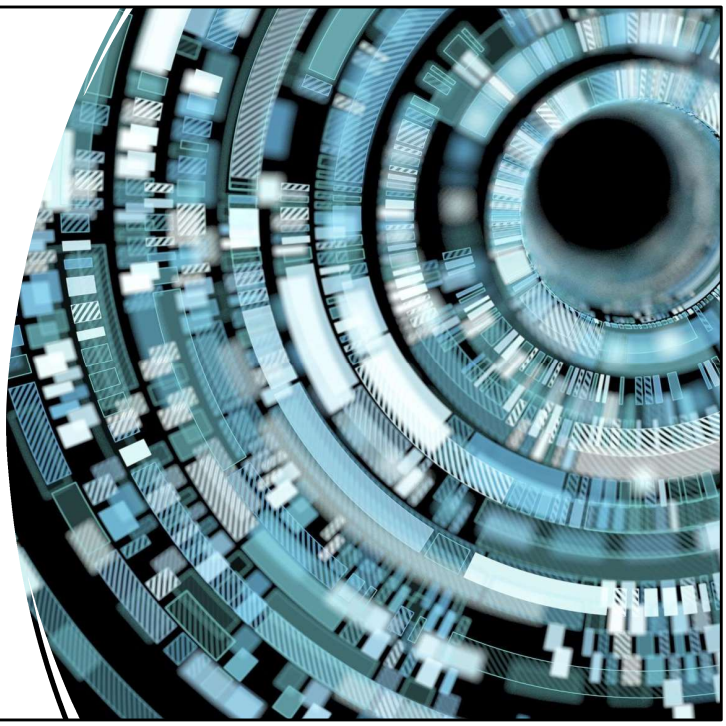
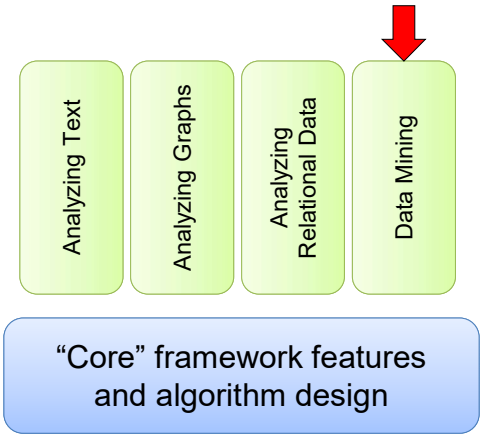
Data-Intensive Distributed Computing CS431/451/631/651

Module 6 – Data Mining / Machine Learning



Woah, trippy

Structure of the Course



What the, we skipped relational data?
(Yes, the assignments flow more easily this way…maybe I should change the graphic…)

How to Train Your Function

(with a small set of example input and expected output)

AKA

Supervised Machine Learning

Classification

Source: Wikipedia (Sorting)

Classification Functions

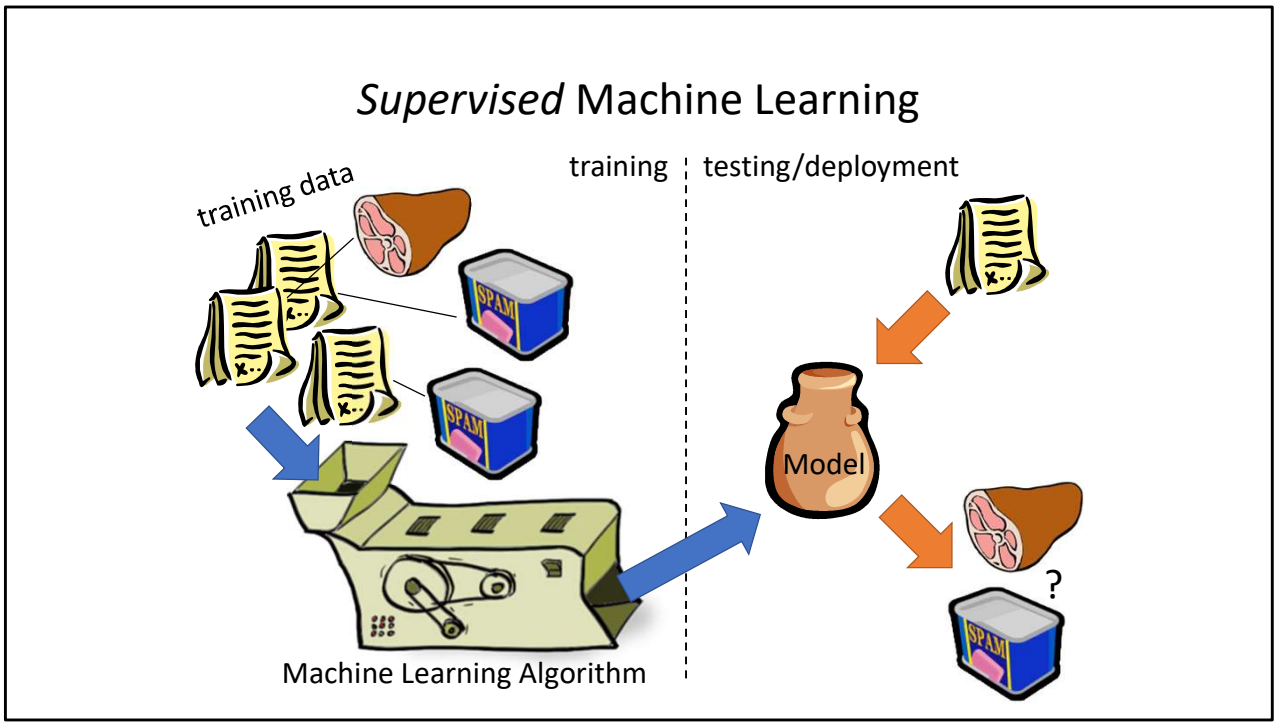
Input: Object
Output: Label(s)

Applications

- Spam Detection
- Sentiment Analysis
- Topic classification
- Link Prediction
- Document Ranking
- Object Recognition
- Fraud Detection

There's so many more, but I'm out of space before it shrinks the font on me

Label: a symbol, usually. So, a natural number. Maybe just 0 or 1



Supervised: You give is a training set, where emails have been classified as "SPAM" or "HAM" (meaning not spam…it's a bit of a joke you see)

Then, some sort of MATH happens, and out comes a model that can classify things.
It's called a model because it's sort of like the computer's "mental model" of the problem.
In this case, it's the "mental model" of what makes SPAM different than HAM (legitimate messages).

Object Representation

Q: When is an email not an email?

A: When it's a "feature vector"

Objects are represented as a vector of features:

- Dense Features: sender IP, timestamp, # of recipients, etc.
- Sparse Features: message contains "Viagra", subject contains "URGENT", etc.

In terms of numbers (and everything's a number, really)

Dense Feature: zeros are rare or non-existent.
Sparse Feature: non-zeros are rare

There tend to be a LOT of potential sparse features. How can the computer know about all of them??? There are a lot of words…

A rose by any other name…

A ***feature vector*** is also called an ***embedding***

Why? We've ***embedded*** a complex object (email) into lower dimensional space (for a vector of 16 features, $\mathbb{R}^{16}$)

What Features are Important For:

- Spam Detection
- Sentiment Analysis
- Content Classification
- IR Rankings
- Object Recognition
- Fraud Detection

For the same object, each application will have its own set of Features!

Spam: certain phrases, certain words, are very likely to be spam associated.
Sentiment Analysis: words that aren't important to spam/not spam are very important to inferring the writer's tone
Content classification: (Might overlap a bit with spam)

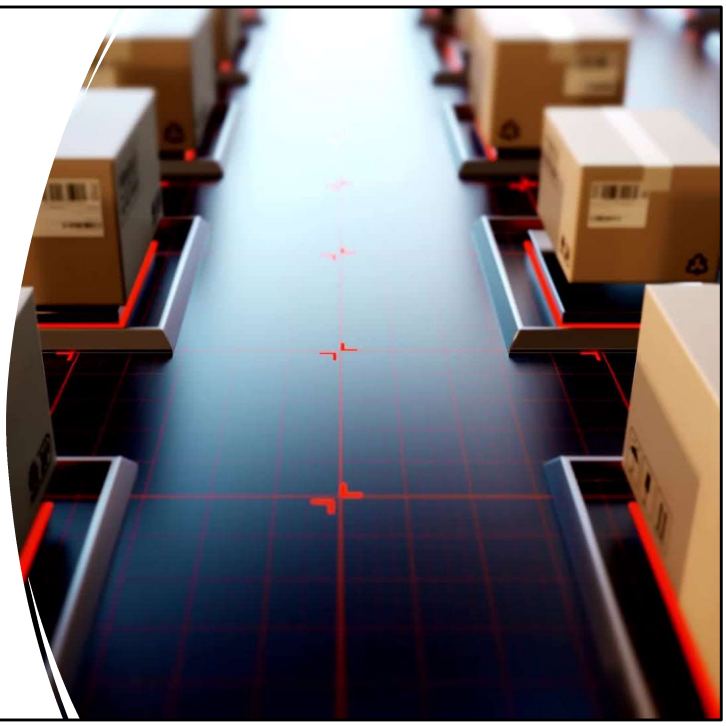
On Feature Selection

It's important that you select good features. This is a whole topic all on its own

Generally you want features that are independent. If two features strongly correlate you don't get much out of having both.

ML Solution Looks Like:

1. Acquire Data
2. Determine Features and Labels (by hand!)
3. Pick a Model
4. Train the Model
5. Repeat as needed

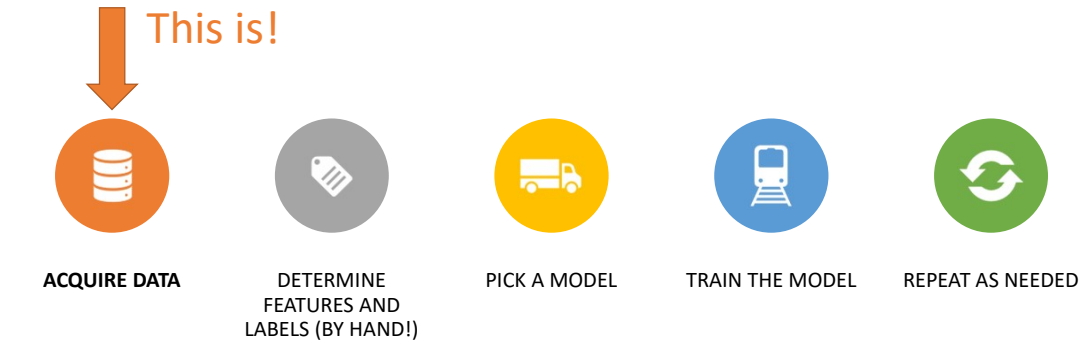


So Many Models

- Logistic regression
- Naïve Bayes
- SVM
- Random Forest
- Perceptrons
- Neural Networks
  - There are lots of kinds of these, too!

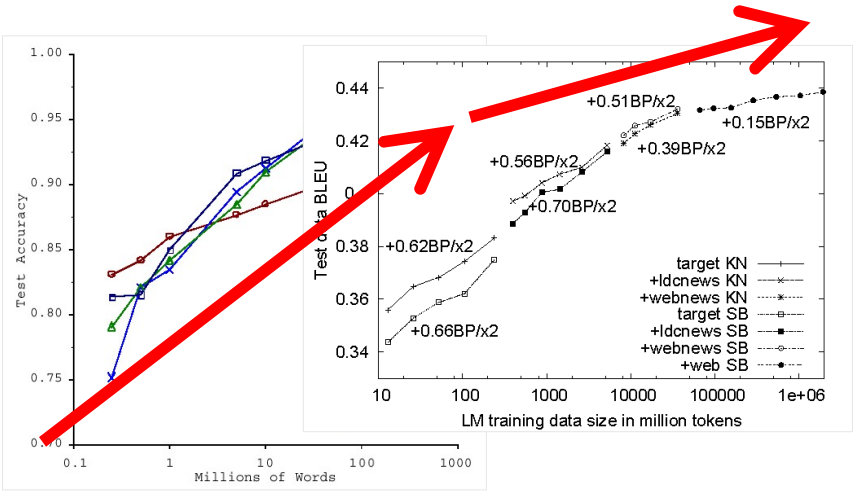
I like this picture as sorting through all of these ML models is a bit like getting lost in the woods…
Also Powerpoint probably suggested it because of “Random Forest”

What's Important?



In a Big Data course, did you expect anything to be more important than Big Data?
With enough data, even a bad model will perform fairly well. That's the theory anyway

No data like more data!



(Banko and Brill, ACL 2001)
(Brants et al., EMNLP 2007)

Take away – more data, more good.
Left grammar checker – It approaches 95% accuracy, which is why e.g. it lets “more data, more good” through.
Right machine translation

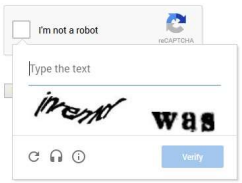
Wait, Hold Up, did you say BY HAND?

I did: “Determine … Labels (by hand!)”

Doesn’t seem like it will scale…

How to do things by hand at scale

Crowdsourcing



Bootstrapping (and other semi-supervised techniques)

Exploiting User Behaviour

(e.g. emojis are self-labelled sentiment analysis)

Crowdsourcing – Might be hiring people on Mechanical Turk, paying Google for surveys, etc.

Bootstrapping – Label a reasonable amount by hand, and train a high-precision model (precision = TP / (FP + TP))

Use model to generate pseudo-labels

Repeat on pseudo-labelled data, until pseudo-labels converge

Binary Classification

Label is a single binary value. Yes or no. Spam or Not Spam.

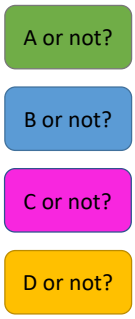
You can make more nuanced classifiers out of binary classifiers

Hotdog or not Hotdog

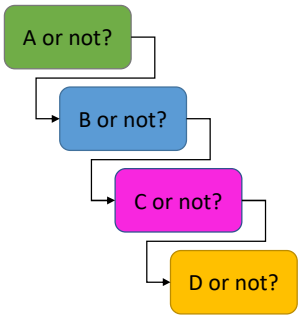
Binary Classifiers as Building Blocks

Example: four-way classification

One vs. rest classifiers



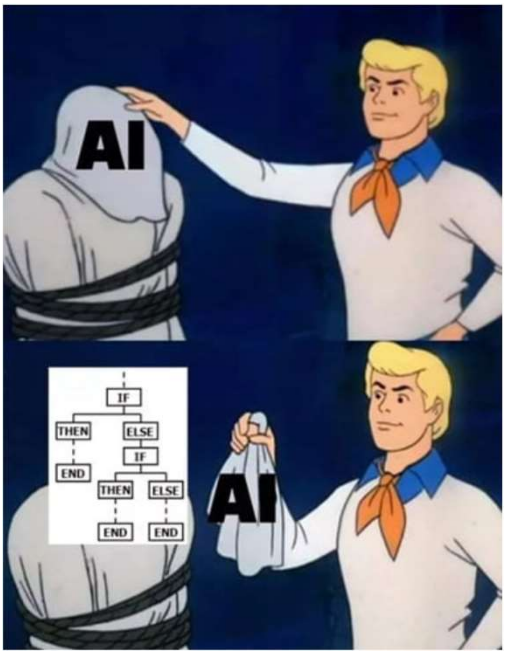
Classifier cascades



The above shows 4 way classification made from 4 “is or isn’t” classifiers.
“Spam or not spam”, “Notification or not notification”, “Personal or not personal”, etc.

You can also use binary classifiers to create a classifier tree, if the labels are not mutually exclusive.
Botany books have decision trees like that:
“Leaf is serrated or not” – “flower is single or cluster” etc. Those might seem more like features than labels, and you’re right! But if you’re working from a PICTURE, you have to use ML to find the “features” of the image, and then again to classify based on those features! Neat.

Dan intends to draw something on the board. Let’s see if he does.



Binary Classifier Cascades aren't the only sort of AI.
Generative Adversarial Networks, Deep Convolutional Networks, etc. don't really fit with the meme…so we'll ignore them.

Also, how did you pick WHICH binary classifiers to train? You have to do that before you can train the decision tree.

Training a Classifier

D = {(x_i, y_i)} D: Training Data, x_i: feature vector, y_i: label

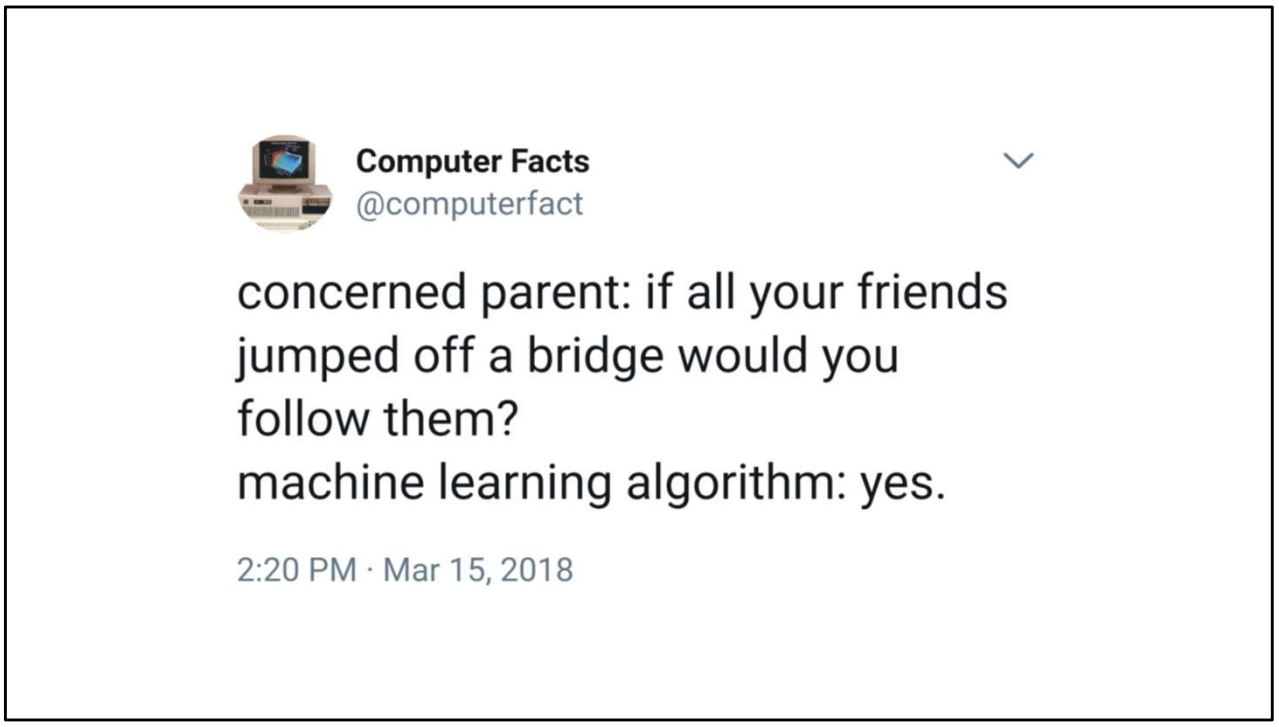
Want to find f: X → Y A function that maps feature vectors to labels

(But not any f, an f that minimizes "entropy" or "loss")

$$\frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

Where ℓ is a "loss" function

A loss function for a Boolean classifier might be as simple as ℓ(a,b) = (a xor b) i.e. 1 if they differ, 0 if they're equal. Of course you can apply different weights to false positives and false negatives if you want.



Labels – what a person says
Machine Learning – copy what people are saying

Everyone: AI art will make designers obsolete

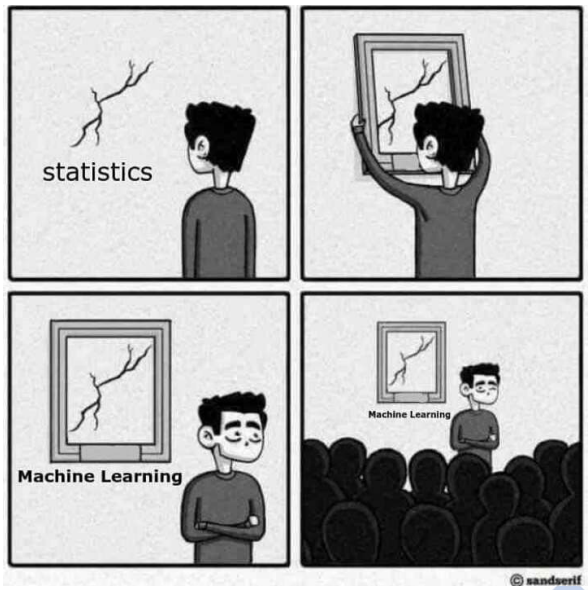
AI accepting the job:

Parameterization

There are LOTS of functions.

We'd rather have a parameterized function.

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i)$$

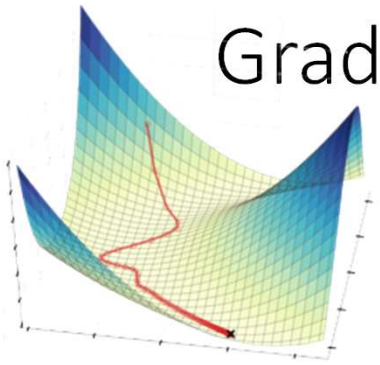


There are lots of jokes about “AI is just…” . Everything is “just” something else with a lot of messy details. Deal with it.
Besides, it’s mostly linear algebra, not statistics!

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i)$$

There probably isn't a closed form solution to this so…

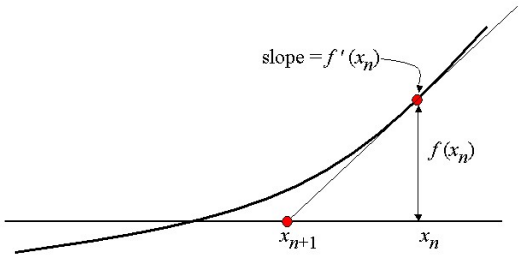
Gradient Descent



Intuition

Remember Newton's Iteration? Sure ya do!

1. Get the slope at point x_n
2. Compute the intercept for the tangent line at point x_n, call it x_{n+1}
3. Repeat

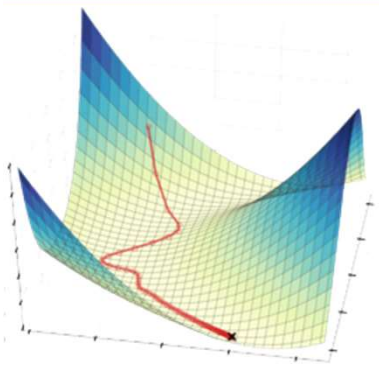


Intuition

Gradient Descent is the same thing

It just has many dimensions, not just x!

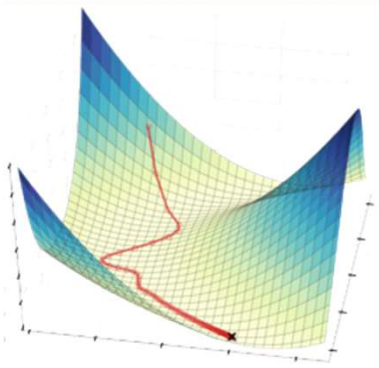
There's that "just" word again



Intuition

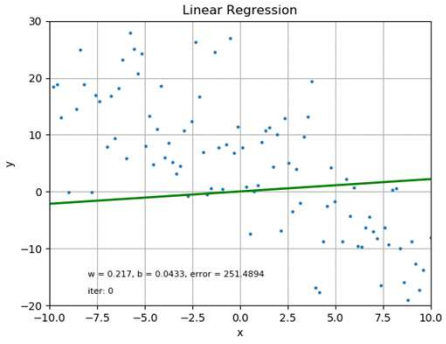
For each component x of θ:

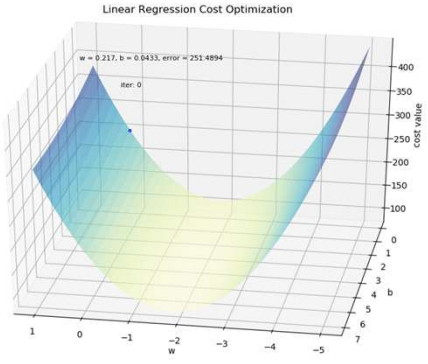
1. Compute $\frac{d}{dx}\ell$
2. Compute "downhill" (like Newton's Method with many dimensions)
3. Compute new θ



Ugh, differential equations. This is the one course I almost failed in undergrad…

Linear Regression + Gradient Descent





Gradient Descent

To find: $\text{argmin}_\theta L(\theta)$ $\quad L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i;\theta), y_i)$

Compute Gradient: $\nabla L(\theta) = \left[\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial wd}\right]$

So at any point θ…

$\theta' = \theta - \gamma \nabla L(\theta)$ Wait, gamma?

$L(\theta) \geq L(\theta')$ Yes, it's "step", or how far we move along the gradient

Replaced the complicated function with L(x) – since nothing else changes, we only really need one parameter.
The inequality is "for sufficiently small gamma".

Missing Details

What step should we use?

What about local minima?

How fast does this converge?

A1: How small is ‘sufficiently small’? – There are approaches to picking gamma. It’ll depend on if the function is convex or concave too.
A2: it will get trapped in them ☹
A3: not very quickly

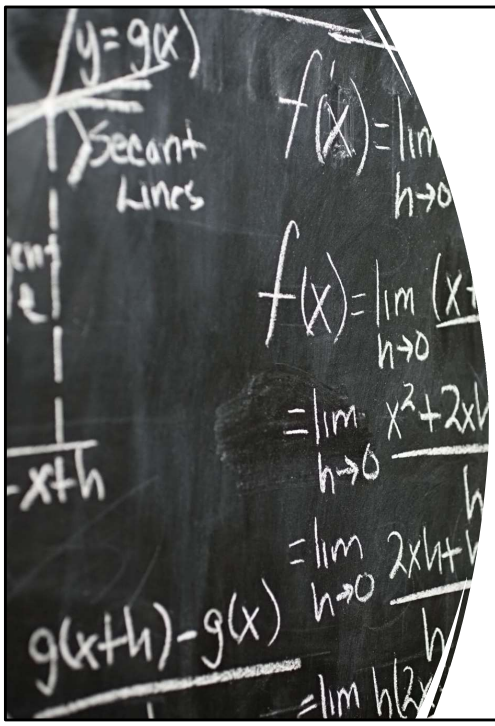
More Details

Gradient Descent is "first order"

- Local linear approximations
- Slow convergence

Fixes to avoid local minima

- Momentum
- Stochastic Gradient Descent
  - Randomly select a subset of training data
  - Gradient depends only on selected subset
- Can still get stuck



Other Approaches

Second order methods (Quasi-Newton)

- Quadratic approximation instead of linear
- Requires the Hessian (matrix of second order partial derivatives)
- Usually Impractical

I tried this for fitting database snippets into a protein with missing parts. It was tedious and then didn't work very well. In a bit we'll get to something somewhat similar to what I did! (Not really…but close enough for a detour)

Weren't we talking Boolean Classifiers?

Right, sorry.

$$f(X\,;W):\ \mathbb{R}^d \rightarrow \{0,1\}$$

Parameters to optimize:

- Weight vector w

Loss?

$$f(X\,;W):\ \mathbb{R}^d \to \{0,1\}$$
$$L = \frac{1}{2}(y-t)^2$$

Loss is 0 if prediction is right, 1 if prediction is wrong.

Problem: Not continuous, not differentiable.
Gradient Descent won't work.

Solution: **surrogate loss function**
Something that's not as accurate, but easier to optimize

My PhD work involved "squishing" protein snippets to make them fit into gaps, and the problem was something called "Multi-dimensional scaling". Loss was called "stress" in the papers I read. The algorithm I used was called "SMACOF – Scaling by Majorizing a COmplicated Function"
"Majorizing" is a sort of surrogate function…where you find a way to create a simpler function that:

1. Is equal to the complicated one at a given point
2. Must be <= the complicated function at all other points.

With SMACOF the majorizing function has a closed form minimum! No gradient descent needed, it goes straight to the minimum. Neat.
However, finding the minimum requires computing the pseudo-inverse of an n x n matrix, which is $O(n^3)$

Linear Function

$$f(X\,;W,b):\ \mathbb{R}^d \to \mathbb{R} = w \cdot x + b$$
$$L(y,t) = \frac{1}{2}(y - t)2$$

y: predicted value (any real number)

t: true value (0 or 1)

Problem: Cost is unbounded!

This seems like we've broken the classification function…but we haven't. Just set some cutoff point to map the values into "true" and "false"

Unbounded? If our classifier is "really confident" then it might return 100. If that's wrong, the loss will be immense!
We'll train the algorithm to never be confident. Might not be great.

Sigmoidal (Logistic) Function

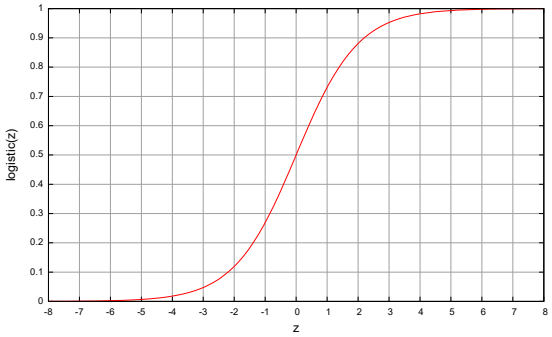
$$\sigma(z) = \frac{e^z}{e^z + 1}$$

- Sigmoidal functions are nice

$f(X; W, b) : \mathbb{R}^d \rightarrow [0,1] = \sigma(w \cdot x)$
L(y, t) = ½(y-t)^2

Prediction is between 0 and 1
Loss is between 0 and 1

Continuous and differentiable:
$\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z))$



Not just differentiable, but easy! Gotta love exponentials.

More Maths

$$\frac{\partial L}{\partial z} = (y - t)y(1 - y)$$

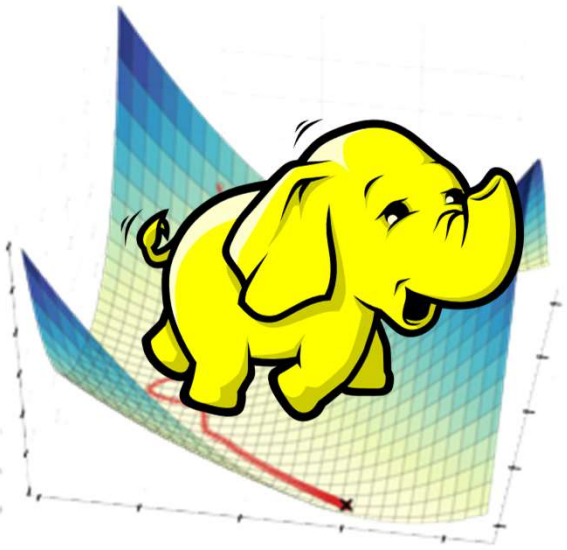
$$\frac{\partial L}{\partial w_i} = (y - t)y(1 - y)x_i$$

Now we can do gradient descent

To Learn More…

Take an actual ML course. This is just a tribute

- Lots of other optimization techniques
- Lots of different loss functions
- Sigmoidal has its own problems:
  - 0.00001 vs 0.0000001, way more confident in negative…same loss



Gradient Descent on Hadoop

MapReduce is bad at iteration

- Startup Costs
- Retain State between Iterations
- Stark Line – Iteration is bounded by the slowest worker

Gradient Descent: Must have only one reducer (bottleneck)

Spark to the Rescue? Again?

```
val points =
spark.textFile(...).map(…).cache()
var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x)))-1)*p.y
  }.reduce((a,b) => a+b)
  w -= gradient
}
```

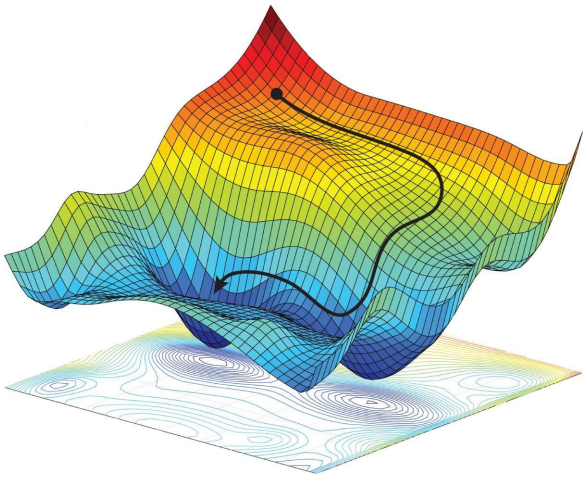
- Is that really better?

Yes

- But why?

1. Cache data
2. No Shuffles
3. Driver only does final reduce: 1 value per partition

Stochastic Gradient Descent



Source: Wikipedia (Water Slide)

Stochastic Gradient Descent

- Pick a random subset, only compute gradient for that subset of values
- If the subset is small enough for one machine…not Big Data anymore

In other words, no thanks. Unless…

If we have k mappers. Have each compute the gradient for its subset, then transform independently?

"mini-batching"

Ensemble Learning

(This was called “consensus modelling” when I was doing it for protein folding).

Ensemble Learning

1. Train multiple INDEPENDENT Models
2. Combine Predictions
   1. Voting
   2. Merging Models

   E.g: Partition test set, train classifiers independently

   Very parallel, perfect for a Hadoop cluster

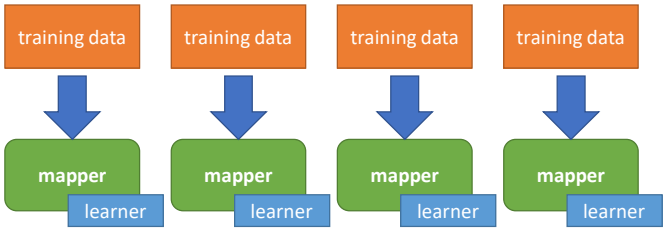
You can't always merge models. For a simple linear / sigmoidal classifier function, averaging the weights is probably OK.
Other kinds of models are not so easily merged. Cannot merge models of different types.

Ensemble Learning

Why does this work?

- Errors won't be correlated (hope)
- Less likely that most models are all wrong (hope)
- Reduced variance

Online Stochastic Gradient Descent as Ensemble



No iteration!

(Well, each mapper iterates over its partition, that's how MapReduce works!)

Additional explanation I missed in lectures: Technically, a single value is a set of 1! We can select the elements in random order and compute the gradient ONE training value at a time. Combined with Inertia we might be on to something…

In MapReduce

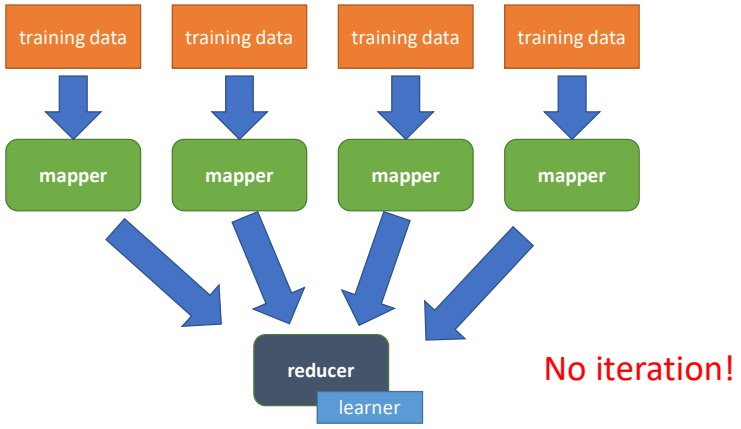
Each mapper holds the current parameter set in memory (create in setup)

Trains model by computing gradient for one element, updating parameters (in map)

Outputs Model (in cleanup)

What to do with all these models?
Ensemble learning!

Online Stochastic Gradient, One Model



If you only want one model, the mappers become “parsers” (parsing the strings and emitting feature – label pairs)
The learner is the run on the reducer.

In MapReduce, One Model

Mappers are parsers only

Reducer becomes the learner:

- creates model (setup)
- Trains model (loop in reduce)
- Emits model

No more ensemble learning!
(Unless we set the number of reducers to something greater than 1)

In Spark

```
model = LogisticRegressionWithSGD(data,
                                  iterations,
                                  step)
```

That's right! The SparkML package has lots of models, estimators, loss functions. No need to write your own.

Except on the assignment, obviously

Sentiment Analysis Case Study

Binary polarity classification: {positive, negative} sentiment

Use the "emoticon trick" to gather data

Data

Test: 500k positive/500k negative tweets from 9/1/2011
Training: {1m, 10m, 100m} instances from before (50/50 split)

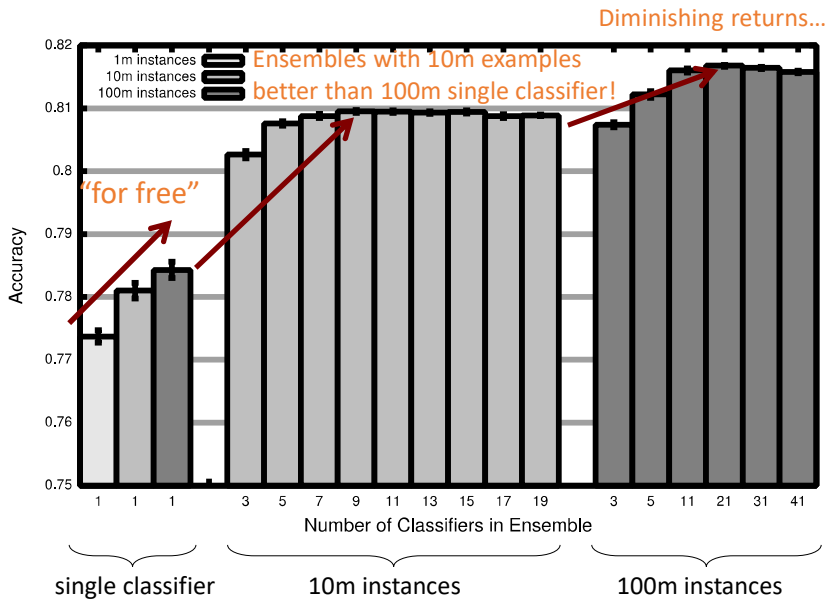
Features:

Sliding window byte-4grams

Models + Optimization:

Logistic regression with SGD (L2 regularization)
Ensembles of various sizes (simple weighted voting)

Source: Lin and Kolcz. (2012) Large-Scale Machine Learning at Twitter. SIGMOD.


Diminishing returns…
1m instances
10m instances
100m instances
Ensembles with 10m examples
better than 100m single classifier!
"for free"
Accuracy
0.82
0.81
0.8
0.79
0.78
0.77
0.76
0.75
1 1 1 3 5 7 9 11 13 15 17 19 3 5 11 21 31 41
Number of Classifiers in Ensemble
single classifier
10m instances
100m instances

Evaluation

How can we evaluate the model?

It's not enough to minimize loss. Why?

Measure Accuracy?

Still not enough! Why?

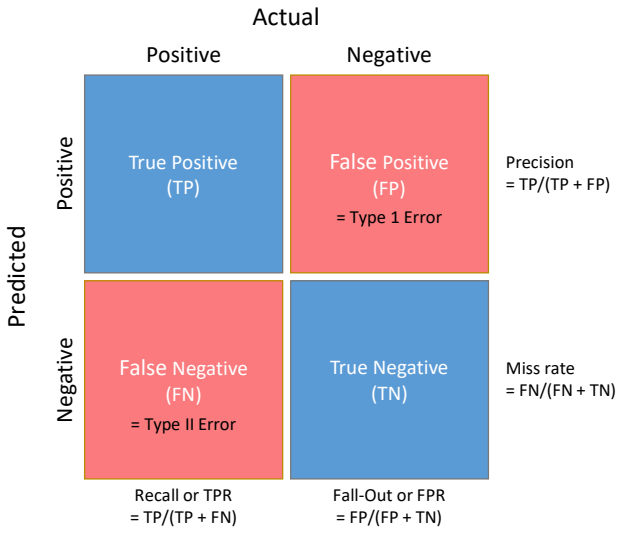
Loss is a measure of "how closely does it match the training data". An overfitted model can have 0 loss, but still be useless.

Accuracy isn't as cut and dry. There's really FOUR different measures that are all interesting, and they're at cross purposes. (See 2 slides forward)

The Struggle is Real

"Overfitting" is the key word. A model that's PERFECT at predicting the training set might end up WORSE.
(Big Data helps this…the bigger the training set, the more "representative" it is.

Metrics



Statistics again!
False Positive – Predictor says “yes”, but it’s a no
False Negative – Predictor says “no”, but it’s a yes

All measures shown (precision, recall, fall-out, miss rate) are interesting! But improving one can harm another

ROC and PR Curves

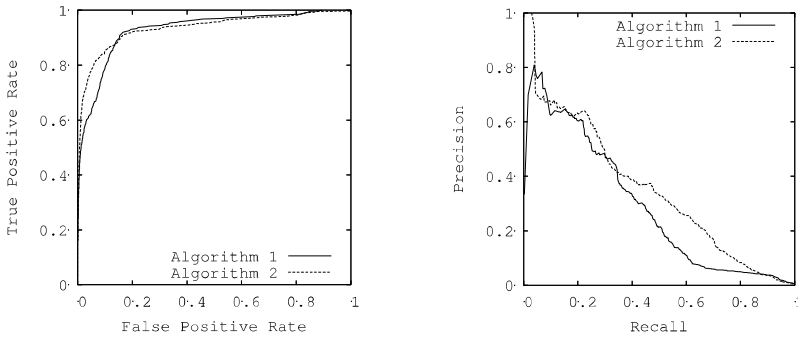
- ROC: Graph Recall (True Positive Rate) vs Fall-Out (False Positive Rate)
- PR: Precision vs Recall

In both cases, illustrates how the performance changes as you vary the threshold

A **receiver operating characteristic curve**, or **ROC curve**, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

Dan, draw them on the board.

ROC and PR Curves



Source: Davis and Goadrich. (2006) The Relationship Between Precision-Recall and ROC curves

ROC represents "as you vary the threshold, how many more false positives slip in, and how many more true positives are successfully found."

What's usually done is the area under the curve, or ROCA. A ROCA of 0.5 means it's random chance. Lowering the threshold introduces as many true positives as it does false positives. ROCA of 1 means it's a perfect predictor : regardless of threshold, false positive rate is 0%, true positive rate is 100%

So summing things up:
"Don't use accuracy! There are many statistical metrics."
Which is best?
"A ROC curve"
Can I put a number to it?
"Yes, Area-under-curve (AUC), or ROCA for short"
And that number represents…?
"OK so yes it represents accuracy. But like, threshold invariant accuracy. It's different. Shut up"

Problem: Big Data Isn't Big Enough!

What's the testing set?

- Some data that wasn't in the training set

  "Holdout Method"

The less data we train on, the worse out model!

The less data we test on, the less we trust our model!

K-Fold Cross Validation

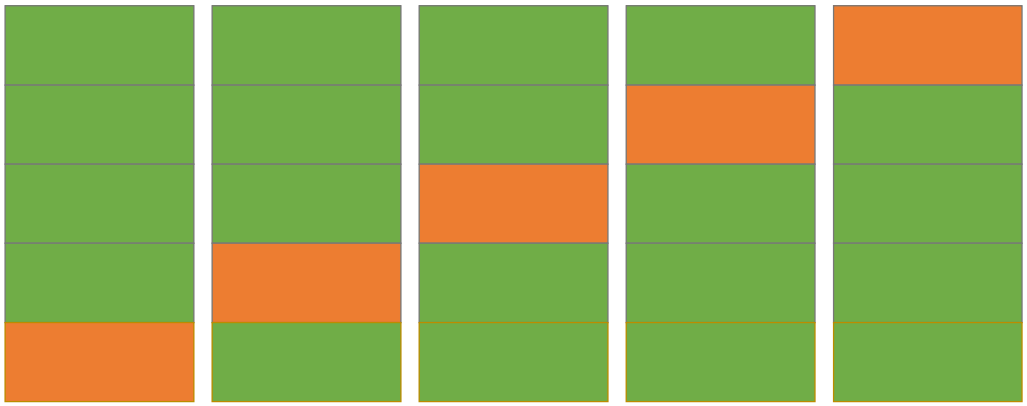
Divide Data into K subsets

Repeat holdout method k times. Each time, one subset is the test set, the rest are the training sets

Each set is used to validate ONCE

Each set is used to train K-1 times

5-Fold Cross Validation



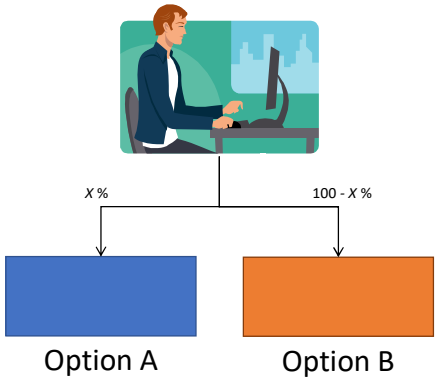
Bonus? We have 5 models, can do 5 way voting? Or just merge the models. It depends on the kind of model.

Why Cross-Validate?

This kind of ML is “offline” – When the technique is validated, then it can be deployed on live data in real-time.

Cross Validation gives you confidence that the technique will work on typical data sets.

A/B (Online) Testing



Gather metrics, compare alternatives

A/B testing is usually used for medical treatments. "Double Blind" – neither the patient nor the physician administering treatment knows which treatment it is. Usually one option (control) is a placebo or an already proven treatment.

Used in Marketing to check for a single variable.

A/B Testing for ML

Step 1: Offline training and evaluation (holdout, cross-validate, etc)

Step 2: A/B testing vs other methods

Return to Step 1 if needed

You can use this to compare different feature selection methods or compare to current best-practice models.

Applied ML in Academia

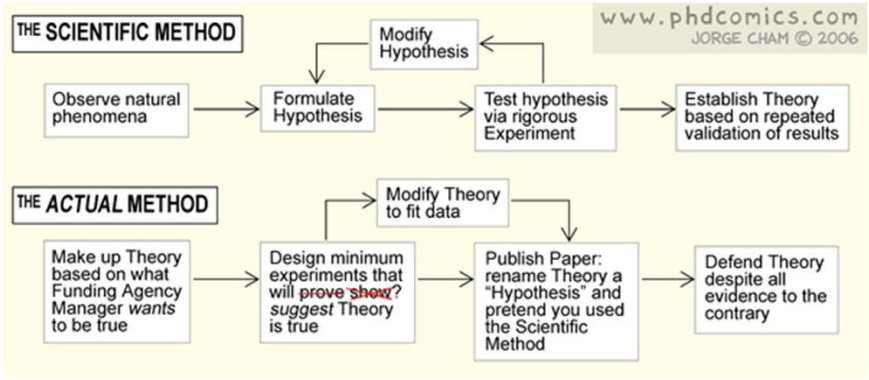
Download interesting dataset (comes with the problem)

Run baseline model
Train/Test

Build better model
Train/Test

Does new model beat baseline?
Yes: publish a paper!
No: try again!

These few slides date back to Jimmy Lin. I'm not a ML person…but…ok, guilty…if you train a model and it's worse than the current standard, you can't publish that! So you keep trying until you can.


THE SCIENTIFIC METHOD
Modify Hypothesis
Observe natural phenomena
Formulate Hypothesis
Test hypothesis via rigorous Experiment
Establish Theory based on repeated validation of results
www.phdcomics.com
JORGE CHAM © 2006
THE ACTUAL METHOD
Modify Theory to fit data
Make up Theory based on what Funding Agency Manager wants to be true
Design minimum experiments that will prove show? suggest Theory is true
Publish Paper: rename Theory a "Hypothesis" and pretend you used the Scientific Method
Defend Theory despite all evidence to the contrary

Harvard Business Review

DATA

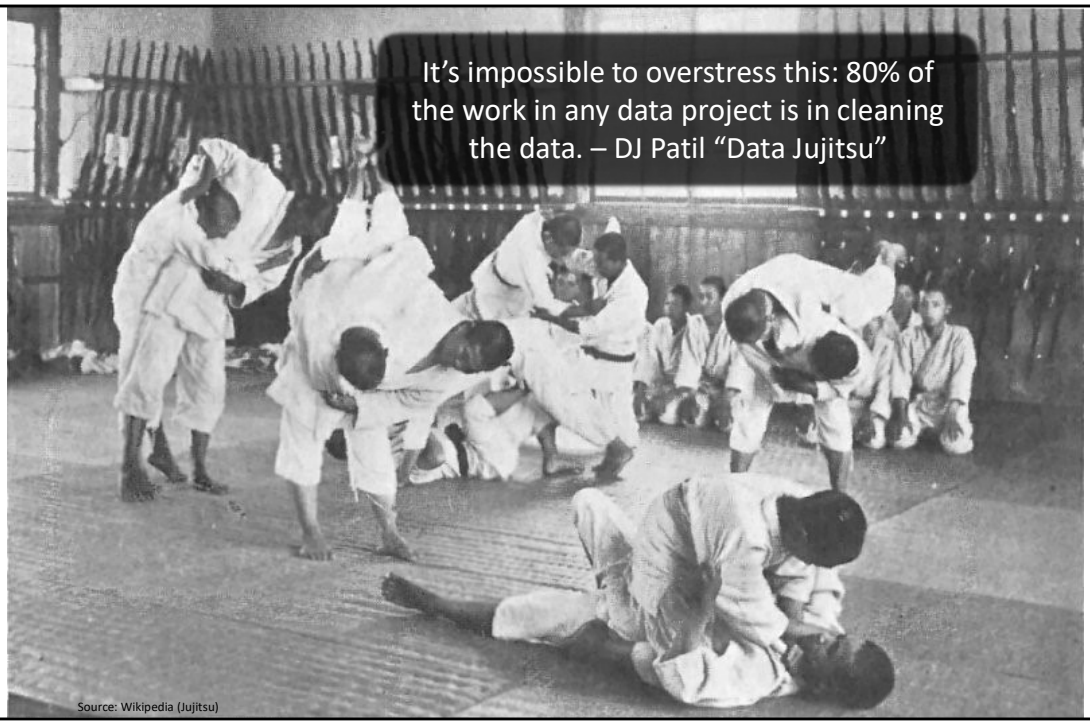
Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

FROM THE OCTOBER 2012 ISSUE

Fantasy	Reality
Extract features	What's the task?
Develop cool ML technique	Where's the data?
#Profit	What's in this dataset?
	What's all the f#$!* crap?
	Clean the data
	Extract features
	"Do" machine learning
	Fail, iterate…

Dirty little secret. Data science is barely about ML at all!


It’s impossible to overstress this: 80% of the work in any data project is in cleaning the data. – DJ Patil “Data Jujitsu”
Source: Wikipedia (Jujitsu)

On finding things…

On naming things…

CamelCase

smallCamelCase

snake_case

camel_Snake

dunder__snake

uid

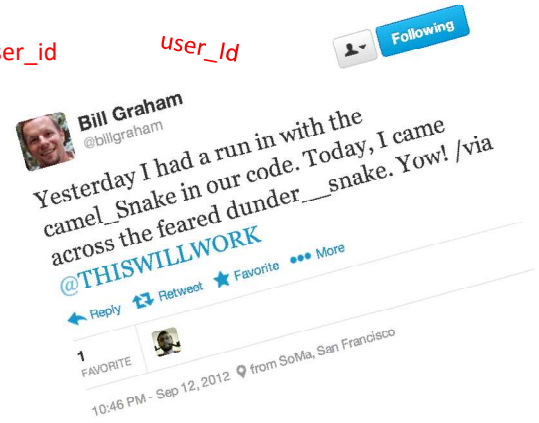
UserId

userId

userid

user_id

user_Id



On feature extraction…

```
^(\\w+\\s+\\d+\\s+\\d+:\\d+:\\d+)\\s+
([^@]+?)@(\\S+)\\s+(\\S+):\\s+(\\S+)\\s+(\\S+)
\\s+((?:\\S+?,\\s+)*(?:\\S+?))\\s+(\\S+)\\s+(\\S+)
\\s+\\[([^\\]]+)\\]\\s+\"(\\w+)\\s+([^\"\\\\]*
(?:\\\\.[^\"\\\\]*)*)\\s+(\\S+)\"\\s+(\\S+)\\s+
(\\S+)\\s+\"([^\"\\\\]*(?:\\\\.[^\"\\\\]*)*)
\"\\s+\"([^\"\\\\]*(?:\\\\.[^\"\\\\]*)*)\"\\s*
(\\d*-[\\d-]*)?\\s*(\\d+)?\\s*(\\d*\\.[\\d\\.]*)?
(\\s+[-\\w]+)?.*$
```

An actual Java regular expression used to parse log message at Twitter circa 2010

Friction is cumulative!

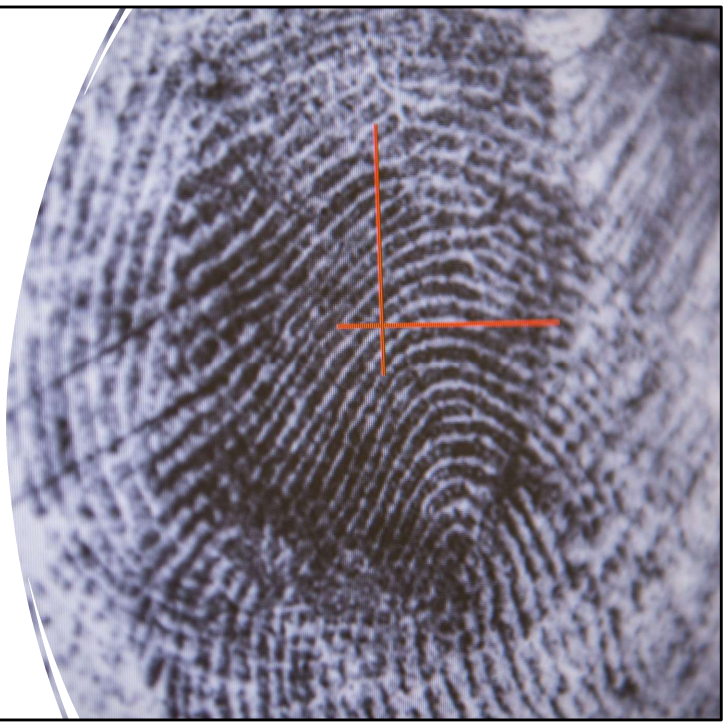
OK, My Model (Finally) Works on the Training Set

Good, you've made a good first step!

☹

I did A/B Testing, and it has good Precision & Recall!

OK, you're half-way there!



What's the other half?

Production

Source: Wikipedia (Oil refinery)

Production

What are your dependencies?
How / When are your job(s) scheduled?
Do you have enough resources?
How do you know if it's working?
What happens if it stops working?

All about infrastructure

TO BE CONTINUED…

(The file size is getting too big, going to make this two files)