Apprentissage supervisé

Apprendre aux ordinateurs à apprendre

Objectif : appliquer la démarche de l'apprentissage par l'exemple à l'ordinateur.

- **Montrer** des exemples à l'ordinateur en lui disant de quoi il s'agit
- Lui faire **apprendre** une règle
- Le laisser **appliquer** la règle à de nouveaux exemples
- **Evaluer** s'il a bien appris en comparant ses prédictions à la réalité.

Exemple 1 : classification binaire

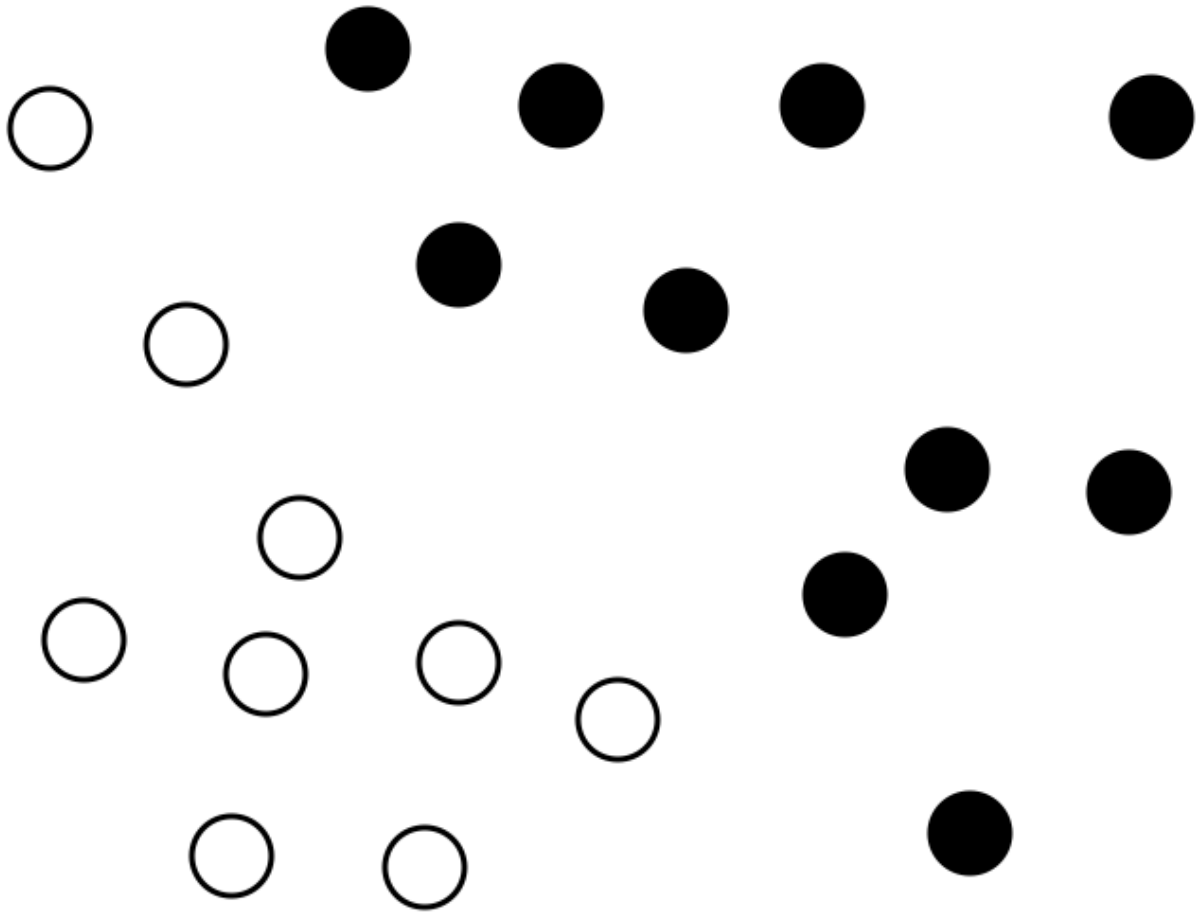


Image empruntée à Jean-Philippe Vert

Exemple 1 : classification binaire

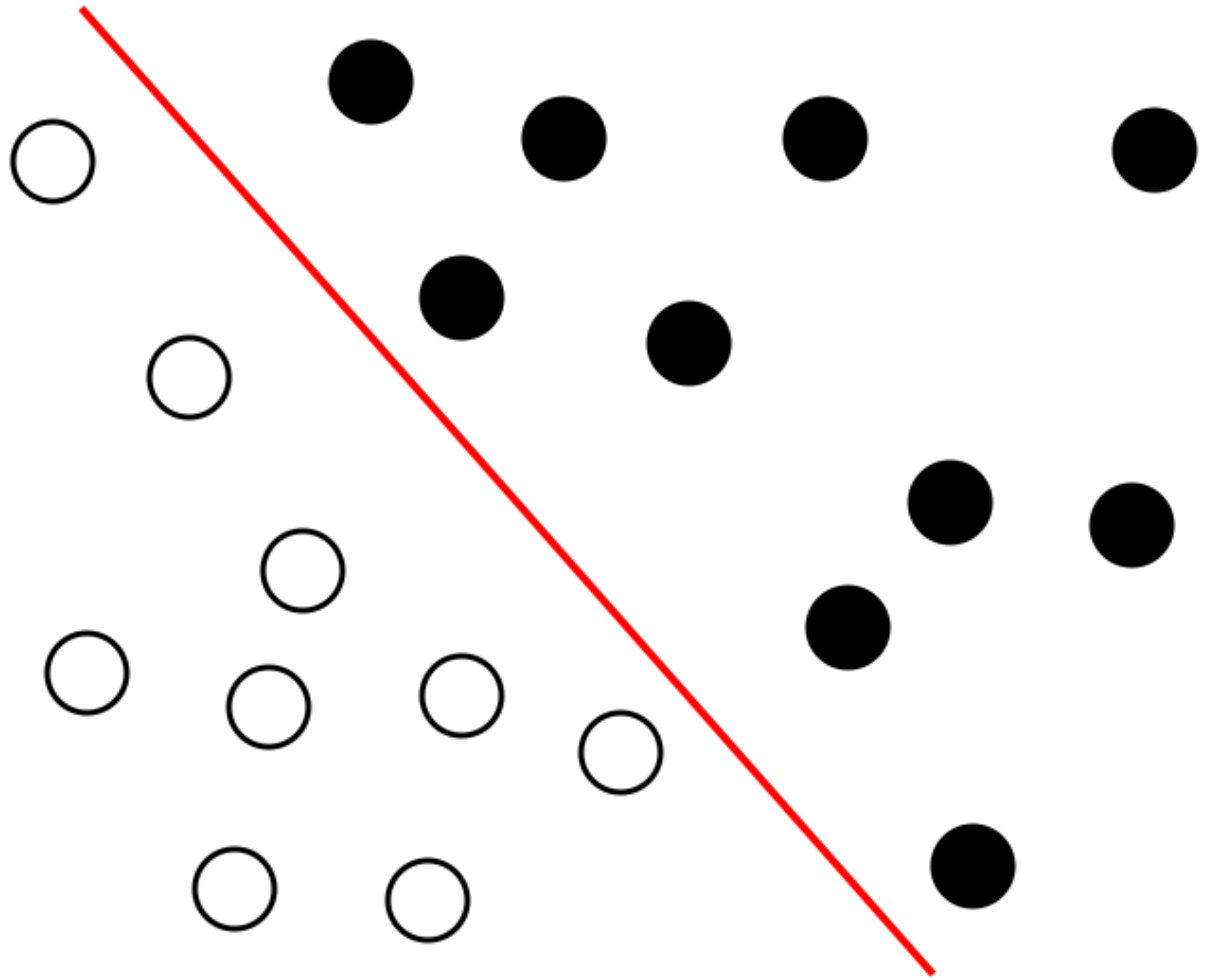


Image empruntée à Jean-Philippe Vert

Exemple 1 : classification binaire

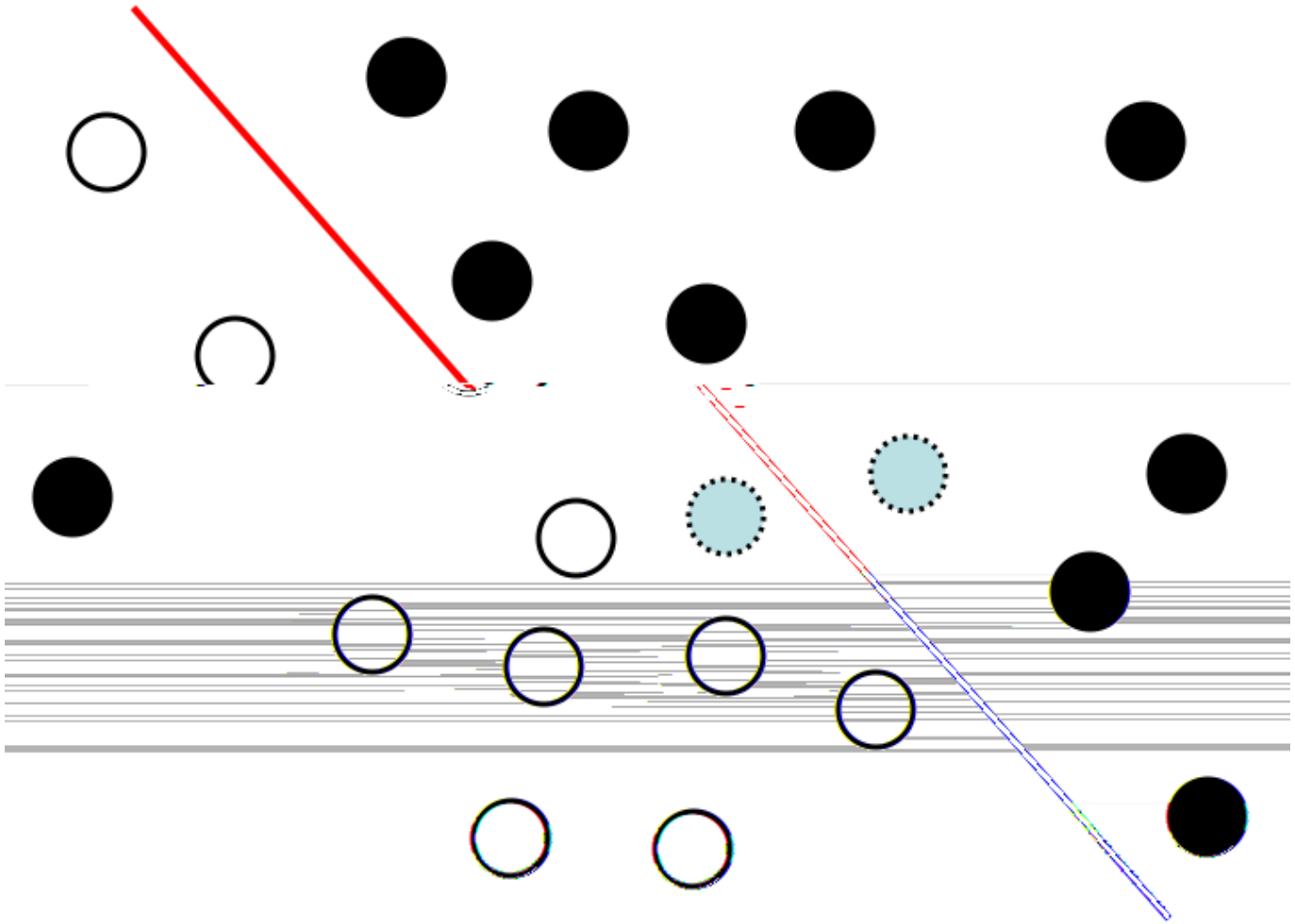


Image empruntée à Jean-Philippe Vert

Exemple 1 : classification binaire

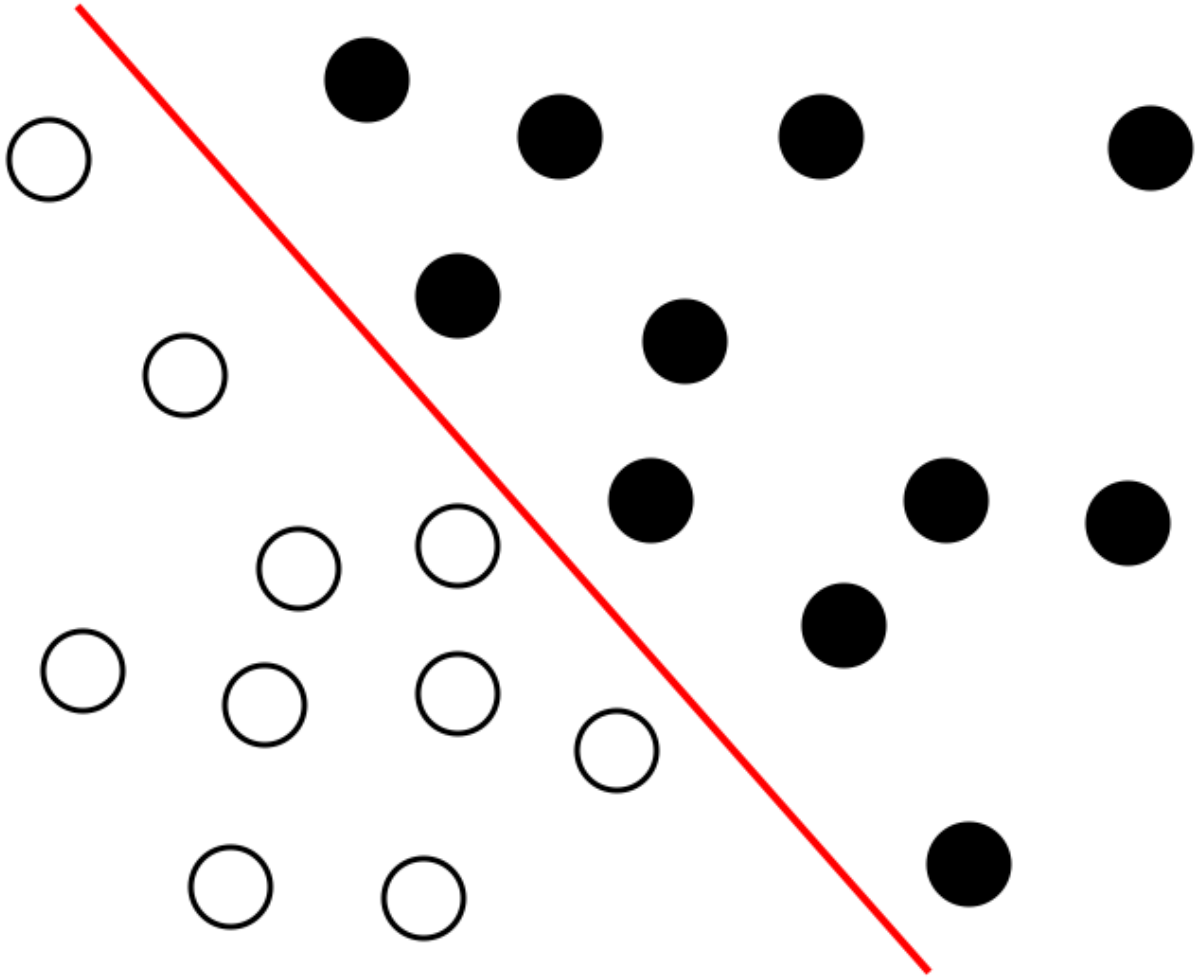
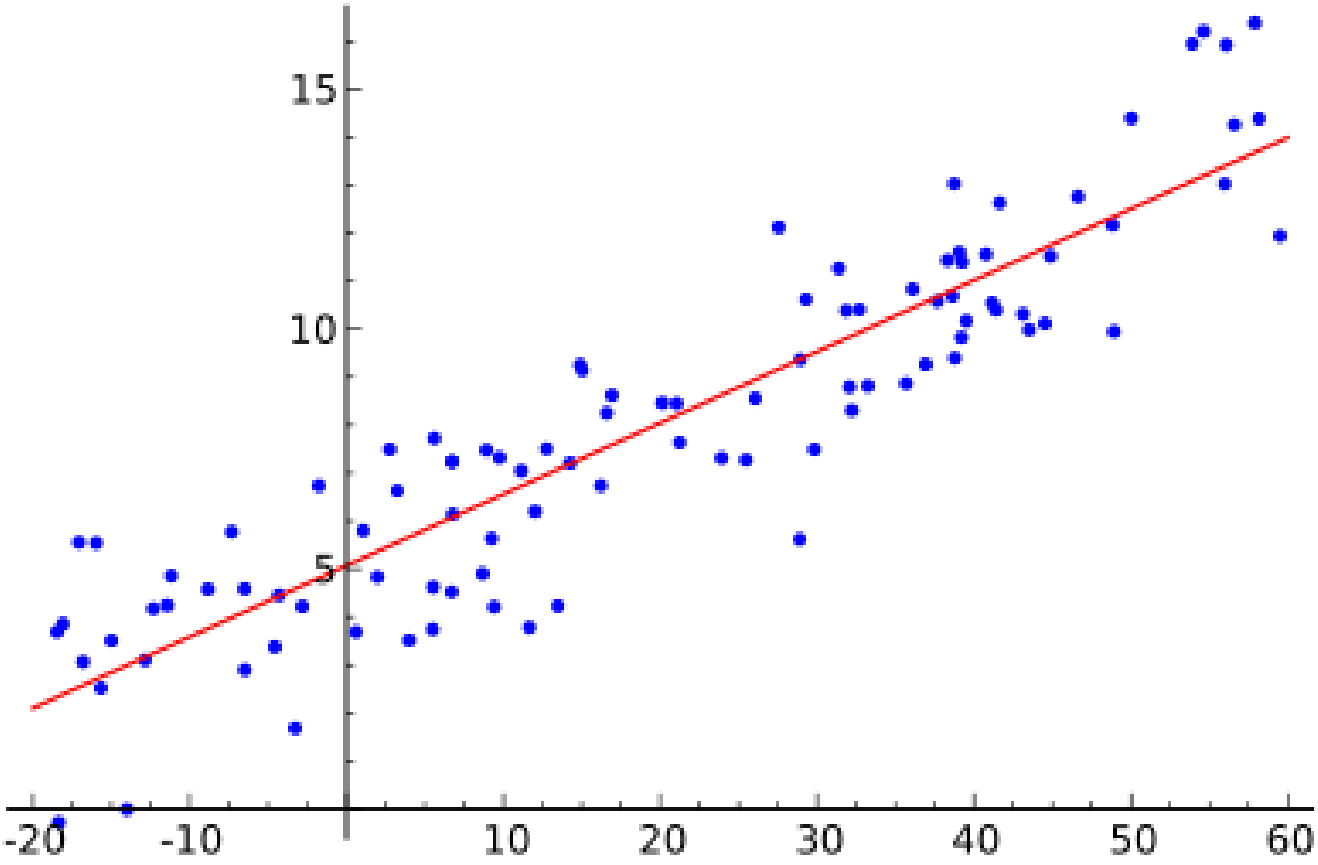


Image empruntée à Jean-Philippe Vert

Exemple 2 : régression linéaire

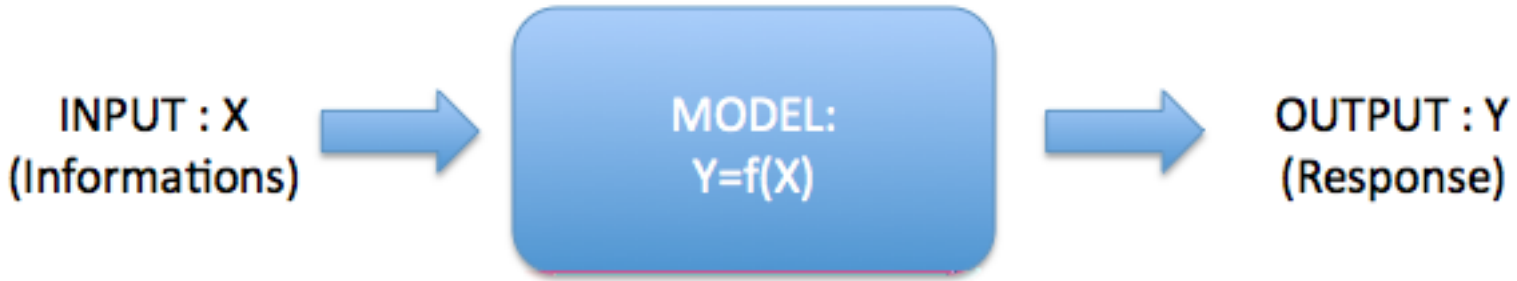


Source : wikipedia.com

Outline

Entrées et sorties

Le principe est toujours le même : X en entrée, Y en sortie. On cherche f tq $Y = f(X)$.



Exemples :

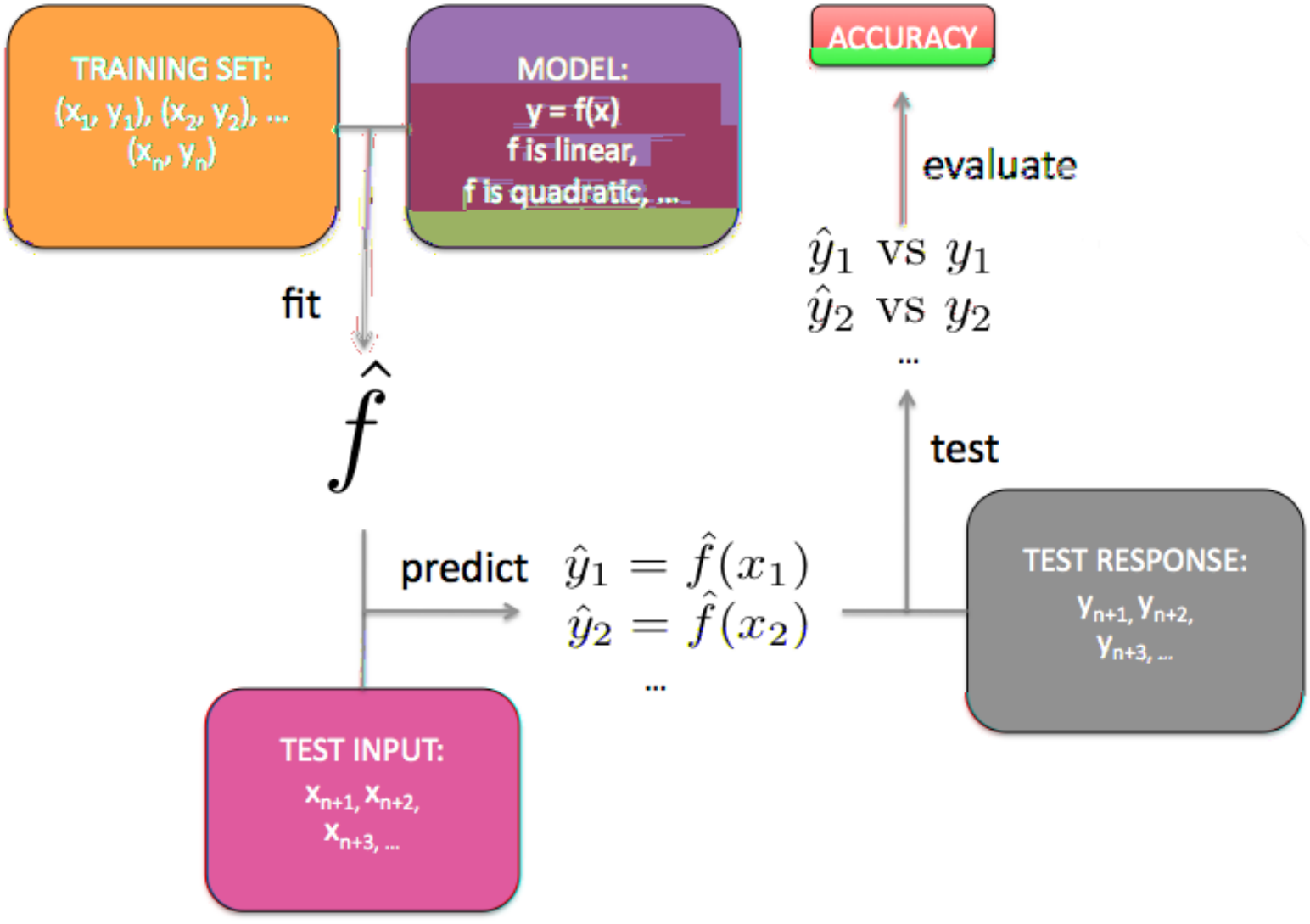
X	Y
emails	spam (oui on non)
profil client	nombre de clics
expression génique	état du patient
profil électeur	vote
âge, expérience	salaire

Types de problèmes

Il existe deux types de problèmes à résoudre:

- **Régression** : la réponse est un nombre réel. Exemples : prédiction du salaire, nombre de clics, prix d'un appartement.
- **Classification** : la réponse est une classe. Exemples : catégorie d'un article (classification multiple), spam (classification binaire).

Pipeline


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

Ensemble d'apprentissage

On entraîne le modèle sur **l'ensemble d'apprentissage** (training set).

Il est composé d'exemples de la forme (x_i, y_i) : pour chaque exemple i, on connaît donc la valeur d'entrée x_i et la réponse y_i.

Dans les cas que nous rencontrerons, x_i est souvent un **vecteur** et y_i est un **scalaire**.

On a n exemples: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$.

	sexe	âge	diplôme
x_1	0	30	5
x_2	1	25	2
x_3	1	53	3
...			
x_n	0	20	0

	salaire
y_1	3000
y_2	1800
y_3	2900
...	
y_n	1200

Exemple du texte

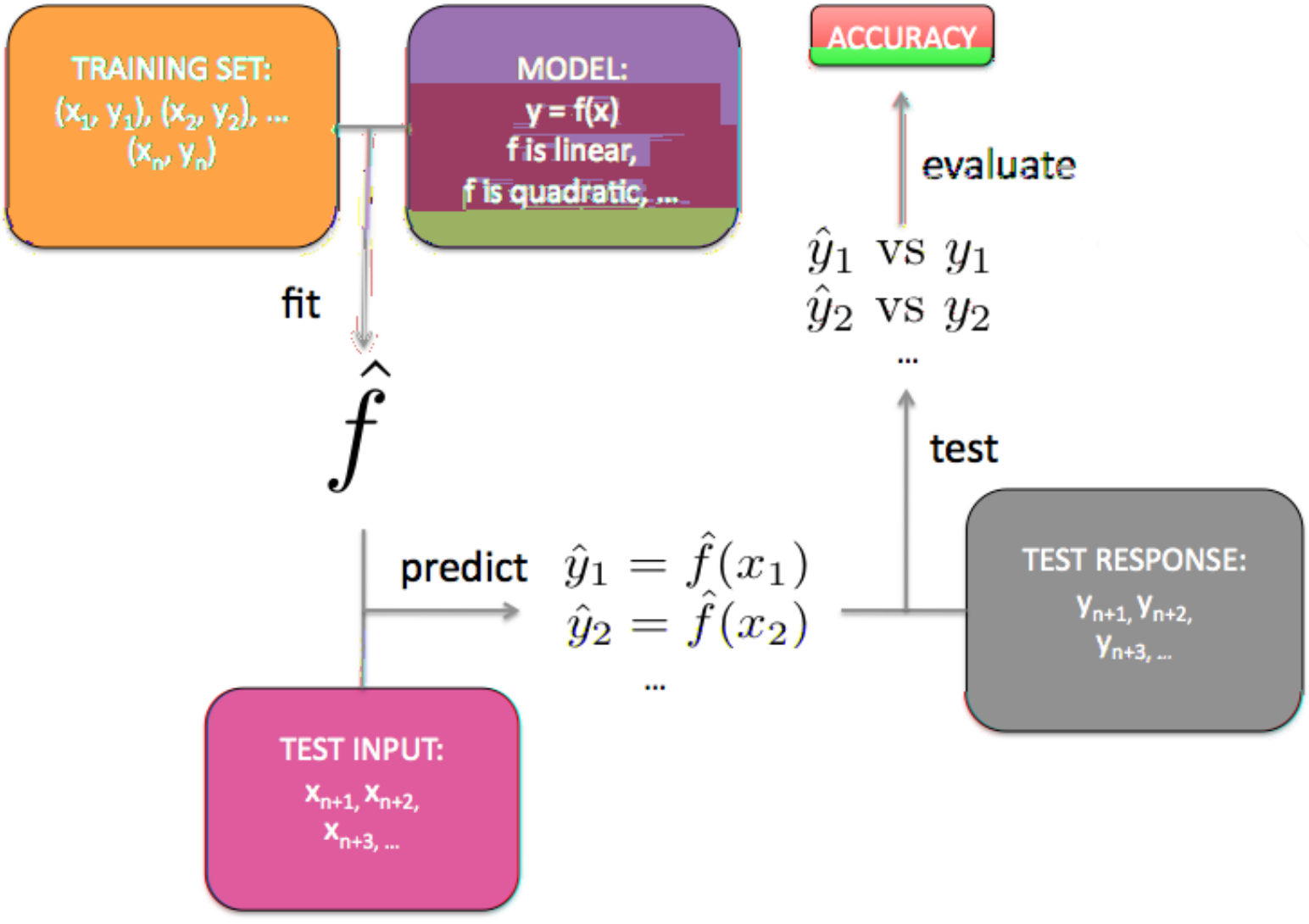
	voici	un	premier	texte	second	ce	document	contient
x_1	0,1	0,1	0,275	0	0	0	0	0
x_2	0,1	0,1	0	0	0,275	0	0	0
x_3	0	0	0	0	0	0,44	0,22	0,22
...								

	spam
y_1	0
y_2	0
y_3	1
...	

On veut apprendre à l'algorithme **ce qui fait que** les deux premiers messages ne sont pas des spams, le troisième oui, etc.

A la différence des premiers cours, on a ajouté l'information y.

Pipeline


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

Le modèle

C'est ici que l'on fait des **hypothèses** sur la forme de f.
Par exemple:

- f est **linéaire**:

$$y_i = \omega_1 x_{i,1} + \omega_2 x_{i,2} + \omega_3 x_{i,3} + ...$$

Il s'agit de trouver les valeurs de $\omega_1, \omega_2, \omega_3$...

Le modèle

C'est ici que l'on fait des **hypothèses** sur la forme de f.
Par exemple:

- f est **linéaire**:

$$y_i = \omega_1 x_{i,1} + \omega_2 x_{i,2} + \omega_3 x_{i,3} + ...$$

  Il s'agit de trouver les valeurs de ω_1, ω_2, ω_3...

- f est **quadratique**:

$$y_i = \omega_{1,1} x_{i,1}^2 + \omega_{2,2} x_{i,2}^2 + \omega_{3,3} x_{i,3}^2 + \omega_{1,2} x_{i,1} x_{i,2} + \omega_{1,3} x_{i,1} x_{i,3} + \omega_{2,3} x_{i,2} x_{i,3}$$

  A nouveau, on cherche les valeurs des ω.

Contraintes?

De deux choses l'une :

- Soit on a une **connaissance a priori** ou une **hypothèse pertinente** sur la forme de *f* : alors on peut **contraindre** *f*. C'est ce qu'on appelle un **data model**. *Exemples : Régression Linéaire, Analyse Discriminante, Naive Bayes.*
- Soit on ne sait **rien** de *f* : on ne pose aucune contrainte. On parle alors de **modèle algorithmique**. *Exemples : Plus Proches Voisins, Arbres de Décision, Support Vector Machines (SVM), Random Forests, Réseaux de Neurones.*

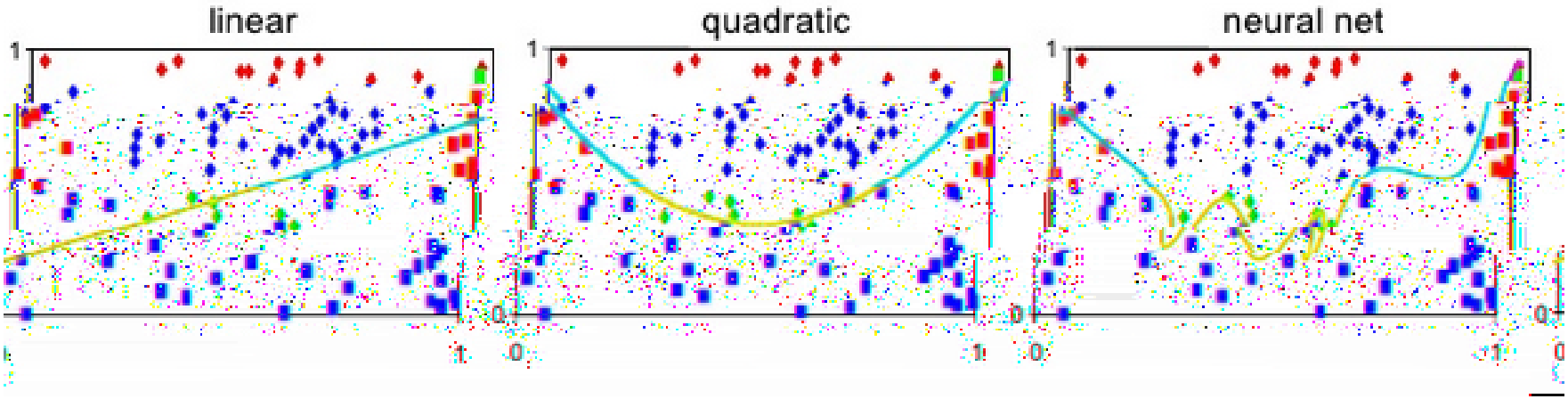


Image empruntée à Pierre Geurts

Deux manières de voir les choses

Statistical Science
2001, Vol. 16, No. 3, 199–231

Statistical Modeling: The Two Cultures

Leo Breiman

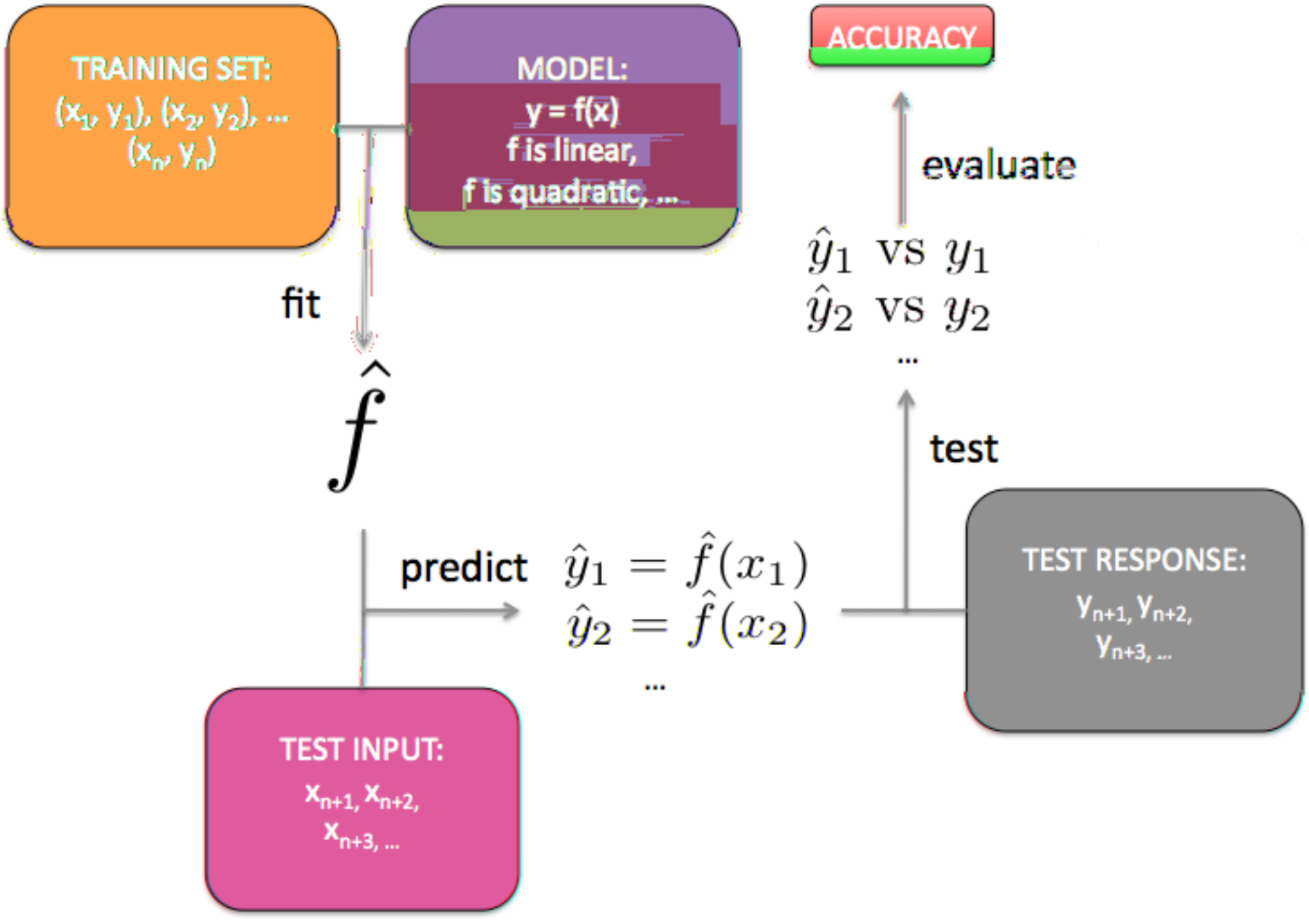
Abstract. There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses algorithmic models and treats the data mechanism as unknown. The statistical community has been committed to the almost exclusive use of data models. This commitment has led to irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems. Algorithmic modeling, both in theory and practice, has developed rapidly in fields outside statistics. It can be used both on large complex data sets and as a more accurate and informative alternative to data modeling on smaller data sets. If our goal as a field is to use data to solve problems, then we need to move away from exclusive dependence on data models and adopt a more diverse set of tools.

Dilemme performance/complexité

- Plus le modèle est **simple**, plus il est facile de l'estimer mais moins il est proche de la réalité.
- Plus le modèle est **complexe**, plus il s'approche de la réalité mais plus on risque de se tromper en l'estimant.

Dilemme complexité/performance : il faut trouver la complexité optimale.

Pipeline


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

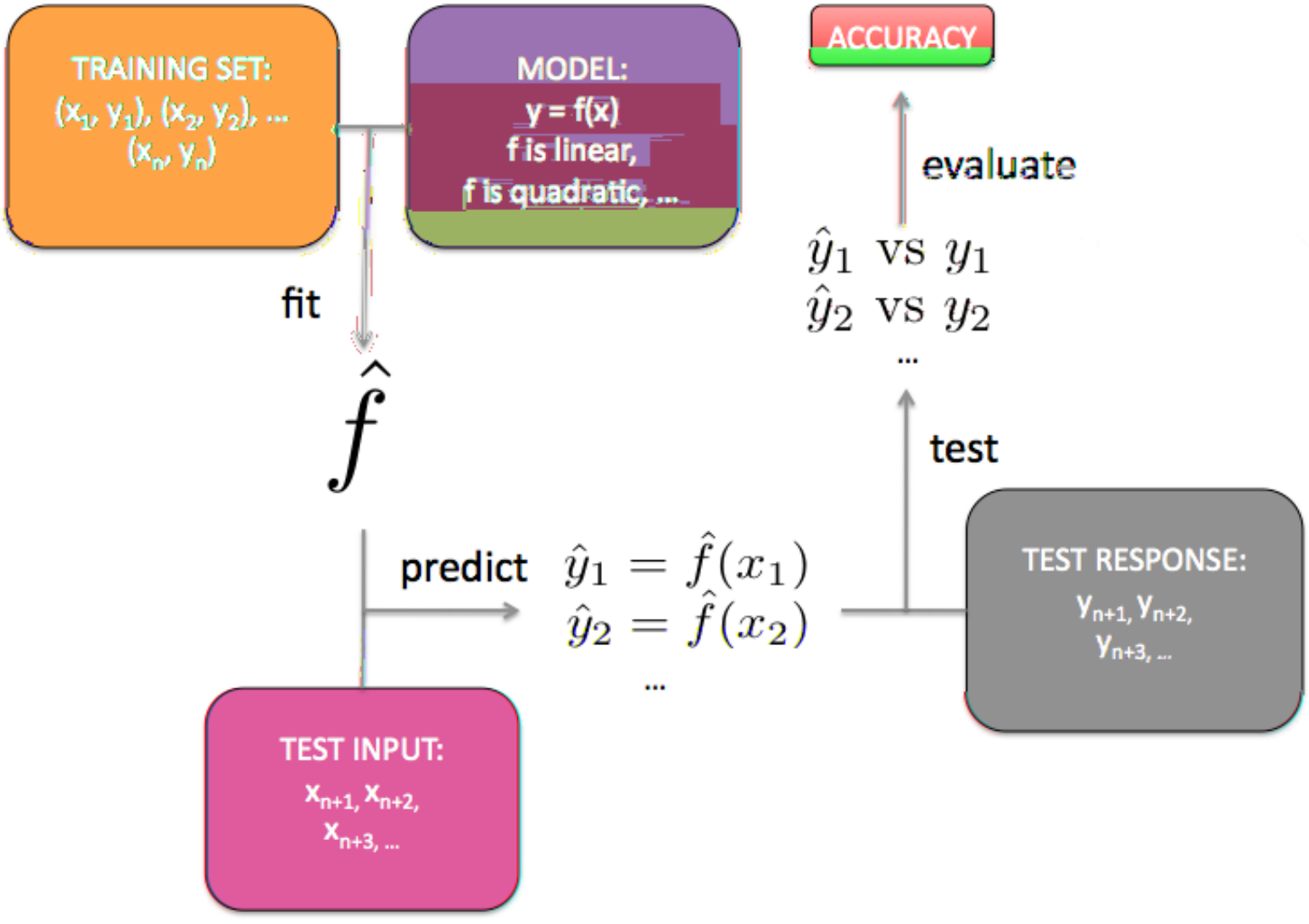
Estimation

La phase **d'estimation** (fit) consiste, en fonction des hypothèses faites sur f, à **estimer la meilleure fonction** f dans le cadre des contraintes imposées.

La meilleure fonction est celle qui se **généralise** le mieux et donne les meilleures prédictions. On appelle cette fonction $\hat{f}$.

La manière de l'estimer dépend du modèle. Nous reviendrons là-dessus plus tard.

Pipeline


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
fit
$\hat{f}$
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

Ensemble de test

Comme l'ensemble d'apprentissage, l'ensemble de test est composé d'exemples de la forme (x_i, y_i) : pour chaque exemple i, on connaît la valeur d'entrée x_i et la réponse y_i.

Ce sont des exemples que l'on a **mis de côté** au départ.

Prédiction

Une fois le modèle estimé sur l'ensemble d'apprentissage, on l'utilise pour prédire les valeurs de l'ensemble de test. Pour chaque x_i du test, on prédit la réponse $\hat{y}_i$ avec $\hat{f}$:

$$\hat{y}_i = \hat{f}(x_i)$$

On peut alors **comparer** le $\hat{y}_i$ prédit avec le "vrai" y_i. Si les prédictions sont bonnes, le modèle est bon.

Evaluation

En fonction du cas:

- **Régression** : distance moyenne entre les prédictions et les vraies valeurs

$$erreur = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (y_i - \hat{y}_i)^2$$

- **Classification** : pourcentage des fois où le modèle a trouvé la bonne classe

$$erreur = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \delta(\hat{y}_i = y_i)$$

où $\delta(A) = 1$ si A est vraie, 0 sinon.

Nous verrons plus tard des manières avancées d'évaluer l'algorithme.

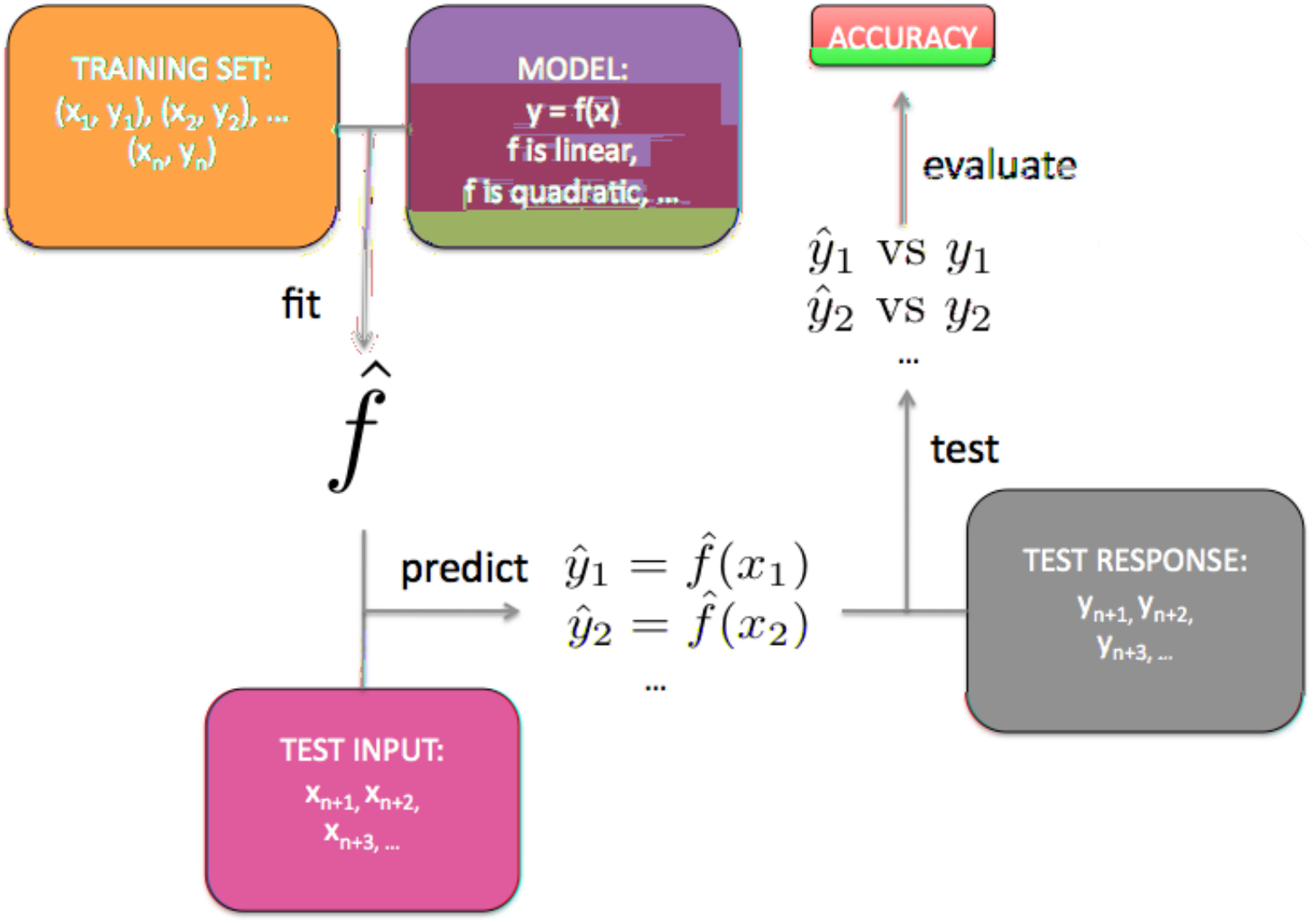
Sur-apprentissage

On parle de **sur-apprentissage** (over-fitting) lorsque l'algorithme apprend **par coeur** l'ensemble d'apprentissage mais n'arrive pas à **généraliser** sur l'ensemble de test.

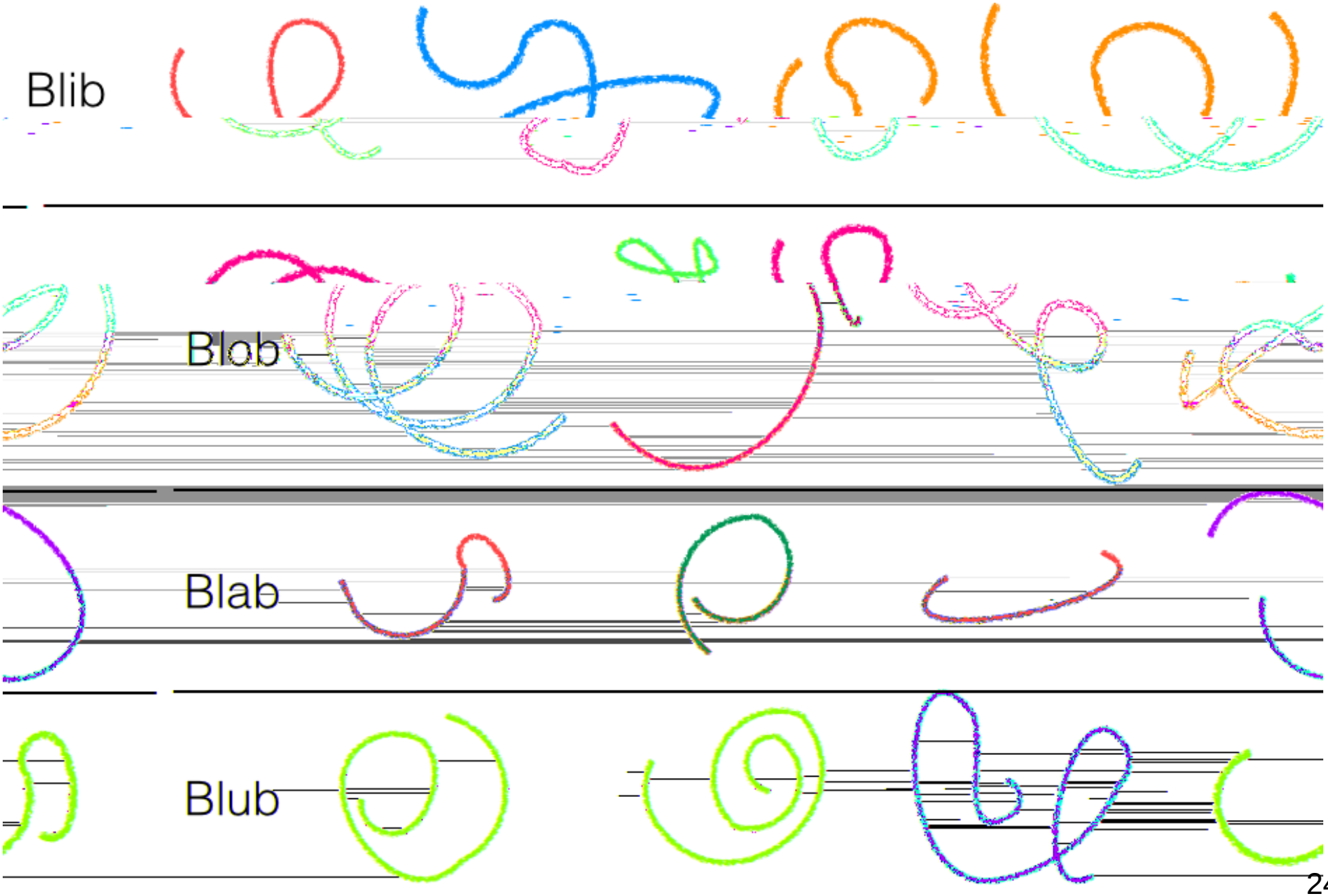
C'est pourquoi il est très important de **tester** le modèle.

Nous reviendrons sur ce point crucial en fin de chapitre.

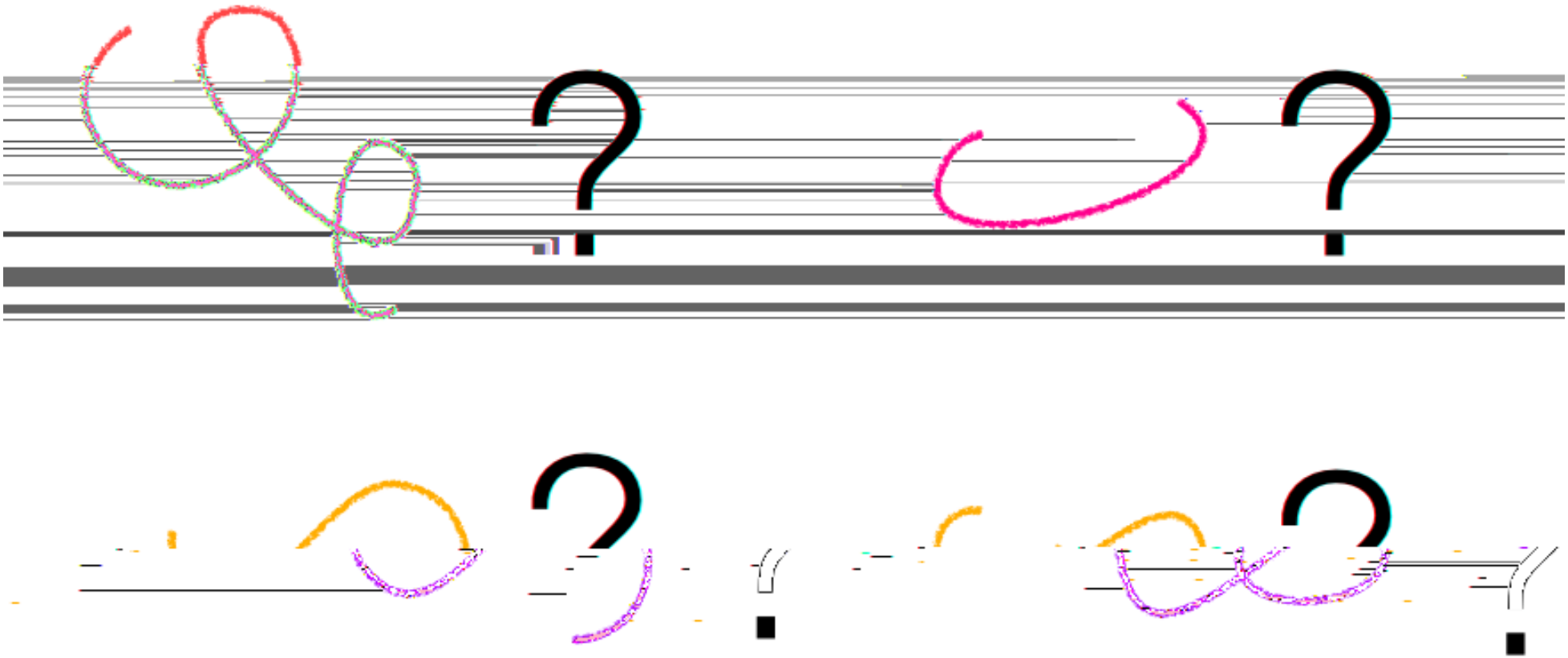
Pipeline


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
fit
$\hat{f}$
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

A la place de l'ordinateur...



Que prédiriez-vous ?



Comment apprendre ?

- Les techniques d'apprentissage dépendent du problème.
- Plusieurs techniques peuvent fonctionner.
- La performance d'un algorithme dépend beaucoup de l'encodage de vos données.

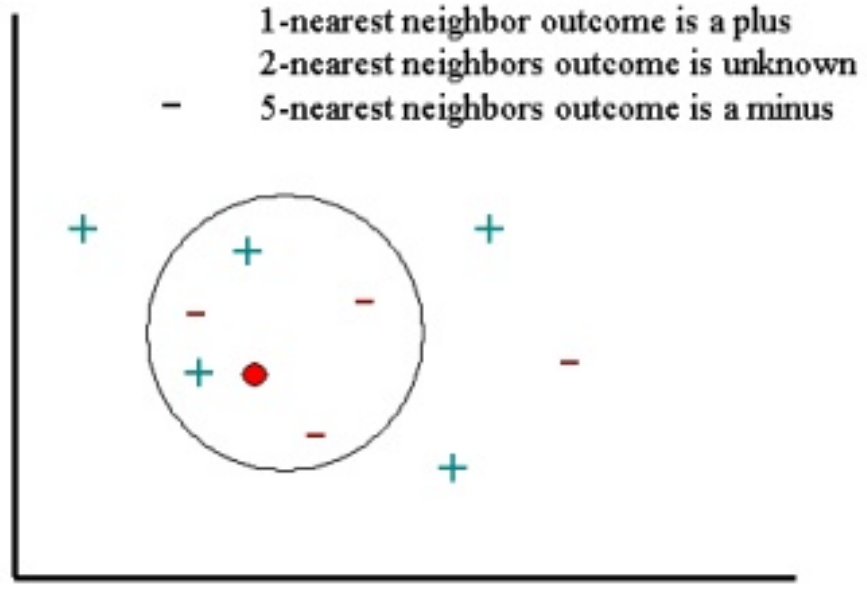
Nous allons voir différentes classes d'algorithmes qui ne **réfléchissent pas de la même manière**.

Outline

Outline

Principe

K-plus proches voisins (K-nearest neighbors) : La valeur d'un point dépend de celles des points proches de lui.



Source : statsoft.com

Modélisation / Apprentissage

Presque rien à faire, si ce n’est choisir un **nombre de voisins** et une **distance**.

Classification

Pour chaque élément de l'ensemble de test, on regarde la valeur des points voisins de l'ensemble d'apprentissage et on procède à un **vote majoritaire**.

Algorithm 1 Classification K-plus proches voisins

1: **INPUT :** : Ensemble d'apprentissage (X_{train}, Y_{train}), ensemble de test (X_{test}, Y_{test}), distance D, nombre de voisins K
2: **for** Chaque point du test x **do**
3: Calculer la distance avec chacun des points d'apprentissage;
4: Choisir les K voisins les plus proches au sens de la distance D;
5: Assigner à x la classe la plus fréquente chez ses K voisins;
6: En cas d'indécision, choisir la classe du voisin le plus proche ;
7: **end for**

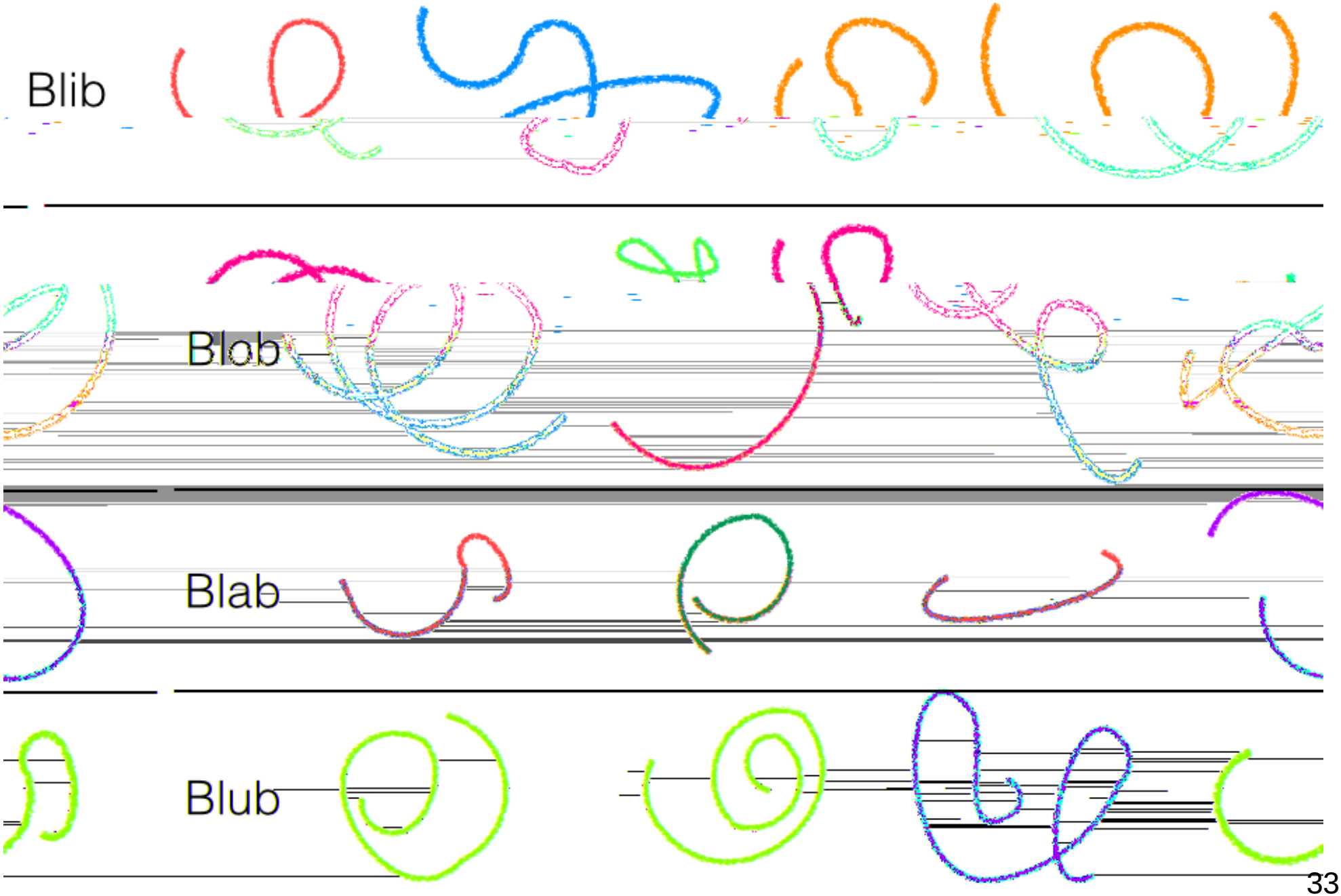
Régression

Pour chaque élément de l'ensemble de test, on regarde la valeur des points voisins de l'ensemble d'apprentissage et on procède à une **moyenne**.

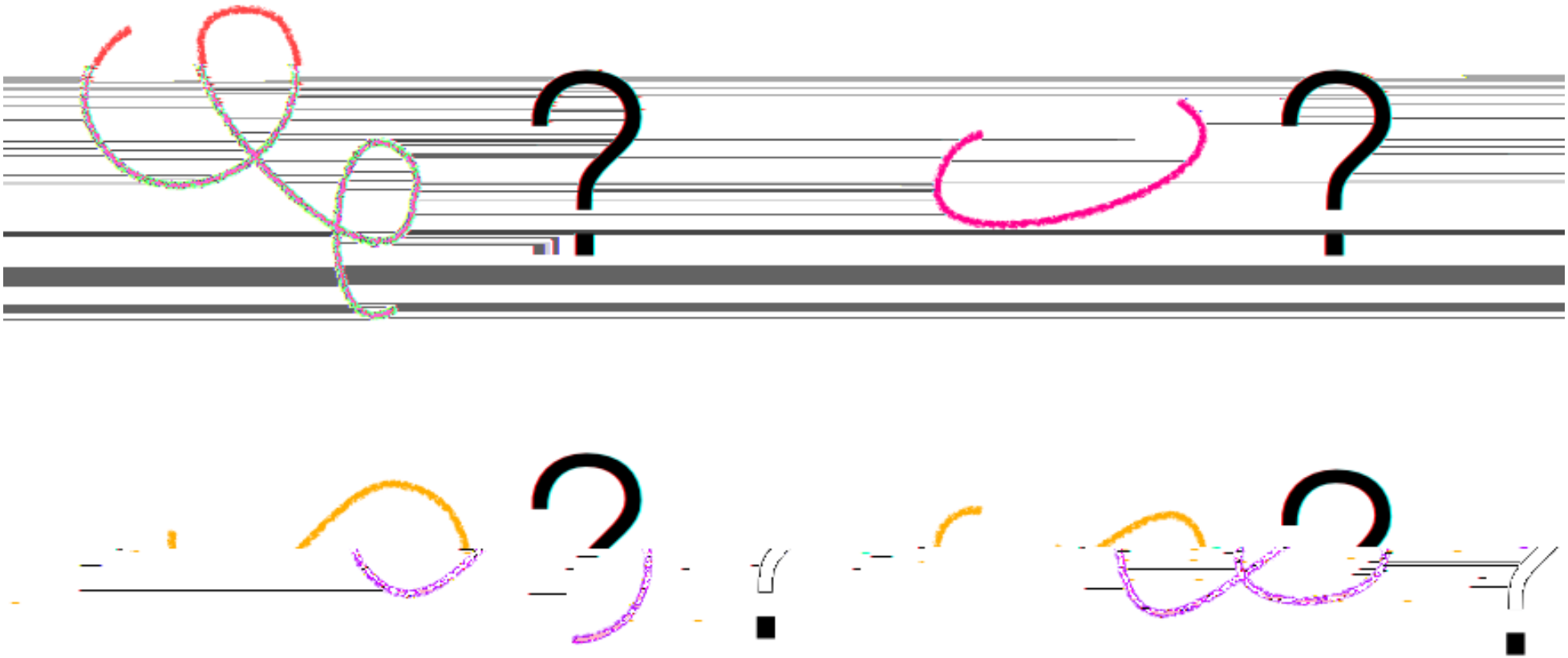
Algorithm 2 Régression K-plus proches voisins

1: **INPUT :** : Ensemble d'apprentissage (X_{train}, Y_{train}), ensemble de test (X_{test}, Y_{test}), distance D, nombre de voisins K
2: **for** Chaque point du test x **do**
3: Calculer la distance avec chacun des points d'apprentissage;
4: Choisir les K voisins les plus proches au sens de la distance D;
5: Assigner à x la moyenne des réponses de ses K voisins;
6: **end for**

Blobs, blibs, blabs et blubs

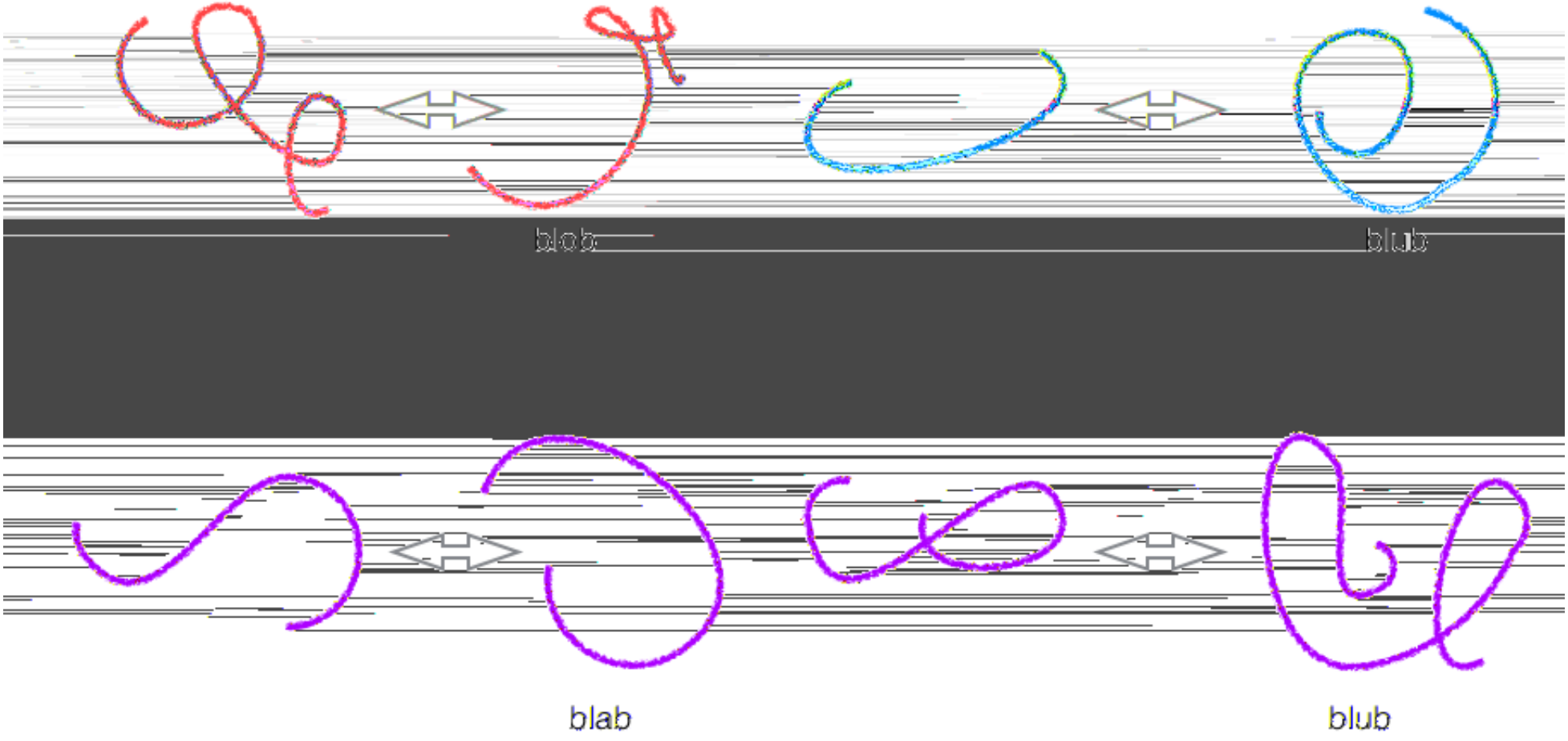


Blobs, blibs, blabs et blubs



Blobs, blibs, blabs et blubs

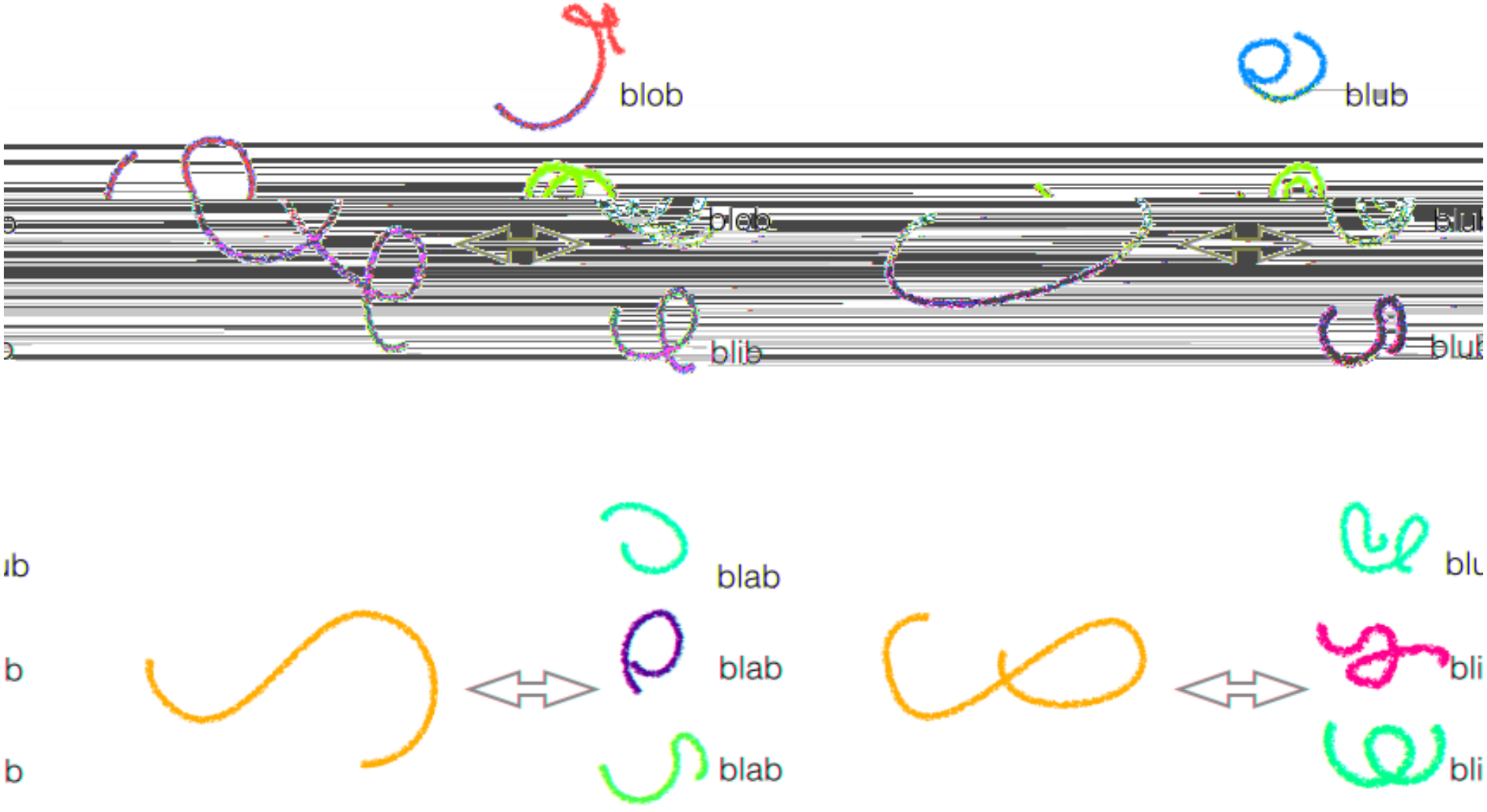
Algorithme du plus proche voisin (K=1):



Résultat : $4/4 = 100\%$

Blobs, blibs, blabs et blubs

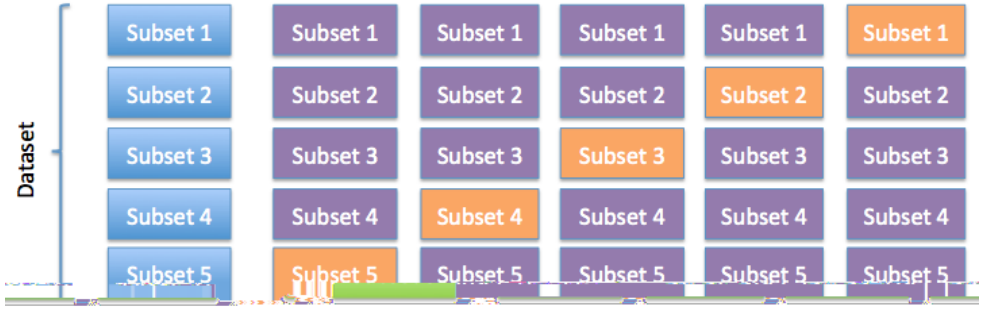
Algorithme du 3-plus proches voisins (K=3):



Résultat : $3/4 = 75\%$

Comment choisir K? La validation croisée.

Afin de choisir la meilleure valeur pour K, on va procéder par **validation croisée** (cross-validation).



- On **divise** l'ensemble d'apprentissage en N parties (ici, $N = 5$).
- Tour à tour, on **entraîne** le modèle sur $N/(N-1)$ parties et on le **teste** sur la partie restante.
- On calcule la **performance moyenne** sur les N tests.
- On procède ainsi pour différentes valeurs de K.
- On choisit la valeur de K qui a la **meilleure performance moyenne**.

Validation croisée, suite

Pourquoi utiliser la validation croisée ?

- Lors du processus d'apprentissage, on ne peut pas utiliser l'ensemble de test, à aucun moment, ce serait "tricher".
- En effet, si l'on utilise l'ensemble de test pour apprendre, le modèle sera forcément bon en test : **sur-apprentissage**.
- Donc il faut choisir les paramètres sur l'ensemble d'apprentissage uniquement.
- La validation croisée est un moyen de **tester durant l'apprentissage**.

Cette méthode est valable pour tous les algorithmes. C'est ce qu'on utilisera pour choisir le bon paramètre

Conclusion sur les K-plus proches voisins

Avantages :

- Extrêmement simple
- Intuition : faire comme ses voisins
- Efficace dans certains cas et si la distance est bien choisie.

Inconvénients:

- Il faut choisir K
- Calculs potentiellement lourds si le nombre de points/le nombre de variables est grand.

En python :

```
from sklearn.cluster import KNeighborsClassification,
KNeighborsRegression
model = KNeighborsClassification(n_clusters = 5)
model.fit(Xtrain, Ytrain)
predictions = model.predict(Xtest)
```

Outline

Une intuition simple...

- Principe : réfléchir dimension par dimension, probablement le plus proche d'une décision humaine.
- Les arbres s'utilisent dans des problèmes de régression et de classification.
- Ils acceptent les données de types différents : qualitatives/quantitatives, discrètes/continues.
- Ils sont très facilement **interprétables**.

Rappels: variables discrètes et continues

- On dit qu'une variable est **discrète** lorsqu'elle prend un nombre dénombrable de valeurs. (Dénombrable $\neq$ fini.) Par exemple : nombre de pièces d'un appartement (1,2,3,4,5...), nom de l'auteur d'un livre (Victor Hugo, Stendhal, Balzac...), match à domicile (oui/non), etc.
- On dit qu'une variable est **continue** lorsqu'elle prend un nombre non dénombrable de valeurs (i.e. qu'on ne peut pas compter, même si on a un temps infini). Par exemple : prix d'un appartement (tous les prix possibles entre 0 et l'infini), valeur TF/IDF (toutes les valeurs entre 0 et 1), etc.

Remarque 1 : on peut **discrétiser** une variable continue. Exemple du prix d'un appartement : classe 1 (entre 0 et 500), classe 2 (entre 500 et 700), classe 3 (entre 700 et 800), etc.

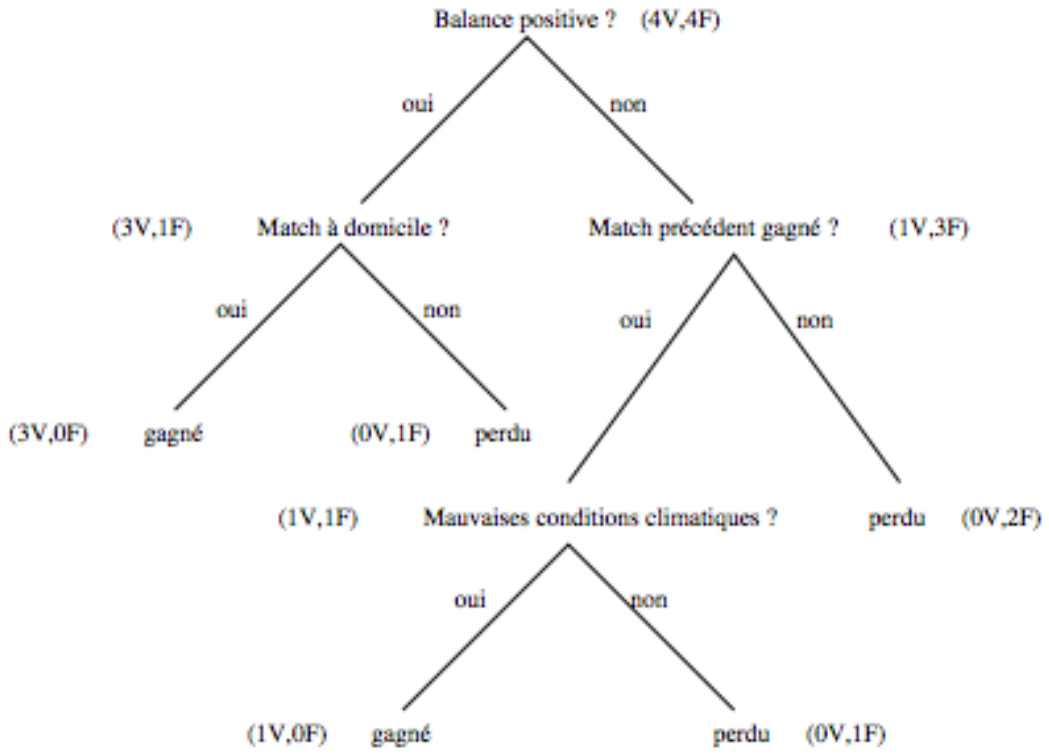
Remarque 2 : si une variable discrète a trop de valeurs, on peut la considérer comme continue.

Variables discrètes : phase d'apprentissage

Match à domicile ?	Balance positive ?	Mauvaises conditions climatiques ?	Match précédent gagné ?	Match gagné
V	V	F	F	V
F	F	V	V	V
V	V	V	F	V
V	V	F	V	V
F	V	V	V	F
F	F	V	F	F
V	F	F	V	F
V	F	V	F	F

Source : Cours de François Denis

Variables discrètes : phase d'apprentissage



Source : Cours de François Denis

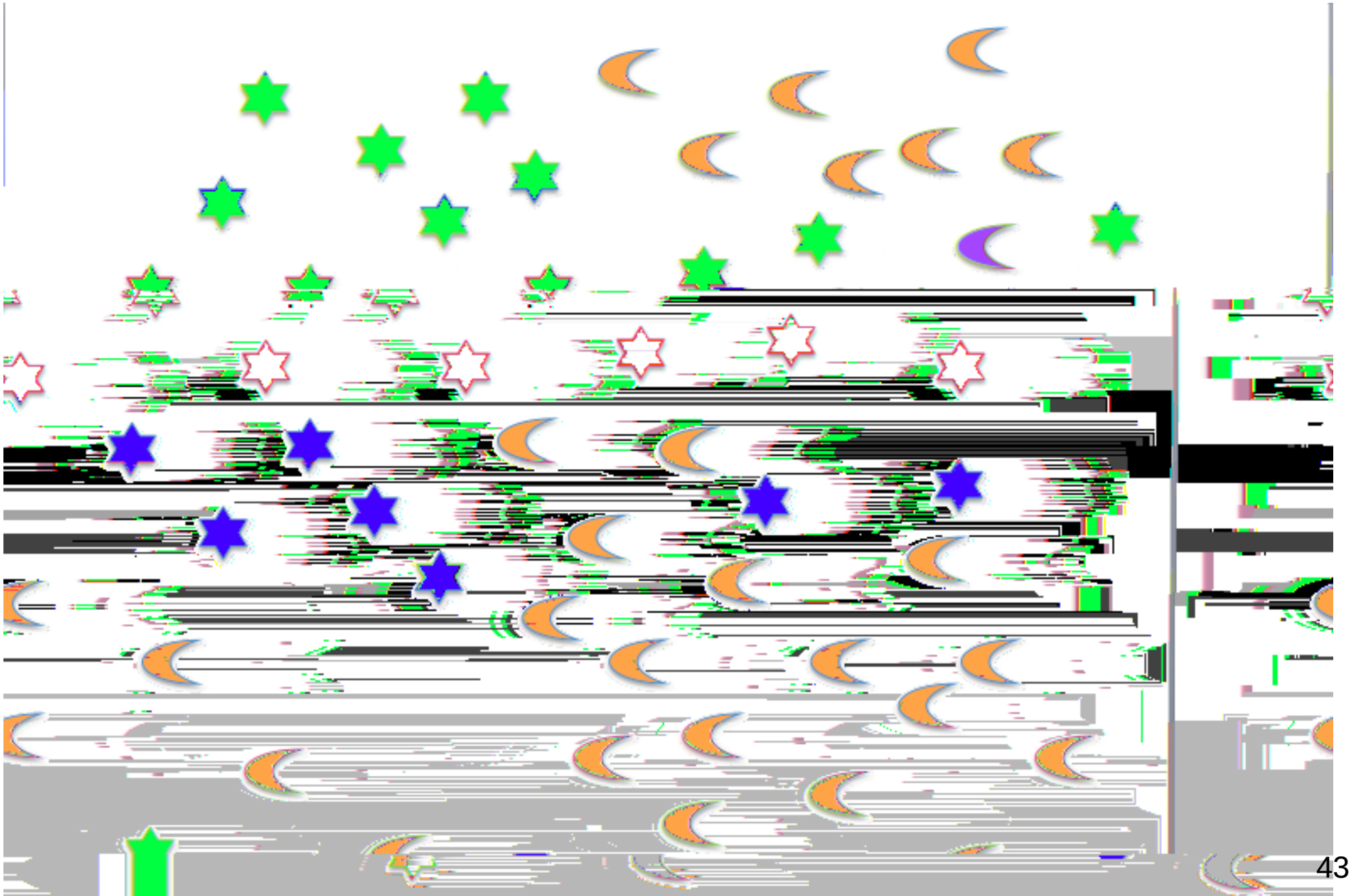
Variables discrètes : phase de test

Le prochain match aura les caractéristiques suivantes:

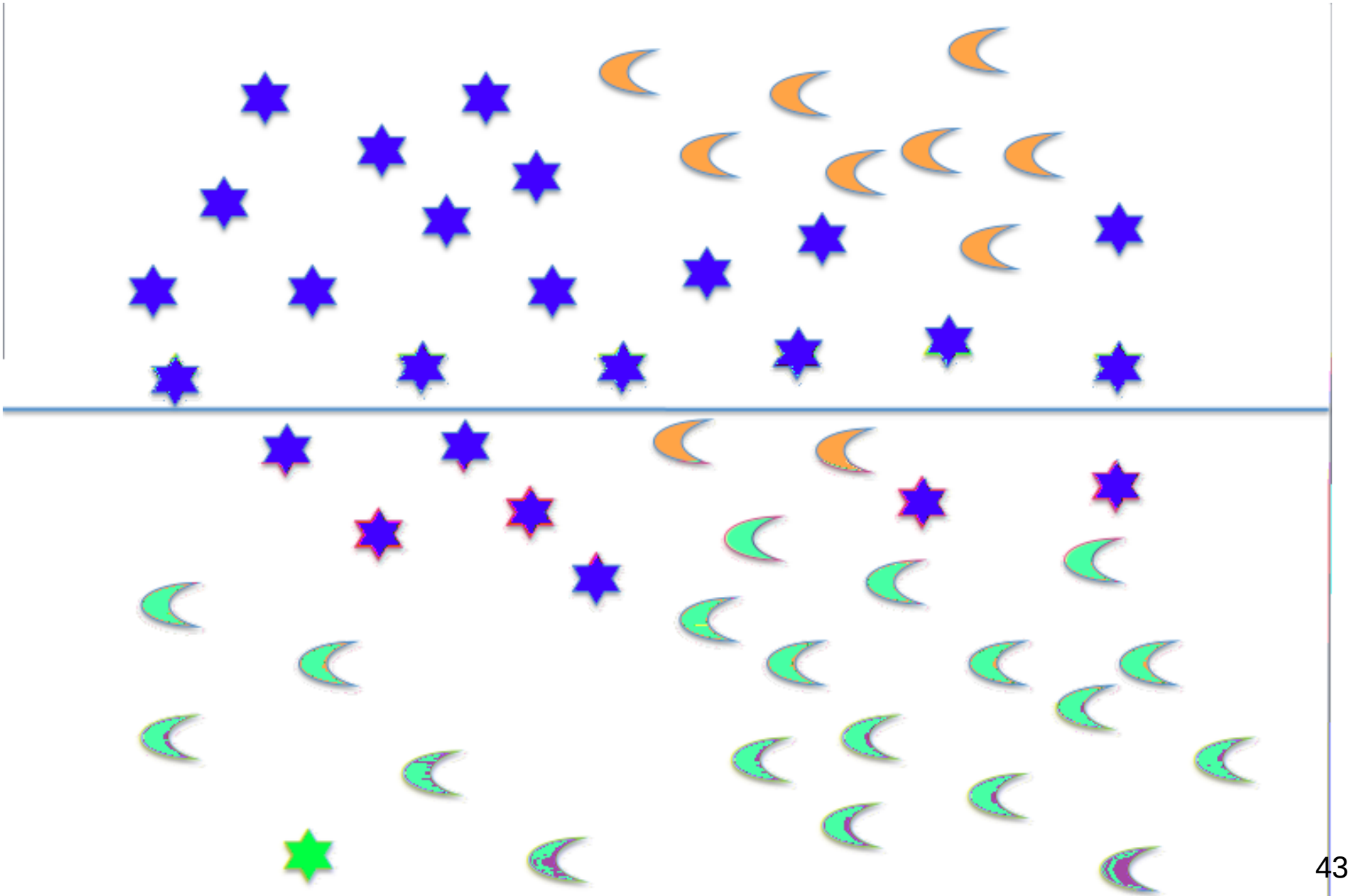
- à domicile
- balance négative
- bonnes conditions climatiques
- match précédent perdu

=> Prédiction : le match sera perdu.

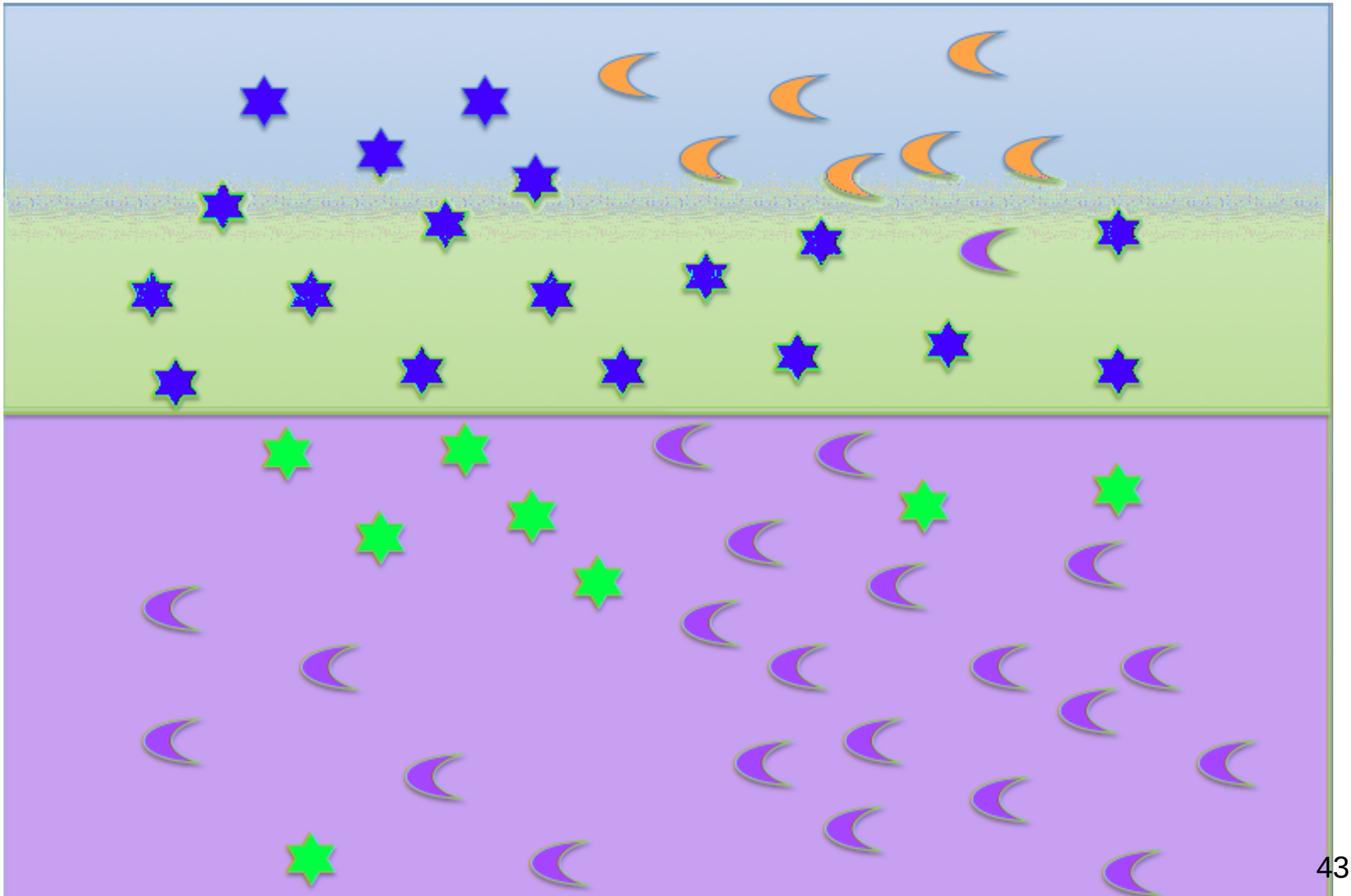
Variables continues : phase d'apprentissage



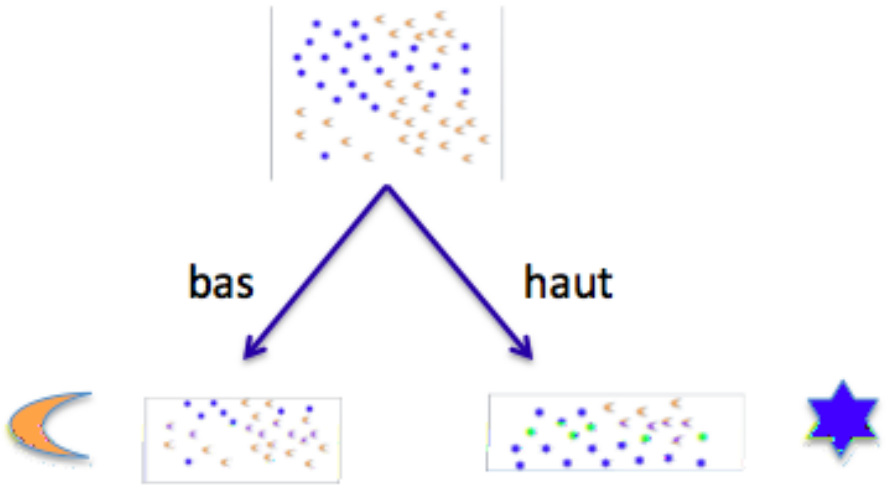
Variables continues : phase d'apprentissage



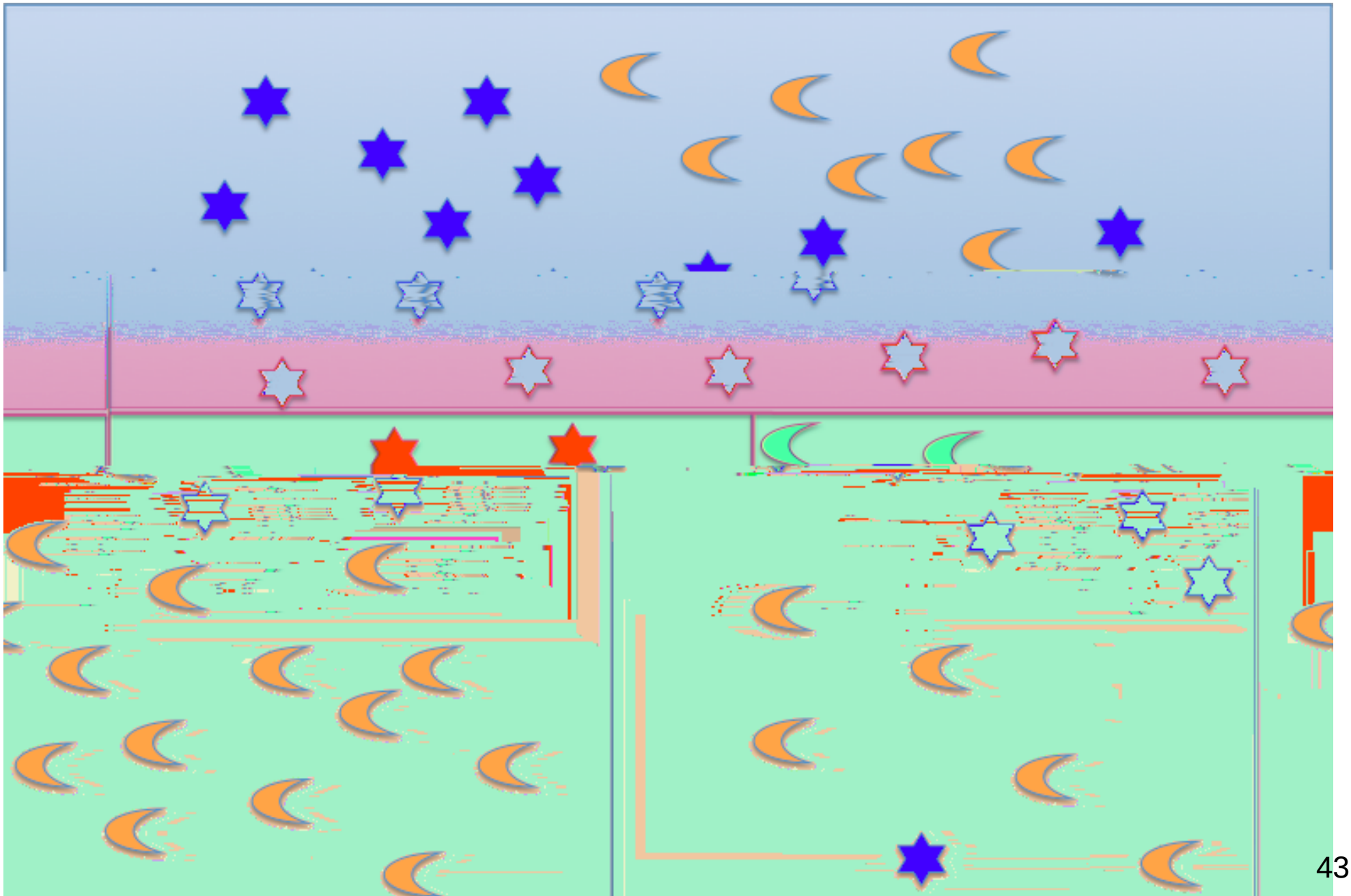
Variables continues : phase d'apprentissage



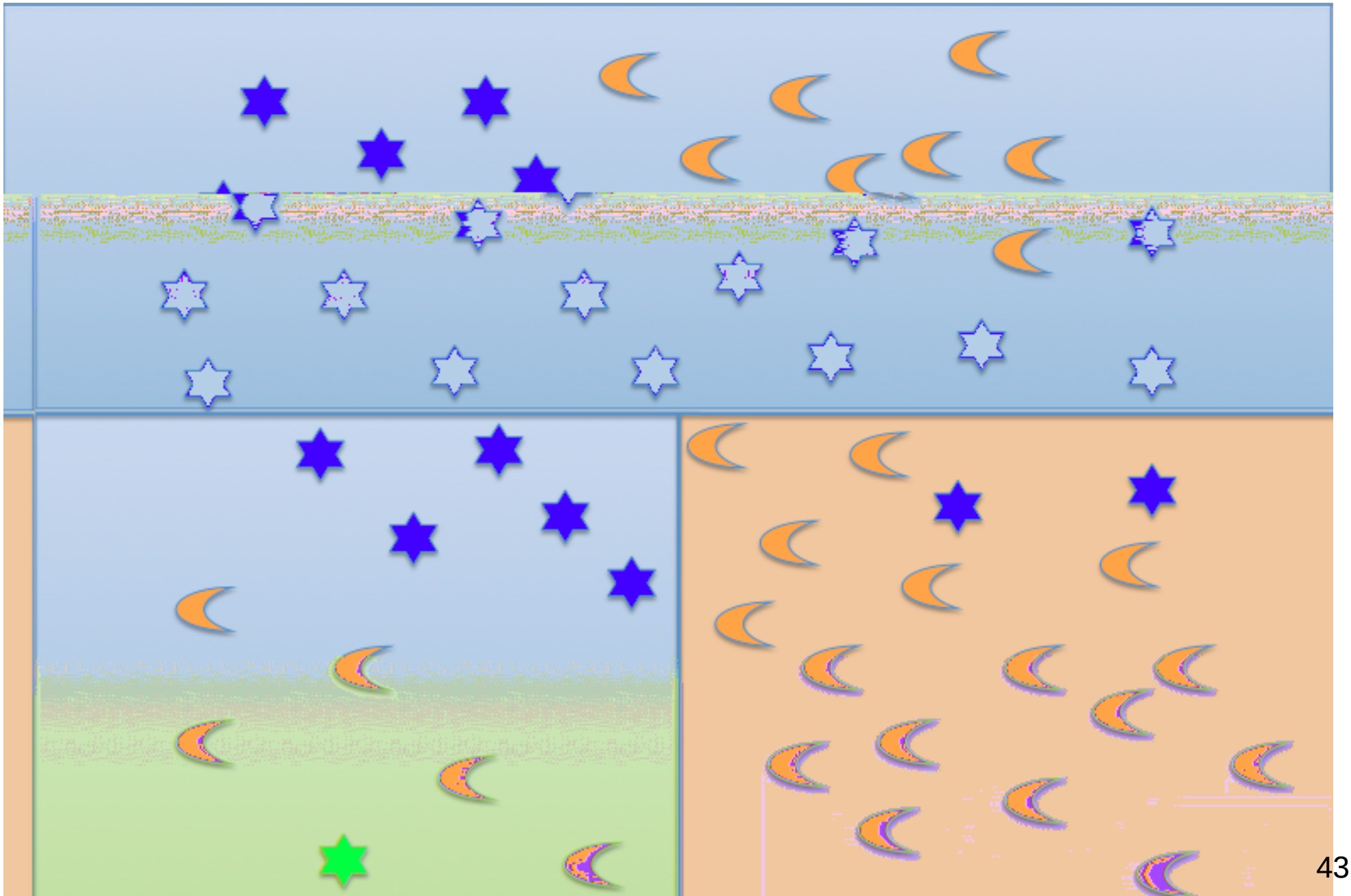
Variables continues : phase d’apprentissage



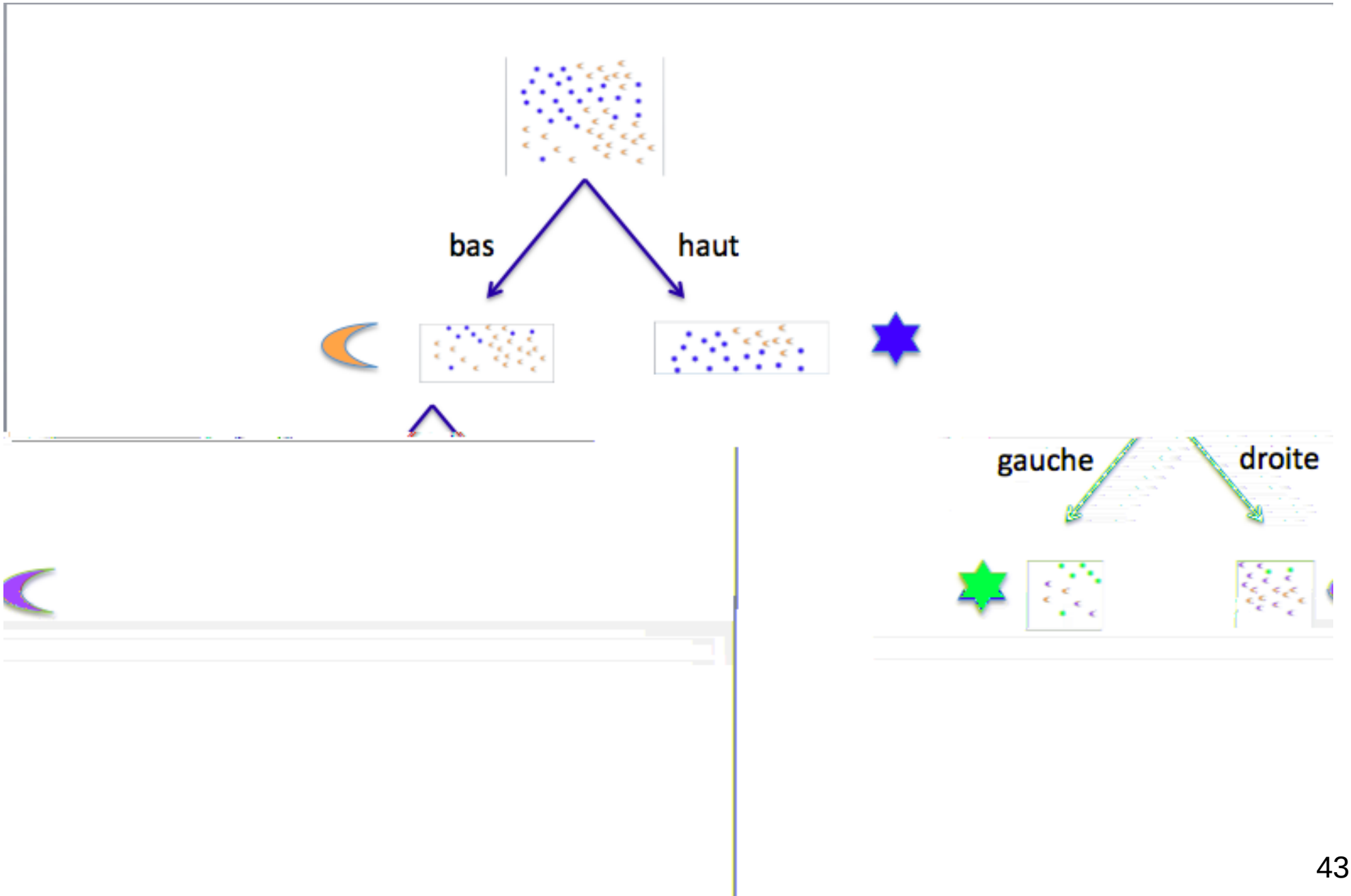
Variables continues : phase d’apprentissage



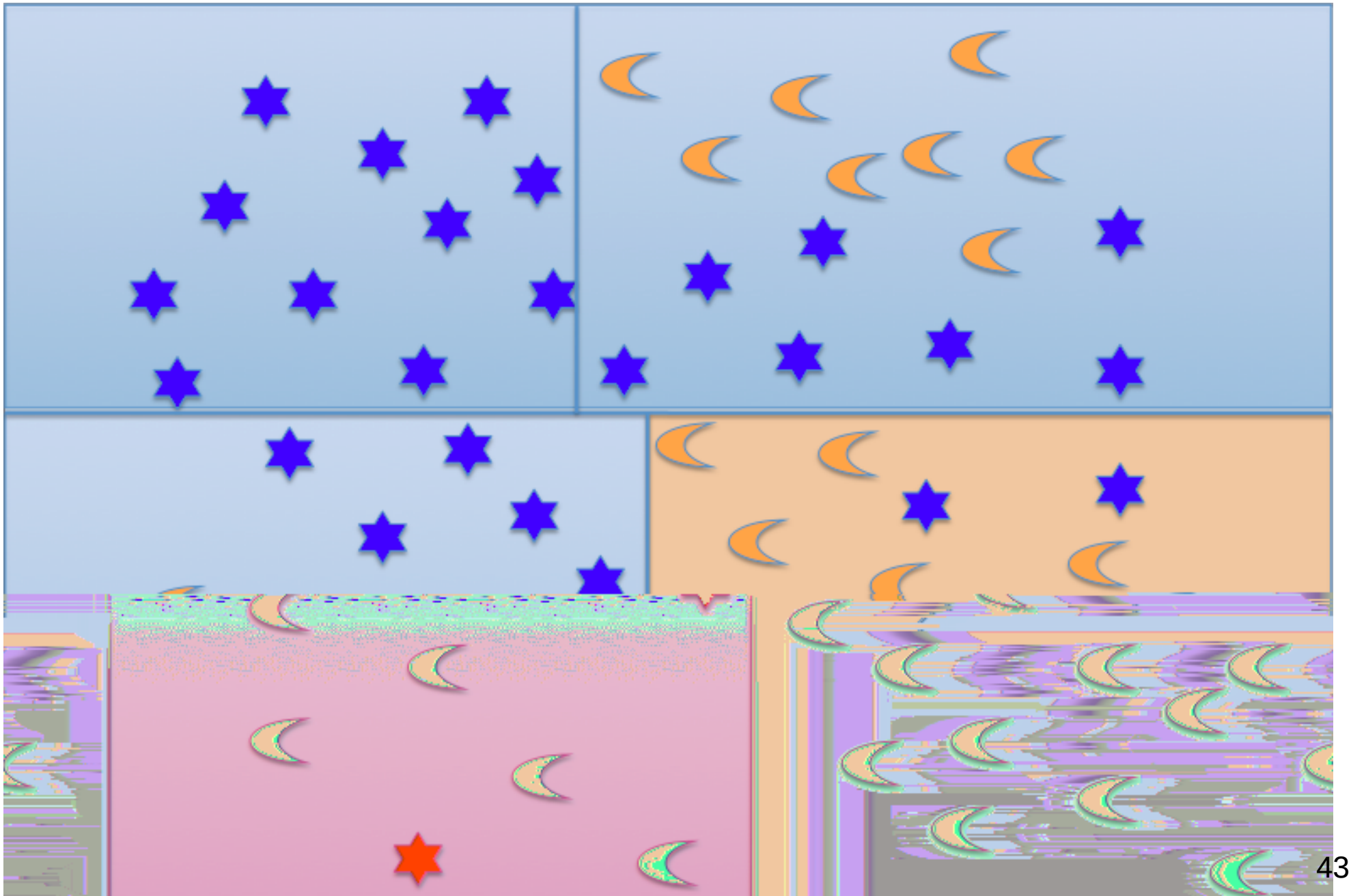
Variables continues : phase d'apprentissage



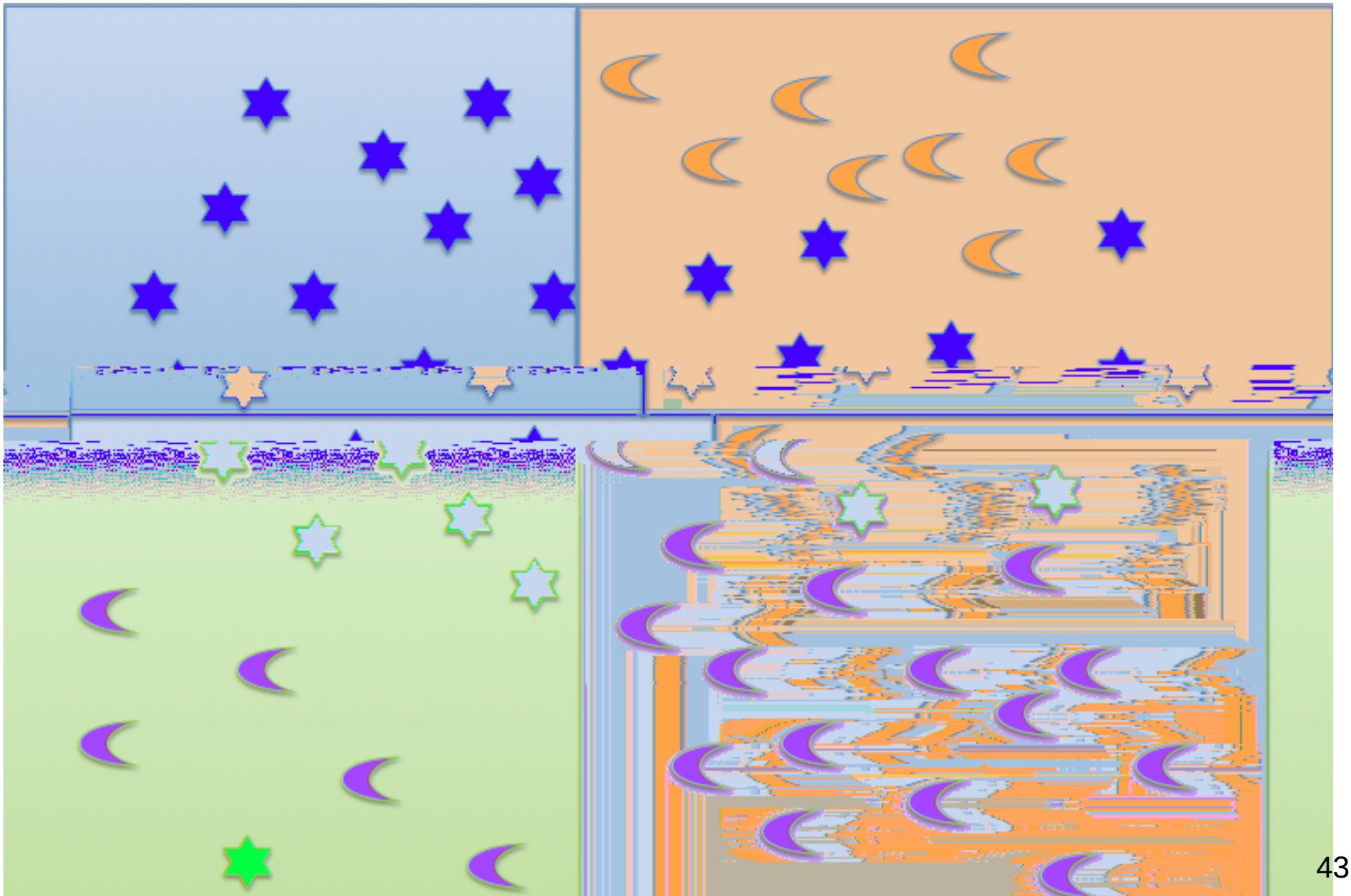
Variables continues : phase d'apprentissage



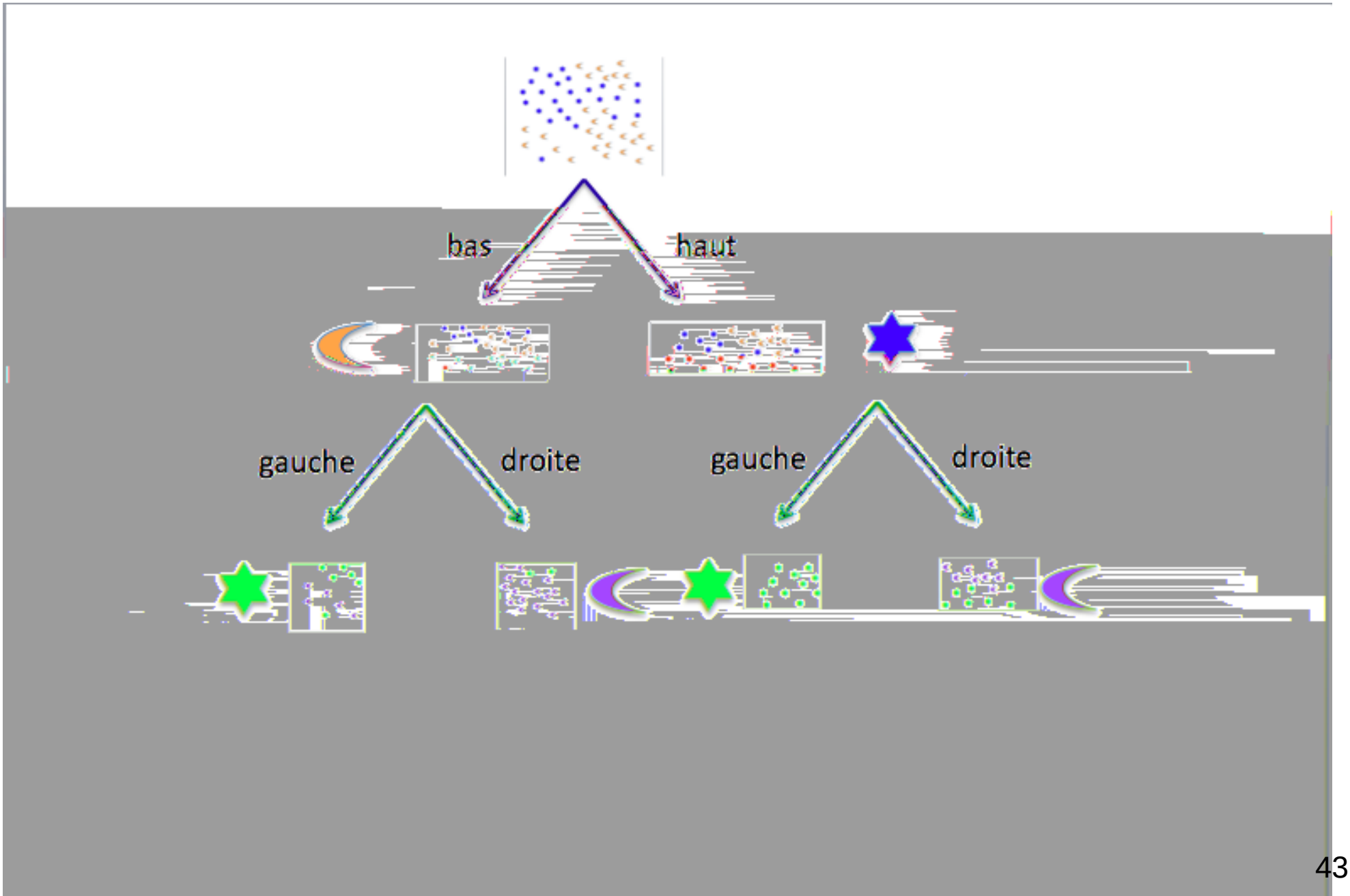
Variables continues : phase d'apprentissage



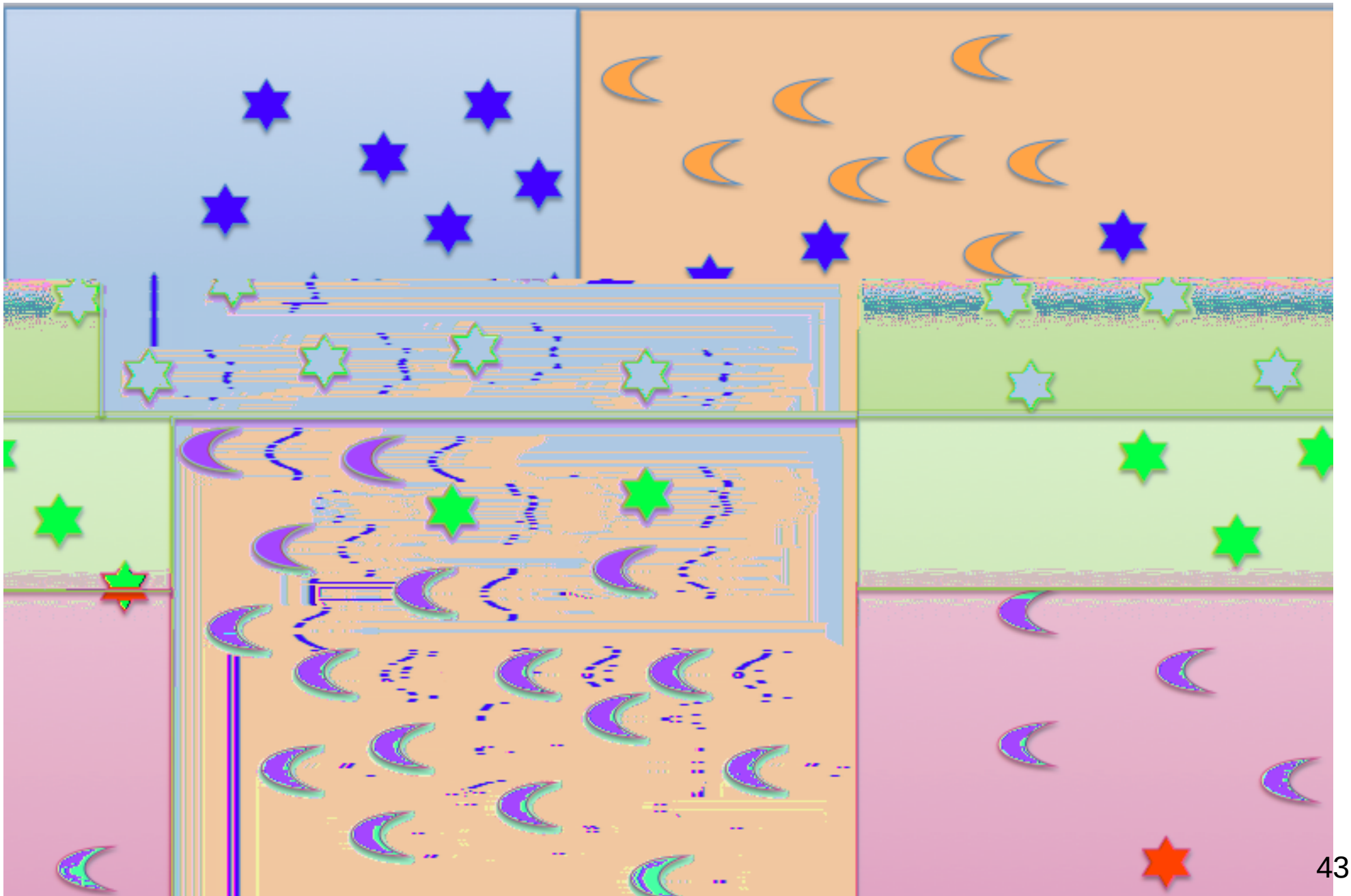
Variables continues : phase d'apprentissage



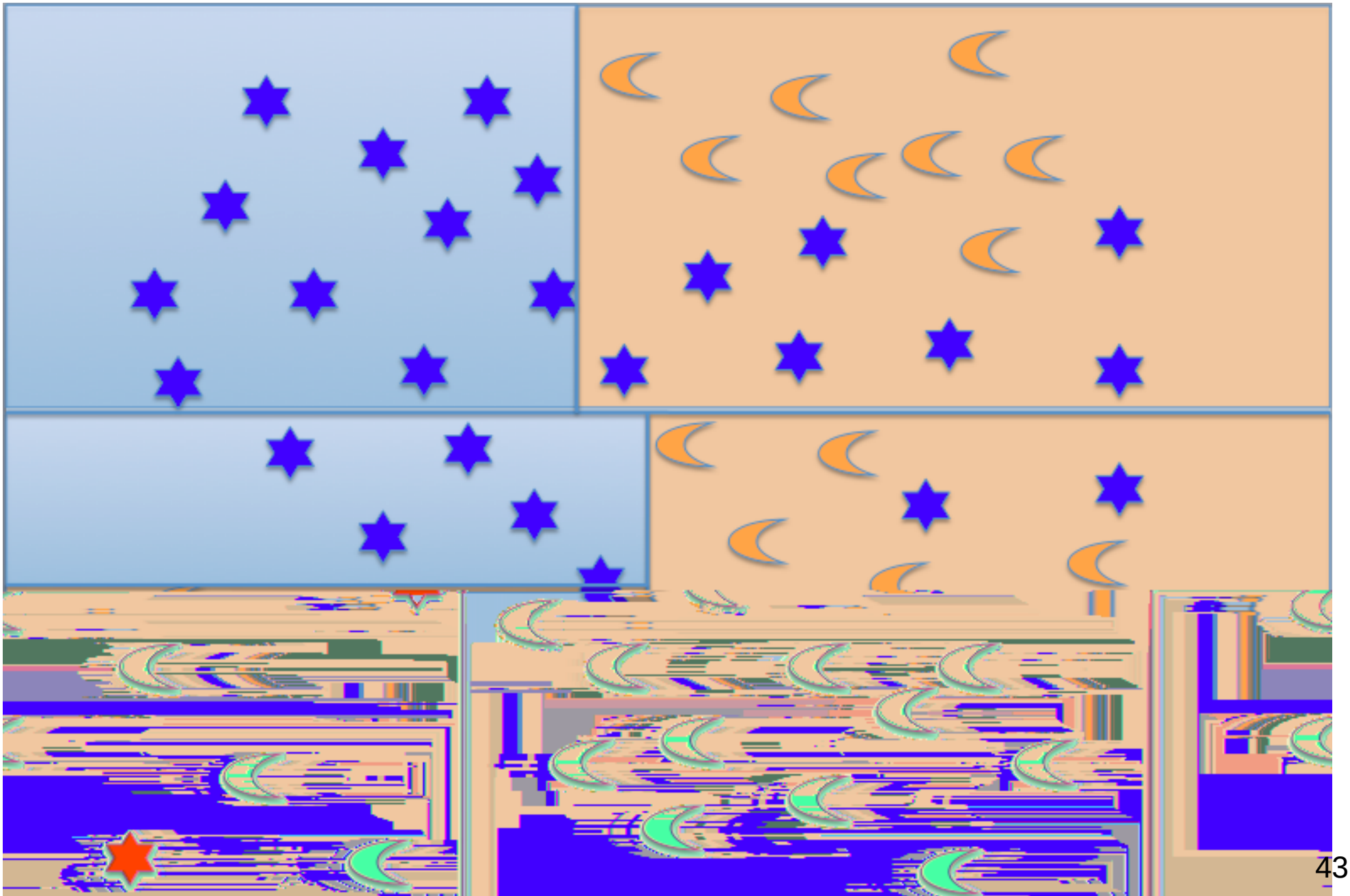
Variables continues : phase d'apprentissage



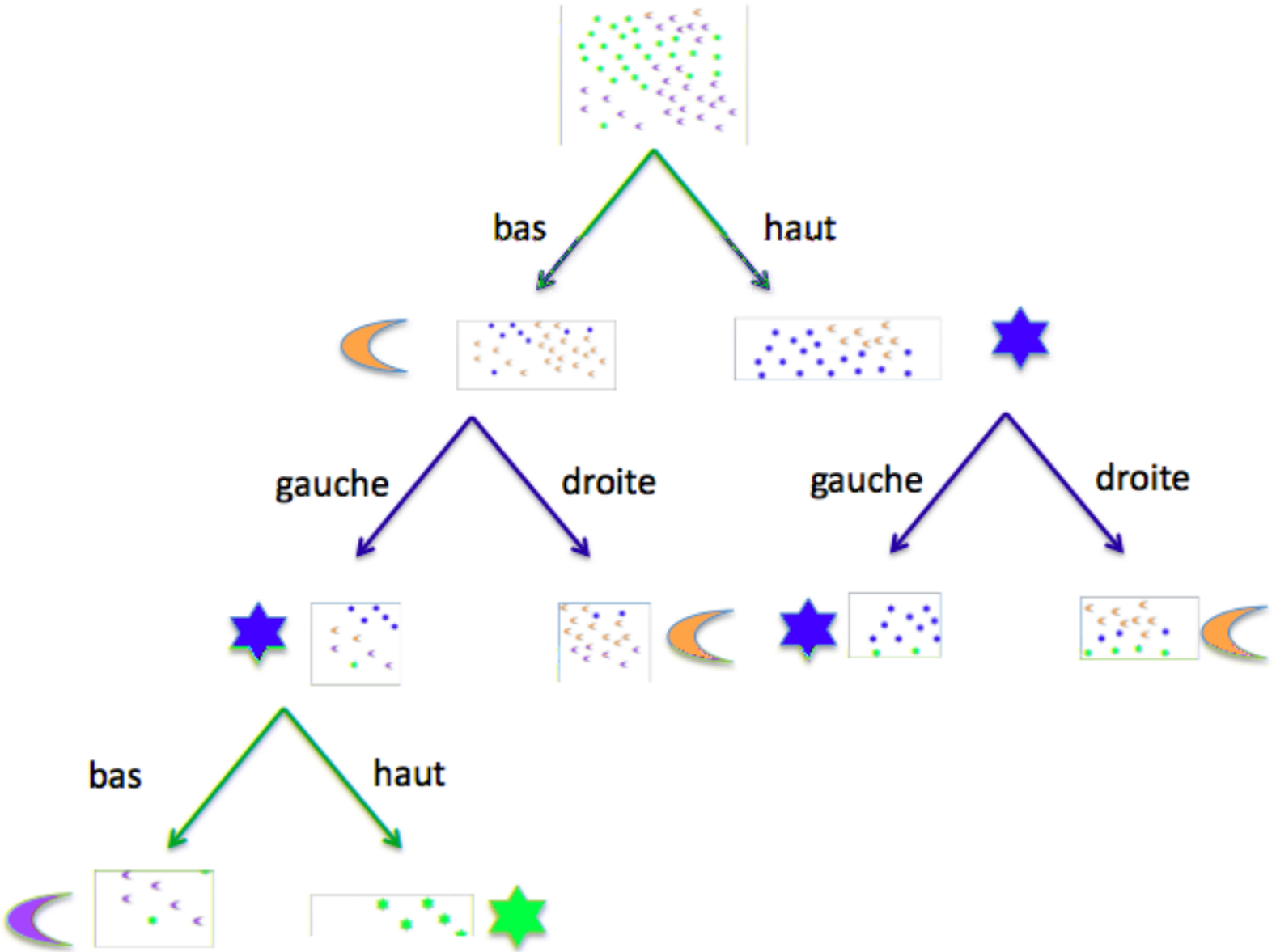
Variables continues : phase d’apprentissage



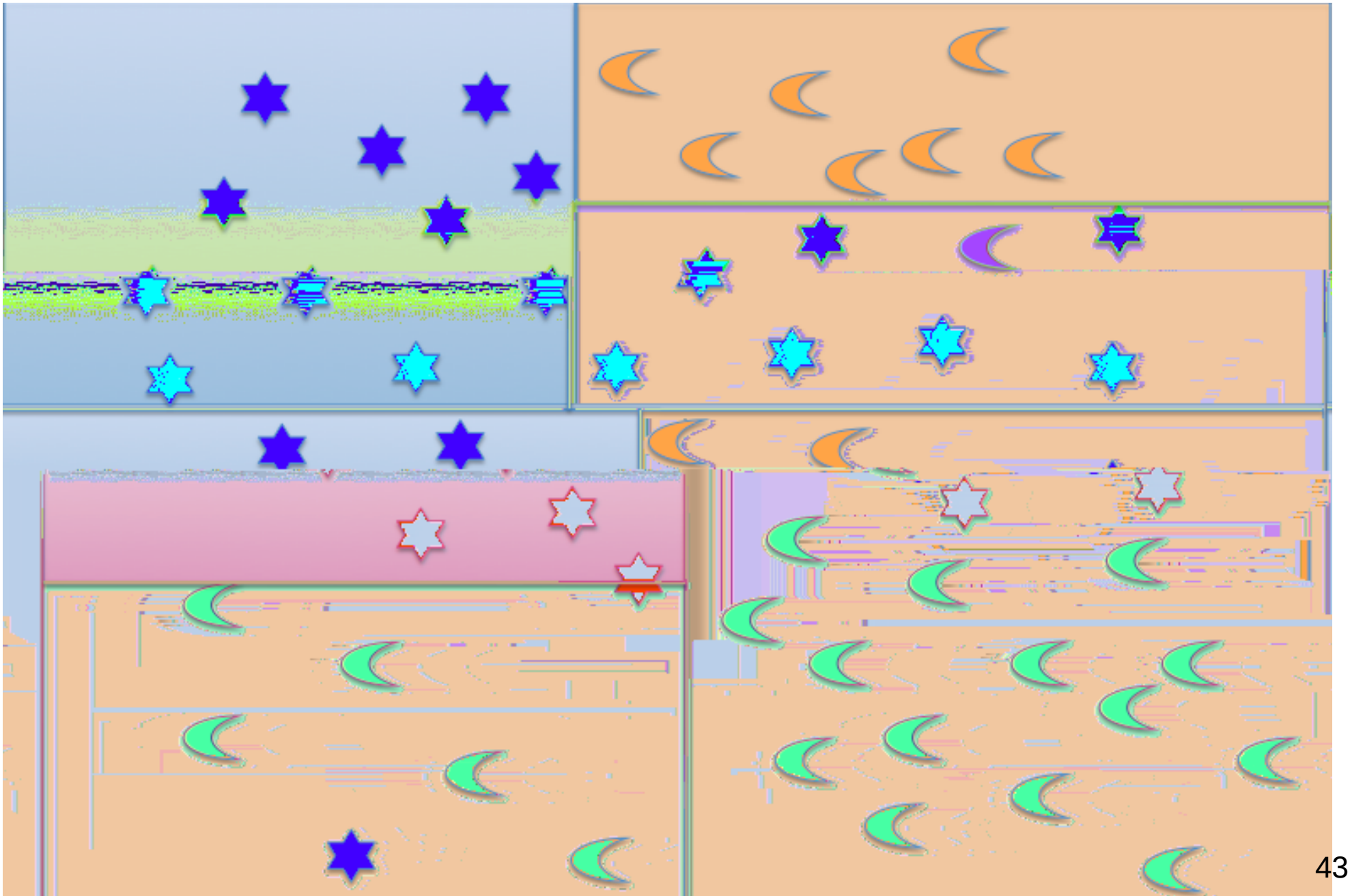
Variables continues : phase d'apprentissage



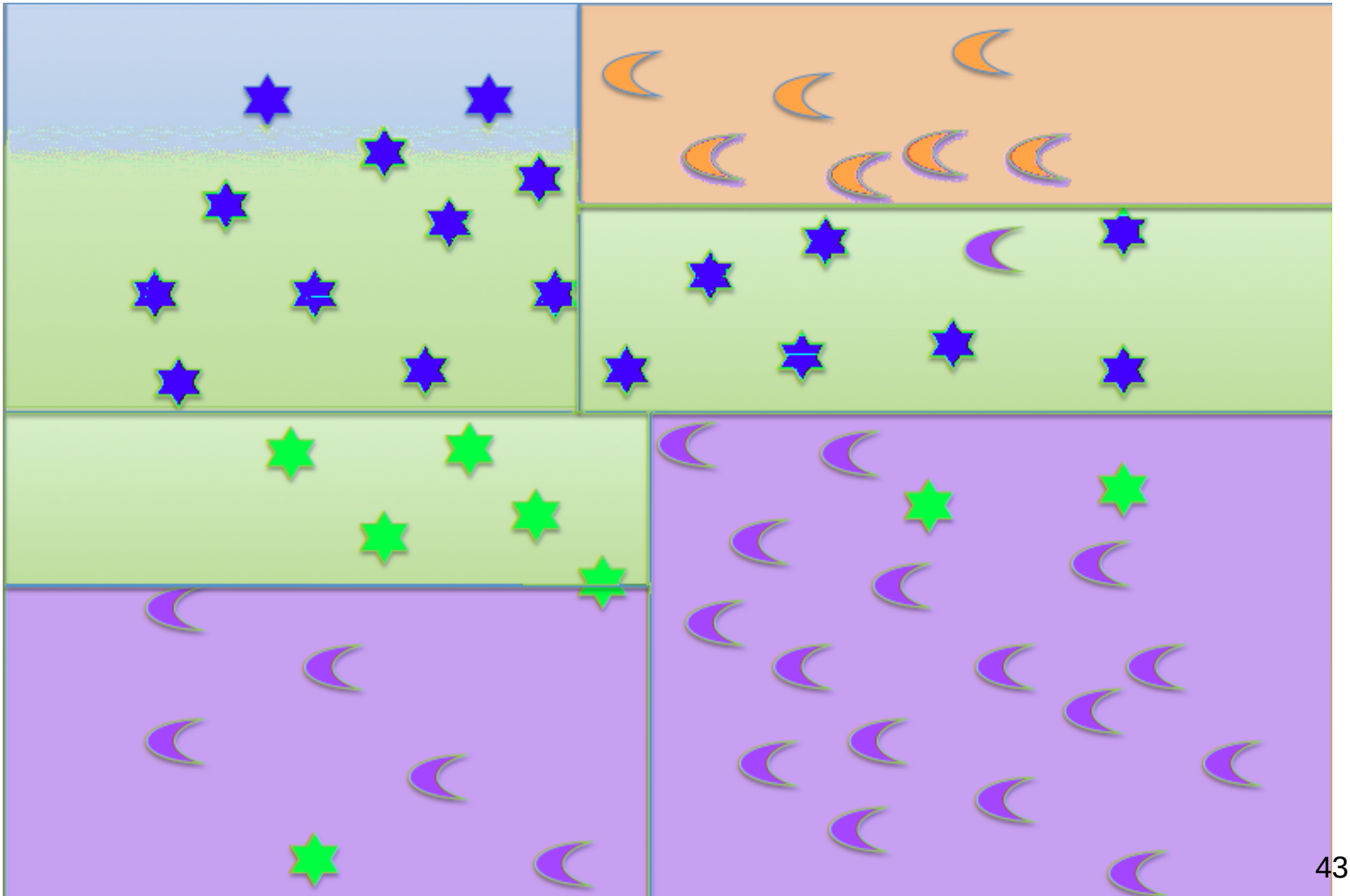
Variables continues : phase d'apprentissage



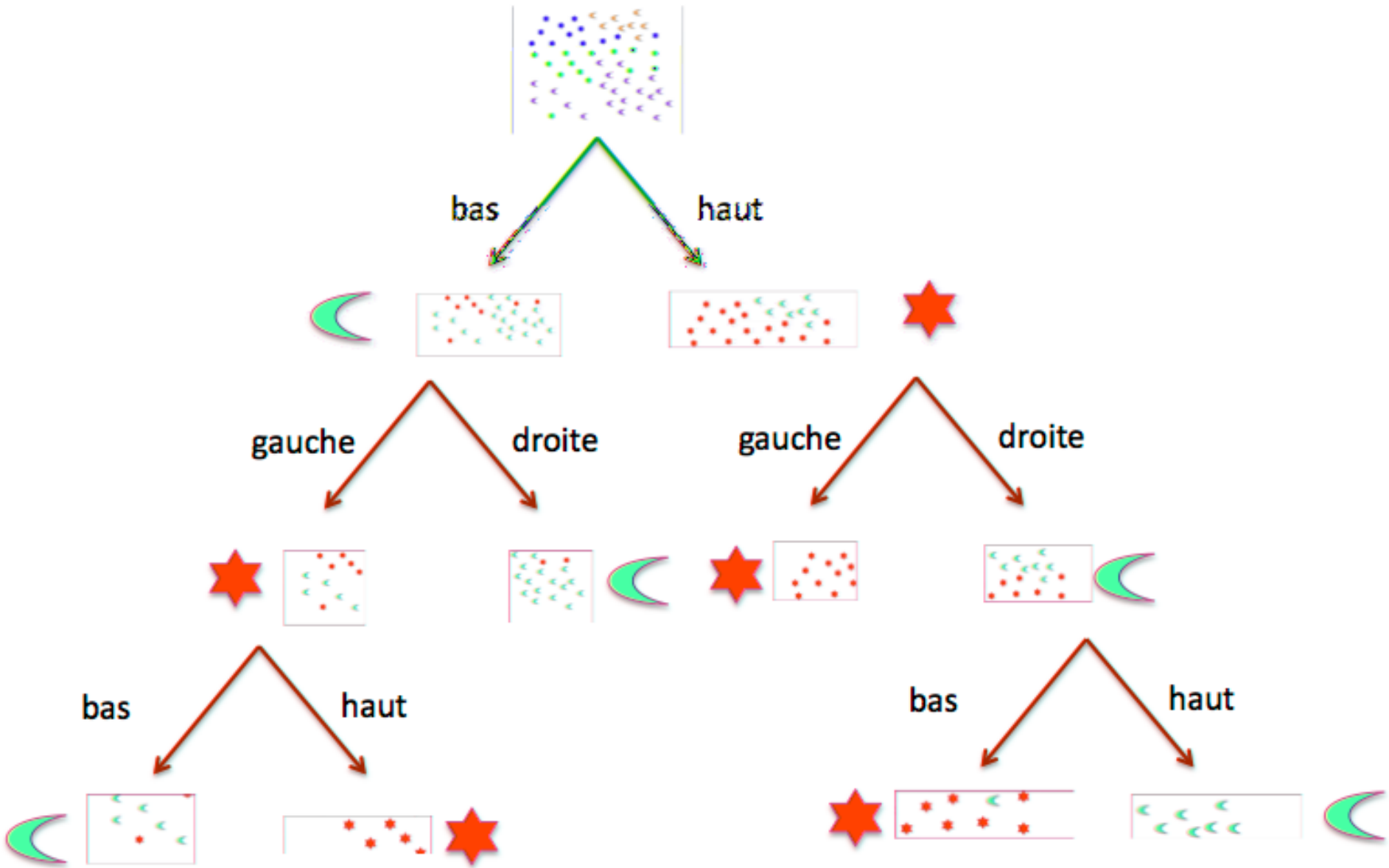
Variables continues : phase d’apprentissage



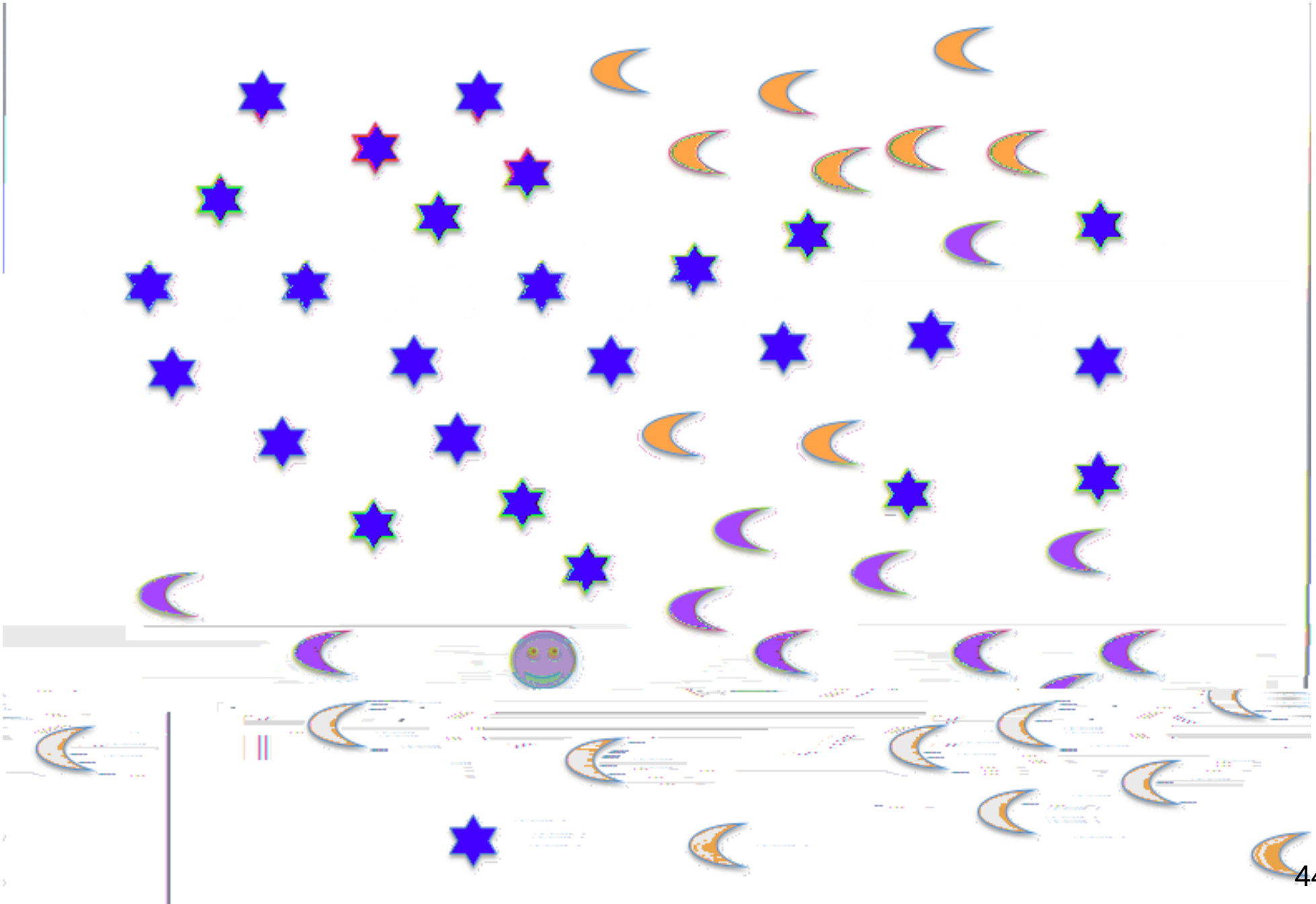
Variables continues : phase d'apprentissage



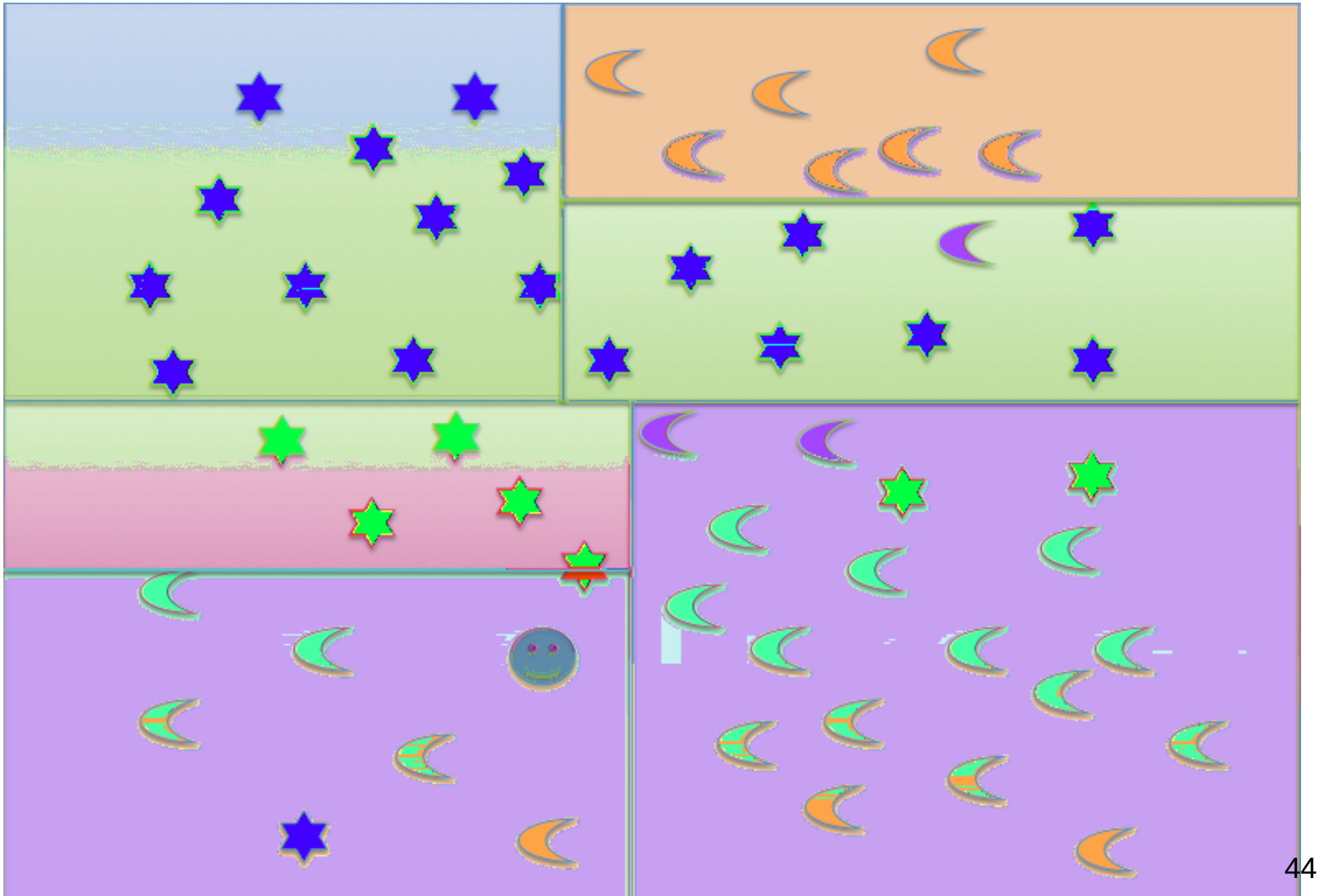
Variables continues : phase d'apprentissage



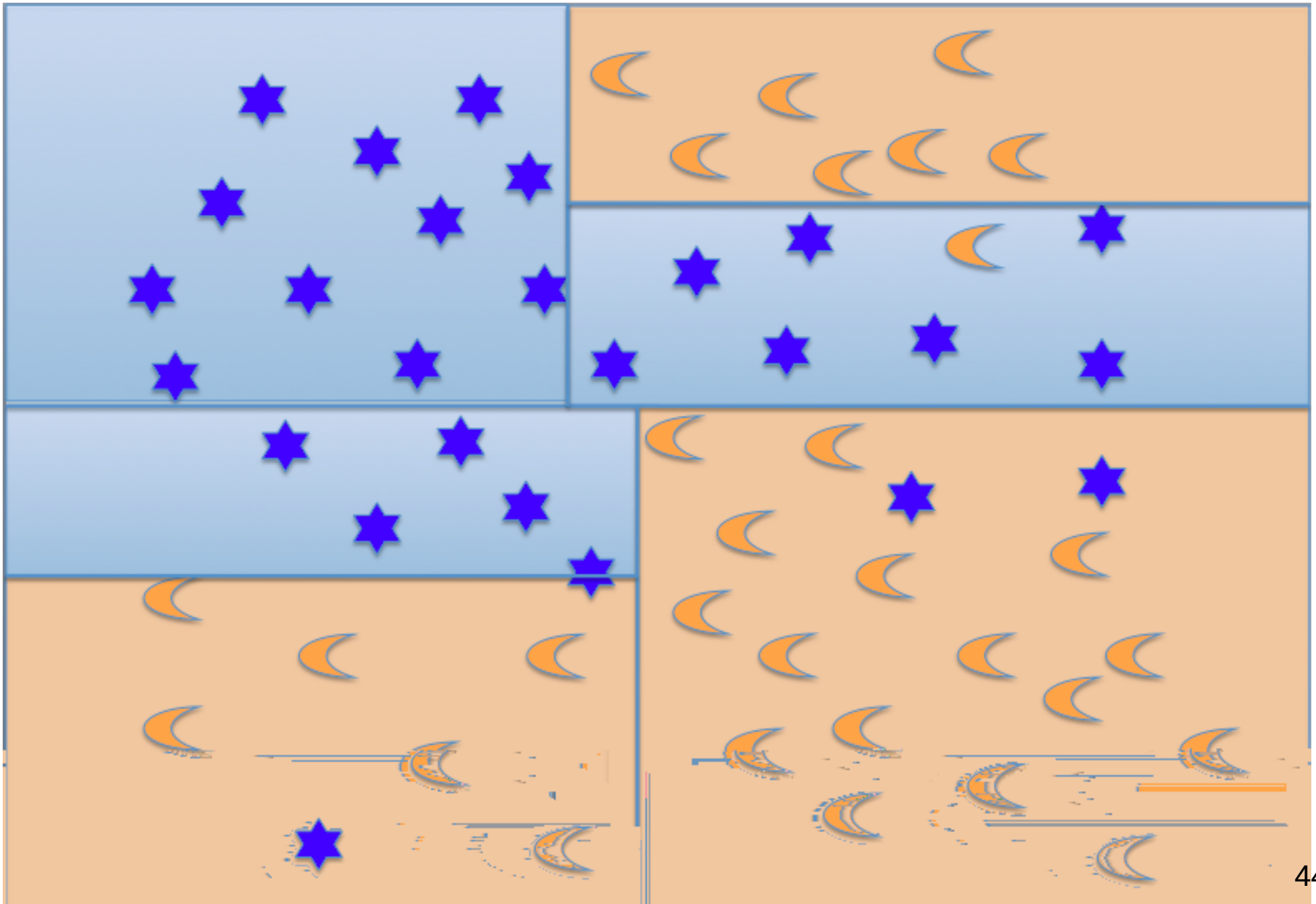
Variables continues : phase de test



Variables continues : phase de test



Variables continues : phase de test



Vocabulaire

- **Noeud** : endroit de coupure
- **Racine** : noeud initial, aucune coupure n'a été faite.
- **Feuille** : extrémité de l'arbre (noeud qui n'est pas divisé).
- **Profondeur** : nombre de niveaux de l'arbre
- **Taille minimale des feuilles** : nombre minimal de points de l'ensemble d'apprentissage toléré dans une feuille.
- **Critère de séparation** : critère selon lequel on choisit la variable/la coupure.

Algorithme

Algorithm 3 Arbre de décision

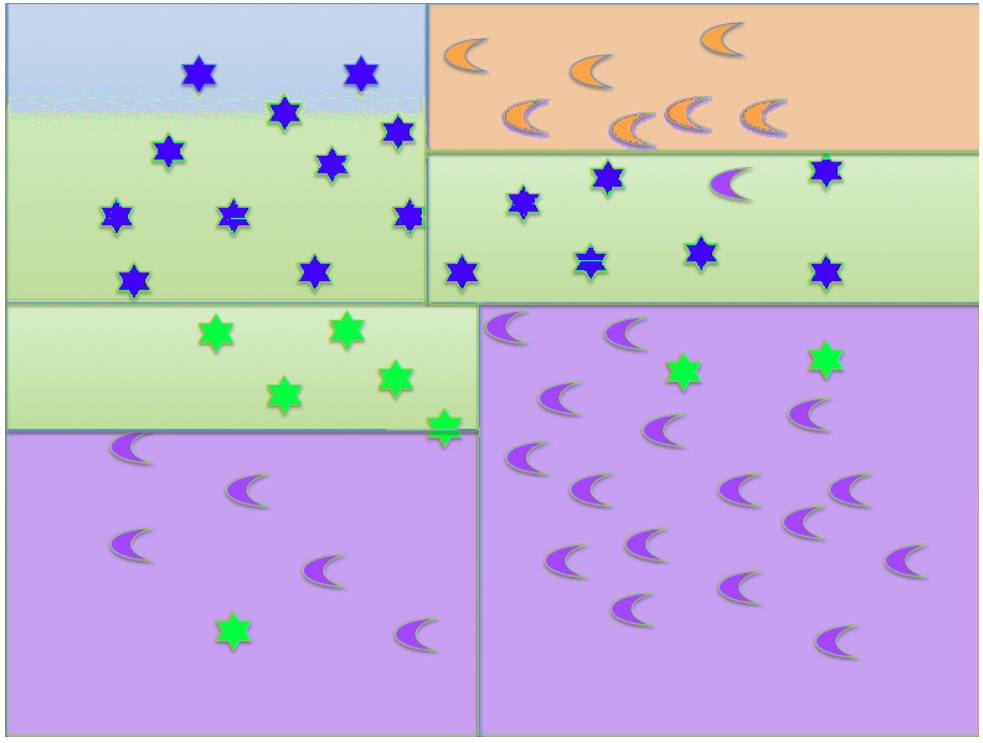
Require: Ensemble d'apprentissage $(x_1, y_1), (x_2, y_2)..., (x_n, y_n)$.

1: **Initialisation** : Arbre = Arbre vide ; N = racine.
2: **while** N existe **do**
3: Tester si le noeud N est terminal
4: **if** Le noeud N est terminal **then**
5: N est une feuille. Donner une classe à N.
6: **else**
7: Tester N et créer des noeuds fils
8: **end if**
9: N = noeud non encore exploré s'il existe.
10: **end while**

Tester si un noeud est terminal

Deux critères :

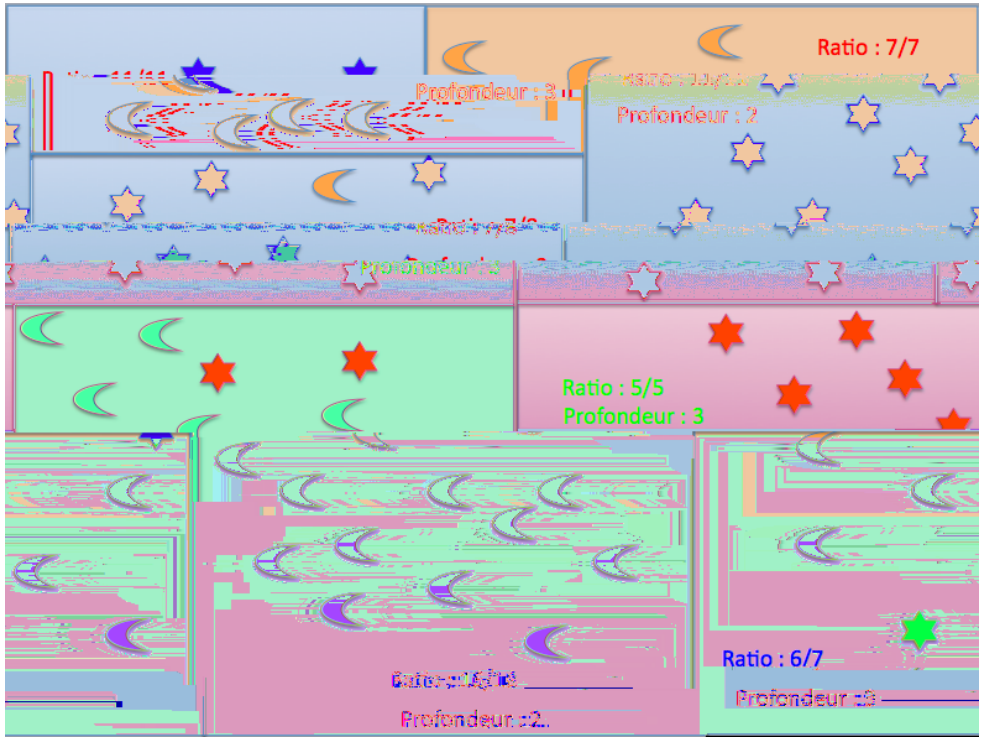
- **Profondeur de l'arbre** : si l'on a atteint la profondeur maximale, on arrête.
- **Ratio des classes dans le noeud** : si le ratio classe majoritaire / total des points du noeud est supérieur à une valeur, on arrête.



Tester si un noeud est terminal

Deux critères :

- **Profondeur de l'arbre** : si l'on a atteint la profondeur maximale, on arrête.
- **Ratio des classes dans le noeud** : si le ratio classe majoritaire / total des points du noeud est supérieur à une valeur, on arrête.



Comment choisir la variable selon laquelle on segmente ?

- **Objectif:** Choisir la variable qui maximise le gain en **pureté** ou en information.
- On n'a le droit de segmenter que sur **une variable à la fois** (un axe).
- Cas discret : la segmentation se fait sur toutes les valeurs possibles de la variable.
- Cas continu : la segmentation est forcément simple : de la forme "variable > valeur, variable <= valeur".

2 critères couramment utilisés :

- l'indice de Gini
- l'entropie de Shannon

L'approche est gloutonne : on teste chaque variable et chaque coupure possible et on choisit la meilleure.

Indice de Gini

Définition : L'indice de Gini (ou coefficient d'**impureté** de Gini) mesure combien de fois un exemple choisi au hasard dans un ensemble serait mal classifié si il était classifié selon la distribution dans l'ensemble.

Traduction : Combien me coûterait-il de choisir cette coupure à tort?

Indice de Gini

Définition : L'indice de Gini (ou coefficient d'**impureté** de Gini) mesure combien de fois un exemple choisi au hasard dans un ensemble serait mal classifié si il était classifié selon la distribution dans l'ensemble.

Traduction : Combien me coûterait-il de choisir cette coupure à tort?

Exemple : soit un ensemble avec 12 exemples de la classe 1 et 8 exemples de la classe 2. La distribution est donc $P_1 = 12/20 = 3/5$ et $P_2 = 8/20 = 2/5$. Si l'on prend un exemple au hasard, on lui attribuerait $3/5$ du temps le label 1 et $2/5$ du temps le label 2. En moyenne, cet exemple serait mal classé $\frac{2}{5} \times \frac{3}{5} + \frac{3}{5} \times \frac{2}{5} = 12/25$ du temps (calcul d'espérance).

Indice de Gini

Définition : L'indice de Gini (ou coefficient d'**impureté** de Gini) mesure combien de fois un exemple choisi au hasard dans un ensemble serait mal classifié si il était classifié selon la distribution dans l'ensemble.

Traduction : Combien me coûterait-il de choisir cette coupure à tort?

Exemple : soit un ensemble avec 12 exemples de la classe 1 et 8 exemples de la classe 2. La distribution est donc $P_1 = 12/20 = 3/5$ et $P_2 = 8/20 = 2/5$. Si l'on prend un exemple au hasard, on lui attribuerait $3/5$ du temps le label 1 et $2/5$ du temps le label 2. En moyenne, cet exemple serait mal classé $\frac{2}{5} \times \frac{3}{5} + \frac{3}{5} \times \frac{2}{5} = 12/25$ du temps (calcul d'espérance).

On calcule donc l'indice de Gini pour chaque coupure possible et on choisit la variable/ la coupure pour laquelle cet indice est le plus petit.

Tests de coupures : cas discret

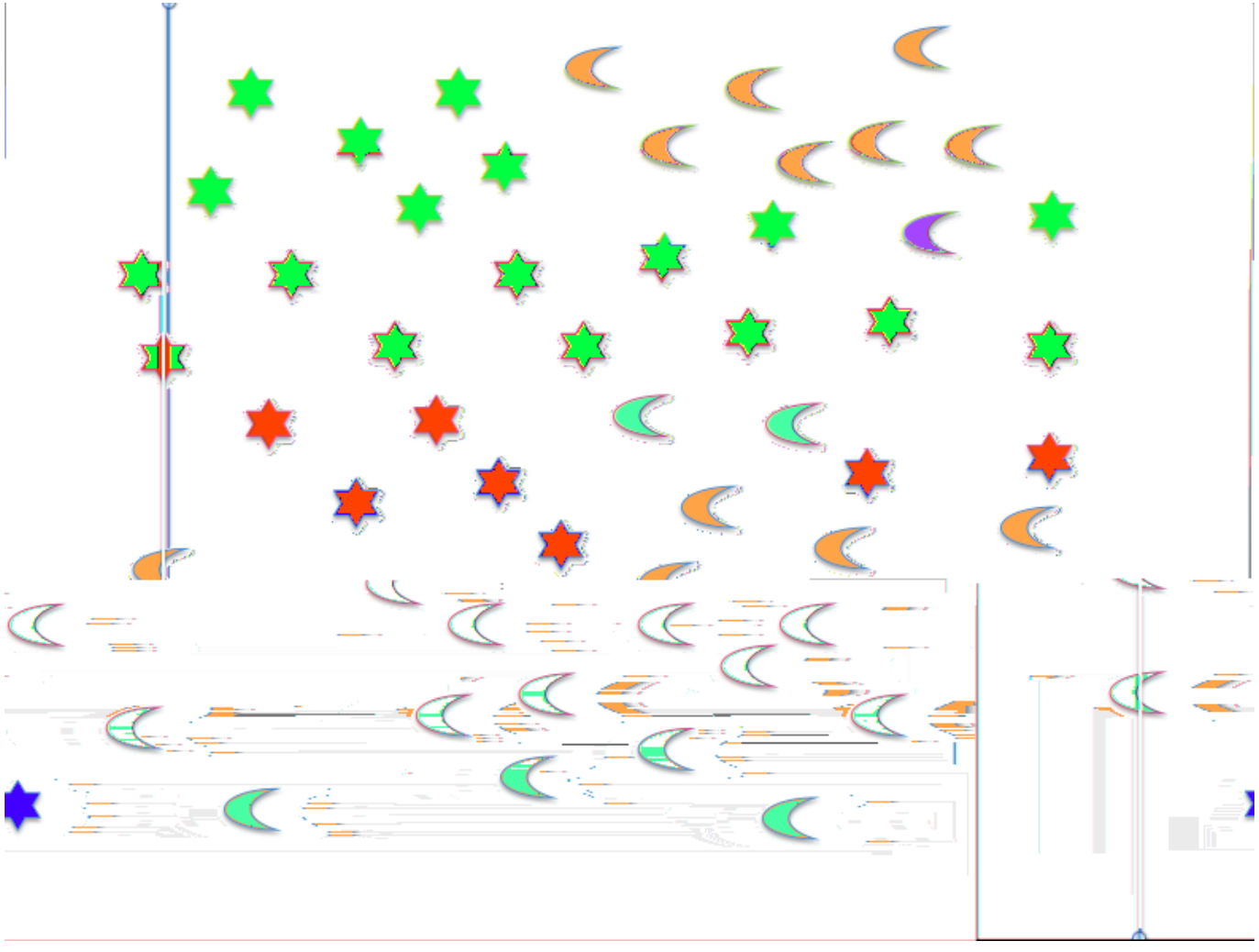
On calcule l'indice pour chaque variable:

Match à domicile ?	Balance positive ?	Mauvaises conditions climatiques ?	Match précédent gagné ?	Match gagné
V	V	F	F	V
F	F	V	V	V
V	V	V	F	V
V	V	F	V	V
F	V	V	V	F
F	F	V	F	F
V	F	F	V	F
V	F	V	F	F

Exemple : couper selon "match à domicile":

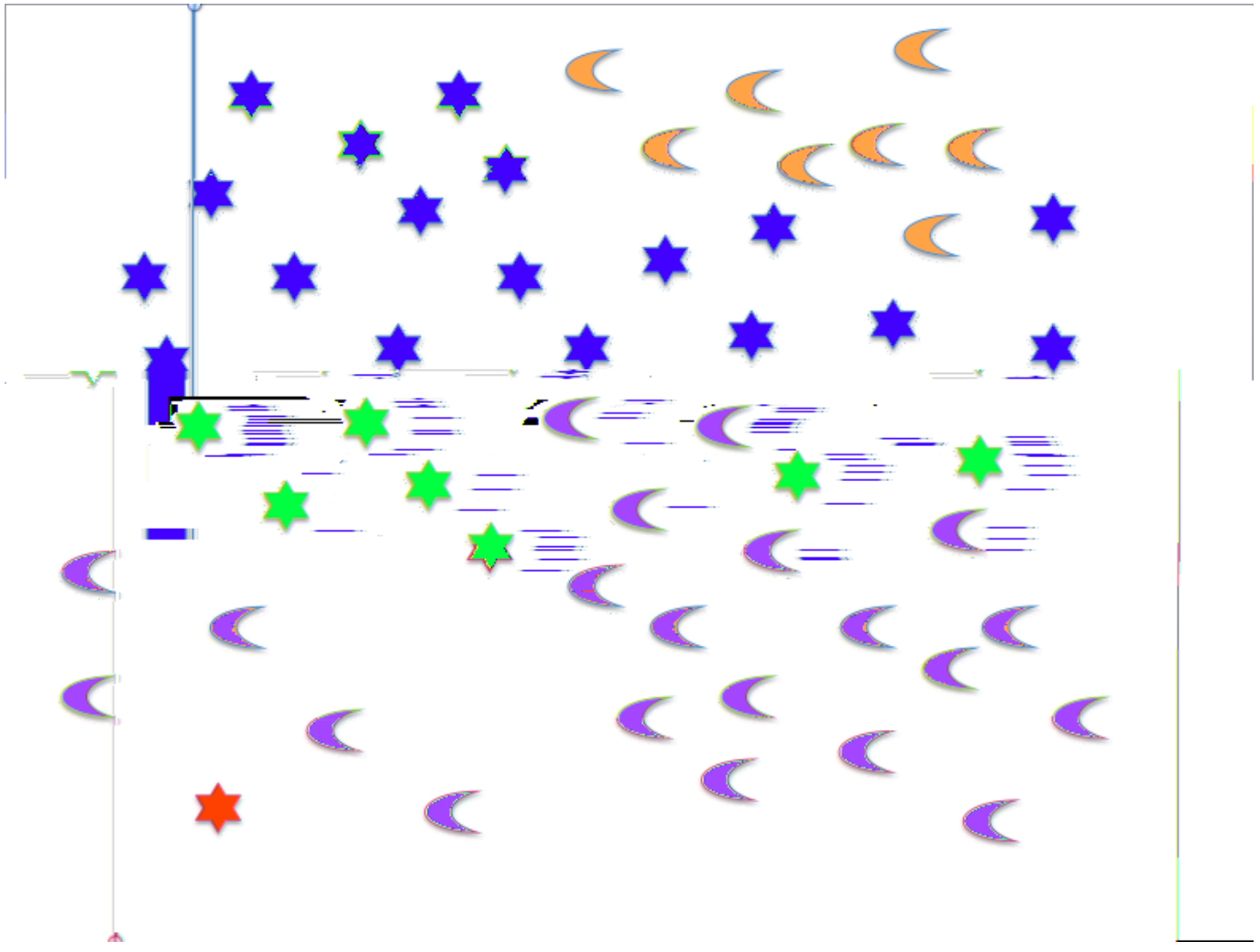
- 5V, 3F
- V => 3 gagné, 2 perdus
- F => 1 gagné, 2 perdus
- Indice de Gini : $5/8 \times 3/5 \times 2/5 + 3/8 \times 1/3 \times 2/3 = 0,46$

Tests de coupures : cas continu



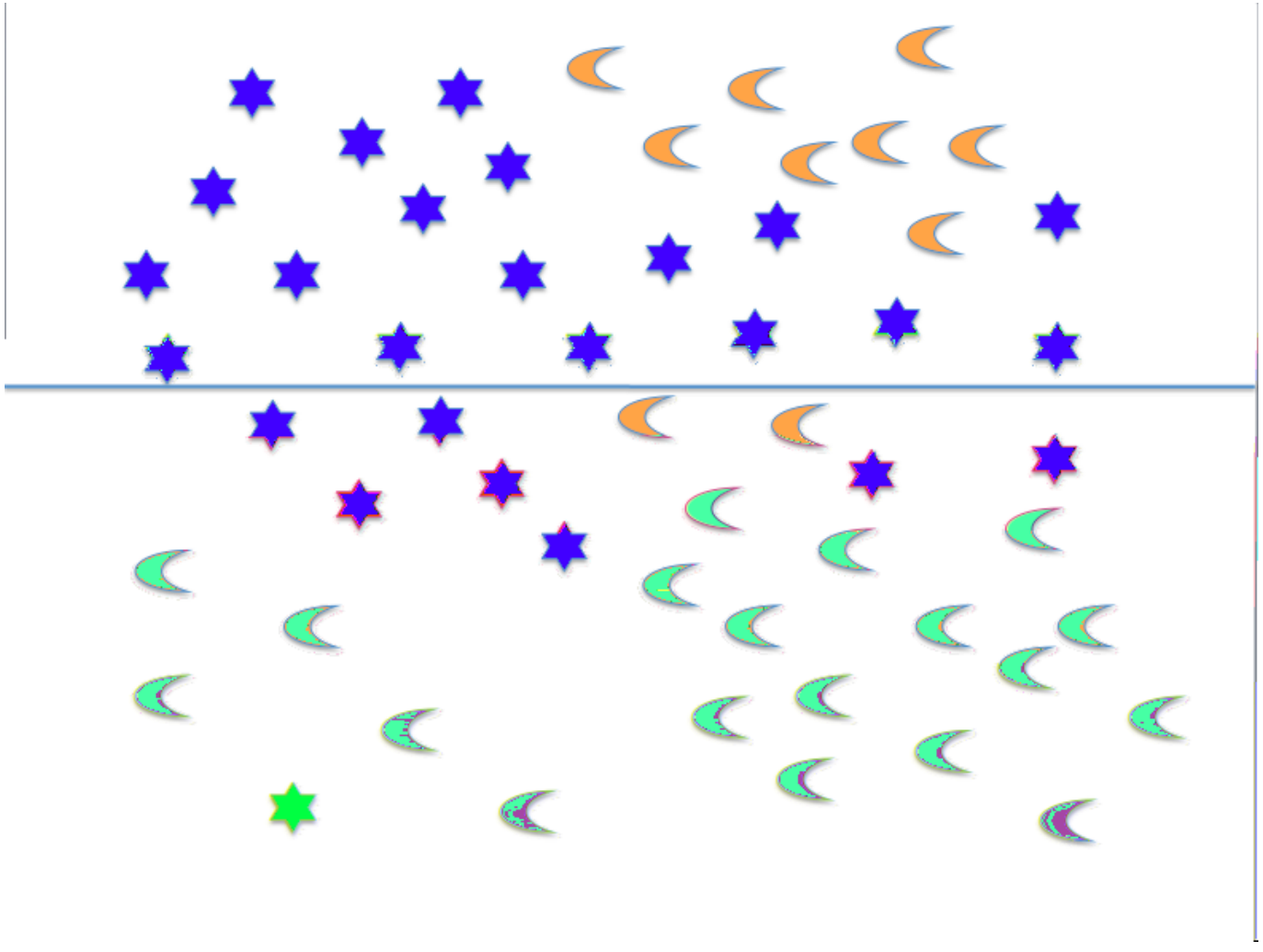
Gini : $1/55 \times 1/26 \times 0/29 + 54/55 \times 25/26 \times 29/29 = 0,94$

Tests de coupures : cas continu



Gini : $4/55 \times 2/26 \times 2/29 + 51/55 \times 24/26 \times 27/29 = 0,79$

Tests de coupures : cas continu



Gini : $36/55 \times 18/26 \times 8/29 + 29/55 \times 8/26 \times 21/29 = 0,24$

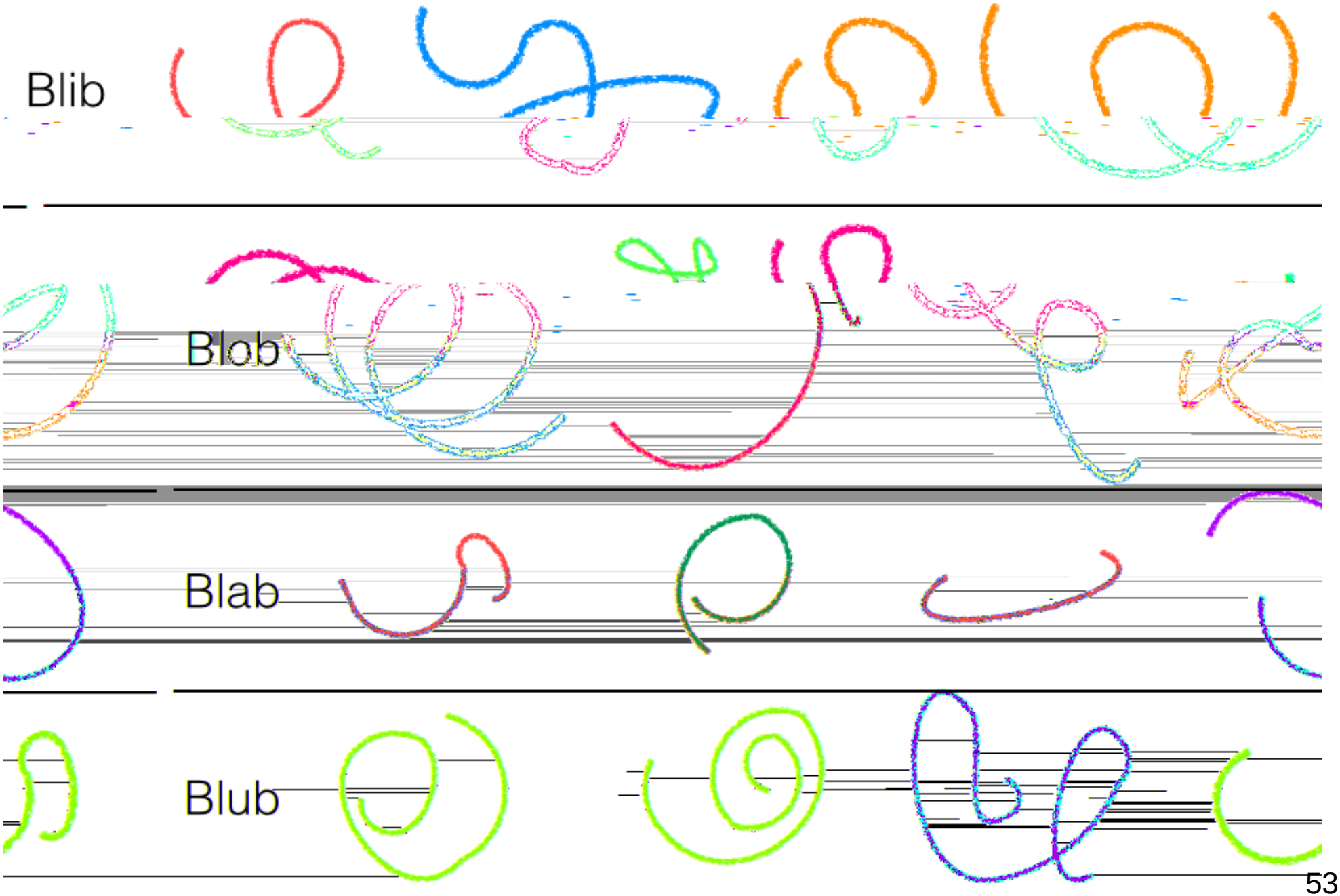
Entropie de Shannon

L’entropie de Shannon est un autre critère selon lequel on peut couper /choisir la bonne variable.

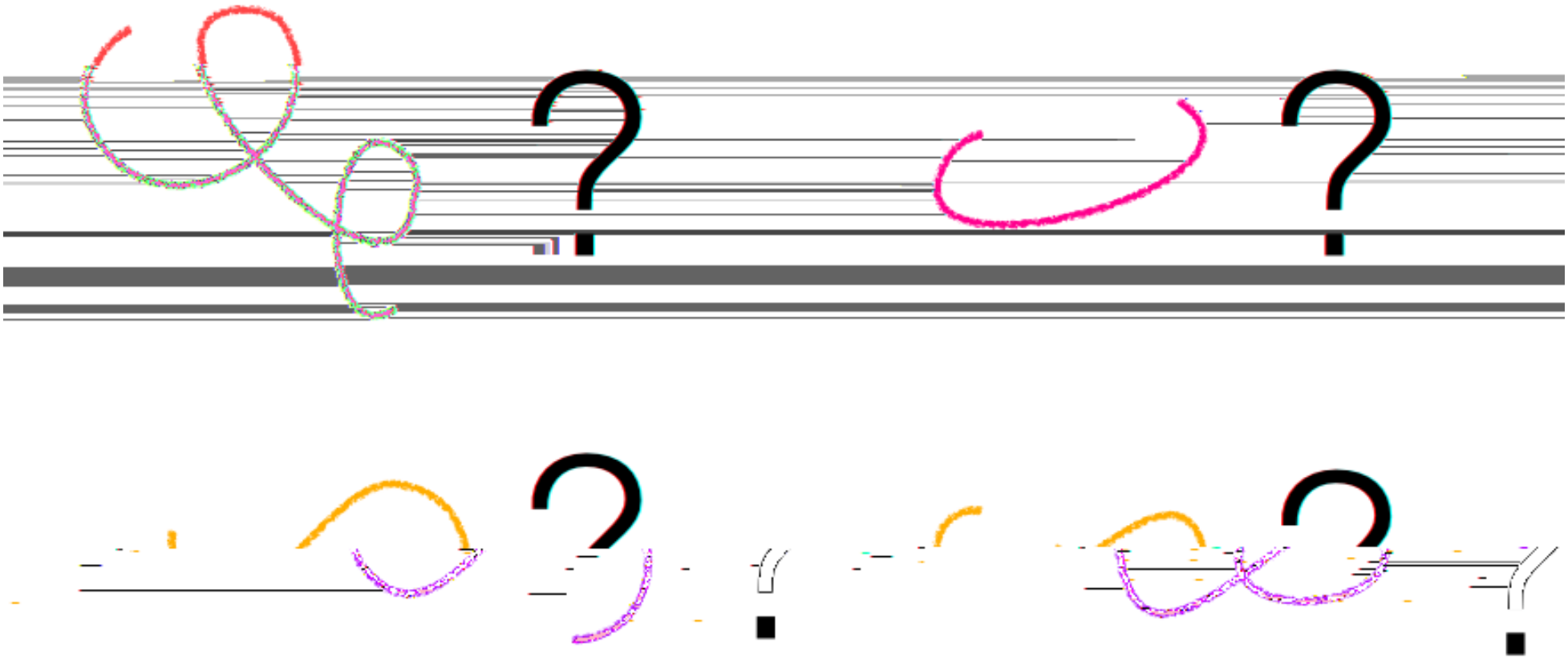
Principe : plus les données sont dispersées, plus l’entropie est grande. On cherche à la réduire.

Minimisation du nombre de formules par cours : nous ne détaillerons pas cette notion...

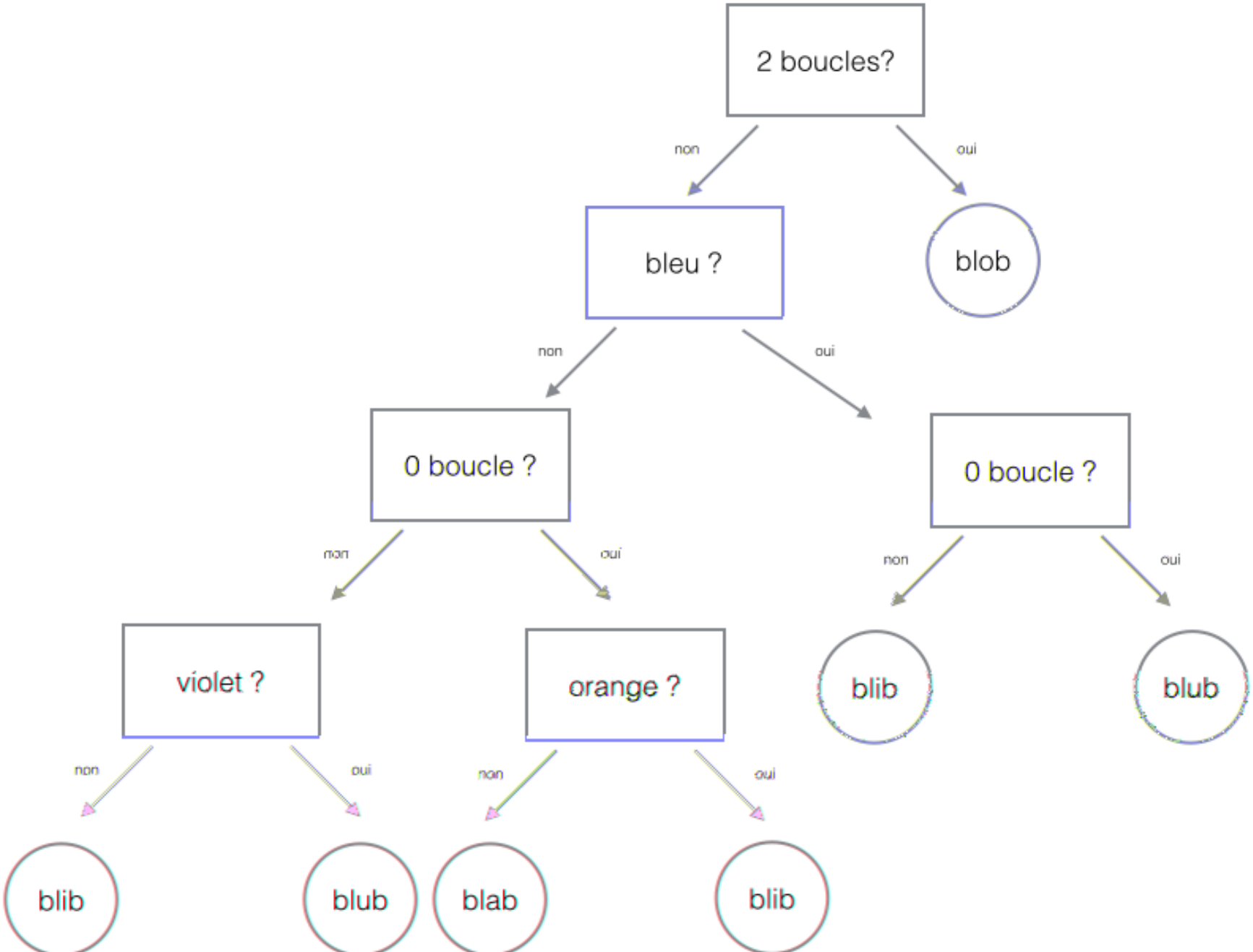
Blibs, blobs, blabs et blubs



Blibs, blobs, blabs et blubs



Blibs, blobs, blabs et blubs



Conclusion sur les arbres de décision

Avantages:

- Très interprétables !
- Faciles à implémenter

Inconvénients:

- Forte élasticité aux exemples : si l'on change un exemple, l'arbre peut changer complètement. On risque donc de faire du **sur-apprentissage**.
- Peuvent nécessiter beaucoup de calculs à cause de l'approche gloutonne.

En python :

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = "gini",
max_depth = 4, min_samples_split = 3)
clf.fit(X,y)
```

Outline

Introduction

- Algorithme datant de 1999 (Breiman, "Random Forests")
- Idée : au lieu de faire un arbre, on fait une forêt.
- Algorithme très puissant, qui règle les problèmes liés à un arbre seul.
- Un des meilleurs algorithmes existants.
- Basé sur la **randomisation**

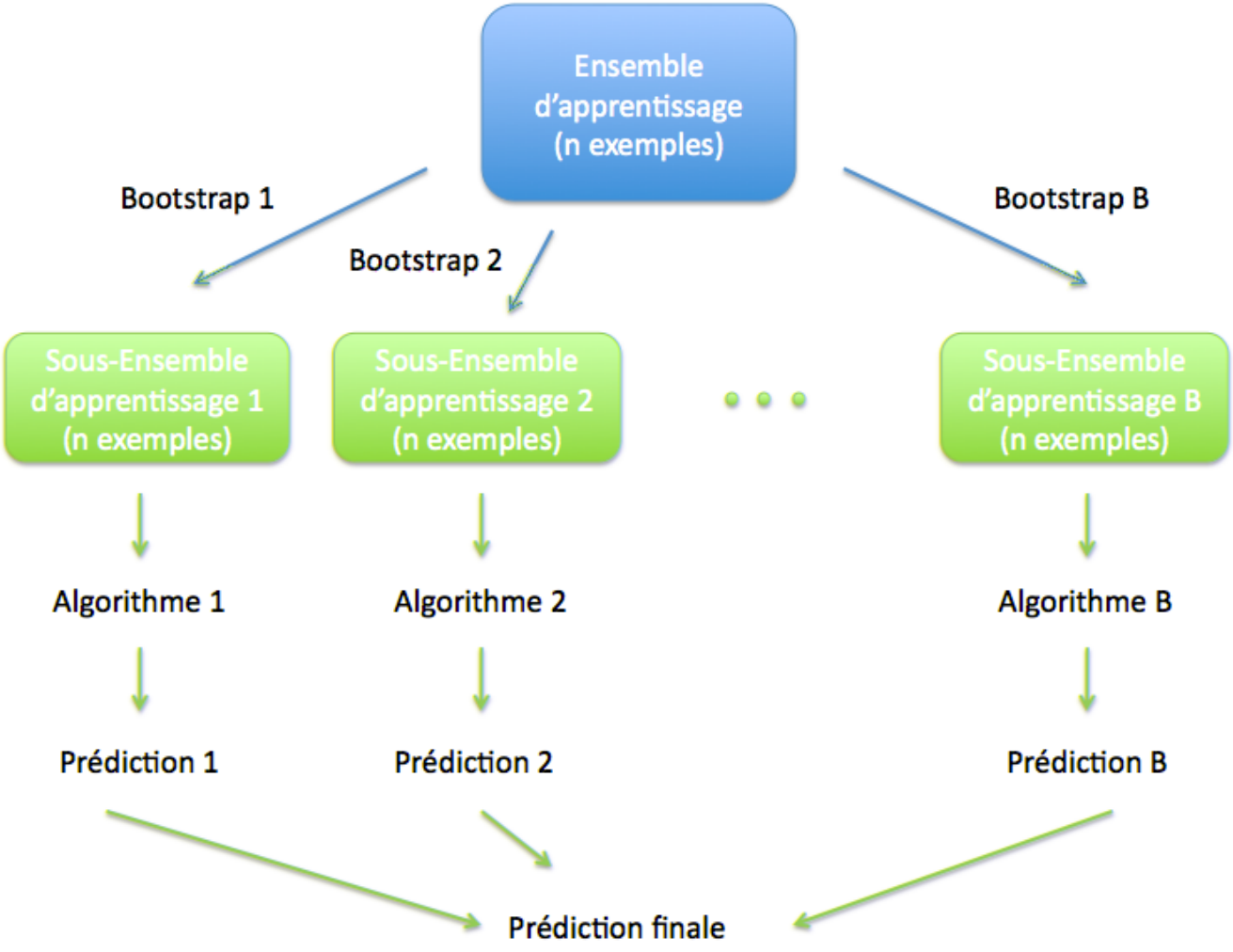
Le bootstrap

Idée : on transforme un ensemble d'apprentissage en B ensembles d'apprentissage.

- On part avec n exemples.
- B fois de suite, on tire au hasard **avec remise** n exemples parmi les n.
- Exemple: n = 10. B = 5.
  1. 3, 6, 7, 4, 1, 8, 3, 3, 2, 9
  2. 2, 8, 5, 0, 0, 8, 6, 1, 0, 4
  3. 5, 9, 4, 9, 7, 8, 2, 9, 9, 1
  4. 5, 8, 5, 3, 1, 5, 1, 5, 4, 6
  5. 3, 3, 5, 0, 1, 6, 3, 2, 0, 4
- De cette manière, on décuple la variabilité de l'ensemble d'apprentissage et on évite le **sur-apprentissage**

Algorithmes d'ensemble

Pour booster la puissance d'un algorithme, on peut le lancer sur *B* ensembles d'apprentissage boostrappés et agréger les résultats :



Algorithme

Algorithm 4 Random Forests

Require: Ensemble d'apprentissage $(x_1, y_1), (x_2, y_2)..., (x_n, y_n)$, nombre d'arbres B, autres paramètres pour les arbres.

1: **for** i = 1 to B **do**
2: Faire un arbre T_i sur le sous-ensemble d'apprentissage i. Le stocker.
3: **end for**

Quelques précisions

- Afin de limiter le temps de calcul, on introduit un paramètre supplémentaire : à chaque noeud, on choisit au hasard un nombre de variables à regarder. Exemple : si on fait du texte, on a 1M de valeurs TF-IDF. A chaque noeud, on en choisit un petit nombre (quelques centaines) au hasard et on calcule le critère sur celles-ci seulement.
- Plus B est grand plus l'algorithme est efficace (et plus il est long).
- Les autres paramètres sont les mêmes que pour les arbres seuls.

Aggrégation

Dans le cas de la classification, on fera un **vote majoritaire** sur tous les arbres :

- Le premier arbre prédit "perdu"
- Le second arbre prédit "perdu"
- Le trosième arbre prédit "gagné"
- => Prédiction finale : "gagné"

Dans le cas de la régression (pas vue dans ce cours), on fera une **moyenne** sur tous les arbres :

- Le premier arbre prédit $1,1$
- Le second arbre prédit $2,0$
- Le trosième arbre prédit $1,9$
- => Prédiction finale : $1,7$

Conclusion sur les random forests

Avantages:

- Très puissant car basé sur **la sagesse des foules** (*B* avis valent mieux qu'un)
- La vitesse de calcul peut être réglée en fonction des paramètres.
- Parallélisable.
- Empiriquement un des meilleurs algorithmes de prédiction.

Inconvénients:

- Difficile à interpréter.
- Peut être long si *B* est grand.

En python

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 1000,
criterion = "gini", max_features = "sqrt")
model.fit(X,y)
```

Conclusion sur les méthodes algorithmiques

- L'apprentissage par des modèles algorithmiques ne demande **pas d'hypothèse sur la forme des données**.
- Il faut choisir les paramètres méthodiquement : un changement de paramètre peut modifier votre résultat.
- Comprendre **la logique de l'algorithme** permet de mieux l'utiliser et de mieux savoir quoi lui donner en entrée.
- Mieux vaut un algorithme **moins bon sur l'ensemble d'apprentissage mais qui sait généraliser**.