kNN - local weighting and efficiency in high-dimensional spaces and large datasets

David Fiedler
Artificial Intelligence Center, Department of Computer Science, Faculty of Electrical Engineering, CTU in Prague, Czech Republic
Outline

- Quick introduction to kNN
- Problem of the data structure locality
  - Problem
  - Princip and Analysis
  - Results
  - Conclusion and Further reading
- High-dimensional spaces and large datasets
  - Problem
  - Princip and Analysis
  - Results
  - Conclusion and Further reading
kNN - quick introduction

Algorithm:

1. Find \textbf{k} nearest neighbours of the sample in the training dataset
2. Then:
   a. Classify the sample according to weighted majority of the neighbours - \textit{kNN classification}
   b. Compute the value of the sample as a weighted average of the neighbours - \textit{kNN regression}
kNN - properties

- Very simple :)
- *Instance-based learning* - sample is classified directly from the training data ie - there is no model.
- *Lazy learning algorithm* - all computation is in the classification step
- Very sensitive to local structure of the data - most samples are ignored in the testing phase
Motivation

- Most kNN methods choses the same $k$ for the whole dataset
- This can work well if the data are homogenous
  - Similar class sizes
  - Similar placement of the points inside classes

http://mnemstudio.org/clustering-k-means-introduction.htm
Motivation

- But what if the size of the classes differs?
Idea

- New idea - choose different $k$ for each sample

(a) First scenario  
(b) Second scenario  
(c) Third scenario

[Anava and Levy, 2016]
Idea - Main problems

- Do not increase computation time significantly
- Yield confidence bounds

Minimal assumptions

- The real/underlying model is a Lipschitz function: for each $x,y$, there is a constant $K$ where $|f(x) - f(y)| \leq K|x - y|$
- Labels of the datapoints are independent
For each sample:

● Choose the **optimal number of nearest points** - Two extremes:

  - Big Lipschitzness (low noise) -> we use only nearest point
  - Lipschitzness goes to 0 (high noise) -> We use all points

  How to choose $k$ (the number of points) for a dataset with an arbitrary $K$ (Lipschitzness)?

● Choose the **optimal weights**: How?
Solution

A Greedy algorithm is presented as a solution:

- The decrease of the computed weight of the neighbours is slower than in classical kNN
- There is a stopping condition that triggers when the solution is optimal with high probability
  - Probability definition
  - Exploits Lipschitz to noise ratio to determine the optimal $k$
- Similar cost to kNN
  - Computing the costs: $nd$
  - Sorting: $n \log n$
  - Algorithm itself: $k^*$
Results

- New algorithm outperforms classic kNN in 7 datasets from 8
- The number of nearest neighbours was set optimally in all classic kNN scenarios, based on the best results from the learning phase

![Table showing performance comparison](image)

[Anava and Levy, 2016]
Current kNN only consider distance between points. Could we utilize the geometric properties in the future?

- Something like capsule networks in neural network area?
High dimensional spaces and large datasets
Motivation

We determine the class of the sample from its $k$ nearest neighbours.

However, to choose the nearest neighbours, we have to compute the distance of the sample from all points in the training dataset.

- Dataset can have a lot of samples!
- The dimension of the data can be high!
Intuitive speedup - k-d tree

- Only returns the k-nearest neighbours, no need for sorting - complexity $O(n) \rightarrow O(\log n)$
- When the number of dimensions is high (close to $n$), the improvement over the linear search is very low

Approximate nearest neighbour (ANN)

- Idea: find the k probably nearest neighbours
- Less time needed at the cost of less accuracy
- Solves the problem of k-d tree with high dimensionality
- Common methods:
  - Locality Sensitive Hashing
  - Best Bin First
Save the datapoints to a hashtable
Contrary to the classic hashing, here we try to maximize the probability that similar data are saved in the same row
After hashing, we compute the distance only to the points with the same hash
ANN - Dimensionality reduction

- We can reduce the dimensionality of the data in a preprocessing step
- Principal Component Analysis (PCA) is a standard way to do that
- PCA finds the bases of the new space so that the error of the projection is minimal

http://www.nlpca.org/pca_principal_component_analysis.html
1. Divide the training dataset using the clustering
2. Find the appropriate cluster for the sample
3. Chose the $k$ nearest neighbours from the cluster
4. Classify according to the weighted majority of the neighbours
First we choose the clustering method

*Landmark-based spectral clustering* [Cai and Chen]

- Properties:
  - Low complexity compared to ordinary Spectral clustering -> linear scaling with respect to data
  - Keeps the properties of the spectral clustering
Solution - Training

- Landmark-based spectral clustering principe:
  - Create new points to represent the data - *landmarks*
    - Landmarks are computed using k-means
  - Apply the *spectral clustering* [Luxburg]
Solution - Training - Spectral Clustering

- Focused on connectivity rather than on compactness
- It can create non-convex clusters

https://www.cs.cmu.edu/~aarti/Class/10701/slides/Lecture21_2.pdf
Solution - Training - Spectral Clustering

- Algorithm:
  - Start with a similarity matrix \( A \) where \( A_{i,j} \) is similarity between \( x_i \) and \( x_j \)
  - Create a Laplacian Matrix \( L \) from \( A \)
  - Compute \( k \) eigenvectors \( V \) of \( L \)
  - Build matrix \( U \) from \( V \) as columns
  - Interpret the rows of \( U \) as the original data points and cluster them using k-means - projecting into spectral dimension, dimensionality reduction
Solution - Training - Spectral Clustering

Dimension is future count

Original data

Dimension is the chosen number of eigenvectors

Projected data

https://www.cs.cmu.edu/~aarti/Class/10701/slides/Lecture21_2.pdf
Solution - Testing

1. Find the appropriate cluster for the sample - the cluster with the nearest center to the sample
2. Chose the $k$ nearest neighbours from the cluster
3. Classify according to the weighted majority of the neighbours
Solution - Schema

**Training**
- **Landmarks**
  - size reduction before creating clusters
  - uses k-means to find landmarks

- **Spectral clustering**
  - dimensionality reduction
  - uses k-means to create clusters in the spectral space

**Testing**
- **Finding the nearest cluster**
  - size reduction before kNN

- **classic kNN applied on the selected cluster**
Results

- Here for 10 clusters created using 1 eigenvector
- 7-9 times faster than kNN
- 1-2.6% less accuracy than kNN
Remarks

- We can use different optimization for spectral clustering (instead of landmarks):
  - KASP - k-means-based approximate spectral clustering [Yan, Huang, and Jordan 2009]
  - CSC - Committees-based Spectral Clustering [Shinnou and Sasaki 2008]
  - Nyström [Chen et al. 2010]

- We can use different algorithm than k-means for creating the landmarks, for example *random sampling*
Thank you!
References

References