kNN や ε NN に対する新しいハッシュ

宇野 毅明　情報研 & 総研大
津田 宏治　産総研 & ERATO
田部井 靖生　ERATO

動機

・ キーワード探索（正確な一致）は、2分担策の技術で高速にできる
・ 一方で、あいまい性を許し、「距離がk番目以内」や「距離が ε 以内」のものを見つけようとすると、とたんに難しくなる。
　←問題を「分割」するのが難しいから
・ 検索の問題ととらえ、上手なインデックスを作ろう、という研究が中心
・ 主な方法は、LSHのようなハッシュを使う方法と、木を作り、その上を縦横無尽に探索する方法がメイン。

・ ハッシュは、「距離が近いと同じ値を取る確率が高い」ようなハッシュをデザインし、ハッシュ値が同じものとのみ比較して、計算を省略する。
・ 基本的に乱択で、精度を上げるにはハッシュの枚数増加が必要
・ ε 近傍探索に有効。K近傍には役に立たないこともある（データが一点に固まっていると、距離 ε 以下のものが大量にあることになり、パフォーマンスがでないため）

・木型のものは、データを再帰的に分割して作るような木を作り、その上を探索する。普通は子供を一人選んでそこを調べるが、子供を複数選ぶことがある。
・厳密な探索ができるのが良いが、冗長な探索をするため遅い

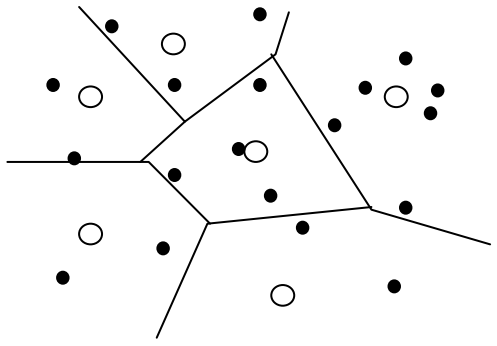
木探索型の手法のうまさを使った、新しいハッシュが作れたらなあ

新しいハッシュ

[アイディア]
LSHなどのハッシュは、ランダムにこさえた何かでハッシュを作っている。
対して、木型のほうは、データを元にして構造を作っている
　➔ データを使ってハッシュを作ろう

・データから幾つかの項目をサンプリングする。
その、どれに一番近いか、でハッシュを作ればいいだろう

・ 空間を「どれに一番近いか」で分割しているので、ボロノイ図と同じ
　➔ ボロノイハッシュ、と名付ける



・ 局所的に要素が集中しているところからは、複数の点がサンプリングされるはずなので、ほとんどのデータが同じ値を持つ、ということはなくなるだろう

設計と実験　（kNN）

・ サンプル点の数を増やすほど、精度が上がる
・ 特に、h個のペアを使って 2^h 種類の値を持つハッシュを作るより、2^h 個の要素をサンプリングしたほうが、ずっと精度がいい
・ 再帰的に分割する、木構造の方法でハッシュすると、精度が悪い
・ 計算時間とのトレードオフを考えると、サンプル数20、くらいのものを複数組合せて、同一のハッシュ値を持つものの数が k になるくらいに調整するのがよさそう
・そのハッシュを複数個 (30個) 作ると精度が上がる
・ミスマッチを1許すほうが、計算時間がだいぶ速くなる
・見つけ損ないの平均ランクは、ちょうど半分。遠いものほど見つけ損ないやすい、ということはないようだ
・それぞれの要素に対する見つけ損ないの数は、指数分布のようだ

・ユークリッド距離・コサイン距離の場合、100万要素のデータで、40倍ほどの加速（70時間→2時間。誤差1/1000程度, Corei7 3GHz）
・ 誤差を1割にすると、100倍くらいの加速（40分）
・l1-ノルム、l∞ノルムでは、少し性能が落ちるようだ

設計と実験（完全）

・完璧なハッシュは、やはりというか、遅い。全対比較のほうが速い
・ランダマイズドバージョンは結構良い。精度保証ができ、かつ実際にもだいたいその精度が出る
・内積計算が必要ない分は速いが、あまり精度が出ない。
・ミスマッチを1に設定するのがいいようだ

完全なハッシュ（別の話）

・ユークリッド距離に対する LSH は、ランダムに作った単位ベクトル a との内積値を離散化してハッシュを作る（距離が ε 以下のものを見つけるなら、内積値を hε で割る（hは整数））
　➔ 距離が近ければ、同じ離散値を取る可能性が高い（失敗確率 1/h 以下）

・ここで、離散化のしかたをちょっと変えたものを幾つか作ってみる
内積値に ε ,2ε ,…hε を足したもの、で h個のハッシュ値を作る
　➔ 距離が ε 以内なら、必ずh-1個のハッシュ値が等しくなる
　➔ h個のハッシュのハミング距離が 1 以下のもののみを比較すれば、距離 ε 以下のものは必ず見つかる

・この「h個のハッシュ」を複数個用意して「全てでハミング距離が1以下のもののみ」比較することで高速化する

・ベクトルをランダムに選ぶ必要がないので、軸方向のベクトルを選べばよい（内積の手間が無くなる。けっこう重要）

・ランダマイズド手法も作れる。軸に直交するベクトルとの内積がある程度以下になる確率が90%以上、という数値を計算して、それの定数倍の幅をとってハッシュを作る。（xi を 定数 b で割った商がハッシュ値）
（複数枚のハッシュを kNN と同じように組合わせる）

・サンプルする次元数 → 1つのブロックの中のハッシュ数 → 定数 b の順番に定め、精度を保証する