# Classification of points in a euclidean space

Luc Blassel, Romain Gautron

8 mars 2018

# Our problem

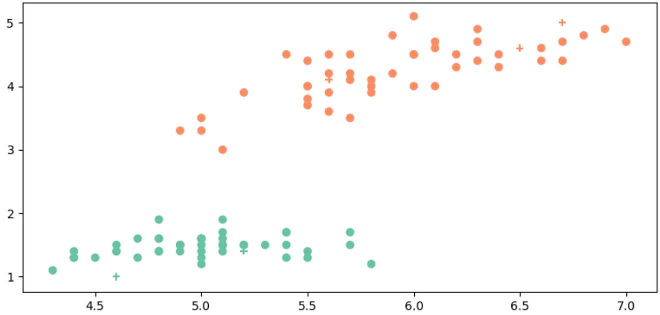



Figure – Classification problem in euclidean space

# 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)

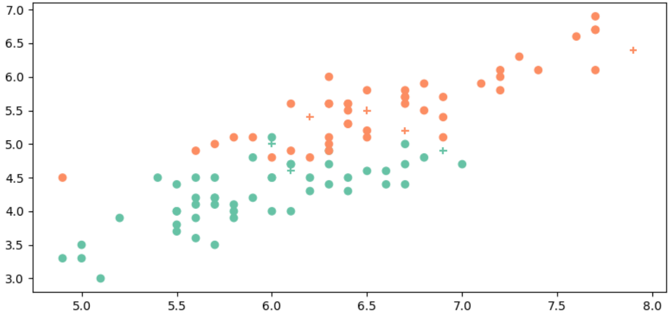



Figure – KNN example

# 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

- if $d \ll n$ search time complexity is $o(n)$
- else $\Omega(n)$ and $o(d^2)$

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

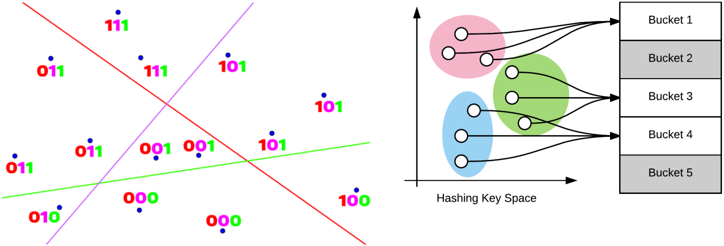



Figure – LSH

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

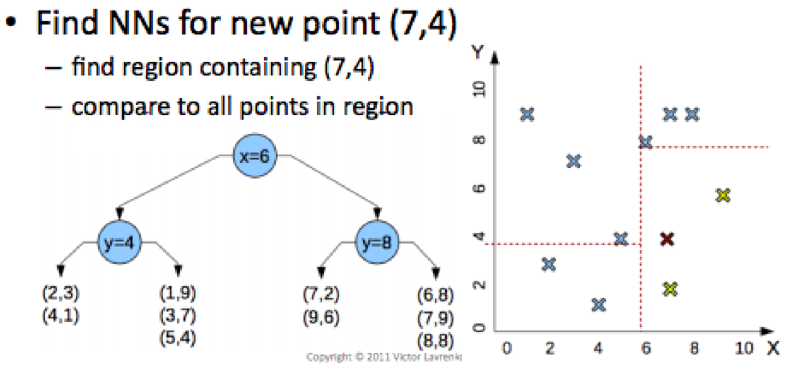



Figure – KD trees V1 (copyright V. Lavrenko)

$o(klog(n))$

# KD trees V2

$\rightarrow$KDtrees with exacts NNs : our work !

# A simple example

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

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

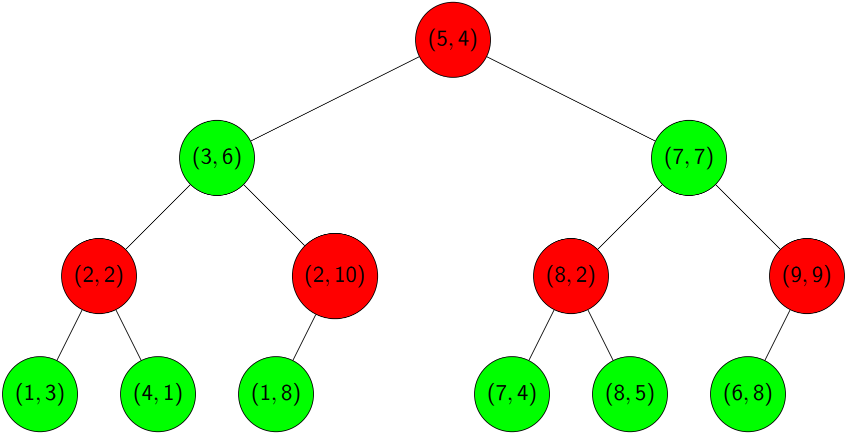
$$Y = \{Blue,\ Blue,\ Blue,\ Blue,\ Blue,\ Blue,\ Red,\ Red,\ Red,\ Red,\ Red,\ Red,\ Red\}$$

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

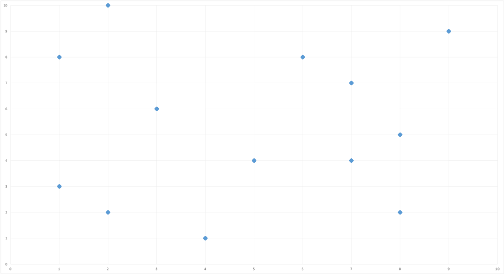
# How to opitimze k-nn search

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

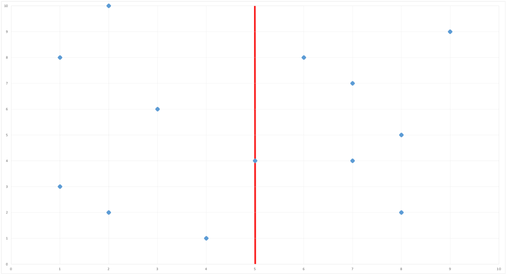
# building the k-d tree



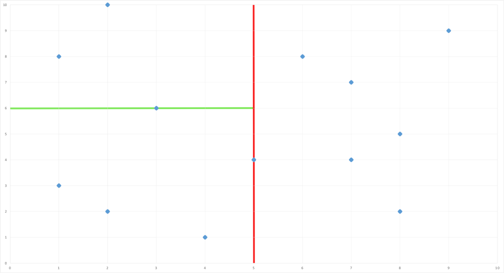
# How do we partition the space ?



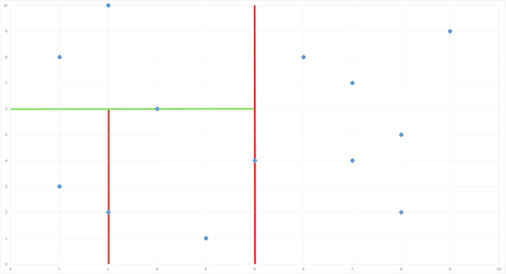
# How do we partition the space ?



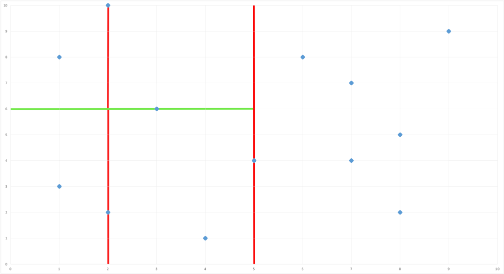
# How do we partition the space ?



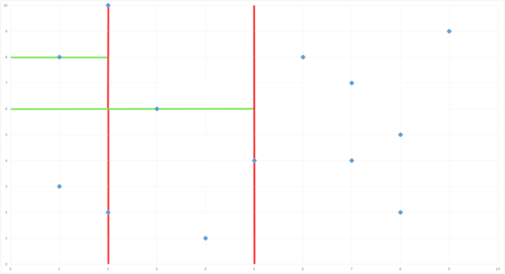
# How do we partition the space ?



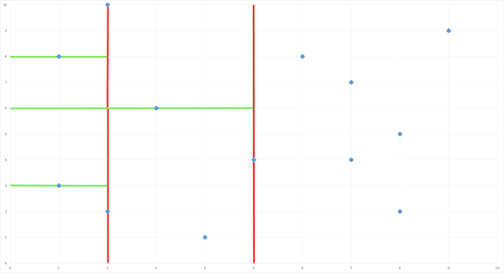
# How do we partition the space ?



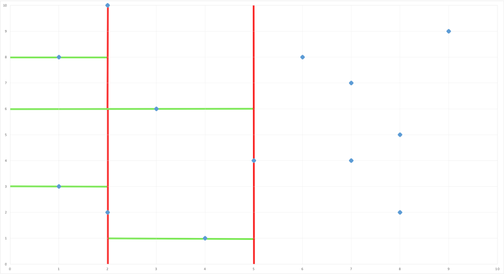
# How do we partition the space ?



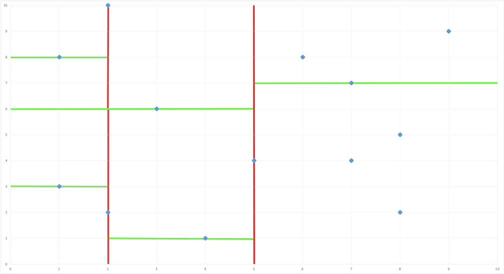
# How do we partition the space ?



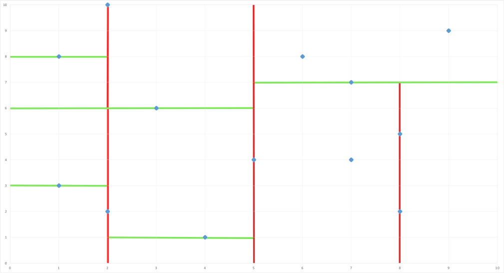
# How do we partition the space ?



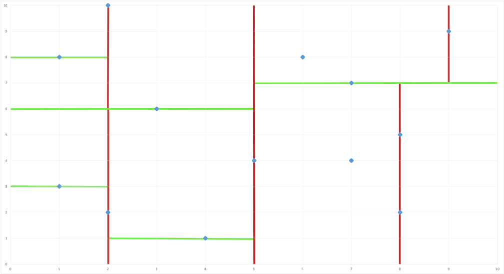
# How do we partition the space ?



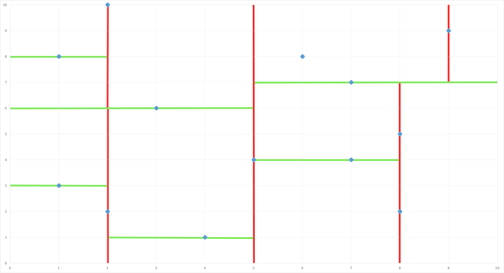
# How do we partition the space ?



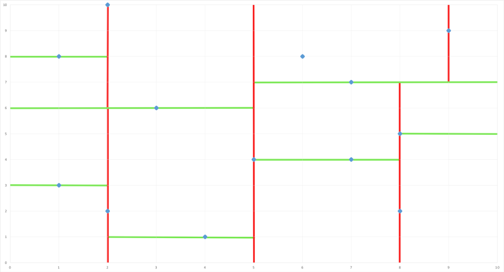
# How do we partition the space ?



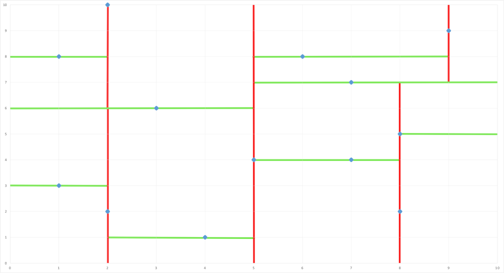
# How do we partition the space ?



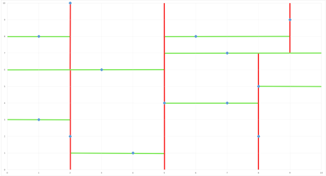
# How do we partition the space ?

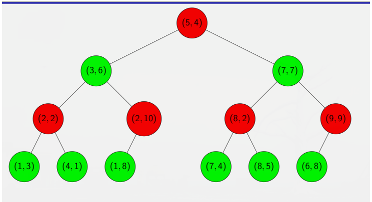


# How do we partition the space ?



# 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

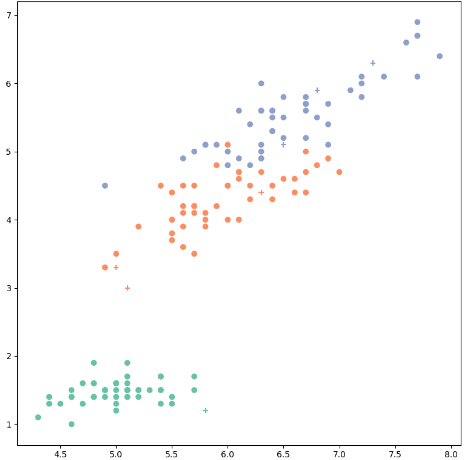
time complexity of our full program :

- 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 $T(n) = o(n^3)$

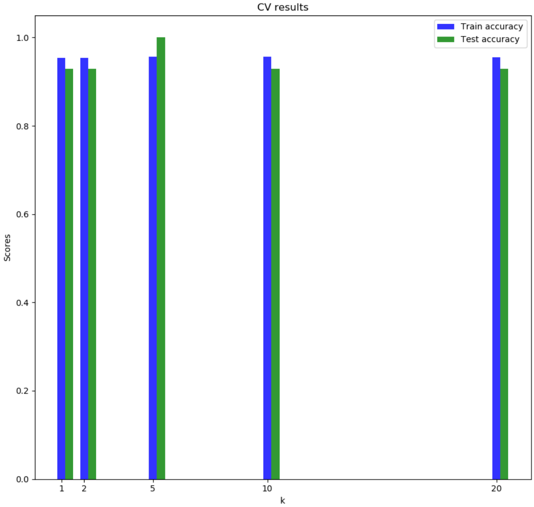
Most of the complexity is due to tree creation.

$k$-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 ?

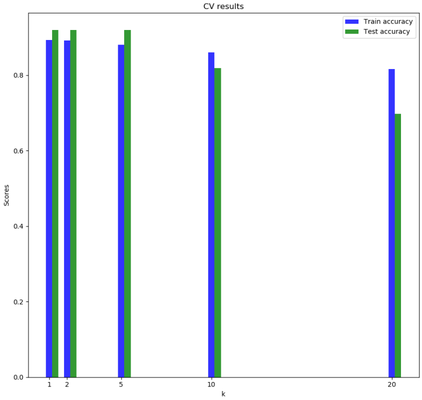



Figure – CV results on leaf dataset *(d=192, 99 classes, n=990)*

- 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)