universität wien
Fakultät für Informatik

How do you use LSH for k-Means?

How to use LSH for k-Means

We have:

n Points distributed in $\mathbb{R}^d$

We want:

Distribute the data points into k cluster while minimizing the distance within a cluster

We can:

We can speed up the algorithm using LSH

How to use LSH for k-Means

Iterations in k-means need lots of distance calculations

$\Rightarrow$ How can we avoid as many as possible?

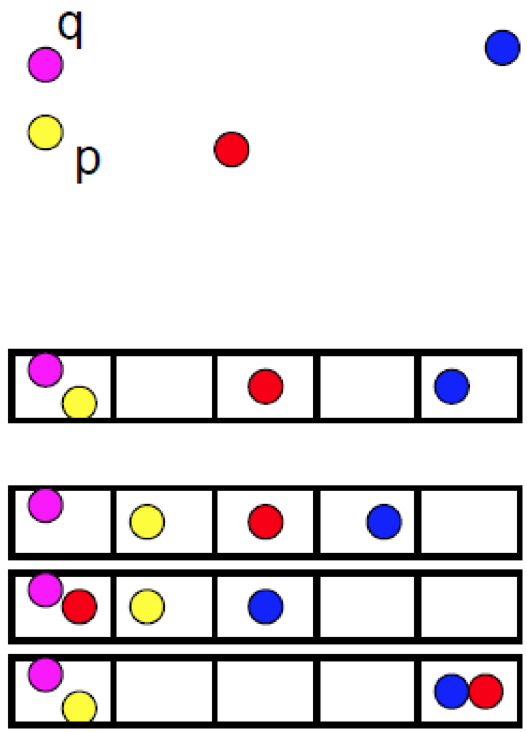
We only calculate the distance for those pairs of points that are (most likely) very close to each other

For most distance calculations: We don‘t calculate the exact distance but check out hash-values.

How to use LSH for k-Means

Idea: We construct hash-functions $g:\mathbb{R}^d \rightarrow \mathbb{N}$, so that for every *p, q* holds:

- if ||p-q||<r, than P[g(p)=g(q)] big
- if ||p-q||>r, than P[g(p)=g(q)] small



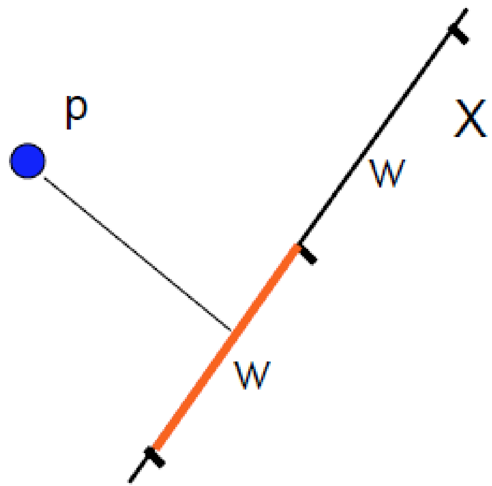
We can solve the problem using hash-functions

=> Which ones do we use?

How to use LSH for k-Means

Projection of the points:

It‘s possible to project the points onto one dimension → there: *buckets* of length *w* to get hash-values



How exactly?

Projection of d-dim vector onto 1 dimension:

Choose v d-dim with v_i $N(0,1)$-distributed, than the projection $v.p$, i.e. $\sum p_i v_i$, is $||p||*N(0,1)$-distributed

=> „Sketch“ of p

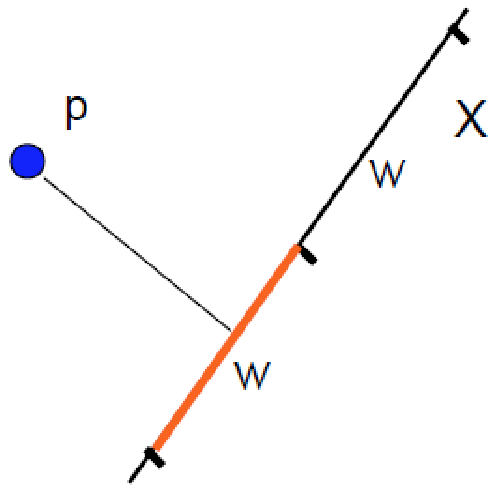
How to use LSH for k-Means

universität wien
Fakultät für Informatik

Example:

We have p=(2,4,5,2)

Hash-function: v=(-0.09,0.15,0.33,-0.51)



$$\Rightarrow v.p = \sum p_i v_i = 1.05$$

If w_1=[0,1), w_2=[-1,0), w_3=[1,2), …

p would be in the third *bucket* with the associated hash-value

Note: Can take the intervals mod(n)

Note: Possible to have only $\mathbb{R}^+/\mathbb{R}^-$ as a *bucket*

How to use LSH for k-Means

It holds:

$p.v\text{-}q.v = (p\text{-}q).v$, and hence $\|p\text{-}q\| N(0,1)$-distributed
=> If points are close to each other, than their projections will also be close
=> It is possible to show: The probability of two vectors to be in the same *bucket* grows monotonous with decreasing distance.

Hash-functions: $h_{a,b}(p)$: $\mathbb{R}^d \rightarrow \mathbb{N}$ is a map into the whole numbers with $a \in \mathbb{R}^d$ and b $\in [0,w]$

$h_{a,b}(p) = \lfloor (a.p+b)/w \rfloor$

=> Arbitrary big/small *buckets* possible

=> They determine the probability r to be in the same *bucket*

How to use LSH for k-Means

universität wien
Fakultät für Informatik

A single hash-function will not be enough:
→ Multiple hash-values

Combination of hash-values: AND/OR

AND: 2 hashvalues h_1 and h_2 of *p* and *q*: If we want both points to be identical in both hash-values, thus $h_1 \wedge h_2$, then P[g(p)=g(q)] drops down to r^2

OR: If we want $h_1 \vee h_2$, than *p* and *q* have to be equal in only one *bucket* → $P[g(p)=g(q)] = 1-(1-r)^2$

How to use LSH for k-Means

Combination of AND/OR:

We assume: 4x4 hash functions, combine 4 with AND and those blocks with OR
=> $f(r) = 1-(1-r^4)^4$

=> Same form as the S-curve – Rows ≈ AND, Bands ≈ OR

⇒ Arbitrary optimization quality possible, the more hash functions the better is the approximation quality, i.e. the less false positives and the less false negatives

⇒ **But: computational and storage overhead**

How to use LSH for k-Means

Uses for k-Means:

Calculate the hash-values for all points at the start and save them.

In each iteration→ Hash the centers, i.e. compute their hash bucket. Assign all points in this bucket to the center. Compute the full distances only for the remaining points.

How to use LSH for k-Means

Some ideas for a more radical usage:

For the remaining buckets: Compute the similarity of a single or a few random point to all centers. Assign all the points of the bucket to that center.

Update the centers only from the bucket of the center plus the randomly selected points from the other buckets (one or few per bucket).

Literature

LSH for text data:

Chapter 3 of the Book „Mining Massive Datasets“ by J. Leskovec, A. Rajamaran and J. Ullman, Availabe for download at http://www.mmds.org/

Introduction to LSH for vector data:

Locality-Sensitive Hashing Scheme Based on p-Stable Distributions (by A. Andoni et al), in the book Nearest Neighbor Methods in Learning and Vision: Theory and Practice, by T. Darrell and P. Indyk and G. Shakhnarovich (eds.), MIT Press, 2006

http://theory.lcs.mit.edu/~indyk/nips-nn.ps