November 28, 2019

ロバスト高速オンライン多変量ノンパラメトリック密度推定法

大谷　和樹

富山県立大学　情報基盤工学講座

はじめに

背景

近年，センサーやネットワーク技術の発展に伴って，実環境から大量のデータが継続的にリアルタイムに生成されるようになってきている．どのようなデータが得られるかあらかじめ仮定するのが困難な場合が多く，また実環境のデータは少なからずノイズを含んでいる．確率密度推定は機械学習において重要なタスクであるが，このようなビッグデータに対して行うのは困難である．

目的

本研究では実環境の大量のデータに対応するために，高いロバスト性を有する高速オンライン多変量ノンパラメトリック密度推定法を提案する．

はじめに

ビッグデータに対して，確率密度推定を行うことは既存手法には困難である．なぜなら，次の 3 つの条件を満たす必要があると考えられるからである．

条件 1

生成されるデータを逐次的に学習できる高速なオンライン学習手法であること．

条件 2

ノンパラメトリック密度推定法であること．

条件 3

高いロバスト性を有していること．

カーネル密度推定法

代表的なノンパラメトリック密度推定法としてカーネル密度推定法 (kernel density estimation : KDE) が挙げられる．入力データ $\{\boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathbb{R}^d, i = 1, 2, \cdots, N\}$ に対する KDE による推定密度関数 $\hat{p}(x)$ は，以下のように表せる．

推定密度関数 $\hat{p}(x)$

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N}\sum_{k=1}^{N} K_{\boldsymbol{H}}(\boldsymbol{x} - \boldsymbol{x}_i) \tag{1}$$

ここで K はカーネル関数と呼ばれ，次のガウスカーネルがよく用いられる．

ガウスカーネル

$$K_{\boldsymbol{H}}(\boldsymbol{x} - \boldsymbol{\mu}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{H}|}} \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{H}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})) \tag{2}$$

ここで $\boldsymbol{H}$ はバンド幅行列と呼ばれるパラメータである．

Self-Organizing Incremental Neural Network

Self-Organizing Incremental Neural Network(SOINN) は Growing Neural Gas(GNG) を拡張したプロトタイプベースの教師なし学習手法である.

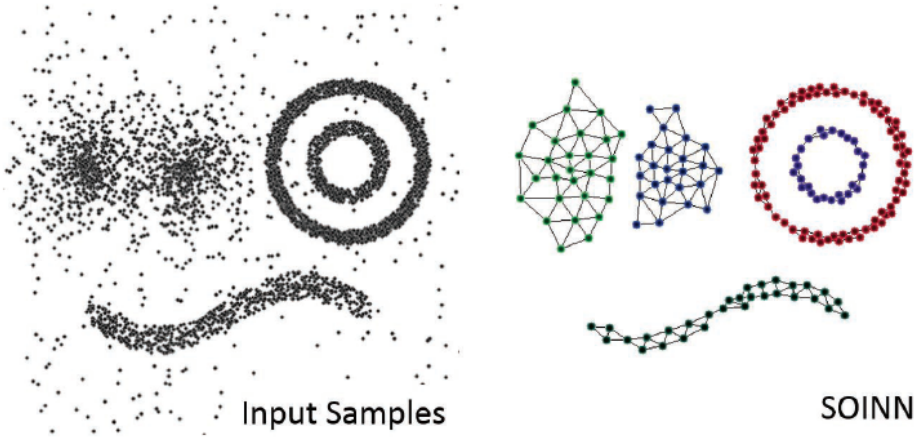

Figure 1. SOINN による学習

SOINN による学習の例を図 1 に示す．SOINN はオンライン学習及び追加学習が可能であり，複雑で非定常な分布からのサンプルも，ノイズを除去しつつ，入力サンプル数より大幅に少ないノード数のネットワーク構造として学習できる．ノード数やノードの初期位置を事前に設定する必要がないため追加学習に適している.

提案手法

提案手法では SOINN のネットワーク構造が表現するサンプルの分布の情報を用いて，密度推定を行う．提案手法の推定密度関数 $\hat{p}(x)$ を以下のように定義する．

推定密度関数 $\hat{p}(x)$

$$\hat{p}(x) = \frac{1}{T_{\mathcal{N}}} \sum_{n \in \mathcal{N}} t_n K_{C_n}(\boldsymbol{x} - \boldsymbol{\omega}_n) \tag{3}$$

ここで $T_{\mathcal{N}} = \sum_{n \in \mathcal{N}} t_n$，$K$ は式 (2) のガウスカーネルである．各ノードの位置にガウスカーネルを配置して，その線形和としてサンプル集合の確率密度を推定する．ノード n に配置したカーネルは，ノード n が代表しているサンプルの数を表している勝者回数 t_n によって重み付けされている．

提案手法

ガウスカーネルの形状を決定する共分散行列 C_n は各ノードの周辺のネットワーク構造から各ノードごとに適応的に決定される．この共分散行列を局所ネットワーク共分散行列と定義する．

局所ネットワーク共分散行列 C_n

$$\boldsymbol{C}_n = \frac{1}{T_{\mathcal{P}_n}} \sum_{p \in \mathcal{P}_n} t_p (\boldsymbol{\omega}_p - \boldsymbol{\omega}_n)(\boldsymbol{\omega}_p - \boldsymbol{\omega}_n)^T \tag{4}$$

ここで $T_{\mathcal{P}_n} = \sum_{i \in \mathcal{P}_n} t_i$ である．

提案手法

図 2 に学習アルゴリズムの全体を示す.

Algorithm 2 学習アルゴリズム

1: **if** 初回の学習である **then**
2: $\mathcal{N} \leftarrow \{c_1, c_2\}$ (c_1, c_2：学習データからランダムに選択したベクトルを位置ベクトルとしてもつ二つのノード)
3: $\mathcal{E} \leftarrow \phi$
4: **end if**
5: **while** 入力パターン $\boldsymbol{\xi} \in \mathbb{R}^d$ が存在する **do**
6: $s_1 \leftarrow \arg\min_{c \in \mathcal{N}} ||\boldsymbol{\xi} - \boldsymbol{w}_c||$
7: $s_2 \leftarrow \arg\min_{c \in \mathcal{N} \setminus \{s_1\}} ||\boldsymbol{\xi} - \boldsymbol{w}_c||$
8: **if** $(\boldsymbol{\xi} - \boldsymbol{w}_{s_1})^T \boldsymbol{M}_{s_1}^{-1} (\boldsymbol{\xi} - \boldsymbol{w}_{s_1}) > 1$ **or** $(\boldsymbol{\xi} - \boldsymbol{w}_{s_2})^T \boldsymbol{M}_{s_2}^{-1} (\boldsymbol{\xi} - \boldsymbol{w}_{s_2}) > 1$ **then**
9: $\mathcal{N} \leftarrow \mathcal{N} \cap \{\boldsymbol{\xi}\}$
10: **else**
11: **if** $(s_1, s_2) \notin \mathcal{E}$ **then**
12: $\mathcal{E} \leftarrow \mathcal{E} \cap \{(s_1, s_2)\}$
13: **end if**
14: $a_{(s_1,s_2)} \leftarrow 0$
15: $a_{(s_1,i)} \leftarrow a_{(s_1,i)} + 1 \ (\forall i \in \mathcal{P}_{s_1})$
16: $t_{s_1} \leftarrow t_{s_1} + 1$
17: $\boldsymbol{w}_{s_1} \leftarrow \boldsymbol{w}_{s_1} + \psi_1(t_{s_1})(\boldsymbol{\xi} - \boldsymbol{w}_{s_1})$
18: エッジ $\mathcal{E}_{old} = \{e \mid e \in \mathcal{E},\ a_e > age_{max}\}$ を削除
19: ノード $\{i \mid \exists j\ (i,j) \in \mathcal{E}_{old}, |\mathcal{P}_i| = 0\}$ を削除
20: **end if**
21: **if** 入力パターン数が λ の倍数 **then**
22: $|\mathcal{P}_i| = 0$ であるノード i を全て削除
23: $\mathcal{N}$ に対して k 近傍グラフ $G = (\mathcal{N}, \mathcal{E}_k)$ を作成
24: $\mathcal{E} \leftarrow \mathcal{E} \cap \{(i,j) \mid (i,j) \in \mathcal{E}_k, (j,i) \in \mathcal{E}_k\}$
25: **end if**
26: **end while**
27: $\mathcal{N}$ に対して k 近傍グラフ $G = (\mathcal{N}, \mathcal{E}_k)$ を作成
28: $\mathcal{E} \leftarrow \mathcal{E} \cap \{(i,j) \mid (i,j) \in \mathcal{E}_k, (j,i) \in \mathcal{E}_k\}$

Figure 2. 学習アルゴリズム

提案手法

学習アルゴリズムにおいて SOINN から拡張した重要な点に，しきい値領域の決定が挙げられる．新規入力サンプルが勝者ノードのしきい値領域内に入るかどうかでエッジ及び新規ノードの作成の判断を行うため，SOINN のアルゴリズムにおいて最も重要な部分である．

SOINN において，ノード i のしきい値領域は次の Θ_i を半径とする超球である．

$$\Theta_i = \begin{cases} \max\limits_{p \in \mathcal{P}_i} ||\boldsymbol{\omega}_p - \boldsymbol{\omega}_i|| & (\mathcal{P}_i \neq \phi) \\ \min\limits_{p \in \mathcal{N}_i} ||\boldsymbol{\omega}_p - \boldsymbol{\omega}_i|| & (otherwise) \end{cases} \tag{5}$$

提案手法

図 3 の (a) に SOINN のしきい値領域を，(b) に提案手法のしきい値領域を図示する．

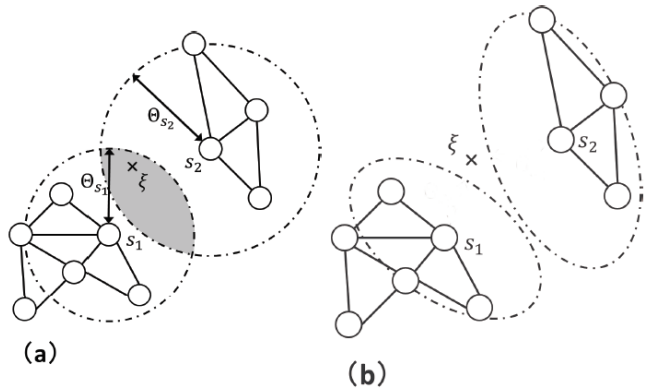

Figure 3. しきい値領域の比較

提案手法では，ネットワーク構造がサンプルの密度を表しているとし，しきい値領域をネットワークに沿う形にし，エッジのない方向にはしきい値領域が広がらないようにしている．こうすることで，SOINN ではサンプルの密度と関係のないエッジになっていた入力サンプルは，しきい値領域に入らずにノードとなり，その後の入力サンプルによってネットワークに組み込まれるか，ノイズとして削除されるか決定される．

提案手法

提案手法ではしきい値領域をネットワークに沿う形にするために，局所ネットワーク共分散行列式 (5) を用いたマハラノビス距離によってしきい値領域を定義する．ノード i のしきい値領域は，サンプル $\boldsymbol{\xi}$ に対して

$$(\boldsymbol{\xi} - \boldsymbol{\omega}_i)^T \boldsymbol{M}_i^{-1} (\boldsymbol{\xi} - \boldsymbol{\omega}_i) \leq 1$$

を満たす領域とする．ここで,

$$\boldsymbol{M}_i = \boldsymbol{C}_i + \rho\gamma_i \boldsymbol{I}$$

$$\gamma_i = \begin{cases} \max\limits_{p \in \mathcal{P}_i} ||\boldsymbol{\omega}_p - \boldsymbol{\omega}_i|| & (\mathcal{P}_i \neq \phi) \\ \min\limits_{p \in \mathcal{N}_i} ||\boldsymbol{\omega}_p - \boldsymbol{\omega}_i|| & (otherwise) \end{cases}$$

である．単位行列を加えているのは，スムージングのためである．

比較手法

ロバスト性，学習時間，オンライン学習，精度に関して提案手法の優位性を示すための評価実験を行った．

既存の比較手法として，クラスタリングによりカーネルの形状と数を適応的に決定するオンライン KDE(oKDE)，ゆう度を評価尺度とし交差検証法によりバンド幅行列を最適化するバッチ学習による手法 (CVML)，ロバスト性を考慮したバッチ手法の KDE (RKDE) を用いた．RKDE は他の手法でバンド幅行列を求める必要があるため，CVML が最適化したバンド幅行列を RKDE のバンド幅行列として用いた．

ロバスト性

ロバスト性を評価するために，学習データにノイズが含まれる場合の推定精度を既存手法と比較した．学習データは，

$$\boldsymbol{X} \sim f = (1-\alpha)f_0 + \alpha f_1$$

とする．f_0 が真の分布，f_1 がノイズの分布であり，α はノイズの割合を表す．各手法は $\boldsymbol{X}$ を学習し，f_0 を推定する．α を変化させ，それに伴う真の分布の推定精度の変化を評価した．真の分布には

$$\begin{aligned} f_0(\boldsymbol{x}) &= N(\boldsymbol{x}; \boldsymbol{t}, 0.2 \cdot \boldsymbol{I}) \\ \boldsymbol{t} &= [a, sin(3a)]^T, a \in [-2, 2] \end{aligned} \tag{6}$$

$$\begin{aligned} f_0(\boldsymbol{x}) &= N(\boldsymbol{x}; \boldsymbol{t}, 0.25 \cdot \boldsymbol{I}) \\ \boldsymbol{t} &= [rcos(\theta), rcos(\theta), \theta]^T, \\ r &= 10 - 0.5\theta, \theta \in [-7, 7] \end{aligned} \tag{7}$$

の二つの分布を用い，ノイズの分布 f_1 にはそれぞれ一様分布（各次元の範囲を $[-4, 4]$ とした）とガウス分布 $N(\boldsymbol{x}; [0, 0, -15]^T, 30 \cdot \boldsymbol{I})$ を用いた．学習サンプル数は $5,000$ とした．

ロバスト性

それぞれ式 (7)，式 (8) の分布に 20 %のノイズを含めた学習データが図 4 である．

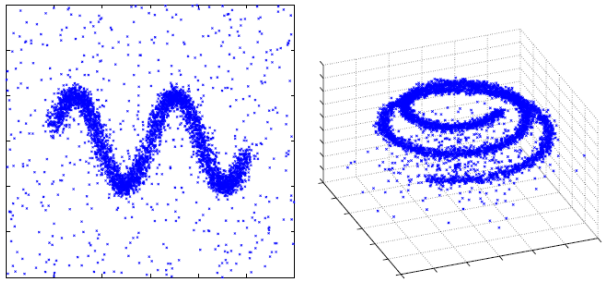

Figure 4. 学習データ

ロバスト性

評価尺度には JS divergence を用いた．JS divergence は確率分布間の距離尺度であり，推定分布と真の分布を比較した場合，値が低いほど推定精度が高いといえる．KL divergence と違い，対称であり無限大に発散することがないので今回はこれを用いた．推定確率分布 $\hat{f}$ と f_0 間の JS divergence $D_{JS}(\hat{f}||f_0)$ は

$$D_{JS}(\hat{f}||f_0) = \frac{1}{2}D_{KL}(\hat{f}||m) + \frac{1}{2}D_{KL}(f_0||m)$$

である．ここで $m = \frac{1}{2}(\hat{f} + f_0)$，$D_{KL}$ は KL divergence であり，

$$D_{KL}(p||q) \approx \frac{1}{n}\sum_{i=1}^{n} x_i \log \frac{p(x_i)}{q(x_i)}$$

である．ここで $\{x_i\}_{i=1}^{n}$ は p からのサンプルである．よって，

$$D_{JS}(\hat{f}||f_0) \approx \frac{1}{2n}\sum_{i=1}^{n} \log \frac{\hat{f}(x_{\hat{f}_i})}{m(x_{\hat{f}_i})} + \frac{1}{2n}\sum_{i=1}^{n} \log \frac{f_0(x_{f_{0i}})}{m(x_{f_{0i}})} \tag{8}$$

となる．

ロバスト性

ここで，$\{x_{\hat{f}_i}\}_{i=1}^n$ は $\hat{f}$ からのサンプルであり，$\{x_{f_{0i}}\}_{i=1}^n$ は f_0 からのサンプルである．今回の実験においては，$n = 20,000$ とした．各ノイズ割合に対して 20 回ずつ実験を行い，その平均を評価した．

oKDE のハイパーパラメータを $D_{th} = 0.01$ とした．提案手法のハイパーパラメータは式 (7) の場合，$\lambda = 500$，$age_{max} = 100$，$\rho = 0.1$，$k = 2d$，式 (8) の場合，$\lambda = 1000$，$age_{max} = 100$，$\rho = 1.0$，$k = 2d$ とした．
$\mathcal{N}$

ロバスト性

実験結果が図 5 である．(a)，(b) はそれぞれ式 (6)，式 (7) の分布に対する実験結果である．

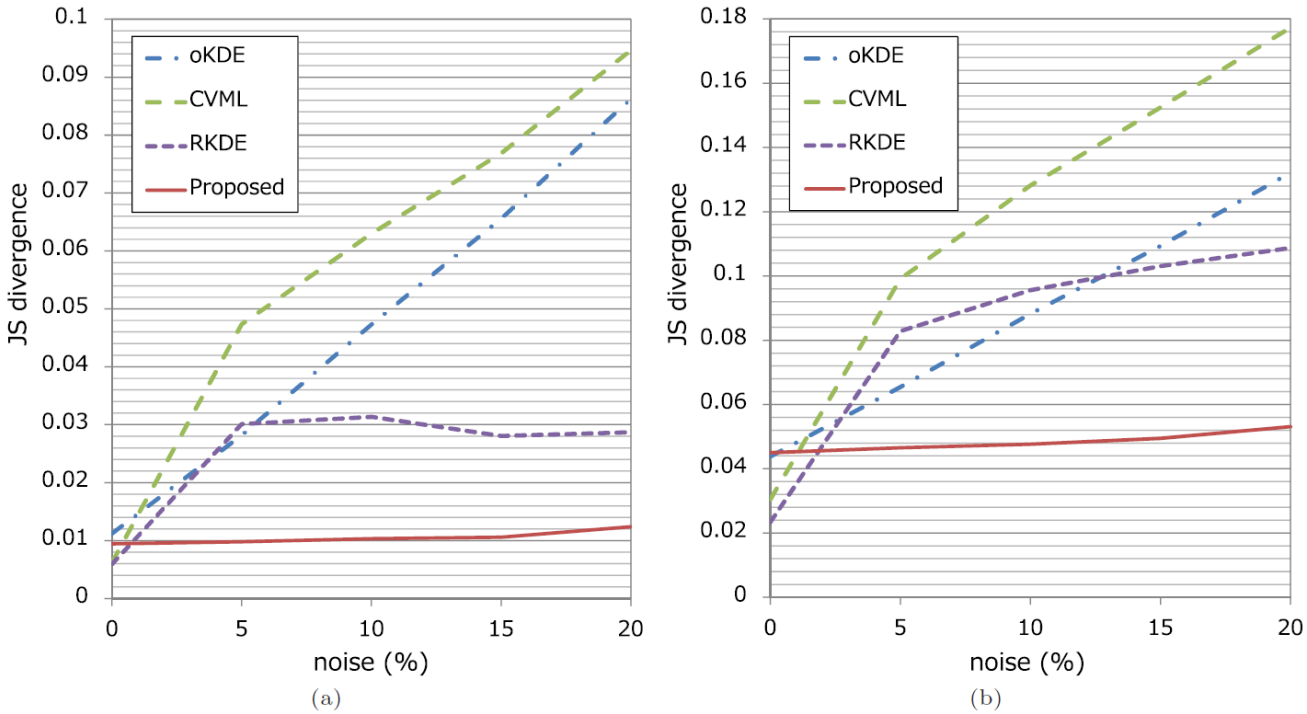

Figure 5. ロバスト性に関する既存手法との比較実験の結果

ロバスト性

図 5(a) において，学習データにおけるノイズ割合が 0 %の場合では，提案手法は他手法とほぼ同等の精度を出している．

既存手法の CVML と oKDE は学習データにおけるノイズの割合が増加するに従って大きく精度が低下している．CVML は 0 %と 5 %の差が大きいことからノイズに敏感に反応してしまうことが分かる．oKDE はカーネルの形状，大きさ，及び数をデータに合わせて適応的に設定するため CVML ほど敏感には反応していないと考えられる．しかし，それだけではロバスト性は十分ではなくノイズの割合が増加するに従って精度が低下している．

これに対して，ロバスト性のある提案手法と RKDE はノイズが増加しても精度が低下しない特に提案手法は 0 %のときと同程度の水準を保っている．図 5(b) においても同様のことが言える．このことから提案手法が高いロバスト性を有していると示された．

学習時間

学習データ数に対する学習時間の変化を評価した．ノイズを含まない式 (7) の分布からのサンプルを用いて，データ数を $1,000$ から $100,000$ まで増加させたときの計算時間の変化を既存手法と比較した．実験結果が図 6 である．

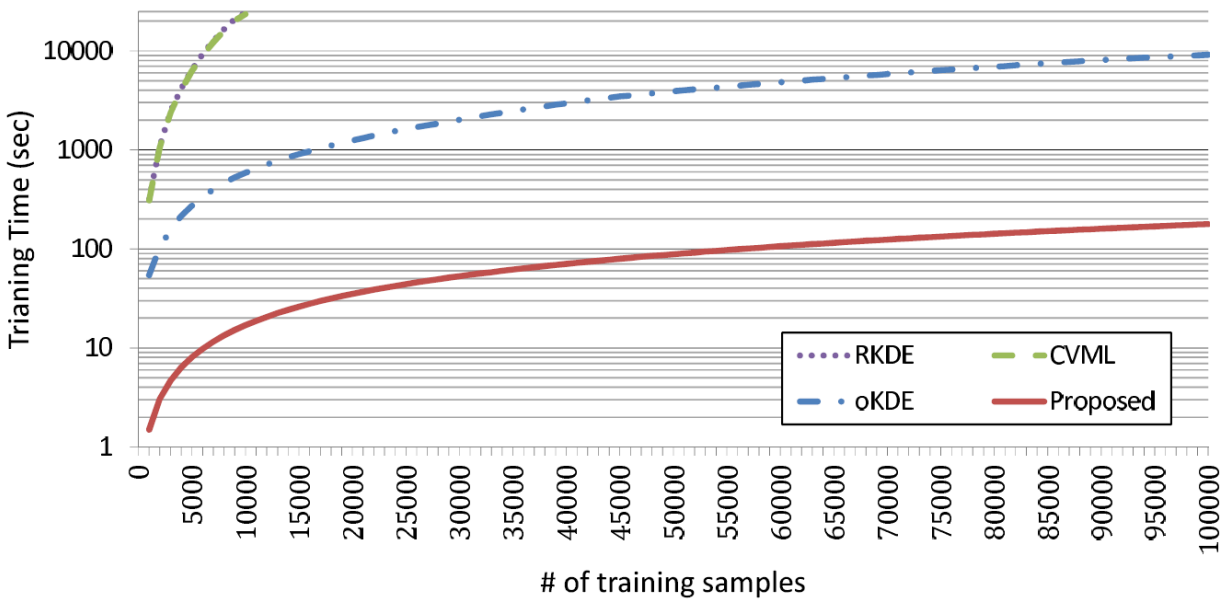

Figure 6. 学習時間に関する評価実験の結果

学習時間

CVML 及び RKDE は非常に学習に時間が掛かるため，学習サンプル数が $10,000$ の時点で実験を中止した．このとき，提案手は CVML より $1,390$ 倍高速であった．CVML はバッチ手法であり，学習サンプル数の増加に伴って学習時間が指数関数的に増加している．

oKDE と提案手法はオンライン手法であるため CVML ほど学習時間は掛かっていないが，oKDE は学習サンプル数が増加するとともに学習時間が大きく増加しているのに対し，提案手法は増加が小さい．提案手法は，学習サンプル数が $100,000$ のとき oKDE より 51.4 倍高速であった．oKDE はクラスタリングを用いて混合数を減らし，バンド幅行列を入力サンプルごとに計算し直しているのに対し，提案手法はプロトタイプのネットワーク構造の中にカーネルの配置位置とバンド幅行列の情報を含めているため，より高速に実行できるのではないかと考えられる．

オンライン学習

オンライン学習手法はデータを追加的に学習できる．そこで，オンライン学習手法である提案手法と oKDE に対して，学習サンプルを追加したときの性能の変化を評価した．学習サンプル数を 100,000 まで追加的に増やし，1,000 サンプル学習するごとに各手法の学習時間と推定精度を評価した．実験結果は図 7 と図 8 である．

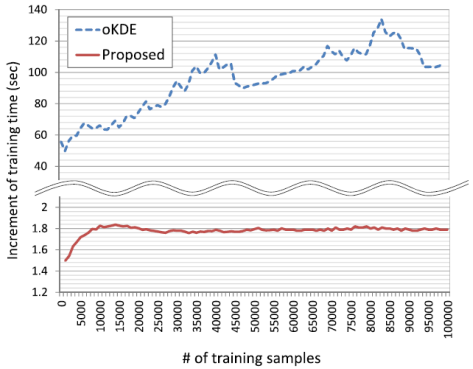

Figure 7. 追加的学習における学習時間の増加量に関する評価実験結果

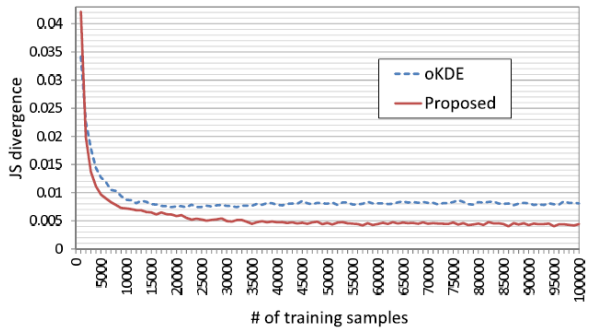

Figure 8. 学習サンプル数の増加に伴う推定精度の変化の評価実験結果

オンライン学習

図 7 は学習したサンプル数に伴う学習時間の増加量，つまり 1,000 サンプルを追加的に学習するのに要した時間の変化を示す．学習したサンプル数の増加に伴い，oKDE の学習時間の増加量は増加する傾向にある．これに対して，提案手法の増加量は初期は増加するが，その後ほぼ一定である．つまり，提案手法の増加量はこれまで学習したサンプル数に直接的には依存しないことを示している．提案手法の学習における計算はノードとの距離計算が大きな部分を占めているがネットワークがデータの分布を十分近似するとノードの数はほとんど変化しないため，学習時間の増加量もほとんど変化しない．このことから，提案手法はデータ量が増大して大規模になっても高速に学習を行えることが分かる．

図 8 は学習したサンプル数に伴う推定精度の変化を示す．oKDE は学習サンプル数が増えるに従って学習時間の増加量が大きくなるにもかかわらず，JS divergence の値は学習サンプル数が 10,000 を超えたあたりからほとんど変わらず，学習の効果がない．これに対して，提案手法はサンプル数が増えても学習時間の増加量はほとんど増えないが JS divergence の値は徐々に下がっている．

実データに対する密度推定

UCI Machine Learning Repository のデータセットを使って実データにおける確率密度推定を評価した．実験に用いたデータセットを図 9 に示す．データセットのクラスごとに実験を行った．ただし，Skin は正例，負例の 2 クラスあるが，正例 (50,859 サンプル) のみ用いた．各実験のサンプル集合を白色化により正規し，そこからランダムに抽出した 75 %のサンプルを学習データ，残りをテストデータとし，密度関数の推定精度を評価した．これを各実験ごとに 20 回繰り返し，データセットごとに平均し評価した．評価尺度には Negative Log Likelihood (NLL) を用いた．NLL は無限大に発散する可能性があるため，順序平均をとることで対応した．提案手法のハイパーパラメータを $\lambda = 1000$，$age_{max} = 100$，$\rho = 1.0$，$k = 2d$，oKDE のハイパーパラメータを $D_{th} = 0.1$ とした．

	サンプル数	次元数	クラス数
Iris	150	4	3
Wine	178	13	3
Pima	768	8	2
Wine_Red	1,599	11	6
Wine_White	4,898	11	7
MAGIC	19,020	10	2
Skin	245,057	3	2

Figure 9. 実験に用いた実データセット一覧

実データに対する密度推定

図 10 は実験結果である.

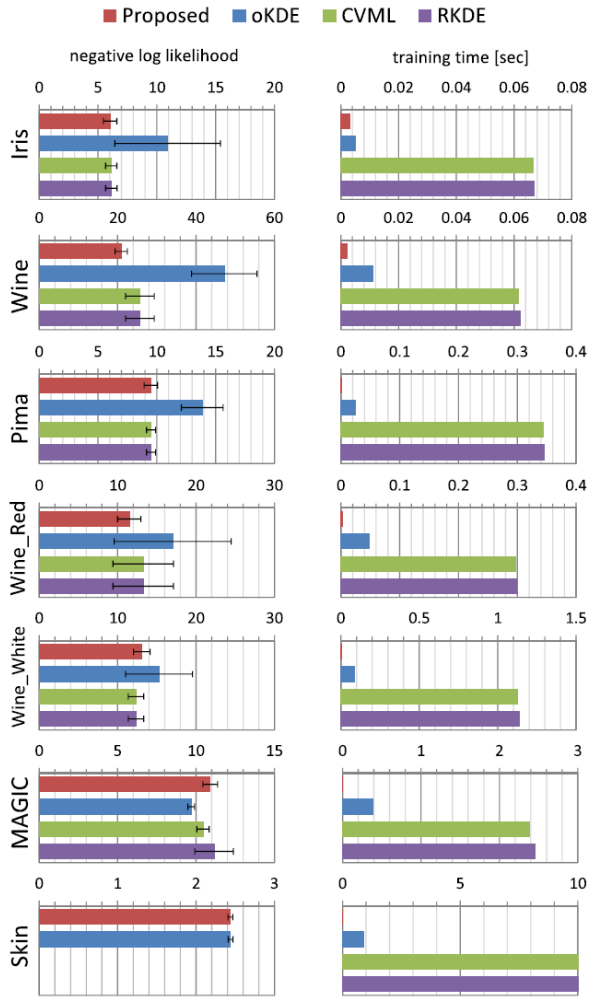

Figure 10. 実データに対する評価実験の結果

実データに対する密度推定

左列が NLL，右列が 1 サンプルあたりの学習時間である．棒グラフは平均値，それに付随するバーは標準偏差を表している．Skin データセットにおいては，CVML と RKDE は 5 日間計算し続けたが 1 度も実験が終わらなかったため値は記していない．提案手法は既存のバッチ学習手法と同等またはそれ以上の推定精度を出している．また，学習時間も他手法と比べて非常に短い．提案手法の学習時間は値が小さすぎ，図 10 では見難いため，図 11 に記した．他手法は学習サンプル数が増えるに従って学習時間が増加しているのに対し，提案手法はほとんど変化がない．以上のことから，提案手法は実データに対する密度推定において実用的に使用することができる．

	NLL	training time [sec]
Iris	6.03 ± 0.827	3.03×10^{-3}
Wine	20.9 ± 1.45	1.91×10^{-3}
Pima	9.49 ± 0.591	2.14×10^{-3}
Wine_Red	11.5 ± 2.11	2.31×10^{-3}
Wine_White	13.1 ± 1.28	2.42×10^{-3}
MAGIC	10.9 ± 0.440	2.02×10^{-3}
Skin	2.43 ± 0.0361	2.77×10^{-3}

Figure 11. 提案手法の実データに対する実験結果

おわりに

	CVML [16]	RKDE [9]	oKDE [23]	**Proposed**
Non-parametric	○	○	○	○
Fast online learning	×	×	△	○
Robustness	×	△	×	○
Accuracy	○	○	△	○

Figure 12. 既存手法との比較

まとめ

1. 提案手法はロバスト性及び高速なオンライン学習に関して既存手法より優れており，推定精度に関しても他手法と同等またはそれ以上の性能を示した．

今後の課題

1. 高次元への対応のための提案手法の拡張と，提案手法の実環境のビックデータ解析への応用に取り組んでいきたいと考えている．

確率密度推定の条件

条件 1

生成されるデータを逐次的に学習できる高速なオンライン学習手法であること．

ビックデータの本質はデータの生成される速度にある．データが高速に大量に生成されているため，結果的にデータの総量が莫大になる．そのため，ビックデータに対応するには継続的にリアルタイムに大量に生成されるデータを処理しなくてはならず，バッチ学習では対応できない．なぜなら，学習が終わる前に新たなデータが生成されてしまい，またデータ量が増えるに従って計算量が大幅に増加してしまうため，学習が間に合わないからである．

確率密度推定の条件

条件 2

ノンパラメトリック密度推定法であること．

パラメトリック密度推定法

観測データが既知のパラメトリックな分布から発生したと仮定し，そのパラメータを推定することで密度関数を推定する方法．

ビッグデータの解析の主な目的はデータマイニング・知識発見であり，多様な種類のデータを用いるとともに，いかようなデータが取得できるかをあらかじめ予測することが難しい場合が多い．したがって，パラメトリック密度推定法は適用できない．これは，実際のデータの分布が仮定した分布と異なっていた場合，大幅な推定精度の低下を招いてしまうためである．未知の分布や既知のパラメトリックな分布としてモデル化できない分布に対して，パラメトリック密度推定法を用いて密度関数を推定することは困難である．

確率密度推定の条件

条件 3

高いロバスト性を有していること．

ロバスト性

外れ値や欠損値などのノイズを含まない理想的な状態ではないデータを学習した場合でも，理想的なデータの場合と同様の学習結果が得られる性質．ノイズは以下の 2 種類に大別されると考えられる．
(1) 外れ値や異常値，拡散または出現頻度の低い分布から発生したサンプルなどの，目的の分布とは無関係であり，環境から発生したと考えられるノイズ
(2) クラス内変動，分散，または揺らぎなどの，現象に内在し確率分布として捉えるべきノイズ

データは実環境から取り入れられたものであり，ノイズを多く含むと考えられる．そのまま学習した場合，ロバスト性のない手法では，ノイズに対しても適合してしまい過学習を起こし汎化性能の低下につながる．また．ノイズの分布を仮定することで対処することも考えられるが，仮定したノイズの分布が実際のノイズの分布と異なっていた場合，著しい性能の低下を招いてしまう．人手でノイズを除去することもできるが，大規模に高速に生成されるデータにおいては非常に困難である．したがって，ノイズを含むデータからでも学習できる高いロバスト性を有する手法が必要である．

関連研究

関連研究 1

バンド幅行列は性能に大きく影響を与えるため，この最適化に関して多くの研究が行われているが，これらの手法はバッチ学習手法であり，大規模データに対応できない．KDE では入力サンプル全てに対してカーネルを配置するため，計算量と精度がトレードオフの関係にある．

関連研究 2

計算量を抑えるために，カーネルの適切な配置や，オンライン学習手法を取り入れる等の工夫が必要である．カーネルの数を設定する手法が幾つか提案されているが，オンライン化の方法は，KDE に対して適用した手法は探した限り見つからなかった．KDE はサンプルが増えるたびモデルが変化するため，オンラインに最適化を行うのが困難なためであると思われる．クラスタリング等を用いることで KDE をオンライン化した手法は考案されているが，これらはロバスト性と学習速度が十分とは言えない．

関連研究 3

式 (1) が示すように，KDE では入力サンプル全てに対してカーネルを配置し，その線形和で密度関数表現する．入力サンプルにノイズを含む場合，ノイズにもカーネルを配置するため，ノイズに対しても適合してしまい過学習を起こしてしまう．対処法としては，ノイズに対する事前知識を導入する手法が考案されている．KDE をカーネル法として捉え，ロバストな推定を行うことでロバストな KDE を実現している手法もあるが，この手法はバッチ学習手法であり，大規模データには対応できない．

$\boldsymbol{\xi}$	入力サンプル．$\boldsymbol{\xi} \in \mathbb{R}^d$.
$\mathcal{N}$	全ノードの集合.
$\mathcal{E}$	全エッジの集合．$\mathcal{E} \subset \mathcal{N} \times \mathcal{N}$.
$\mathcal{P}_i$	ノード i の隣接ノード集合.
$\boldsymbol{w}_i$	ノード i の位置ベクトル．$\boldsymbol{w}_i \in \mathbb{R}^d$.
t_i	ノード i が競合学習において勝者になった回数.
a_e	エッジ e の年齢.
Θ_i	エッジ，ノード作成におけるノード i のしきい値.
$\psi_k(t)$	競合学習の勝者ノードの更新における係数決定のための関数．$\psi_1(t) = t^{-1}$，$\psi_2(t) = (100t)^{-1}$
λ	ノード削除ハイパーパラメータ.
age_{max}	エッジ削除ハイパーパラメータ.
ρ	しきい値ハイパーパラメータ.
$\boldsymbol{I}$	単位行列.
$\lvert\mathcal{S}\rvert$	集合 $\mathcal{S}$ の要素数.

Figure 13. アルゴリズム内に現れる記号

Algorithm 1 SOINN

1: **if** 初回の学習である **then**
2: $\mathcal{N} \leftarrow \{c_1, c_2\}$：学習データからランダムに選択したベクトルを位置ベクトルとしてもつ二つのノード c_1，c_2 で全ノードの集合を初期化
3: $\mathcal{E} \leftarrow \phi$
4: **end if**
5: **while** 入力パターン $\boldsymbol{\xi} \in \mathbb{R}^d$ が存在する **do**
6: $s_1 \leftarrow \arg\min_{c \in \mathcal{N}} ||\boldsymbol{\xi} - \boldsymbol{w}_c||$：$\boldsymbol{\xi}$ に対する第 1 勝者ノード s_1 を探索
7: $s_2 \leftarrow \arg\min_{c \in \mathcal{N} \setminus \{s_1\}} ||\boldsymbol{\xi} - \boldsymbol{w}_c||$：$\boldsymbol{\xi}$ に対する第 2 勝者ノード s_2 を探索
8: 式 (6) によって類似度しきい値 Θ_{s_1}，Θ_{s_2} を計算
9: **if** $||\boldsymbol{\xi} - \boldsymbol{w}_{s_1}|| > \Theta_{s_1}$ **or** $||\boldsymbol{\xi} - \boldsymbol{w}_{s_2}|| > \Theta_{s_2}$ **then**
10: $\mathcal{N} \leftarrow \mathcal{N} \cap \{\boldsymbol{\xi}\}$：$\boldsymbol{\xi}$ を新規ノードとして追加
11: **else**
12: **if** $(s_1, s_2) \notin \mathcal{E}$：$s_1$ と s_2 の間にエッジがない **then**
13: $\mathcal{E} \leftarrow \mathcal{E} \cap \{(s_1, s_2)\}$：エッジ (s_1, s_2) を追加
14: **end if**
15: $a_{(s_1,s_2)} \leftarrow 0$：エッジ (s_1, s_2) の年齢を 0 にリセット
16: $a_{(s_1,i)} \leftarrow a_{(s_1,i)} + 1 \ (\forall i \in \mathcal{P}_{s_1})$
：s_1 につながる全エッジの年齢をインクリメント
17: $t_{s_1} \leftarrow t_{s_1} + 1$：$s_1$ の勝者回数をインクリメント
18: $\boldsymbol{w}_{s_1} \leftarrow \boldsymbol{w}_{s_1} + \psi_1(t_{s_1})(\boldsymbol{\xi} - \boldsymbol{w}_{s_1})$
：s_1 の位置ベクトルを更新
19: $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \psi_2(t_i)(\boldsymbol{\xi} - \boldsymbol{w}_i) \ (\forall i \in \mathcal{P}_{s_1})$
：s_1 の全隣接ノードの位置ベクトルを更新
20: エッジ $\mathcal{E}_{old} = \{e \mid e \in \mathcal{E},\ a_e > age_{max}\}$ を削除
21: ノード $\{i \mid \exists j\ (i,j) \in \mathcal{E}_{old}, |\mathcal{P}_i| = 0\}$ を削除
：20 行目において削除されたエッジに接続していたノードのうち，接続エッジがなくなったノードを削除.
22: **end if**
23: **if** 入力パターン数が λ の倍数 **then**
24: $|\mathcal{P}_i| \leq 1$ であるノード i を全て削除
25: **end if**
26: **end while**

Figure 14. SOINN のアルゴリズム

SOINN では，入力サンプルの最近傍 2 ノード（第 1 勝者ノード，第 2 勝者ノード）のしきい値領域に入力サンプルが入らなかった場合は新規ノードが作成され，入った場合は第 1，第 2 勝者ノード間にエッジが作成され新規ノードは作成されず，入力サンプルは第 1 勝者ノードの勝者回数としてカウントされ，第 1 勝者ノード及びその隣接ノードの位置ベクトルを入力サンプルの方向に更新する．

この第 1 勝者ノードの更新，Algorithm 1 の 18 行目は，平均のオンライン更新と一致する．$\{x_1, x_2, \cdots, x_n\}$ の平均を

$$\boldsymbol{\omega}^{(n)} = \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{x}_i$$

とすると，$\{x_1, x_2, \cdots, x_{n+1}\}$ の平均 $\omega^{(n+1)}$ は

$$\begin{aligned} \boldsymbol{\omega}^{(n+1)} &= \frac{1}{n+1}(n\boldsymbol{\omega}^{(n)} + \boldsymbol{x}_{n+1}) \\ &= \boldsymbol{\omega}^{(n)} + \frac{1}{n+1}(\boldsymbol{x}_{n+1} - \boldsymbol{\omega}^{(n)}) \end{aligned} \tag{9}$$

と，$\omega^{(n)}$ と x_{n+1} から計算でき，この式は Algorithm 1 の 18 行目と一致する．

つまり，各ノードは勝者回としてカウントされたサンプルを代表しており，それらの平均に位置している．Algorithm 1 の 19 行目はスムージングのためであり，$\psi_2(t)$ の係数は実験的に決められたものである．

また，Algorithm 1 の 16，20 行目が示すとおり，エッジは，端点のノード間へのサンプルの出現頻度が周辺のエッジと比較して低かった場合，切断されるようになっている．つまり，ノード間にエッジがあるということは，そのノード間のサンプルの出現頻度が相対的に大きいということになり，ノードが代表するサンプルがエッジの方向に広がっているということを意味すると考えられる．

以上をまとめると，ノードは周辺のサンプルを代表するプロトタイプであり，そのノードの周辺ネットワークはノードの周辺サンプルの広がりの大きさとその方向を表していると考えられる．

提案手法のその他の拡張

SOINN のアルゴリズム Algorithm 1 における 19 行目は廃止している．これはスムージングのために実験的に決められたものであり，密度推定の観点からは必要ないと考えられるためである．22 行目では $|\mathcal{P}_i| \leq 1$ ではなく $|\mathcal{P}_i| = 0$ としている．23〜24，27〜28 行目ではネットワークの調整を行っている．SOINN ではノード間のしきい値領域内にサンプルが入力されないとエッジが作成されないため，高次元空間やサンプル数が少ない場合，十分な数のエッジが作成されない．これを補うために，ここでは k 近傍グラフに基いてエッジを追加している．ノード集合に対して k 近傍グラフを形成し，そのグラフにおいて双方向にエッジが存在するノードのペアに対して，エッジを追加する．ノードが密に分布する場所では k 近傍グラフのエッジは双方向に形成されるので，密度を表すエッジが張れると考えられる．

また，6〜7 行目の競合学習の勝者選択においては，マハラノビス距離ではなく，SOINN と同様にユークリッド距離を用いている．これは，各ノードを同一の基準で評価することで，競合学習を適切に行うためである．それに加えて，ここでは全ノードと距離計算を行う必要があるため，マハラノビス距離を用いると計算コストが大きく掛かってしまうためである．