Machine Learning and You

the art and science of building
smarter web applications

Who this talk is for

- well, web developers, clearly
- people with at least an intermediate knowledge of Python, its data structures and the standard library
- Data crunchers
- the occasional math PhD who understands "how" but does not understand "why"

If you're on your laptop

- Install the following packages:
  - libyaml (on Ubuntu you need libyaml and libyaml-dev)
- Download the code:
  - git clone https://www.github.com/thatdatabaseguy/ml4webdevs.git
  - cd ml4webdevs/setup
  - ./bootstrap.sh

Why you should care

- Ever wanted to give your users meaningful recommendations?
- How about finding similar documents?
- or detecting spam/fraudulent transactions?

<digression>

What machine learning is NOT

- re.compile('<\s*h1[^>]*>(.*?)<\s*/\s*h1>')
- ETL / data cleansing
- twitter_id2facebook_id OMG I <3 #mashup!
- Big Data or using a NoSQL data store
- Full-text search
- Reporting and/or data visualization

This IS rocket science (sort of)

- We will use convenient abstractions and vanilla algorithms for today's class
- BUT, more advanced machine learning algorithms require a strong understanding of linear algebra

Some bubble-bursting

- The following does not exist:

```
import machine_learning
learn(my_data)
predict(my_new_data)
```

- And it NEVER will, no matter what any library or company advertises
- This stuff cannot be done through an API

</digression>

...still here?


YOU MAKE
KITTY SCARED

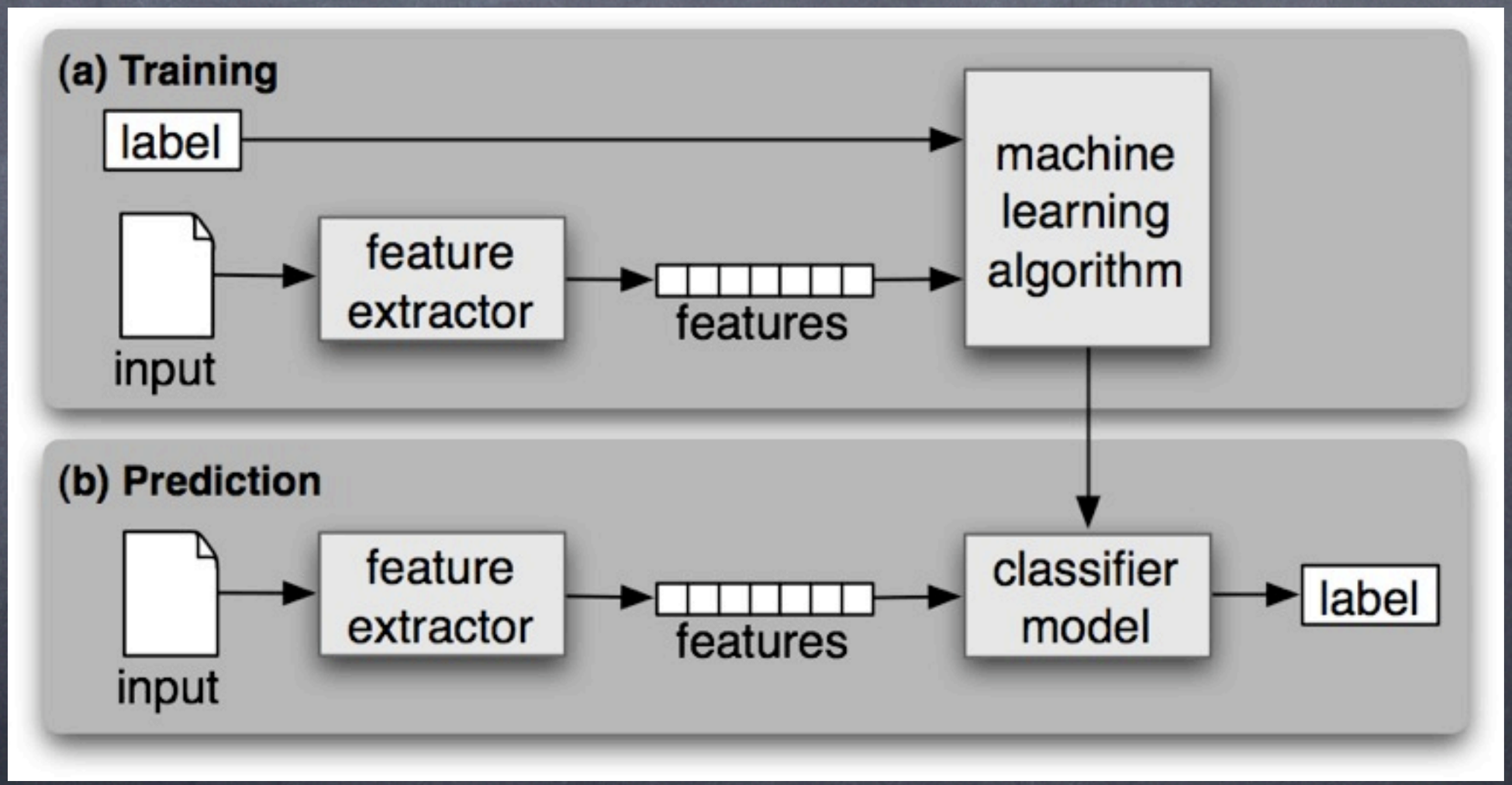
What does exist

- Lots of cool open-source libraries
  - nltk (highly recommended for NLP as well)
  - scikits.learn (fast, uses scipy)
  - pybrain
  - pattern
  - pyml

Supervised Learning

- Have some known, correctly labeled data
- Extract some **features** from the data, forming a set of **training tuples**
- Use an algorithm (there are several) to learn from the data and produce a **model**
- The model can be used in the **prediction** of labels from unseen instances

Or a picture


(a) Training
label
machine learning algorithm
input
feature extractor
features
(b) Prediction
input
feature extractor
features
classifier model
label

Feature extraction

- This is the most important part you need to think about in supervised learning models
- denormalize object graph or relations into a tuple of binary attributes or real numbers
- single features can be complex (contains synonym for "jelly" after synonym for "peanut butter")
- stored in a dictionary or a vector/matrix

Regression

- Predicted value is **continuous**
- Very applicable in the financial sector (e.g. predicting risk of an asset)
- ALSO very applicable to systems administration (how many users should I expect on-site? Given the load on my databases, how much latency should my users see?)

Classification

- Predicted value is **discrete**
- Discrete value can mean:
  - Binary values (e.g. spam/not spam)
  - Most likely category
  - Probability distribution over categories

Classifying political tweets

- Two top political hashtags on Twitter:
  - #p2 (Progressives 2.0)
  - #tcot (Top Conservatives on Twitter)
- Train model with tweets for these hashtags
- Classify new tweets as leaning DEM or GOP

What about A ∩ B?

- Indeed, intersection is ambiguous
- Use set complements (A-B and B-A)
- Lose some data but labels are more certain
- Luckily Twitter allows Boolean searches:
  - #tcot -#p2
  - #p2 -#tcot


Stand back!
MILK
Iz goin to do science!
ICANHASCHEEZBURGER.COM

Old Man Bayes' Law

- Promised to stay light on the notation, but:

$$P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)}$$

- P(C | A) is read "probability of C given A"

A Naive Bayes Classifier

- Hello World of supervised learning
- Single classification (maximum likelihood)

$$\text{classify}(f_1, \ldots, f_n) = \operatorname*{argmax}_c \, p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).$$

- "Naive" - assumes features are independent
- Very efficient to train (multiply probabilities)

Demo time

"Hold onnn to your butts..."

Some random insights

- Democrats can read, apparently 6 : 1
- GOP: we ARE America!

Making it all work

- This is fine in the shell, but I promised to talk about web applications
- Two main problems:
  - Need to save the model somewhere
  - Need to load it in the request cycle to make predictions

Saving the model

- Two methods that worked for me:
  - Stuff it in the cache (for small models) and keep a copy on disk
  - For larger models, just put it on disk on the datamine box and write a simple RESTful service for classification

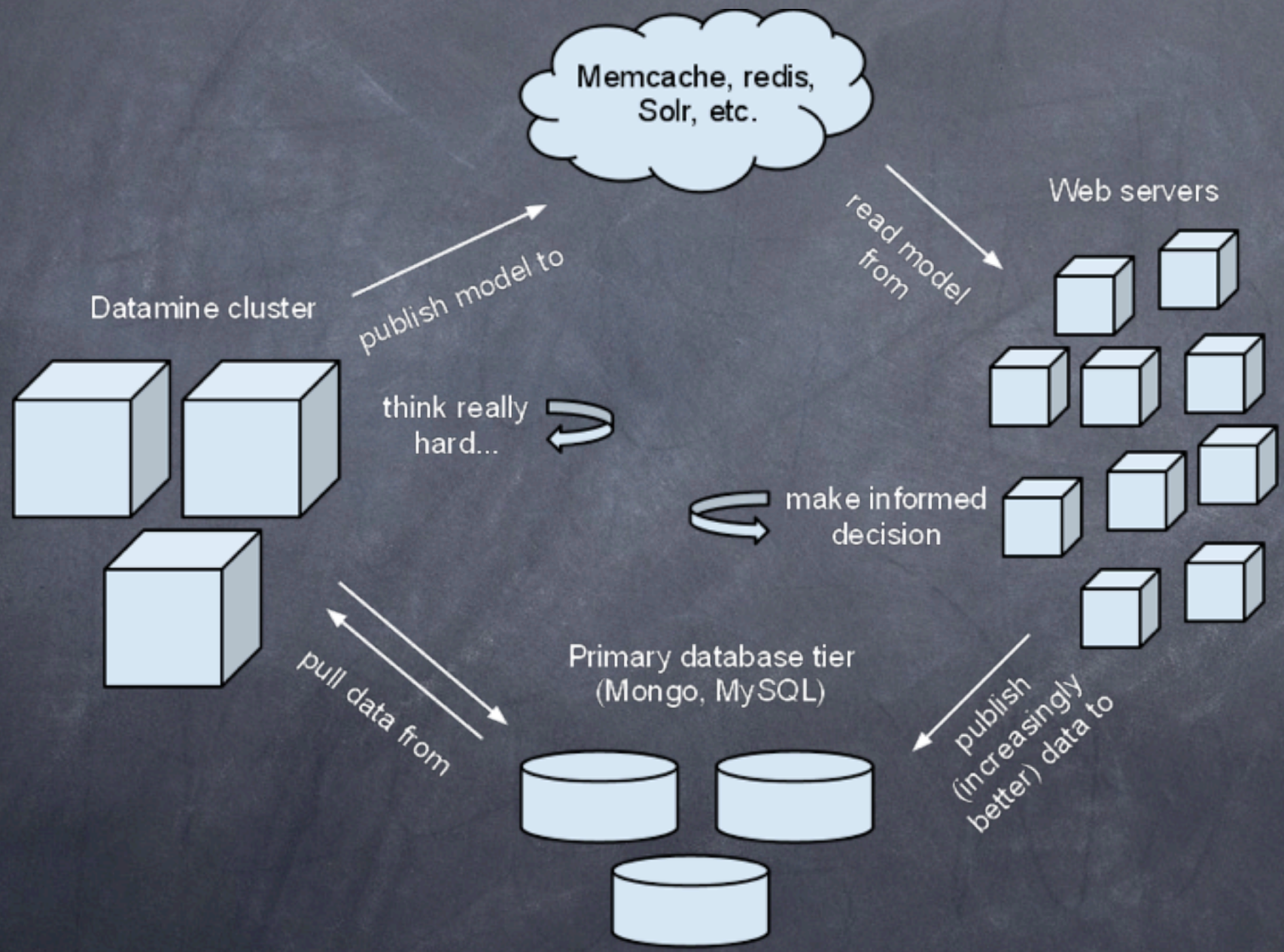
Decoupling

- Probably don't want app servers running nltk or scipy
- Feature extraction code can live on both (sometimes)
- Some models easier to represent than others
  - Decision trees are easy to represent as simple nested dictionaries

Using the model in a request

- If it's a simple model, read it from the cache
- Otherwise, run a RESTful service on your datamine machines
- Try to return ids for app domain data
- Memory-mapped files are nice for larger models

An example architecture



Advanced Topics (if there's time)

A brief look at Unsupervised Learning

- Used for clustering data
- Don't know any right answers beforehand
- Good for similarity (Latent Semantic Indexing or Latent Dirichlet Allocation)
- Difficult to evaluate accuracy

Unsupervised Learning Models

- Usually involves choosing a number of topics or clusters
- Don't need to compare every instance to every other instance, just every instance to every cluster

To check out

- K-means clustering
- Latent Semantic Indexing
- Latent Dirichlet Allocation

scipy

- nltk is great, but not that scalable
- Really need scipy for large-scale problems
- Instead of a dict, we could have built a **term-document matrix** where the rows are documents and the columns are features with numerical ids. Need a mapping.
- Blindingly fast, everything's in C or Fortran

scipy (cont'd)

- sparse matrices (mostly 0's)
- data = numpy.array([10,20,30]); row=numpy.array([0,1,2]); col = numpy.array([5, 8, 9])
- Only store the non-zeroes and their positions
  - matrix = csr_matrix((data, (row, col)), shape=(3,10), dtype=numpy.int32)

scipy (cont'd)

- matrix.todense()

```
matrix([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0, 10,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  0, 20,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 30]],
dtype=int32)
```

scipy (cont'd)

- Very efficient for storing things like text documents as vectors
- Each column is a feature (word, bigram, etc.)
- Need a dict of {feature: col_id}
- scikits.learn can use these sparse matrices instead of dictionaries, tends to scale better
- joblib releases the GIL, good for multicore