香PYCON港 HONG KONG

AxonLabs

Building an Adaptive Learning System using Bayesian Modelling in Python

Albert Au Yeung

Axon Labs Limited

2015-11-08

What is this talk about?

Adaptive learning is a method of education that makes use of **computers** to interactively presents teaching materials according to the **ability** or **needs** of the learner.

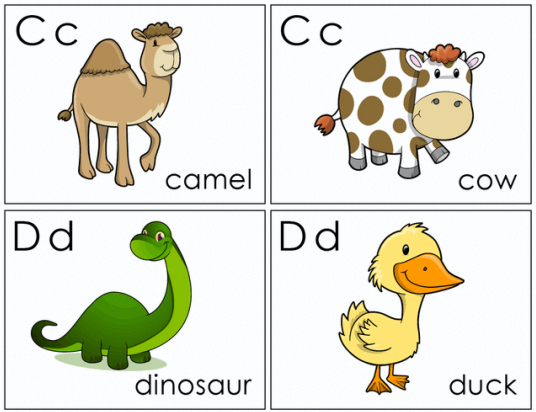
We will cover the following in this talk:

- Discuss an **example** of adaptive learning
- Use **Bayesian modelling** to model a student's ability
- Implement the system in **Python and PyMC**

Learning and Testing

Let’s consider learning vocabulary when studying a new language

A common approach is to **study** and then take a **quiz**



Take a quiz

Go back and revise

A. Camel
B. Cow
C. Duck
D. Dinosaur

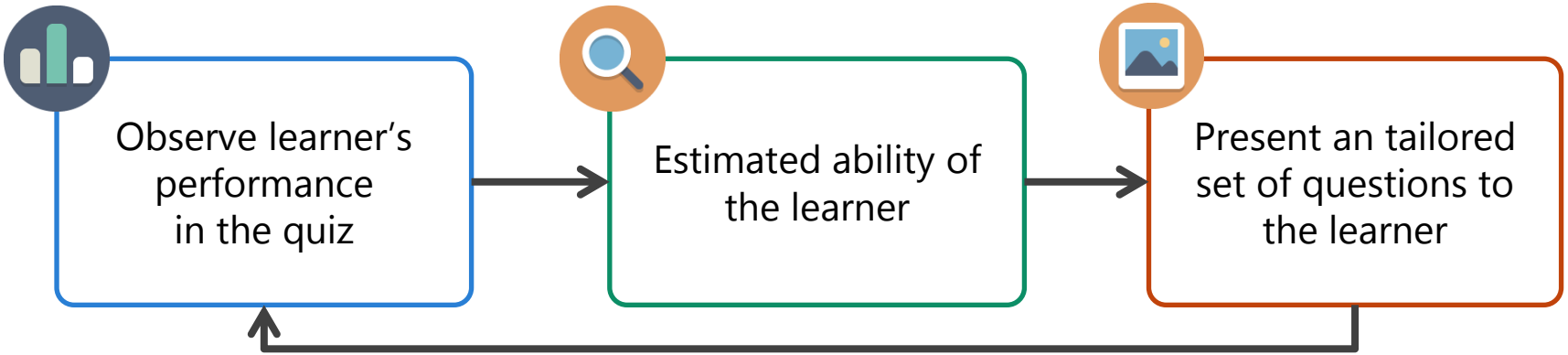
Problems

A straight-forward approach is to present questions randomly in the quiz, but:

- Some questions may be **more difficult** than the others
- It may not be necessary for the learner to revisit questions that he can **answer correctly with confidence**
- The **choices** presented may affect the student's confidence in answering the question

Adaptive Learning

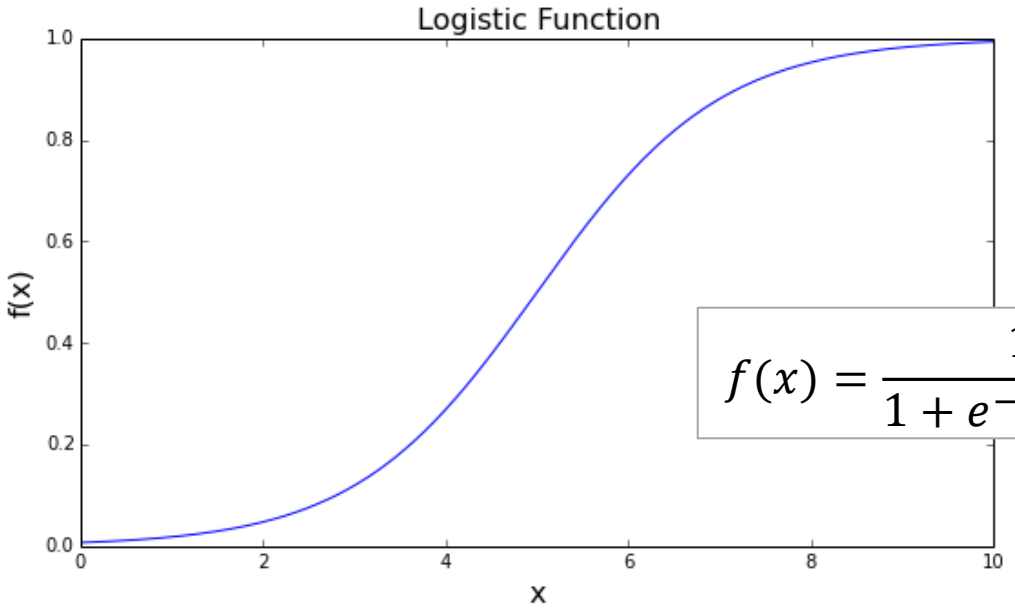
A better way is to **adapt** the questions according to the current **(estimated) ability** of the learner



A Model of Ability and Performance

A way to model a learner is to use the logistic function:

Probability that the learner will give a correct answer

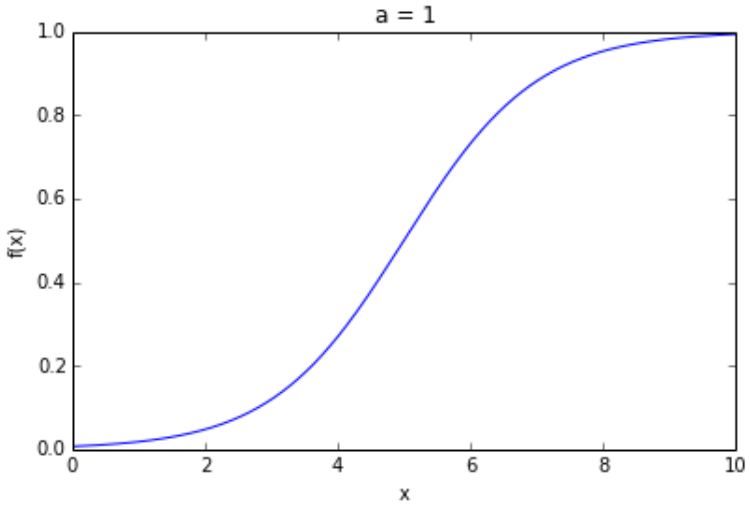


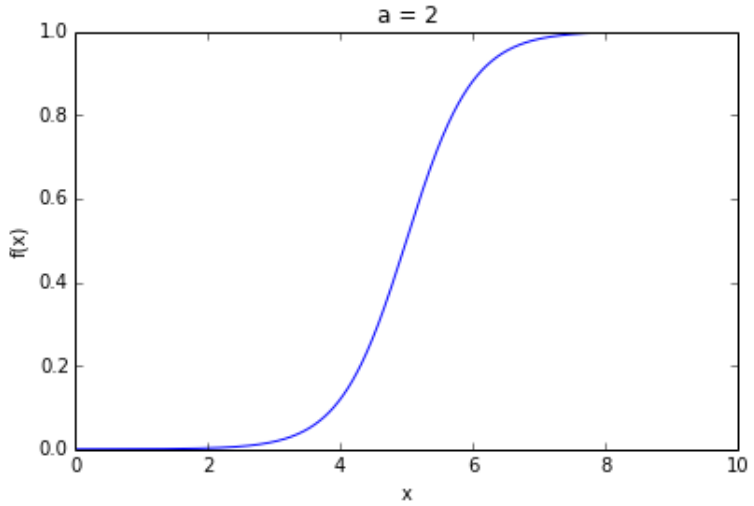
$$f(x) = \frac{1}{1 + e^{-a(x-x_0)}}$$

Estimated ability of the learner

A Model of Ability and Performance

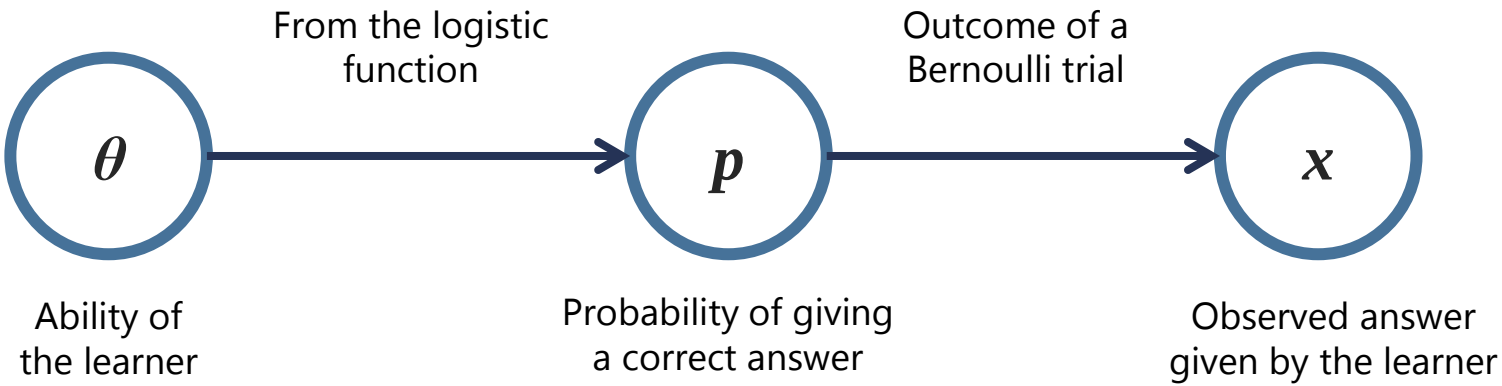
"Difficulty" of the task can be modelled by the parameter a and x_0





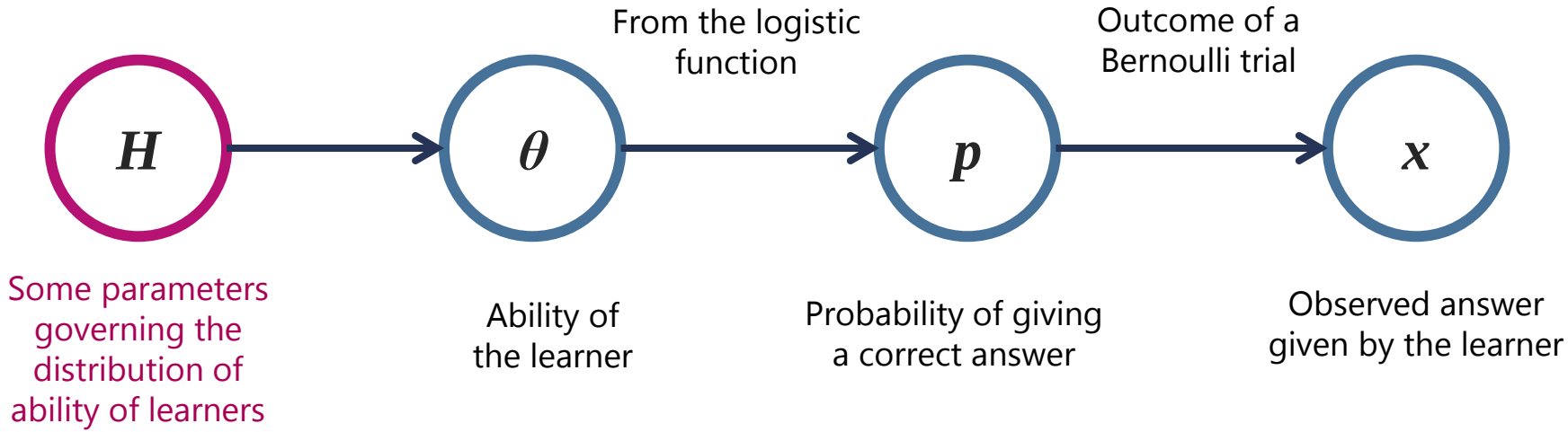
Modelling the Process of Answering a Question

One step further is to mathematically describe how the learner generates the answer to a question

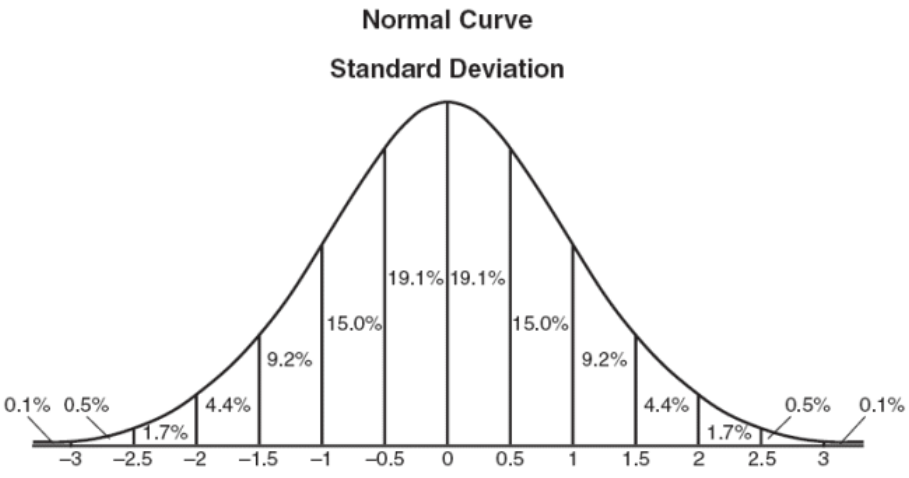


Modelling the Process of Answering a Question

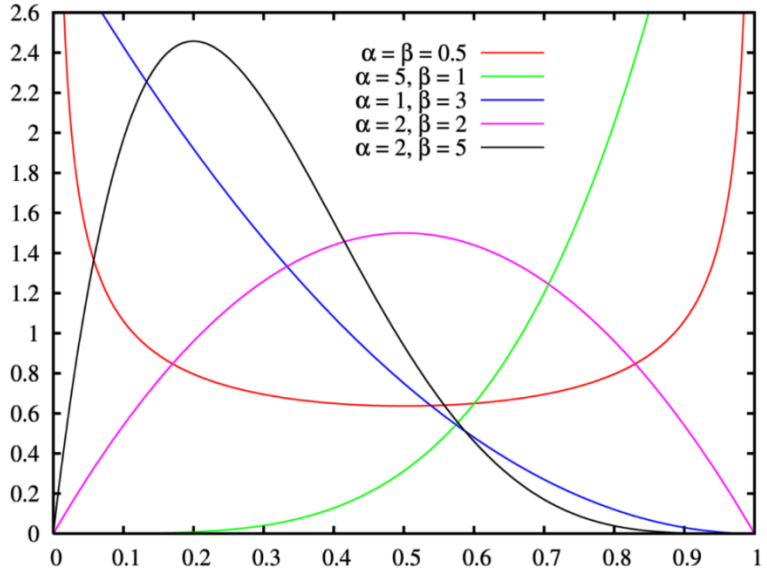
To model our "**prior**" belief of the ability of a general learner, we can add one more step at the beginning



Examples of Prior Distribution



Normal Distribution

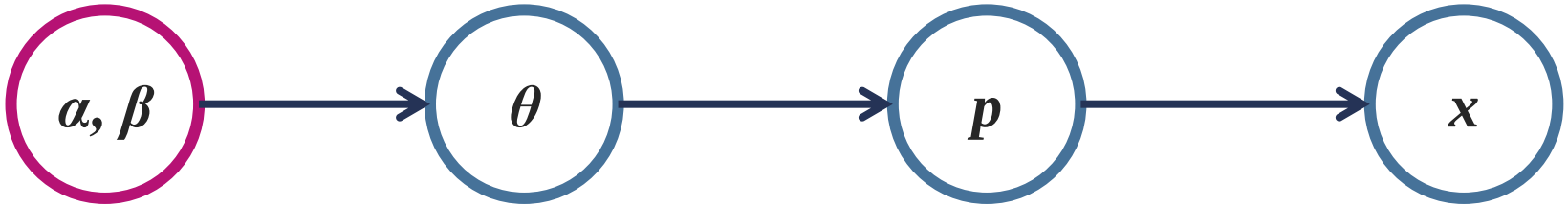


Beta Distribution

Enters Bayesian Modelling!

Bayesian Modelling

Learner's answers observed are obtained from a generative process



Parameters for our prior distribution (beta distribution)

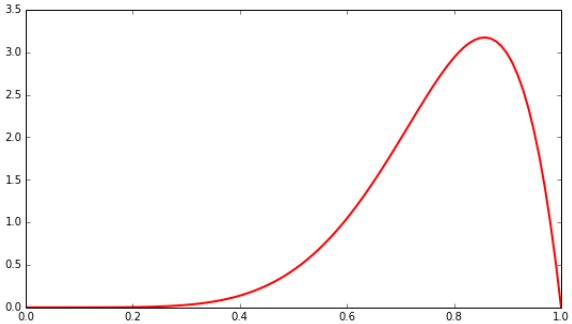
Ability of the learner

Probability of giving a correct answer

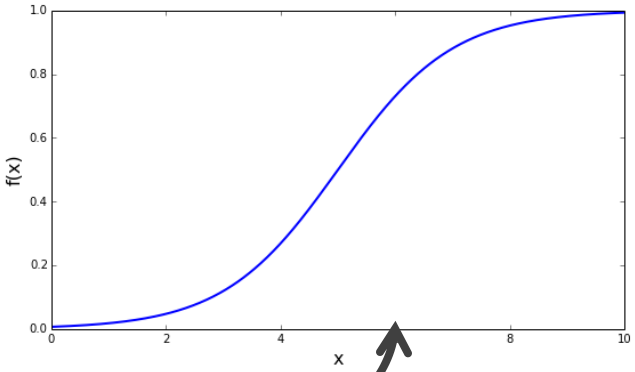
Observed answer given by the learner

Bayesian Modelling

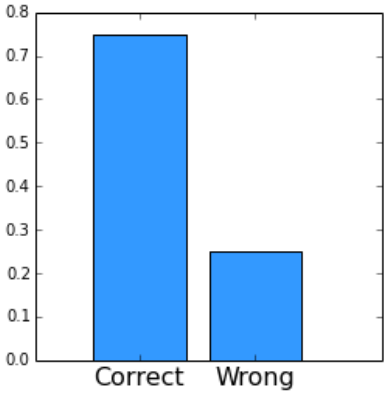
Prior distribution of a learner's ability



0.6



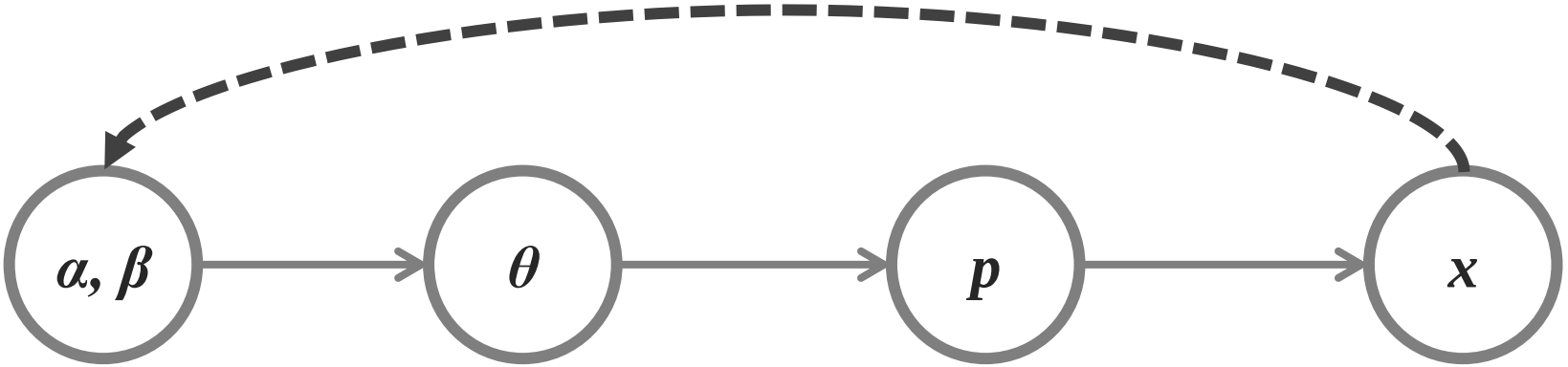
0.75



Answer: Correct!

Bayesian Modelling

With this model, we can update our "belief" of the learner's ability when we observed the answers given by the learner



Parameters for our prior distribution (beta distribution)

Ability of the learner

Probability of giving a correct answer

Observed answer given by the learner

Wait. Do I need to do all the **math** here!?

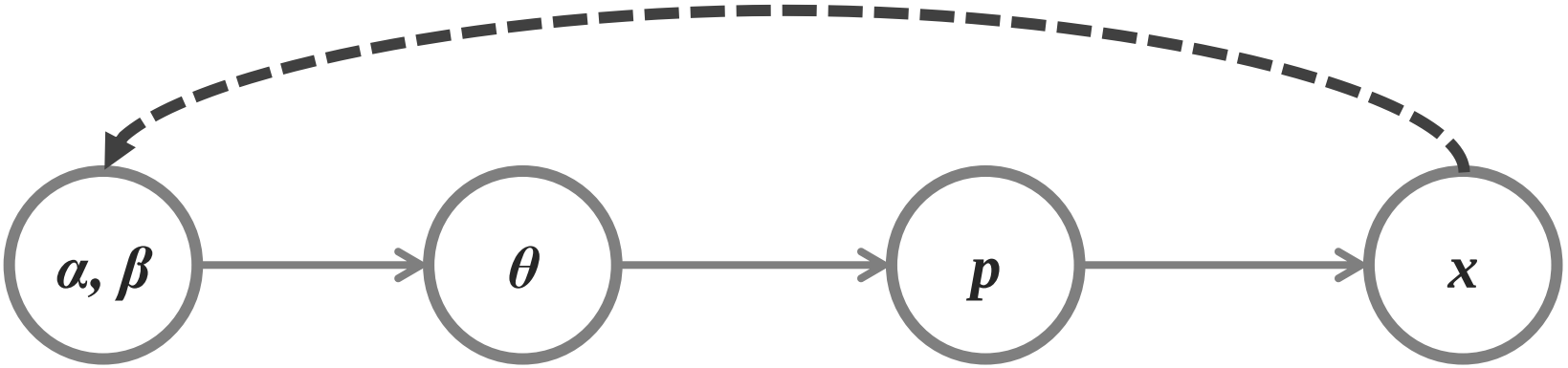
Enters Python and PyMC!

PyMC

- https://github.com/pymc-devs/pymc
- Open source Python module that implements Bayesian statistical models and fitting algorithms
- Version 2.3.6 now, Version 3 in Beta now
- Dependencies: Numpy & Scipy

Bayesian Modelling

In case you have already forgotten what we are modelling…



Parameters for our prior distribution (beta distribution)

Ability of the learner

Probability of giving a correct answer

Observed answer given by the learner

PyMC

AxonLabs

Let's first generate one batch of artificial data (observations):

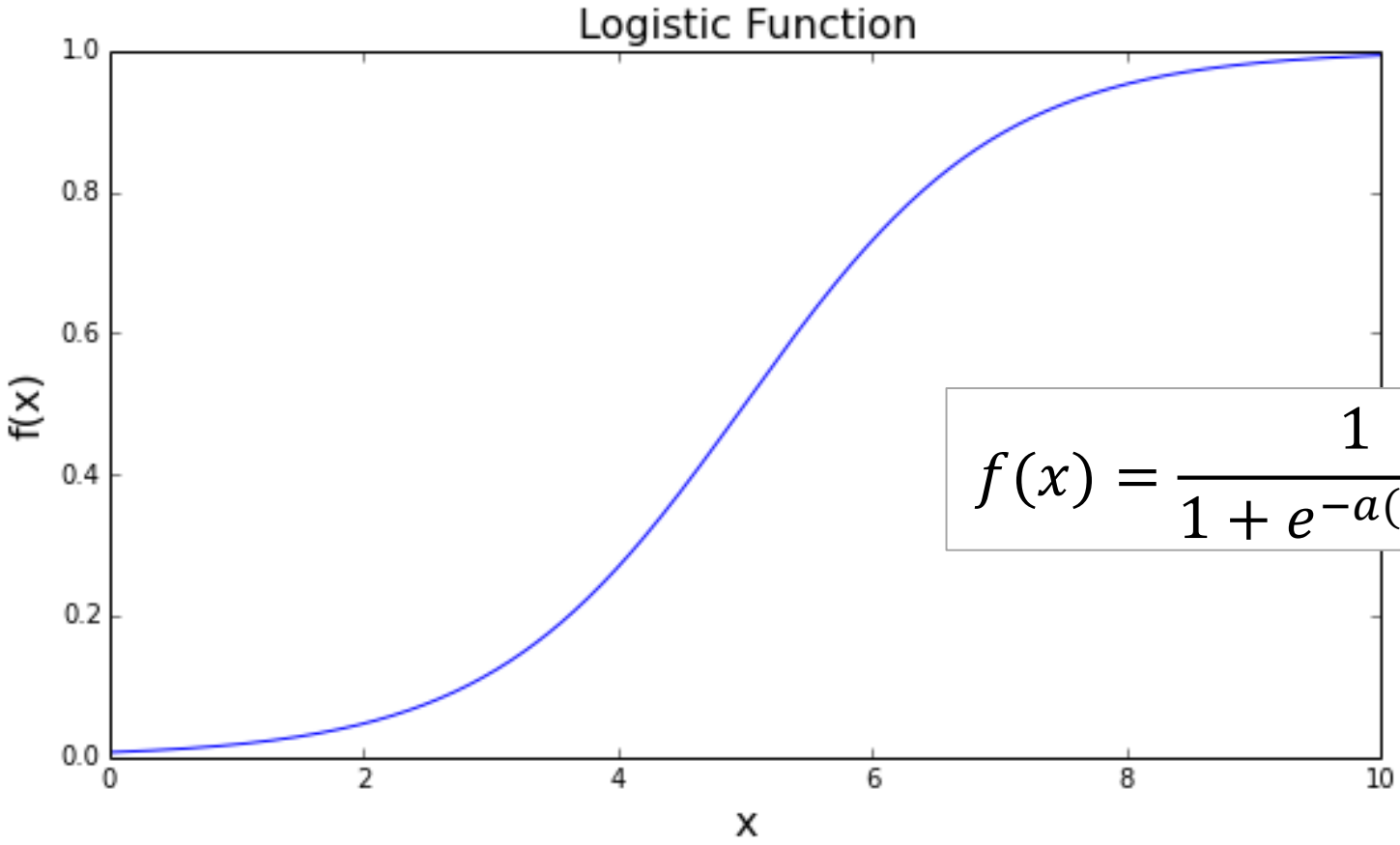
```
import random

# Assume the learner gives a correct answer 70% of the time
# =1 if correct, =0 if wrong
data = []
for i in xrange(50):
    x = random.random()
    if (x > 0.3):
        data.append(1)
    else:
        data.append(0)
```

PyMC

AxonLabs

Let's build our generative model (1)

```
import pymc

# Initial value of the "hyper-parameters" alpha and beta
a, b = 1, 1

# Step 1:
# theta is a random variable drawn from a beta distribution
theta = pymc.Beta('beta', a, b)

# Step 2:
# the probability of giving a correct answer, p, is obtained
# using the logistic function given theta as the input
@pymc.deterministic
def p(theta=theta):
    return 1.0 / (1 + math.exp(-1 * (theta * 10 - 5)))
```



$$f(x) = \frac{1}{1 + e^{-a(x-x_0)}}$$

PyMC

AxonLabs

Let's build our generative model (2)

```
# Step 3:
# Let x be a random variable representing the observed data
# it is drawn from a Bernoulli distribution (yes or no)
# with parameter p, i.e. P(x=1) = p
# 'data' contains observations
x = pymc.Bernoulli('x', p, value=data, observed=True)

# Step 4:
# Define a pymc model with all the components
model = pymc.Model([theta, p, x])
```

PyMC

Fit the model and get an estimated value of the parameters given the observation

```
model = pymc.Model([theta, p, x])

# Fit the model using the observed data,
# and obtain the MAP – maximum a posterior probabilities
m = pymc.MAP(model)
m.fit()

print m.get_node('theta').value # array(0.6059709)
print m.get_node('p').value # array(0.74000046)
```

The values can be different if you fit the model again

PyMC - MCMC

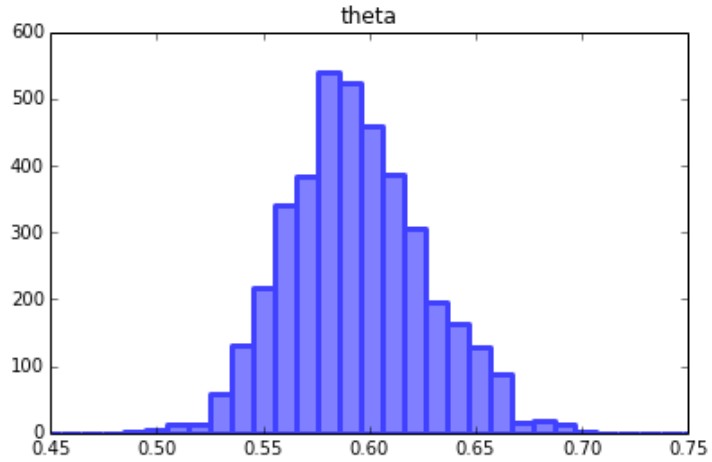
AxonLabs

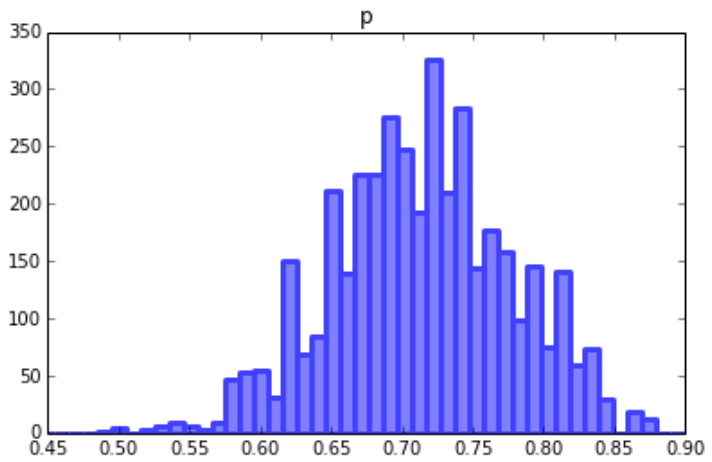
Another approach to obtain more information is to run MCMC (a kind of Monte Carlo simulation to estimate the posterior probability

```
model = pymc.Model([theta, p, x])

# Using MCMC to obtain many samples
# from the posterior probability distributions
mcmc = pymc.MCMC(model)
mcmc.sample(iter=5000, burn=1000)

# Here we can plot the trace of each variable
# and have an idea of their distribution
```

PyMC - MCMC

```
import matplotlib.pyplot as plt

plt.subplot(1, 2, 1)
plt.hist(theta.trace(), bins=np.linspace(0, 1, 100))
plt.title('theta')

plt.subplot(1, 2, 2)
plt.hist(p.trace(), bins=np.linspace(0, 1, 100))
plt.title('p')
```





Bayesian Modelling

Why don’t we simply compute the accuracy (~70%) of the learner?

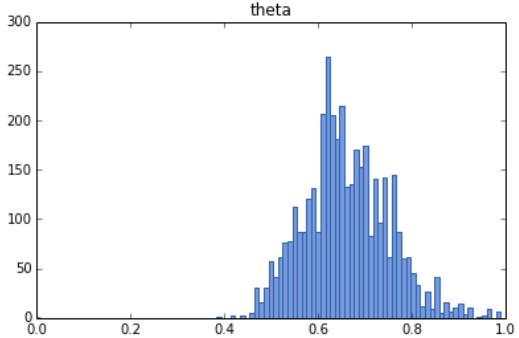
- Because we are doing Bayesian!
- With that we can estimate p, but we cannot obtain any information about (our belief of) the user’s ability
- We cannot easily say how confident we are regarding our estimation

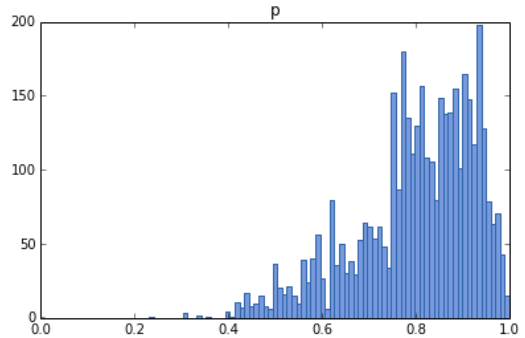
Amount of Data

The more data we have, the more confident will be our belief

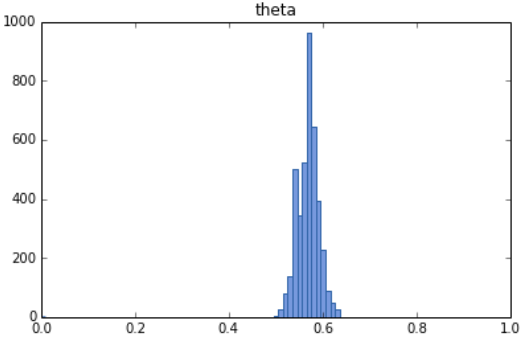
(This affects your strategy to presents new questions to the learner)

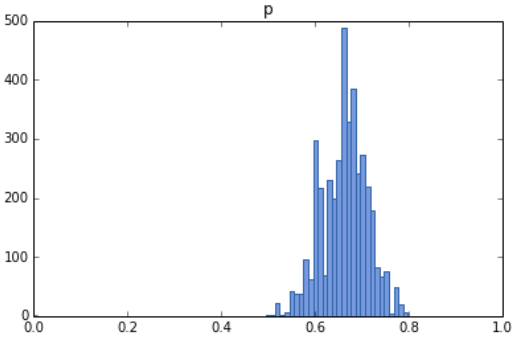
10 Data points



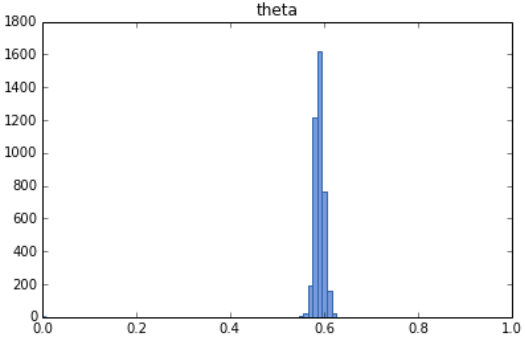


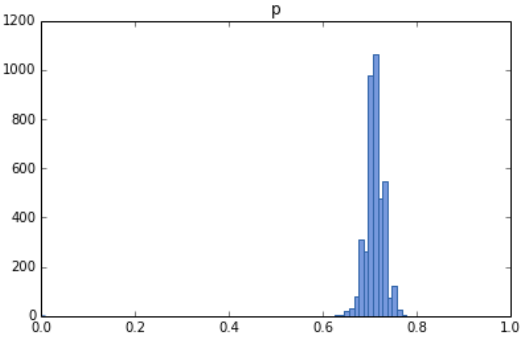
100 Data points





500 Data points





Adaptive Learning in Action

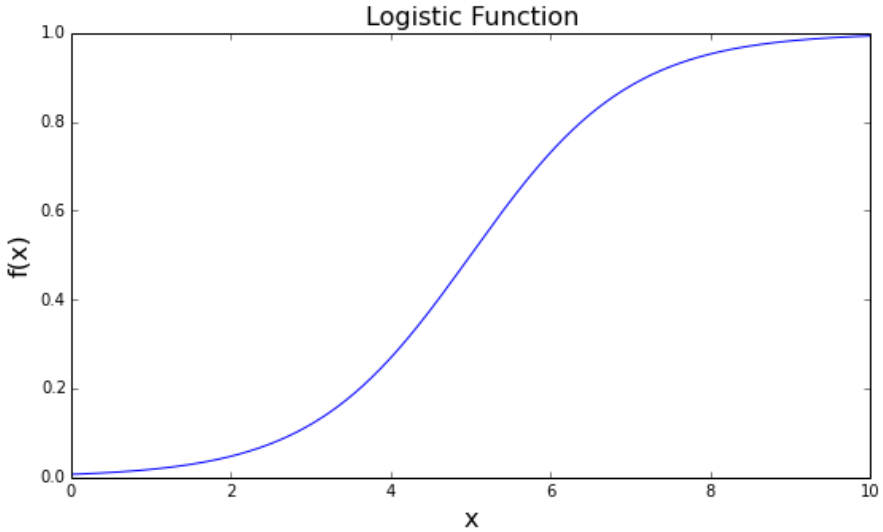
To test and demonstrate adaptive learning, we developed a simple app for learning Japanese Katakana characters: “Learn Katakana - 學習片假名”

n	w–	r–	y–	m–	h–	n–	t–	s–	k–	
ン N	ワ WA	ラ RA	ヤ YA	マ MA	ハ HA	ナ NA	タ TA	サ SA	カ KA	ア A	–a
	ヰ WI	リ RI		ミ MI	ヒ HI	ニ NI	チ CHI	シ SHI	キ KI	イ I	–i
		ル RU	ユ YU	ム MU	フ FU	ヌ NU	ツ TSU	ス SU	ク KU	ウ U	–u
	ヱ WE	レ RE		メ ME	ヘ HE	ネ NE	テ TE	セ SE	ケ KE	エ E	–e
	ヲ WO	ロ RO	ヨ YO	モ MO	ホ HO	ノ NO	ト TO	ソ SO	コ KO	オ O	–o

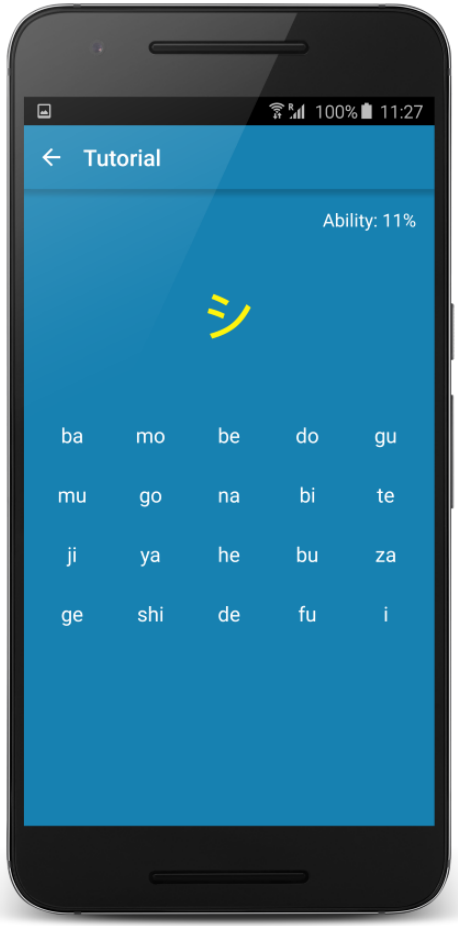
Modelling the Problem

We model the learner's ability to distinguish between two Katakana characters

モ vs. ミ
ヒ vs. ピ
ホ vs. ア
…



Taking the Quiz

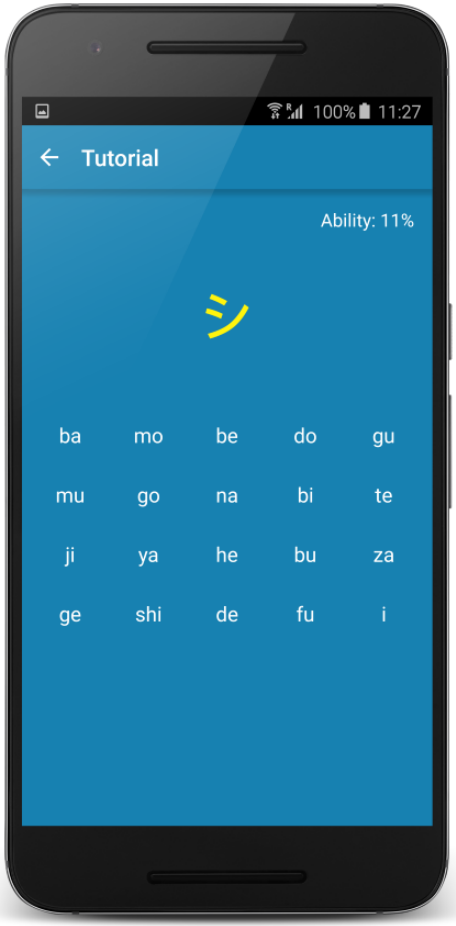


AxonLabs

Adaptive Learning in Action

After each round, we update our estimation of the learner's abilities

In the next round, we presents more difficult characters and more difficult choices with higher probability.

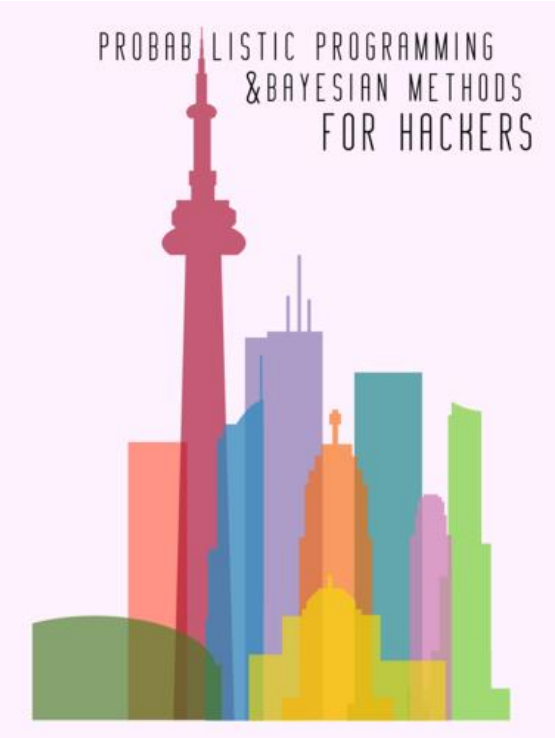


The choices are chosen based on our belief of the learner's abilities

Reference

Highly recommended tutorial on Bayesian Modelling and PyMC:

Probabilistic Programming and Bayesian Methods for Hackers
https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers



Thank you & Happy Learning!

Slides and examples available at:
https://github.com/albertauyeung/pyconhk2015-adaptive-learning

albertauyeung@axon-labs.com
http://www.axon-labs.com/