特徴選択を用いた高次元データに対する逆k最近傍検索の高速化

対比地恭平† 天笠 俊之††

† 筑波大学 理工情報生命学術院 システム情報工学研究群 〒305–8577 茨城県つくば市天王台 1–1–1
†† 筑波大学 計算科学研究センター 〒305–8577 茨城県つくば市天王台 1–1–1
E-mail: †tsuihiji@kde.cs.tsukuba.ac.jp, ††amagasa@cs.tsukuba.ac.jp

あらまし 逆k最近傍検索とは，あるクエリ点が与えられたときに，それをk最近傍に持つようなすべてのデータ点を検索する処理である．逆k最近傍検索は地理情報システム（GIS）や外れ値検出，アカウントベースドマーケティング（ABM）など，幅広い分野で応用されており，近年注目を集めている．しかし，既存の研究は低次元のデータや小規模なデータのみを対象にしているものが多く，高次元のデータや大規模なデータを扱えないという問題がある．そこで我々は高次元の大規模データを対象とし，GPUを用いた逆k最近傍検索の高速化手法を提案したが，次元数の増加に伴ってメモリや計算のコストが増加してしまうという課題があった．本研究では，前処理として特徴選択を採用することで，検索の精度を落とさずに効率よく高次元データを扱えるようにする．

キーワード 特徴選択，逆k最近傍検索，高次元データ，GPU，並列処理，高速化

1 はじめに

近年，情報科学の分野で用いられるデータは増加し続けている．これらのデータを解析する際には膨大なコストがかかるため，効率的に処理する手法が求められている．データ解析に用いられる手法の1つに，逆k最近傍検索[1]がある．逆k最近傍検索とは，あるクエリ点が与えられたときに，それをk最近傍に持つようなすべてのデータ点を検索する処理である．逆k最近傍検索は意思決定支援システムや地理情報システム（GIS），外れ値検出など，幅広い分野で応用されており[1]，近年様々な手法が提案されている．

先述以外の逆k最近傍検索の応用先として，アカウントベースドマーケティング（ABM）が挙げられる．ある商品をクエリとして逆k最近傍検索を行うことで，それに関心のあるユーザアカウントを検出する．企業はその結果を元にターゲットを絞り，新しいサービスを提供することができる．このとき，商品やユーザはいずれも高次元の特徴ベクトルとして扱われる．しかし，高次元空間における逆k最近傍検索は，枝刈りによるコスト削減が難しく，次元の呪いによって計算コストが大きくなってしまう．

従来の手法の多くは低次元・小規模データを対象としていることから，いずれもCPUによる逐次処理のみに対応しており，高次元・大規模データを扱うことは困難である．そこで我々は，GPUを用いた逆k最近傍検索の高速化手法を提案した[2]．この手法は，ナイーブな逆k最近傍検索をk最近傍ステップと問合せステップに分け，それぞれをGPUによって高速に処理するものだ．GPUの並列処理によって高次元データを扱った際でも計算コストを抑えることができ，CPUで実行する場合に比較して高速に処理することを示している．しかし，次元数が増えるごとに確実に実行時間は増加しており，1000次元を超えるデータを扱うには実用的ではない．また，高次元空間におけるユークリッド距離は，次元の呪いにより近傍と遠傍の差が小さくなる傾向があり，距離値そのものは小さくてもベクトル同士が類似しているとは限らない．そこで，前処理として特徴選択手法の1つであるMCFS[3]を取り入れて次元削減を行い，計算コストやメモリコストを削減しつつ，検索の精度を高めることを目指す．

本稿の構成は，以下の通りである．2節で逆k最近傍検索，3節で特徴選択について説明する．4節で関連研究を紹介し，5節で提案手法について述べ，6節で実験結果を示す．最後に7節で本稿のまとめを述べる．

2 逆k最近傍検索

データ解析手法の1つに，逆k最近傍検索[1]がある．逆k最近傍検索とは，あるクエリ点が与えられたときに，それをk最近傍（最も近いk個の点）に含むようなすべてのデータ点を見つける処理のことである．すなわち，クエリ点$\boldsymbol{q}$が与えられたとき，データセットSについて以下の式で定義される．

$$RkNN(\boldsymbol{q}) = \{\boldsymbol{p} | \boldsymbol{p} \in S \wedge \kappa_p \leqq k\}$$

ここで，κ_pは，$\boldsymbol{q}$から$\boldsymbol{p}$よりも近い位置にある点の数を意味し，以下の式で定義される．

$$\kappa_p = |\{\boldsymbol{p'} | \boldsymbol{p'} \in \{S - \{\boldsymbol{p}\}\} \wedge d(\boldsymbol{p'}, \boldsymbol{p}) < d(\boldsymbol{p'}, \boldsymbol{q})\}|$$

ここで，$d(\cdot)$は距離関数を表しており，一般的にユークリッド距離が用いられる．

図1は$k = 2$のときの逆k最近傍検索の例を示している．点$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}$はデータ点，点$\boldsymbol{q}$はクエリ点を表しており，各点を中心とする円は2最近傍までの距離を表している．この例において，$\boldsymbol{b}$と$\boldsymbol{d}$が2最近傍までの距離の内側に$\boldsymbol{q}$を含むため，$\boldsymbol{q}$の逆2最近傍となる．

逆k最近傍検索の別の応用先として，アカウントベースド

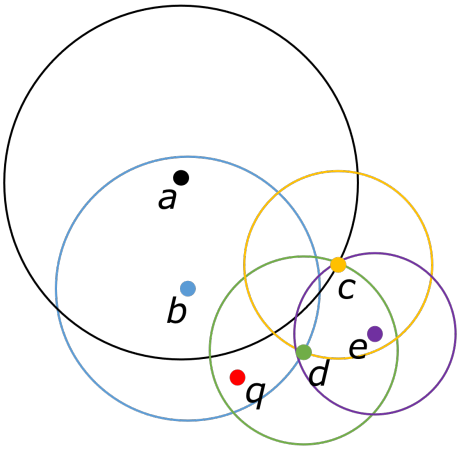

図 1　$k = 2$ のときの逆 k 最近傍検索

マーケティング（ABM）が挙げられる．ある商品をクエリとして逆 k 最近傍検索を行うことで，それに関心のあるユーザアカウントが検出される．企業はその結果を元にターゲットを絞り，新しいサービスを提供することができる．このとき，商品やユーザはいずれも高次元の特徴ベクトルとして扱われる．しかし，高次元空間における逆 k 最近傍検索は，次元の呪いによって計算コストが大きくなってしまう．

3　特 徴 選 択

特徴選択とは，特徴集合のうち意味のある部分集合のみを選択する手法のことである．高次元データにおいて，そのすべての要素が重要というケースはほとんどなく，実際は不要で冗長なデータが多く含まれている．そこで，意味のある特徴を選択し，その特徴量のみを扱うことで，高次元データ特有の次元の呪いの効果を緩和したり，メモリコストを削減したり，本質的な解を得たりすることができる．一般的には機械学習の分野で用いられることが多い．

特徴選択には主に 3 種類のアプローチがある．1 つ目は，フィルター手法である．フィルター手法では，統計学の技術を用いてそれぞれの特徴量のスコアを計算し，最大となるものを選択する．データの分布によっては精度が低くなってしまう可能性もあるが，最もシンプルで計算コストが小さい．2 つ目は，ラッパー手法である．ラッパー手法では，複数の特徴を使って予測精度の検証を行い，精度が最も高くなるような特徴の組み合わせを探索する．フィルター手法よりも高い精度が期待されるが，計算コストが大きいという欠点がある．3 つ目は，組み込み手法である．組み込み手法は機械学習で用いられ，モデルの学習時に特徴選択を行う手法である．フィルター手法とラッパー手法の長所を組み合わせたような手法であり，機械学習のために次元を削減したい場合には有用とされている．

4　関 連 研 究

逆 k 最近傍検索の既存手法の多くは，低次元データのみに対応しており [4] [5] [6]，高次元データに対応した手法は少ない．

Singh らは高次元データに対応した近似逆最近傍検索手法を提案した [7]．まずクエリ点の k 最近傍を求め，それ以外の点を検索の対象から除外する．その後，各データ点に Boolean Range Query を適用し，解の判定を高速に行う．しかし，厳密な解は保証されていない．また，次元削減のために前処理として特異値分解をしているが，大規模データを対象とする場合，計算コストが肥大化してしまう．

我々は，GPU を用いた高次元データに対する逆 k 最近傍検索の高速化手法を提案した [2]．ナイーブな逆 k 最近傍検索の処理を k 最近傍検索ステップと問合せステップの 2 つに分け，それぞれを GPU の並列処理によって高速化するものだ．k 最近傍ステップでは，処理そのものを大規模クエリに対する k 最近傍検索と見立て，距離計算の順序を入れ替えたり，バッチ処理を組み合わせたりすることで，すべての処理を GPU 上で処理する．問合せステップでは，アルゴリズムを GPU のアーキテクチャに最適化し，複数のクエリを同時に処理する．これにより，次元数やデータ数の影響を抑え，CPU のシングルスレッドやマルチスレッドで実行した場合よりも高速に処理することができる．本研究では，この手法を逆 k 最近傍検索の部分で使用する．

次に，3 節で述べた 3 種類の特徴選択のアプローチのうち，ラベル情報が不要なフィルター手法の関連研究について説明する．フィルター手法の中でも，まず単変量と多変量の手法が存在する．単変量特徴選択とは，特徴のランキングを得るために，単一の特徴量を順次評価していき，最終的にランキングの上位にある特徴の部分集合を選択する手法である．代表的なものとして，SUD（Sequential backward selection method for Unsupervised Data）[8] が挙げられる．これは類似度行列から誘導される総エントロピーによって，特徴量を重み付けする手法である．一方，多変量特徴選択は，特徴の関連性を個別に評価するのではなく，複数の特徴を合わせて評価する．そのため，多くの場合，多変量特徴選択によって選択した特徴を用いた方が，機械学習において高い精度で学習することができる．したがって，多変量特徴選択手法について，より詳しく説明する．

多変量特徴選択は，大きく 3 つのグループに分けられる．1 つ目は統計・情報的手法である．これは，分散共分散，線形相関，エントロピー，相互情報量などの統計的・情報理論的な尺度を用いて選択を行う手法である．代表的な手法として，FSFS（Feature Selection using Feature Similarity）[9] が挙げられる．この手法では，分散共分散に基づいた MICI（Maximal Information Compression Index）という統計的な尺度を提案している．同じクラスターに属する特徴量は類似性が高いという観点から，MICI を用いて各特徴量の k 最近傍を計算し，最もコンパクトな（最近傍と k 最近傍との距離が最も近い）k 最近傍を持つ特徴量を選択する．また，RRFS（Relevance Redundancy Feature Selection）[10] と呼ばれる手法は，2 ステップに分けて行われる．最初のステップでは分散によって特徴量をソートし，次のステップで冗長性を測るために類似度を評価する．最終的に冗長性の低い l 個の特徴量を選択する．

2 つ目は生体模倣手法である．これは，ある基準を満たす特徴の部分集合を見つけるために，群知能パラダイムに基づ

く確率的探索戦略を使用する手法である．代表的な手法として，UFSACO（Unsupervised Feature Selection based on Ant Colony Optimization）[11] が挙げられる．この手法において，完全無向グラフを検索の対象としており，ノードは特徴量を表し，エッジの重みは特徴間の類似度を表す．グラフの各ノードはフェロモンと呼ばれる望ましさを意味する値を持ち，エージェント（蟻）によって選択されることで更新される．エージェントはフェロモン値が高く，類似度が低いノードを優先して走査し，最終的に最もフェロモン値が高い特徴が選択される．

3 つ目はスペクトル分析手法と呼ばれ，名前の通りスペクトル分析に基づいて特徴を評価し，選択する手法である．代表的な手法に mR-SP（minimum-Redundancy SPectral feature selection）[12] が挙げられる．これは SPEC（SPECtrum decomposition）[13] によるランキングと最小冗長最適化基準 [14] を組み合わせた手法である．SPEC は単変量特徴選択手法の 1 つで，データ間の類似性から誘導されるグラフの構造と整合性によって，特徴の関連性を評価するものである．つまり，mR-SP は SPEC に特徴の冗長性を制御する方法を加えたものであり，それにより性能を向上させている．本研究で採用する MCFS（Multi-Cluster Feature Selection）[3] もこの分類に含まれる．

5 提案手法

本研究では，特徴選択を用いて高次元データにおける逆 k 最近傍検索を効率化することを目的とする．手法の方針として，特徴選択の中でもスペクトル分析手法の 1 つである，MCFS [3] を採用する．MCFS は，データのクラスター情報を最も保存するような特徴量を選択するもので，比較的計算コストが小さく，精度が高いという特徴がある．l 個の特徴量を選択する場合，MCFS は以下の手順で行われる．

（1） k 最近傍グラフ（隣接行列）を生成する
（2） k 最近傍グラフからラプラシアン行列を生成する
（3） ラプラシアン行列の固有ベクトルを m 個求める
（4） それぞれの固有ベクトルについて，LARs アルゴリズムにより係数ベクトル $\boldsymbol{\beta}$ を求める
（5） m 個の $\boldsymbol{\beta}$ において，各次元で最大の絶対値（MCFS スコア）を求める
（6） MCFS スコアを降順にソートし，上位 l 個のインデックスを選択する特徴量として取得する

次の項から各ステップについて説明する．

5.1 k 最近傍グラフの生成

k 最近傍グラフとは，それぞれのノード（データ点）の k 最近傍にエッジが張られたグラフのことである．Cai ら [3] は最近傍グラフを用いているが，本研究ではより多くの情報を取り入れるため，k 最近傍グラフを採用する．k 最近傍グラフは，一般的には隣接行列を用いて表現される．隣接行列とは，ノード i からノード j へエッジを張る場合にその要素 (i, j) が 1，そうでない場合に 0 となるような正方行列である．図 2 の上段は 2 最近傍グラフとそれに対応する隣接行列の例を表している．この図において，k 最近傍グラフは有向グラフであるため，その隣接行列は非対称行列になる．しかし，次に説明するラプラシアン行列は，元となる隣接行列が非対称だと定義することができない．そこで本研究では k 最近傍グラフを図 2 下段のように定義することで，無向グラフとして扱う．

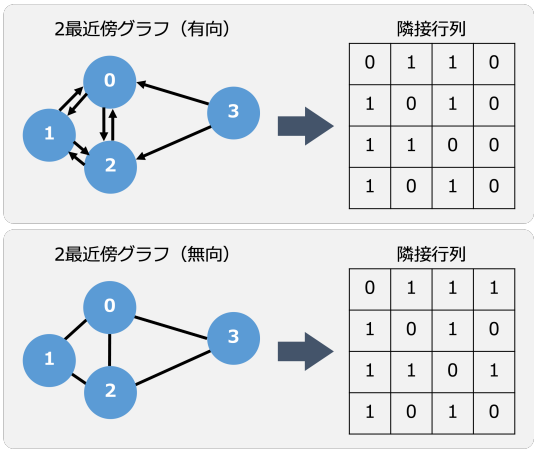

図 2　2 最近傍グラフとその隣接行列の例

隣接行列はノード（データ点）数の二乗のサイズになるため，データが大規模になるほどメモリコストが大きくなる．そこで本研究では，unsigned int 型を採用し，1bit を 1 つの要素として扱う．つまり，unsigned int 型の 1 要素で 32 個分の要素を格納できるようになり，unsigned char 型で 0/1 を格納する場合と比較してメモリコストを 1/8 に抑えることができる．

この部分は逆 k 最近傍検索において各データ点の k 番目の距離を計算する処理とほとんど同じである．したがって，我々の既存手法 [2] における k 最近傍ステップの実装が流用できるため，GPU で実行する．しかし，32 個のまとまりの中に複数のスレッドから書き込むと競合が発生してしまう．そこで，隣接行列を生成する際には CUDA [15] の atomicOr 関数を用いる．atomicOr 関数は，各スレッドがあるメモリ領域に対して排他的に OR 演算を行うものであり，これにより逐次的な処理を安全に実現する．一方で，同じメモリ領域に書き込む回数が多い場合，並列処理が阻害されるため，性能が著しく低下する可能性がある．本研究では隣接行列はスパースであり，大規模であることを想定しているため，この影響は小さいと考えられる．

5.2 ラプラシアン行列の生成

ラプラシアン行列 $\boldsymbol{L}$ とはグラフの行列表現の一種であり，以下のように定義される．

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$$

ここで，$\boldsymbol{D}$ はグラフの次数行列，$\boldsymbol{W}$ は隣接行列である．i 番目のノードを $\boldsymbol{v_i}$，ノード $\boldsymbol{v_i}$ の次数を $\deg(\boldsymbol{v_i})$ とすると，$\boldsymbol{L}$ の要素 $L_{i,j}$ は以下のように表される．

$$L_{i,j} = \begin{cases} \deg(\boldsymbol{v_i}) & (\text{if } i = j) \\ -1 & (\text{if } i \neq j \text{ and } \boldsymbol{v_i} \text{ is adjacent to } \boldsymbol{v_j}) \\ 0 & (\text{otherwise}) \end{cases}$$

本研究において，隣接行列 $\boldsymbol{W}$ には k 最近傍グラフ，次数行列 $\boldsymbol{D}$ には $\boldsymbol{W}$ の各行の非零要素数を要素に持つ行列を用いる．したがって，ラプラシアン行列はほとんどの要素が 0 である疎行列になる．そこで，メモリコストを削減するために，CSR（Compressed Sparse Row）形式でラプラシアン行列を保持する．

図 3 は行数 N の疎行列を CSR 形式で表記する例を表している．CSR 形式は疎行列を val，col_ind，row_ptr の 3 つの配列で表現する．val は非零要素を上から順に並べたもので，col_ind はその行におけるインデックスを表している．row_ptr は各行の非零要素の開始位置を示す．ただし，$N+1$ 番目は非零要素数を表す．非零要素数を nnz とすると，疎行列が int 型（32bit）で表され，col_ind と row_ptr が long long 型（64bit）の場合，CSR 形式によるメモリ消費量は $12nnz+8(N+1)$ となる．疎行列をそのまま int 型（32bit）で表す場合，メモリ消費量は $4N^2$ であるから，データが大規模になるほど CSR 形式によるコスト削減の効果は大きくなる．

実装する上では，ビット配列の隣接行列から直接 CSR 形式のラプラシアン行列を生成する．その際，線形走査によってビット配列にアクセスするため，並列化はできない．したがって，CPU で逐次的に実行する．

5.3 LARs アルゴリズム

次に，LARs（Least Angle Regression）アルゴリズム [16] について説明する．そもそもここでやりたいことは，ラプラシアン行列の固有ベクトルと最も関連するような特徴量の部分集合を見つけることである．そこで，d 次元の行ベクトルから成るデータセット $\boldsymbol{X}$ と i 番目の固有ベクトル $\boldsymbol{y_i}$ に対して，以下のように適合誤差を最小化するような係数ベクトル $\boldsymbol{\beta_i}$ を求める．

$$\min \parallel \boldsymbol{y_i} - \boldsymbol{X\beta_i} \parallel^2$$

しかし，これだけでは係数ベクトルがスパースにならず，特定の数だけ特徴量を選択することは困難である．そこで，以下のように L1 ノルム制約を加える．

$$\min \parallel \boldsymbol{y_i} - \boldsymbol{X\beta_i} \parallel^2 + \lambda \mid \boldsymbol{\beta_i} \mid$$

これは Lasso 回帰 [17] と呼ばれ，スパースな係数ベクトルを解として得ることができる．ところが，Lasso 回帰では原点で微分不可能な絶対値の項が含まれるため，解析的に解を求めることが困難である．そこで，以下のようにパラメータ l を係数ベクトル $\boldsymbol{\beta_i}$ の非零要素数として指定する形に変形する．

$$\min \parallel \boldsymbol{y_i} - \boldsymbol{X\beta_i} \parallel^2, \qquad \text{s.t.} \ \mid \boldsymbol{\beta_i} \mid \leqq l$$

これを解くために用いるのが LARs アルゴリズムであり，Lasso 解とほぼ同じ解を得ることができる．

LARs アルゴリズムを実行するにあたり，まずデータセットを以下のように基準化する必要がある．

$$\sum_{i=1}^{N} x_{i,j} = 0, \quad \sum_{i=1}^{N} x_{i,j}^2 = 1, \qquad \text{s.t. } j = 1, 2, ..., d$$

固有ベクトルについても以下のように中心化する．

$$\sum_{a=1}^{N} y_{a,i} = 0 \qquad \text{s.t. } i = 1, 2, ..., m$$

$\boldsymbol{\mu}$ を推定値（$\boldsymbol{X\beta_i}$）とすると，LARs アルゴリズムは以下の手順で行われる．

（1） $\boldsymbol{\beta_i}$，$\boldsymbol{\mu}$ を $\boldsymbol{0}$ で初期化する

（2） 最も $\boldsymbol{y_i}$ と相関が高い（内積値が大きい）特徴 $\boldsymbol{c_1}$ を選択する

（3） $\boldsymbol{\mu}$ を $\boldsymbol{c_1}$ の方向に移動させる（$\boldsymbol{\mu_1} = \boldsymbol{\mu_0} + \gamma \boldsymbol{x_{c_1}}$）

（4） まだ選択していない特徴と $\boldsymbol{c_1}$ の等分角ベクトル $\boldsymbol{u}$ を計算する

（5） 最も $\boldsymbol{u}$ と相関が高い（内積値が大きい）特徴 $\boldsymbol{c_n}$ を選択する

（6） $\boldsymbol{\mu}$ を $\boldsymbol{u_{c_n}}$ の方向に移動させる（$\boldsymbol{\mu_{n+1}} = \boldsymbol{\mu_n} + \gamma \boldsymbol{u_{c_n}}$）

（7） 条件を満たすまで 4 に戻る（ただし $\boldsymbol{c_1}$ を $\boldsymbol{c_n}$ とする）

ここで本研究における終了条件とは，(i) 選択した特徴量の数が l 個に到達する (ii) $\boldsymbol{\mu}$ が更新しなくなる のどちらかを満たす場合と定める．

図 4 は $\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}$ の 3 つの中から 2 つの特徴量を選択するときの例を示している．なお，すでに $\boldsymbol{y}$ と最も相関が高い特徴として $\boldsymbol{x_1}$ が選択されているものとする．まだ選択されていない $\boldsymbol{x_2}$ と $\boldsymbol{x_3}$ のそれぞれについて，$\boldsymbol{x_1}$ との等分角ベクトル $\boldsymbol{u_2}$，$\boldsymbol{u_3}$ を計算する．この例において，相関（内積）は $\boldsymbol{x_1}^\top \boldsymbol{u_2} > \boldsymbol{x_1}^\top \boldsymbol{u_3}$ となるため，$\boldsymbol{\mu_2}$ は $\boldsymbol{u_2}$ に沿って動かされる．このとき $\gamma_2 < \gamma_3$ であり，相関の高い等分角ベクトルを選択することは，より小さい γ を選択することに等しい．

LARs アルゴリズムは反復的に行われ，その回数は実行するまで不明である．したがって本研究では，固有ベクトルごとに並列に実行する．ただし，GPU で実行しようとすると，それぞれのブロックで独立に保持しなければならない配列が多すぎて，シェアードメモリに収めることができない．グローバルメモリに書き込むにしても，頻繁に読み書きが発生するため，大きなタイムロスになってしまうことが想定される．そこで，CPU のマルチスレッドによって実行する．

5.4 MCFS スコア

MCFS スコアとは，LARs アルゴリズムによって求めた係数ベクトルから実際に選択する特徴を決めるための指標である．j 番目の特徴量の MCFS スコアは以下のように定義される．

$$\mathrm{MCFS}(j) = \max_i \mid \beta_{i,j} \mid$$

ただし $\beta_{i,j}$ は係数ベクトル $\boldsymbol{\beta_i}$ の j 番目の要素を表す．

全部で d 個の MCFS スコアを降順にソートし，上位 l 個を取得することで，特徴選択が完了する．

係数ベクトルはサイズが小さいため，GPU で処理する場合，データ転送等の計算以外にかかる時間の割合が大きくなってしまうことが想定される．したがって本研究では，MCFS スコアの計算は CPU のマルチスレッドによって実行する．

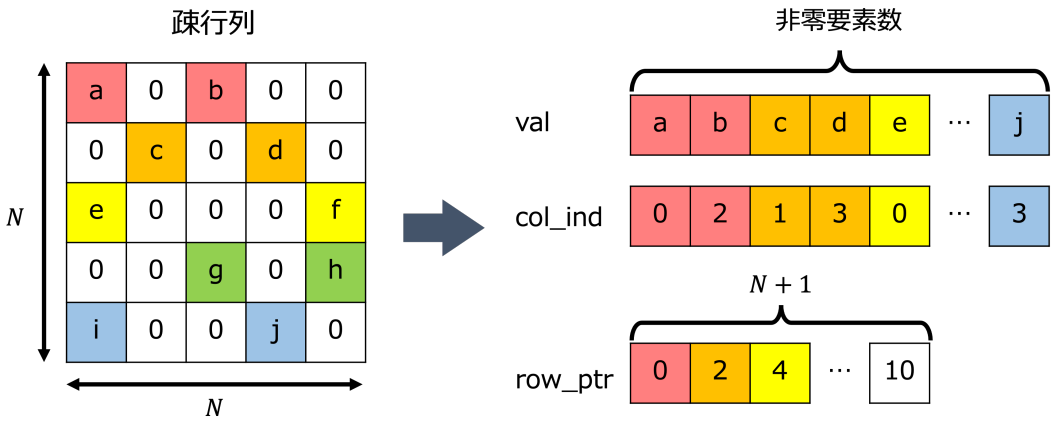

図 3　疎行列の CSR 形式表現

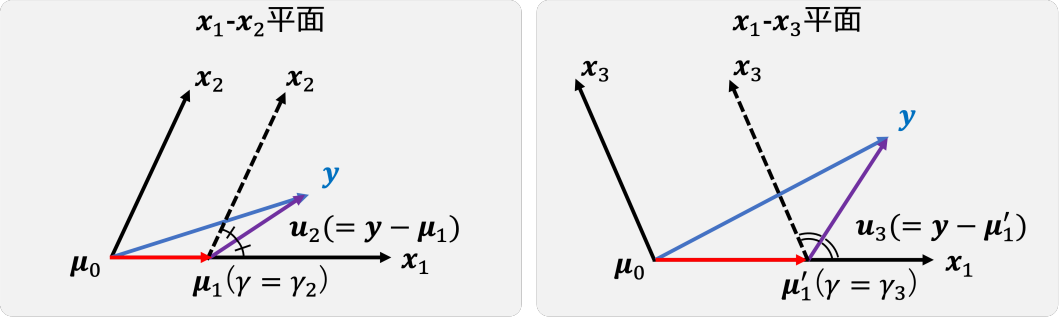

図 4　LARs アルゴリズムの例

6　評 価 実 験

実際に提案手法を実行し，その性能を評価する．データセットには正規分布に従う乱数データを用いる．d 次元のうち上位 1/3 に同じ平均値の正規分布に従う乱数データが入っており，残りにはノイズとして $[-100, 100]$ の範囲で一様分布に従う乱数データが入っている．正規分布に従う部分は 5 つのクラスターを持つように設定されており，それぞれ $-20, -10, 0, 10, 20$ を平均値とし，分散は 1 である．

固有ベクトルの数 m はデータセットのクラスターの数と等しいことが望ましい．したがって本実験では，$m = 5$ とする．また，選択する特徴の数 l は，明記しない限り元の次元数 d に対して $d/4$ とする．

評価指標としては，実行時間と適合率（Precision）を用いる．実行時間に関しては，特徴選択でどの処理にどのくらい時間がかかっているかと，特徴選択を使用した場合の逆 k 最近傍検索における実行時間の変化を確認する．適合率とは，検索の精度を表す指標で，以下のように定義される．

$$\text{適合率} = \frac{\text{検索結果に含まれる正解の数}}{\text{検索結果の数}}$$

本実験では，すべての次元を扱うことで本質的な解を見つけにくくなってしまうことを考慮し，特徴選択によってその問題を解決できるかどうかを評価するために，適合率を使用する．なお，本実験で扱うデータセットはラベル情報を持たないため，同じクラスターに属するデータを正解として扱う．また，逆 k 最近傍検索には我々の GPU を用いた既存手法 [2] を使用する．

本実験は，CPU に Intel Xeon(R) Bronze 3104 @ 1.70GHz, GPU に NVIDIA Tesla P100 を搭載する計算機で実行される．メインメモリは 32GB で GPU のグローバルメモリは 12GB である．OS は CentOS 7.9 で，g++ (GCC) 8.5.0 と CUDA 11.0 によってコンパイルされる．マルチスレッド処理の実装には OpenMP を用いており，最大スレッド数は 12 である．

固有ベクトルの計算には Python3.8 および SciPy [18] を用いる．SciPy は情報科学などのための Python ライブラリである．本実験では，数多くある SciPy パッケージの中でも線形代数に関連する linalg と疎行列に関連する sparse を利用する．また，LARs アルゴリズムの実装には Eigen [19] を用いる．Eigen は線形代数のための C++テンプレートライブラリである．行列積や逆行列の計算などを高速に処理することができ，マルチスレッドにも対応している．

6.1　特徴選択の実行時間

図 5 は異なる次元数における特徴選択全体の実行時間を示している．つまり，GPU による k 最近傍グラフ（隣接行列）の生成と CPU の逐次処理による CSR 形式のラプラシアン行列の生成，Python による固有ベクトルの計算，マルチスレッドによる LARs アルゴリズムおよび MCFS スコアの計算が含まれる．ただし，データ数は 1 万件で固定してある．図から，128 次元までは実行時間がほぼ一定であるのに対し，それ以降増加していることがわかる．それぞれの内訳は表 1 のようになっている．

表 1　異なる次元数における特徴選択の実行時間の内訳

d	k 最近傍グラフ	ラプラシアン行列	固有ベクトル	LARs アルゴリズム	MCFS スコア
4	3837.450 (0.71)	835.315 (0.15)	691.226 (0.12)	12.190 (0.01)	0.012 (0.00)
8	3900.054 (0.66)	838.156 (0.14)	1098.046 (0.18)	22.414 (0.02)	0.011 (0.00)
16	3908.538 (0.71)	847.027 (0.15)	642.072 (0.11)	51.596 (0.02)	0.015 (0.00)
32	3953.726 (0.73)	852.350 (0.15)	416.788 (0.07)	124.958 (0.02)	0.019 (0.00)
64	3914.263 (0.70)	845.523 (0.15)	363.931 (0.06)	458.570 (0.08)	0.021 (0.00)
128	3869.813 (0.53)	847.514 (0.11)	377.801 (0.05)	2071.801 (0.28)	0.053 (0.00)
256	3887.495 (0.23)	846.982 (0.05)	393.714 (0.02)	11500.230 (0.69)	0.083 (0.00)
512	3690.832 (0.04)	848.581 (0.01)	430.563 (0.00)	75496.944 (0.93)	0.214 (0.00)
1024	3883.487 (0.00)	848.257 (0.00)	418.022 (0.00)	555911.981 (0.99)	0.784 (0.00)
2048	4334.518 (0.00)	854.733 (0.00)	443.612 (0.00)	4539993.597 (0.99)	2.789 (0.00)

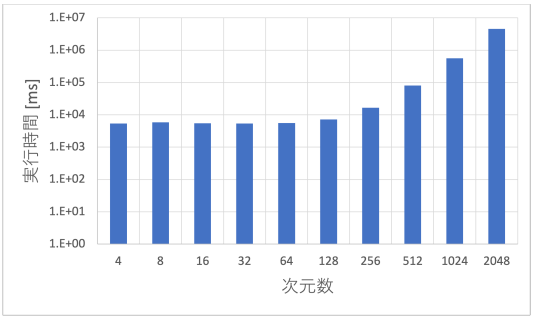

図 5　異なる次元数における特徴選択の実行時間

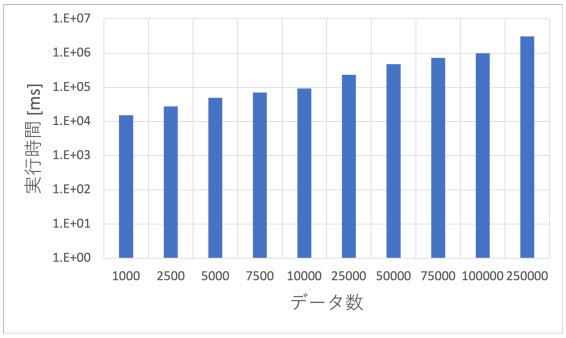

図 6　異なるデータ数における特徴選択の実行時間

表 1 において，左側の数字は実行時間（単位はミリ秒），括弧内の数字はそのときの全体に占める割合（小数第 3 位以下は切り捨て）を表しており，最も割合の大きいところを赤字で示している．この表から，k 最近傍グラフの隣接行列およびラプラシアン行列の生成と固有ベクトルの計算にかかる時間はほぼ一定であるが，次元数の増加に伴って LARs アルゴリズムと MCFS スコアの計算にかかる時間は増加していき，特に LARs アルゴリズムは増加の割合が大きいことがわかる．k 最近傍グラフの隣接行列を生成する部分では，GPU の並列処理によって次元数の増加の影響を受けにくくなっている．ラプラシアン行列は k 最近傍グラフの隣接行列から生成されるため，次元数の影響は受けない．同様に固有ベクトルはラプラシアン行列から生成されるため，次元数の影響は受けない．LARs アルゴリズムはデータセットの次元数および選択する特徴数によってイテレーション回数が変化するため，その回数が増えるほど実行時間が爆発的に増加してしまう．MCFS スコアの計算ではマルチスレッドによって並列に係数ベクトルを走査しているが，実験環境における最大スレッド数は 12 であるため，次元数の増加に伴って実行時間も増加してしまう．しかし，特徴選択の処理全体から見たときに，MCFS スコアの計算の実行時間は無視できるといってよい．

次に，データ数を変化させて特徴選択にかかる時間を計測する．

図 6 は異なるデータ数における特徴選択全体の実行時間を示している．ただし，次元数は 1024，選択する特徴数は 128 に固定してある．図から，特徴選択にかかる実行時間はデータ数におよそ比例することが確認できる．それぞれの内訳は表 2 のようになっている．

表 2 において，左側の数字は実行時間（単位はミリ秒），括弧内の数字はそのときの全体に占める割合（小数第 3 位以下は切り捨て）を表しており，最も割合が大きいところは赤字で示している．この表から，MCFS スコアの計算にかかる時間はほぼ一定であるが，それ以外は次元数の増加に伴って実行時間も増加していき，特に LARs アルゴリズムの占める割合が大きいことがわかる．しかし，データ数が大きくなると徐々にその割合は下がっていき，逆に k 最近傍グラフの隣接行列およびラプラシアン行列の生成にかかる時間の割合が増えていく．これは k 最近傍グラフの隣接行列およびラプラシアン行列がデータ数の二乗のサイズになるために，前者ではバッチ数が増え，後者では走査する範囲が増えるからであると考えられる．ラプラシアン行列のサイズが大きくなるため，それに応じて固有ベクトルの計算にかかる時間も長くなる．LARs アルゴリズムでは，データ数はイテレーションの回数に影響しないが，各イテレーションで行われる行列演算の計算量が大きくなるため，実行時間がデータ数にほぼ比例する．特徴選択全体の実行時間がデータ数にほぼ比例していたのは，測定した範囲だと LARs アルゴリズムの割合が大きいためであると考えられる．MCFS スコアはデータ数に依存しないため，一定となる．

6.2　逆 k 最近傍検索への影響

まずは，特徴選択の有無による実行時間の違いを調査する．図 7 は同じ 1024 次元のデータにおいて，そのまま逆 k 最近傍検索を行ったときの実行時間と，特徴選択によって 256 次元に削減してから逆 k 最近傍検索を行ったときの実行時間を比較し

表 2　異なるデータ数における特徴選択の実行時間の内訳

N	k 最近傍グラフ	ラプラシアン行列	固有ベクトル	LARs アルゴリズム	MCFS スコア
1000	2706.211 (0.17)	8.938 (0.00)	19.300 (0.00)	12432.001 (0.81)	0.292 (0.00)
2500	2695.038 (0.09)	51.741 (0.00)	31.602 (0.00)	24897.711 (0.89)	0.288 (0.00)
5000	2859.591 (0.05)	202.633 (0.00)	48.882 (0.00)	46033.131 (0.93)	0.249 (0.00)
7500	3099.352 (0.04)	455.289 (0.00)	79.262 (0.00)	67155.547 (0.94)	0.256 (0.00)
10000	3883.487 (0.04)	848.257 (0.00)	473.000 (0.00)	88046.057 (0.94)	0.264 (0.00)
25000	8500.348 (0.03)	5282.176 (0.02)	1009.207 (0.00)	214328.937 (0.93)	0.263 (0.00)
50000	22059.067 (0.04)	20498.496 (0.04)	2221.062 (0.00)	423656.438 (0.90)	0.267 (0.00)
75000	42777.463 (0.05)	45520.122 (0.06)	4071.663 (0.00)	633097.004 (0.87)	0.260 (0.00)
100000	69208.189 (0.06)	80535.817 (0.08)	5610.000 (0.00)	836833.592 (0.84)	0.250 (0.00)
250000	443058.812 (0.14)	503366.183 (0.16)	20062.113 (0.00)	2112143.366 (0.68)	0.258 (0.00)

たものである．なお，データ数が 50 万件を超える場合には，本実験環境ではメモリの都合により特徴選択を行うことができないため，10 万件のときの特徴選択の結果を使用する．

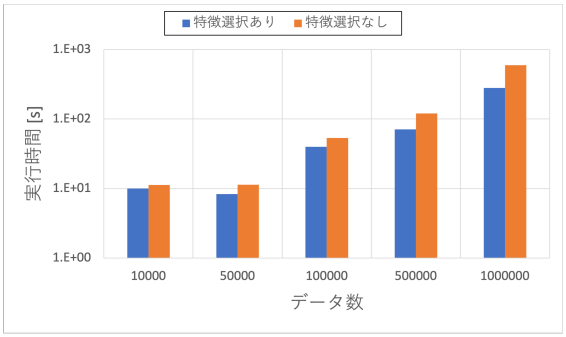

図 7　特徴選択の有無による逆 k 最近傍検索の実行時間

図 7 から，特徴選択を行うことによって実行時間が短くなっていることが確認できる．これはデータの分布によらない逆 k 最近傍検索において，次元数が減ることにより計算コストが減少することから明らかである．また，特徴選択の有無による逆 k 最近傍検索の実行時間の差は，データ数が増加する度に大きくなっていく．データ数が 100 万件で特徴選択を行ったときの実行時間は，そうでないときの約 2.1 倍高速であった．

次に，特徴選択の有無による精度の違いを調査する．図 8 は同じ 1024 次元のデータにおいて，そのまま逆 k 最近傍検索を行ったときの適合率と，特徴選択によって 256 次元に削減してから逆 k 最近傍検索を行ったときの適合率を比較したものである．ただし，k は 100 で固定してある．なお，ここでは一様分布によるノイズを取り除き，0 で埋めたデータに対しても特徴選択および検索を行った．

図 8 から，ノイズがある場合では特徴選択をしないときの方が適合率は高くなっていることがわかる．一方で，ノイズがない場合では特徴選択をしたときの方が適合率は高くなっている．このことから，MCFS はノイズの影響を受けやすいと考えられる．この原因として，高次元データのユークリッド距離を計算する時点で距離集中が発生し，本質とは異なった k 最近傍グラフを生成してしまうことが挙げられる．MCFS はラプラシアン行列の固有ベクトルを基底と見なして適合誤差を最小化するため，そのラプラシアン行列が本質とは異なった k 最近傍グラフ

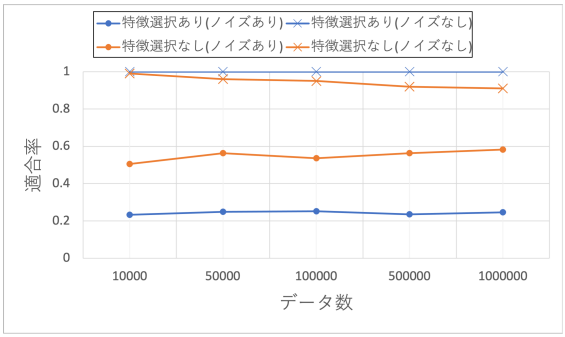

図 8　特徴選択の有無による適合率

に基づいて生成されてしまうと，係数ベクトルも間違って学習してしまう．

また，データ数が増加しても適合率はほぼ一定であることから，データセットが同じ分布をしている場合，サンプリングによって特徴選択の計算コストを削減することが有効だと考えられる．

7　ま　と　め

本研究では，特徴選択手法である MCFS を用いて逆 k 最近傍検索を効率化する手法を提案した．提案手法では，MCFS を GPU や CPU のマルチスレッドなどにより高速化し，大規模で高次元のデータから特徴選択をできるようにした．大規模データへの対応にあたり，k 最近傍グラフの隣接行列をビット配列で保持したり，ラプラシアン行列を CSR 形式で保持したりすることにより，メモリコストを削減した．

実験により，特徴選択自体には時間がかかるものの，前処理として特徴選択を用いることによって逆 k 最近傍検索で 1000 次元を超える高次元データも扱えるようになることを示した．ただし，本手法はノイズの影響を受けやすく，データによっては精度が低下してしまう可能性がある．

今後の方針としては，実データの使用や，他の特徴選択手法を採用することが考えられる．実験では正解データのない乱数データのクラスター情報から近似的に適合率を計算しており，それが真に逆 k 最近傍であるかどうかは保証されていない．さらに，ノイズの有無によって大きく適合率に差が生じたことからも，実データを用いた実験が必要だと考えている．また，本

手法で用いた MCFS および LARs アルゴリズムでは，反復処理や使用する配列の多さから，GPU によって並列処理をすることができない．並列化が可能で，ノイズに強い特徴選択手法があれば，GPU によって高速に処理ができ，逆 k 最近傍検索でより高い精度を示すことができると考えている．

文　献

[1] F. Korn and S. Muthukrishnan. Influenced sets based on reverse nearest neighbor queries, SIGMOD Rec., vol.29, no.2, pp.201-212, May 2000.

[2] Tsuihiji K., Amagasa T., GPU-Accelerated Reverse K-Nearest Neighbor Search for High-Dimensional Data, In International Conference on Network-Based Information Systems, NBiS 2022, Lecture Notes in Networks and Systems, vol 526, Springer, Cham, pp.279-288, 2022.

[3] Cai D., Zhang C. and He X., Unsupervised feature selection for multi-cluster data, In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp.333-342, July 2010.

[4] I. Stanoi, D. Agrawal, and A. E. Abbadi, Reverse nearest neighbor queries for dynamic databases, ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp.44-53, 2000.

[5] W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan, Finch: Evaluating reverse k-nearest-neighbor queries on location data, Proc. PVLDB Endow., vol.1, no.1, pp.1056-1067, Aug 2008.

[6] S. Yang, M. A. Cheema, X. Lin, and Y. Zhang, Slice: Reviving regions-based pruning for reverse k nearest neighbors queries, IEEE 30th ICDE, pp.760-771, 2014.

[7] A. Singh, H. Ferhatosmanoglu, and A. Tosun, High Dimensional Reverse Nearest Neighbor Queries, Proc. 12th Int. Conf. Inf. Knowl. Manage., pp.91-98, 2003.

[8] Dash M., Liu H., and Yao J., Dimensionality reduction of unsupervised data, In Proceedings ninth ieee international conference on tools with artificial intelligence, pp. 532-539, IEEE, November 1997.

[9] Mitra P., Murthy C.A. and Pal S.K., Unsupervised feature selection using feature similarity, IEEE transactions on pattern analysis and machine intelligence, 24(3), pp.301-312, 2002.

[10] Ferreira AJ, Figueiredo MA, An unsupervised approach to feature discretization and selection, Pattern Recognition 45(9), pp.3048–3060, 2012.

[11] Tabakhi S., Moradi P. and Akhlaghian F., An unsupervised feature selection algorithm based on ant colony optimization, Engineering Applications of Artificial Intelligence, 32, pp.112-123, 2014.

[12] Garcia-Garcia D, Santos-Rodriguez R, Spectral clustering and feature selection for microarray data, In International conference on machine learning and applications, 2009 ICMLA ’09, pp 425–428, 2009.

[13] Zhao Z. and Liu H., Spectral feature selection for supervised and unsupervised learning, In Proceedings of the 24th international conference on Machine learning, pp.1151-1157, June 2007.

[14] Peng H, Long F, Ding C, Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy, IEEE Trans Pattern Anal Mach Intell, 27(8), pp.1226-1238, 2005.

[15] CUDA, `https://developer.nvidia.com/cuda-toolkit`.

[16] Efron B., Hastie T., Johnstone I. and Tibshirani R., Least angle regression, The Annals of statistics, 32(2), pp.407-499, 2004.

[17] Tibshirani R., Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society: Series B (Methodological), 58(1), pp.267-288, 1996.

[18] SciPy, `https://scipy.org`

[19] Eigen, `https://eigen.tuxfamily.org/index.php?title=Main_Page`