Machine Learning

Machine learning is a field in computer science, mathematics and statistics, with the goal of programming computers to “learn” from data, usually using statistical inference and big datasets. Finding significant patterns in a messy landscape of all possible patterns requires sophisticated optimization techniques. If you are interested in this field I recommend learning by hands-on exploring and experimenting, for which the `sklearn` module and documentation at scikit-learn.org are especially well-suited, as well as taking classes in linear algebra, optimization, statistics... to understand the theoretical underpinnings. It is risky to use these techniques without understanding the theory, because they will always give you an answer, without telling you how to interpret it or how much of it is noise. However, with caution, there are many powerful techniques that are easy to implement using `sklearn`, and data analysis is an important application where python shines (and is free!). So, let’s dive right in. For this class (which is of course not about the theory of machine learning) we will take a look at three machine learning techniques and some of the theory, so that you have a starting point. You will be analysing your own data, which does not need to be big. Python and `numpy` have features to help handle large datasets, such as sparse matrices, but we won’t focus on that in this course.

Non-negative Matrix Factorization

Suppose that we have a matrix that represents users and movie ratings:

$$M = \begin{array}{c} \\ u0 \\ u1 \\ u2 \\ u3 \\ u4 \end{array} \begin{array}{c} \begin{array}{cccc} m0 & m1 & m2 & m3 \end{array} \\ \begin{pmatrix} 1 & 3 & 1 & 2 \\ 5 & 5 & 1 & 1 \\ 1 & 2 & 5 & 4 \\ 4 & 3 & 1 & 1 \\ 5 & 4 & 2 & 2 \end{pmatrix} \end{array}$$

The goal of NMF is to approximately factorize this non-negative $n \times m$ matrix into two smaller matrices; of size $n \times r$ and $r \times m$, respectively. The underlying idea is that we can describe our data in terms of r “features”, such that each user and each movie is described as a (non-negative) linear combination of the features, and each rating as the dot product of the specific user and movie combination. For illustration, movie genres could be features. Suppose that user A likes comedies a lot and thrillers somewhat, and user B does not like comedies but loves thrillers. If movie C is mostly a comedy with some thriller elements, then user A will like this movie a lot more than user B. Their ratings might look like $\begin{pmatrix} 2 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \end{pmatrix} = 5$ and $\begin{pmatrix} 0 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \end{pmatrix} = 2$, respectively. (Up to some constant factor.) Usually, the features are unlikely to coincide exactly with movie or music genres. It is almost always impossible to write M as a factor of two such matrices, because this is a simplification (and potentially noise reduction) of the data. Therefore, we find two matrices such that the product is “as close as possible” to the original matrix. This is done by minimizing the L2-norm (square root of the sum of squares of all the elements) of the difference matrix. In `sklearn` we implement this as follows (next page). We can see, just from inspecting the matrix $W \times H$, that this is a good approximation of the orginal matrix M. This method is extremely powerful, because we can use only partial data from M to approximate W and H. For example, we do not need to use all of the songs (columns) to approximate W, or all of the users to approximate H. Intuitively, you can guess quite well what kind of movie fan somebody is (and predict how much they will like new movies) by knowing how they have rated some number of past movies. You do not need to know how they have rated all movies in the history of film making.

```
import numpy as np
from sklearn.decomposition import NMF
M=[[1,3,1,2],[5,5,1,1],[1,2,5,4],[4,3,1,1],[5,4,2,2]]
model = NMF(n_components=2)
model.fit(M)
W = model.fit_transform(M)
H = model.components_
W
Out:   array([[ 0.73187392,  0.75167863],
          [ 2.09408547,  0.        ],
          [ 0.34119281,  2.62041194],
          [ 1.47426902,  0.12132919],
          [ 1.84565285,  0.60341817]])
H
Out:   array([[  2.51186087e+00,   2.23168317e+00,   4.53271270e-01,
          5.61520045e-01],
          [  1.37765499e-05,   5.33379161e-01,   1.78497633e+00,
          1.50778417e+00]])
np.matmul(W,H)
Out:   array([[ 1.83837581,  2.03424042,  1.67346599,  1.54433101],
          [ 5.26005135,  4.6733353 ,  0.94918878,  1.17587097],
          [ 0.85706498,  2.15910738,  4.8320262 ,  4.14260223],
          [ 3.70316033,  3.35481582,  0.88481352,  1.01076984],
          [ 4.63603147,  4.44076307,  1.91366857,  1.94619543]])
```