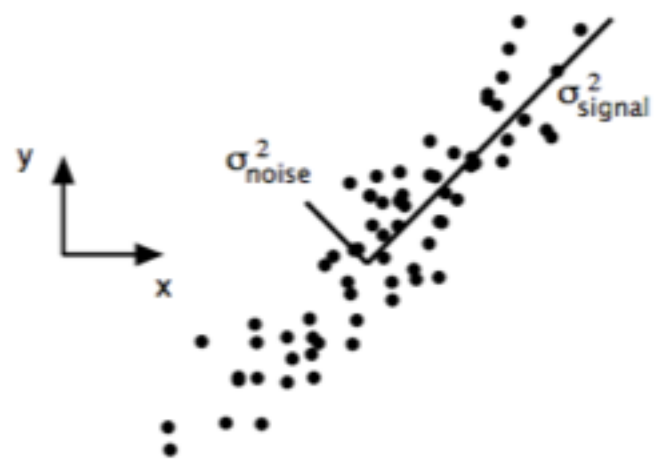
DAT1

12/3/16

Agenda

- Review: Dimensionality Reduction
- Decision Trees
- Recursion, HMMs

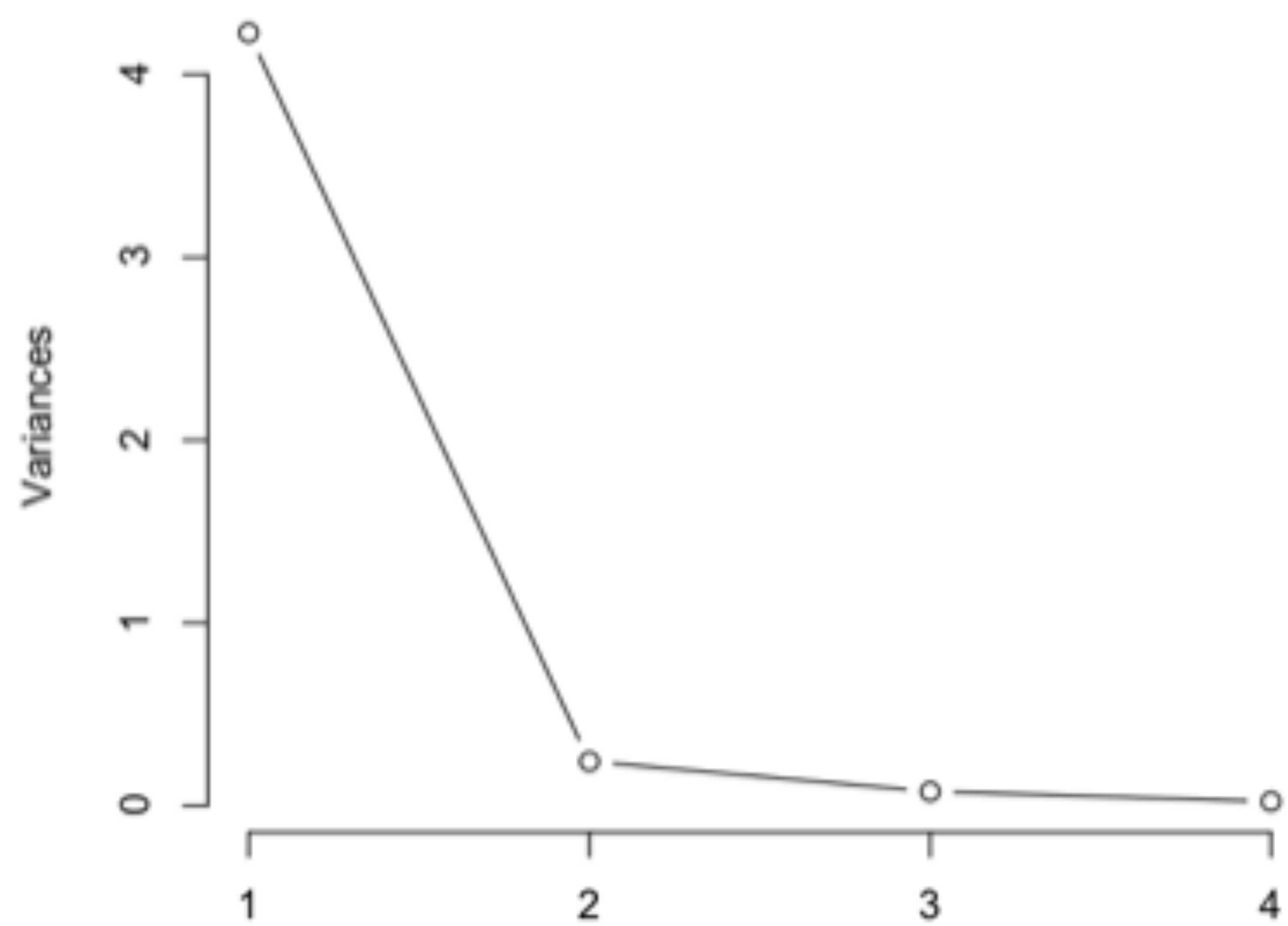
Review



$$C = \begin{bmatrix} \mathrm{E}[(X_1-\mu_1)(X_1-\mu_1)] & \mathrm{E}[(X_1-\mu_1)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_1-\mu_1)(X_n-\mu_n)] \\ \mathrm{E}[(X_2-\mu_2)(X_1-\mu_1)] & \mathrm{E}[(X_2-\mu_2)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_2-\mu_2)(X_n-\mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(X_n-\mu_n)(X_1-\mu_1)] & \mathrm{E}[(X_n-\mu_n)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_n-\mu_n)(X_n-\mu_n)] \end{bmatrix}.$$

$$C = Q \Lambda Q^{-1}$$

$$Cv = \lambda v$$


Variances
0
1
2
3
4
1
2
3
4

$$\underset{(n \times d)}{X} = \underset{(n \times n)}{U} \; \underset{(n \times d)}{\Sigma} \; \underset{(d \times d)}{V^T}$$

Recursion

Fibonacci relationship

$$F_1 = 1$$
$$F_2 = 1$$
$$F_3 = 1 + 1 = 2$$
$$F_4 = 2 + 1 = 3$$
$$F_5 = 3 + 2 = 5$$

In general :

$$F_n = F_{n-1} + F_{n-2}$$

or

$$F_{n+1} = F_n + F_{n-1}$$

1. Try coding this using a *for* loop

def fib(n): # returns n’th fib number

2. Try using recursion

def fib(n):
  # calls itself!

Recursion in data exploration and scraping

```
{
  "div": {
     "a": {
        "div": {
            "a": "foo"
        }
     }
  },

  "a": {
     "span": {
        "a": {
            "div": "moo"
        }
     }
  }

}
```

Write a recursive function to find all children of “a” tags

```
node {
  value e.g. “div”
  child # child node
}
```

final node’s value is *None*

you have access to the *root*

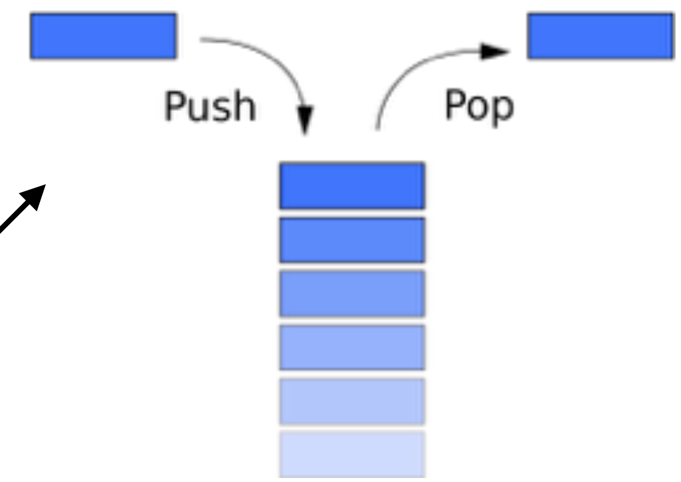
Scope and the call stack

```
def f():
  x = 2
  return x + 4


x = 3
print f()

# what is printed?
```

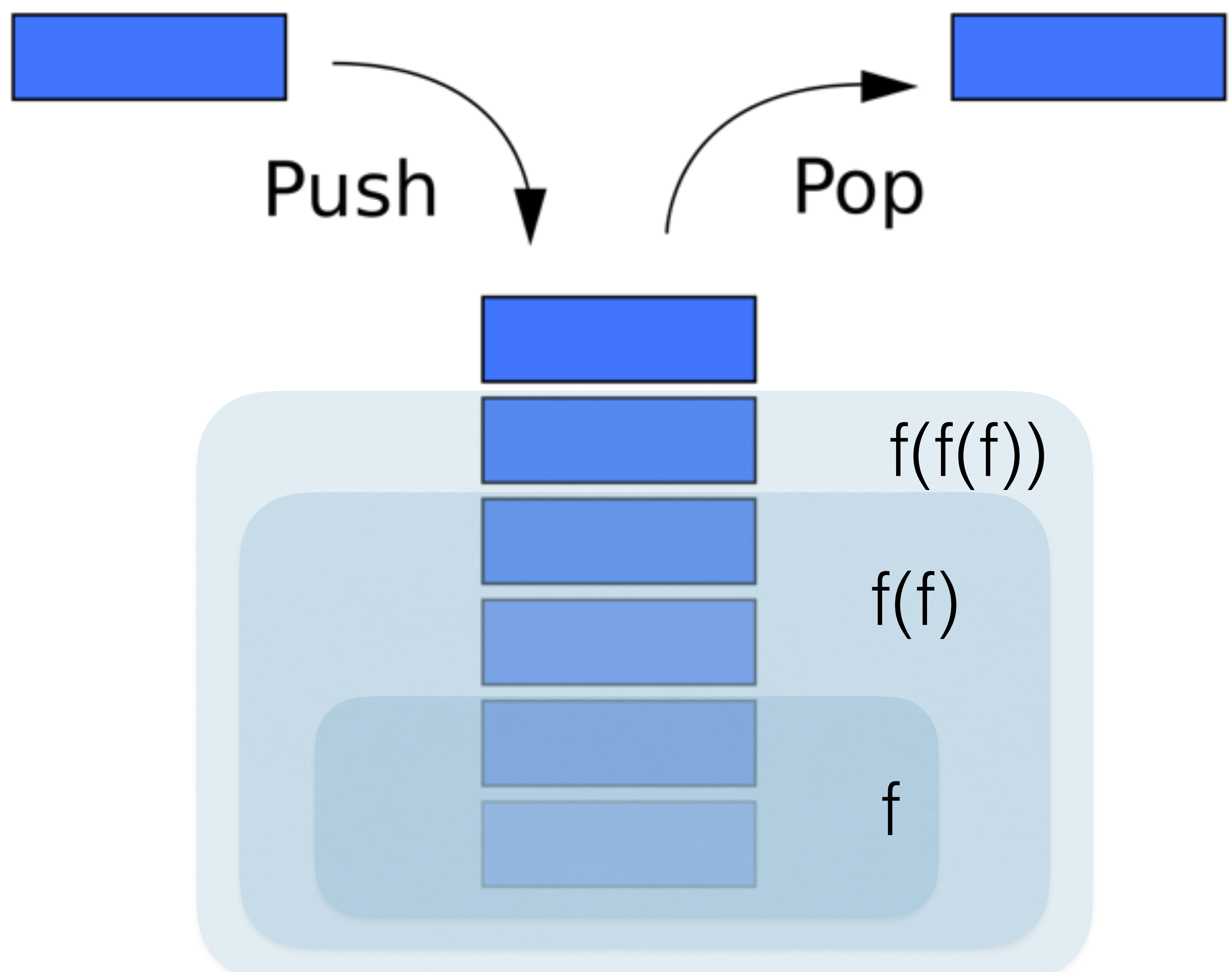
```
def f():
  x = 2
  return x + 4
```




```
x = 3
print f()

# what is printed?
```

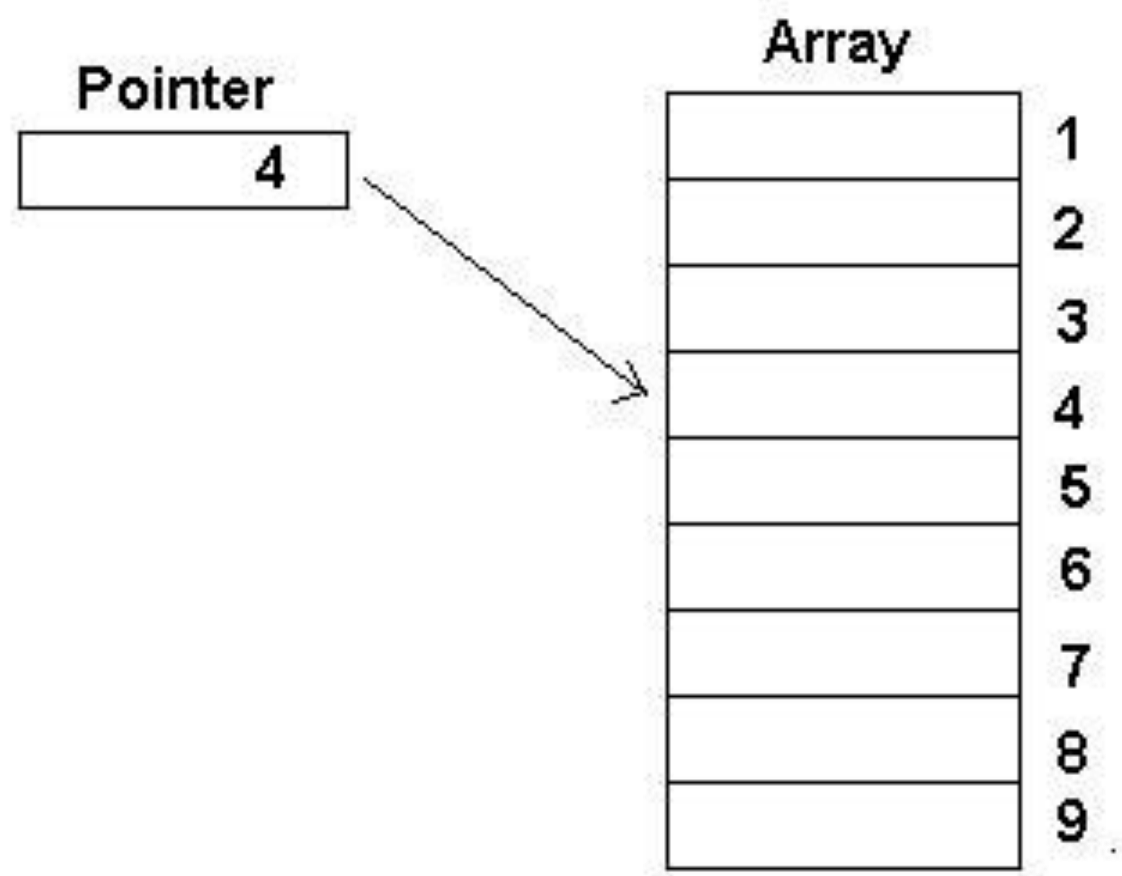
Recursion: function calling itself



Scala review
http://www.scala-lang.org/documentation/

Coding exercises
https://www.hackerrank.com/domains/fp/recursion

Pointers!



How pythons can bite!

```
arr = [ { … } , { … }, { … } ]
results = []

for …
  x = arr[i]
  x.moo = "foo"

  results.append(x)
```

What’s wrong?

…

results = [x, x, x, x]
x —> the last item

```
arr = [ { … } , { … }, { … } ]
results = []

for …
  x = copy.deepcopy(arr[i])
  x.moo = “foo”

  results.append(x)
```

Practice!

```
int *moo  // pointer
int moo // actual value

int moo = 3
&moo // address of value

int *moo
*moo = 3 // set value!
```

http://www.gdsw.at/languages/c/programming-bbrown/c_0771.htm

Hidden Markov Models

Sequence of words over time based on audio

Other examples?

Easy case: we observe the actual state

S = { rain, sun, cloud }

Observed over time:

z1 = sun, z2 = cloud, z3 = sun…

$$P(z_t|z_{t-1}, z_{t-2}, ..., z_1) = P(z_t|z_{t-1})$$

Simplifying Assumptions

Limited horizon: $P(z_t|z_{t-1}, z_{t-2}, ..., z_1) = P(z_t|z_{t-1})$

Stationary process: $P(z_t|z_{t-1}) = P(z_2|z_1);\ t \in 2...T$

Can you explain this in a picture?

State transition matrix

$$A = \begin{array}{c|cccc} & s_0 & s_{sun} & s_{cloud} & s_{rain} \\ \hline s_0 & 0 & .33 & .33 & .33 \\ s_{sun} & 0 & .8 & .1 & .1 \\ s_{cloud} & 0 & .2 & .6 & .2 \\ s_{rain} & 0 & .1 & .2 & .7 \end{array}$$

What's the prob of a given sequence?

$$
\begin{aligned}
P(\vec{z}) &= P(z_t, z_{t-1}, ..., z_1; A) \\
&= P(z_t, z_{t-1}, ..., z_1, z_0; A) \\
&= P(z_t|z_{t-1}, z_{t-2}, ..., z_1; A)P(z_{t-1}|z_{t-2}, ..., z_1; A)...P(z_1|z_0; A) \\
&= P(z_t|z_{t-1}; A)P(z_{t-1}|z_{t-2}; A)...P(z_2|z_1; A)P(z_1|z_0; A)
\end{aligned}
$$

$$
\begin{aligned}
&= \prod_{t=1}^{T} P(z_t|z_{t-1}; A) \\
&= \prod_{t=1}^{T} A_{z_{t-1}\, z_t}
\end{aligned}
$$

Compute this:

$$P(z_1 = s_{sun}, z_2 = s_{cloud}, z_3 = s_{rain}, z_4 = s_{rain}, z_5 = s_{cloud})$$

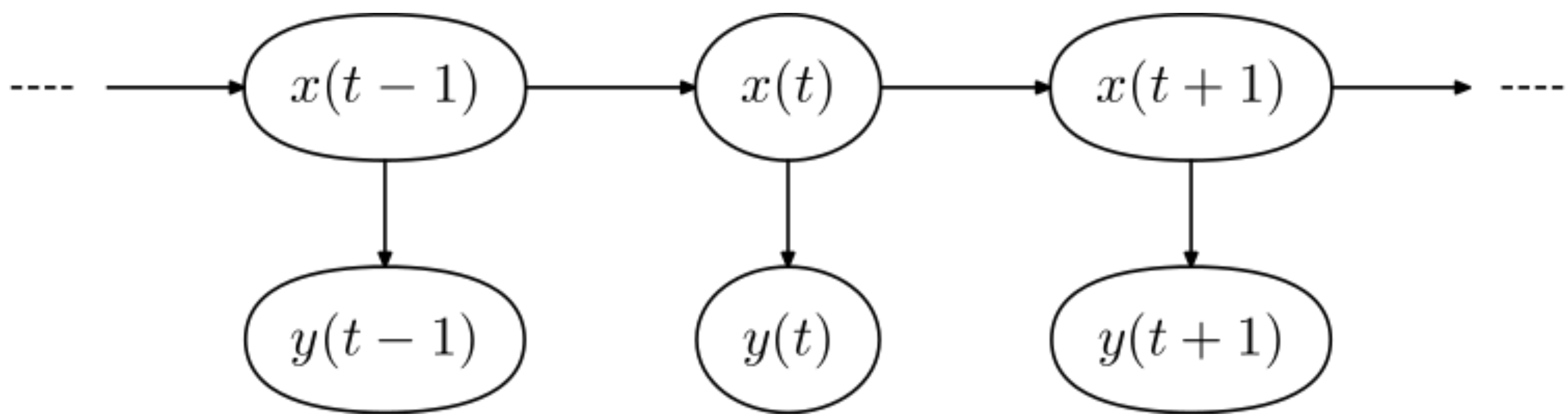
$$A = \begin{array}{c|cccc} & s_0 & s_{sun} & s_{cloud} & s_{rain} \\ \hline s_0 & 0 & .33 & .33 & .33 \\ s_{sun} & 0 & .8 & .1 & .1 \\ s_{cloud} & 0 & .2 & .6 & .2 \\ s_{rain} & 0 & .1 & .2 & .7 \end{array}$$

If we had a sequence **z**, we could use maximum likelihood to figure out **A**

Examples?

Hidden Markov Models

We don't observe the sequence directly



e.g. words vs audio waves

New matrix **B** that also tells us P(y_t = i | x_t = j) use observe sequence **z**

$$
\begin{aligned}
P(\vec{x}; A, B) &= \sum_{\vec{z}} P(\vec{x}, \vec{z}; A, B) \\
&= \sum_{\vec{z}} P(\vec{x}|\vec{z}; A, B) P(\vec{z}; A, B)
\end{aligned}
$$

use HMM assumptions

$$
\begin{aligned}
P(\vec{x}; A, B) &= \sum_{\vec{z}} P(\vec{x}|\vec{z}; A, B) P(\vec{z}; A, B) \\
&= \sum_{\vec{z}} \left(\prod_{t=1}^{T} P(x_t|z_t; B)\right) \left(\prod_{t=1}^{T} P(z_t|z_{t-1}; A)\right) \\
&= \sum_{\vec{z}} \left(\prod_{t=1}^{T} B_{z_t\, x_t}\right) \left(\prod_{t=1}^{T} A_{z_{t-1}\, z_t}\right)
\end{aligned}
$$

This is relatively advanced. Even sklearn has outsourced it:

https://github.com/hmmlearn/hmmlearn