CHAPTER 13

Recommender systems

The problem of choosing items from a large set to recommend to a user comes up in many contexts, including music services, shopping, and online advertisements. As well as being an important application, it is interesting because it has several formulations, some of which take advantage of a particular interesting structure in the problem.

Concretely, we can think about a company like Netflix, which recommends movies to its users. Netflix knows the ratings given by many different people to many different movies, and knows your ratings on a small subset of all possible movies. How should it use this data to recommend a movie for you to watch tonight?

There are two prevailing approaches to this problem. The first, *content-based recommendation*, is formulated as a supervised learning problem. The second, *collaborative filtering*, introduces a new learning problem formulation.

1 Content-based recommendations

In content-based recommendation, we try to learn a predictor, f, that uses the movies that you have rated so far as training data, find a hypothesis that maps a movie into a prediction of what rating you would give it, and then return some movies with high predicted ratings.

The first step is designing representations for the input and output.

It's actually pretty difficult to design a good feature representation for movies. Reasonable approaches might construct features based on the movie's genre, length, main actors, director, location, or even ratings given by some standard critics or aggregation sources. This design process would yield

$$\phi : \text{movie} \to \text{vector} \quad .$$

Movie ratings are generally given in terms of some number of stars, so the output domain might be $\{1, 2, 3, 4, 5\}$. It's not appropriate for one-hot encoding on the output, and pretending that these are real values is also not entirely sensible. Nevertheless, we will treat the output as if it's in $\mathbb{R}$.

Thermometer coding might be reasonable, but it's hard to say without trying it. Some more advanced techniques try to predict rankings (would I prefer movie A over movie B) rather than raw ratings.

Study Question: What is the disadvantage of using one-hot? What is the disadvantage of using $\mathbb{R}$?

Now that we have an encoding, we can make a training set based on *your* previous ratings of movies. Here, $x^{(i)}$ represents the ith movie, $\phi(x^{(i)})$ gives our feature representation of the ith movie, and $y^{(i)} = \text{rating}(x^{(i)})$ is your rating for the ith movie. If you rated j movies so far, our resulting training set looks like

$$D_a = \left\{ \left(\phi(x^{(1)}), \text{rating}(x^{(1)})\right), \left(\phi(x^{(2)}), \text{rating}(x^{(2)})\right), \ldots, \left(\phi(x^{(j)}), \text{rating}(x^{(j)})\right) \right\}$$

The next step is to pick a loss function. This is closely related to the choice of output encoding. Since we decided to treat the output as a real, we can formulate the problem as a regression from $\phi \to \mathbb{R}$, with $\text{Loss}(p, y) = \frac{1}{2}(y - p)^2$ We will generally need to regularize because we typically have a very small amount of data (unless you really watch a lot of movies!).

Finally, we need to pick a hypothesis space. The simplest thing would be to make it linear, but you could definitely use something fancier, like a neural network.

If we put all this together, with a linear hypothesis space, we end up with the objective

$$J(\theta) = \frac{1}{2} \sum_{i \in D_a} (y^{(i)} - \theta^T \phi(x^{(i)}) - \theta_0)^2 + \frac{\lambda}{2} \|\theta\|^2 \quad .$$

This is our old friend, ridge regression, and can be solved analytically or with gradient descent.

2 Collaborative filtering

There are two difficulties with content-based recommendation systems:

- It's hard to design a good feature set to represent movies.
- They only use your previous movie ratings, but don't have a way to use the vast majority of their data, which is ratings from other people.

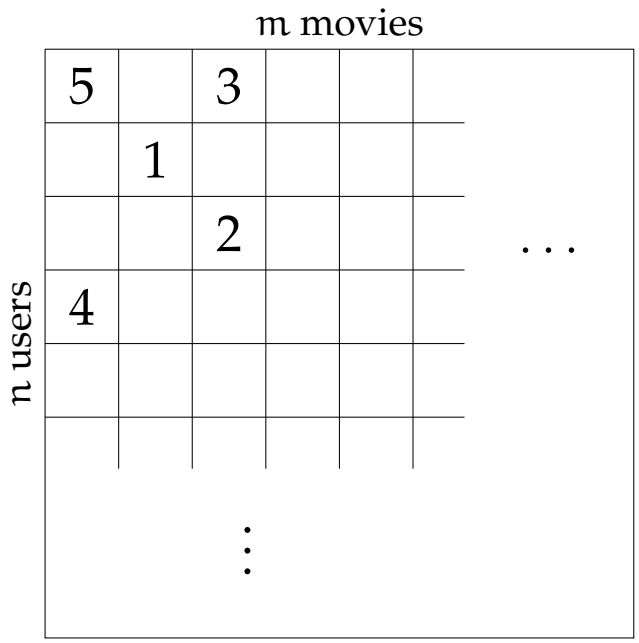
In collaborative filtering, we'll try to use *all* the ratings that other people have made of movies to help make better predictions for you.

Intuitively, we can see this process as finding the kinds of people who like the kinds of movies I like, and then predicting that I will like other movies that they like.

Formally, we will start by constructing a *data matrix* Y, where Y_{ai} represents the score given by user a to movie i. So, if we have n users and m movies, Y has shape $n \times m$.

In fact, there's a third strategy that is really directly based on this idea, in which we concretely try to find other users who are our "nearest neighbors" in movie preferences, and then predict movies they like. The approach we discuss here has similar motivations but is more robust.

We will in fact not *actually* represent the whole data matrix explicitly—it would be too big. But it's useful to think about.



Y is very sparse (most entries are empty). So, we will think of our training data-set as a set of tuples $\{(a, i, r)\}$, where a is the index assigned to a particular user, i is the index assigned to a particular movie, and r is user a's rating of movie i. We will use $D = \{(a, i) : Y_{ai} \text{ is non-empty}\}$ as the set of indices for which we have a rating.

In the Netflix challenge data set, there are 400,000 users and 17,000 movies. Only 1% of the data matrix is filled.

We are going to try to find a way to use D to predict values for missing entries. Let X be our predicted matrix of ratings. Now, we need to find a loss function that relates X and Y, so that we can try to optimize it to find a good predictive model.

Idea #1 Following along with our previous approaches to designing loss functions, we might want to say that our predictions X_{ai} should agree with our data Y_{ai}, and then add some regularization, yielding loss function

$$\text{Loss}(X, Y) = \frac{1}{2} \sum_{(a,i) \in D} (X_{ai} - Y_{ai})^2 + \sum_{\text{all } (a,i)} X_{ai}^2 \ .$$

This is a **bad** *idea!* It will set $X_{ai} = 0$ for all $(a, i) \notin D$.

Study Question: Convince yourself of that!

We need to find a different kind of regularization that will force some generalization to unseen entries.

Linear algebra idea: The *rank* of a matrix is the maximum number of linearly independent rows in the matrix (which is equal to the maximum number of linearly independent columns in the matrix).
If an $n \times m$ matrix X is rank 1, then there exist U and V of shapes $n \times 1$ and $m \times 1$, respectively, such that

$$X = UV^T \ .$$

If X is rank k, then there exist U and V of shape $n \times k$ and $m \times k$, respectively, such that

$$X = UV^T \ .$$

Idea #2 Find the rank 1 matrix X that fits the entries in Y as well as possible. This is a much lower-dimensional representation (it has $m + n$ parameters rather than $m \cdot n$ parameters) and the same parameter is shared among many predictions, so it seems like it might have better generalization properties than our previous idea.

So, we would need to find vectors U and V such that

$$UV^T = \begin{bmatrix} U^{(1)} \\ \vdots \\ U^{(n)} \end{bmatrix} \begin{bmatrix} V^{(1)} & \cdots & V^{(m)} \end{bmatrix} = \begin{bmatrix} U^{(1)}V^{(1)} & \cdots & U^{(1)}V^{(m)} \\ \vdots & \ddots & \vdots \\ U^{(n)}V^{(1)} & \cdots & U^{(n)}V^{(m)} \end{bmatrix} = X \ .$$

And, since we're using squared loss, our objective function would be

$$J(U, V) = \frac{1}{2} \sum_{(a,i) \in D} (U^{(a)}V^{(i)} - Y_{ai})^2 \ .$$

Now, how can we find the optimal values of U and V? We could take inspiration from our work on linear regression and see what the gradients of J are with respect to the parameters in U and V. For example,

$$\frac{\partial J}{\partial U^{(a)}} = \sum_{\{i | (a,i) \in D\}} (U^{(a)}V^{(i)} - Y_{ai})V^{(i)} \ .$$

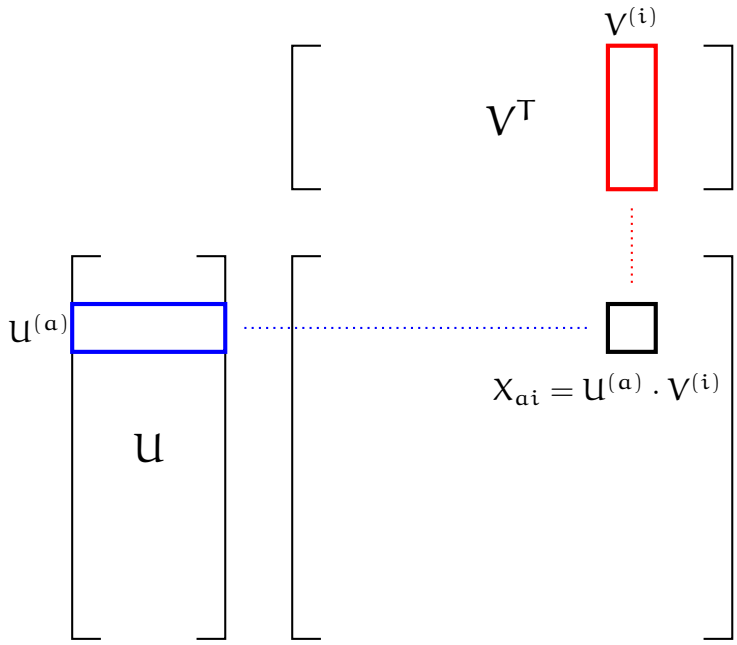
We could get an equation like this for each parameter $U^{(a)}$ or $V^{(i)}$. We don't know how to get an immediate analytic solution to this set of equations because the parameters U and V are multiplied by one another in the predictions, so the model does not have a linear dependence on the parameters. We could approach this problem using gradient descent, though, and we'll do that with a related model in the next section.

But, before we talk about optimization, let's think about the expressiveness of this model. It has one parameter per user (the elements of U) and one parameter per movie (the elements of V), and the predicted rating is the product of these two. It can really represent only each user's general enthusiasm and each movie's general popularity, and predict the user's rating of the movie to be the product of these values.

Study Question: What if we had two users, 1 and 2, and two movies, A and B. Can you find U, V that represents the data set $(1, A, 1), (1, B, 5), (2, A, 5), (2, B, 1)$ well?

Idea #3 If using a rank 1 decomposition of the matrix is not expressive enough, maybe we can try a *rank* k decomposition! In this case, we would try to find an $n \times k$ matrix U and an $m \times k$ matrix V that minimize

$$J(U, V) = \frac{1}{2} \sum_{(a,i) \in D} (U^{(a)} \cdot V^{(i)} - Y_{ai})^2 \ .$$



Here, the length k vector $U^{(a)}$ is the a^{th} row of U, and represents the k "features" of person a. Likewise, the length k vector $V^{(i)}$ is the i^{th} row of V, and represents the k "features" of movie i. Performing the matrix multiplication $X = UV^T$, we see what the prediction for person a and movie i is $X_{ai} = U^{(a)} \cdot V^{(i)}$.

The total number of parameters that we have is $nk + mk$. But, it is a redundant representation. We have 1 extra scaling parameter when $k = 1$, and k^2 extra parameters in general. So, we really effectively have $nk + mk - k^2$ "degrees of freedom."

Study Question: Imagine $k = 3$. If we were to take the matrix U and multiply the first column by 2, the second column by 3 and the third column by 4, to make a new matrix U', what would we have to do to V to get a V' so that $U'V'^T = UV^T$? How does this question relate to the comments above about redundancy?

It is still useful to add offsets to our predictions, so we will include an $n \times 1$ vector b_U and an $m \times 1$ vector b_V of offset parameters, and perform regularization on the parameters in U and V. So our final objective becomes

$$J(U, V) = \frac{1}{2} \sum_{(a,i) \in D} (U^{(a)} \cdot V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai})^2 + \frac{\lambda}{2} \sum_{a=1}^{n} \left\| U^{(a)} \right\|^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \left\| V^{(i)} \right\|^2 .$$

Study Question: What would be an informal interpretation of $b_U^{(a)}$? Of $b_V^{(i)}$?

2.1 Optimization

Now that we have an objective, it's time to optimize! There are two reasonable approaches to finding U, V, b_U, and b_V that optimize this objective: alternating least squares (ALS), which builds on our analytical solution approach for linear regression, and stochastic gradient descent (SGD), which we have used in the context of neural networks and other models.

2.1.1 Alternating least squares

One interesting thing to notice is that, if we were to fix U and b_U, then finding the minimizing V and b_V is a linear regression problem that we already know how to solve. The same is true if we were to fix V and b_V, and seek U and b_U. So, we will consider an algorithm that takes alternating steps of this form: we fix U, b_U, initially randomly, find the best V, b_V; then fix those and find the best U, b_U, etc.

This is a kind of optimization sometimes called "coordinate descent," because we only improve the model in one (or, in this case, a set of) coordinates of the parameter space at a time. Generally, coordinate descent has similar kinds of convergence properties as gradient descent, and it cannot guarantee that we find a global optimum. It is an appealing choice in this problem because we know how to directly move to the optimal values of one set of coordinates given that the other is fixed.

More concretely, we:

1. Initialize V and b_V at random
2. For each a in $1, 2, \ldots, n$:
   - Construct a linear regression problem to find $U^{(a)}$ to minimize
   $$\frac{1}{2} \sum_{\{i|(a,i)\in D\}} \left(U^{(a)} \cdot V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \left\| U^{(a)} \right\|^2 \quad .$$
   - Recall minimizing the least squares objective (we are ignoring the offset and regularizer in the following so you can see the basic idea):
   $$(W\theta - T)^T (W\theta - T) \quad .$$
   In this scenario,
     - $\theta = U^{(a)}$ is the $k \times 1$ parameter vector that we are trying to find,
     - T is a $m_a \times 1$ vector of target values (for the m_a movies a has rated), and
     - W is the $m_a \times k$ matrix whose rows are the $V^{(i)}$ where a has rated movie i.

   The solution to the least squares problem using ridge regression is our new $U^{(a)}$ and $b_U^{(a)}$.
3. For each i in $1, 2, \ldots, m$
   - Construct a linear regression problem to find $V^{(i)}$ and $b_V^{(i)}$ to minimize
   $$\frac{1}{2} \sum_{\{i|(a,i)\in D\}} \left(U^{(a)} \cdot V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai} \right)^2 + \frac{\lambda}{2} \left\| V^{(i)} \right\|^2$$
   - Now, $\theta = V^{(i)}$ is a $k \times 1$ parameter vector, T is a $n_i \times 1$ target vector (for the n_i users that have rated movie i), and W is the $n_i \times k$ matrix whose rows are the $U^{(a)}$ where i has been rated by user a.

     Again, we solve using ridge regression for a new value of $V^{(i)}$ and $b_V^{(i)}$.
4. Alternate between steps 2 and 3, optimizing U and V, and stop after a fixed number of iterations or when the difference between successive parameter estimates is small.

2.1.2 Stochastic gradient descent

Finally, we can approach this problem using stochastic gradient descent. It's easier to think about if we reorganize the objective function to be

$$J(U,V) = \frac{1}{2}\sum_{(a,i)\in D}\left(\left(U^{(a)}\cdot V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai}\right)^2 + \lambda_U^{(a)}\left\|U^{(a)}\right\|^2 + \lambda_V^{(i)}\left\|V^{(i)}\right\|^2\right)$$

where

$$\lambda_U^{(a)} = \frac{\lambda}{\#\text{ times } (a,\_)\in D} = \frac{\lambda}{\sum_{\{i|(a,i)\in D\}} 1}$$
$$\lambda_V^{(i)} = \frac{\lambda}{\#\text{ times } (\_,i)\in D} = \frac{\lambda}{\sum_{\{a|(a,i)\in D\}} 1}$$

Then,

$$\frac{\partial J(U,V)}{\partial U^{(a)}} = \sum_{\{i|(a,i)\in D\}}\left[\left(U^{(a)}\cdot V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai}\right)V^{(i)} + \lambda_U^{(a)}U^{(a)}\right]$$
$$\frac{\partial J(U,V)}{\partial b_U^{(a)}} = \sum_{\{i|(a,i)\in D\}}\left(U^{(a)}\cdot V^{(i)} + b_U^{(a)} + b_V^{(i)} - Y_{ai}\right)$$

We can similarly obtain gradients with respect to $V^{(i)}$ and $b_V^{(i)}$.

Then, to do gradient descent, we draw an example (a,i,Y_{ai}) from D at random, and do gradient updates on $U^{(a)}$, $b_U^{(a)}$, $V^{(i)}$, and $b_V^{(i)}$.

Study Question: Why don't we update the other parameters, such as $U^{(a')}$ for some other user a' or $V^{(i')}$ for some other movie i'?