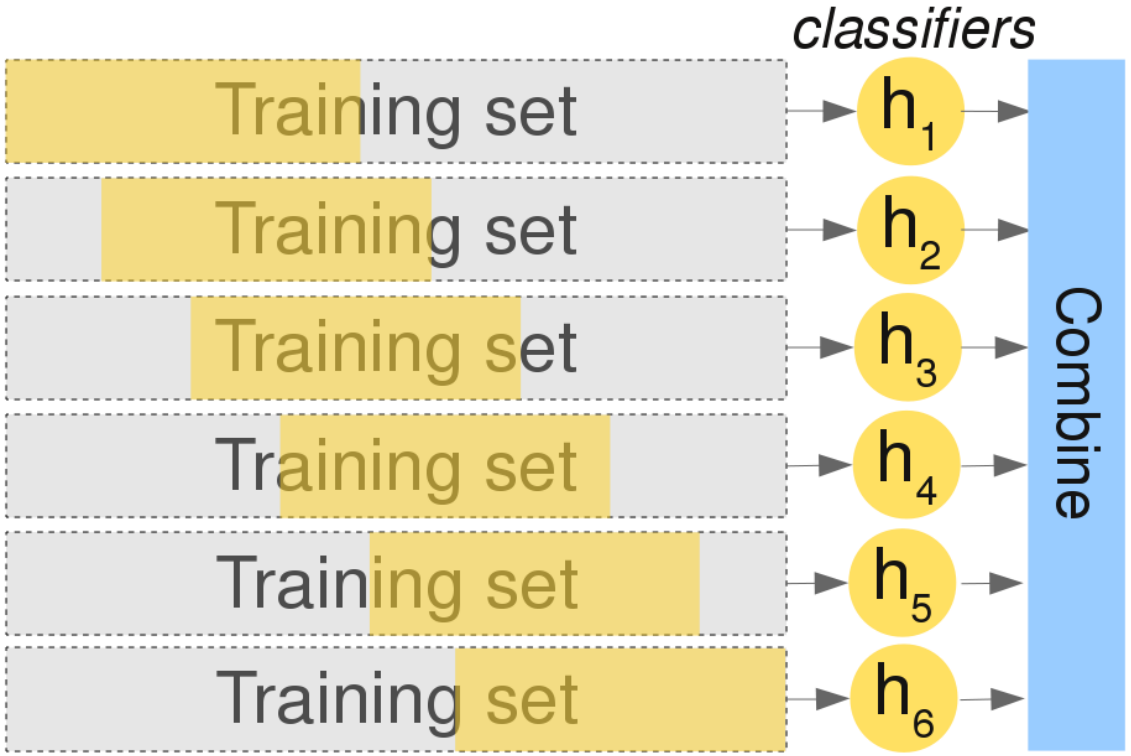
Creating ensembles of classifiers — methods (3)

Training set split — disjunct partitions



Creating ensembles of classifiers — methods (3)

Cross validation ensemble



Creating ensembles of classifiers — methods (4)

(4) Input features manipulation

- Training classifiers using different subspaces of the feature space: random subspace method (RSM) (HO, 1998)

(5) Output features manipulation

- Each component classifier solves a subset of the N classes problem.
- Example: Error-Correcting Output Coding (ECOC) (DIETTERICH; BAKIRI, 1995)

Bagging — Bootstrap aggregation

Idea

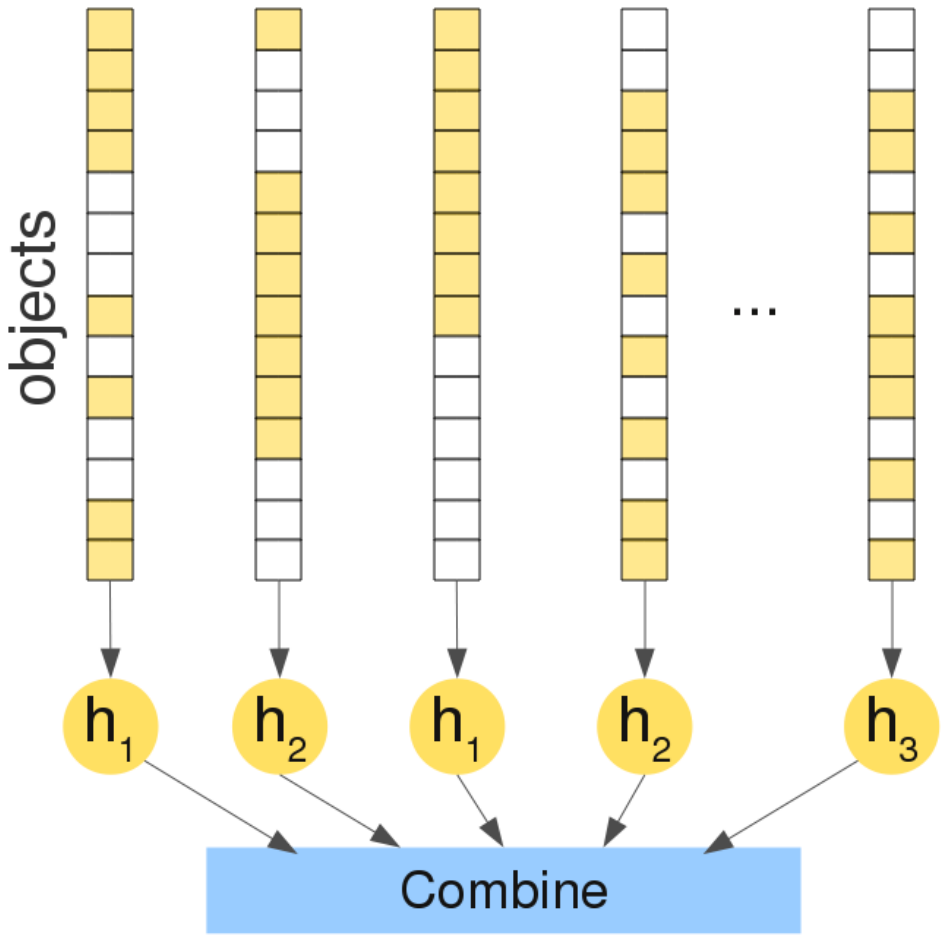
Bootstrap samples of the original training set will present:

- a small change with respect to the original training set, and
- sufficient difference to produce diverse classifiers

Method

- Each member of the ensemble is trained using a different training set:
  - sampling from the original set, choosing N items uniformly at random with replacement.
- The outputs are combined by averaging or majority voting.

Bagging — Bootstrap aggregation



Bagging — Bootstrap aggregation

Require: Ensemble size L, training set S of size N.

1: **for** $i = 1$ to L **do**
2: $S_i \leftarrow N$ sampled items from S, with replacement.
3: Train classifier h_i using S_i.
4: **end for**
5: **for** each new object **do**
6: **if** outputs are continuous **then**
7: Average the decisions of h_i, $i = 1, ..., L$.
8: **else if** outputs are are class labels **then**
9: Compute the majority voting of h_i, $i = 1, ..., L$.
10: **end if**
11: **end for**

Bagging — Bootstrap aggregation

Bootstraps

The probability for any object to not be selected is $p = (1 - 1/N)^N$.

- For large N, a bootstrap is expected to contain $\sim$63% of the original set, while 37% are not selected.

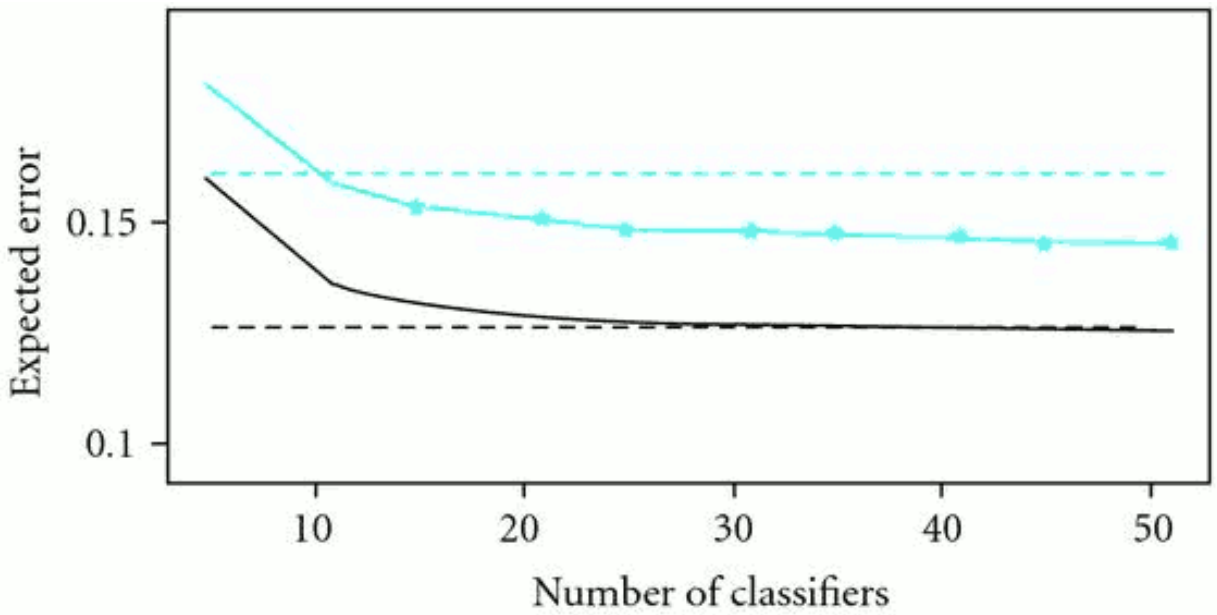
Pros

- Instances of an unstable classifier trained using different bootstraps can show significant differences,
- Variance reduction (proof by bias-variance decomposition),
- Works better with unstable models (decision trees, neural networks),
- Can fail to improve performance of lazy learners (k-NN)

Bagging — Bootstrap aggregation

Parameter

- Number of bagged classifiers
- In the literature it is common to see from 50 to 100 classifiers
- **However with 10 classifiers the error reduction is around** 90%



Adaboost — adaptive boosting

Idea

- Combine a **sequence** of a base classifier in order to produce an accurate "strong" classifier.
- Is a boosting method, based on the idea that: a **weak** model, performing only slightly better than random guessing, could be boosted into an arbitrarily accurate **strong** model.

Iterative Method

- The algorithm trains classifiers sequentially, a new model per iteration.
- After training, the misclassified patterns are **weighted** in order to be considered more important in the next round.
- By weighting, subsequent models compensate error made by earlier classifiers.

Adaboost — adaptive boosting

Require: Ensemble size L; training set S of size N, where $y_i \in \{+1, -1\}$;
initialize uniform distribution W_i over S.

for $i = 1$ to L **do**
  Train classifier h_i using distribution W_i.
  Compute $\varepsilon_i \leftarrow P_{W_i}(h_i(x) \neq y)$.
  if $\varepsilon_i \geq 0.5$ **then**
    break
  end if
  $\alpha_i \leftarrow \frac{1}{2} \ln \left(\frac{1-\varepsilon_i}{\varepsilon_i} \right)$
  $W_{i+1} \leftarrow \frac{W_{i+1} \exp(-\alpha_i y_i h_i(x_i))}{Z_i}$,
  where Z_i normalizes to ensure that W_{i+1} is a valid distribution
end for
for each new object x **do**
  $H(x) \leftarrow \text{sign} \left(\sum_{i=1}^{L} \alpha_i h_i(x) \right)$
end for

Adaboost — adaptive boosting

Distribution W_i

- After each iteration, the distribution is updated so that **half** the distribution mass are over the **samples misclassified** by h_i.
- Example: if the error is 0.15, the next classifier will put 50% of effort in order to classify correctly the 15% wrongly classified objects, while the others are less emphasized:

$$\sum_{h_i(x_n) \neq y_n} W_{i+1}(n) = 0.5.$$

Random Subspace Method (RSM)

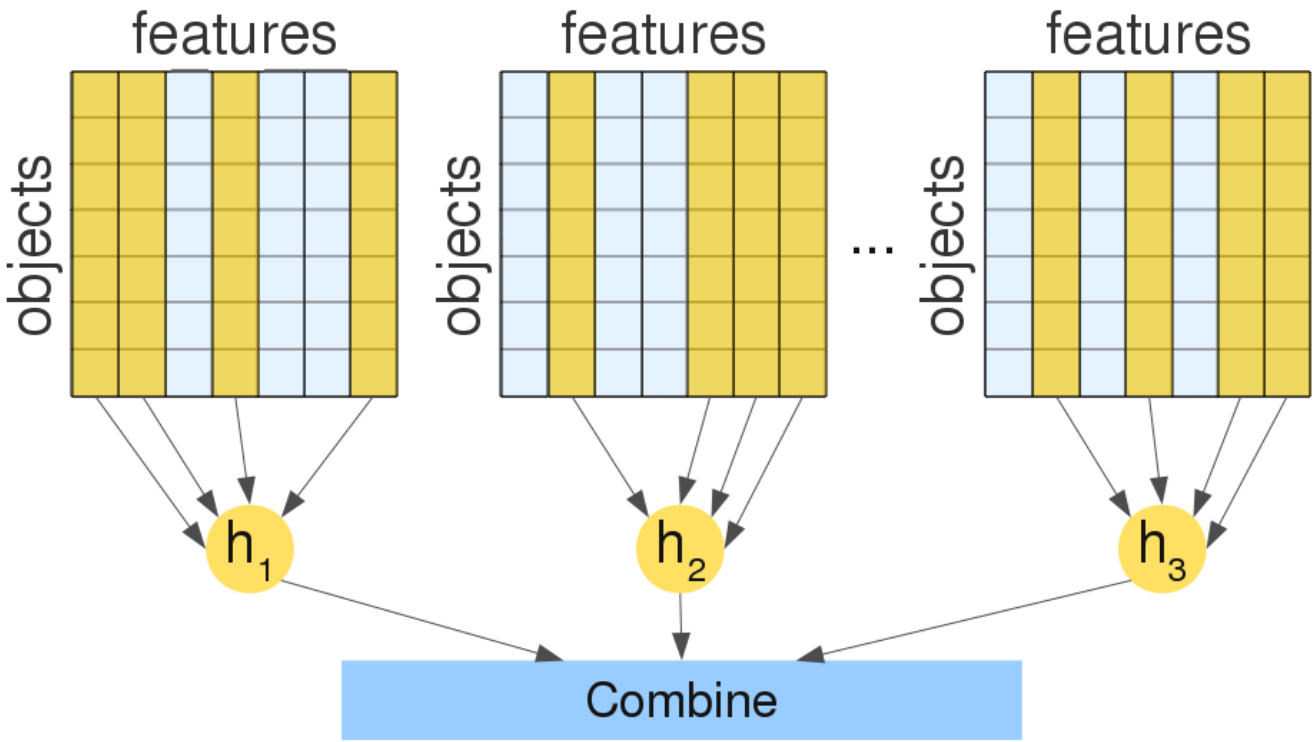
Idea

- Randomly selects an **arbitrary number of features** from the original feature space, and build a classifier on each subspace.
- This randomization should create classifiers that are complementary
- Works well with large feature sets and redundant features, avoiding the curse of dimensionality

Iterative Method

- The algorithm trains classifiers sequentially, a new model per iteration.
- After training, the misclassified patterns are **weighted** in order to be considered more important in the next round.
- By weighting, subsequent models compensate error made by earlier classifiers.

Random Subspace Method (RSM)



Random Subspace Method (RSM)

Require: Ensemble size L; training set S of size N, where the number of features is D; choose d_i to be the number of features to train each individual classifier, where $d_i < D$, for $i = 1, ..., L$.

1: **for** $i = 1$ to L **do**
2: $S_i \leftarrow d$ randomly choosen features out of D, without replacement.
3: Train classifier h_i using S_i.
4: **end for**
5: **for** each new pattern **do**
6: **if** outputs are continuous **then**
7: Average the decisions of h_i, $i = 1, ..., L$.
8: **else if** outputs are are class labels **then**
9: Compute the majority voting of h_i, $i = 1, ..., L$.
10: **end if**
11: **end for**

Random Subspace Method (RSM)

Parameter

- Define the dimensionality of the subspaces (variable d).
- There is no clear guideline, for each application experiments have to be carried out to understand the effect of d on the performance.

Agenda

Combining multiple classifiers

Levels of classifier outputs

The type of information produced by each single classifier can be:

- **Abstract**: outputs the class label for each input pattern.
- **Rank**: outputs a ranking list of possible classes for each input pattern.
- **Measurement**: outputs a score, probability or confidence level for each input pattern.

Combining multiple classifiers

Approaches to combine the decisions

1. **Integration (or fusion)**: all classifiers contribute to the final decision, assuming competitive classifiers.
2. **Selection**: one classifier is used to give the final decision to each pattern. It assumes that classifiers are complementary.

Elementary combiners — fixed rules

The matrix where each cell represents the output measure of a classifier (row) about a class (column), is called *decision profile* (DP).

	ω_1	ω_2	ω_3	ω_4
h_1	0.1	0.5	0.2	0.1
h_2	0.3	0.3	0.3	0.1
h_3	0.2	0.0	0.8	0.0

Rule					*Result*
Minimum	0.1	0.0	0.2	0.0	ω_3
Maximum	0.3	0.5	0.8	0.1	ω_3
Product	0.01	0.00	0.05	0.00	ω_3
Average	0.2	0.3	0.4	0.1	ω_3
Median	0.2	0.3	0.3	0.1	tie
Majority Vote	1	2	2	0	tie

Elementary combiners — fixed rules

A theorectical framework for fixed rules was proposed Kittler et. al (1998).

Combine on measurement level

- **minimum**: finds the *mininum* score of each class between the classifiers and selects the class with the maximum score.
- **maximum**: finds the *maximum* score of each class between the classifiers and selectsthe class with the maximum score.
- **product**: *multiplies* the score provided by each classifier and selects the class with the maximum product.
- **sum**: *adds* the score provided by each classifier and selects the class with the maximum sum.
- **average**: finds the *mean* of the scores of each class and selects the class with the maximum mean. It is equivalent to the sum rule.
- **median**: finds the *median* of the scores of each class and selects the class with the maximum median.

Elementary combiners — majority voting

The **majority vote** works in the abstract level. For several methods to create ensembles, it is the optimal combiner.

Method

1. The class output of each classifier is counted as vote for a class
2. The input pattern is assigned to the class with the majority vote.
3. In order to avoid ties, the number of classifiers used for voting is usually not multiple of the number of classes.

Elementary combiners — majority voting

Weighted majority vote

1. A "trainable" variant.
2. The votes are multiplied by a weight that is often obtained by estimating the classifiers' accuracies on a validation set.
3. One possible selection is:

$$w_i = \log\left(\frac{p_i}{1-p_i}\right),$$

   where p_i is the accuracy of the i^{th} classifier.
4. This weight selection guarantees mininum error for the weighted majority vote when the outputs are independent.
5. A similar approach can be used to produce a trainable weighted average.

Agenda

Diversity and Ensemble Performance

By intuition the classifiers of an ensemble should be at the same time as accurate as possible and as diverse as possible.

However...

- While it is known that the classifiers must be accurate, i.e, produce an error rate that is better than random guessing (HANSEN, 1990),
- "Diversity" is an elusive concept (BROWN, 2005), not trivial to define and use in practice.

Point of consensus

When the classifiers make statistically independent errors, the combination has the potential to increase the performance of the system.

Diversity and Ensemble Performance

Knowledge of diversity can help

Simple fusers can be used for classifiers with a simple complementary pattern, but more complex ones are necessary for classifiers with a complex dependency model.

In order to understand better this idea, we can classify diversity in levels:

1. no more than one classifier is wrong for each new object
2. the majority is always correct
3. at least one classifier is correct for each new object
4. all classifiers are wrong for some patterns

Diversity and Ensemble Performance

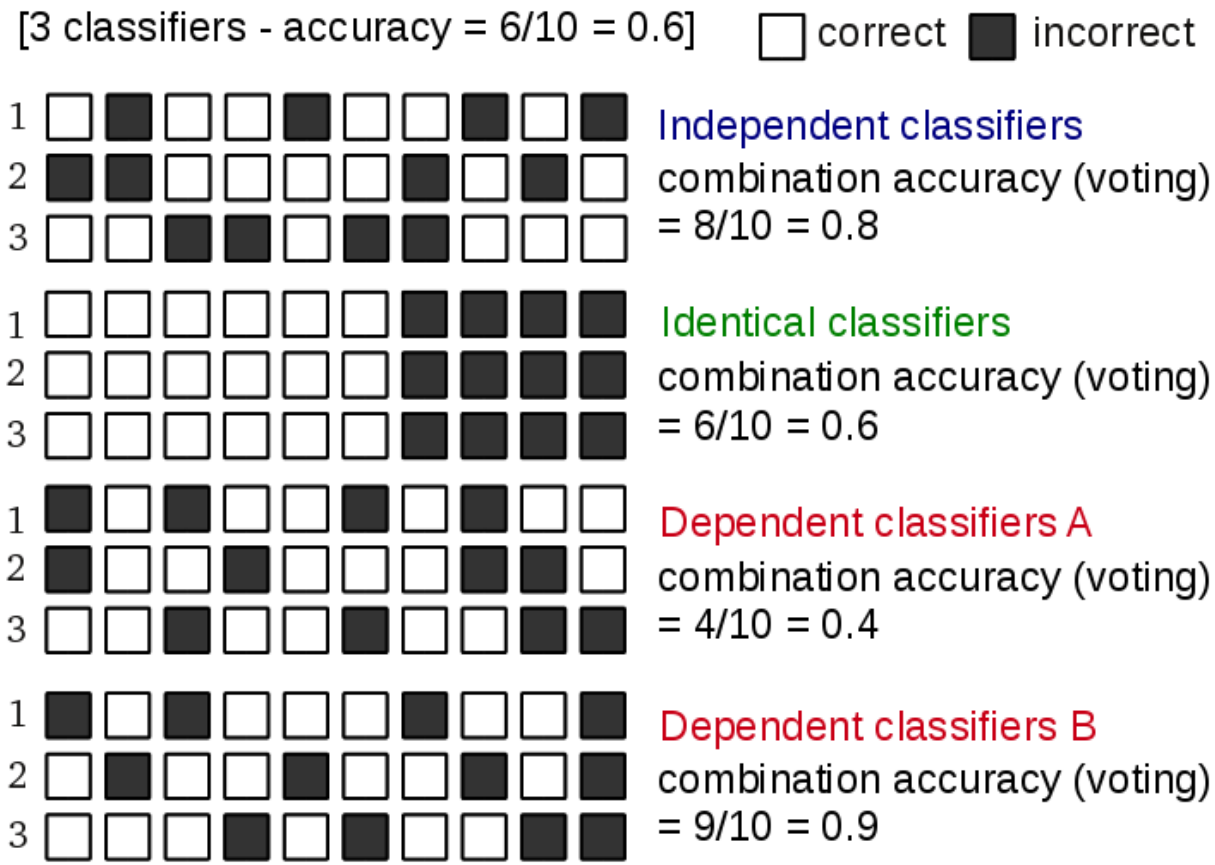
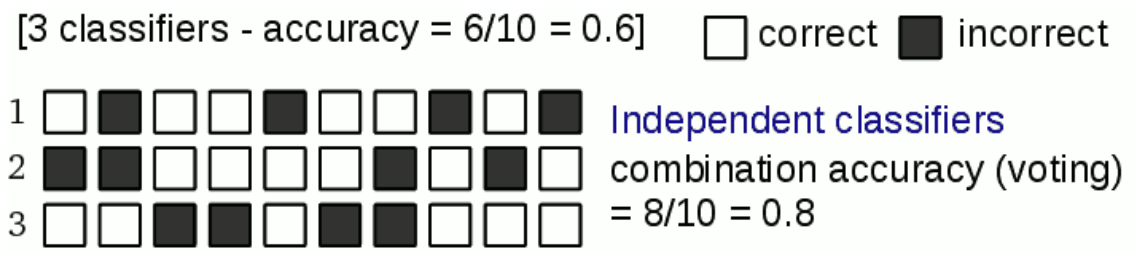


Figure: Four ensemble diversity settings: independent classifiers potentially increase the individual performance, and two cases of dependent classifiers with different results.

Diversity and Ensemble Performance: pairwise measures



Consider the first two classifiers of the independent case:

- the number of samples both classified correctly was $a = 3/10 = 0.3$,
- the number of samples misclassified by the first one is $b = 4/10 = 0.4$, by the second $c = 4/10 = 0.4$,
- the ones both misclassified was $d = 1/10 = 0.1$.

Note that $a + b + c + d = 1$.
Pairwise measures can be computed using a, b, c, d.

Diversity and Ensemble Performance: pairwise measures

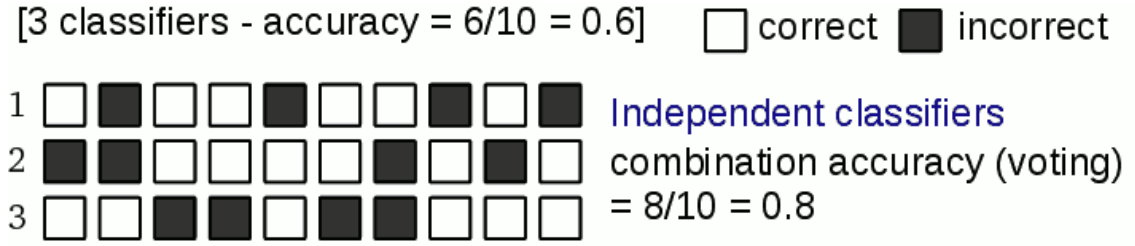
1. **Double fault measure** uses just d
2. Q **statistic** outputs a number in the $[1, -1]$ interval, where lower numbers means higher diversity:

$$Q = \frac{ad - bc}{ad + bc},$$

3. **inter-rated agreement**:

$$k = \frac{2(ad - bc)}{(a + c)(c + d) + (a + b)(b + d)}.$$

Diversity and Ensemble Performance: pairwise measures



Considering the first two classifiers:

1. **Double fault measure**: $d = 0.1$.
2. Q **statistic**

$$Q = \frac{ad - bc}{ad + bc} = -0.68.$$

3. **inter-rated agreement**:

$$k = \frac{2(ad - bc)}{(a + c)(c + d) + (a + b)(b + d)} = -0.37.$$

Diversity and Ensemble Performance

Once the diversity is estimated, it is possible to design better the system (FUMERA; ROLI, 2005)

In general

- simple average or majority voting are optimal for classifiers for approximate accuracies and same pairwise correlation
- weighted average is preferred for ensemble of classifiers with different accuracy or different pair-wise correlations

References

- Duda, R.; Hart, P.; Stork, D. Pattern Classification, 2.ed, 2000.
- Bishop, C. Pattern Recognition and Machine Learning, 2.ed, 2006.
- Ponti-Jr., M. Combining Classifiers: from the creation of ensembles to the decision fusion (a survey). 24th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2011.
- Kuncheva, L. Combining Pattern Classifiers, 2004.