UKR のアルゴリズム

2021 年 6 月 23 日

表 1　変数記号表 (論文では表記が違うので注意)

記号	
N	データ数
D	データの次元数
L	潜在空間の次元数
$\mathbf{X}$	データ集合
	$\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^T \in \mathbb{R}^{N \times D}$
	$\mathbf{x}_n = (x_{n1}, \ldots, x_{nD})^T$
$\mathbf{Y}$	観測データの推定値集合
	$\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)^T \in \mathbb{R}^{N \times D}$
	$\mathbf{y}_n = (y_{n1}, \ldots, y_{nD})^T$
$\mathbf{Z}$	潜在変数集合
	$\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^T \in \mathbb{R}^{N \times L}$
	$\mathbf{z}_n = (z_{n1}, \ldots, z_{nL})^T$
T	総学習回数
η	学習率
σ	平滑化カーネルのカーネル幅（SOM でいう近傍半径）

1　UKR のシミュレーションコードの作成手順

1.1　人工データの作成

今回は SOM と同じく X を人工的に作成する.

1.2　アルゴリズム部の作成

1.2.1　初期化

潜在変数 $\mathbf{z}$ を乱数によって初期化し、学習をスタートする. その際、配列のサイズに注意する.

1.2.2　誤差関数が最小となるように勾配法で潜在変数の更新を学習回数 T 回繰り返す.

- 写像の定義

$$\mathbf{y}_n = \sum_i \frac{k\left(\mathbf{z}_n, \mathbf{z}_i\right)}{K(\mathbf{z}_n)} \mathbf{x}_i \tag{1}$$

$$k\left(\mathbf{z}, \mathbf{z}'\right) = \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{z} - \mathbf{z}'\right\|^2\right) \tag{2}$$

$$K\left(\mathbf{z}\right) = \sum_i k\left(\mathbf{z}, \mathbf{z}_i\right) \tag{3}$$

- 潜在変数の推定

  誤差関数

$$E(\mathbf{Z}) = \frac{1}{N}\sum_i \left(\left\|\mathbf{x}_i - \mathbf{y}_i\right\|^2 + \lambda R(\mathbf{z}_i)\right) \tag{4}$$

  $R(\mathbf{z}_i)$ は潜在変数に対する正則化項である（λ は正則化の強さを決めるハイパーパラメータ）.

  潜在変数が ± 1 の一様分布に従うときは,

$$R(\mathbf{z}) := \begin{cases} 0 & \text{if} -1 \leq \mathbf{z} \leq 1 \\ \infty & \text{otherwise} \end{cases} \tag{5}$$

  潜在変数が ガウス分布に従うときは,

$$R(\mathbf{z}) := \left\|\mathbf{z}\right\|^2 \tag{6}$$

  となる.

  誤差関数の微分

$$\frac{1}{N}\frac{\partial}{\partial \mathbf{z}_n}\sum_i \left\|\mathbf{x}_i - \mathbf{y}_i\right\|^2 = \frac{2}{N\sigma^2}\sum_i \left[r_{ni}\mathbf{d}_{nn}^{\mathrm{T}}\mathbf{d}_{ni}\boldsymbol{\delta}_{ni} - r_{in}\mathbf{d}_{ii}^{\mathrm{T}}\mathbf{d}_{in}\boldsymbol{\delta}_{in}\right] \tag{7}$$

$$= \frac{2}{N\sigma^2}\sum_i \left[r_{ni}\mathbf{d}_{nn}^{\mathrm{T}}\mathbf{d}_{ni} + r_{in}\mathbf{d}_{ii}^{\mathrm{T}}\mathbf{d}_{in}\right]\boldsymbol{\delta}_{ni} \tag{8}$$

式の簡単化のために用いた各変数

$$r_{ij} = \frac{k\left(\mathbf{z}_i, \mathbf{z}_j\right)}{K(\mathbf{z}_i)} \tag{9}$$

$$\mathbf{d}_{ij} = \mathbf{y}_i - \mathbf{x}_j \tag{10}$$

$$\delta_{ij} = \mathbf{z}_i - \mathbf{z}_j \tag{11}$$

勾配法による潜在変数の更新

$$\mathbf{z}_n^{\text{new}} = \mathbf{z}_n^{\text{old}} - \eta \frac{\partial E(\mathbf{Z}^{\text{old}})}{\partial \mathbf{z}_n^{\text{old}}} \tag{12}$$

1.3 近傍半径

UKR では（基本的に）近傍半径 σ は固定し，スケジューリングは行わない．

2 検証

実装したモデルは可能な限りの検証を行い学習が正常に動作するかの確認が必須である．

今回は二通りの検証を行う．一つは，学習経過を描画する方法である．もう一つは，他人が組んだプログラムと自分が組んだプログラムの結果を一致させるクロステストである

今回はクロステストに関してはエクストラ目標とする（強制ではない。できればやって欲しいが優先順位を考えて一旦置いておいても良い）

2.1 描画

描画した結果を注意深く観察することによって，目に見える範囲での間違いは修正できる．描画は他の人とも結果を共有できる手段であるので，一人で研究を進める場合は描画に慣れておくことが必須である．

今回は以下の二つの図をアニメーションでプロットする．

- 潜在空間: 潜在変数のプロット
- データ空間: データ点と多様体のプロット

上の 2 つに加えて，目的関数の値の時間変化をプロットすると，学習が収束したか否かの判定や，学習の安定性の確認ができるなど有用である．

2.2 クロステスト

クロステストは複数人が別々に作ったプログラムの実行結果が一致するかを検証するプロセスである．お互いのプログラムの実行結果が一致するなら，実装のミスがある可能性は小さいだろうとする考え方である．論文を提出するような段階にきたら誰かに頼んで一度はするべきである．

今回はサンプルの UKR と自分が組んだモデルの結果を一致させる．結果が一致するか否かは，潜在変数集合 $\mathbf{Z}$ を numpy の allclose 関数によって確認する．潜在変数集合 $\mathbf{Z}$ の他，その写像先の集合 $\{f(z)\}$ や目的関数の値 $E(\mathbf{Z})$ も一致することを確認すると尚良い．また，自動微分による実装でも結果は一致する．

クロステストの手順を以下に記す．

1. 比較する複数のモデルについて，実験条件 (初期値、学習率などのパラメータ) を完全に一致させる．
2. モデルをそれぞれ学習させ，学習結果を変数に格納する．
3. numpy の allclose 関数を用いて，学習結果が一致するかを判定する．このとき，潜在変数集合，写像先などの変数ごとに比較することで，どの変数が一致してどの変数が一致しないのかを判断でき，デバッグしやすくなる．
4. もし一致しなければ，自分のコードと比較対象のコードを見比べるなどしながら，何故一致しないのかを探る．このとき，全 epoch について比較せず，「学習を 1 回だけさせたとき」の学習結果を比較することで問題がより明確になることがある．
5. 学習結果が一致すればクロステスト通過である．これにてそのプログラムを安心して研究に用いることができる．

クロステストのサンプルコードも用意しているので参考にされたい．（Wiki 参照）