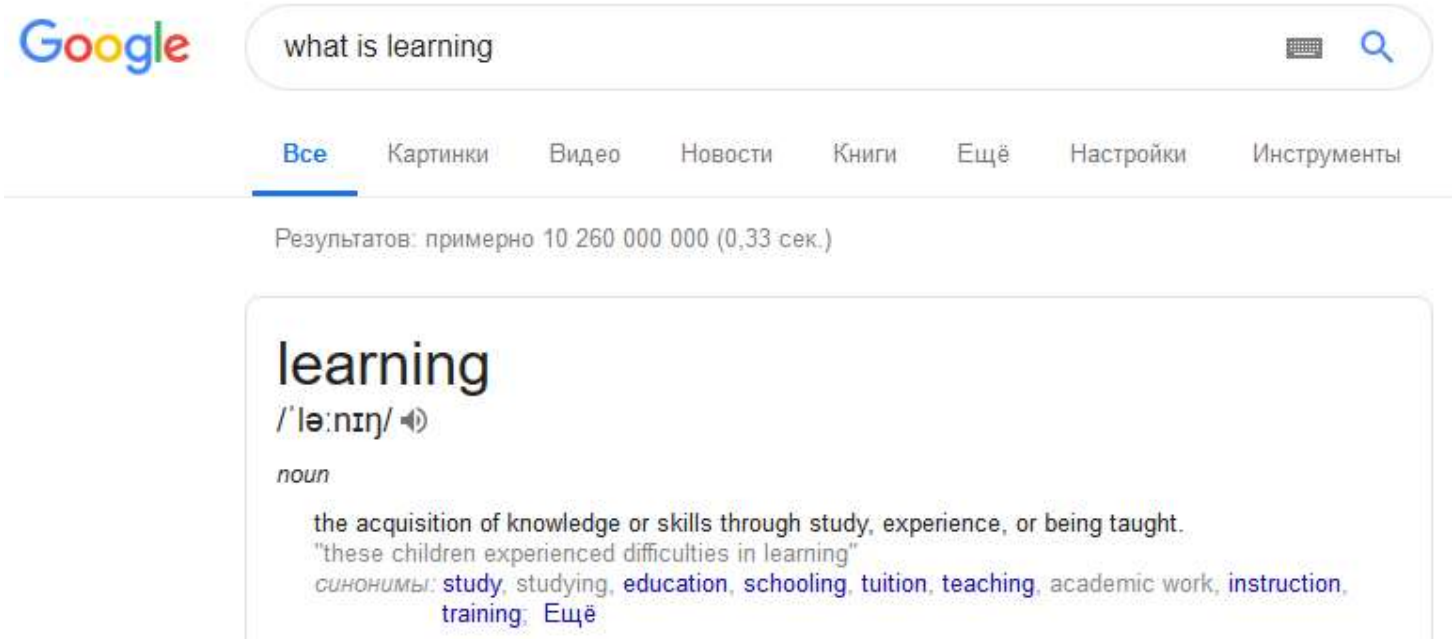
What is learning?



What is learning?

«Machine learning is the science (and art) of programming computers so they can learn from data»

Aurélien Géron

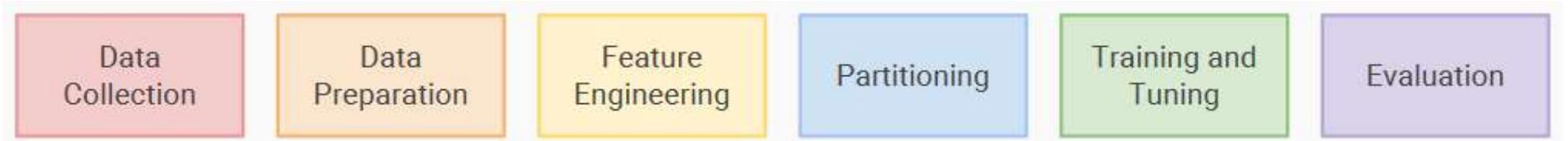
«A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*»

(Tom Mitchell)

«Roughly speaking, learning is the process of converting experience into expertise or knowledge. The input to a learning algorithm is training data, representing experience, and the output is some expertise, which usually takes the form of another computer program that can perform some task»

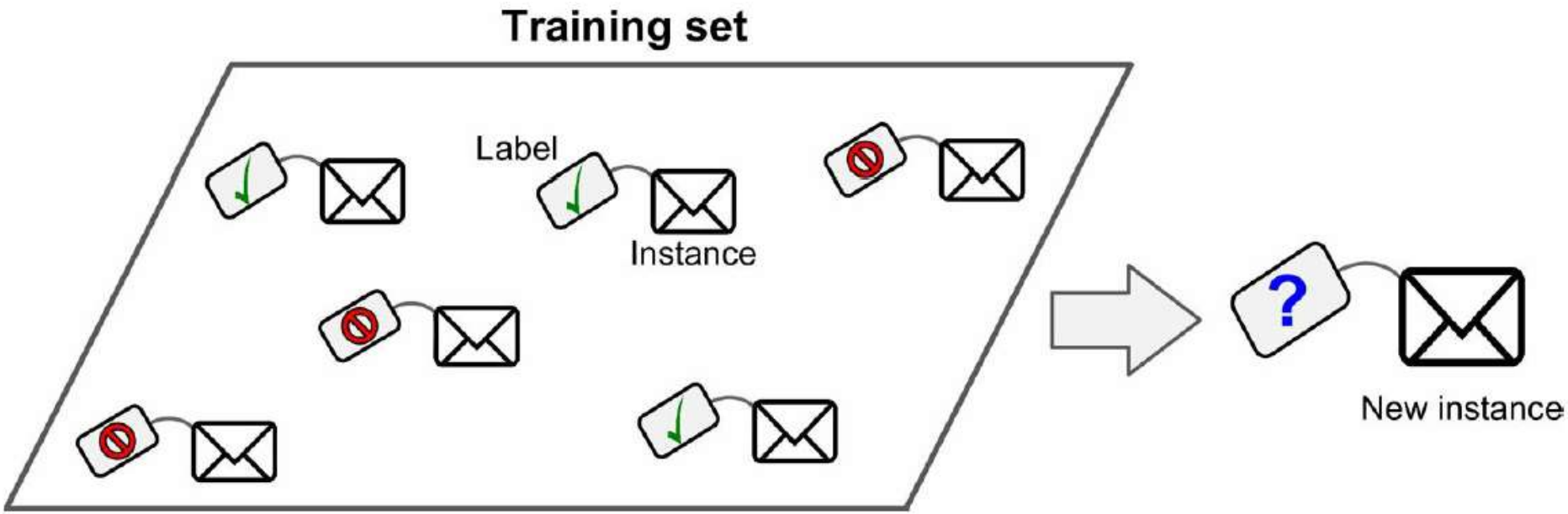
(Shalev-Shwartz and Ben-David)

Machine Learning Workflow



(aiukraine.com)

Supervised Learning – 1



(© O'Reilly)

Supervised Learning – 2. Basic Setting

$\mathcal{X}$ – instance space;
$\mathcal{Y}$ – label set;
$\mathcal{D}$ – joint distribution on $\mathcal{X} \times \mathcal{Y}$;
$S = \big((x_1, y_1),\ \ldots,\ (x_n, y_n)\big)$ – training set (обучающая выборка)

$x_1, \ldots, x_n$ – instances (образцы, экземпляры, примеры, прецеденты);
$y_1, \ldots, y_n$ – labels (метки);
$(x_1, y_1),\ \ldots,\ (x_n, y_n)$ – training examples.

- Classification: $\mathcal{Y}$ is finite, $\mathcal{Y} = \{0, 1, \ldots, n-1\}$
  - $|\mathcal{Y}| = 2$: binary classification, $\mathcal{Y} = \{0,1\}$, $\mathcal{Y} = \{-1, +1\}$
- Regression: $\mathcal{Y}$ is interval, $\mathcal{Y} = (-\infty; +\infty)$, $\mathcal{Y} = (0; +\infty)$, $\mathcal{Y} = (0; 1)$
- Ranking: $\mathcal{Y}$ is the set of permutations or partial orders
- Structured Prediction: $\mathcal{Y}$ is the set of «structured objects»: sequences, trees, graphs etc.

The task: find **predictor** h: $\mathcal{X} \to \mathcal{Y}$ (aka prediction rule, hypothesis, classifier, regressor).
We need some **learning algorithm** A with

- input: training set S;
- output: predictor $h = A(S)$.

Supervised Learning – 3. Loss and Risk

Loss function: how much the predicted label is distinct from the true label?

$$l\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$$

0-1 loss: $l_{0-1}(h(x), y) = \begin{cases} 0, & h(x) = y \\ 1, & h(x) \neq y \end{cases}$

square loss: $l_{\text{sq}}(h(x), y) = (h(x) - y)^2$

absolute loss: $l_{\text{abs}}(h(x), y) = |h(x) - y|$

and many more . . .

True risk: what's the average loss of predictor h?

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}[\, l(h(x), y)\,]$$

Empirical risk:

$$\frac{1}{n}\sum_{i=1}^{n} l(h(x_i), y_i)$$

Hypothesis space: the set $\mathcal{H}$ of all predictors that we consider.

We want our learning algorithm to find a predictor $h \in \mathcal{H}$ that minimizes true risk. But we usually don't know the underlying distribution $\mathcal{D}$. Thus we try to minimize empirical risk.

Supervised Learning – 4. kNN

«Show Me Your Friends And I'll Tell You Who You Are»

Let $d\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ be distance function on $\mathcal{X}$ and $x \in \mathcal{X}$.
Let $(x_1, y_1), \ldots, (x_n, y_n)$ be training set.
Sort x_i's by distance from x: $x_{i_1}, \ldots, x_{i_n}$, $d(x, x_{i_1}) \leq \cdots \leq d(x, x_{i_n})$.

Take first k **nearest neighbors** and return their mode (classification) or mean (regression).

- How to deal with ties?
  - Take all equidistant: what if there are too much of them?
  - Take exactly first k: sensitive to original order of instances.
- Should all k-neighbors have equal weight?
  - Equal weights
  - Weights inversely proportional to distance
  - Other weighting schemes

scikit-learn specific warning:

> **Warning:** Regarding the Nearest Neighbors algorithms, if two neighbors $k+1$ and k have identical distances but different labels, the result will depend on the ordering of the training data.

Supervised Learning – 5. Train-test Split

We don't know the data-generating distribution. How can we estimate the quality of our predictor?

Option 1 (BAD). Use error on train set. It's bad because the predictor was chosen to give a small error so it's either a very bad predictor or we can't be sure that it can *generalize* beyond the seen data.

Option 2. Let's check how well the predictor performs on the data *from the same distribution* that it has never seen. This leads us to the idea of **train-test split**: put some data aside and test performance on this part.

- **underfitting**: large error on train subset
- **overfitting**: error on test subset is much larger than error on train subset
- good model: no underfitting and no overfitting.

Possible issues:

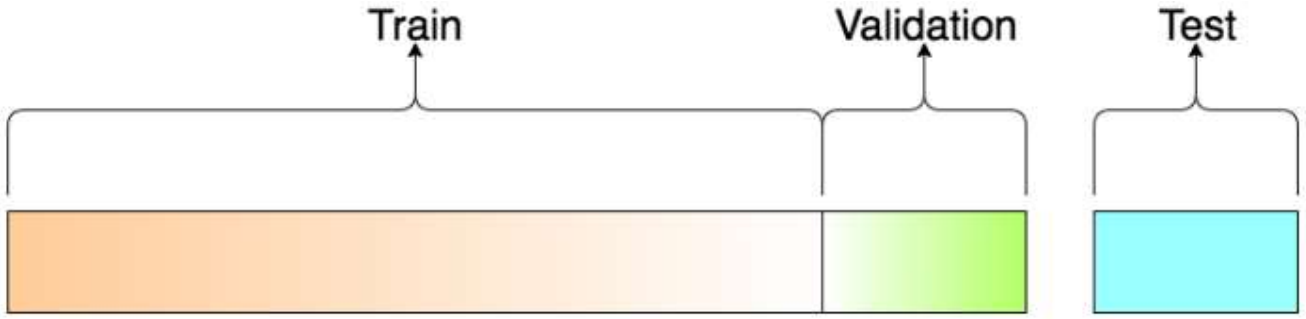
- sensitivity to the way the split was made: what if train subset is much harder than typical subset of real-world data? what if train subset is too small (pessimistic estimate of performance)? what if test set is too small? (highly uncertain estimate of performance) what if some important examples have never been seen by the predictor?
- what if we compare several models? will the model performing best on the test set be a good model for real-world data?
- models can have **hyperparameters** (k in kNN). How to choose the good values? If we choose the values performing best on the test set will they be good values for real-world data?

Supervised Learning – 6. Train-val-test Split

Option 3. Let's choose models and hyperparameters with respect to one subset of the original data and estimate their real-world performance with another subset of the data.

Train-val-test split:

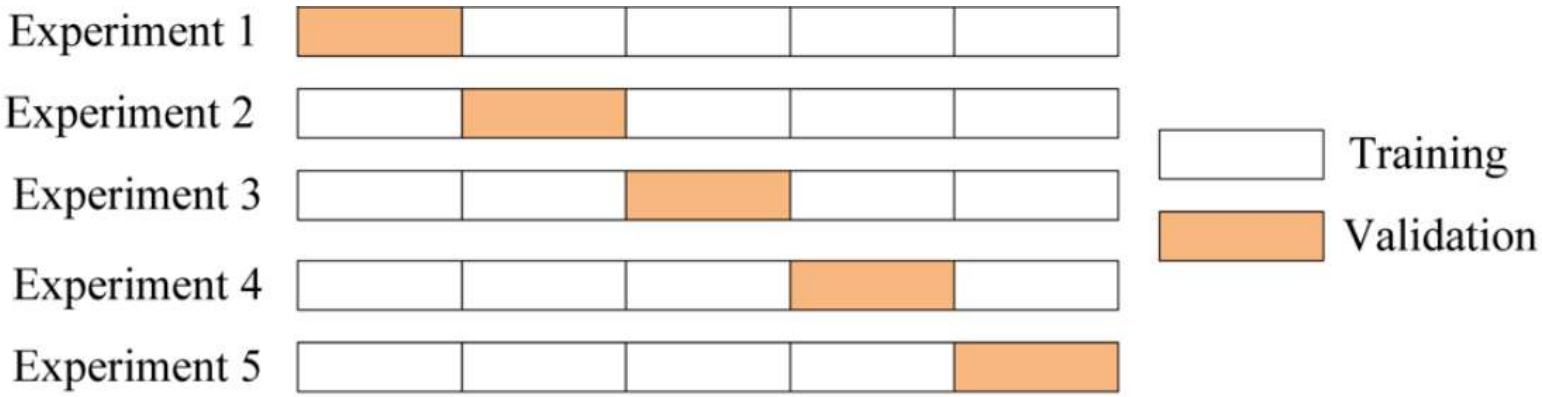
- split original data in three parts: train, validation and test subset;
- choose models/hyperparameters' values that minimize error on validation subset while training model only on train subset;
- then train the model with chosen hyperparameters on train+validation subset;
- then evaluate it on test subset.

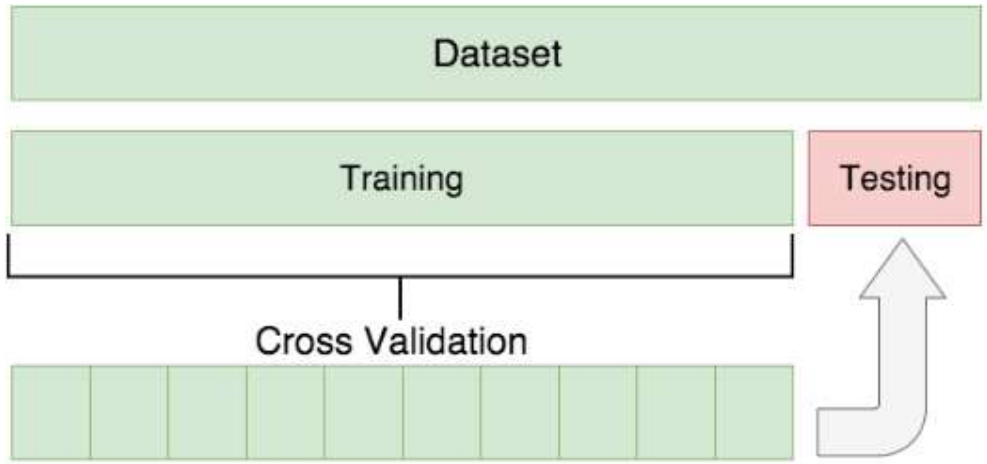


(towardsdatascience.com)

Supervised Learning – 7. Cross-validation

Option 4. One more option for train-val-test split is **cross-validation**:



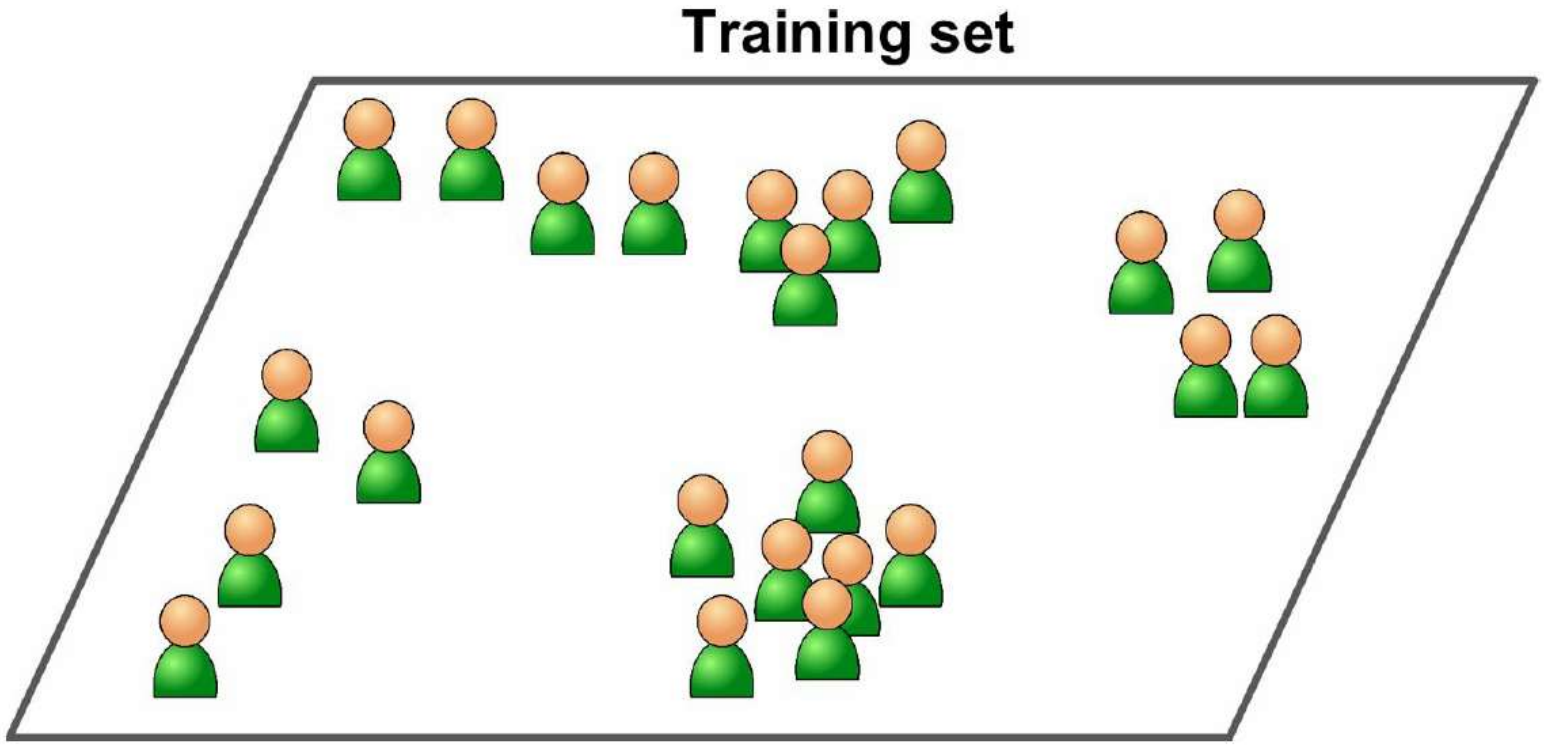


Supervised Learning – 8. Splitting rules of thumb

Rules of thumb:

- you should *always* have a test subset;
- if you tune hyperparameters then you should have a validation subset; the more hyperparameters the larger validation subset must be;
- for «small data» one recommendation is often mentioned: 70-30 for train-test split and 70-15-15 for train-val-test split; but there is no strict rule;
- for «big data» you can take as small as several percent for validation and test subsets;
- k-fold split:
  - too small k: train set is small; possibly pessimistic estimate;
  - too large k: high variation of estimates;
  - can be prohibitively computationally expensive for high k;
  - typical k values for k-fold split: 5 and 10;
  - for «very small data» you can use **leave-one-out** split (synonym for n-fold split): the model will be tested on each object
- **target leakage**: train set should contain only the data available at prediction time in real world (train subset for time series prediction should not contain «far right» values and so on)

Unsupervised Learning – 1



(© O'Reilly)

Unsupervised Learning – 2. Typical Problems

- Clustering
  - split data in groups so that objects in one group are similar and dissimilar from objects in other groups

- Dimensionality Reduction
  - find compact representation of data

- Anomaly Detection
  - detect outliers/novelties

- Association Rules Mining
  - understand co-occurrence of objects

- Density estimation and generative Models
  - estimate the distribution of given data
  - generate new data similar to given data

Supervised learning can sometimes be considered a particular case of unsupervised learning: if we can estimate the distribution of data then we can find perfect predictors:

- for regression: $h^*(x) = \mathbb{E}[Y \mid X = x]$;
- for K-class classification: $h^*(x) = \operatorname{argmax}_{0 \le k \le K-1} \mathbb{P}(X = x \mid Y = k)\mathbb{P}(Y = k)$.

Unsupervised Learning – 3. MDS

MDS (Multidimensional scaling, многомерное шкалирование).

Let $d\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ be some distance function on $\mathcal{X}$.

Let $D = D(x_1, \dots, x_n) \in \mathbb{R}^{\{n \times n\}}$ be **distance matrix** of $x_1, \dots, x_n$ with respect to d

$$d_{ij} = d(x_i, x_j)$$

We fix $q \in \mathbb{N}_+$ and want to find $x'_1, \dots, x'_n \in \mathbb{R}^q$ such that pairwise distances are preserved as good as possible.

Let $D'(x'_1, \dots, x'_n) \in \mathbb{R}^{\{n \times n\}}$ be distance matrix of $x'_1, \dots, x'_n$ with respect to Euclidean distance:

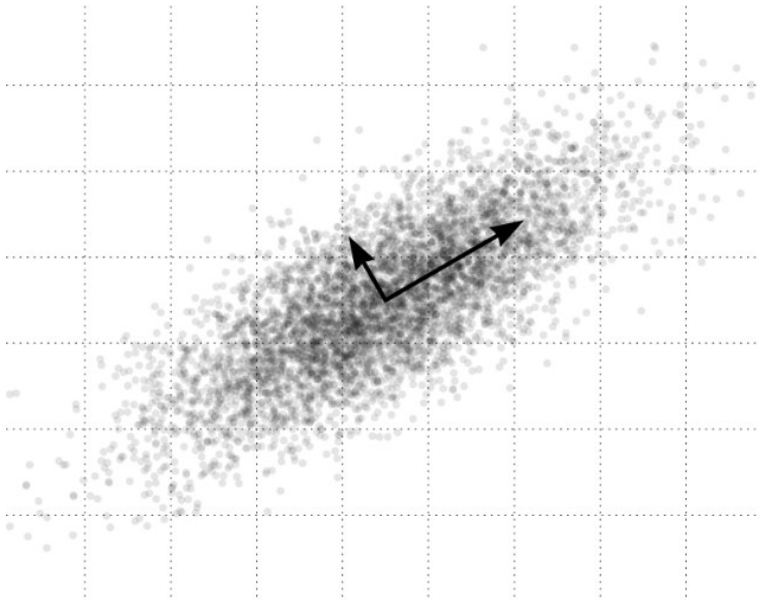
$$d'_{ij} = \|x'_i - x'_j\|$$

We look for $x'_1, \dots, x'_n$ that minimize the following objective function (or some variation of it):

$$\text{Stress}(x'_1, \dots, x'_n) = \sum_{\substack{i,j=1 \\ i<j}}^{n} \left(d_{ij} - d'_{ij}\right)^2 = \sum_{\substack{i,j=1 \\ i<j}}^{n} \left(d(x_i, x_j) - \|x'_i, x'_j\|\right)^2$$

Unsupervised Learning – 4. PCA – A Very Short Intro

Let $x_1, \dots, x_n \in \mathbb{R}^p$. Construct new «nice» coordinate system which preserves variability of data.



(Wikipedia)

Let V_e be a measure of *variability* of data along arbitrary unit vector e.
Let $e_1, \dots, e_p$ be orthonormal basis of $\mathbb{R}^p$ such that $V_{e_1} \leq \cdots \leq V_{e_p}$.
Now choose $q < p$ and project the data onto the space spanned by the first q basis vectors.

Unsupervised Learning – 5. K-Means

Most common case: given data $x_1, \dots, x_n \in \mathbb{R}^p$ find k *centroids* $(k \leq n)$ $c_1, \dots, c_k$ such that

$$\sum_{i=1}^{n} \left(\min_{1 \leq j \leq k} \|x_i - c_j\| \right)^2$$

The l-th cluster is the set of inputs for which c_l is the closest point among $c_1, \dots, c_k$:

$$C_l = \left\{ x_i \;\middle|\; \min_{1 \leq j \leq k} \|x_i - c_j\| = \|x_i - c_l\| \right\}$$

If there are ties then point is assigned to random center.

A very popular algorithm:

1) Select k points as initial centroids
2) Repeat until stopping condition:
   - 2.1) Form k clusters by assigning each point to its closest centroid
   - 2.2) Recompute the centroid of each cluster (as mean of cluster's points)

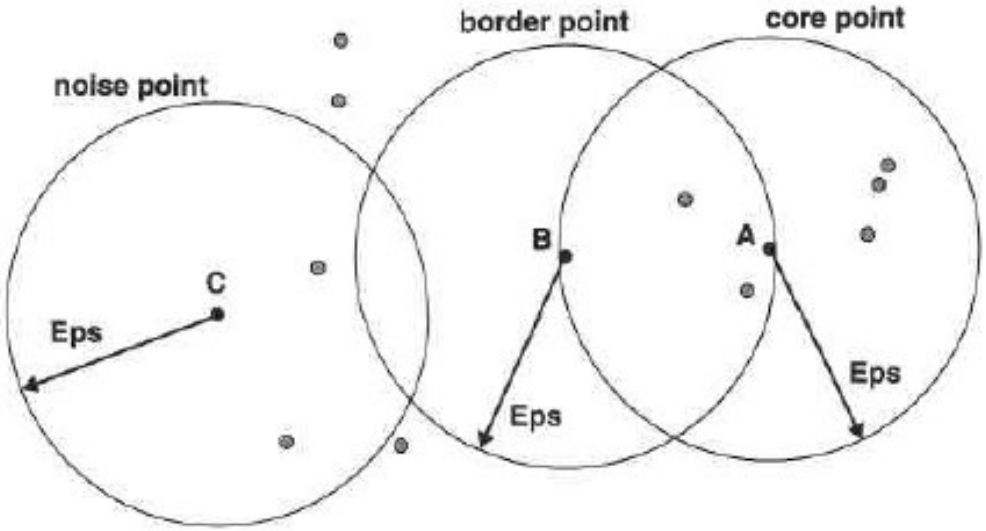
Each iteration of this algorithm does not increase the objective function (called *inertia*).
Possible stopping condition: inertia has changed

Unsupervised Learning – 6. DBSCAN – Intro

Fix two parameters: $m \in \mathbb{N}_{>0}$ and $\varepsilon \in \mathbb{R}_{>0}$. Let $d\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ be distance function. **ε-neighborhood** of $x \in \mathcal{X}$ is defined as $N_\varepsilon(x) = \{x_i \mid d(x, x_i) \leq \varepsilon,\ i = \overline{1,n}\}$.

We define three groups of points:

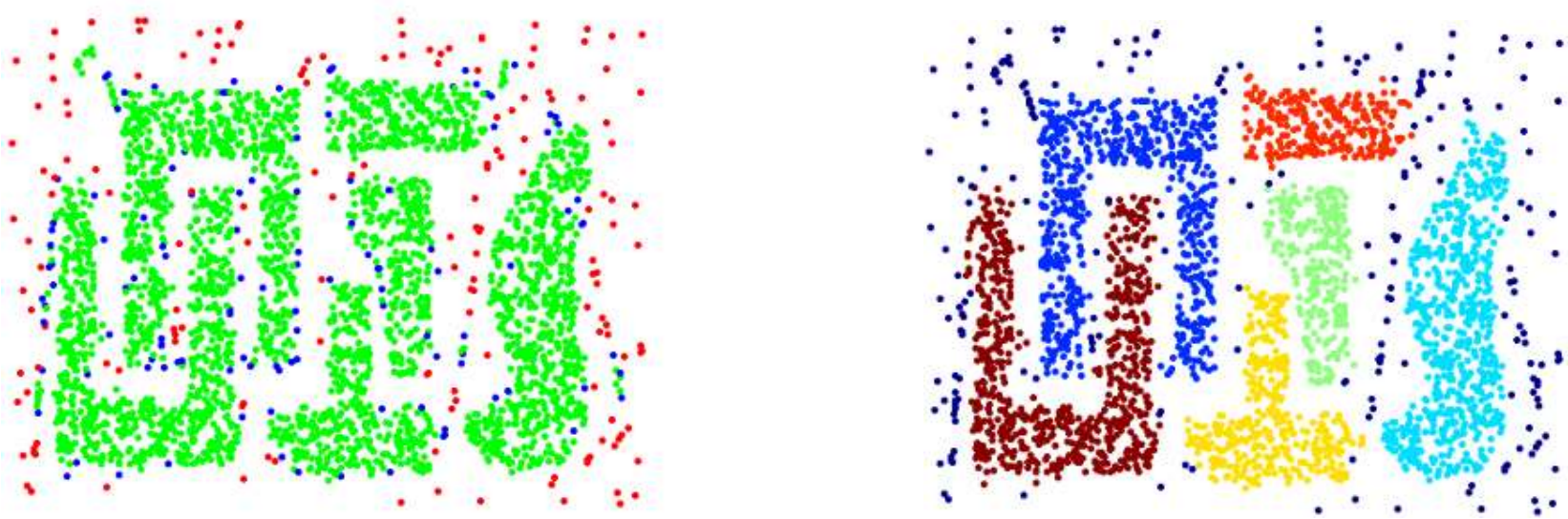
- x_i is a **core** point iff $|N_\varepsilon(x_i)| \geq m$;
- x_i is a **border** point iff it's not a core point but lieas in an ε-neighborhood of some core point; we **associate** x_i with this core point;
- x_i is a **noise** point iff it's neither core point nor border point.



(Tan et al.)

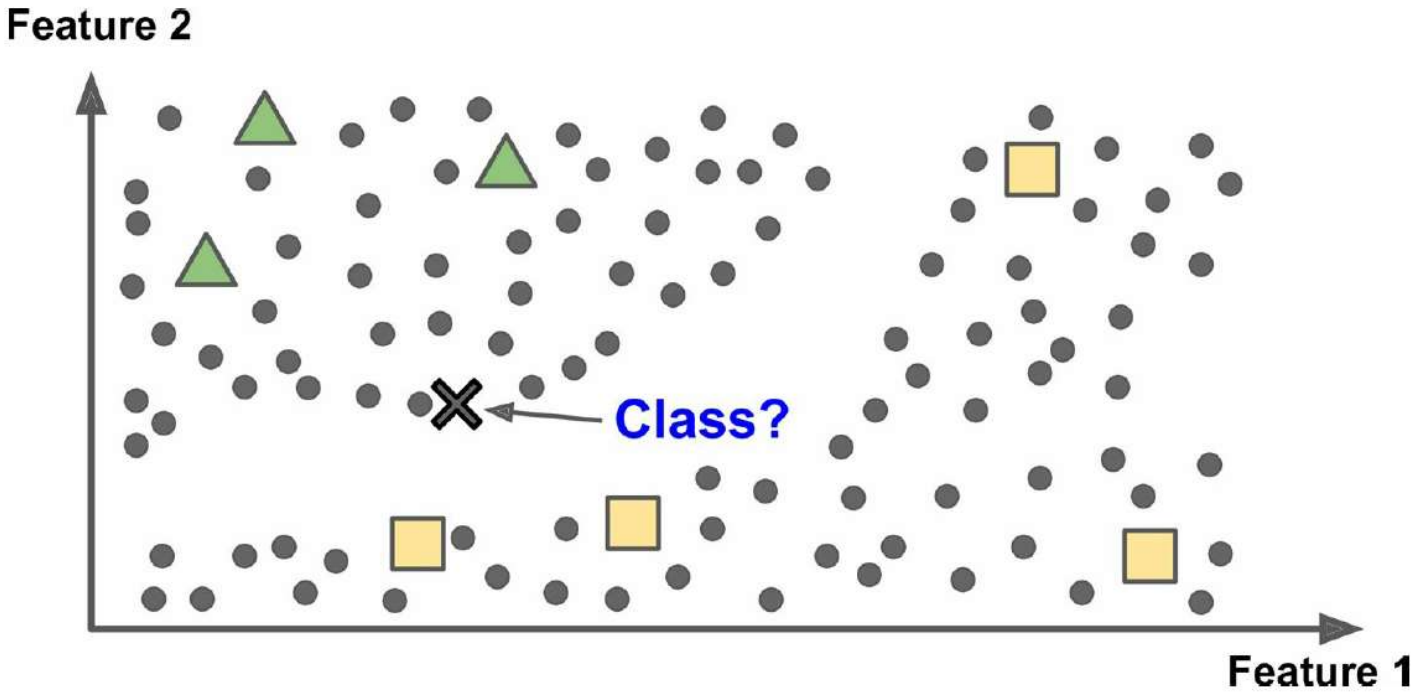
Unsupervised Learning – 7. DBSCAN – Algo

1) Split all points $x_1, \ldots, x_n$ into core, border and noise points.
2) Leave only core and border points.
3) Construct an undirected graph: put an edge $x_i - x_j$ iff x_i and x_j are both core points and $d(x_i, x_j) \leq \varepsilon$.
4) Find connected components in this graph.
5) Assign each border point to the same component with any of its associated core points (if there is more then one option then assign at random).



(cse.buffal.edu)

Semi-supervised Learning

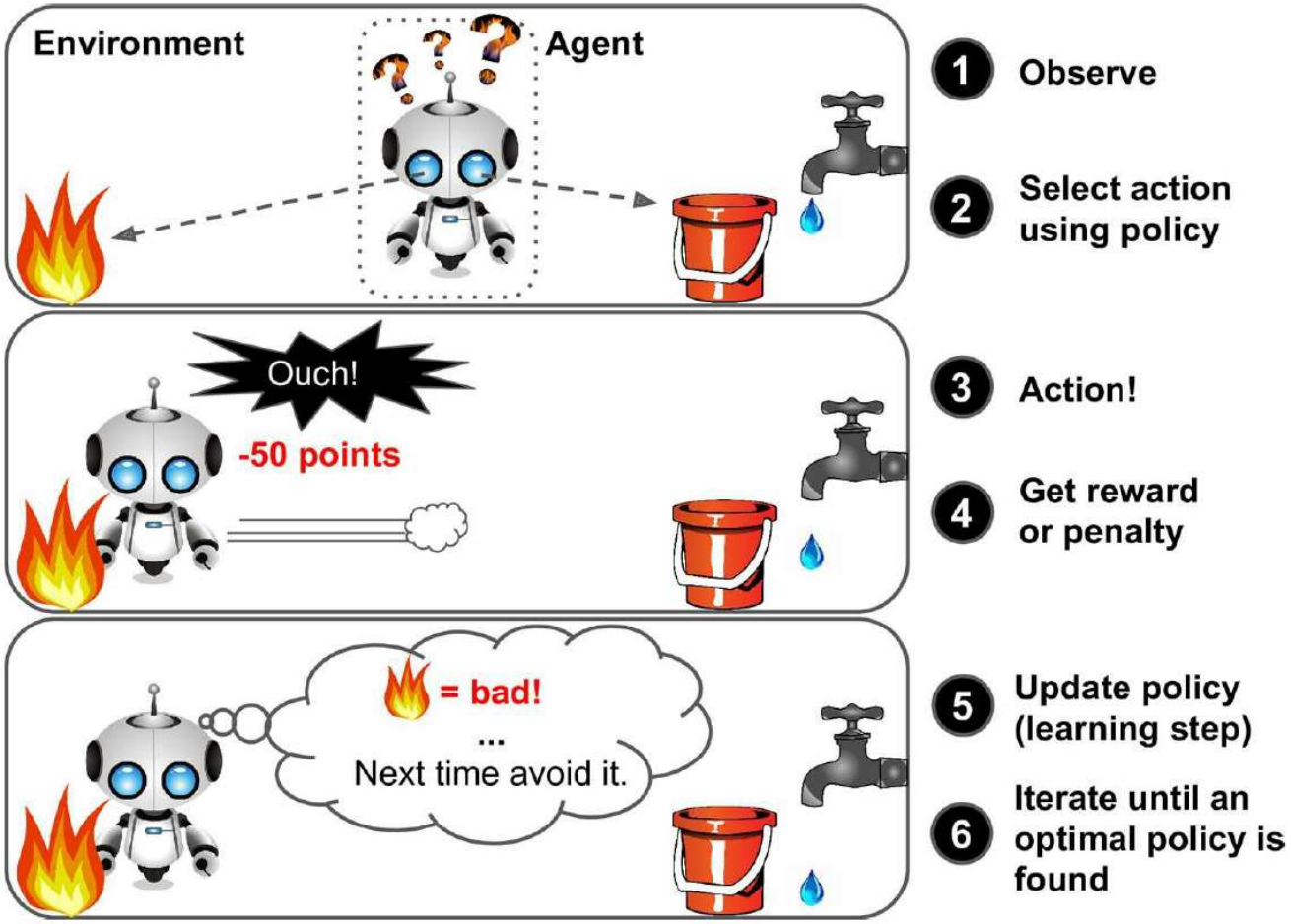


(© O'Reilly)

$$(x_1, y_1), \ \ldots, \ (x_l, y_l) \in \mathcal{X} \times \mathcal{Y}, \qquad x_{l+1}, \ \ldots, \ x_n \in \mathcal{X}$$

- Partially labeled data: lots of unlabeled and a little bit of labeled.
  - Cluster-and-label. Split data into clusters, then label each cluster by the majority/average of labeled points in that cluster; or learn a separate predictor on each cluster's labeled data and apply it to that cluster's unlabeled data.
  - Self-training. Train on labeled data; predict on unlabeled data; leave *best* predictions (with predictor's highest confidence); repeat till convergence.

Reinforcement Learning



(© O'Reilly)

- There are *agents* that observe *state* of the *environment* and perform some *actions* trying to maximize some *reward*. Agent's task is to learn the best *policy*.