Classification methods

INTRODUCTION

- Aim: Label subjects defined by features.
- Supervised / Unsupervised / Semi supervised methods

→ Here: supervised algorithms

INTRODUCTION

- Training Set $Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in Y\}$.
- Objective: for $x \in X$, find its label based on an algorithm built on Z.
- Central problem in this course (SVM, NN, ...)
- Here: description of four simple algorithms
  - Naïve Bayes classifier
  - K Nearest Neighbors
  - Linear/Quadratic discriminant analysis
  - Decision trees
- Some others will be described later in the course

NAÏVE BAYES RULE

Simple decision rule

$$(x \textbf{ is in class } k \in Y) \Leftrightarrow (k = arg\max_l P(y = l|x))$$

using Bayes' rule $(\forall k \in Y) \quad P(y = l|x) = \frac{P(x|y=l)P(y=l)}{P(x)}$
the rule becomes

$$(x \textbf{ is in class } k \in Y) \Leftrightarrow (k = arg\max_l P(x|y = l)P(y = l))$$

with

- $P(y = l) = p_l = \frac{n_l+m}{n+m|Y|}$
- $P(x|y = l)$ needs assumptions to be estimated

$\Rightarrow$ Maximum A Posteriori rule

ESTIMATION OF $P(x|y=l)$

$P(x|y=l)$: class conditional probability.
Assumption: Features are independent, conditionaly to Y.

$$P(x|y=l) = \prod_{j=1}^{d} P\left(f_j = x_j | y = l\right)$$

where $d = |X|$, f_j is the j^{th} feature. If f_j takes Q possible discrete values then

$$(\forall q) \quad P\left(f_j = q | y = l\right) = \frac{n_{lq} + m}{n_l + mQ}$$

Using log values, the final Naïve Bayes rule is

$$(x \text{ is in class } k \in Y) \Leftrightarrow \left(k = arg\max_l \left[log(P(y=l) + \sum_{j=1}^{d} logP\left(f_j = x_j | y = l\right) \right] \right)$$

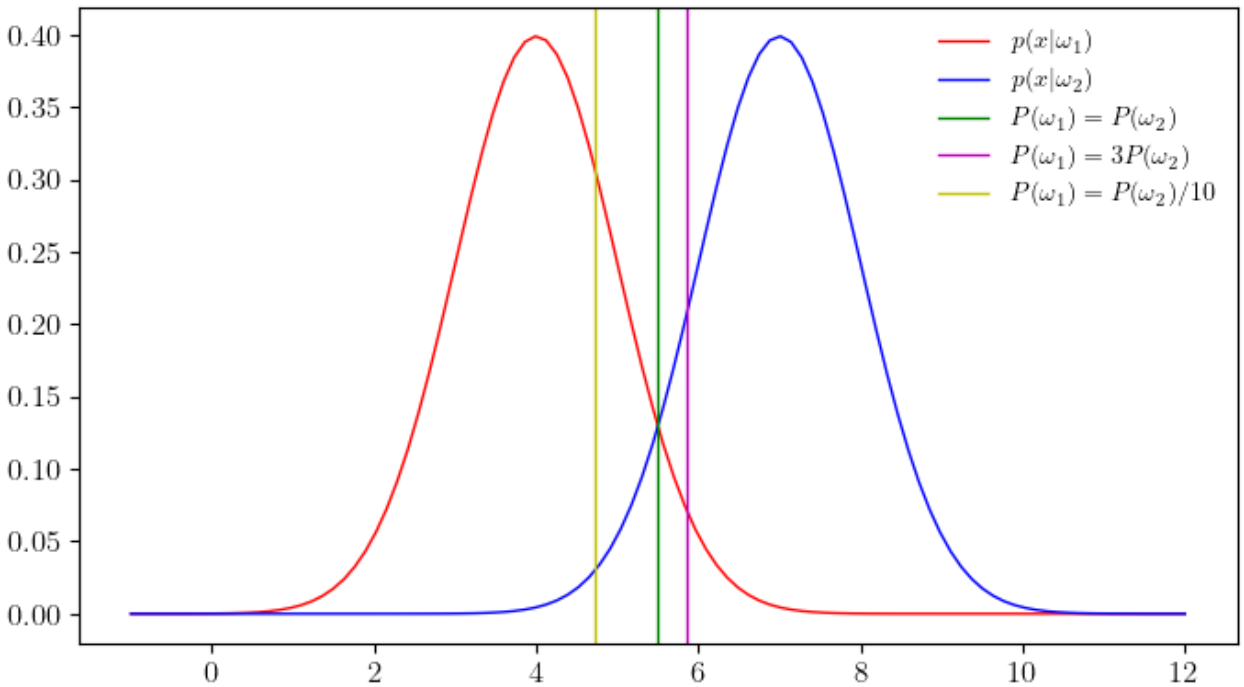
```python
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
X, y = ...
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
naive = GaussianNB()
y_pred = naive.fit(X_train, y_train).predict(X_test)
```

Example

$$p(x|\omega_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right), i \in \{1,2\}$$

$$(\mu_1, \sigma_1) = (4,1),\ (\mu_2, \sigma_2) = (7,1)$$

- $P(\omega_1) = P(\omega_2) = \frac{1}{2} \Rightarrow x = 5.5$
- $P(\omega_1) = 3P(\omega_2) \Rightarrow x = 5.86$
- $P(\omega_1) = P(\omega_2)/10 \Rightarrow x = 4.74$



ERROR

$$P(error|x) = \begin{cases} P(\omega_1|x) & \text{if } \omega_2 \text{ is chosen} \\ P(\omega_2|x) & \text{if } \omega_1 \text{ is chosen} \end{cases}$$

$$\Rightarrow P(error) = \int P(error|x)p(x)dx$$

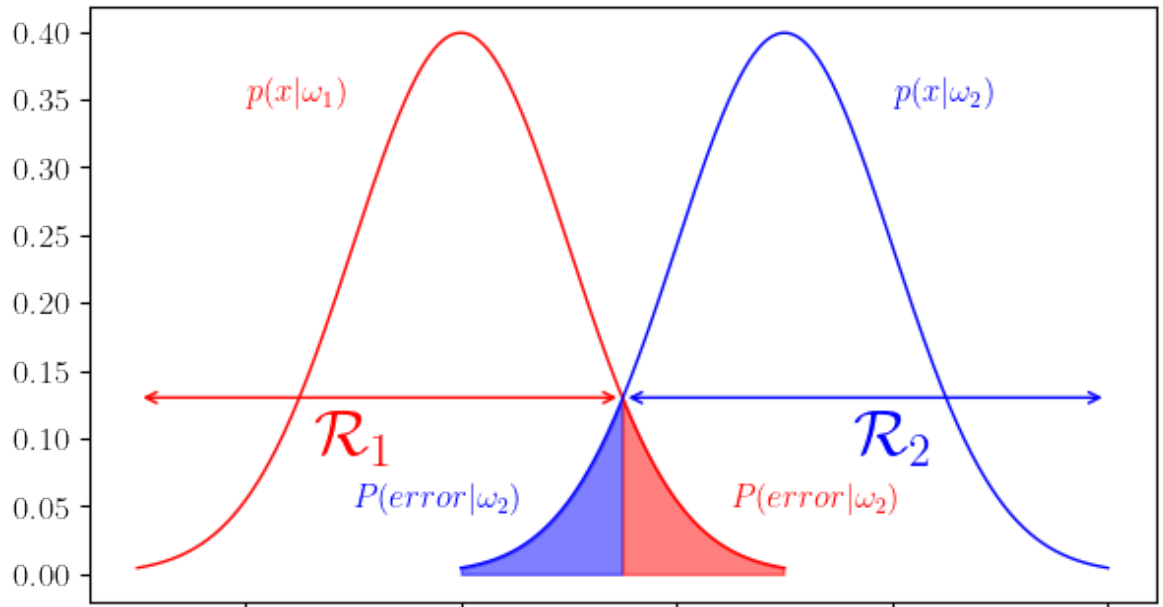
Minimizing $P(error)$ $\Leftrightarrow$ MAP rule.

$$P(error) = P(error|\omega_1)P(\omega_1) + P(error|\omega_2)P(\omega_2)$$

with

$$P(error|\omega_1)P(\omega_i) = P(\text{choose } \omega_j|\omega_i) = \int_{x \in \mathcal{R}_j} p(x|\omega_i)dx$$

where $\mathcal{R}_j$ is the decision region for x to belong to ω_j



Definition

Parametric methods considering the log ratio

$$\log\left(\frac{P(y=k|X=x)}{P(y=l|X=x)}\right)$$

for all class pairs (k,l), and returning the class for which these log ratio are always positive.

Notation: for $i \in [\![1, C]\!]$:

$$g_i(x_0) = P(y=i|x=x_0) = \frac{P(x=x_0|y=i)\,P(y=i)}{\sum_{j=1}^{C} P(x=x_0|y=j)\,P(y=j)} = \frac{f_i(x_0)\pi_i}{\sum_{j=1}^{C} f_j(x_0)\pi_j}$$

with $f_i(x_0) = P(x=x_0|y=i)$ and $\pi_i = P(y=i)$

Example

$C = 2$,

$$f_i(x) = \frac{1}{(2\pi)^{d/2}\,|\Sigma_i|^{1/2}}\, e^{-\frac{1}{2}(x-\mu_i)^\top \Sigma_i^{-1}(x-\mu_i)}$$

Find x such that $g_0(x) = g_1(x)$ (decision boundary)

$$g_0(x) = g_1(x) \Leftrightarrow \frac{f_0(x)\pi_0}{\sum_{j=1}^{C} f_j(x)\pi_j} = \frac{f_1(x)\pi_1}{\sum_{j=1}^{C} f_j(x)\pi_j}$$

then $f_0(x)\pi_0 = f_1(x)\pi_1$
or

$$\frac{1}{(2\pi)^{d/2}\,|\Sigma_0|^{1/2}}\, e^{-\frac{1}{2}(x-\mu_0)^\top \Sigma_0^{-1}(x-\mu_0)} = \frac{1}{(2\pi)^{d/2}\,|\Sigma_1|^{1/2}}\, e^{-\frac{1}{2}(x-\mu_1)^\top \Sigma_1^{-1}(x-\mu_1)}$$

If $\forall i\, \Sigma_i = \Sigma$ then

$$\pi_1 e^{-\frac{1}{2}(x-\mu_1)^\top \Sigma^{-1}(x-\mu_1)} = \pi_0 e^{-\frac{1}{2}(x-\mu_0)^\top \Sigma^{-1}(x-\mu_0)}$$

taking the log:

$$log\pi_1 - \frac{1}{2}(x-\mu_1)^\top \Sigma^{-1}(x-\mu_1) = log\pi_0 - \frac{1}{2}(x-\mu_0)^\top \Sigma^{-1}(x-\mu_0)$$

thus

$$log\left(\frac{\pi_1}{\pi_0}\right) + \frac{1}{2}\left[(\mu_0^T\Sigma^{-1}\mu_0)^T - (\mu_1^T\Sigma^{-1}\mu_1)^T\right] + (\mu_1-\mu_0)^T\Sigma^{-1}x = 0$$

If $a^T = (\mu_1-\mu_0)^T\Sigma^{-1}$ and $b = log\left(\frac{\pi_1}{\pi_0}\right) + \frac{1}{2}\left[(\mu_0^T\Sigma^{-1}\mu_0)^T - (\mu_1^T\Sigma^{-1}\mu_1)^T\right]$ then this is a linear decision boundary

$$a^T x + b = 0$$

Gaussian distributions are estimated using Z:

- $\hat{\pi}_i = \frac{n_i}{n}$, where $n_i = \#\{x \in Z, y = i\}$
- $\hat{\mu}_i = \frac{1}{n_i} \sum_{j, y_j = i} x_j$
- $\Sigma_i = \frac{1}{n_i - n} \sum_{j, y_j = i} (x_j - \hat{\mu}_i)(x_j - \hat{\mu}_i)^T$ and $\Sigma = \frac{1}{n} \sum_{j=1}^{C} \Sigma_j$

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
X,y = ...
lda = LinearDiscriminantAnalysis()
lda.fit(X, y)
```

QDA

If $\forall i\, \Sigma_i = \Sigma$ doesn't hold anymore, the decision boundary is quadratic.

$$x \in \ class\ k \Leftrightarrow k = arg\max_i -\frac{1}{2} log|\Sigma_i| - \frac{1}{2}(x-\mu_i)^\top \Sigma_i^{-1}(x-\mu_i) + log\pi_i$$

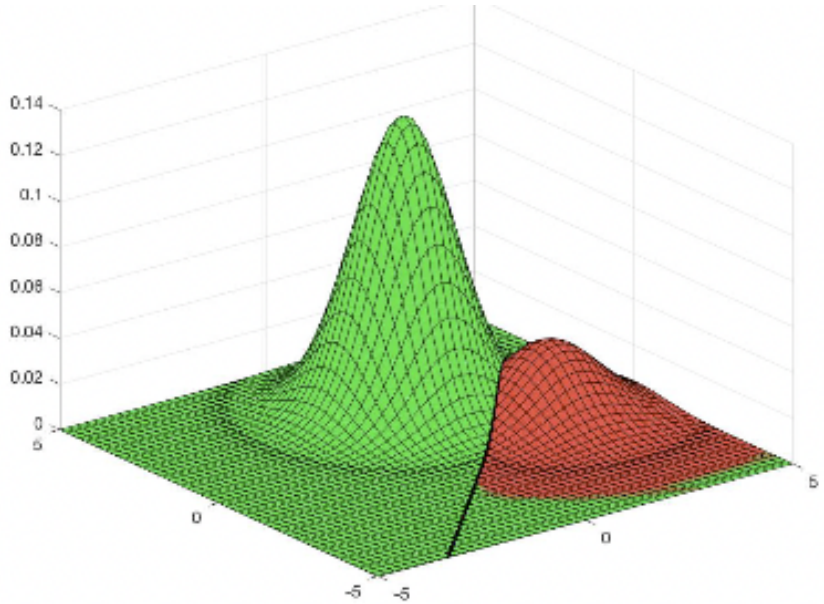
- Spherical classes ($\Sigma_i = I, \forall i$) : $-\frac{1}{2}\|x-\mu_i\|^2 + log\pi_i$.
  - if all π_i are equal, the decision rule minimizes $\|x-\mu_i\|^2$: 1-NN
  - otherwise, $\|x-\mu_i\|^2$ is adjusted w.r.t the class sizes
- otherwise $\Sigma_i = USV^T$, with $U = V$ (Σ_i symmetric), U eigenmatrix of $\Sigma_i\Sigma_i^T$ (SVD) and we can prove that if $A^T = S^{-1/2}U^T$ then

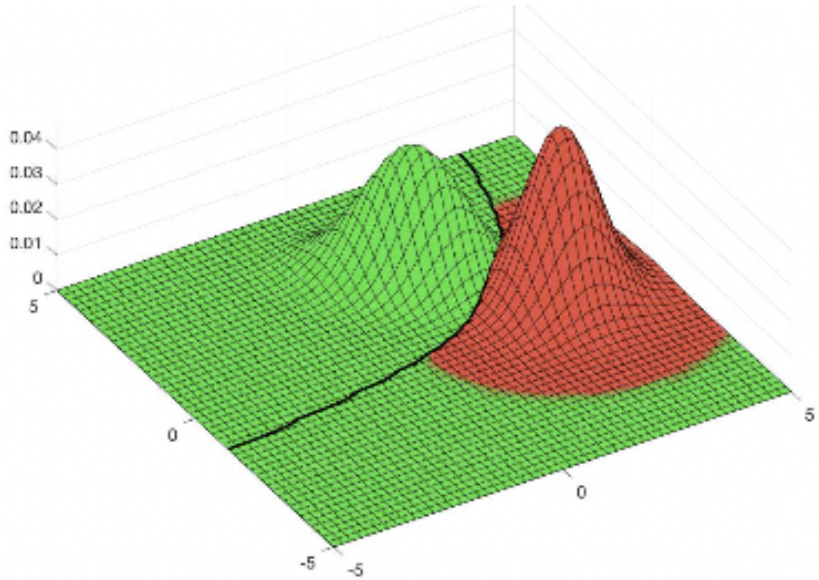
$$(x-\mu_i)^\top \Sigma_i^{-1}(x-\mu_i) = \|A^T x - A^T \mu_i\|^2$$

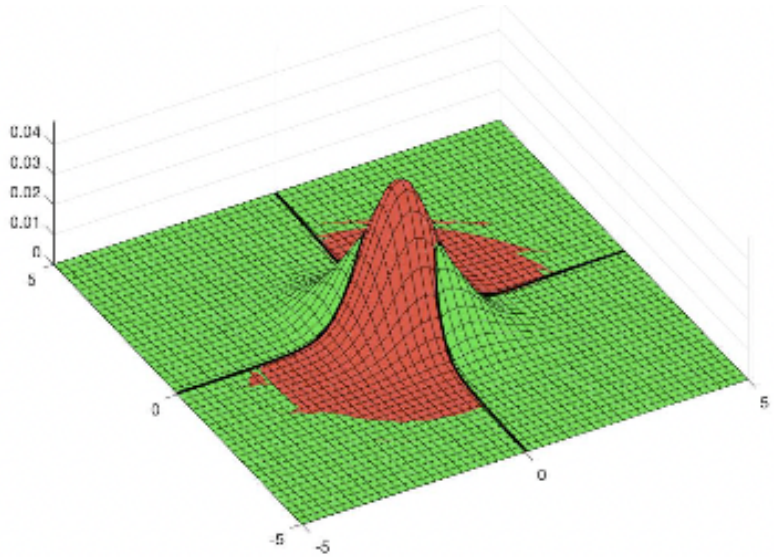
and this is the spherical case with the linear transform A^T.

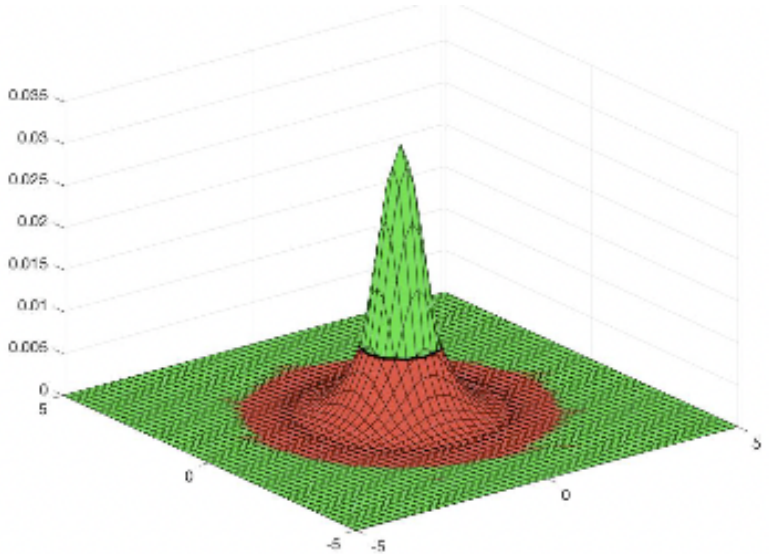
```
from sklearn.qda import qda
X,y = ...
qda = qda()
qda.fit(X, y)
```

Example


0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
5
0
-5
-5
0
5


0.04
0.03
0.02
0.01
0
5
0
-5
-5
0
5


0.04
0.03
0.02
0.01
0
5
0
-5
-5
0
5


0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
5
0
-5
-5
0
5

K Nearest Neighbors

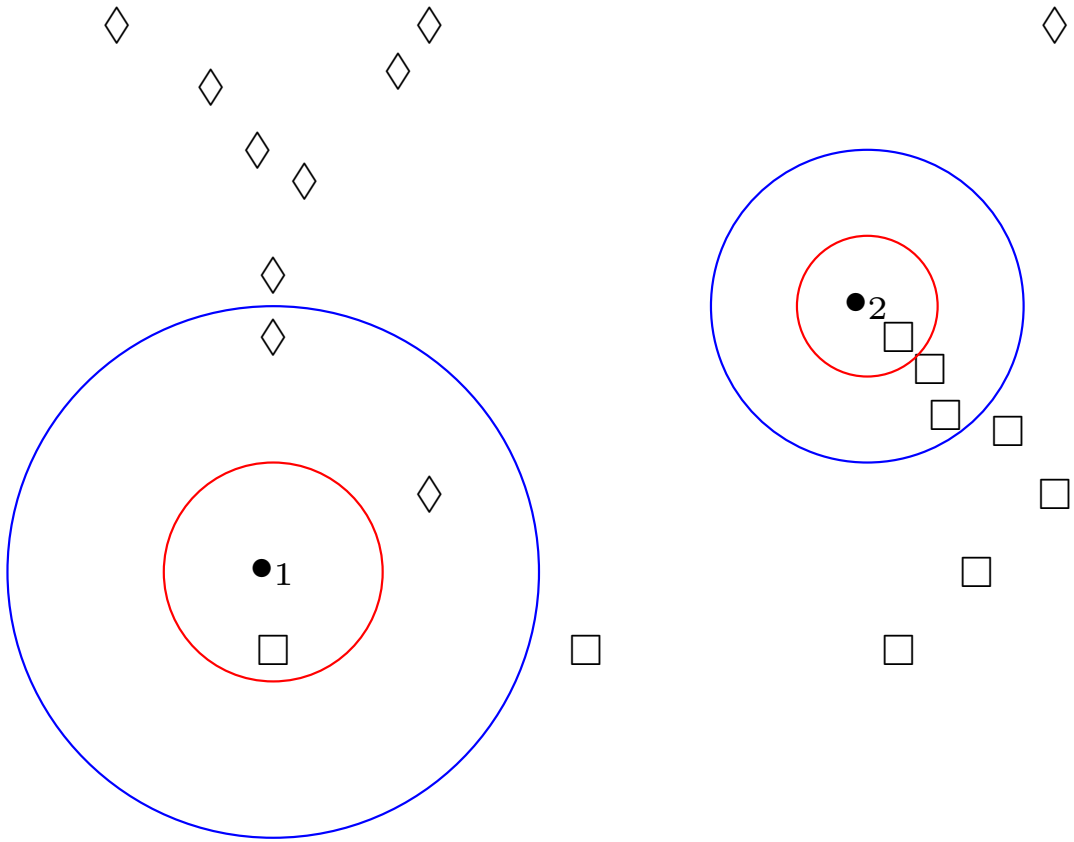
Given:

1. δ : metric on X
2. $k \in \mathbb{N}$
3. $x \in X$

the kNN rule assigns x to the most class represented in the k neighbors $(x_i, y_i) \in Z$ of x, in the sense of δ.

Can also be used in regression: $y(x) = \frac{1}{k} \sum_{i=1}^{k} y_i$

1 and 3-NN



Parameters

Parameters

- $K \approx \sqrt{n/C}$ where n/C if the average number of training points per class.
- Training points can be weighted by their distance to x

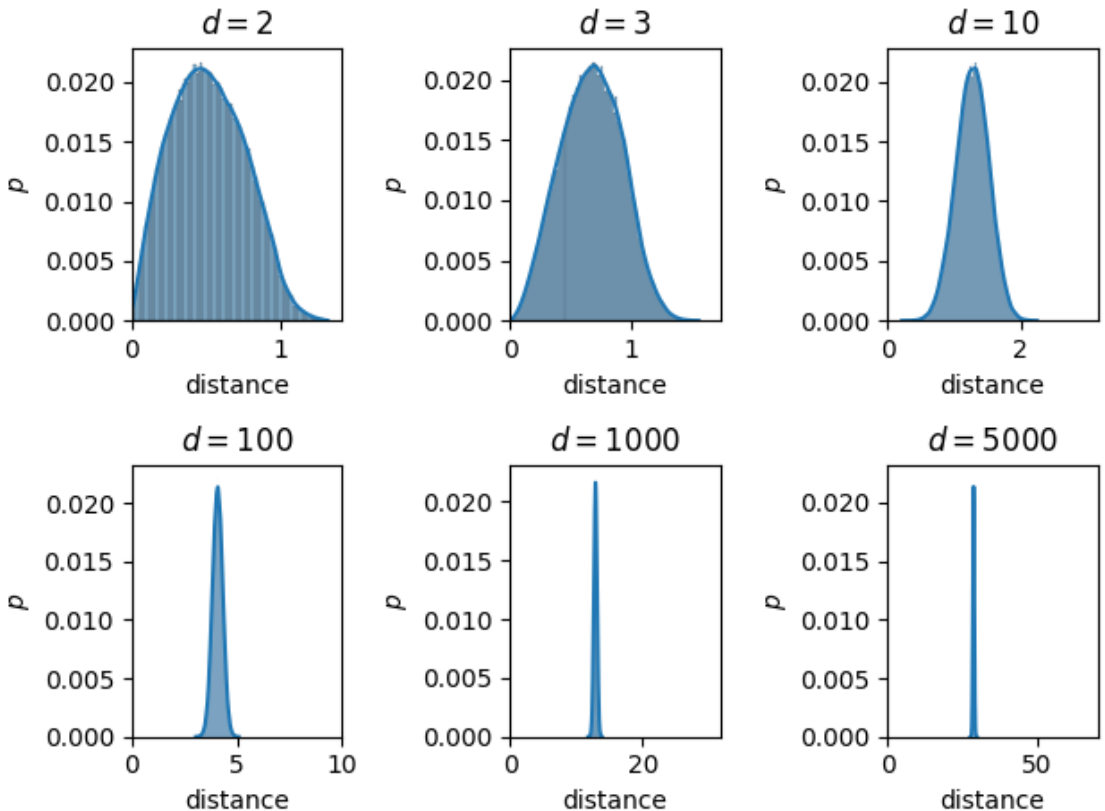
```
from sklearn.neighbors import KNeighborsClassifier
X,y=...
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, y)
```

Curse of dimensionality

KNN hypothesis

Close points belong to the same class. In high dimension spaces, data points sampled from a random probability distribution, are far from each other with (almost) the same distance value.

Sample points uniformly at random within the unit cube and compute the distance between all pair of points when the dimensionality increases.

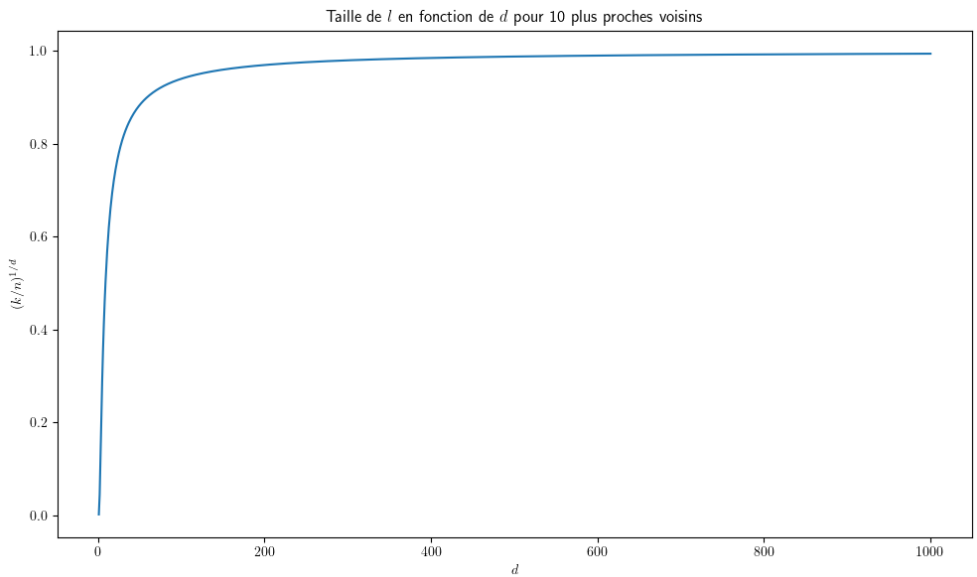


CURSE OF DIMENSIONALITY

Example

- Uniform sample in the unit hypercube in $\mathbb{R}^d$.
- Compute the volume occupied by the K-NN of a point.
- $K = 10$, l^3: minimal volume of the hypercube containing k NN

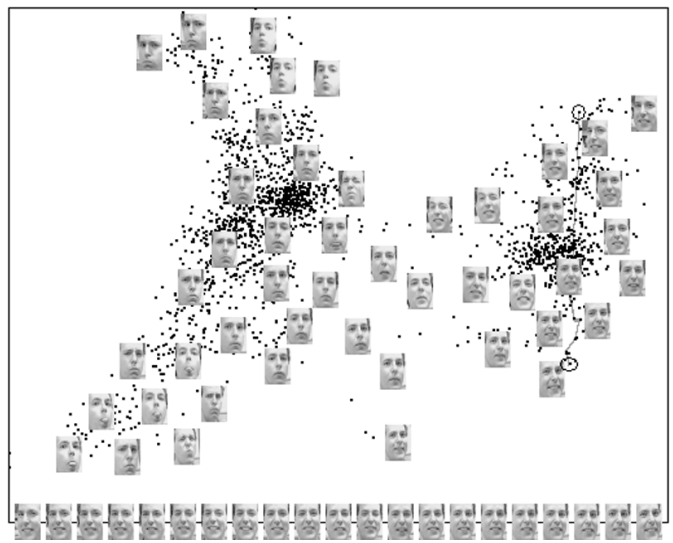
$\Rightarrow l^d \approx \frac{k}{n}$



Curse of dimensionality

- Real-world data does not follow a random probability distribution
- Data has structure (edges, textures)

$\Rightarrow$ Lie in a much lower dimensional sub-space (not necessarily linear) than $\mathbb{R}^d$.



See...

Manifold learning lecture

KNN complexity

Complexity: $O(n.d.k)$ where n is the number of training samples, d the dimension of the feature space.

$\Rightarrow$ KNN becomes very slow and memory consuming when n is large

- BUT we want to have n as large as possible to get the best possible accuracy.

$k-d$ tree

Solution to speed up the process

- Leveraging data structure
- When we search for the closest point(s), most data points are actually far away

$\Rightarrow$ There is no need to compute the distances for these far away points.

$\Rightarrow$ Partition the feature space with a binary tree structure.

KNN COMPLEXITY

Example: let Z be the full dataset

1. cut along one feature dimension (hyperplane H) that divides the data into two sets Z_1 and Z_2, with approximatively $|Z_1| \approx |Z_2| \approx |Z|/2$
2. let x be a new data point x: we want to find the closest neighbor.
3. identify in which set the data x lies, e.g Z_1.
4. find the nearest neighbor $y \in Z_1$ in $O(n/2)$.
5. compute $d(x, H)$ between x and H.
6. if $d(x, H) > d(x, y)$ then all $\in Z_2$ can be discarded (by triangular inequality)

$\Rightarrow$ $2\times$ speed-up!

7. if $d(x, H) < d(x, y)$: it is possible that the nearest neighbor lies in Z_2

$\Rightarrow$ worst case complexity = KNN complexity, but average complexity is better.

K-D TREE

Tree construction

- Split recursively in half along each feature dimension.
- Iterate over all feature dimensions.

Tree depth is quite small : $O(log_2(n))$

Heuristic to select which next feature dimension: select the one that captures the largest variation of data ($\approx$ PCA).

Pros and Cons

Pros

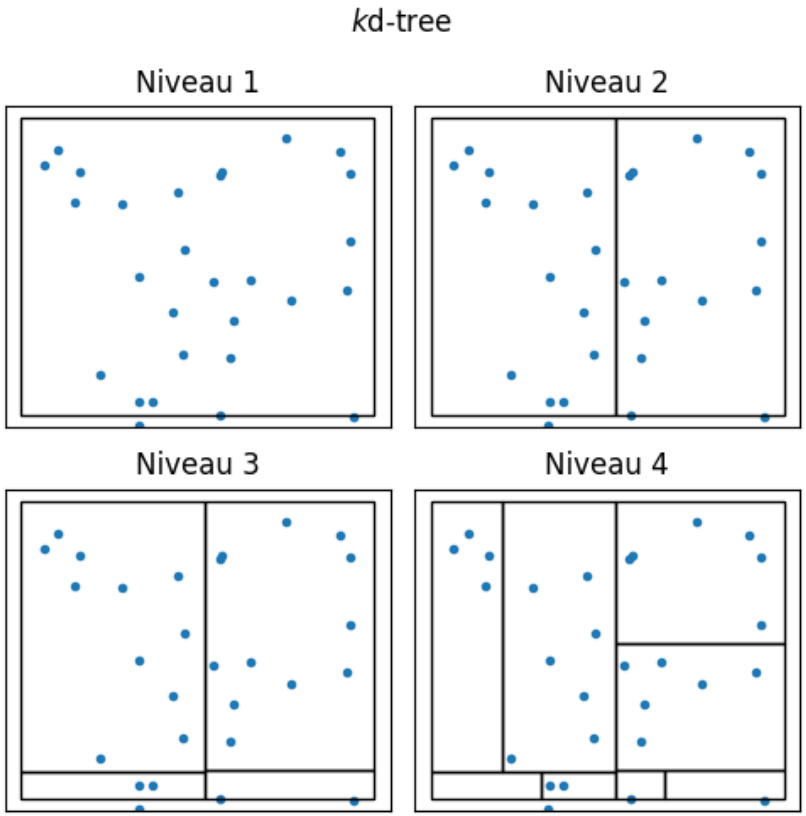
- Exact KNN, but approximation can be used e.g. no backtracking in parent nodes.
- Easy to implement.
- Average inference complexity: $O(d.log_2(n))$ and $O(d.n)$ with KNN.

Cons

- Cuts are axis-aligned: does not generalize well to higher dimensions

K-D TREE

```python
from sklearn.neighbors import KDTree
X = ...
tree = KDTree(X, leaf_size=2)
d, indices = tree.query(X[:1], k=5)
print(indices) # Indices of the first 5 neighbors
```



Motivation

KNN requires to store the full dataset to make a prediction. n large $\rightarrow$ intractable.
Most data are not random and usually concentrate in regions with the same predicted class or regression value $\rightarrow$ k-d tree.

Goal

Solve a classification or a regression problem.

What is critical is to identify areas where all points have the same class label.

Decision trees

Leverage the idea that a data point has the same class label or same regression value when it falls into a cluster of same label or same regression value.
$\Rightarrow$ There is no need to load the full training set for inference.
$\Rightarrow$ Build and load a tree structure that recursively splits the feature space into regions with similar label/value.

Definition

Decision trees/Regression trees

Predict the class (or the value) of an object x by a series of tests on the features that describe x.
Tests are organized in such a way that the answer to one of them indicates the next test to do on x structuring the tests into a tree.

Construction

1. Init : a root containing Z
2. Iteration: each node is split into several nodes. Each element of Z goes into one node only
3. Stop: when ... see next

$\Rightarrow$ Recursive partition of each node according to the value of the feature tested at each iteration.
$\Rightarrow$ Feature choice: based on the maximization of a homogeneity measure of the descendants with respect to the target variable.

STOPPING RULE AND AFFECTATION

The growth of the tree stops at a given node = terminal node (*leaf*) when:

Classification

- it is homogeneous $\Rightarrow$ predicted value = y
- it reaches a maximum depth (`max_depth`)
- there is no admissible partition left (`min_samples_split`,criterion)
- it contains a number of observations less than some value (`min_samples_leaf`)

Regression

Predicted value associated to a leaf = mean of the values of the y_i's among the observations belonging to this terminal node.

ID3

ID3(Z)
if $\forall\, i \in [\![1, n]\!]\ y_i = \tilde{y}$=*constant* **then**
  return *a tree with a node containing* $\tilde{y}$
end
else
  (i) Let $k \in [\![1, d]\!]$ the feature with the highest gain $G(Z, k)$.
  (ii) $\{k_1 \cdots k_m\}$: modalities of the feature k.
  (iii) $\{Z_1 \cdots Z_m\}$: subsets of Z having value $k_1 \cdots k_m$ for k.
  (iv) Build the tree with root $k$and subtrees ID3(Z_i),$i \in [\![1, m]\!]$, connected to k with an edge labeled with $k_i,\ i \in [\![1, m]\!]$.
end

```
from sklearn.tree import DecisionTreeClassifier
X, y = ...
dt = DecisionTreeClassifier()
dt = dt.fit(X, y)
tree.plot_tree(dt)
```

INFERENCE WITH DECISION TREE

Inference

- Once the tree is constructed, there is no need to keep the training set in memory.
- What we need to store
  1. The tree structure, depth $\leq log_2(n)$
  2. Class probability/regression value in the leaf nodes.
- Decision tree does not require any distance computation.
- The cut is based on feature value.

$\Rightarrow$ inference is very fast $\leq O(log_2(n))$, independent of d.

Gain

Entropy

- $C_k = \{x \in Z, y = y_k\},\ k \in [\![1, C]\!]$.
- $for\ x \in Z,\ P(x \in C_k) \approx p_k = \frac{|C_k|}{|Z|}$,
- Entropy of Z

$$H(Z) = -\sum_{k=1}^{C} p_k log_2\ p_k$$

  - if $H(Z) = 0$, all $x \in Z$ belong to the same class
  - $H(Z)$ is maximum if all the p_k are equal

Gain: information that is gained by splitting a set of data points.

For a feature k, the gain is computed as follow

1. Z is partitioned w.r.t. values of k in m subsets $\{Z_1 \cdots Z_m\}$
2. $p_i = P(x \in Z_i) \approx p_i = \frac{|Z_i|}{|Z|}$,
3. Information gain on k: $G(Z, k) = H(Z) - \sum_{i=1}^{m} p_i H(Z_i)$.

EXAMPLE

- 4 features: C, T, H et V
- 14 situations
- decision y: play golf

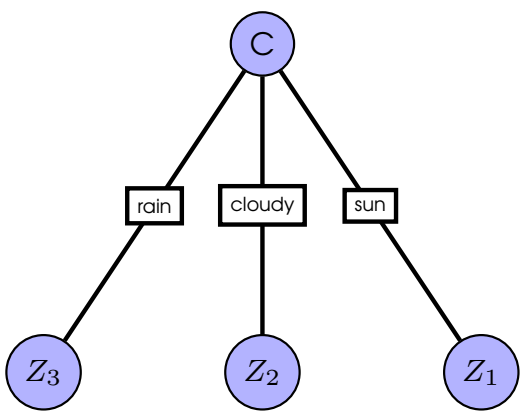
x	C	T	H	V	y
x_1	sun	hot	high	no	0
x_2	sun	hot	high	yes	0
x_3	cloudy	hot	high	no	1
x_4	rain	good	high	no	1
x_5	rain	cold	normal	no	1
x_6	rain	cold	normal	yes	0
x_7	cloudy	cold	normal	yes	1
x_8	sun	good	high	no	0
x_9	sun	cold	normal	no	1
x_{10}	rain	good	normal	no	1
x_{11}	sun	good	normal	yes	1
x_{12}	cloudy	good	high	yes	1
x_{13}	cloudy	hot	normal	no	1
x_{14}	rain	good	high	yes	0

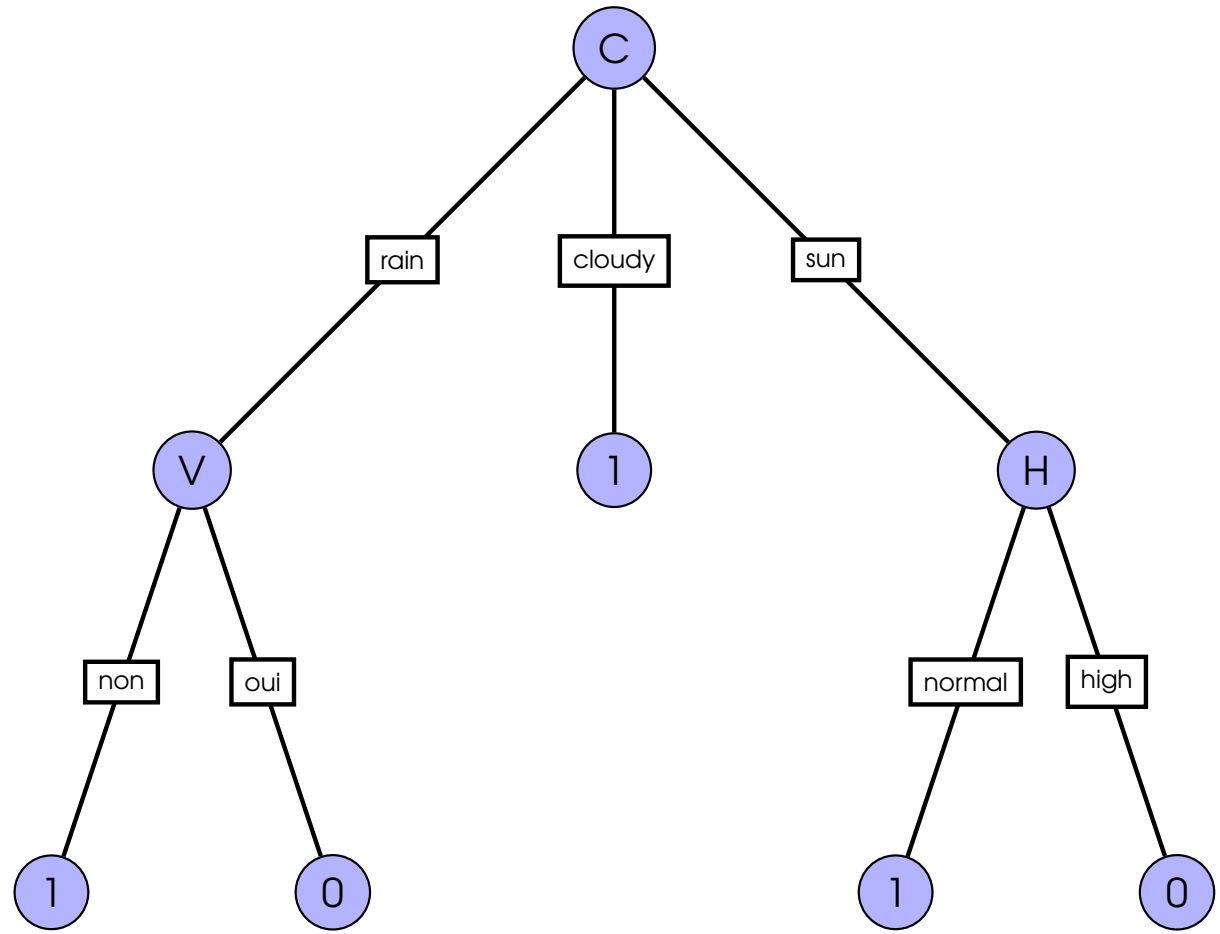
Init: root containing $Z = \{(\boldsymbol{x}_i, y_i), i \in [\![1, 14]\!]\}$.
First step: $q_0 = 5/14$, $q_1 = 9/14$, $H(Z) = 0.41 + 0.53 = 0.94$.
For each feature:

C	$y = 1$	$y = 0$	p_i	$H(Z_i)$
sun	2	3	5/14	0.971
cloudy	4	0	4/14	0
rain	3	2	5/14	0.971

and $G(Z, C)$ = 0.247.
For the three other features $G(Z, T)$=0.029, $G(Z, H)$=0.152 et $G(Z, V)$=0.048.
$\Rightarrow C$ is retained.

$Z_1 = \{x_1, x_2, x_8, x_9, x_{11}\}$, $Z_2 = \{x_3, x_7, x_{12}, x_{13}\}$, $Z_3 = \{x_4, x_5, x_6, x_{10}, x_{14}\}$.





OTHER MEASURES OF GAIN

A binary attribute a split each subset n_j in two parts of cardinality l_j (a=TRUE) and r_j (a=FALSE).

If $l = \sum_{j=1}^{C} l_j$ and $r = \sum_{j=1}^{C} r_j$:

- l_j/n and r_j/n are estimates of $P(a = TRUE, y = y_j)$ and $P(a = FALSE, y = y_j)$.
- l/n and r/n are estimates of $P(a = TRUE)$ and $P(a = FALSE)$.
- n_j/n is an estimate of $P(y = y_j)$.

Measures

- Gini index:

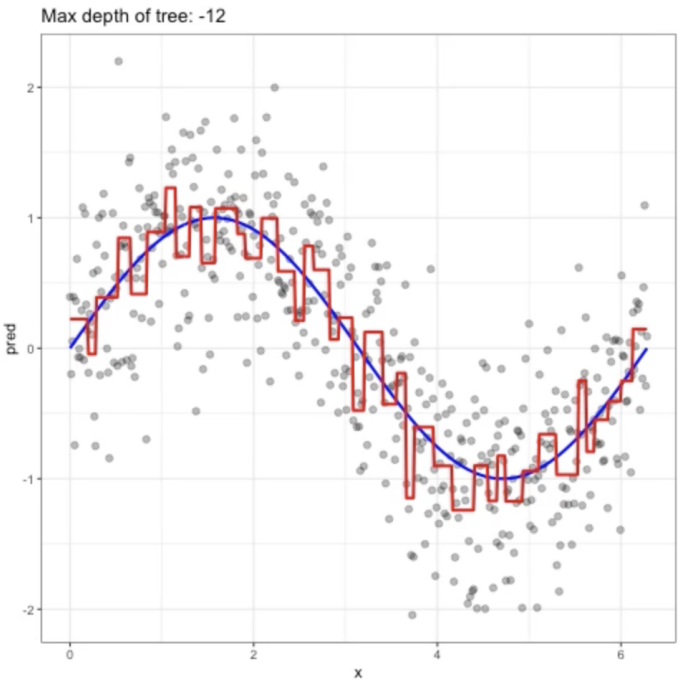
$$Gini(y \mid a) = \frac{l}{n} \sum_{j=1}^{C} \frac{l_j}{n_j}(1 - \frac{l_j}{n_j} + \frac{r}{n} \sum_{j=1}^{C} \frac{r_j}{n_j}(1 - \frac{r_j}{n_j})$$

- χ^2 criteria:

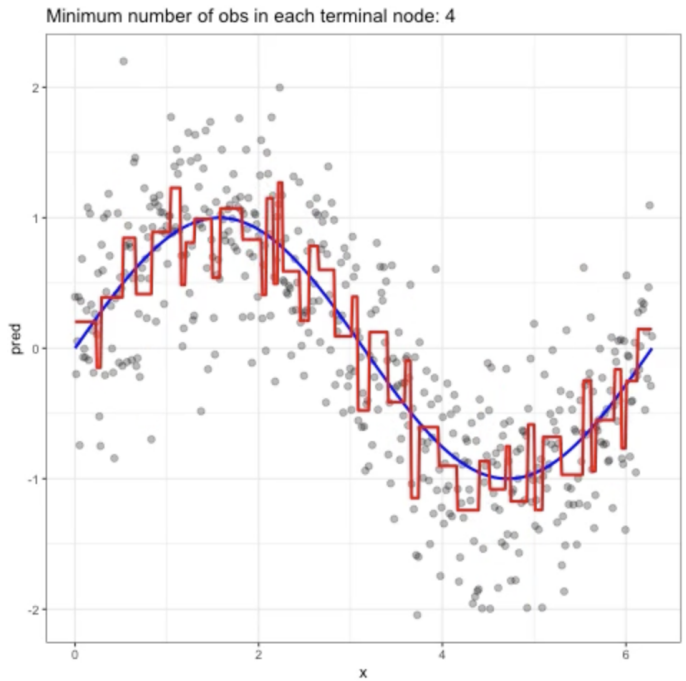
$$\chi^2(c \mid a) = \sum_{j=1}^{C} (\frac{l_j - (ln_j/n)}{\sqrt{ln_j/n}})^2 + (\frac{r_j - (rn_j/n)}{\sqrt{rn_j/n}})^2$$

REGRESSION TREES

It is straightforward to extend decision tree to regression.
Replace $H(Z)$ with the variance of Z



Limit tree depth



Minimum node size

Pruning trees

- Tradeoff between maximal tree (overfits) and the constant tree (too rough)
- Nice theory to find an optimal tree, minimizing prediction error penalized by complexity (number of leaves)

Notations

Complexity of T: $|T|$ number of leaves
Adjustment error of T

$$D(T) = \sum_{i=1}^{|T|} D_i$$

D_i: heterogeneity of leaf i.

SEQUENCE OF TREES

Adjustment error penalized by the complexity:

$$\mathcal{C}_\gamma(T) = D(T) + \gamma|T|$$

$\rightarrow$ $\gamma = 0$: maximal tree T_{max} minimizes $\mathcal{C}_\gamma(T)$

$\rightarrow$ $\gamma \nearrow$ the division for which the improvement of D is smaller than γ is cancelled and

- two leaves are pruned
- new tree

$\Rightarrow$ Sequence of trees $T_{max} \supset T_1 \supset T_2 \cdots T_K$: Breiman's sequence.

OPTIMAL TREE

1. Compute T_{max}
2. Compute Breiman's sequence $T_1 \supset T_2 \cdots T_K$ associated to the sequence of parameters $\gamma_1, \cdots \gamma_K$
3. For $v = 1$ to V (V-fold cross validation error)
   - 3.1 For each sample composed of $V - 1$ folds, estimate the sequence of trees associated with $\gamma_1, \cdots \gamma_K$
   - 3.2 Estimate the error on the validation fold.
4. For each $\gamma_1, \cdots \gamma_K$ compute the mean of these errors.
5. Determine the optimal value γ_{opt} minimizing the error mean.
6. Retain the tree corresponding to γ_{opt} in $T_1 \supset T_2 \cdots T_K$