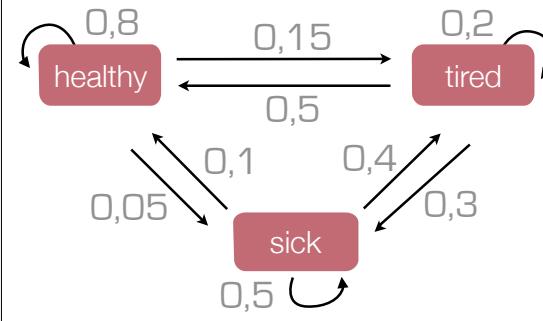


## Hidden Markov Models

Thierry Parmentelat — INRIA  
[thierry.parmentelat@inria.fr](mailto:thierry.parmentelat@inria.fr)

<https://github.com/parmentelat/mig2013>

## Markov Models



- model :  $\lambda = (A, \pi)$
  - A: Transitions
  - $\pi$ :  $p(\text{initial state}=i)$
- |     |      |      |          |
|-----|------|------|----------|
| 0,8 | 0,15 | 0,05 | $\sum=1$ |
| 0,5 | 0,2  | 0,3  |          |
| 0,1 | 0,4  | 0,5  |          |
- A
- |     |     |     |
|-----|-----|-----|
| 0,7 | 0,2 | 0,1 |
|-----|-----|-----|
- $\pi$

## Markov Models

- Markov hypothesis
  - behaviour depends only on current state (**not** on history)
- Observation
  - $E_1 E_2 .. E_t ..$  : sequence of states,  $E_i$  in  $\{1..N\}$
- Problems
  - (I) Probability of a given sequence
  - (II) Probability to observe state i at time t

## Notations

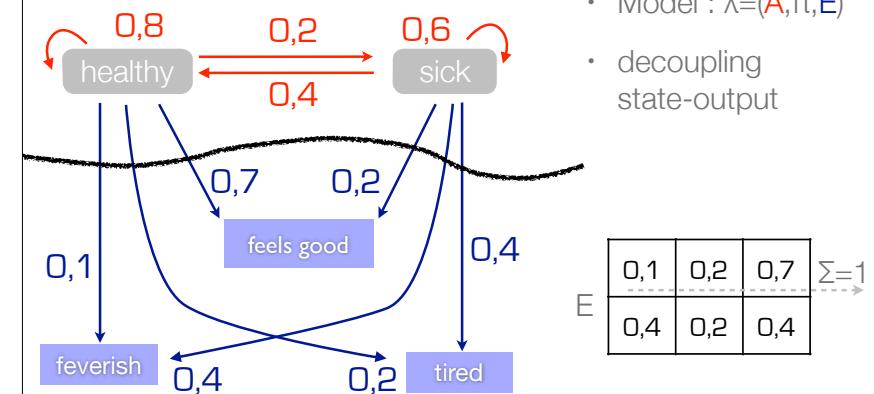
- Mathematical:  $A_{ij}$
- Computer Science:  $A[i][j]$
- Example Problem (I)
  - Maths: sequence  $E_1 E_2 .. E_t$   
 $\text{proba} = \pi(E_1) * \prod_{i=2..t} A_{(E_{i-1})(E_i)}$
  - CS: sequence  $\text{seq}[t]$   
 $\text{proba} = \pi[\text{seq}[0]] * \prod_{i=2..t} (A[\text{seq}[i-1]] [\text{seq}[i]])$

From now on,  
we will have  
all indices start at 0

## Python features, and modules of interest

- range                  `range(5) -> [0, 1, 2, 3, 4]`                  `range(1,5) -> [1, 2, 3, 4]`
- [ function(x) for x in inputs ]    ou [ function(x) for x in inputs if condition(x) ]
- built-in sum              `sum ( range(5) ) -> 10`
- module operator    `from operator import add, mul`
- reduce                  `reduce (add, range(5), 0) -> 10`    `reduce (mul, range(1,5), 1) -> 24`
- numpy                  [http://wiki.scipy.org/Tentative\\_NumPy\\_Tutorial](http://wiki.scipy.org/Tentative_NumPy_Tutorial)
- matplotlib              <http://matplotlib.org>

## Hidden Markov Models



## Two major kinds of Hidden Markov Model

- Discrete: Emission among a **finite** set of observations
- Continuous: Emission in  $R^d$  - second chapter

## More on notations

- **sequence** : a list of observations (a.k.a. **seq**)
- **path** : a list of states
- indices:
  - **i,j** : for denoting states (**N**: nb of states)
  - **t** : for denoting time (**T**: max time for a given seq/path)
  - **o** : for denoting outputs/observations (**O**: nb of signals)

## Classical problems in HMM's

- Reference paper - Lawrence Rabiner  
A tutorial on Hidden Markov Models...  
<http://www.cs.ubc.ca/~murphyk/Bayes/rabiner.pdf>
- (I) Probability to observe a given sequence  
algs : **forward** and **backward** (a.k.a. alpha and beta)
- (II) Decoding: estimate (most probable) sequence of states that produce a given sequence - algo: **Viterbi**
- (III) Learning: given observed sequences, and from a given HMM, find an HMM that is more likely to produce them  
algo: **Baum-Welch**

## Probability for a given sequence of observations

- Naive algorithm:
  - probability for observing sequence along a given path
  - sum on all state paths
$$P(\text{seq}/\text{path}) = \pi(\text{path}_0) * \prod_{t=1}^{T-1} A_{\text{path}_{t-1}, \text{path}_t} * \prod_{t=0}^{T-1} E_{\text{path}_t, \text{seq}_t}$$

$$P(\text{seq}) = \sum_{\text{path} \in N^T} P(\text{seq}/\text{path})$$
- unpracticable, we have  $N^T$  paths to sum on

## Forward

- greedy variation
- alpha[i][t]**  
probability to observe seq **until t** and end up in **state i**
- initialization  

$$\alpha_{i,0} = \pi_i * E_{i,\text{seq}_0}$$
- for i in range(N):  

$$\alpha_{i,0} = \pi_i * E_{i,\text{seq}_0}$$

$$\alpha_{i,0} = \pi_i * E_{i,\text{seq}_0}$$

## Forward - continued

- A: NxN  
E: NxO  
Pi: N  
sequence: T
  - Recurrence  

$$\alpha_{j,t+1} = \sum_{i=0}^{N-1} \alpha_{i,t} * A_{i,j} * E_{j,\text{seq}_{t+1}}$$
  - alpha[j,t+1]=\**  
reduce (add, [  $\alpha_{i,t} * A_{i,j} * E_{j,\text{seq}_{t+1}}$ ] for i in range(N))
- 
- The diagram illustrates the forward pass in an HMM. At time t, states S1, S2, ..., SN are shown. At time t+1, state Sj is shown. Transitions from state Si at time t to state Sj at time t+1 are labeled  $a_{ij}$ . The emission probability of observing sequence  $\text{seq}[t+1]$  from state  $S_j$  is labeled  $E_{j,\text{seq}[t+1]}$ . The forward variable  $\alpha$  is represented by a grey cylinder that accumulates probability over time, starting from the initial state  $S_i$  at time t.

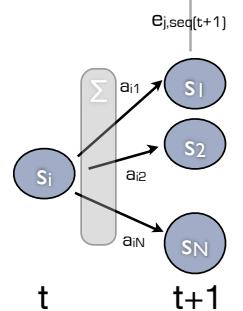
## Forward - finalization

- Original purpose : compute proba (seq)
  - obtained by summing all  $\alpha[i][T]$
- Note that the details of alpha are needed as well
  - as they will be re-used in the other algorithms
- Complexity of this algorithm
  - $\Theta(N^2 * T)$  in space and time

## Backward

- Same idea as *forward*, except .. the other way around
- $\beta[i][t]$**   
probability to be in **i** at **t** and observe seq **from t+1** and on
- $\beta[i][T-1] = 1$
- $\beta[i][t] = <\text{yours to say}>$
- final result

$$P(\text{seq}) = \sum_{i=0}^{N-1} \pi_i * E_{i,\text{seq}_0} * \beta(i, 0)$$



## Implementation

- Use your locally installed python (v2 recommended ?)
- To implement a function that can be used like this  
(proba, alpha) = forward (Pi, A, E, sequence)
- Same for backward
- Using numpy or not - your call

## unfaircasino — a study model

2 states

A	0,9	0,1
	0,1	0,9

fair dice  
biased dice

E	1/6	1/6	1/6	1/6	1/6	1/6
	3/13	2/13	2/13	2/13	2/13	2/13

(•) (••) (•••) (••••) (•••••) (••••••)

starting with the fair dice

$\pi$	1	0
-------	---	---

## Available datasets

- For comparing your results: see datasets/

## Viterbi – details

- Initialization

$$\text{delta}[i][0] = P[i] * E[i][\text{sequence}[0]]$$

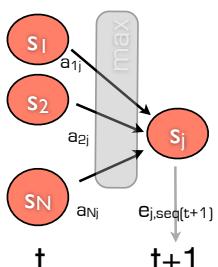
`psi[i][0] = None`

- Recurrence

```
delta[j][t+1] = \
```

`max ( [ delta [i][t] * A[i][j] ] for i in range(N) ) * E[j][seq[t+1]]`

$\text{psi}[j][t] = \text{argmax} (\text{ same expression })$



## Viterbi – decoding

- Given an observation sequence  
find out most likely path that has caused this sequence  
a.k.a. **Viterbi path**

- **delta[i][t]**

best probability of path **until t** that **ends up at i**  
(and of course emits sequence until t)

- **psi[i][t]**

state at  $t-1$  for the path that corresponds to  $\delta$   
 $\psi[i][0]$  undefined / does not matter

## Viterbi – a useful trick

```
3. ~  
⌘1  
>>> l=[(3,'three'), (5,'five'), (1,'one')]  
>>> max(l)  
(5, 'five')  
>>> |
```

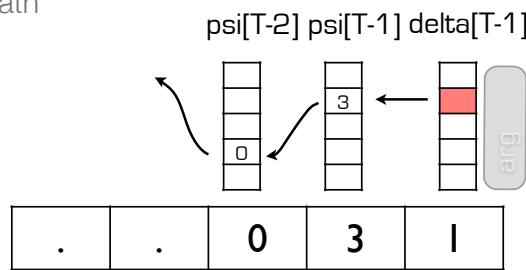
```
(proba, rank) = max ( [ ( delta[i][t-1] * A[i][j] , i ) for i in range(N) ] )
```

$$\delta[j][t] = \text{proba} * E[j][\text{seq}[t]]$$

$$\psi[j][t] = \text{rank}$$

## Viterbi - finalization

- retrieve full path



- Viterbi is a greedy algorithm too
  - $\Theta(N^2 * T)$  in space and time

## Implementation

- implement a function that can be used like this

(proba, path, delta, psi) = viterbi (Pi, A, E, sequence)

- (although delta and psi are not really useful outside)

## One possible implementation for forward

```
# iterative style
def forward (Pi, A, E, sequence):
    N = len(A)
    T = len(sequence)
    states = range(N)

    alpha = [ [ 0. for t in range(T) ] for i in states ]

    # time=0
    for i in states: alpha[i][0]=Pi[i] * E[i][sequence[0]]

    # fill in from time=1
    for t in range (1,T):
        for j in states:
            for i in states:
                alpha[j][t] += alpha[i][t-1]*A[i][j]
            # multiply once rather than for each iteration
            alpha[j][t] *= E[j][sequence[t]]
    # resulting proba is a sum from that last time column
    total = 0.
    for i in states: total += alpha[i][T-1]
    return (total, alpha)
```

## One possible implementation for backward

```
from operator import add
# return ( proba, beta ) -- beta dimension: beta[i][t]
# functional style
def backward (Pi, A, E, sequence):
    N = len(A)
    T = len(sequence)
    states = range(N)

    beta = [ [ 0. for t in range(T) ] for i in states ]

    # beta starts with 1's on last time
    for i in states: beta[i][T-1]=1.

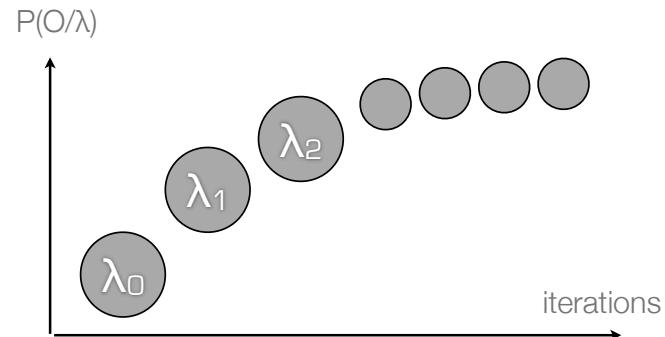
    # starting with time but last, i.e. T-2
    for t in range (T-2, -1, -1):
        for i in range (N):
            beta[i][t] =
                reduce (add, [ beta[j][t+1] * A[i][j] * E[j][sequence[t+1]] for j in states ])
    # resulting proba, summing on time=0
    total = reduce (add, [ Pi[i]*E[i][sequence[0]]*beta[i][0] for i in states ])
    return (total,beta)
```

## Baum-Welch — learning

- Problem: given a set of output sequences  
Find out the “best” HMM that would give this out
- Improve model  $\lambda = (A, E, \Pi)$  by successive iterations
- maximize likelihood over  $\lambda$   
likelihood defined as the **product** of probabilities of all sequences in a sample
- in our case, multiply  $P(O/\lambda)$  over sequences

## Expectation Maximization (EM)

- Baum-Welch is a special case of  
“Expectation - Maximization” class of algorithms



## Baum Welch basics

- Given sequence seq, and model  $\lambda$
- $\xi_{t[i][j]} \quad \xi_{t[i,j]}$   
probability to be in state **i** at time **t**, and in state **j** at **t+1**
- $\gamma_{t[i]}$   
probability to be in state **i** at time **t**

## computing $\xi_t$ and $\gamma_t$

- $\xi_t(i, j)$  probability to be in **i** at **t**, **j** at **t+1**(knowing seq)

$$\xi_t(i, j) = \frac{\alpha_t(i) A_{i,j} E_{j,seq_{t+1}} \beta_{t+1}(j)}{P(seq/\lambda)}$$

- $\gamma_t(i)$  probability to be in **i** at **t** (knowing seq)

$$\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i, j)$$

## Rationale for re-estimation

$$\sum_{t=0}^{T-2} \gamma_t(i) = \text{expected number of transitions from state } i$$

$$\sum_{t=0}^{T-2} \xi_t(i, j) = \text{expected number of transitions from state } i \text{ to } j$$

- note that one instant is not taken into account  
we are counting transitions, so there are T-1  
will be used for re-estimating PI instead

## Re-estimation

$$\bar{\pi}_i = \gamma_0(i)$$

$$\bar{A}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i}$$

$$= \frac{\sum_{t=0}^{T-2} \xi_t(i, j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$$

$$\bar{E}_{jo} = \frac{\text{expected number of times in } j \text{ and observing } o}{\text{expected number of times in } j}$$

$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j)}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

reminder from previous slides

\* in **i** at **t** and in **j** at **t+1**

$$\xi_t(i, j) = \frac{\alpha_t(i) A_{i,j} E_{j,seq_{t+1}} \beta_{t+1}(j)}{P(seq/\lambda)}$$

\* in **i** at **t**

$$\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i)(j)$$

## Convergence criteria

- monitor overall P(seq) at each iteration
- should grow asymptotically
- stop when  $(P'/P) < (1+\varepsilon)$
- $\varepsilon$  passed as argument

## Deal with multiple sample sequences

$$\bar{\pi}_i = \gamma_0(i) \quad \xrightarrow{\text{Average on sequences}}$$

$$\bar{A}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i} \quad \begin{matrix} \xrightarrow{\text{Sum over sequences}} \\ \xrightarrow{\text{Sum over sequences}} \end{matrix}$$

~~$$= \frac{\sum_{t=0}^{T-2} \xi_t(i, j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$$~~

$$\bar{E}_{jo} = \frac{\text{expected number of times in } j \text{ and observing } o}{\text{expected number of times in } j} \quad \begin{matrix} \xrightarrow{\text{Sum over sequences}} \\ \xrightarrow{\text{Sum over sequences}} \end{matrix}$$

~~$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j)}{\sum_{t=0}^{T-1} \gamma_t(i)}$$~~

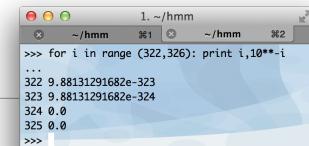
## Implementation

- implement a function that can be used like this

```
(PI', A', E', proba, iteration,time) = \
baum_welch (PI, A, E,sequenceS,\ 
max_iters=100, convergence=1.e-4)
```

## Continuous Hidden Markov Models

## Scaling



```
1. ~/hmm
~/hmm #1 ~/hmm #2
>>> for i in range (322,326): print i,10**-i
...
322 9.88131291682e-323
323 9.88131291682e-324
324 0.0
325 0.0
>>>
```

- If  $P_{\max}$  is the maximum of the terms in A and E
- Order of magnitude for e.g.  $\alpha[i][t]$  :  $P_{\max}^{2T}$
- Even if  $P_{\max}$  is, say, 0.9, T=300 gives you fairly small values
- Quickly reach limits of hardware implementation
- Solution: for each time t  
select some relevant factor (e.g. based on sum of values)  
store in memory real value multiplied by factor

## Continuous Hidden Markov Model

- Replace finite set of O signals (outputs)
- With a finite set of O gaussian distributions in  $R^d$
- Each of these being defined by
  - an average  $\mu = \{\mu_1, \mu_2, \dots, \mu_d\}$
  - a covariance matrix  $\sigma_{ij}$  in dimension d
- Then each state is attached a linear stochastic combination of these gaussians (a **mixture**)

## Continuous HMM – example

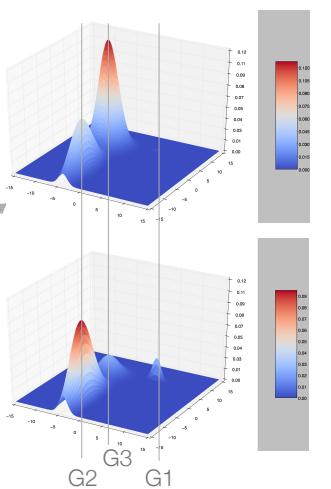
- 3 Outputs:

$$\begin{aligned} G1 : \mu = (4., 8.), & \quad \sigma = (1., 0.25) \\ G2 : \mu = (-4., -8.), & \quad \sigma = (1., 4.) \\ G3 : \mu = (-6., 6.), & \quad \sigma = (2., 2) \end{aligned}$$

say  $G1 \sim$  tired,  
 $G2 \sim$  feverish,  $G3 \sim$  feels good

- 2 states:

$$\begin{aligned} E[1] : (.1, .1, .8) \\ E[2] : (.7, .15, .15) \end{aligned}$$



## General form of d-dimension gaussian

- a.k.a. multivariate normal distribution (wiki)
- $\mu$  average;  $\Sigma$  covariance  
 $\Sigma$  is positive semidefinite and symmetric

$$\mu \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d,d}, G_{\mu, \Sigma} : \mathbb{R}^d \mapsto \mathbb{R}$$

$$G_{\mu, \Sigma}(X) = \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

- check out “covariance matrix” in wiki

## Gaussian distribution - single dimension

- a.k.a. Normal Distribution (wiki)

$$f(x) = e^{-x^2}$$

- normalize

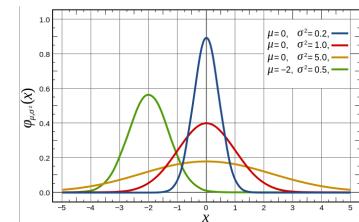
$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

- translate

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-\mu)^2}$$

- scale

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$



## Degenerate case

- Diagonal covariance matrix (unrelated data)

$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & \dots & \dots & 0 \\ 0 & \sigma_2^2 & \dots & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \sigma_{d-1}^2 & 0 \\ 0 & \ddots & \ddots & 0 & \sigma_d^2 \end{pmatrix}$$

$$G_{\mu, \Sigma}(X) = \frac{1}{\prod_{i=1}^d \sigma_i \sqrt{(2\pi)^d}} e^{-\frac{1}{2} \sum_{i=1}^d \left( \frac{x_i - \mu_i}{\sigma_i} \right)^2}$$

## Continuous HMM theory (See Rabiner p. 267)

- each state has a pdf that is a mixture of  $m$  Gaussians
  - in the most general framework where an estimation method is known, not all  $m$  Gaussians are equal
- $$p_j(X) = \sum_{m=0}^{M-1} c_{jm} G_{\mu_m, \Sigma_m}[X] \quad \sum_m c_{jm} = 1$$
- that is a total of  $n * m$  gaussians in the model
- Note** the example above considers  $m$  gaussians in total

$$p_j(X) = \sum_{m=0}^{M-1} c_{jm} G_{\mu_m, \Sigma_m}[X] \quad \sum_m c_{jm} = 1$$

## Model summary - case (2)

- $\Pi$  ( $N$ )
- $A$  ( $N \times N$ )
- $E$  ( $N \times O$ )
- $\text{seq}$  ( $T$ )
- $\Pi$  ( $N$ )
- $A$  ( $N \times N$ )
- $C$  ( $M \times N$ )
- $G_{\mu}$  ( $M \times D$ )
- $G_{\sigma}$  ( $M \times D \times D$ )
- $\text{seq}$  ( $T \times D$ )

## Adapting discrete to continuous - $\alpha$ , $\beta$ & viterbi

- For forward and backward
- we had terms like " $E[j][\text{seq}[t]]$ "
- that corresponds to the probability to emit output  $\text{seq}[t]$  in state  $j$
- now  $\text{seq}[t]$  is a  $d$ -dimension vector
- we can compute this probability directly from corresponding gaussian mixture

$$p_j(X) = \sum_{m=0}^{M-1} c_{jm} G_{\mu_m, \Sigma_m}[X]$$

## Re-estimation in the discrete case

✓  $\bar{\pi}_i = \gamma_0(i)$

✓  $\bar{A}_{ij} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i}$

$$= \frac{\sum_{t=0}^{T-2} \xi_t(i, j)}{\sum_{t=0}^{T-2} \gamma_t(i)}$$

✗  $\bar{E}_{jo} = \frac{\text{expected number of times in } j \text{ and observing } o}{\text{expected number of times in } j}$

$$= \frac{\sum_{t=0}^{T-1} \xi_t(j, o) \gamma_t(j)}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

reminder from previous slides

\* in  $i$  at  $t$  and in  $j$  at  $t+1$

✓  $\xi_t(i, j) = \frac{\alpha_t(i) A_{ij} E_{j, \text{seq}_{t+1}} \beta_{t+1}(j)}{P(\text{seq}/\lambda)}$

\* in  $i$  at  $t$

✓  $\gamma_t(i) = \sum_{j=0}^{N-1} \xi_t(i)(j)$

## Adapting discrete to continuous - Baum-Welch

- **gamma[i,t,m]** probability to be in state **i** at time **t** with mixture **m** being responsible for that output

$$\gamma_t(i, m) = \left[ \frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)} \right] \left[ \frac{c_{im} G_{\mu_{im}, \sigma_{im}}(X_t)}{\sum_{k=0}^{M-1} c_{ik} G_{\mu_{ik}, \sigma_{ik}}(X_t)} \right]$$

## Re-estimation rules for continuous Baum-Welch

$$\overline{c_{im}} = \frac{\sum_t \gamma_t(i, m)}{\sum_t \sum_k \gamma_t(i, k)}$$

$$\overline{\mu_{im}} = \frac{\sum_t \gamma_t(i, m) \cdot X_t}{\sum_t \gamma_t(i, m)}$$

$$\overline{\Sigma_{im}} = \frac{\sum_t \gamma_t(i, m) \cdot (X_t - \mu_{im}) \cdot (X_t - \mu_{im})}{\sum_t \gamma_t(i, m)}$$

reminder from previous slide

\* in **i** at **t** and in from mixture **m**

$$\gamma_t(i, m) = \left[ \frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)} \right] \left[ \frac{c_{im} G_{\mu_{im}, \sigma_{im}}(X_t)}{\sum_{k=0}^{M-1} c_{ik} G_{\mu_{ik}, \sigma_{ik}}(X_t)} \right]$$

## Source Code Management systems

### Source Code Management systems

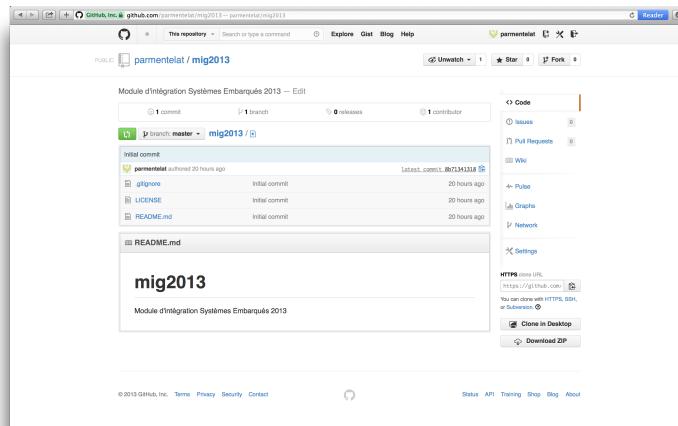
- purpose
  - collaborative work
  - keep track of changes
  - changes are done concurrently
  - teams do not necessarily know of each other
- git
  - <http://git-scm.com>
  - decentralized
  - used by many open source projects
- svn
  - <http://subversion.tigris.org>
  - deprecated / previous generation
  - simpler mental model, but centralized
- mercurial
  - <http://mercurial.selenic.com> is nice too

## git basics

- git init initialize git from a working directory
- git clone create a local working dir from remote
- git add put changes aside for next commit
- git commit create a commit
- git pull get changes from remote
- git push push your commits on remote
- git stash hide local changes under carpet

Check out UI tools : <http://www.sourcetreeapp.com>

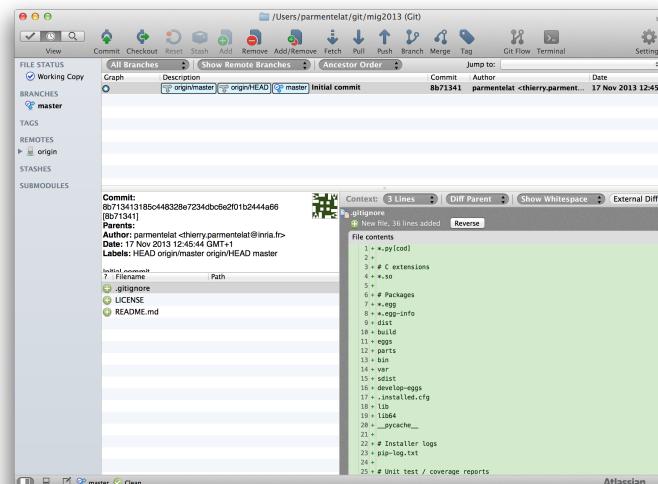
## example (1) - create repo in github



## example (2) - clone repo locally

```
3. ~/git/mig2013 %1
~/git $ git clone https://github.com/parmentelat/mig2013.git
Cloning into 'mig2013'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
Checking connectivity... done
~/git $ cd mig2013/
~/git/mig2013 $ ls
LICENSE      README.md
~/git/mig2013 $
```

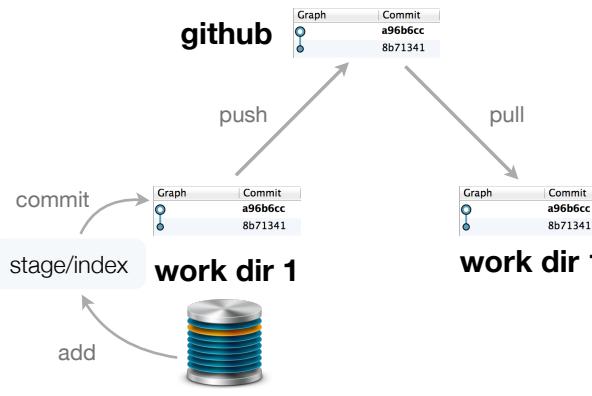
## example (3) - visualize local repo with sourcetree



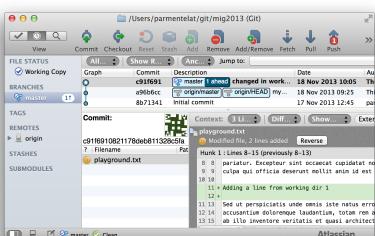
## example (4) - two working directories

```
$ git ls-files playground.txt
$ git add playground.txt
$ git commit -m "my message"
$ git ls-files playground.txt
playground.txt
$ git push
$ git ls-files playground.txt
$ playground.txt
```

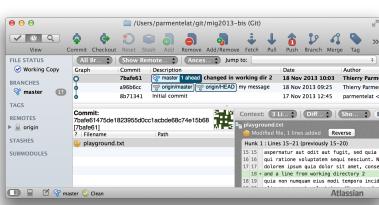
## example (5) - the repositories at work



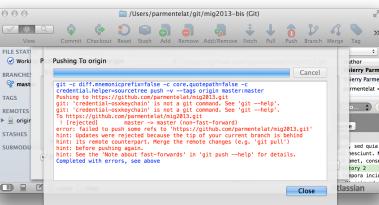
## example (6) - merging two concurrent changes



user 1 can push fine

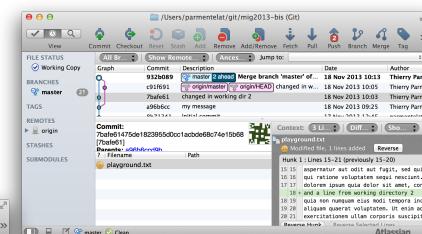


user 2 gets conflicts when pushing

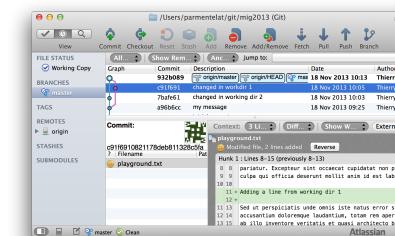


## example (7) - merging - continued

user 2 needs to pull first

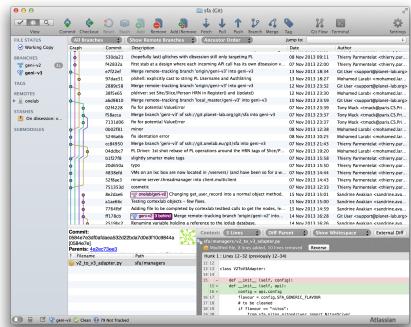


user 2 can then push fine



## Good practices

- Branches are easy to create, merge, delete..
- Use as many branches as needed
- Typically one per “feature”



## ssh basics

## ssh history and purpose

- successor of (very unsecure) *rsh* (remote shell)
- ssh = secure shell
- basic purpose : remote terminal
- advanced purpose : all sorts of secure tunnels / bouncing
- esp. e.g. rsync (to keep files in sync), git, ...

## ssh authentication mechanism(s)

- **password** authentication is **evil**, don't ever enable
- use **public key** authentication only
- ssh-keygen : create a key pair
- keep private key (*id\_rsa*), well... private
  - never out of your computer
  - watch out access rights
  - private keys are password-protected
- expose public key (*id\_rsa.pub*) to your peers

## ssh public key authentication - basics

---

- how to enable access
  - add public key in `~/.ssh/authorized_keys`
  - again watch out for access rights
- how it works
  - $E_{\text{Public}} \circ E_{\text{private}} = E_{\text{private}} \circ E_{\text{public}} = \text{Identity}$
  - challenge remote to “be” X (to have private key X)
  - send  $E_{\text{public}}(\text{message})$  over the wire
  - $E_{\text{private}}$  is required to decode message