

Semi-Supervised Learning

Last modified: 2026-10-25

```
# The Python chunks on this page import numpy; reticulate provisions it.  
# The Julia chunks run through JuliaCall.  
library(reticulate)  
py_require(c("numpy", "scipy"))
```

i Source

Adapted from Logan Sizemore's Spring 2025 lecture on semi-supervised learning.

Course Overview sorts problems by what the data carries: with labels the problem is supervised, without them unsupervised, and with a mixture semi-supervised. This chapter is about that mixture. In practice it is the commonest of the three situations.

1 The setting

- Labels are expensive and unlabeled data is nearly free. Scraping a million web pages costs an afternoon; having a person read and categorize them does not. The same asymmetry holds for medical images, audio recordings and sensor logs
- So the realistic dataset is a small labeled set $\mathcal{L} = \{(x_{(i)}, y_{(i)})\}_{i=1}^{N_L}$ beside a large unlabeled pool $\mathcal{U} = \{x_{(j)}\}_{j=1}^{N_U}$, with $N_U \gg N_L$
- A fit on $\mathcal{L}$ alone cannot pin down the decision boundary. There are too few points, and a boundary that separates a handful of examples perfectly can lie almost anywhere
- The unlabeled points carry no labels, but they do reveal where the data *are*. That is enough to eliminate many boundaries

i Read

GeeksforGeeks explains semi-supervised learning^a [1].

^a<https://www.geeksforgeeks.org/machine-learning/ml-semi-supervised-learning/>

Exercise 1.1 (Brainstorm uses for unlabeled data). You have a few hundred labeled examples and a million unlabeled ones, and you want a better classifier. Unlabeled points have no labels to fit. List as many ways as you can think of in which they could still improve a supervised model.

Solution

Solution 1.1. Every answer rests on the same fact: unlabeled points reveal where the data *are*, even though they say nothing directly about the labels. Some answers this chapter develops:

- **Locate low-density regions.** If the data form clusters, a decision boundary should pass through the gaps between them (the cluster assumption of Section 2), and only unlabeled points can show

where the gaps are.

- **Fit a generative model.** Fit a mixture to all the points, and let the few labels name the components (semi-supervised Expectation-Maximization, or EM, below).
- **Pseudo-label.** Train on the labeled points, label the confident unlabeled points with the model's own predictions (these are *pseudo-labels*), and retrain (self-training).
- **Use neighbors.** Build a graph over all the points and spread each label to nearby points (smoothness and manifold assumptions).

Others appear in practice: pre-train a neural network on the unlabeled pool to learn features, then fine-tune on the labeled set; or ask the model to give the same prediction for an unlabeled point and a slightly perturbed copy of it.

2 What has to be true for this to work

Unlabeled data cannot help on its own. Where the points lie says nothing about their labels unless location and label are tied together. Each semi-supervised method states that tie as an assumption, and most rest on one of three [2]:

- **Smoothness.** If two inputs are close, their labels are probably the same. K-nearest neighbors already relies on this
- **Cluster.** Inputs fall into groups, and each group mostly carries a single label. A decision boundary should then run through a *low-density* gap between groups, not through the middle of one, where it would separate points that mostly agree
- **Manifold.** Inputs in many dimensions often sit close to a surface of far fewer dimensions. Distances measured along that surface are the ones that reflect similarity. The unlabeled pool lets you estimate its shape

When these hold, unlabeled data helps. When they fail — when classes overlap, or the clusters cut across the labels — semi-supervised learning can do **worse** than ignoring the unlabeled data, because it confidently propagates structure that is not aligned with the thing being predicted. This problem is not rare, so we state the assumptions before using the method.

3 Semi-supervised EM for Gaussian mixtures

Unsupervised Learning fits a Gaussian mixture by Expectation-Maximization (EM). The E-step, or expectation step, computes each component's *responsibility* for each point,

$$\gamma_{ic} = \frac{p(x_{(i)}; \mu_{(c)}, \Sigma_{(c)}) w_c}{\sum_{k=1}^M p(x_{(i)}; \mu_{(k)}, \Sigma_{(k)}) w_k} \quad (1)$$

and the M-step, or maximization step, re-fits each component to the data, weighted by those responsibilities.

The semi-supervised version changes **one line**:

- For an **unlabeled** point, γ_{ic} is computed by Equation 1, unchanged
- For a **labeled** point with label c^* , the responsibility is not computed. It is *fixed* to the label:

$$\gamma_{ic} = \begin{cases} 1 & c = c^* \\ 0 & \text{otherwise} \end{cases}$$

- The M-step is untouched. It was already a weighted fit, and the labeled points simply arrive with weights of 1 and 0 rather than with weights the E-step chose
- Iterate to convergence as usual, stopping when the change in log-likelihood falls below a threshold

The two extremes show why this is the right generalization. With no labeled points it is ordinary EM. With every point labeled, every responsibility is fixed, the E-step does nothing, and a single M-step fits a multivariate normal distribution to each class on its own, with the mixing weights becoming the class priors — a purely supervised fit, reached in one step with nothing to iterate. Semi-supervised EM interpolates between the two, and the mixing is automatic: the labeled points tie the components to the right classes while the unlabeled points supply the shape.

Watch

- Andrew Ng fits a mixture of Gaussians by EM, from 20:40^a, and derives EM from Jensen's inequality, from 48:12^b [3], Lecture 13.
- The next lecture^c continues with EM [3], Lecture 14.

^a<https://www.youtube.com/watch?v=rVfZHWTwXSA&t=1240s>

^b<https://www.youtube.com/watch?v=rVfZHWTwXSA&t=2892s>

^c<https://www.youtube.com/watch?v=tw6cmL5STuY>

Exercise 3.1 (Compute a responsibility for an unlabeled point). A one-dimensional mixture of two Gaussians has components $N(0, 1)$ and $N(4, 1)$, with equal mixing weights $w_1 = w_2 = \frac{1}{2}$. The training set holds two labeled points, $x = -1$ in component 1 and $x = 5$ in component 2, and one unlabeled point, $x = 1.5$.

Compute the responsibilities γ_{i1} and γ_{i2} of each component for the two labeled points.

Solution

Solution 3.1. The two labeled points have fixed responsibilities: $(\gamma_{11}, \gamma_{12}) = (1, 0)$ for $x = -1$ and $(0, 1)$ for $x = 5$.

Exercise 3.2 (Compute a responsibility for an unlabeled point (part 2)). Use the setup of Exercise 3.1. Compute the responsibilities γ_{i1} and γ_{i2} of each component for the unlabeled point.

Solution

Solution 3.2. For the unlabeled point, Equation 1 applies. The equal weights cancel, so the responsibility is the component's density over the sum of both densities. The unit variances^a leave only the exponentials, with $(1.5 - 0)^2/2 = 1.125$ and $(1.5 - 4)^2/2 = 3.125$:

$$\begin{aligned}\gamma_{31} &= \frac{\exp\{-1.125\}}{\exp\{-1.125\} + \exp\{-3.125\}} \\ &= \frac{1}{1 + \exp\{-2\}} \\ &\approx 0.881,\end{aligned}$$

so $\gamma_{32} = 1 - 0.881 = 0.119$. The point is nearer component 1, which has most of the responsibility, but component 2 keeps a share because 1.5 is not far from the gap between the means^b.

3.0.1 Python

```
from scipy.stats import norm
```

```
x_u = 1.5
```

```
# Equal mixing weights and equal variances, so the responsibility is  
# each component's density over the sum of the two densities.
```

```
dens = [norm.pdf(x_u, loc=m, scale=1.0) for m in (0.0, 4.0)]  
[round(float(d / sum(dens)), 3) for d in dens]
```

```
[0.881, 0.119]
```

3.0.2 R

```
x_u <- 1.5
# Equal mixing weights and equal variances, so the responsibility is
# each component's density over the sum of the two densities.
dens <- dnorm(x_u, mean = c(0, 4), sd = 1)
resp_u <- dens / sum(dens)
round(resp_u, 3)

[1] 0.881 0.119
```

^a<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

^b<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Exercise 3.3 (Update a component's mean). Use the responsibilities of Solution 3.1 and Solution 3.0.2. What mean does the M-step give component 1?

Solution

Solution 3.3. The M-step is a weighted mean. Component 1's weights on the three points are 1, 0 and γ_{31} (the unlabeled point's responsibility from Solution 3.0.2), so

$$\begin{aligned}\mu_{(1)} &= \frac{1(-1) + 0(5) + \gamma_{31}(1.5)}{1 + 0 + \gamma_{31}} \\ &= \frac{-1 + 0.881 \times 1.5}{1.881} \\ &\approx 0.171.\end{aligned}$$

```
# Weighted mean of all three points for component 1: its responsibilities are
# 1 and 0 on the labeled points and resp_u[1] on the unlabeled one.
x_all <- c(-1, 5, 1.5)
resp_1 <- c(1, 0, resp_u[1])
sum(resp_1 * x_all) / sum(resp_1)

[1] 0.1707763
```

Exercise 3.4 (Update a component's mean (part 2)). Use the setup of Exercise 3.3. How does this mean^a differ from the mean of the labeled points alone?

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Solution

Solution 3.4. Fitted to the one labeled point alone, component 1 would have mean^a -1 . The unlabeled point pulls it to the right, to about 0.17.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Exercise 3.5 (Update a component's mean (part 3)). Use the setup of Exercise 3.3. Why does this mean^a differ from the mean of the labeled points alone?

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Solution

Solution 3.5. The unlabeled point pulls the mean^a in proportion to the share of the point that component 1 claims. This is the mechanism of the chapter in miniature: the labels tie each component to a class, and the unlabeled points move the components toward where the data lie.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

4 Self-training

Self-training is the simplest semi-supervised method and needs no special model — it works with any supervised learner that reports confidence.

1. Fit a supervised model on the labeled set $\mathcal{L}$
2. Predict labels for every point in $\mathcal{U}$
3. Keep the predictions whose confidence exceeds a threshold
4. Move those points into $\mathcal{L}$, carrying their *predicted* labels — their **pseudo-labels**
5. Refit on the enlarged labeled set
6. Repeat from step 2 until a stopping criterion is met: the pool empties, no prediction clears the threshold, or a fixed number of rounds elapses

A confidently classified unlabeled point is evidence about where the boundary is, even though nobody checked its label. Absorbing it pulls the model towards the data, and the boundary moves into the low-density region the cluster assumption promised.

i Watch

Abbeel's lecture on semi-supervised learning, in a course on deep unsupervised learning, includes self-training [4].

i Read

GeeksforGeeks explains self-training^a [5].

^a<https://www.geeksforgeeks.org/machine-learning/self-training-in-semi-supervised-learning/>

Exercise 4.1 (Follow a confident mistake through self-training). In the first round of self-training, the model labels an unlabeled point class A with confidence 0.95, but the point's true class is B. Describe what happens to this point over the next two rounds.

Solution

Solution 4.1. The point enters the training set with the pseudo-label A, on the same footing as a real label. The refitted model is trained to call that point A, and so is more confident about it, and about nearby points, than before. In the next round the point stays in the set, and its neighbors are now more likely to be pseudo-labeled A too, so one early error spreads.

Exercise 4.2 (Follow a confident mistake through self-training (part 2)). Use the setup of Exercise 4.1. Can the procedure notice the mistake?

Solution

Solution 4.2. The procedure cannot notice the mistake, because it never sees the true class. Only a held-out labeled set can reveal the damage.

Exercise 4.3 (Follow a confident mistake through self-training (part 3)). Use the setup of Exercise 4.1. What can you do to limit the damage?

Solution

Solution 4.3. To limit the damage, use a high confidence threshold so that fewer pseudo-labels are added, run only a few rounds rather than until the pool is empty, and check accuracy on held-out labeled data after each round. This is the confirmation bias^a described in the failure mode below.

^a<https://morrison-lab.github.io/sds/estimation.html#def-bias>

4.1 The failure mode

- The procedure cannot detect a confident mistake. It adds a pseudo-label with exactly the same status as a real one, so an early error becomes training data, the refitted model grows more confident in it, and the error reinforces itself
- This is *confirmation bias*, so we run self-training with a high confidence threshold and few rounds rather than until the pool is empty
- In summary, self-training amplifies whatever the initial model got right, and also whatever it got wrong. Its value depends on the initial model being right more often than the threshold implies

4.2 Student–teacher training

- This variant gives the two jobs to two models. We fit a **teacher** to the labeled data and use it to pseudo-label the unlabeled pool. We then fit a **student** to those pseudo-labels
- The student is a different model, so it may be larger than the teacher. We can also train it with noise and augmentation that the teacher never saw, so that it learns to be robust and does not just copy the teacher [6]. Most *knowledge distillation* and *noisy student* pipelines have this form

i Read

GeeksforGeeks explains knowledge distillation^a [7].

^a<https://www.geeksforgeeks.org/machine-learning/knowledge-distillation/>

5 Other approaches

Semi-supervised learning is an active research area and the list is long. Three more worth knowing by name:

- **Co-training.** Split the features into two views that are each sufficient for the task — the text of a web page and the text of the links pointing at it [8]. Train one model per view, and let each teach the other by supplying pseudo-labels the other's view cannot see. The two views' independence keeps the mutual reinforcement from being circular
- **Label propagation.** Build a graph over all points, labeled and unlabeled, with edges weighted by similarity, then spread labels along the edges until they settle [9]. This is the smoothness assumption implemented literally

- **Consistency regularization.** Add a loss term penalizing the model for changing its prediction when an unlabeled input is perturbed in a label-preserving way. It needs no pseudo-labels, so it avoids the confirmation bias of self-training

i Read

GeeksforGeeks explains graph-based semi-supervised learning^a [10], the family label propagation belongs to.

^a<https://www.geeksforgeeks.org/data-science/introduction-to-graph-based-semi-supervised-learning/>

6 Self-training in code

```
# Repeats the R tab's self-training run (below), recording each round so
# the prose can quote it.
st_boundary <- function(x, y) 0.5 * (mean(x[y < 0]) + mean(x[y > 0]))
st_x <- c(-0.6, 2.8)
st_y <- c(-1, 1)
st_pool <- c(-3.1, -2.6, -2.2, -1.8, -1.4, -0.9,
            0.4, 0.7, 1.0, 1.3, 1.7, 2.1, 2.5, 3.0)
st_b <- st_boundary(st_x, st_y)
st_n_pool <- length(st_pool)
st_wrong <- sign(st_pool - st_b) != sign(st_pool)
st_rank <- rank(abs(st_pool - st_b))
ordinal <- c("first", "second", "third", "fourth", "fifth", "sixth", "seventh")
st_wrong_ranks <- ordinal[sort(st_rank[st_wrong])]
st_wrong_x <- st_pool[st_wrong]
st_hist <- st_b
st_round <- rep(NA_integer_, st_n_pool) # the round each point is absorbed in
names(st_round) <- st_pool
for (step in 1:7) {
  take <- order(abs(st_pool - st_b), decreasing = TRUE)[1:2]
  st_round[as.character(st_pool[take])] <- step
  st_x <- c(st_x, st_pool[take])
  st_y <- c(st_y, sign(st_pool[take] - st_b))
  st_pool <- st_pool[-take]
  st_b <- st_boundary(st_x, st_y)
  st_hist <- c(st_hist, st_b)
}
pseudo <- seq(3, length(st_x))
stopifnot(all(sign(st_x[pseudo]) == st_y[pseudo]))
stopifnot(all(st_round[as.character(st_wrong_x)] >= 6))
st_mid <- round(range(st_hist[3:6]), 2)
```

The code in the Python, R and Julia tabs runs self-training on a one-dimensional problem where the true boundary is at 0. Only two points carry labels, and they sit badly enough that the supervised-only boundary lands at 1.1, which misclassifies 3 of the 14 unlabeled points. The classifier is nearest-centroid, so the boundary is the midpoint of the two class means, and confidence is distance from that boundary. Each round absorbs the two most confident points.

6.1 Python

```
import numpy as np
```

```
def boundary(labeled_x, labeled_y):
    return 0.5 * (labeled_x[labeled_y < 0].mean() + labeled_x[labeled_y > 0].mean())

labeled_x = np.array([-0.6, 2.8])
labeled_y = np.array([-1.0, 1.0])
pool = np.array([-3.1, -2.6, -2.2, -1.8, -1.4, -0.9,
                 0.4, 0.7, 1.0, 1.3, 1.7, 2.1, 2.5, 3.0])
truth = np.where(pool > 0, 1.0, -1.0)

b = boundary(labeled_x, labeled_y)
print(0, round(b, 3), round(np.mean(np.sign(pool - b) == truth), 3))
```

```
0 1.1 0.786
```

```
for step in range(1, 8):
    confidence = np.abs(pool - b)
    take = np.argsort(-confidence)[:2]
    labeled_x = np.concatenate([labeled_x, pool[take]])
    labeled_y = np.concatenate([labeled_y, np.sign(pool[take] - b)])
    pool = np.delete(pool, take)
    b = boundary(labeled_x, labeled_y)
    print(step, round(b, 3),
          round(np.mean(np.sign(labeled_x[2:] - 0.0) == labeled_y[2:]), 3))
```

```
1 0.35 1.0
2 0.387 1.0
3 0.353 1.0
4 0.325 1.0
5 0.31 1.0
6 0.129 1.0
7 -0.039 1.0
```

6.2 R

```
boundary <- function(labeled_x, labeled_y) {
  0.5 * (mean(labeled_x[labeled_y < 0]) + mean(labeled_x[labeled_y > 0]))
}

labeled_x <- c(-0.6, 2.8)
labeled_y <- c(-1, 1)
pool <- c(-3.1, -2.6, -2.2, -1.8, -1.4, -0.9,
         0.4, 0.7, 1.0, 1.3, 1.7, 2.1, 2.5, 3.0)
truth <- ifelse(pool > 0, 1, -1)

b <- boundary(labeled_x, labeled_y)
cat(0, round(b, 3), round(mean(sign(pool - b) == truth), 3), "\n")
```

```
0 1.1 0.786
```

```
for (step in 1:7) {
  confidence <- abs(pool - b)
  take <- order(confidence, decreasing = TRUE)[1:2]
  labeled_x <- c(labeled_x, pool[take])
  labeled_y <- c(labeled_y, sign(pool[take] - b))
  pool <- pool[-take]
  b <- boundary(labeled_x, labeled_y)
```

```

pseudo <- seq(3, length(labeled_x))
cat(step, round(b, 3),
     round(mean(sign(labeled_x[pseudo]) == labeled_y[pseudo]), 3), "\n")
}

```

```

1 0.35 1
2 0.387 1
3 0.353 1
4 0.325 1
5 0.31 1
6 0.129 1
7 -0.039 1

```

6.3 Julia

```

boundary(labeled_x, labeled_y) =
    0.5 * (sum(labeled_x[labeled_y .< 0]) / count(labeled_y .< 0) +
           sum(labeled_x[labeled_y .> 0]) / count(labeled_y .> 0));

labeled_x = [-0.6, 2.8];
labeled_y = [-1.0, 1.0];
pool = [-3.1, -2.6, -2.2, -1.8, -1.4, -0.9,
         0.4, 0.7, 1.0, 1.3, 1.7, 2.1, 2.5, 3.0];
truth = [x > 0 ? 1.0 : -1.0 for x in pool];

b = boundary(labeled_x, labeled_y);
println(0, " ", round(b, digits = 3), " ",
        round(count(sign.(pool .- b) .== truth) / length(truth), digits = 3))

```

```

0 1.1 0.786

```

```

for step in 1:7
    confidence = abs.(pool .- b)
    take = sortperm(confidence, rev = true)[1:2]
    append!(labeled_x, pool[take])
    append!(labeled_y, sign.(pool[take] .- b))
    deleteat!(pool, sort(take))
    global b = boundary(labeled_x, labeled_y)
    pseudo = 3:length(labeled_x)
    println(step, " ", round(b, digits = 3), " ",
            round(count(sign.(labeled_x[pseudo]) .== labeled_y[pseudo]) /
                  length(pseudo), digits = 3))
end

```

```

1 0.35 1.0
2 0.387 1.0
3 0.353 1.0
4 0.325 1.0
5 0.31 1.0
6 0.129 1.0
7 -0.039 1.0

```

Each line prints the round, the current boundary, and the fraction of pseudo-labels so far that are actually correct; round 0, before any pseudo-labels exist, prints instead the fraction of the pool the supervised-only boundary gets right. The boundary starts at 1.1, where the two labeled points put it; the first round moves it to 0.35, the next four leave it between 0.31 and 0.39, and the last two carry it to 0.129 and then -0.039 once the pool is exhausted. Every pseudo-label the procedure assigns along the way is correct, including the 3

points the supervised-only model had wrong — those 3 rank first, third and fifth by ascending confidence, so they are among the least confident points at the start and the procedure absorbs them in the final two rounds, by which time the boundary has already moved past them.

That ordering is the whole mechanism, and it is also the weakness. Absorbing the most confident points first gives self-training a chance to correct itself. Absorb in a different order, or set the threshold low enough that ambiguous points enter early, and the same procedure reinforces the initial mistake instead.

7 Further reading

- Chapelle et al. [2] — where the three assumptions of Section 2 are set out
- van Engelen and Hoos [11] — a survey of the main families
- Xie et al. [6] — the student–teacher pipeline at scale

References

- [1] GeeksforGeeks, “Semi-supervised learning in ML.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/ml-semi-supervised-learning/>
- [2] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-supervised learning*. MIT Press, 2006. doi: 10.7551/mitpress/9780262033589.001.0001¹.
- [3] A. Ng *et al.*, “CS229: Machine learning, autumn 2018.” Stanford Online, lecture videos on YouTube, 2018. Available: <https://www.youtube.com/playlist?list=PLoROMvovd4rMiGQp3WXShtMGgzqpfVfbU>
- [4] P. Abbeel, “Semisupervised learning and unsupervised distribution alignment.” UC Berkeley, Spring 2024, Lecture 10; video on YouTube, 2024. Available: <https://www.youtube.com/watch?v=HTtN7rO-jBI>
- [5] GeeksforGeeks, “Self-training in semi-supervised learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/self-training-in-semi-supervised-learning/>
- [6] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, “Self-training with noisy student improves ImageNet classification,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2020, pp. 10684–10695. doi: 10.1109/CVPR42600.2020.01070².
- [7] GeeksforGeeks, “Knowledge distillation.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/knowledge-distillation/>
- [8] Wikipedia contributors, “Co-training — Wikipedia, the free encyclopedia.” 2026. Available: <https://en.wikipedia.org/wiki/Co-training>
- [9] Wikipedia contributors, “Label propagation algorithm — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Label_propagation_algorithm
- [10] GeeksforGeeks, “Introduction to graph-based semi-supervised learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/data-science/introduction-to-graph-based-semi-supervised-learning/>
- [11] J. E. van Engelen and H. H. Hoos, “A survey on semi-supervised learning,” *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020, doi: 10.1007/s10994-019-05855-6³.

¹<https://doi.org/10.7551/mitpress/9780262033589.001.0001>

²<https://doi.org/10.1109/CVPR42600.2020.01070>

³<https://doi.org/10.1007/s10994-019-05855-6>