Contrastive-Inspired Semi-Supervised Learning

This repo is intended to to serve as a framework for comparing loss functions in a semi-supervised learning context.

Sources To Read

Papers

- MixMatch: A Holistic Approach to Semi-Supervised Learning
- A Survey on Semi-Supervised Learning
- Contrastive Representation Learning: A Framework and Review
- Self-training with Noisy Student improves ImageNet classification

Wikipedia

- Wikipedia: Semi-supervised learning
- Wikipedia: K-means Clustering

Local Files

See the Files folder

Coding

See the main README

Takeaways

Disclaimer: this section is about reporting from sources as well as my own understanding and insights from the sources.
As such anything written below should only be accepted after some critical thought.

Wikipedia: Semi-Supervised Learning

Supervised Learning: Transductive learning - Generalization p(y|x)

Unsupervised Learning: Inductive learning - Inductions from patterns in p(x) to get p(y|x)

Assumptions

- Continuity Assumption: points close together are likely to share labels -> motivates putting boundaries
  in low density areas. Intuition: this is about probabilistic lipschitzness of $p(y|x)$ weighted on the density p(x)
- Cluster Assumption: Data tends to cluster. With continuity assumption, this means clusters share labels, which motivates
  feature learning by cluster.

- Manifold Assumption: Most of the dimensions are redundant. All useful information can be projected onto a lower-dimensional
  manifold. Assumption motivated by the scientific method: learning from observation is to reduce the complexity of the theory.

Methods

- Generative Models: model all of the information. These methods focus on finding $p(x|y)$. Once it is known, $p(y|x)$ is then
  inferred using Bayes' rule: $p(y|x) = p(x|y)p(y)/p(x)$. Philosophy of this method: guess what the elements of set looked like before
  it got mixed with other sets (ex: in the task of separating cats from dogs, focus on what cats look like, and what dog looks like).
  Each of Sup, Unsupervised learning finds some of the information related to $p(x|y)$.
  Generative modeling finds more information then either sup or unsupervised learning alone.
  Unsupervised Learning learns about p(x), and then (with help of labels) tries to decompose the function $p(x) = \sum_i p(x|y_i)p(y_i)$.
  Supervised Learning finds $\max(p(y|x))$ directly-by learning a hypothesis function $h(x) = y$, and then would have to additionally find measures
  of confidence in order to model $p(x|y)$.
  Such models learn $p(x|y, \theta)$, and make assumptions about which $p(x|y)$ are possible (choice of a hypothesis class).
  Wrong assumptions can hurt(negative impact! not neutral. source: Gweon et al, 2010, "Infants consider both the sample and the sampling process in inductive generalization").
  Good assumptions necessarily helps performance (source: Younger B. A. (1999) "Parsing Items into Separate Categories: Developmental Change in Infant Categorization").
  Example of generative model: Gaussian mixture distribution.
  Used by fitting: Kaylee divergence on both the joint distribution and the unlabeled distribution
  $argmax_\theta(\sum_i \log p(x_i, y_i|theta) + \lambda \sum_i \log p(\{x_i\}|\theta))$
- Low-density Separation:
  Idea: Place boundaries in low density regions. Ex: TSVM transductive support vector machine. SVM's in unsupervised learning: maximize margins,
  but indiscriminately of labels. Hinge loss: $(1 - yf(x))$. Unsupervised equivalent: $(1 - |f(x)|)$. Loss is made with sum of both losses with weighting λ_1, with a λ_2-weighted normalization on the RKHS norm of h, for f(x)= h(x) + b. Other processes for low-density estimation exists.
- Graph-Base methods:
  Idea: build a graph by proximity in x, and then model low-dim manifold by fitting the function f(x) by optimizing smoothness of f to the manifold. The smoothness of manifold is optimized, and smoothness over input space as well

Wikipedia: k-means

Idea

We have a number k of centers, each associated with a partition of the data. Minimize sum of Euclidian distance squared from nodes in a set to the center. For any given set, the point that does this is the mean. Therefore, minimize the partitioning of fixed size that minimizes distance squared with the mean of the subset. This is equivalent to minimizing the normalized sum of distance squared for every pairs of elements in the set. This equivalence is proved through the factored-out expression for C_l the partition elements,
and $\mu_l = \frac{1}{|C_l|} \sum_{x \in C_l} x$ the mean of the partition element.
$\sum_{x \in C_l} ||x||^2 - |C_l|||\mu_l||^2$

Proof: proof (link might not work, file placement dependent)
Points:

- Non-convex function, so no guarantee of finding global max
- Choice of k crucial and difficult. For this, one can try out many or use external information (ex: numbers of classes)
  However,# of classes is not always the best choice, if classes are not clustered well. The best choice of k is the number of distinct clusters.
- Can be generalized to gaussian mixtures. K means is Gaussian mixtures for unit diagonal gaussians.
- Features can be learned by defining a feature as the nearest center of a point.

Cluster Energy Notes

- Inspired from Mido's contrastive learning paper. The idea of the paper is to combine contrastive learning on labeled
  and unlabeled data. This is done through cosine similarity between feature representations.
- unsupervised: data to learn features with similarity between elements and their data augmented counterparts.
- supervised: optimize similarity between elements of the same label.

Once this pre-training is done, the network is trained with the labeled data only.

Principle is to learn a representation feature vector with an appropriate metric
between them. It is also important that the feature vectors retain information about the input space.

- Question: Why is cosine similarity used?

Cluster Energy Notes Takeaway

- multiple views: Data augmentation -> unsupervised similarity
- using known labels: For Mido, labels are used twice: to learn distances and to learn labeling function
- cluster energy: Absent from Mido's paper! new input.\

First Idea: embed all labeled elements to fixed "target" vectors.
ISSUE: We cannot learn then! The function from elements to labeled centers is the trivial realizable hypothesis.

Fix the center vectors c_i corresponding to the labels (can be the basis vectors).

- Minimize d(x, C(x)) where C(x) is the center nearest to x (for an unsupervised sample)
  ISSUE: contrary to k-means, minimizing this does not pitch distance of elements against each other.
  When centroid goes closer to other elements, it leaves others behind (compromise between samples).
  This is not the case here as the gradient in distance with different points is independent (modulo complexity of the hypothesis class).
  Result: The gradients will be determined by initial config. C(x) will (almost) never change for training,
  not learning clusters, but taking the initial clustering as true and as needed to be enforced in features.

Remark: key difference between Mido and this: this algorithm attempts to learn the entire problem (Pre-training and fine-tuning)
Whereas position in Mido's feature learning is not decided (definition of pre-training to some extent)

Mido:
learn local metric. Then transform this space to the answers(logit space)

Cluster Energy Notes:
Try to speed up pre-training by using a target
Design a pre-training loss using some combination of:

- $d(x, c_i)$: distance from labeled (x,y) where c_i is the one-hot vector corresponding to the label y of x. This is the training that is used during the fine-tuning procedure in Mido's paper.
- $d(x^+, c_i)$: distance from augmented sample (x^+, y) to the one-hot vector c_i corresponding to the label y. This is standard supervised learning with data augmentation.
- $d(u, C(u))$: Train unlabeled data to get closer to their nearest center. This trains samples to cluster in an arbitrary way, where the center of clusters is the nearest center at initialization for each label. This is done in Mixmatch by sharpening the SoftMax output(aka increasing the temperature). This operation is a softer version than the clustering training proposed by $d(u, C(u))$. The motivation behind this training is learning to cluster is one step towards learning to cluster correctly. In MixMatch they mention that this has to do with setting the boundaries in low-density regions of $p(x)$, although it should be noted that this process starts by setting the
boundaries first, and then enforcing low densities in the region of these boundaries. Nonetheless, Mixmatch uses it, so it is probably empirically shown to help unsupervised learning.
- $d(u, q)$: Define q as the averaged, sharpened target as the mean of the output of u and all its augmentations. This term is only from Mixmatch.
- $d(u, u^+)$: learn that the distance in the feature space between an image and one of its augmentations should be small. Intuition: the relevant features in an image and one of its transformations are the same. This term has to do with learning a metric in feature space. Note: here u can be a supervised or an unsupervised sample.
- $d(x_1, x_2)$: For two labeled samples x_1, x_2 for which y_1, y_2 have the same label, their feature vectors should be close to each other. The intuition here is that two elements of the same class should share some set of features.
- $d(u^+, C(u))$: learn a random but consistent target for the augmentations of u. A similar term appears in MixMatch, but instead of $C(u)$ they take the average predictions of u and its augmentations at initialization, which is then temperature-sharpened (aka softly projected to the nearest center). Note: the effect of this term should resemble(to which degree? don't know) the combined effect of $d(u, C(u))$ and $d(u, u^+)$ by the triangle inequality. This can be seen also by the fact that if $d(u, u^+)$ is small enough, then they will be mapped to the same target. However, $d(u^+, C(u))$ being small intuitively does not imply the former, since two distinct feature vectors can be mapped to the same target.

Categorization

Contrastive vs target based

We wish to categorize the terms described above. The primary categories are
contrastive vs target-based
this is the same distinction as pre-training vs fine-tuning in Mido's paper, roughly. With this categorization comes also questions of order: should some loss terms be used first and then

dropped.

- contrastive: $\{d(u, u^+), d(x_1, x_2)\}$
  These all attempt to learn a feature space that satisfies constraints on the metric in that feature space. Elements that should be close are trained to be close. Here "should be close" is inferred from either shared labels (d(x_1, x_2)) or shared original image (with different representations):d(u, u^+).

- target based: $\{d(x, c_i), d(x^+, c_i), d(u, C(u)), d(u^+, C(u)), d(u, q)\}$
  All the terms that train an sample to be mapped to a target. This is by nature much closer to supervised learning as the process is the same. The difference with supervised learning lies in how the "sample" and "target" are obtained.

Categorization by Data type

This would be sub-categories for the Contrastive-Target categories.
Data in the input space can be categorized as augmented or not, and labeled or not. Augmented Data can be done by exploiting a known symmetry that is label-preserving, or convexity in the input space (MixUp).

The targets can also be augmented: Usually this is done by assigning some label to unlabeled data. These synthetic targets are the initial outputs of the unlabeled data that are transformed in some way. They can be projected to the closest label center, or fitted to the sharpened initial distribution, or it can be determined with an average of related samples (as in the variable q in MixUP).