Classification of points in a euclidean space

Luc Blassel, Romain Gautron

8 mars 2018

Our problem

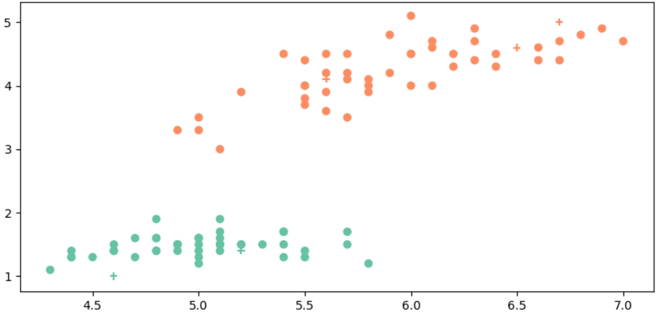


Figure – Classification problem in euclidean space

Our problem

Let S be a set of l points of $\mathbf{R^d}$.

Each element of S has a known label y_i in $\{-1, 1\}$.

Let U be a set of $\mathbf{R^d}$ points for which we don't know the labels.

We suppose that S and U come from the same ensemble.

We want to assign labels from $\{-1, 1\}$ to the points of U.

Our problem

A lot of ways to solve it :

- generative approach →learning the distribution of individual classes
  - Naives Bayes
  - Logistic regression
  - ...
- discriminative approach →learning the boundaries between classes
  - KNN
  - Random Forest
  - LDA
  - NNets
  - ...

Our framework

Discriminative approach : KNN

- simple algorithm
- well performing
- non linear

$\rightarrow$we will deal with the multiclass case

The KNN algorithm

KNN principle (1)

In order to assign a label to an unknown point :

1. find the k-nearest neighbours according to a distance
2. take the majority of the corresponding labels ; if equality pick one randomly

KNN principle (2)

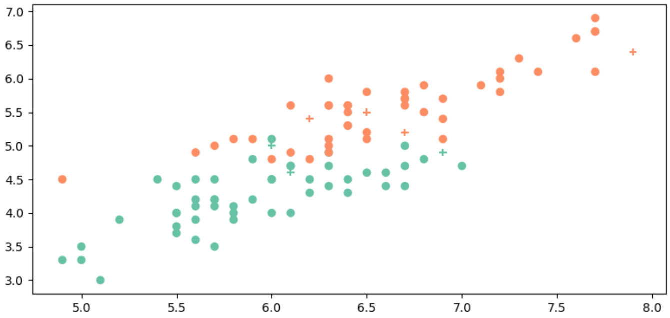


Figure – KNN example

Naive KNN approach

Given an unknown point X :

1. compute all the distances :
   for $s \in S, i \in I : \|s_i - X\|$
2. sort the distances and select the k-least
3. take the majority of the corresponding labels ; if equality pick one randomly

Given a space dimension d, research costs for one unknown point (assuming $k \ll n$) :

- $O(d)$ per known point
- $O(nd)$ for the whole data set

Space complexity is $O(dn)$

How to improve naive knn ?

- approximate knn
- exact knn

→based on pre-partionning the space

LSH approach

Random hyperplan →hash function

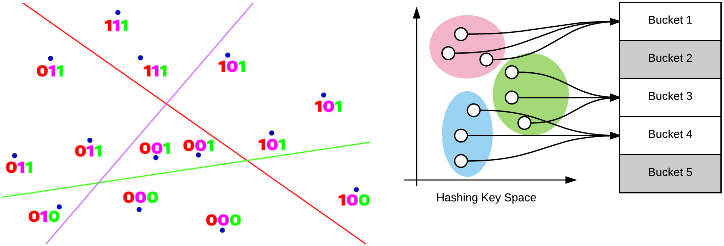


Figure – LSH

The region where falls the new point gives the candidates
Repeat it - gather candidates - compute distances
$O(kd + \frac{nd}{2^k}) \approx O(dlog(n))$

KD trees V1

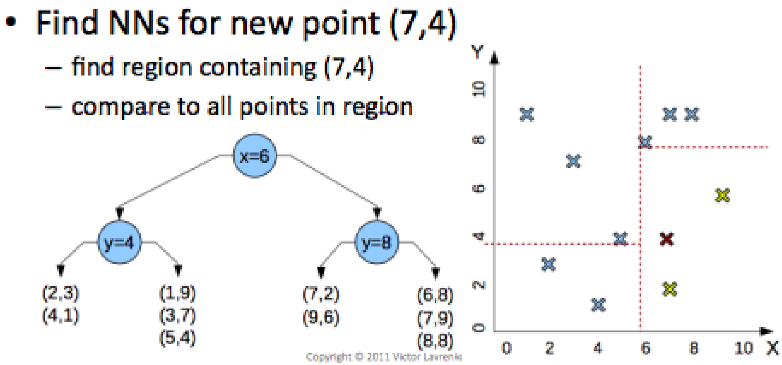


Figure – KD trees V1 (copyright V. Lavrenko)

$O(log(n) + kd)$

KD trees V2

→KDtrees with exact NNs : our work !

A simple example

We have the following dataset in a $2d$ space :

$$X = \{(1,3),(1,8),(2,2),(2,10),(3,6),(4,1),(5,4),(6,8),$$
$$(7,4),(7,7),(8,2),(8,5),(9,9)\}$$

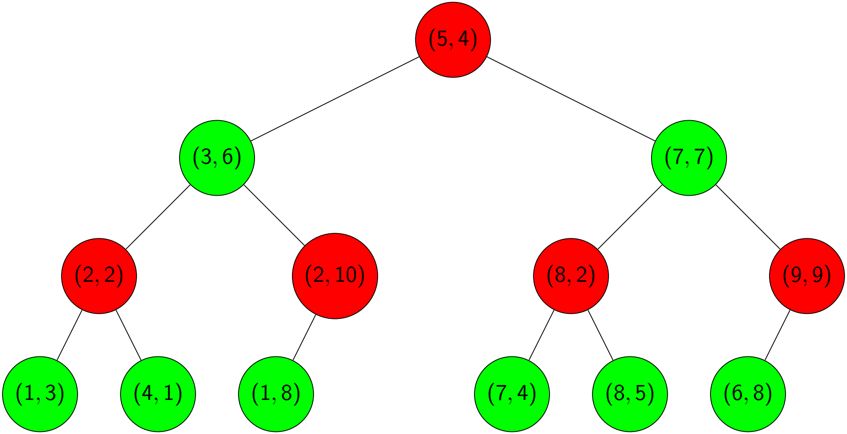
$$Y = \{Blue,\ Blue,\ Blue,\ Blue,\ Blue,\ Blue,\ Red,\ Red,\ Red,\ Red,$$
$$Red,\ Red,\ Red\}$$

We want to assign a color to the following point : $(4,8)$

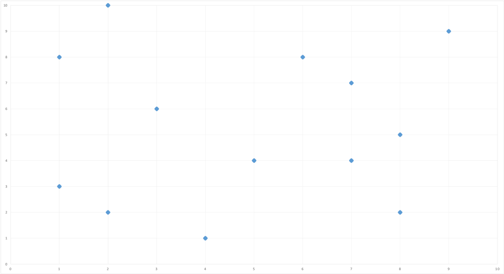
How to opitimze k-nn search

We use a data structure known as a k-d tree.

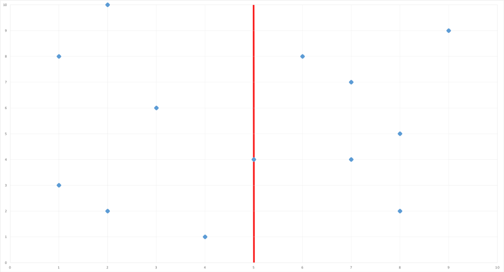
building the k-d tree



How do we partition the space ?

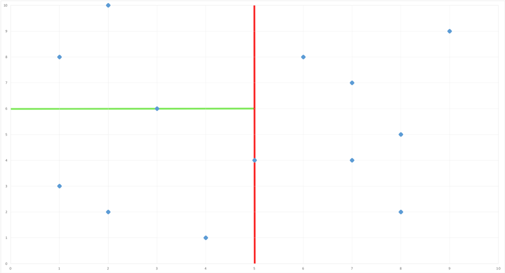


How do we partition the space ?



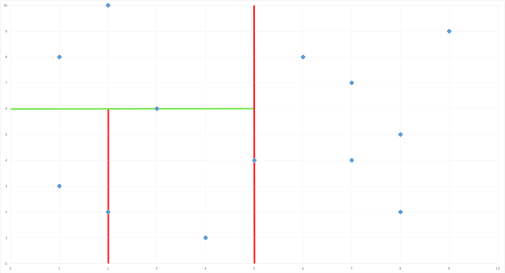
4

How do we partition the space ?



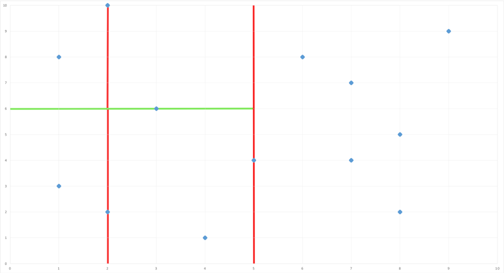
4

How do we partition the space ?



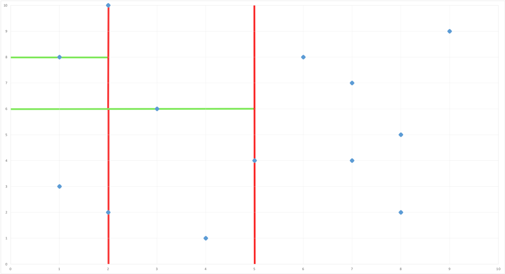
4

How do we partition the space ?



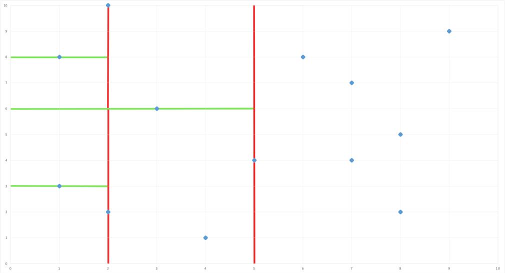
4

How do we partition the space ?



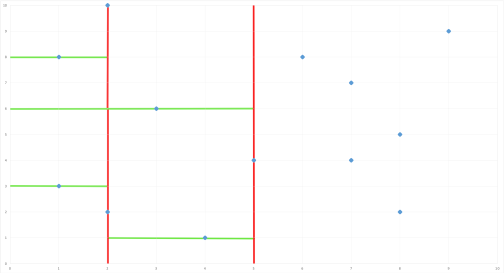
4

How do we partition the space ?



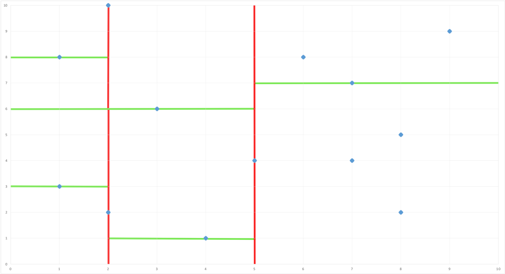
4

How do we partition the space ?



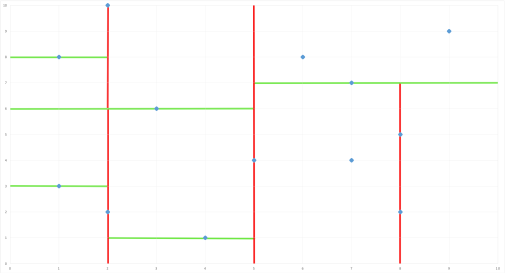
4

How do we partition the space ?



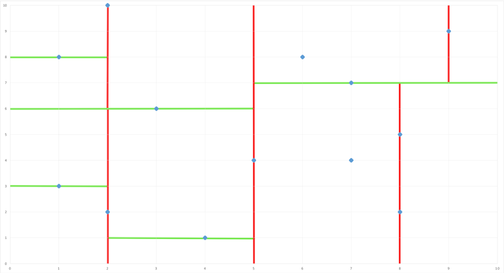
4

How do we partition the space ?



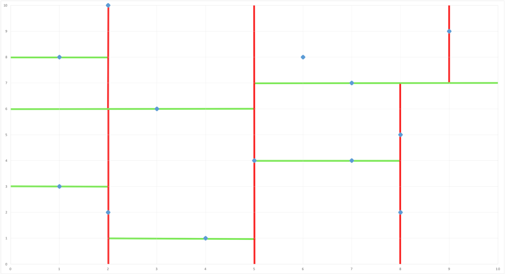
4

How do we partition the space ?



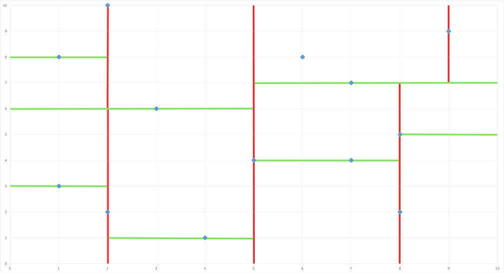
4

How do we partition the space ?



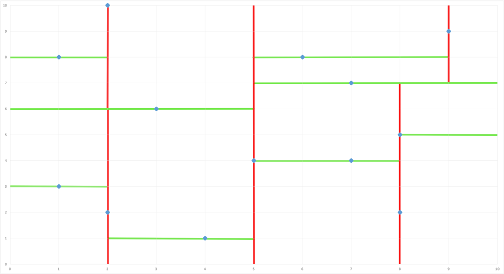
4

How do we partition the space ?



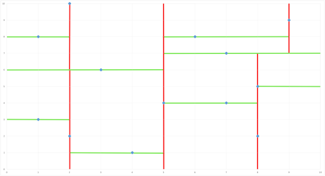
4

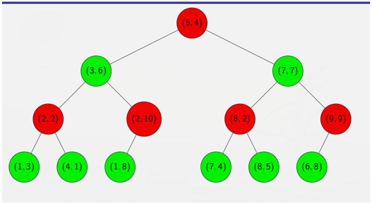
How do we partition the space ?



4

how do we find the k nearest neighbours ?





What is the time complexity ?

For the tree creation (best case) :

$$
\begin{aligned}
T(n) &= nlog(n) + 2T\left(\frac{n}{2}\right) \\
&= nlog(n) + 2\left(\frac{n}{2}log\left(\frac{n}{2}\right) + 2T\left(\frac{n}{4}\right)\right) \\
&= nlog(n) + nlog\left(\frac{n}{2}\right) + 4T\left(\frac{n}{4}\right) \\
&= nlog(n) + nlog\left(\frac{n}{2}\right) + nlog\left(\frac{n}{4}\right) + \cdots + nlog\left(\frac{n}{2^{log(n)}}\right) \\
&= n\sum_{i=0}^{log(n)} log\left(\frac{n}{2^i}\right) \\
&= O\left(n\sum_{i=0}^{log(n)} log(n)\right) \\
&= O(nlog^2(n))
\end{aligned}
$$

What is the time complexity ?

For the tree creation (worst case) :

$$\begin{aligned}
T(n) &= n^2 + 2T\left(\frac{n}{2}\right) \\
&= n^2 + 2\left(\left(\frac{n}{2}\right)^2 + 2T\left(\frac{n}{4}\right)\right) \\
&= \sum_{i=0}^{log(n)} \frac{n^2}{2^i} \\
&= n^3(2 - 2^{-log(n)}) \\
&= O(n^3)
\end{aligned}$$

what is the time complexity

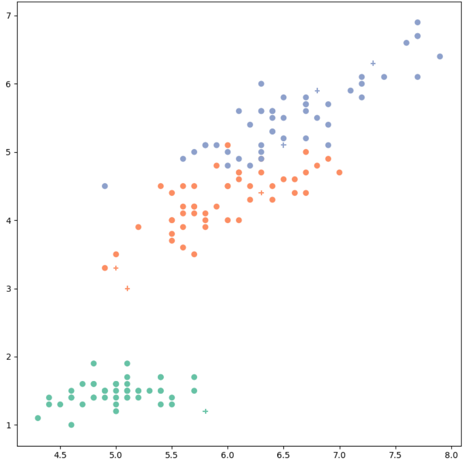
time complexity of our full program :

- For the nearest neighbour search :
  - best : $T(n) = O(log(n)d)$
  - worst : $T(n) = O(nd) \equiv$ *depth-first traversal*
- For k-nearest neighbours of p points :
  - best :
    - if $pd > nlog(n) \Rightarrow T(n) = O(pdlog(n))$
    - otherwise $T(n) = O(nlog^2(n))$
  - worst :
    - if $pd > n^2 \Rightarrow T(n) = O(pdn)$ (very unlikely)
    - otherwise $T(n) = O(n^3)$

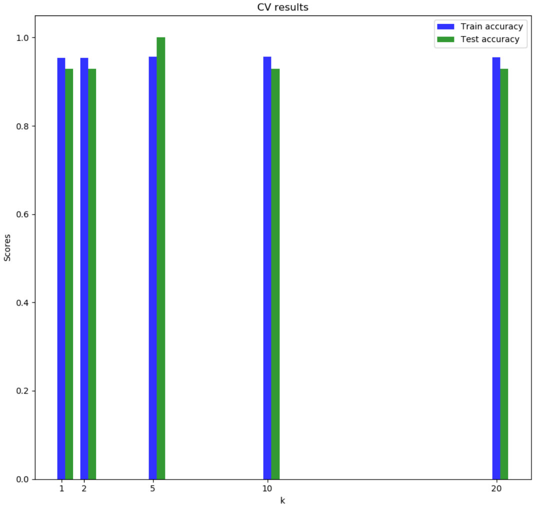
Most of the complexity is due to tree creation.

k-fold cross validation does not increase complexity unless the number of folds is very high.

How does our program perform ?



How does our program perform ?



does it work for bigger spaces ?

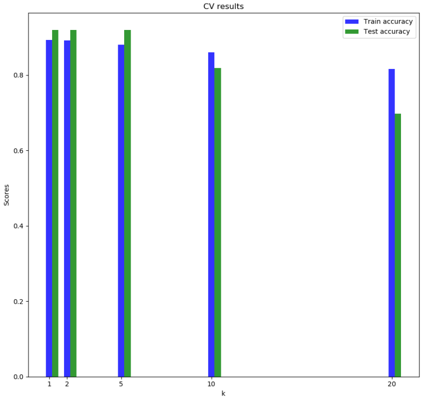


Figure – CV results on leaf dataset *(d=192, 99 classes, n=990)*

- Implement approximative in CV then apply exact Knn with optimal k
- be able to handle categorical data
- implement faster median selection algorithm (for faster tree creation)

sources I

[1] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1) :21–27, 1967.

[2] Gopal Das. k-d tree and nearest neighbor search. https://gopalcdas.com/2017/05/24/construction-of-k-d-tree-and-using-it-for-nearest-neighbour-search/, May 24, 2017.

[3] Sandipan Dey. Implementing kd-tree for fast range-search, nearest-neighbor search and k-nearest-neighbor search algorithms in 2d in java and python. https://www.datasciencecentral.com/profiles/blogs/implementing-kd-tree-for-fast-range-search-nearest-neighbor, May 24, 2017.

[4] Victor Lavrenko. Iaml : Nearest neighbours. http://www.inf.ed.ac.uk/teaching/courses/iaml/2011/slides/knn.pdf, 2011.

[5] PyChen. How to calculate the average time complexity of the nearest neighbor search using kd-tree ? https://stackoverflow.com/questions/22577032/how-to-calculate-the-average-time-complexity-of-the-nearest-neighbor-search-usin, March 22, 2014.

[6] Ronald L Rivest and Charles E Leiserson. *Introduction to algorithms*. McGraw-Hill, Inc., 1990.

[7] Gilbert Saporta. *Probabilités, analyse des données et statistique*. Editions Technip, 2006.

[8] Michael Sharma, Aneesh & Wand. Approximate nearest neighbors search in high dimensions and locality-sensitive hashing. https://graphics.stanford.edu/courses/cs468-06-fall/Slides/aneesh-michael.pdf, March 22, 2014.

[9] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal processing magazine*, 25(2) :128–131, 2008.

[10] Unknown. Mit 6.854 spring 2016 lecture 6 : Nearest neighbor search and lsh. https://www.youtube.com/watch?v=vAboxtLEeH0, 2017.