2018-06-19

1 Introduction

Last week, we considered least squares and related regression models. In these models, we want to predict some dependent variable Y (also called an outcome variable) as a function of some independent variables X (also called features, regressors, or explanatory variables). In the simplest case, we look for a linear prediction:

$$Y \approx Xw$$

where X is a row vector of features and w is a column vector of weights. We can make the model a little more expressive with a linear prediction in a higher-dimensional space:

$$Y \approx \psi(X)w$$

where ψ is a (nonlinear) *feature map* from the original vector of dependent variables to a new, expanded set of variables[1]. In either case, we choose w by minimizing a sum of loss functions over examples drawn from a population, maybe together with a regularization term. For the least squares loss function and some standard regularizers, we can solve the resulting optimization problem using factorization methods from numerical linear algebra.

In some cases, though, the distinction between explanatory and output variables is not clear. Sometimes we may know different subsets of the variables for each experiment, and we want to fill in the rest. Or we may know all the variables in our experiment, and want to look for relationships between them rather than trying to make predictions. Or we might want to use the attributes associated with different objects not for prediction, but to cluster the objects, or to find outliers. Remarkably, we can view these tasks as well through the lens of matrix factorization.

[1] We did not talk about feature maps yet in this class. But many of you will have seen feature maps and "the kernel trick" in a machine learning class, and you are implicitly using feature maps in the polynomial fitting homework problem from last week. We will discuss these ideas in more detail next week when we discuss function approximation.

2 Matrix factorizations and latent variables

We can use matrices to encode many types of data: images, graphs, user preferences, and distributions of jointly distributed random variables are but a few. Often, as in our linear regression examples, the rows of the matrix represent objects or experiments and the columns represent associated attributes. In other cases, as when encoding a graph, both rows and columns represent objects, and the entries of the matrix represent pairwise interactions. Factoring these data matrices can help us compress and denoise data, separate effects of different factors, find similarities between objects, or fill in missing data.

We generally seek to approximately factor a data matrix $A \in \mathbb{R}^{m \times n}$ as

$$A \approx LMR, \quad L \in \mathbb{R}^{m \times r}, M \in \mathbb{R}^{r \times r}, R \in \mathbb{R}^{r \times n}.$$

In this view, we can think of R (or MR) as a map from the original attributes to a smaller set of *latent factors*. Different factorization methods differ both in the structural constraints on L, M, and R and on how the approximation is chosen.

It is worth being a little careful in how we think of these factorizations! In most of linear algebra, we are interested in a matrix as a representation of a linear map or a quadratic form with respect to some particular basis — but we have a rich set of bases we can choose. For data analysis, though, we may want to restrict the bases we consider for the sake of interpretability. For example, if we want to choose as our latent factors a subset of the original factors, or if we want to enforce non-negativity in the factors, we cannot consider arbitrary changes of basis. Because of this, some of the methods that we like best for interpretability are also the most difficult to compute, as they involve an optimization problem that is combinatorial rather than continuous in nature. We will see this issue repeatedly this week.

3 A gallery of examples

Before we turn to our first batch of factorization tools, let us first set the stage with some concrete example applications.

3.1 Document search and latent semantic analysis

The *vector space model* was one of the early triumphs in the field of information retrieval. In this model, documents are treated as "bags of words," and each document is represented as a vector of term frequencies, one for each word in the vocabulary[2]. There are several different ways that the term frequencies can be computed. We might use a binary indicator that says whether a term is present or absent; raw count information or relative frequency; or something affine or nonlinear (usually logarithmic) function of the frequency. We also usually scale by the *inverse document frequency*, which measures how common the term is across all documents. For example, a common choice is

$$\text{idf}(t, D) = \log \frac{N}{|\text{documents in } D \text{ containing } t|}.$$

The tf-idf matrix for term t and document d in corpus D is then

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D).$$

In *latent semantic indexing*, we approximate the tf-idf matrix by the truncated singular value decomposition, and use the result to compute a measure of query relevance. For example, suppose the tf-idf matrix was arranged so that each row represents a term, and each column represents a document. Let q be the tf-idf vector for the words in a query; for example, it might be a vector of all zeros except for a one in the row indicating the word "vehicle." To find documents relative to the query, we would compute the row vector

$$r = q^T U_k \Sigma_k V_k^T$$

and use r_i as the relevance score. If we used the full SVD, the relevance score would be exactly the row of the original tf-idf matrix associated with the term "vehicle," which might not include highly relevant documents in which the word "vehicle" never appears but words like "boat" or "truck" or "car" do appear. By using the SVD, we "blur out" the specific words and get more semantically meaningful results.

The same idea of latent semantic indexing (LSI) or latent semantic analyis (LSA) can apply in other settings as well. For example, similar ideas appear

[2]Very common words (stop words) and very rare words may be removed from the vocabulary before taking counts.

in bibliometrics, where one wants to find highly relevant papers. But there are also some real difficulties with LSI. One problem is that the nonlinear mapping from term counts to matrix entries is not always easy to justify, and an appropriate choice may require some experimentation. Another problem with LSI is that it is generally impossible to assign any real meaning to the factors.

3.2 k-means clustering as a matrix factorization

The k-means algorithm is a standard clustering method. Given m points in n-dimensional space (which we think of as the rows in an $m \times n$ matrix A), the k-means algorithm repeatedly updates a set of k representative vectors $r_1, \ldots, r_k$ as follows:

- Assign each point in the data set to a cluster based on the nearest representative vector (e.g. in Euclidean distance, though we could also look at angles).
- Recompute each representative vector as the mean of all the points in the cluster.

In matrix terms, the k-means algorithm computes the factorization

$$A \approx LR$$

where the rows of R are the representative vectors r_j and the rows of L indicate cluster membership; that is, L_{ij} is 1 if point i is in cluster j, and zero otherwise. More particularly, the k-means method is an example of an *alternating iteration*: first we optimize L while holding R fixed, then we optimize R while holding L fixed. The optimization minimizes the least squares error, and it generally converges to a local minimum in practice.

3.3 Eigenfaces, fisherfaces, and image analysis

The method of *eigenfaces* (or more generally *eigenimages*) has been used for image recognition and classification since it was developed by Sirovich and Kirby in 1980. The method essentially extracts a low-dimensional feature representation of images of (gray scale) faces. The "eigenfaces" are computed by principal component analysis on a collection of (gray scale, possibly low

resolution) face images, which are each laid out in the columns of a large matrix. Classification is done by mapping a new face into a low-dimensional space of eigenface features, then looking for the nearest neighbor in that space. As with latent semantic indexing, the method works in part because it captures identifying features while "blurring out" irrelevant fine details.

An alternative to eigenfaces is *fisherfaces*. Where eigenfaces are may be written in terms of the eigenvalue decomposition of a covariance matrix, fisherfaces come from Fisher's linear discriminant analysis (LDA) approach. Here we are interested in the largest eigenvales and vectors for the generalized problem

$$\Sigma_b w = \lambda \Sigma w$$

where Σ is the common covariance for each class of (faces) examples in the data set, and

$$\Sigma_b = \frac{1}{C} \sum_{j=1}^{C} (\mu_i - \mu)(\mu_i - \mu)$$

is the between-class variability matrix. Here the μ_i are the class means and μ is the mean of the class means. Unlike the eigenfaces technique, which is agnostic to class labels on the images, the fisherfaces approach tailors the choice of features to the classification problem at hand.

3.4 Collaborative filtering and the Netflix challenge

In 2009, Netflix awarded a $1M prize in a competition to beat the accuracy of their in-house *collaborative filtering* algorithm to predict how users would rate films. One of the key ideas in collaborative filtering is *matrix completion*. The ratings are given in a giant matrix in which rows correspond to users and columns correspond to movies. But as most users have not watched most movies, only a relatively small number of the matrix entries are known. The idea of matrix completion is to use those few entries to learn a low rank factorization that matches the data and can be used to predict the remaining entries. The intuition is that the low rank factorization represents a mapping of users and movies into a low-dimensional space that captures certain common attributes (e.g. how much action there is in the movie, or whether the tone is light or serious).

Somewhat remarkably, one can prove that this type of reconstruction is possible (and even reasonably straightforward to compute) under *incoherence assumptions* that we will discuss on Friday.

3.5 Anchor words and interpretable topic models

In latent semantic indexing, we used the SVD to compute a low-dimensional feature space to describe documents. However, that space is very difficult to interpret. We would ideally like to summarize documents in terms of their relation to meaningful topics; the same idea also applies to other collections, such as lists of movies or songs that we might want to characterize by genre. In *topic modeling*, we explain each document in terms of a distribution over of a few stopic, where each topic is in turn a (sparse) distribution over words. We generally insist that all these distribution vectors are properly stochastic: that is, the entries should be non-negative, and they should sum to one. Hence, topic modeling boils down to a *probabilistic matrix factorization problem*, a particular type of *non-negative matrix factorization* (NMF).

As we will see later in the week, non-negative matrix factorizations are generally hard to compute. However, the problem becomes much easier if we assume that for each topic there is at least one word that is mostly associated with that topic (and not with others). This word is called an *anchor word* for the topic. We can find anchor words by applying the pivoted QR algorithm, which we will turn to shortly, to a matrix of word-word co-occurrence statistics. Once we have the anchor words, we can compute word-topic distributions by solving *non-negative least squares* problems.

4 Some basic factorization tools

Next time, we will start our discussion of factorizations used in data analysis with the symmetric eigenvalue problem. This is a useful building block on its own, particularly for low rank approximation of symmetric matrices. It is also useful as a prelude to another discussion of the singular value decomposition (SVD), that Swiss Army knife of matrix factorizations. But both the symmetric eigenvalue decomposition and the singular value decomposition involve a very flexible choice of bases; as we have mentioned, this is not always ideal when we want an interpretable model. For interpretability, it is helpful to talk again about pivoted QR and the closely-related interpolative decomposition (ID), in which we choose a subset of the data matrix columns as a basis for the column space. We will also mention the closely-related CUR decomposition, in which both the left and right factors in our approximation are drawn from the columns and rows of the data matrix.