1
Introduction

Machine learning is starting to take over decision-making in many aspects of our life, including:

(a) keeping us safe on our daily commute in self-driving cars,
(b) making accurate diagnoses based on our symptoms and medical history,
(c) pricing and trading complex securities, and
(d) discovering new science, such as the genetic basis for various diseases.

But the startling truth is that these algorithms work without any sort of provable guarantees on their behavior. When they are faced with an optimization problem, do they actually find the best solution, or even a pretty good one? When they posit a probabilistic model, can they incorporate new evidence and sample from the true posterior distribution? Machine learning works amazingly well in practice, but that doesn't mean we understand *why* it works so well.

If you've taken traditional algorithms courses, the usual way you've been exposed to thinking about algorithms is through worst-case analysis. When you have a sorting algorithm, you measure its running time based on how many operations it takes on the worst possible input. That's a convenient type of bound to have, because it means you can say meaningful things about how long your algorithm takes without ever worrying about the types of inputs you usually give it.

But what makes analyzing machine learning algorithms, especially modern ones, so challenging is that the types of problems they are trying to solve really are *NP*-hard on worst-case inputs. When you cast the problem of finding the parameters that best fit your data as an optimization problem, there are instances where it is *NP*-hard to find a good fit. When you posit a probabilistic model and want to use it to perform inference, there are instances where that is *NP*-hard as well.

In this book, we will approach the problem of giving provable guarantees for machine learning by trying to find more realistic models for our data. In many applications, there are reasonable assumptions we can make, based on the context in which the problem came up, that can get us around these worst-case impediments and allow us to rigorously analyze heuristics that are used in practice, as well as design fundamentally new ways of solving some of the central, recurring problems in machine learning.

To take a step back, the idea of moving beyond worst-case analysis is an idea that is as old[1] as theoretical computer science itself [95]. In fact, there are many different flavors of what it means to understand the behavior of algorithms on "typical" instances, including:

(a) probabilistic models for your input, or even hybrid models that combine elements of worst-case and average-case analysis like semi-random models [38, 71] or smoothed analysis [39, 130];
(b) ways to measure the complexity of your problem and ask for algorithms that are fast on simple inputs, as in parameterized complexity [66]; and
(c) notions of stability that attempt to articulate what instances of your problem have meaningful answers and are the ones you actually want to solve [20, 32].

This is by no means an exhaustive list of topics or references. Regardless, in this book, we will approach machine learning problems armed with these sorts of insights about ways to get around intractability.

Ultimately, we hope that theoretical computer science and machine learning have a lot left to teach each other. Understanding why heuristics like expectation- maximization or gradient descent on a nonconvex function work so well in practice is a grand challenge for theoretical computer science. But to make progress on these questions, we need to understand what types of models and assumptions make sense in the context of machine learning. On the other hand, if we make progress on these hard problems and develop new insights about *why* heuristics work so well, we can hope to engineer them better. We can even hope to discover totally new ways to solve some of the important problems in machine learning, especially by leveraging modern tools in our algorithmic toolkit.

In this book, we will cover the following topics:

(a) Nonnegative matrix factorization
(b) Topic modeling

[1] After all, heuristics performing well on real life inputs are old as well (long predating modern machine learning), hence so is the need to explain them.

(c) Tensor decompositions
(d) Sparse recovery
(e) Sparse coding
(f) Learning mixtures models
(g) Matrix completion

I hope more chapters will be added in later versions as the field develops and makes new discoveries.

2

Nonnegative Matrix Factorization

In this chapter, we will explore the nonnegative matrix factorization problem. It will be helpful to first compare it to the more familiar singular value decomposition. In the worst case, the nonnegative matrix factorization problem is *NP*-hard (seriously, what else did you expect?), but we will make domain-specific assumptions (called *separability*) that will allow us to give provable algorithms for an important special case of it. We then apply our algorithms to the problem of learning the parameters of a topic model. This will be our first case study in how to not back down in the face of computational intractability, and to find ways around it.

2.1 Introduction

In order to better understand the motivations behind the nonnegative matrix factorization problem and why it is useful in applications, it will be helpful to first introduce the singular value decomposition and then compare the two. Eventually, we will apply both of these to text analysis later in this section.

The Singular Value Decomposition

The singular value decomposition (SVD) is one of the most useful tools in linear algebra. Given an $m \times n$ matrix M, its singular value decomposition is written as

$$M = U\Sigma V^T$$

where U and V are orthonormal and Σ is a rectangular matrix with nonzero entries only along the diagonal, and its entries are nonnegative. Alternatively, we can write

$$M = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

where u_i is the i^{th} column of U, v_i is the i^{th} column of V, and σ_i is the i^{th} diagonal entry of Σ. Throughout this section we will fix the convention that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. In this case, the rank of M is precisely r.

Throughout this book, we will have occasion to use this decomposition as well as the (perhaps more familiar) eigendecomposition. If M is an $n \times n$ matrix and is diagonalizable, its eigendecomposition is written as

$$M = PDP^{-1}$$

where D is diagonal. For now, the important facts to remember are:

(1) **Existence:** Every matrix has a singular value decomposition, even if it is rectangular. In contrast, a matrix must be square to have an eigendecomposition. Even then, not all square matrices can be diagonalized, but a sufficient condition under which M can be diagonalized is that all its eigenvalues are distinct.
(2) **Algorithms:** Both of these decompositions can be computed efficiently. The best general algorithms for computing the singular value decomposition run in time $O(mn^2)$ if $m \geq n$. There are also faster algorithms for sparse matrices. There are algorithms to compute an eigendecomposition in $O(n^3)$ time and there are further improvements based on fast matrix multiplication, although it is not clear whether such algorithms are as stable and practical.
(3) **Uniqueness:** The singular value decomposition is unique if and only if its singular values are distinct. Similarly, the eigendecomposition is unique if and only if its eigenvalues are distinct. In some cases, we will only need that the nonzero singular values/eigenvalues are distinct, because we can ignore the others.

Two Applications

Two of the most important properties of the singular value decomposition are that it can be used to find the best rank k approximation and that it can be used for dimension reduction. We explore these next. First, let's formalize what we mean by the best rank k approximation problem. One way to do this is to work with the Frobenius norm:

Definition 2.1.1 (Frobenius norm) $\|M\|_F = \sqrt{\sum_{i,j} M_{i,j}^2}$

It is easy to see that the Frobenius norm is invariant under rotations. For example, this follows by considering each of the columns of M separately as a vector. The square of the Frobenius norm of a matrix is the sum of squares of the norms of its columns. Then left-multiplying by an orthogonal matrix preserves the norm of each of its columns. An identical argument holds for right-multiplying by an orthogonal matrix (but working with the rows instead). This invariance allows us to give an alternative characterization of the Frobenius norm that is quite useful:

$$\|M\|_F = \|U^T MV\|_F = \|\Sigma\|_F = \sqrt{\sum \sigma_i^2}$$

The first equality is where all the action is happening and uses the rotational invariance property we established above.

Then the Eckart–Young theorem asserts that the best rank k approximation to some matrix M (in terms of Frobenius norm) is given by its truncated singular value decomposition:

Theorem 2.1.2 (Eckart–Young) $\operatorname{argmin}_{\text{rank}(B)\leq k} \|M - B\|_F = \sum_{i=1}^{k} \sigma_i u_i v_i^T$

Let M_k be the best rank k approximation. Then, from our alternative definition of the Frobenius norm, it is immediate that $\|M - M_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}$.

In fact, the same statement – that the best rank k approximation to M is its truncated singular value decomposition – holds for *any* norm that is invariant under rotations. As another application, consider the operator norm:

Definition 2.1.3 (operator norm) $\|M\| = \max_{\|x\|\leq 1} \|Mx\|$

It is easy to see that the operator norm is also invariant under rotations and, moreover, $\|M\| = \sigma_1$, again using the convention that σ_1 is the largest singular value. Then the Eckart–Young theorem with respect to the operator norm asserts:

Theorem 2.1.4 (Eckart–Young) $\operatorname{argmin}_{\text{rank}(B)\leq k} \|M - B\| = \sum_{i=1}^{k} \sigma_i u_i v_i^T$

Again, let M_k be the best rank k approximation. Then $\|M - M_k\| = \sigma_{k+1}$. As a quick check, if $k \geq r$ then $\sigma_{k+1} = 0$, and the best rank k approximation is exact and has no error (as it should). You should think of this as something you can do with any algorithm to compute the singular value decomposition of M – you can find the best rank k approximation to it with respect to any rotationally invariant norm. In fact, it is remarkable that the best rank k approximation in many different norms coincides! Moreover, the best rank k approximation to

M can be obtained directly from its best rank $k+1$ approximation. This is not always the case, as we will see in the next chapter when we work with tensors.

Next, we give an entirely different application of the singular value decomposition in the context of data analysis before we move on to applications of it in text analysis. Recall that M is an $m \times n$ matrix. We can think of it as defining a distribution on n-dimensional vectors, which we obtain from choosing one of its columns uniformly at random. Further suppose that $\mathbb{E}[x] = 0$; i.e., the columns sum to the all-zero vector. Let $\mathcal{P}_k$ be the space of all projections onto a k-dimensional subspace.

Theorem 2.1.5 $\operatorname{argmax}_{P \in \mathcal{P}_k} \mathbb{E}[\|Px\|^2] = \sum_{i=1}^{k} u_i u_i^T$

This is another basic theorem about the singular value decomposition, and from it we can readily compute the k-dimensional projection that maximizes the projected variance. This theorem is often invoked in visualization, where one can visualize high-dimensional vector data by projecting it to a more manageable lower-dimensional subspace.

Latent Semantic Indexing

Now that we have developed some of the intuition behind the singular value decomposition, we will see an application of it to text analysis. One of the central problems in this area (and one that we will return to many times) is this: given a large collection of documents, we want to extract some hidden *thematic* structure. Deerwester et al. [60] invented latent semantic indexing (LSI) for this purpose, and their approach was to apply the singular value decomposition to what is usually called the term-by-document matrix:

Definition 2.1.6 *The term-by-document matrix M is an $m \times n$ matrix where each row represents a word and each column represents a document where*

$$M_{i,j} = \frac{\text{count of word } i \text{ in document } j}{\text{total number of words in document } j}.$$

There are many popular normalization conventions, and here we have chosen to normalize the matrix so that each of its columns sums to one. In this way, we can interpret each document as a probability distribution on words. Also, in constructing the term-by-document matrix, we have ignored the order in which the words occur. This is called a *bag-of-words representation*, and the justification for it comes from a thought experiment. Suppose I were to give you the words contained in a document, but in a jumbled order. It should still be possible to determine what the document is about, and hence forgetting all

notions of syntax and grammar and representing a document as a vector loses some structure, but should preserve enough of the information to make many basic tasks in text analysis still possible.

Once our data is in vector form, we can make use of tools from linear algebra. How can we measure the similarities between two documents? The naive approach is to base our similarity measure on how many words they have in common. Let's try

$$\langle M_i, M_j \rangle.$$

This quantity computes the probability that a randomly chosen word w from document i and a randomly chosen word w' from document j are the same. But what makes this a bad measure is that when documents are sparse, they may not have many words in common just by accident because of the particular words each author chose to use to describe the same types of things. Even worse, some documents could be deemed to be similar because they contain many of the same common words, which have little to do with what the documents are actually about.

Deerwester et al. [60] proposed to use the singular value decomposition of M to compute a more reasonable measure of similarity, and one that seems to work better when the term-by-document matrix is sparse (as it usually is). Let $M = U\Sigma V^T$ and let $U_{1\ldots k}$ and $V_{1\ldots k}$ be the first k columns of U and V, respectively. The approach is to compute

$$\langle U_{1\ldots k}^T M_i, U_{1\ldots k}^T M_j \rangle$$

for each pair of documents. The intuition is that there are some *topics* that occur over and over again in the collection of documents. And if we could represent each document M_i on the basis of topics, then their inner product on that basis would yield a more meaningful measure of similarity. There are some models – i.e., hypotheses for how the data is stochastically generated – where it can be shown that this approach provably recovers the true topics [118]. This is the ideal interaction between theory and practice – we have techniques that work (somewhat) well, and we can analyze/justify them.

However, there are many failings of latent semantic indexing that have motivated alternative approaches. If we associate the top singular vectors with topics, then

(1) topics are orthonormal.

However, topics like *politics* and *finance* actually contain many words in common, so they cannot be orthonormal.

(2) topics contain negative values.

Hence, if a document contains such words, their contribution (to the topic) could cancel out the contributions from other words. Moreover, a pair of documents can be judged to be similar because of particular topics that they are both not about.

Nonnegative Matrix Factorization

For exactly the failings we described in the previous section, nonnegative matrix factorization is a popular alternative to the singular value decomposition in many applications in text analysis. However, it has its own shortcomings. Unlike the singular value decomposition, it is *NP*-hard to compute. And the prevailing approach in practice is to rely on heuristics, with no provable guarantees.

Definition 2.1.7 *A nonnegative matrix factorization of inner-dimension r is a decomposition*

$$M = AW$$

where A is $n \times r$, W is $r \times n$, and both are entrywise nonnegative. Moreover, let the nonnegative rank of M – denoted by $\text{rank}^+(M)$ – be the minimum r so that such a factorization exists.

As we will see, this factorization, when applied to a term-by-document matrix, can find more interpretable topics. Beyond text analysis, it has many other applications in machine learning and statistics, including in collaborative filtering and image segmentation. For now, let's give an interpretation of a nonnegative matrix factorization specifically in the context of text analysis. Suppose we apply it to a term-by-document matrix. Then it turns out that we can always put it in a convenient canonical form: Let D be a diagonal matrix where

$$D_{j,j} = \sum_{i=1}^{m} A_{i,j}$$

and further suppose that each $D_{j,j} > 0$. Then

Claim 2.1.8 *Set $\widetilde{A} = AD^{-1}$ and $\widetilde{W} = DW$. Then*

(1) $\widetilde{A}, \widetilde{W}$ are entrywise nonnegative and $M = \widetilde{A}\widetilde{W}$, and
(2) the columns of $\widetilde{A}$ and the columns of $\widetilde{W}$ each sum to one.

We leave the proof of this claim as an exercise, but the hint is that property (2) follows because the columns of M also sum to one.

Hence we can, without loss of generality, assume that our nonnegative matrix factorization $M = AW$ is such that the columns of A and the columns of W each sum to one. Then we can interpret this factorization as follows: Each document is itself a distribution on words, and what we have found is

(1) a collection of r topics – the columns of A – that are themselves distributions on words, and
(2) for each document i, a representation of it – given by W_i – as a convex combination of r topics so that we recover its original distribution on words.

Later on, we will get some insight into why nonnegative matrix factorization is *NP*-hard. But what approaches are used in practice to actually compute such a factorization? The usual approach is *alternating minimization*:

Alternating Minimization for NMF

Input: $M \in \mathbb{R}^{m \times n}$
Output: $M \approx A^{(N)} W^{(N)}$

Guess entrywise nonnegative $A^{(0)}$ of dimension $m \times r$
For $i = 1$ to N
    Set $W^{(i)} \leftarrow \text{argmin}_W \|M - A^{(i-1)}W\|_F^2$ s.t. $W \geq 0$
    Set $A^{(i)} \leftarrow \text{argmin}_A \|M - AW^{(i)}\|_F^2$ s.t. $A \geq 0$
End

Alternating minimization is quite general, and throughout this book we will come back to it many times and find that problems we are interested in are solved in practice using some variant of the basic approach above. However, it has no provable guarantees in the traditional sense. It can fail by getting stuck in a locally optimal solution that is much worse than the globally optimal one. In fact, this is inevitable, because the problem it is attempting to solve really is *NP*-hard.

However, in many settings we will be able to make progress by working with an appropriate stochastic model, where we will be able to show that it converges to a globally optimal solution provably. A major theme in this book is to not take for granted heuristics that seem to work in practice "as immutable," because the ability to analyze them will itself provide new insights into when and why they work, and also what can go wrong and how to improve them.