princeton univ. Spr'15 cos 598D: Overcoming intractability in machine learning

Lecture 2: Linearized models and SVD

Lecturer: *Sanjeev Arora* Scribe:

1 Concluding thoughts about last time and segue to today's topics

The dominant model for supervised learning from a theory viewpoint is SVM or Kernel SVM. We discussed SVM's last time; just a linear classifier.

A kernel is a mapping from datapoint x to a point $\phi(x)$ in a higher-dimensional space. Example: map $(x_1, x_2, \ldots, x_n)$ to the n^3-dimensional vector $(x_{i_1}x_{i_2}x_{i_3})$. The coordinates of $\phi(x)$ can be seen as *features*.

What makes kernel SVMs reasonably practical are the following two facts: (i) the standard algorithm for fitting SVM's only needs the ability to compute *inner products* of pairs of data points. Thus they also work for kernel SVMs if the kernel is such that $\langle \phi(x), \phi(y) \rangle$ is easy to compute given x, y. This is true for all the popular kernels. Thus in the above example, there is no need to work with the explicit n^3 size representation. (ii) the running time and the sample complexity of the algorithm is proportional to $1/\lambda$, where λ is the *margin* between the 0 examples and the 1 examples. Thus the running time can be small even if the kernel implicitly is mapping to a very high dimensional space.

The theoretical justification for kernel SVMs is the *margin hypothesis*: For every classification task there is a reasonable kernel for it that also has large margin (hence, can be fitted efficiently)

The lure of kernel SVMs is that they promise to fold in the *feature learning* with the classification task. Unfortunately, in practice people need to use more explicit ways of learning features, which gives a new representation for the data and then one can fit a classifier on top.

This brings us to today's topic, which is feature learning. Just as with classfication, this can be done in a *generative* and *nongenerative* setting.

Example 1 In the k-means problem one is given points $x_1, x_2, \ldots, \in \Re^d$ and one is trying to find k points (called *means*) $c_1, c_2, \ldots, c_k$ so as to minimize

$$\sum_i |x_i - p_i|^2,$$

where p_i is the closest mean to x_i.

After learning such means, each point has been labeled from 1 to k, corresponding to which mean it is closest to.

The generative analog of this would be say mixture of k gaussians. Each datapoint could be labeled with a k-tuple, describing its probability of being generated from each of the gaussians.

The nongenerative feature learning problems (such as k-means) are often NP-hard, whereas the generative version seems potentially easier (being average case). So I am attracted to the generative setting.

2 Linear Algebra++

The mathematical problems most directly useful to feature learning have to do with extensions of linear algebra. Recall that your freshman linear algebra class consisted of the following three main methods. (a) Solving linear equations. $Ax = b$. (b) Computing rank, which we can also think of as *matrix factorization:* Given $n \times m$ matrix M rewrite it as $M = AB$ where A is $n \times r$, B is $r \times m$ and r is as small as possible. (c) Eigenvalues/eigenvectors and singular values/singular vectors. For instance, every symmetric matrix M can be rewritten as

$$\sum_i \lambda_i u_i u_i^T,$$

where u_i's are eigenvectors and λ_i's are eigenvalues. This is called the *spectral decomposition* (if the matrix is not symmetric, the analogous expression is called *Singular Value Decomposition*).

Linear Algebra++ is my name for the above problems with any subset of the following extensions: (i) require some coordinates of the solution to be nonnegative. (ii) require a solution with a specified number or pattern of nonzeroes. (iii) solve in the presence of some kind of noise.

EXAMPLE 2 If we have to solve $Ax = b$ subject to x being nonnegative, then that is tantamount to linear programming, which was only discovered to be solvable in poly time in 1979.

If we have to solve $Ax = b$ subject to x having only k nonzeroes then this is the *sparse recovery problem* that is NP-hard.

If we have to solve $M = AB$ in presence of coordinate-wise noise, we may be looking for desired A, B such that we minimize

$$\sum_{ij} |M_{ij} - (AB)_{ij}|^2.$$

This is minimized by truncating the SVD to the first r terms. We will denote the best rank k approximation to M by M_k.

You never saw most of these extensions in your freshman linear algebra because they are NP-hard. But they are ubiquitous in ML settings.

3 Linearized models

The notion of features is often in some linearized setting: topic models, sparse coding, sparse recovery. We will see these in later lectures.

Rest of the lecture covered sparse recovery (Moitra's lecture notes); SVD (my lecture notes from COS 521) and use of SVD for clustering (Hopcroft-Kannan book).