Matrix Factorization Bootcamp

Gal, Hilik and Noam

June 25, 2018

In these notes we will develop the ideas behind the model and algorithm used in production for OSG recommendations.

1 Making Personalized Recommendations

In our main scenario a user, indexed by m, requests K recommendations out of a catalog with N items. We achieve this by building a model that computes an *affinity score*, $S(m, n)$, that indicates how much user m likes item n. Given this score providing the K requested recommendations in our scenario is achieved by ranking the catalog items $n = 1 \dots N$ according to $S(m, n)$ (descending), keeping m fixed and taking the top K.

2 A Matrix Factorization model for affinity

We will define our model parameters, denoted θ as a whole, are composed of the following:

1. for each user $m = 1 \dots M$:

   $\mathbf{u}_m \in \mathbb{R}^d$ is a d dimensional continuous vector representing user m

2. for each item $n = 1 \dots N$:

   $\mathbf{v}_n \in \mathbb{R}^d$ is a d dimensional continuous vector representing item n

That is, we model both users and items to be points in an abstract vector space sometimes referred to as the 'taste space'. The inner-product between user and item vectors in this space determines the affinity between users and items. Our 'user-item' score is defined as:

$$S(m, n) = \mathbf{u}_m^\top \mathbf{v}_n$$

this is sometimes referred to as a 'bilinear form'. Further if we define $\mathbf{U}$ as the $d \times M$ matrix obtained by stacking user vectors:

$$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_M]$$

and similarly $\mathbf{V}$ as the $d \times M$ matrix :

$$\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N]$$

then according to our model, the matrix $\mathbf{S}$ whose (m,n)-th element is given by $S(m,n)$ factors as:

$$\mathbf{S} = \mathbf{U}^\top \mathbf{V}$$

hence the name *Matrix Factorization.*

3 Learning the model from explicit ratings data

One scenario of interest to us is when users provide us with explicit numerical ratings expressing their affinity for items in the catalog. In this scenario we would have access to a dataset of ratings $\mathcal{R} = \{r_{mn}\}$. Further let $\mathcal{I} = \{(m,n)\}$ be the set of indexes represented in $\mathcal{R}$. We can define the following loss function with respect to the model parameters:

$$\mathcal{E}(\mathcal{R}|\theta) = \sum_{(m,n)\in\mathcal{I}} \frac{1}{2} \left(r_{mn} - S(m,n)\right)^2 = \sum_{(m,n)\in\mathcal{I}} \frac{1}{2} (r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n)^2 \tag{1}$$

This function gives us an approximation of how closely the model approximates the ratings in the the training set,$\mathcal{R}$. We can then reduce the problem of learning a good model from data to the problem of finding a configuration of the model parameters, θ that minimizes the loss function in Equation 1.

3.1 Approach 1: (Stochastic) Gradient Descent

Taking the gradient of Equation 1 w.r.t. the parameter vector θ we obtain:

$$\nabla_\theta \mathcal{E}(\mathcal{R}|\theta) = -\sum_{(m,n)\in\mathcal{I}} e_{mn} \cdot \nabla_\theta \mathbf{u}_m^\top \mathbf{v}_n \tag{2}$$

where $e_{mn} = \left(r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n\right)$ is called the *residual.* Noting that $\nabla_\theta \mathbf{u}_m^\top \mathbf{v}_n = 0$ for data points that don't involve user m or item n we can consider the gradient with respect to each one of the user and item parameter vectors, specifically:

$$\nabla_{\mathbf{u}_m}\mathcal{E}(\mathcal{R}|\theta) = -\sum_{n\in\mathcal{I}_m} e_{mn}\cdot\mathbf{v}_n \tag{3}$$

$$\nabla_{\mathbf{v}_n}\mathcal{E}(\mathcal{R}|\theta) = -\sum_{m\in\mathcal{I}_n} e_{mn}\cdot\mathbf{u}_m \tag{4}$$

where we define $\mathcal{I}_m = \{n : (m,n) \in \mathcal{I}\}$ as the indexes of items rated by user m, and with slight overloading of notation, $\mathcal{I}_n = \{m : (m,n) \in \mathcal{I}\}$ as the indexes of users that rated item n.

In the gradient descent algorithm, we minimize our loss (Equation 1) by taking a step of size η in the direction *opposite* to the gradient direction. Thus, our update step is given by:

$$\theta \leftarrow \theta - \eta\cdot\nabla_\theta\mathcal{E}(\mathcal{R}|\theta)$$

Specifically for each one of the user and item parameter vectors this translates to:

$$\mathbf{u}_m \leftarrow \mathbf{u}_m + \eta\cdot\sum_{n\in\mathcal{I}_m}\cdot e_{mn}\cdot\mathbf{v}_n$$

$$\mathbf{v}_n \leftarrow \mathbf{v}_n + \eta\cdot\sum_{m\in\mathcal{I}_n}\cdot e_{mn}\cdot\mathbf{u}_m$$

Algorithm 1 gives the pseudocode for gradient descent

Gradient descent requires all training data to be considered before performing an update. If we have a large training set, this could involve a significant slow-down. To achieve large scale computational tractability a common approach is to apply stochastic gradient descent, a variant of gradient descent which considers one training example at a time. Taking this approach can be conceived as sampling from the entire training set to estimate the total gradient. The faster running times comes at the cost of guaranteed convergence. Thus, stochastic gradient descent is often run for a fixed number of passes (called epochs) over the data rather than testing for convergence.

The pseudo-code for stochastic gradient descent (SGD) is very similar to gradient descent and is given in Algorithm 2.

3.2 Approach 2: Alternating Least Squares

An alternative approach to learning the parameters from data comes from the analysis of the objective in Equation 1 as a function of the item vector parameters $\mathbf{v}_n$, for $n = 1, \ldots, N$ keeping the other (user vector parameters)

Algorithm 1 Gradient Descent

```
Input: Training data of explicit ratings $\mathcal{R} = \{r_{mn}\}$ , step size $\eta$
Output: setting of $\theta$ that minimizes $\mathcal{E}(\mathcal{R}|\theta)$
  initialize params $\theta$ randomly
  repeat
    initialize gradient vector $\mathbf{g}_\theta$ to $\mathbf{0}$
    for $r_{mn} \in \mathcal{R}$ do
      $e_{mn} \leftarrow r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n$
      $\mathbf{g}_{u_m} \leftarrow \mathbf{g}_{u_m} + e_{mn} \cdot \mathbf{v}_n$
      $\mathbf{g}_{v_n} \leftarrow \mathbf{g}_{v_n} + e_{mn} \cdot \mathbf{u}_m$
    end for
    for $m = 1, \ldots, M$ do
      $\mathbf{u}_m \leftarrow \mathbf{u}_m - \eta \cdot \mathbf{g}_{u_m}$
    end for
    for $n = 1, \ldots, N$ do
      $\mathbf{v}_n \leftarrow \mathbf{v}_n - \eta \cdot \mathbf{g}_{v_n}$
    end for
  until Convergence of $\mathcal{E}(\mathcal{R}|\theta)$
```

Algorithm 2 Stochastic Gradient Descent

```
Input: Training data of explicit ratings $\mathcal{R} = \{r_{mn}\}$ , step size $\eta$ , number
  of epochs $E$
Output: setting of $\theta$ that minimizes $\mathcal{E}(\mathcal{R}|\theta)$
  initialize params $\theta$ randomly
  for $E$ epochs do
    for $r_{mn} \in \mathcal{R}$ do
      $e_{mn} \leftarrow r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n$
      $\mathbf{u}_m \leftarrow \mathbf{u}_m - \eta \cdot e_{mn} \cdot \mathbf{v}_n$
      $\mathbf{v}_n \leftarrow \mathbf{v}_n - \eta \cdot e_{mn} \cdot \mathbf{u}_m$
    end for
  end for
```

constant. Our objective now becomes a quadratic form in the item vectors. To see this consider a particular term in the loss:

$$\left(r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n\right)^2 = -2\mathbf{v}_n{}^\top \left[r_{mn} \cdot \mathbf{u}_m\right] + 2\mathbf{v}_n{}^\top \left[\mathbf{u}_m \mathbf{u}_m^\top\right] \mathbf{v}_n + \mathit{const}$$

where *const* groups all constant terms with respect to $\mathbf{v}_n$.

Now considering the sum in Equation 1 we get

$$\underset{w.r.t.\mathbf{v}_n}{\mathcal{E}(\mathcal{R}|\theta)} = \sum_{(m,n)\in\mathcal{I}} \frac{1}{2}(r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n)^2 = \sum_{(m,n)\in\mathcal{I}} \cdot \left(\mathbf{v}_n{}^\top \left[r_{mn} \cdot \mathbf{u}_m\right] - \mathbf{v}_n{}^\top \left[\mathbf{u}_m \mathbf{u}_m^\top\right] \mathbf{v}_n\right) + \mathit{const} \tag{5}$$

If we consider the above for as a function of vector parameter $\mathbf{v}_n$, we get the sum:

$$\underset{w.r.t.\mathbf{v}_n}{\mathcal{E}(\mathcal{R}|\theta)} = \cdot \left(\mathbf{v}_n{}^\top \left[\sum_{m\in\mathcal{I}_n} r_{mn} \cdot \mathbf{u}_m\right] - \mathbf{v}_n{}^\top \left[\sum_{m\in\mathcal{I}_n} \mathbf{u}_m \mathbf{u}_m^\top\right] \mathbf{v}_n\right) + \mathit{const}$$

In this form we can make several observations:

1. This is a quadratic form in $\mathbf{v}_n$ with an analytic minimizer.
   i.e. $\arg\min_{\mathbf{x}} \left(\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x}\right) = -\mathbf{A}^{-1}\mathbf{b}$
2. It is completely independent of all other item vector parameter. That is, no vector parameters $\mathbf{v}_{n'}$ appear in the above expression for $n' \neq n$
3. the sums in the expression consider only users which have rated the item n in the training dataset. While this number may be large for popular items, for most items it is significantly smaller than M the total number of users in the dataset.

If we consider Equation 1 the above for as a function of vector parameter $\mathbf{u}_m$, we get the similar equation:

$$\underset{w.r.t.\mathbf{u}_m}{\mathcal{E}(\mathcal{R}|\theta)} = \kappa \cdot \left(\mathbf{u}_m{}^\top \left[\sum_{n\in\mathcal{I}_m} r_{mn} \cdot \mathbf{v}_n\right] - \mathbf{u}_m{}^\top \left[\sum_{n\in\mathcal{I}_m} \mathbf{v}_n \mathbf{v}_n^\top\right] \mathbf{u}_m\right) + \mathit{const}$$

The above observations lead to the alternating least squares algorithm. Keeping each "side" fixed in turn we have an analytic minimizer. That is, with all user vector parameters fixed the minimizer for each item vector n is given by:

$$\hat{\mathbf{v}}_n = \left(\sum_{m \in \mathcal{I}_n} \mathbf{u}_m \mathbf{u}_m^\top \right)^{-1} \cdot \sum_{m \in \mathcal{I}_n} r_{mn} \cdot \mathbf{u}_m$$

Similarly with all item vector parameters fixed the minimizer for each user vector m is given by:

$$\hat{\mathbf{u}}_m = \left(\sum_{n \in \mathcal{I}_m} \mathbf{v}_n \mathbf{v}_n^\top \right)^{-1} \cdot \sum_{n \in \mathcal{I}_m} r_{mn} \cdot \mathbf{v}_n$$

Further, since when all user vector parameters are fixed item vectors parameters can be updated independently, these updates can be done in parallel, and similarly for user parameter updates.

If there is a small number of elements in the internal sums in the update ensures that the complexity of the update is dominated by the $d \times d$ matrix inversion (which is manageable since d is small). However, in some cases, the number of elements in the sums is quite large and takes significantly longer in practice than does the matrix inversion. Such a scenario can occur, for example, when a very popular item is rated by a large proportion of the users in the dataset.

The full pseudo-code for the alternating least squares (ALS) algorithm is given below in Algorithm 4.

Algorithm 3 Alternating Least Squares

```
Input: Training data of explicit ratings $\mathcal{R} = \{r_{mn}\}$
Output: setting of $\theta$ that minimizes $\mathcal{E}(\mathcal{R}|\theta)$
  initialize params $\theta$ randomly
  repeat
    for $m = 1, \ldots, M$ (In Parallel) do
      $\mathbf{u}_m \leftarrow \left(\sum_{n \in \mathcal{I}_m} \mathbf{v}_n \mathbf{v}_n^\top\right)^{-1} \cdot \sum_{n \in \mathcal{I}_m} r_{mn} \cdot \mathbf{v}_n$
    end for
    for $n = 1, \ldots, N$ (In Parallel) do
      $\mathbf{v}_n \leftarrow \left(\sum_{m \in \mathcal{I}_n} \mathbf{u}_m \mathbf{u}_m^\top\right)^{-1} \cdot \sum_{m \in \mathcal{I}_n} r_{mn} \cdot \mathbf{u}_m$
    end for
  until Convergence of $\mathcal{E}(\mathcal{R}|\theta)$
```

The advantages of ALS over SGD is that convergence is guaranteed, there are less hyper parameters to consider, and the approach is straightforward to parallelize. However, ALS requires all data to be in memory, which is prohibitive for large datasets. Further, full convergence to the optimum on the training loss may lead to an overfitted solution. Thus, we sometimes seek a stopping point prior to full convergence using a validation set.

4 Additional Modeling Considerations

The simple model and algorithms outlined above gives a starting point, but additional elements are required to achieve optimal results. We describe these elements in this section, however, note that the derivation of the algorithms above will remain structurally the same. Of course some additional parts will need to be added.

4.1 A Global Mean

In practice it is very useful to determine the mean rating value of our training data, denoted μ, and make use of this quantity in our model as follows:

$$S(m,n) = \mu + \mathbf{u}_m^\top \mathbf{v}_n$$

This allows our parameters to model offsets from the mean of the dataset, rather than absolute quantities which are larger in magnitude. Note that μ is not a parameter and is not learned.

4.2 Bias Parameters

The score in the model we have specified so far, depends strictly on the interaction between user and items. In the jargon of our team, we may refer to this as the personalized component of the score. However, some items naturally achieve higher (or lower) ratings (e.g. high quality or popular items). Further some users tend to give higher (or lower ratings) regardless of the item under consideration. To capture these biases in our model we add some additional parameters to our model, fittingly called user biases and item biases respectively. That is, our new model parameters, collectively denoted θ will now be composed as follows:

1. for each user $m = 1 \ldots M$:

   $\mathbf{u}_m \in \mathbb{R}^d$ is a d dimensional continuous vector representing user m

   $b_m \in \mathbb{R}$ is a scalar valued parameter representing the bias of user m's ratings

2. for each item $n = 1 \ldots N$:

   $\mathbf{v}_n \in \mathbb{R}^d$ is a d dimensional continuous vector representing item n

$b_n \in \mathbb{R}$ is a scalar valued parameter representing the bias of ratings of item n

Taking into account our new parameters our model for the affinity score now becomes:

$$S(m,n) = \mu + b_m + b_n + \mathbf{u}_m^\top \mathbf{v}_n$$

4.3 Regularization

A general concept in machine learning is *overfitting*, when a model performs well on training data, but is unable to generalize to unseen test data. A set of techniques to avoid overfitting by preferring a particular group of learned functions is called *regularization*. A common type of regularization consists of adding a penalty term to the optimization objective that penalizes large magnitudes of parameters values. In this type of regularization the new optimization objective takes the form:

$$\mathbf{L}(\theta) = \mathcal{E}(\mathcal{R}|\theta) + \frac{\lambda}{2} \cdot \mathcal{P}(\theta)$$

where $\mathcal{E}(\mathcal{R}|\theta)$ is the unregularized error function , sometimes referred to as the 'data term', and $\mathcal{P}(\theta)$ is called the penalty or regularization term. λ denotes a hyper-parameter which controls the impact of the penalty term. A very common choice for the penalty term is the squared L2 norm i.e. $\mathcal{P}(\theta) = \|\theta\|_2^2 = \sum_i \theta_i^2$.

Thus if we add bias parameters and regularization our objective function becomes:

$$\mathbf{L}(\theta) = \sum_{(m,n)\in\mathcal{I}} \frac{1}{2} \left(r_{mn} - \left(\mu + \mathbf{u}_m^\top \mathbf{v}_n + b_m + b_n \right) \right)^2 + \frac{\lambda}{2} \left(\sum_{m=1}^{M} \left(\|\mathbf{u}_m\|^2 + b_m^2 \right) + \sum_{n=1}^{N} \left(\|\mathbf{v}_n\|^2 + b_n^2 \right) \right) \tag{6}$$

Alternatively, we could define a seperate hyper-parameter for each type of parameter as follows:

$$\begin{aligned}\mathbf{L}\left(\theta\right) = & \sum_{(m,n)\in\mathcal{I}} \frac{1}{2}\Big(r_{mn} - \Big(\mu + \mathbf{u}_m^\top \mathbf{v}_n + b_m + b_n\Big)\Big)^2 \\ & + \frac{\lambda_{\mathbf{u}}}{2}\left(\sum_{m=1}^{M} \left\|\mathbf{u}_m\right\|^2\right) \\ & + \frac{\lambda_{\mathbf{v}}}{2}\left(\sum_{n=1}^{N} \left\|\mathbf{v}_n\right\|^2\right) \\ & + \frac{\lambda_{b_m}}{2}\left(\sum_{m=1}^{M} b_m^2\right) \\ & + \frac{\lambda_{b_n}}{2}\left(\sum_{n=1}^{N} b_n^2\right)\end{aligned} \tag{7}$$

5 A Probabilistic Perspective

Consider the following probabilistic generative model for the data:

$$r_{mn} = \mathbf{u}_m^\top \mathbf{v}_n + \epsilon_{mn}$$

where ϵ_{mn} is some random normally distributed noise with mean 0 and variance $\frac{1}{\beta}$. That is:

$$\epsilon_{mn} \sim \mathcal{N}\left(\epsilon_{mn}; 0, \beta^{-1}\right)$$

Under these assumptions our observed rating will be a random variable normally distributed about our model prediction:

$$r_{mn} \sim \mathcal{N}\Big(r_{mn}; \mathbf{u}_m^\top \mathbf{v}_n, \beta^{-1}\Big) \tag{8}$$

These assumptions along with an i.i.d. assumption about our data allows us to define a *Likelihood* function which gives the probability of our observed data given the parameters. Note that, as the notation implies, the likelihood is considered as a function of the parameters, because it is always evaluated when the data is given and, therefore, constant.

$$\mathbf{L}(\theta) = \prod_{r_{mn} \in \mathcal{R}} \mathcal{N}\left(r_{mn}; \mathbf{u}_m^\top \mathbf{v}_n, \beta^{-1}\right) \tag{9}$$
$$= \prod_{r_{mn} \in \mathcal{R}} \left(\frac{\beta}{2\pi}\right)^{\frac{1}{2}} \exp\left(-\frac{\beta}{2}\left(r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n\right)^2\right)$$

In the *maximum likelihood* framework we seek the pointwise maximizer of Equation 9:

$$\hat{\theta}_{ML} = \arg\max_{\theta} L(\theta)$$

In practice, it is more convenient to maximize the log-likelihood, which shares the same optima as the likelihood owing to the monotonicity of the log function:

$$\ell(\theta) = \log \mathbf{L}(\theta) = \sum_{r_{mn} \in \mathcal{R}} -\frac{\beta}{2}\left(r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n\right)^2 + const \tag{10}$$

where *const* groups all constant terms with respect to θ.

Inspecting equation 10 we see that it is (up to a constant) equivalent to the negative of Equation 1. Of course, the maximum of $\ell(\theta)$ and the minimum of $-\ell(\theta)$ are the same, thus maximum likelihood under this model and minimizing square error as we have previously approached the problem are equivalent. This means all our previous approaches are also maximum likelihood algorithms.

5.1 A Bayesian Model

In Bayesian statistics we specify our prior belief on the value of the parameters and combine this belief with the likelihood term to obtain the posterior distribution according to Bayes' Theorem:

$$\underbrace{P(\theta \mid \mathcal{D})}_{Posterior} \propto \underbrace{P(\mathcal{D} \mid \theta)}_{Likelihood} \underbrace{P(\theta)}_{Prior}$$

One common choice of priors is a spherical Gaussian:

$$P(\theta) = \mathcal{N}\left(\theta; \mathbf{0}, \alpha^{-1} \cdot \mathbf{I}_d\right)$$

We use a slightly more specific prior model which assumes independence between all vector parameters:

$$P(\theta) = \prod_{m=1}^{M} P(\mathbf{u}_m) \prod_{n=1}^{P} (\mathbf{v}_n) \tag{11}$$

where

$$P(\mathbf{v}_n) = \left(\frac{\alpha_{\mathbf{v}_n}}{2\pi}\right)^{\frac{d}{2}} \exp\left(-\frac{\alpha_{\mathbf{v}_n}}{2}\mathbf{v}_n^\top \mathbf{v}_n\right) \tag{12}$$

$$P(\mathbf{u}_m) = \left(\frac{\alpha_{\mathbf{u}_m}}{2\pi}\right)^{\frac{d}{2}} \exp\left(-\frac{\alpha_{\mathbf{u}_m}}{2}\mathbf{u}_m^\top \mathbf{u}_m\right) \tag{13}$$

are multidimensional spherical Gaussian distributions.

With this prior model our negative log posterior becomes equivalent to squared error minimization with L2 Regularization:

$$\begin{aligned} -\log P(\theta \mid \mathcal{D}) &= -\log(P(\mathcal{D} \mid \theta)) - \log P(\theta) + const \\ &= \sum_{r_{mn} \in \mathcal{R}} \frac{\beta}{2}\left(r_{mn} - \mathbf{u}_m^\top \mathbf{v}_n\right)^2 \\ &+ \frac{\alpha_{\mathbf{v}}}{2}\left(\sum_{n=1}^{N} \|\mathbf{v}_n\|^2\right) \\ &+ \frac{\alpha_{\mathbf{u}}}{2}\left(\sum_{m=1}^{M} \|\mathbf{u}_m\|^2\right) \\ &+ const \end{aligned} \tag{14}$$

Comparing with Equation 7 we see the equivalence between seeking a pointwise maximizer of the posterior distribution with spherical Gaussian Priors and minimizing a squared error with L2 regularization. The Bayesian probabilistic setting, allows us to expand our models to other types of likelihoods and priors, as we shall see shortly.

6 Modeling binary data

The scenario described in Section 3 requires users to actively provide their affinity for items. A more realistic setting in industry in general, and our team specifically, is passively observing user interaction with items in our catalog. A very strong signal is whether a user buys/does not buy a particular item. Perhaps a bit unrealistically, but for instructional purposes, we will consider the case where we have binary feedback information on both the user purchases and non-purchases (this is the unrealistic part !).

A bit more abstractly, in this scenario we would have access to a dataset of binary observations $\mathcal{D} = \{d_{mn}\}$. This is nearly the same as as in the explicit ratings scenario, but now each $d_{mn} \in \{0, 1\}$ is a binary observation indicating whether user m purchased item n.

Since our data is binary we would like the output of our model to take on values in the range $[0, 1]$, while still being continuous (this is important because we want to use the affinity score to do ranking). While there are many ways to achieve this a standard approach is to use a sigmoid function. Thus our affinity score between user and item becomes:

$$S(m, n) = \sigma\left(\mathbf{u}_m^\top \mathbf{v}_n\right)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$.

Further, for binary data, a likelihood (the distribution of the data given the parameters) which is a product of Gaussians (which are a continuous distribution no longer makes sense). Thus, we define the likelihood as a product of Bernoulli distributions as follows:

$$\begin{aligned}
\mathbf{L}(\theta) =& P(\mathcal{D} \mid \theta) && (15)\\
=& \prod_{d_{mn} \in \mathcal{D}} \mathrm{Bern}\Big(d_{mn}; \sigma\left(\mathbf{u}_m^\top \mathbf{v}_n\right)\Big) && (16)\\
=& \prod_{d_{mn} \in \mathcal{D}} \sigma\left(\mathbf{u}_m^\top \mathbf{v}_n\right)^{d_{mn}} \cdot \Big[1 - \sigma\left(\mathbf{u}_m^\top \mathbf{v}_n\right)\Big]^{1-d_{mn}}
\end{aligned}$$

Note that, although we have changed the model likelihood to account for the type of data, our model parameters θ remain unchanged from those specified in Section 2 Taking the logarithm of the above expression to obtain the log-likelihood:

$$\ell(\theta) = \sum_{d_{mn} \in \mathcal{D}} d_{mn} \log \sigma\left(\mathbf{u}_m^\top \mathbf{v}_n\right) + (1 - d_{mn}) \log\Big(1 - \sigma\left(\mathbf{u}_m^\top \mathbf{v}_n\right)\Big) \qquad (17)$$

Now let us again model our prior knowledge of our parameters as a product of spherical Gaussians using Equations 11, 12, and 13.

Our negative log posterior for classification is then given by:

$$\begin{aligned}\mathbf{NLP}\left(\theta\right) &= -\log P(\theta \mid \mathcal{D}) = -\log(P(\mathcal{D} \mid \theta)) - \log P(\theta) + const \qquad (18)\\ &= -\left(\sum_{d_{mn}\in\mathcal{D}} \left(d_{mn}\log\sigma\left(\mathbf{u}_m^\top\mathbf{v}_n\right) + (1-d_{mn})\log\left(1-\sigma\left(\mathbf{u}_m^\top\mathbf{v}_n\right)\right)\right)\right)\\ &+ \frac{\alpha_{\mathbf{v}}}{2}\left(\sum_{n=1}^{N}\|\mathbf{v}_n\|^2\right)\\ &+ \frac{\alpha_{\mathbf{u}}}{2}\left(\sum_{m=1}^{M}\|\mathbf{u}_m\|^2\right)\\ &+ const\end{aligned}$$

6.1 A learning algorithm: Stochastic gradient descent

Unlike the probabilistic regression objective in Equation 14, the expression for the classification model (Equation 18) involves the sigmoid function, making analytical solutions unavailable. This precludes us from using the ALS algorithm we previously developed for regression (try taking the derivative w.r.t. to $\mathbf{u}_m$, setting to zero and solving as we did before to see this)

However, the stochastic gradient descent framework still applies. To apply stochastic gradient descent we will again consider the gradient of our optimization objective (Equation 18) with respect to each of our parameters.

$$\begin{aligned}\nabla_\theta\mathbf{NLP}\left(\theta\right) = -&\left(\sum_{d_{mn}\in\mathcal{D}}\left(d_{mn}\left(1-\sigma\left(\mathbf{u}_m^\top\mathbf{v}_n\right)\right)\cdot\nabla_\theta\left(\mathbf{u}_m^\top\mathbf{v}_n\right)\right.\right. \qquad (19)\\ &\left.\left. -(1-d_{mn})\left(-\sigma\left(\mathbf{u}_m^\top\mathbf{v}_n\right)\right)\cdot\nabla_\theta\left(\mathbf{u}_m^\top\mathbf{v}_n\right)\right)\right)\\ &+\alpha_{\mathbf{v}}\left(\sum_{n=1}^{N}\mathbf{v}_n\cdot\nabla_\theta\left(\mathbf{v}_n\right)\right)\\ &+\alpha_{\mathbf{u}}\left(\sum_{m=1}^{M}\mathbf{u}_m\nabla_\theta\left(\mathbf{u}_m\right)\right)\end{aligned}$$

where we have used the fact that $\frac{d}{dz}\left(\log\sigma\left(z\right)\right) = 1-\sigma(z)$.

Considering the above for particular parameters $\mathbf{u}_m$ and $\mathbf{v}_n$, respectively we get:

$$\nabla_{\mathbf{u}_m}\mathbf{NLP}\left(\mathbf{u}_m\right) = -\left(\sum_{d_{mn}\in\mathcal{D}}\left(\left(d_{mn} - \sigma\left(\mathbf{u}_m^\top\mathbf{v}_n\right)\right)\mathbf{v}_n\right)\right) + \alpha_{\mathbf{u}}\mathbf{u}_m \tag{20}$$

$$\nabla_{\mathbf{v}_n}\mathbf{NLP}\left(\mathbf{v}_n\right) = -\left(\sum_{d_{mn}\in\mathcal{D}}\left(\left(d_{mn} - \sigma\left(\mathbf{u}_m^\top\mathbf{v}_n\right)\right)\mathbf{u}_m\right)\right) + \alpha_{\mathbf{v}}\mathbf{v}_n \tag{21}$$

In stochastic gradient descent we take the gradient with respect to a particular example so that the sum is over a single example d_{mn}. Thus we see that if we define $e_{mn} = \left(d_{mn} - \sigma\left(\mathbf{u}_m^\top\mathbf{v}_n\right)\right)$ then our SGD updates remain the "same" as in the regression case:

$$\mathbf{u}_m \leftarrow \mathbf{u}_m - \eta\left(-e_{mn}\cdot\mathbf{v}_n + \alpha_{\mathbf{u}}\mathbf{u}_m\right)$$
$$\mathbf{v}_n \leftarrow \mathbf{v}_n - \eta\left(-e_{mn}\cdot\mathbf{u}_m + \alpha_{\mathbf{v}}\mathbf{v}_n\right)$$

Algorithm 4 gives the pseudocode for stochastic gradient descent for the classification model.

Algorithm 4 Stochastic Gradient Descent For Classification Model

```
Input: Training data of explicit binary labels $\mathcal{D} = \{d_{mn}\}$ , step size $\eta$ ,
  number of epochs $E$, precision hyper params $\alpha_{\mathbf{u}}$,$\alpha_{\mathbf{v}}$
Output: setting of $\theta$ that minimizes $\mathbf{NLP}(\theta))$
  initialize params $\theta$ randomly
  for $E$ epochs do
    for $d_{mn} \in \mathcal{D}$ do
      $e_{mn} \leftarrow d_{mn} - \sigma\left(\mathbf{u}_m^\top\mathbf{v}_n\right)$
      $\mathbf{u}_m \leftarrow \mathbf{u}_m - \eta\left(-e_{mn}\cdot\mathbf{v}_n + \alpha_{\mathbf{u}}\mathbf{u}_m\right)$
      $\mathbf{v}_n \leftarrow \mathbf{v}_n - \eta\left(-e_{mn}\cdot\mathbf{u}_m + \alpha_{\mathbf{v}}\mathbf{v}_n\right)$
    end for
  end for
```

7 Beyond Point Estimates

The Bayesian point of view gives us a probability distribution over the parameters, the posterior distribution, that takes into account our domain knowledge (or lack thereof) via the prior , and the information given to us by the data, via the likelihood term.