Machine Learning Course - CS-433

Unsupervised Learning

Nov 13, 2018

minor changes by Martin Jaggi 2016

changes by Martin Jaggi 2017

changes by Ruediger Urbanke 2018

Last updated: November 13, 2018

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Unsupervised learning

So far we have assumed that we are given a training set of the form $S_{\text{train}} = \{(\mathbf{x}, y_n)\}_{n=1}^N$ and we are then asked to predict the label y associated to any new feature vector $\mathbf{w}$. This is called *supervised* learning since we are given examples of features *and* labels and are asked to learn from this.
There is a second very important framework in ML called *unsupervised* learning. Here the training set consists exclusively of feature vectors, i.e., $S_{\text{train}} = \{(\mathbf{x})\}_{n=1}^N$. We are asked to learn from the set of features alone without having access to training labels. Unsupervised learning seems to play an important role in how living beings learn.
How can systems learn meaningful information given just the feature vectors?
Two main directions in unsupervised learning are

- representation learning & feature learning and
- density estimation & generative models

These concepts are best explained by means of explicit examples. We will discuss many of these concepts in much more detail later. The idea of this lecture is to take a bird's eye view and to quickly survey some of the existing techniques. This is similar to the overview that we did before talking about specific classification schemes.

Representation Learning & Feature Learning

In feature learning the task is to automatically learn the most important features given a data set. Below is a list of techniques that belong into this category.

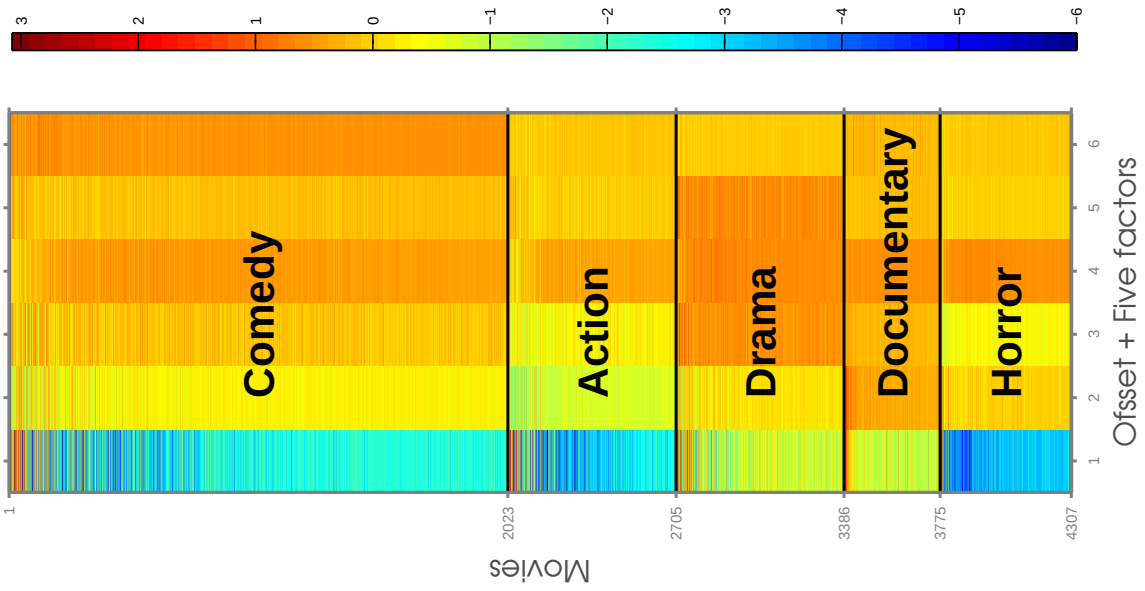
1. **Matrix Factorization:** Matrix factorization can be used both in a supervised fashion as well as an unsupervised fashion. We give one example for each case.

   a) Netflix, collaborative filtering: This is an application of matrix factorization in the supervised case. Assume that we are given ratings of movies by viewers. This can be conveniently represented in matrix form, where movies are lets say represented by columns and users by rows. In each row we write the rating (lets say a number between 1 to 5).

   Assume now that we can approximate the matrix reasonably well by a matrix of rank one, i.e., an outer product of two vectors. Then implicitly this extracts useful features both for the users and also the movies. E.g., it might implicitly group the movies by "genres" and the users by "types."

   Of course, we can get better approximations by looking at matrix factorizations with higher rank.

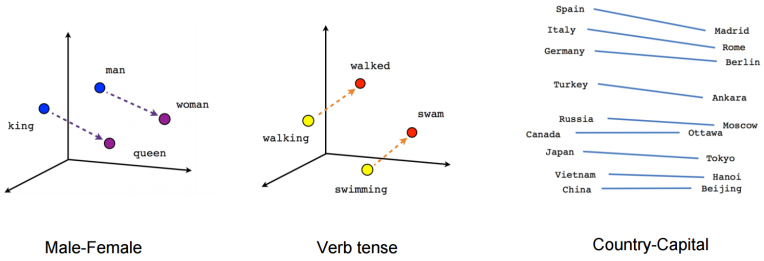
   b) **word2vec:** One of the most frequent type of data is text. But text itself is hard to manipulate



if we want to do learning. Can we represent text numerically? This is the aim of word2vec.

The idea is to map every word w_i into a vector $\mathbf{w}_i \in \mathbb{R}^K$, where K is relatively large. Is it possible to do this mapping so that the semantics of the word is captured by this representation? (Mikolov et al. 2013).

The answer is yes. To get this representation, we start with a large corpus of text. We then build a matrix (whose dimensions are equal to the number of words in our dictionary). The entries in this matrix are the number of times word pairs appear together in each "context." We then factorize this matrix by means of two matrices of rank K to give us the desired representation. The results are quite astonishing as examples below show.



Male-Female Verb tense Country-Capital

2. **Principle Component Analysis and Singluar Value Decomposition:** Recall that features are typically vectors in $\mathbb{R}^d$ for some d. Assume that we want to "compress" the feature vectors, e.g., we want to only keep one number per input. What is the direction onto which we should project our input vectors so that the projected data preserves as much about the original input as possible? One way to formulate this is to ask that the variance of the projected data is as large as possible. We can ask the same question when we are allowed to keep two or more numbers per feature vectors. This view point will lead us to discuss what is called the principal component analysis and the technical tool that we will use is the singular value decomposition.

    a) Given voting patterns of regions across Switzerland, we use PCA to extract useful features (Etter et al. 2014).

    b) Genes mirror geography within Europe. In this experiment the featue vector consists of several numerical values related to the genome of various populations in Europe. Then a SVD is applied and only the first two components are kept. If the features are then projected onto these components

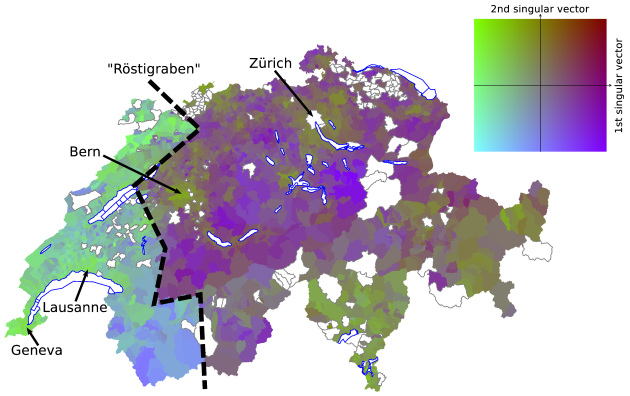
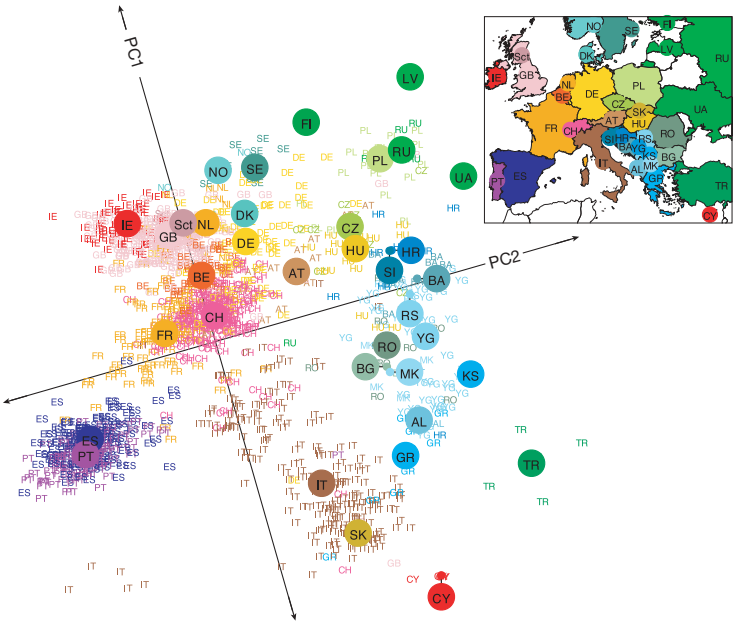


Figure 9: Voting patterns of Swiss municipalities. The color of a municipality is assigned using its location in Figure 8 and the color gradient shown in the upper right corner. Two municipalities with similar colors have similar voting patterns. The *Röstigraben*, corresponding to the cultural difference between French-speaking municipalities and German-speaking ones, is clearly visible from the difference in voting patterns. Regions shown in white are lakes or municipalities for which some vote results are missing (due to a merging of municipalities, for example). A more detailed map can be found online [2].

and properly displayed the geography of Europe is found back!



Nature 2008,
http://dx.doi.org/10.1038/nature07331

The above two examples are quite amazing. But caution is in order. In the above examples there were quite a few degrees of freedom how the projection was done, and the result most likely did not come about completely "randomly." Further, the singular value decomposition is *not* invariant to scaling in the data

and hence there is a considerable degree of arbitrariness inherent in this procedure.

3. **Clustering:** One way to reveal structure in the input data is to cluster the feature points given some similarity measure between inputs (e.g., like Euclidean distance). There is considerable degree of freedom we have: Besides the choice of similarity measure, we also have a choice of how many clusters we want. In addition we can ask clusters to be hierarchically.

   a) **Hirarchical Clustering:** Our first example shows a hirarchical clustering of types of cancer. Such a clustering can e.g. hint at an evolutionary process and make it clear which types are similar or not. Clearly, this can be useful when it comes e.g. to possibly treatments.

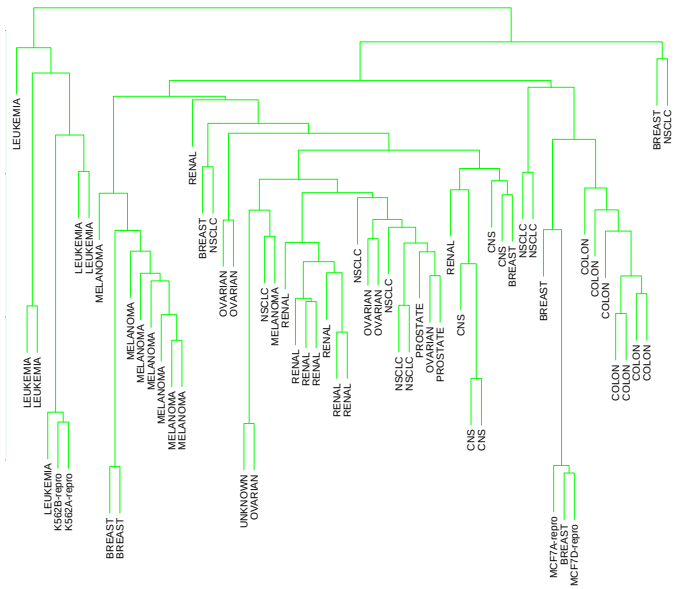


FIGURE 14.12. *Dendrogram from agglomerative hierarchical clustering with average linkage to the human tumor microarray data.*

   b) **Topic Model:** A somewhat naive but very useful model for documents is the so-called topic model. The idea is the following. It is assumed that each

documents consists of a collection of "topics." A topic here is something very simple. It is just a probability distribution on the words of a dictionary. E.g., we completely ignore any grammatical considerations and assume that if we are talking about a particular topic we simply sample words from a dictionary according to a particular probability distribution. E.g., if we talk about tennis, we might frequently use the words, "Federer", "set", "tie break", etc. If we are now given a large corpus of documents we can try to identify various "topics", i.e., probability distributions. And if we want to consider only a small number of topics out of all possible topics we might have discovered we can use clustering to pick out some representative examples.

Clustering articles published in Science (Blei & Lafferty 2007)

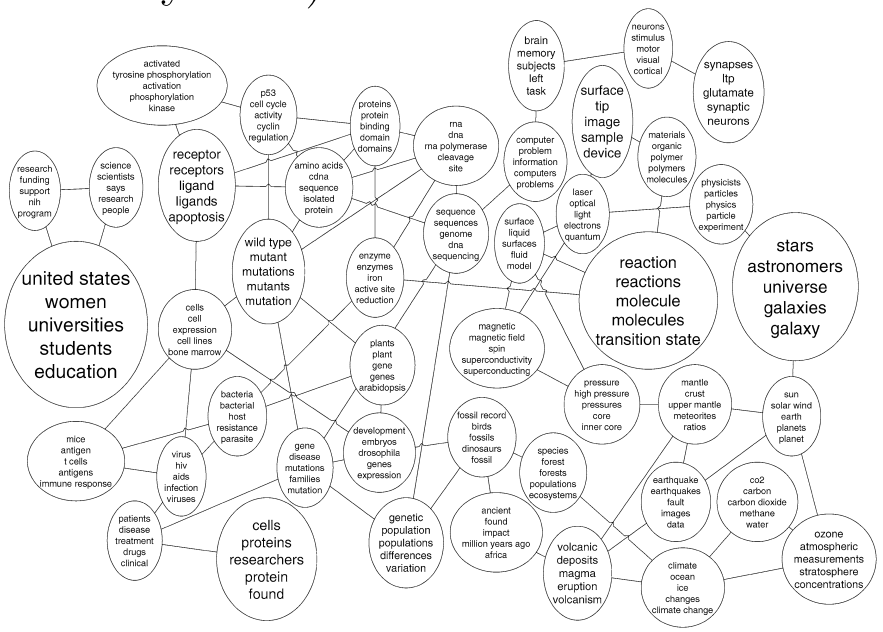


FIG. 2. *A portion of the topic graph learned from* 16,351 *OCR articles from Science* (1990–1999). *Each topic node is labeled with its five most probable phrases and has font proportional to its popularity in the corpus.* (*Phrases are found by permutation test.*) *The full model can be found in http://www.cs.cmu.edu/~lemur/science/ and on STATLIB.*

Generative Models: In supervised learning, a generative model, given some data set $S_{\text{train}} = \{(\mathbf{x}_n, y_n)\}$, is a model for the joint distribution of $\mathbf{x}$ and y. But of course, we can also model just the distribution of $\mathbf{x}$.

1. Auto-Encoders: Auto-encoders were invented in the framework of neural nets. But the basic idea can also be applied in other areas. Auto-encoders are a form of a compression algorithm. And we can think of them as a generative model but we can also think of them as a feature learning algorithm.

   Assume that we have a function $f_{\mathbf{w}}$ parametrized by $\mathbf{w}$ that maps a given feature vector $\mathbf{x}$ to a much "smaller" feature vector $\tilde{\mathbf{x}}$ and in addition a second function $g_{\mathbf{w}'}$ that acts like an *inverse map* in the sense that for our given data $(\mathbf{x} - g_{\mathbf{w}'}(f_{\mathbf{w}}(\mathbf{x}))^2$ is small in average.

   Assume at first that there is no error. Then $\tilde{\mathbf{x}}$ is as useful for a ML task as $\mathbf{x}$ since afterall we can always recover $\mathbf{x}$ from $\tilde{\mathbf{w}}$ by applying the map $g_{\mathbf{w}'}$. But by assumption $\tilde{\mathbf{x}}$ is small. In this sense we have extracted from the original feature vector a smaller feature vector that is as good as the original one irrespective of the task to be learned – we have performed featue selection. Of course, in any real application we will have a small error between $\tilde{\mathbf{w}}$ and its reconstruction $g_{\mathbf{w}'}(\tilde{\mathbf{x}})$. But if this error is small then we can hope not to have lost any essential information. The advantage of this approach is that often we have lots of unlabeled data but only very little labeled data. So we can first apply

this approach for the feature selection and then run the learning algorithm on the small feature set.

We can also think of this as a generative model for $\mathbf{x}$. If we truly compressed $\mathbf{x}$ then $\tilde{\mathbf{x}}$ should follow a simple distribution (e.g., if $\tilde{\mathbf{w}}$ is a vector of bits, then it should follow an iid distribution with uniform bits. Otherwise we could compress further.

The above description has be be viewed with some grain of salt. If the vectors $\mathbf{x}$ and $\tilde{\mathbf{x}}$ take on real values then the notion of "compression" is not so clear. But the above explanation still gives a good intuition why autoencoders are useful.

2. Generative Adversarial Networks (GANs)
   GAN's are a fairly recent invention, also again in the framework of neural nets. The idea is to use two neural nets, one that tries to generate samples $\mathbf{x}$ that look like the data we get, the other one that tries to distinguish real samples from artificially generated samples. These two networks are training alternatively. The aim is so that after sufficient training even a good classifier cannot distinguish real samples from artificially generated ones. If we can achieve this, then this means that we have built a good model for the set of feature vectors we were given.

   If we apply this e.g. to photos of rooms, we can then generate many artificial rooms that might even fool a human observer.

"Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Network", ICLR 2016, https://arxiv.org/abs/1511.06434