ML2P-VAE

Geoffrey Converse

```
library(ML2Pvae)
```

Introduction

Overview of everything.

Multidimensional Logistic 2-Parameter (ML2P) Model

In Item Response Theory, the goal is often to quantify the relation between the underlying ability (or abilities) of students and their performance on an assessment. Formally, we assume that every student j posesses some latent ability value Θ_j, which is not directly observable. Often, we assume that Θ_j is distributed normally across the population of N students. In a practical setting, our data for each student may be a binary vector $u_j \in \mathbb{R}^n$ where a 1 (resp. 0) corresponds to a correct (resp. incorrect) answer, to an exam consisting of n questions (items). Given a student j's assessment results, how can we infer the latent value Θ_j?

A naive approach is to simply use the raw exam score, or accuracy. But if two students both score 80% on an exam while missing different items (so $u_1 - u_2 \neq 0_n$), it is unlikely that they have the exact same ability value. Not all exam items are equal - they often vary in difficulty and in the skills required to answer them correctly.

Instead, theory has been developed [1] that states that the probability of student j answering item i correctly is a function of Θ_j, along with parameters Λ_i associated with item i:

$$P(u_{ij} = 1|\Theta_j) = f(\Theta_j; \Lambda_i)$$

In this work, we are concerned with the case where $\Theta_j = (\theta_{j1}, ..., \theta_{jK})^T \in \mathbb{R}^K$, so that an exam is designed to assess multiple skills. For example, an elementary math exam may test the abilities "add," "subtract," "multiply," and "divide." In order to keep track of which items asses which abilities, we can develop a binary matrix $Q \in \mathbb{R}^{K \times n}$ where

$$Q_{ki} = \begin{cases} 1 & \text{if item } i \text{ requires skill } k \\ 0 & \text{otherwise} \end{cases}$$

It is important to note that multiple items can require multiple skills.

The model that this package focuses on is the Multidimensional Logistic 2-Parameter (ML2P) model. The probability of a correct response is given by

$$P(u_{ij} = 1|\Theta_j) = \frac{1}{1 + \exp[-\vec{a}_i^T \Theta_j + b_i]} = \frac{1}{1 + \exp[-\sum_{k=1}^K a_{ik}\theta_{jk} + b_i]}$$

where the discrimination parameters $\vec{a}_i^T = (a_{ik})_{1 \leq k \leq K}$ determine *how much* of skill k is required to answer item i correctly, and the difficulty parameter b_i quantifies the difficulty of item i.

In application, we often only have response vectors for a set of students - so we need to estimate the ability parameters Θ_j for each student, along with the discrimination and difficulty parameters for each item. While there are many parameter estimation methods, including the Expectation-Maximization algorithm and various MCMC methods, many of them become infeasible as the number of latent traits increases beyond $\dim(\Theta_j) = 7$.

Variational Autoencoders

How a VAE works.

ML2P-VAE

Basic model description

List of package functions

Implementation Details

While here are plenty of examples of implementing a basic VAE available, the modified architecture we use here has a few specifications which are non-trivial. The first of these is to use the Q matrix to restrict non-negative weights in the decoder. Since most neural networks are a black box and use densely connected layers, specifying particular connections between nodes is uncommon. In order to achieve this, we use a custom kernal constraint function that multiplies a weight matrix element-wise by the given binary Q matrix. The matrix dimensions match by construction.

The other, and more difficult, feature to implement is building a VAE which fits the encoded dimension to a multivariate normal distribution $\mathcal{N}(\mu_1, \Sigma_1)$ with a full covariance matrix Σ_1. All available code examples of a VAE assume that $\mu_1 = 0$ and $\Sigma_0 = I$. This is a reasonable assumption in nearly all VAE applications. Since VAE are both uninterpretable and unsupervised, the user doesn't know what features the encoded space represents, so it is convenient to force each node in the encoded layer to be independent and normally distributed. In ML2P-VAE, we know exactly what the values in this hidden layer should represent. Further, we (ideally) know how each node in the enoded layer should relate to each other. In other words, we are given a covariance matrix that gives the relationship between each latent ability θ_k.

Note that after training, we can feed any data sample x_0 through the encoder to obtain nodes correlating to $\mathcal{N}(\mu_0, \Sigma_0)$, the latent distribution for that data point. There are two things to consider in the full covariance matrix VAE implementation: (i) we need to be able to sample from the distribution, and (ii) we need to calculate the Kullback-Liebler Divergence as part of our loss function.

In order to sample from $\mathcal{N}(\mu_0, \Sigma_0)$, we first need a matrix G_0 such that $G_0 G_0^T = \Sigma_0$. Next, sample $\varepsilon_0 = (\varepsilon_1, ..., \varepsilon_k)^T$ with each $\varepsilon_i \sim \mathcal{N}(0, 1)$. Then the sample we generate is $z_0 = \mu_0 + G_0\varepsilon_0$. The KL Divergence between two multivariate normal distributions is given as

$$\mathcal{D}_{KL}\left[\mathcal{N}(\mu_0, \Sigma_0)||\mathcal{N}(\mu_1, \Sigma_1)\right] = \frac{1}{2}\left(\mathrm{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T\Sigma_1^{-1}(\mu_1 - \mu_0) - k + \ln\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right)\right) \tag{1}$$

In our case, μ_1 and Σ_1 are constant, since $\mathcal{N}(\mu_1, \Sigma_1)$ is the distribution we desire to fit out data to. So calculating Σ_1^{-1} just one time shouldn't give any computational issues. However, notice that computing Equation 1 requires $\ln(\det\Sigma_0)$. The matrix Σ_0 depends on the input x_0, as well as every weight and bias in the encoder. Since weights are often initiallized randomly, we need a clever way to ensure that $\det\Sigma_0 > 0$, regardless of the input or network parameters.

To acheive this, for a latent distribution of dimension K, the encoder outputs $K + K(K+1)/2$ nodes. K of these represent the mean vectore μ_0, while the remaining $K(K+1)/2$ represent a lower triangular matrix L_0. In order to sample, we calculate the matrix exponential $G_0 = e^{L_0}$. Note that G_0 is lower triangular and nonsingular. We can then use the matrix G_0 to generate a sample to pass through the decoder.

In our computation of KL Divergence, we use the same matrix G_0 to compute $\Sigma_0 = G_0 G_0^T$. From this

construction, we have

$$\begin{aligned}\det \Sigma_0 &= \det(G_0 G_0^T) \\ &= \det e^{L_0} \cdot \det(e^{L_0})^T \\ &= e^{\mathrm{tr} L_0} \cdot e^{\mathrm{tr} L_0^T} \\ &> 0\end{aligned}$$

This ensures that there will be no problems with computing Equation 1 at any point during training. But we can also show that Σ_0 represents a covariance matrix, i.e., that Σ_0 is symmetric and positive definite.

Theorem 1. *Σ_0 under the previously described construction is symmetric and positive definite.*

Proof. Consider any lower triangular $L_0 \in \mathbb{R}^{k \times k}$. Consider the matrix exponential

$$G_0 := e^{L_0} = \sum_{n=0}^{\infty} \frac{L_0^n}{n!} = I + L_0 + \frac{1}{2} L_0^2 + \cdots$$

G_0 is lower triangular, since addition and multiplication of matrices preserve lower triangular. G_0 is also nonsingular, as $\det G_0 = \det e^{L_0} = e^{\mathrm{tr} L_0} \neq 0$. Set $\Sigma_0 := G_0 G_0^T$. Now for any nonzero $y \in \mathbb{R}^k$,

$$\langle \Sigma_0 y, y \rangle = y^T \Sigma_0 y = y^T G_0 G_0^T y = \langle G_0^T y, G_0^T y \rangle = ||G_0^T y||_2^2 > 0$$

□

Code Examples of how to use the package

References

[1] Wainer and Thissen, D. "Test Scoring". Erlbaum Associates, Publishers, 2001.