Revision: SSL

Minor in AI - IIT ROPAR

3rd May, 2025

The Role of Assumptions in Learning

In machine learning, particularly in settings where labeled data is scarce, it is common to make certain **assumptions** about the structure of the data in order to enable effective learning. These assumptions help models generalize from a small set of labeled examples to the larger, unlabeled dataset. Among the most critical of these is the **Smoothness Assumption**, especially in semi-supervised learning.

What is the Smoothness Assumption?

The **Smoothness Assumption** posits that:

> If two data points x and x' in the input space are "close" to each other (under some appropriate metric), then their corresponding labels y and y' should also be "similar" or the same.

Mathematically, we can express this as:

$$\text{If } \|x - x'\| \text{ is small} \quad \Rightarrow \quad \text{Then } |f(x) - f(x')| \text{ is also small}$$

where $f(x)$ is the function (classifier or regressor) mapping inputs to outputs, and $\|\cdot\|$ is a suitable distance metric (e.g., Euclidean distance).

This assumption implies that the target function f is **Lipschitz continuous** in some regions:

$$|f(x) - f(x')| \leq L \cdot \|x - x'\|$$

for some constant $L > 0$, at least in the local neighborhood of x.

Illustrative Example in Machine Learning

Consider the task of classifying handwritten digits (such as using the MNIST dataset). Let x be an image of the handwritten digit "3". If we slightly modify this image—say by adding small amounts of noise or shifting a pixel or two—we obtain a new image x'. To a human observer, this modified image clearly still represents the digit "3".

The Smoothness Assumption implies that:

$$f(x) = \text{``3''} \quad \Rightarrow \quad f(x') \approx \text{``3''}$$

This behavior is crucial for robustness: we want our classifier not to change its output due to minute, insignificant variations in input.

Relevance to Semi-Supervised Learning

In **semi-supervised learning**, we are often given:

- A small set of labeled data points $\{(x_i, y_i)\}_{i=1}^{l}$
- A much larger set of unlabeled data points $\{x_j\}_{j=l+1}^{l+u}$

The Smoothness Assumption underpins one of the fundamental techniques in this paradigm: **label propagation** or **manifold regularization**. If we believe that labels change smoothly over the data manifold, then we can propagate the labels from labeled points to nearby unlabeled points based on proximity.

This is often formalized by minimizing a loss function with a smoothness regularizer, such as:

$$\sum_{i=1}^{l} \mathcal{L}(f(x_i), y_i) + \lambda \sum_{i,j} W_{ij} \|f(x_i) - f(x_j)\|^2$$

Here, W_{ij} is a weight measuring the similarity or closeness between x_i and x_j (e.g., using a Gaussian kernel), and the second term enforces that similar inputs should lead to similar outputs.

Real-world Intuition: Market Stall Analogy

To understand this assumption intuitively, consider a market stall with several boxes of apples. If you examine a few apples from a box and determine that they are apples of a certain variety (say, Fuji), you are likely to assume that all nearby apples in the same box are also Fuji apples.

This is an application of the Smoothness Assumption:

- Input space: Spatial proximity of apples in the box.
- Label: Apple variety.
- Assumption: Nearby apples (small changes in position) should have the same label (variety).

What is the Cluster Assumption?

The **Cluster Assumption** (also known as the **low-density separation assumption**) posits that:

> Data naturally forms clusters in the input space, and points within the same cluster are likely to belong to the same class label. Therefore, decision boundaries should lie in low-density regions between these clusters.

This can be visualized as:

- High-density regions (clusters) contain examples with the same label.
- Low-density regions (between clusters) are optimal places to place class decision boundaries.

Formally, this can be expressed as:

$$\text{If } x \text{ and } x' \text{ belong to the same cluster} \Rightarrow f(x) = f(x')$$

$$\text{Decision boundary } \partial f \text{ lies in a region where the data density } p(x) \text{ is low}$$

Illustrative Example in Machine Learning

Consider the classification of handwritten digits (e.g., using the MNIST dataset). If you visualize these high-dimensional digit images using a dimensionality reduction technique like t-SNE or UMAP, you will notice that:

- Images of the digit "0" cluster together.
- Images of "1" form another tight cluster.
- And so on for each digit.

Thus, if two images x and x' are in the same cluster in the embedded space, they likely represent the same digit:

$$x, x' \in \mathcal{C}_3 \Rightarrow f(x) = f(x') = \text{``3''}$$

A decision boundary between digits "3" and "5" should ideally lie in the sparse region between their clusters, where ambiguous or low-probability samples exist.

Implications in Semi-Supervised Learning

The Cluster Assumption is essential for designing algorithms that work with minimal supervision. Here's how it contributes:

1. Only a few labeled examples per cluster are needed.
2. The model can then propagate the label through the entire cluster using unsupervised density structure.
3. Label propagation algorithms (like label spreading, spectral graph methods) often rely on this assumption.

Many semi-supervised algorithms build a similarity graph over data points (using k-nearest neighbors or Gaussian kernels), then optimize a label assignment function f that is smooth with respect to the graph Laplacian. This smoothness is motivated directly by the cluster assumption.

Formal Objective (Graph-based Interpretation)

Let W_{ij} be the similarity between data points x_i and x_j in a graph G over the data. Then, under the cluster assumption, we minimize:

$$\sum_{i,j} W_{ij}(f(x_i) - f(x_j))^2$$

This encourages $f(x)$ to be approximately constant within clusters and vary only between them, consistent with placing boundaries in low-density areas.

Real-world Analogy: Songs and Music Genres

Imagine you are sorting a large collection of songs using audio features like tempo, rhythm, instrumentation, and mood. Songs with similar properties (e.g., fast tempo, heavy distortion) naturally group together.

- Songs in one cluster may all belong to the "Rock" genre.
- Another cluster might correspond to "Classical" music.

If you label just a few songs from each genre, and trust the cluster structure, you can infer the genres of all other songs in the dataset.

This reflects the cluster assumption:

- Songs = data points
- Features = input space
- Genre = class label
- Cluster = high-density region of similar songs

What is the Manifold Assumption?

The **Manifold Assumption** is a central hypothesis in machine learning, especially in fields involving high-dimensional input data such as images, speech, and text. It proposes that although data may be represented in a very high-dimensional input space, the true degrees of freedom underlying the data are much fewer. Specifically, it states:

> *Real-world high-dimensional data is distributed near or on a low-dimensional manifold embedded within the high-dimensional ambient space.*

In mathematical terms, if our input data lives in $\mathbb{R}^D$, where D is very large (e.g., $D = 784$ for a 28×28 grayscale image), then there exists a lower-dimensional manifold $\mathcal{M} \subset \mathbb{R}^D$ of dimension d (where $d \ll D$) such that:

$$x \in \mathcal{M} \quad \text{or} \quad x \approx \mathcal{M}, \quad \forall x \in \mathcal{X}$$

This manifold captures the essential structure of the data, filtering out the redundant or irrelevant dimensions in which the data does not vary meaningfully.

Geometric Intuition: The Crumpled Paper Analogy

One of the most widely used visual analogies for the manifold assumption is that of a piece of paper crumpled into a ball and placed inside a three-dimensional space:

- The sheet of paper, when flat, is a 2D surface.
- When crumpled, it appears more complex and fills more space, yet it still has only two degrees of freedom.
- Although embedded in $\mathbb{R}^3$, the crumpled paper is still intrinsically a 2D manifold.

Similarly, even though image data may be represented using hundreds or thousands of pixels, the actual variation in meaningful content (e.g., rotation, lighting, pose, expression) often resides on a much smaller number of axes.

Illustrative Example in Machine Learning: The Cat Manifold

Let us consider a machine learning problem where we are training a model to classify images of cats. Each image is represented as a high-dimensional vector of pixel intensities. However, not all combinations of pixel values are valid images of cats—only a small subset forms natural-looking cats.

- These natural variations of cat images—due to changes in pose, lighting, background, or fur texture—form a **smooth and continuous manifold** in the input space.
- We can denote this manifold as $\mathcal{M}_{\text{cat}}$.
- Even though $\mathcal{M}_{\text{cat}} \subset \mathbb{R}^D$ (where D is large), the intrinsic dimension of $\mathcal{M}_{\text{cat}}$ is relatively small.

This structure implies that machine learning algorithms need not learn to classify over the entire ambient space, but rather only over this much smaller, structured region.

Concrete Real-World Analogy: Fruit Image Manifolds

Imagine a dataset of photographs of different fruits: apples, bananas, and oranges. Each photograph is an image that can vary in background, lighting conditions, and camera angle. However, the class of each fruit defines a distinct set of visual characteristics that vary in a structured way.

- All valid apple images form a surface-like structure $\mathcal{M}_{\text{apple}}$.
- All valid banana images form another structure $\mathcal{M}_{\text{banana}}$, and similarly for oranges.
- These manifolds may be curved and twisted, but they are smooth, coherent surfaces.

These manifolds are likely to be well-separated and low-dimensional relative to the full image space, and this separation helps classifiers distinguish between classes with fewer labeled examples.

Importance in Semi-Supervised Learning

The manifold assumption plays a foundational role in **semi-supervised learning** (SSL), where only a small subset of the data is labeled. The key insight is that:

- Unlabeled data can help to reveal the shape and connectivity of the manifold.
- If the classifier's decision function is constrained to vary smoothly along the manifold, then labels can be propagated to nearby unlabeled points efficiently.

This enables label-efficient learning, where the geometric structure of the data itself guides the model toward correct generalizations.

Graph-Based Algorithms and Manifold Learning

A common way to operationalize the manifold assumption is to construct a graph over the data:

- Nodes represent data points.
- Edges encode similarity, often via k-nearest neighbors or Gaussian kernels.
- Edge weights reflect proximity on the manifold.

Using this graph, we can define smoothness over the manifold and regularize the learning function accordingly.

Manifold Regularization

In this framework, the learning objective becomes:

$$\sum_{i=1}^{l} \mathcal{L}(f(x_i), y_i) + \lambda \|f\|_{\mathcal{H}}^2 + \gamma \int_{\mathcal{M}} \|\nabla_{\mathcal{M}} f(x)\|^2 dp(x)$$

Where:

- $\mathcal{L}$ is the supervised loss over labeled data.
- $\|f\|_{\mathcal{H}}$ is a regularization term from the function space (e.g., RKHS norm).
- The third term penalizes the gradient of the function f along the manifold, encouraging smooth variation over the data geometry.

This approach is used in methods such as Laplacian SVMs, label propagation, and diffusion-based learning.

Semi Supervised Learning Methods

1. Inductive Learning in SSL

Definition

Inductive learning refers to learning a function $f : \mathcal{X} \to \mathcal{Y}$ that generalizes beyond the observed data. The model learns a decision boundary or function that can be applied to any unseen data point drawn from the same distribution.

Formally:

- Given a labeled training set $L = \{(x_1, y_1), \ldots, (x_l, y_l)\}$,
- And an unlabeled set $U = \{x_{l+1}, \ldots, x_{l+u}\}$,
- Learn a mapping $f : \mathcal{X} \to \mathcal{Y}$ that works for all future $x \in \mathcal{X}$.

Purpose

The goal is to learn a predictive function that works on any unseen input from the data distribution, not just a fixed test set.

Example Use Cases

- Email spam detection: The model must classify future incoming emails.
- Medical diagnosis: Once trained, the model must diagnose new patients.
- Voice assistants: Should understand commands from new speakers.

Common Methods

- Semi-supervised SVMs
- Graph-based label propagation + classifier training
- Consistency regularization methods (e.g., FixMatch, Mean Teacher)

2. Transductive Learning in SSL

Definition

Transductive learning, in contrast, does not attempt to learn a general function over the entire input space. Instead, it focuses solely on predicting the labels of a specific set of unlabeled data given during training.

Formally:

- Given labeled data $L = \{(x_1, y_1), \ldots, (x_l, y_l)\}$,
- And a **specific** unlabeled test set $U = \{x_{l+1}, \ldots, x_{l+u}\}$,
- The goal is to infer $\{y_{l+1}, \ldots, y_{l+u}\}$ without building a generalizable function.

Purpose

Rather than generalize, the aim is to optimally classify only the known test instances. No assumptions are made about performance on unseen data not present during training.

Example Use Cases

- Document classification: Label a fixed batch of documents.
- Social network analysis: Infer roles of specific users.
- Market segmentation: Predict cluster labels of a known customer pool.

Common Methods

- Label propagation on graphs
- Transductive SVMs (TSVM)
- Spectral methods and graph Laplacians

Theoretical Contrast

Inductive

Aims to approximate a global mapping function. Unlabeled data assists in improving the model's decision boundary, enhancing generalization.

Transductive

No function is explicitly constructed. The model makes local inferences only on the given test set, potentially using unlabeled examples as anchor points or graph nodes.

Summary: Comparison Table

Aspect	Inductive Learning	Transductive Learning
Goal	Learn a general function $f : \mathcal{X} \to \mathcal{Y}$	Predict labels for a specific test set
Use of Unlabeled Data	Helps shape a better decision boundary	Helps infer structure within the known dataset
Applicability	Any future input instance	Only current known unlabeled examples
Generality	Generalizes to unseen data	Does not generalize beyond test set
Common Methods	Semi-supervised SVMs, consistency-based methods	Label propagation, transductive SVMs
Typical Use Cases	Spam filtering, diagnostics, recommendation systems	Fixed-batch document classification, network role detection

Table 1: Comparison between Inductive and Transductive Learning in SSL

Introduction to Self-Training

Self-training is a classic yet powerful approach to semi-supervised learning. It extends supervised learning by making use of unlabeled data through a process called *pseudo-labeling*. The core idea is intuitive: allow a model trained on a small amount of labeled data to predict labels for unlabeled data, then retrain using the most confident predictions as if they were true labels.

Self-training is particularly useful when:

- Labeled data is scarce or expensive to obtain.
- Unlabeled data is abundant and relatively easy to collect.
- The classifier can estimate confidence scores or probabilities.

Step-by-Step Workflow

The self-training process typically unfolds over several iterations, where the model gradually improves by expanding its training set with pseudo-labeled data.

Step 1: Train Initial Model on Labeled Data

- Begin with a small labeled dataset $L = \{(x_i, y_i)\}_{i=1}^{l}$.
- Train a supervised model f using this dataset.
- The model should produce not just class labels, but also **confidence scores** or probabilities (e.g., softmax outputs).

Step 2: Predict on Unlabeled Data

- Use the model f to predict labels for the unlabeled set $U = \{x_j\}_{j=1}^{u}$.
- Record both the predicted labels $\hat{y}_j$ and their confidence scores $p_j = f(x_j)$.

Step 3: Select High-Confidence Predictions

- Define a confidence threshold τ (e.g., $\tau = 0.80$).
- Only keep predictions where $\max(p_j) \geq \tau$.
- These are treated as **pseudo-labeled examples** $\tilde{L} = \{(x_j, \hat{y}_j)\}$.

Step 4: Augment Labeled Set

- Merge pseudo-labeled data with original labeled data: $L \leftarrow L \cup \tilde{L}$.
- Optionally, remove selected unlabeled samples from U to avoid repeated use.

Step 5: Retrain the Model

- Retrain the model f using the expanded labeled set.
- Improved coverage of the data distribution is expected with more training samples.

Step 6: Iterate

- Repeat steps 2–5 for multiple rounds until convergence or until no high-confidence pseudo-labels remain.
- Optionally adjust the confidence threshold over time.

Illustrative Example: Fruit Classification

Problem Setup

Suppose we want to build a fruit image classifier to distinguish between apples and bananas.

- Labeled set: 50 fruit images labeled as either apple or banana.
- Unlabeled set: 950 fruit images with unknown labels.

Self-Training Process

1. Train a convolutional neural network (CNN) on the 50 labeled samples.
2. Predict labels for all 950 unlabeled images using the CNN.
3. Select predictions with a confidence ≥ 0.80 (e.g., model is 90% sure an image is an apple).
4. Add these confident samples to the labeled set and retrain.
5. Repeat for several rounds. With each round, more data is labeled, improving the model.

Challenges in Self-Training

1. Error Propagation

One of the most critical limitations of self-training is the risk of **error propagation**. Since the model is responsible for generating its own pseudo-labels on the unlabeled data, any incorrect predictions can be mistakenly assumed to be true and added to the labeled dataset. When these incorrect samples are used during retraining, the model reinforces its own misjudgments, leading to a feedback loop of compounding errors. This is particularly problematic in early iterations, when the model is still weak due to limited labeled data. Without safeguards, this self-reinforcement can lead to significant degradation in model performance over time.

2. Sensitivity to Confidence Threshold

Self-training heavily depends on a pre-defined **confidence threshold** to decide which pseudo-labels are trustworthy. A low threshold might allow noisy or incorrect labels into the training set, degrading model accuracy. Conversely, a high threshold may result in very few pseudo-labels being selected, slowing down the learning process or even stalling progress. This delicate trade-off means that the threshold often needs to be fine-tuned, and it may require dynamic adjustment across iterations to balance between learning speed and label quality.

3. Class Imbalance

Class imbalance is another challenge that arises during pseudo-label selection. The model tends to be more confident about certain classes—often the ones with more labeled examples or clearer features. As a result, those classes are overrepresented in the pseudo-labeled data. This imbalance skews the training distribution and may cause the model to be biased towards dominant classes, neglecting underrepresented ones. If left uncorrected, the imbalance can lead to poor generalization and unfair performance across different categories.

Variants and Improvements in Self-Training

1. Soft Labeling

Soft labeling is a technique that uses the full class probability distribution predicted by the model instead of assigning a single hard label. For example, instead of labeling a sample strictly as "apple," the model might assign a probability of 70% to "apple," 20% to "banana," and 10% to "orange." This soft target provides richer information to the model during retraining, preserving uncertainty and reducing the risk of reinforcing incorrect assumptions. Training with soft labels (e.g., using Kullback-Leibler divergence loss) encourages smoother decision boundaries and better generalization.

2. Top-k Class-Wise Selection

To mitigate class imbalance, a **top-k selection strategy per class** can be employed. Instead of selecting pseudo-labels based solely on global confidence, this method identifies the top k most confident predictions *within each class*. This ensures that all classes contribute pseudo-labeled samples, even if some are inherently harder to classify. By maintaining balanced representation across categories, this approach helps the model learn a more equitable decision surface.

3. Class Balance Correction

Class balance correction addresses imbalanced pseudo-labeling by explicitly enforcing class distribution constraints. Techniques include resampling pseudo-labeled data to match target distributions or applying weighting schemes during training that penalize overrepresented classes and boost underrepresented ones. Some approaches also apply adjusted loss functions (e.g., focal loss) to dynamically balance class influence during retraining. This ensures that the classifier does not disproportionately favor certain classes, maintaining fairness and diversity.

4. Ensemble-Based Pseudo-Labeling

Ensemble-based pseudo-labeling improves reliability by leveraging multiple models to generate pseudo-labels collaboratively. Instead of relying on a single model's prediction, an ensemble (e.g., a group of models trained on different subsets or with different seeds) can combine outputs using techniques like majority voting or averaging class probabilities. Only those pseudo-labels that are consistently agreed upon across the ensemble are selected. This method reduces the chance of incorporating noisy or overconfident predictions and enhances the robustness of the label propagation process.

Co-Training Method in Semi-Supervised Learning

Co-Training is a classic semi-supervised learning technique that leverages the idea of training two separate classifiers on two distinct and ideally independent *views* (feature sets) of the data. It is based on the intuition that different subsets of features might provide complementary information for classification, and that one classifier can assist the other by sharing high-confidence pseudo-labels. Co-Training is especially effective when such views are conditionally independent given the class label and are each sufficient for learning.

Co-Training Algorithm Steps

Step 1: Split Data into Two Views

The first step in co-training is to identify or engineer two different feature sets (called **views**) for each input example. These views should ideally be conditionally independent and provide sufficient information on their own to predict the target label. For example, in a web page classification task, one view might consist of the words on the page, while another could consist of the anchor text in links pointing to the page.

Step 2: Train One Classifier Per View

A separate classifier is trained on each view using the available labeled data. Let C_1 and C_2 denote the classifiers trained on view-1 and view-2, respectively. Each classifier learns to make predictions based solely on its assigned feature set.

Step 3: Predict on Unlabeled Data

Both classifiers are then used to predict labels for the unlabeled instances. Each classifier processes its own view of the unlabeled data and assigns labels along with associated confidence scores.

Step 4: Select Top-k Confident Predictions

From their predictions, each classifier selects the **top-k** unlabeled examples for which it is most confident about the predicted labels. Confidence may be measured using posterior probabilities, margins, or other classifier-specific scores.

Step 5: Exchange Pseudo-Labels Between Classifiers

Each classifier then *shares* its selected pseudo-labeled examples with the other classifier. That is, C_1 teaches C_2 using its confident predictions, and vice versa. The assumption is that each view provides information not available in the other, thereby allowing both classifiers to generalize better.

Step 6: Retrain Classifiers

Each classifier augments its labeled training set with the new pseudo-labeled examples received from its peer. Using the expanded labeled dataset, the classifiers are retrained to improve performance and update their decision boundaries.

Step 7: Iterate

The process of prediction, pseudo-label exchange, and retraining is repeated over several iterations. With each cycle, both classifiers ideally become stronger as they benefit from each other's confident knowledge over unlabeled examples.

Challenges and Limitations of Co-Training

1. Requires Independent and Sufficient Views

A fundamental assumption behind co-training is that the two feature sets (views) are both **conditionally independent given the label** and individually sufficient for learning the target concept. This assumption is often difficult to satisfy in real-world applications, where feature sets might be correlated or one view might dominate in informativeness. If this assumption fails, the classifiers may reinforce shared biases rather than providing complementary learning signals.

2. Sensitivity to Early Pseudo-Label Noise

Like other self-labeling techniques, co-training is vulnerable to **early-stage errors**. If one classifier generates incorrect pseudo-labels in the early iterations, and these are shared with the second classifier, the mistakes may propagate. Since each classifier relies on the other for additional training data, early noise can significantly degrade the quality of both models, especially in the absence of sufficient labeled data to anchor learning.

3. Ineffectiveness with Highly Correlated Views

Co-training is less effective when the two views are not truly independent or informative on their own. If both classifiers are trained on redundant or highly correlated information, the supposed benefit of complementary learning disappears. Instead of correcting each other's mistakes, both classifiers may end up amplifying shared weaknesses or learning nearly identical patterns, which limits the effectiveness of the method.

Key Takeaways

- **Smoothness Assumption**: Nearby points in input space should have similar labels.
- **Cluster Assumption**: Data points naturally form clusters, and decision boundaries lie between them.
- **Manifold Assumption**: Data lies on lower-dimensional manifolds embedded in high-dimensional space.
- **Inductive vs. Transductive Learning**: Inductive generalizes to unseen data, while transductive infers labels for specific unlabeled data.
- **Self-Training**: A model iteratively labels unlabeled data and retrains using confident pseudo-labels.
- **Co-Training**: Two classifiers learn from each other using complementary feature views and confident pseudo-labels.