RECAP: THE COURSE SO FAR…

First Lecture - Review

- Admin stuff
- Review – **Why** to scale, **how** to count and **what** to count
  - How: O(…)

Why to scale: *c.* 2001 (Banko & Brill, ACL 2001)

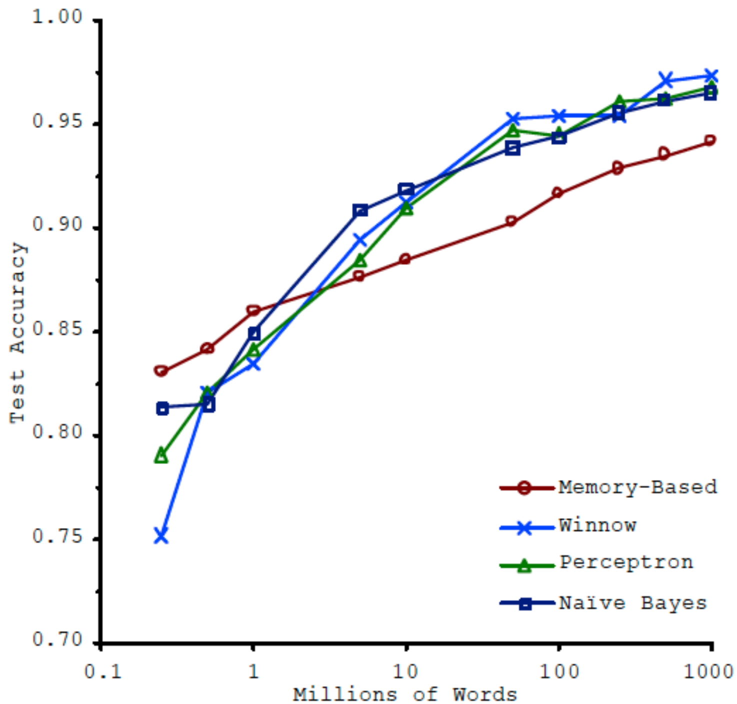


Figure 1. Learning Curves for Confusion Set Disambiguation

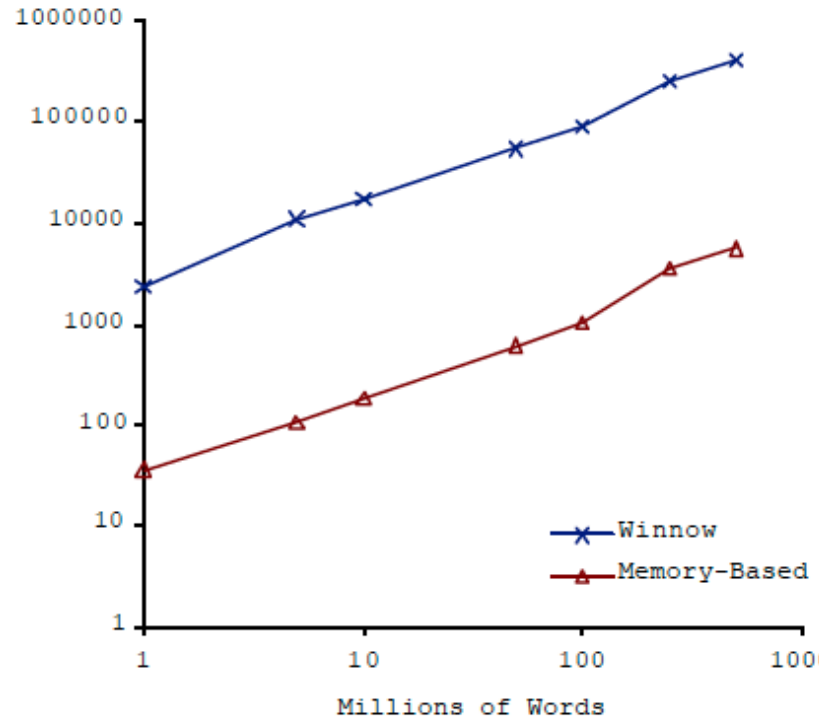


Figure 2. Representation Size vs. Training Corpus Size

Task: distinguish pairs of easily-confused words ("affect" vs "effect") in context

Numbers (Jeff Dean says) Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Update: Colin Scott, UCB

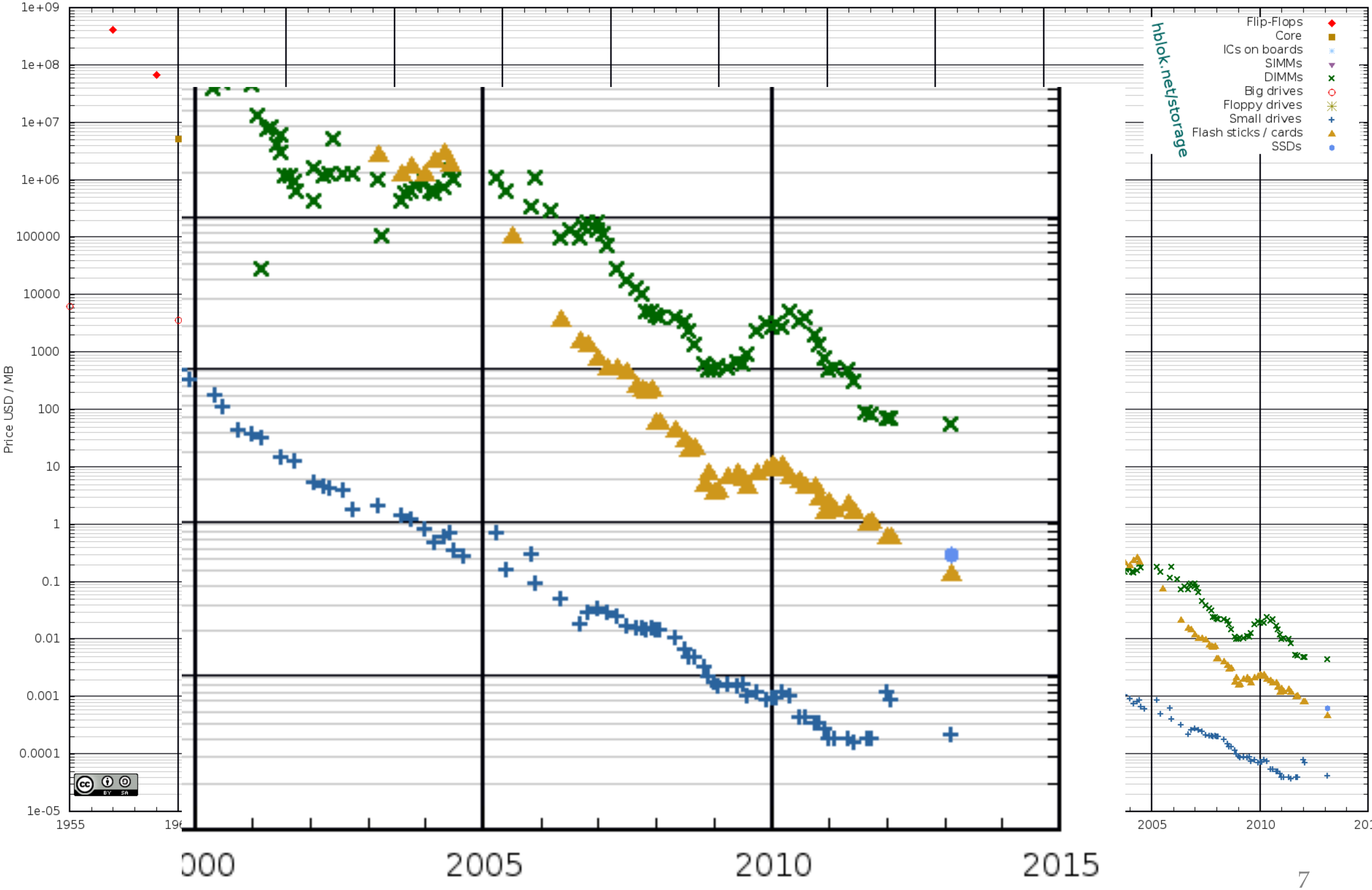
file:///Users/wcohen/Documents/code/interactive_latencies/interactive_latency.html

This isn't on the web but is available thru GitHub

What's Happening with Hardware?

- Clock speed: stuck at 3Ghz for ~ 10 years
- Net bandwidth doubles ~ 2 years
- Disk bandwidth doubles ~ 2 years
- SSD bandwidth doubles ~ 3 years
- Disk seek speed doubles ~ 10 years
- SSD latency nearly saturated

Historical Cost of Computer Memory and Storage

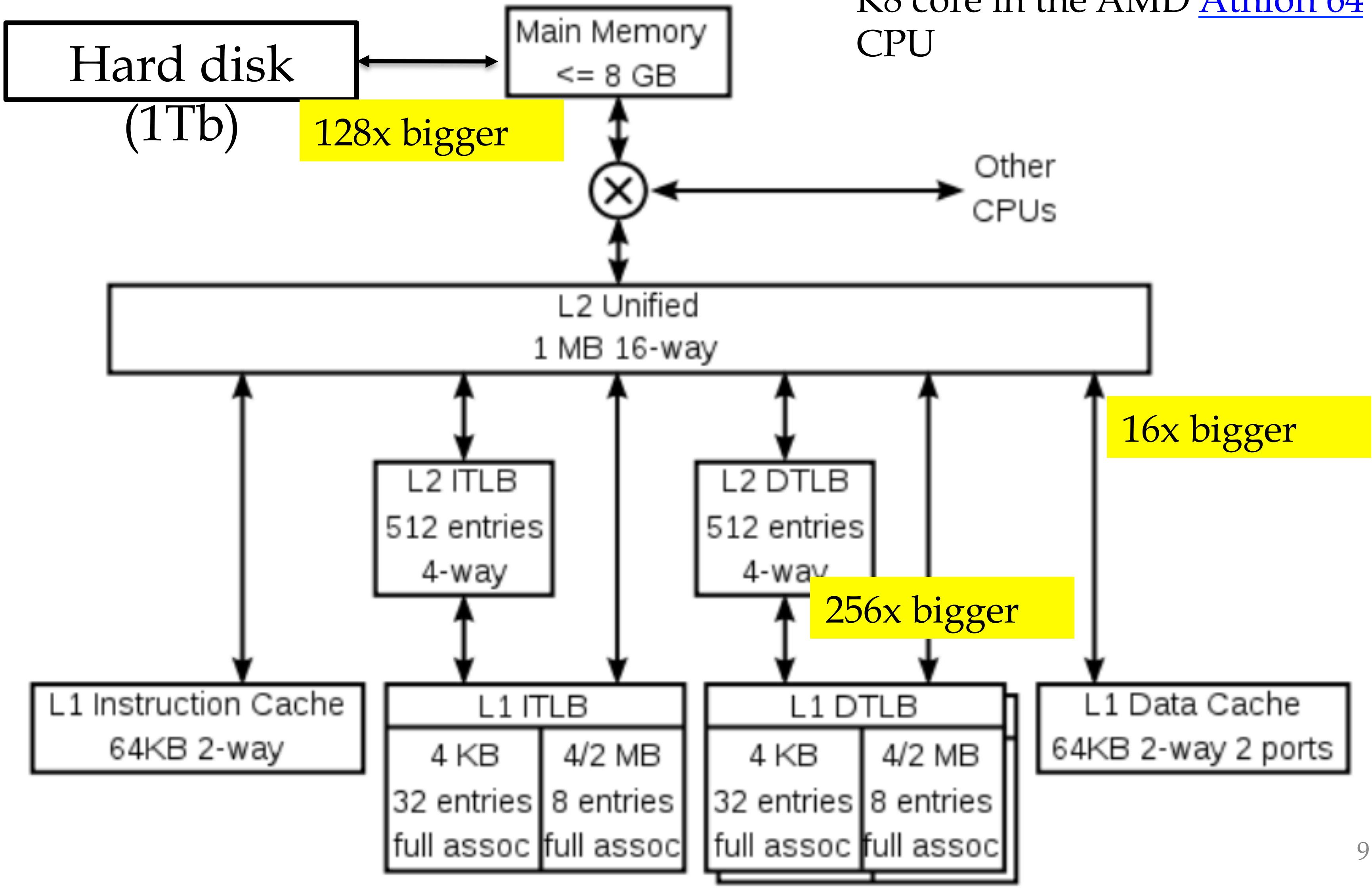


Numbers (Jeff Dean says) Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

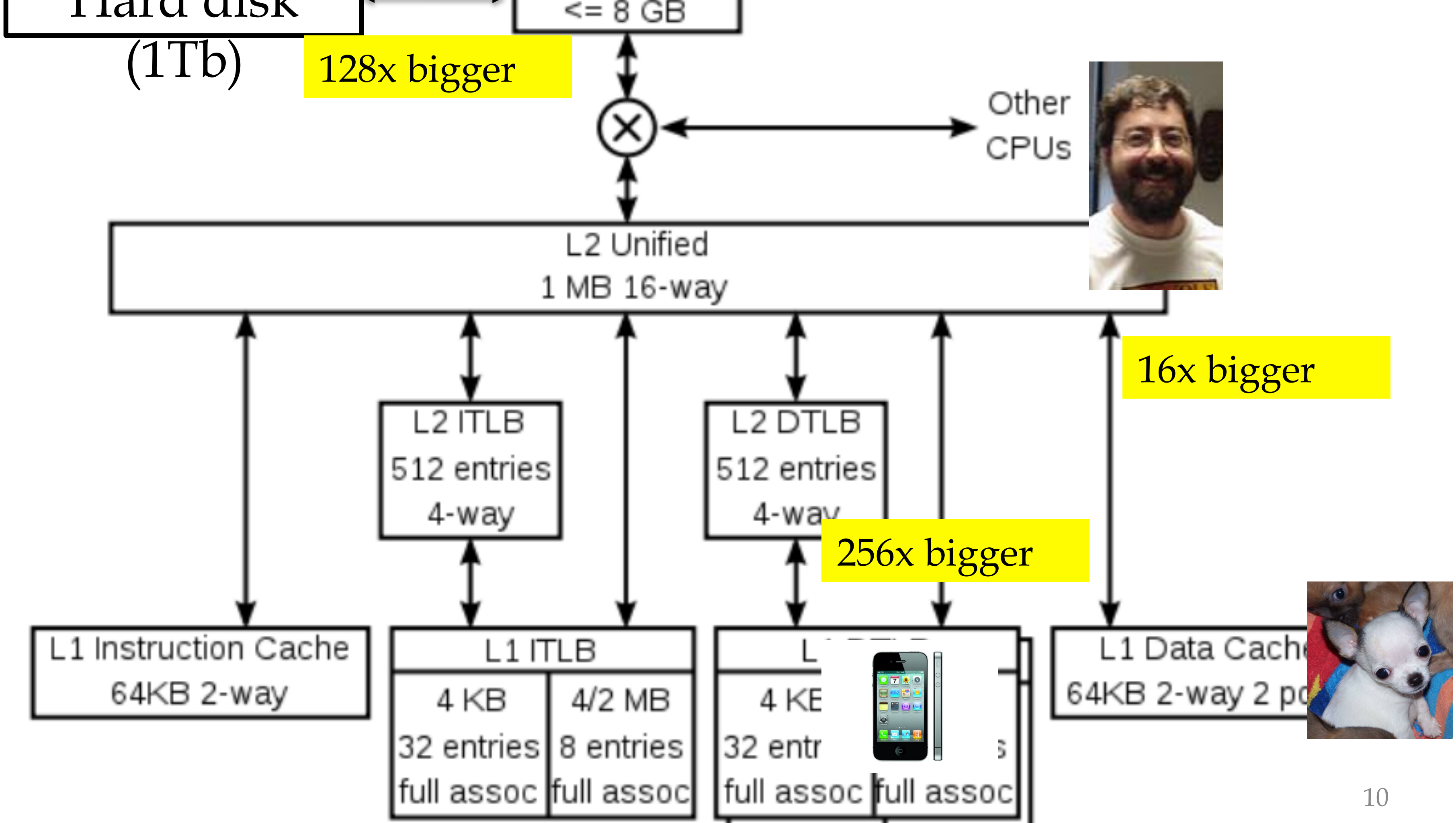
A typical CPU (not to scale)

K8 core in the AMD [Athlon 64]() CPU



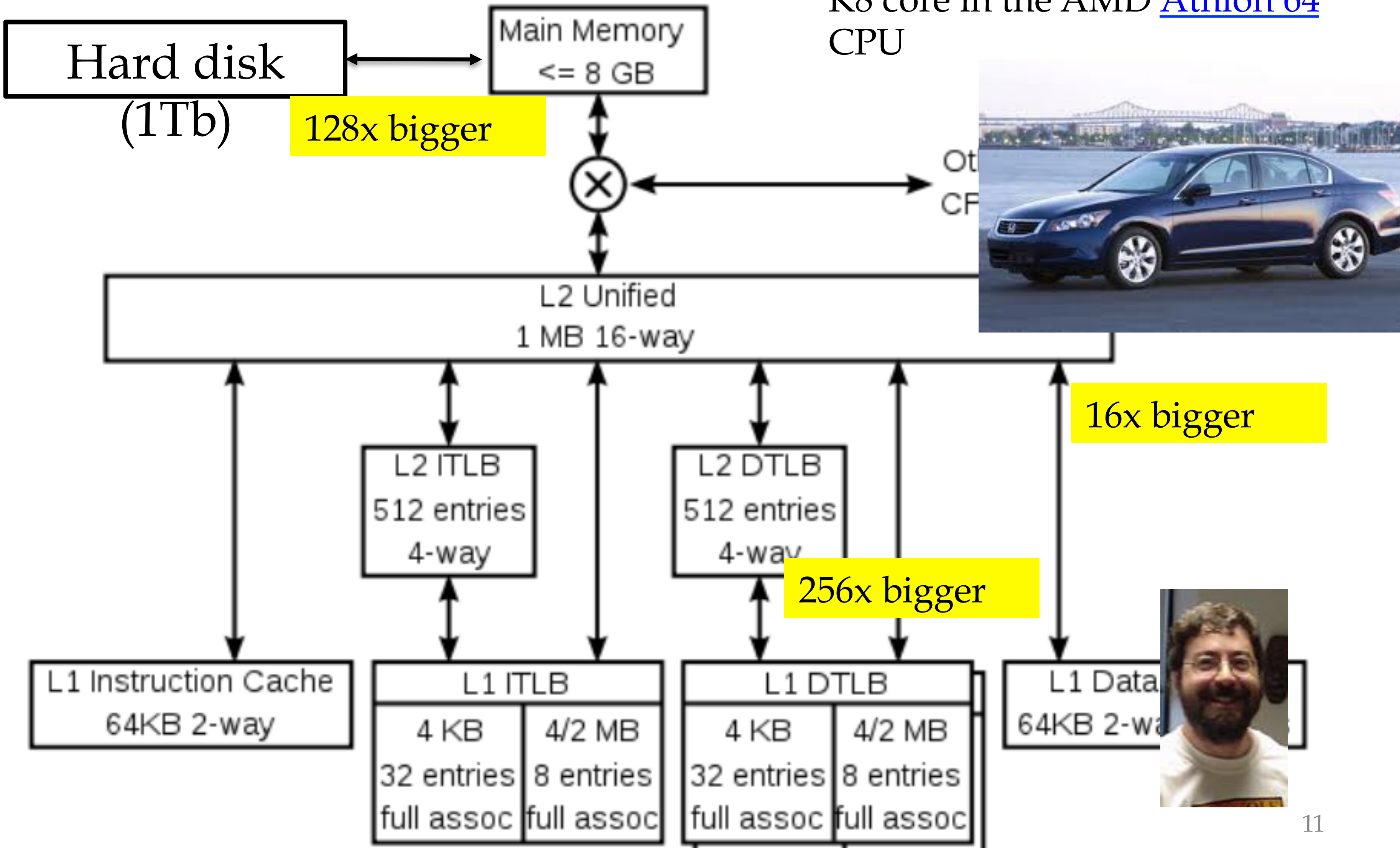
A typical CPU (not to scale)

K8 core in the AMD Athlon 64 CPU



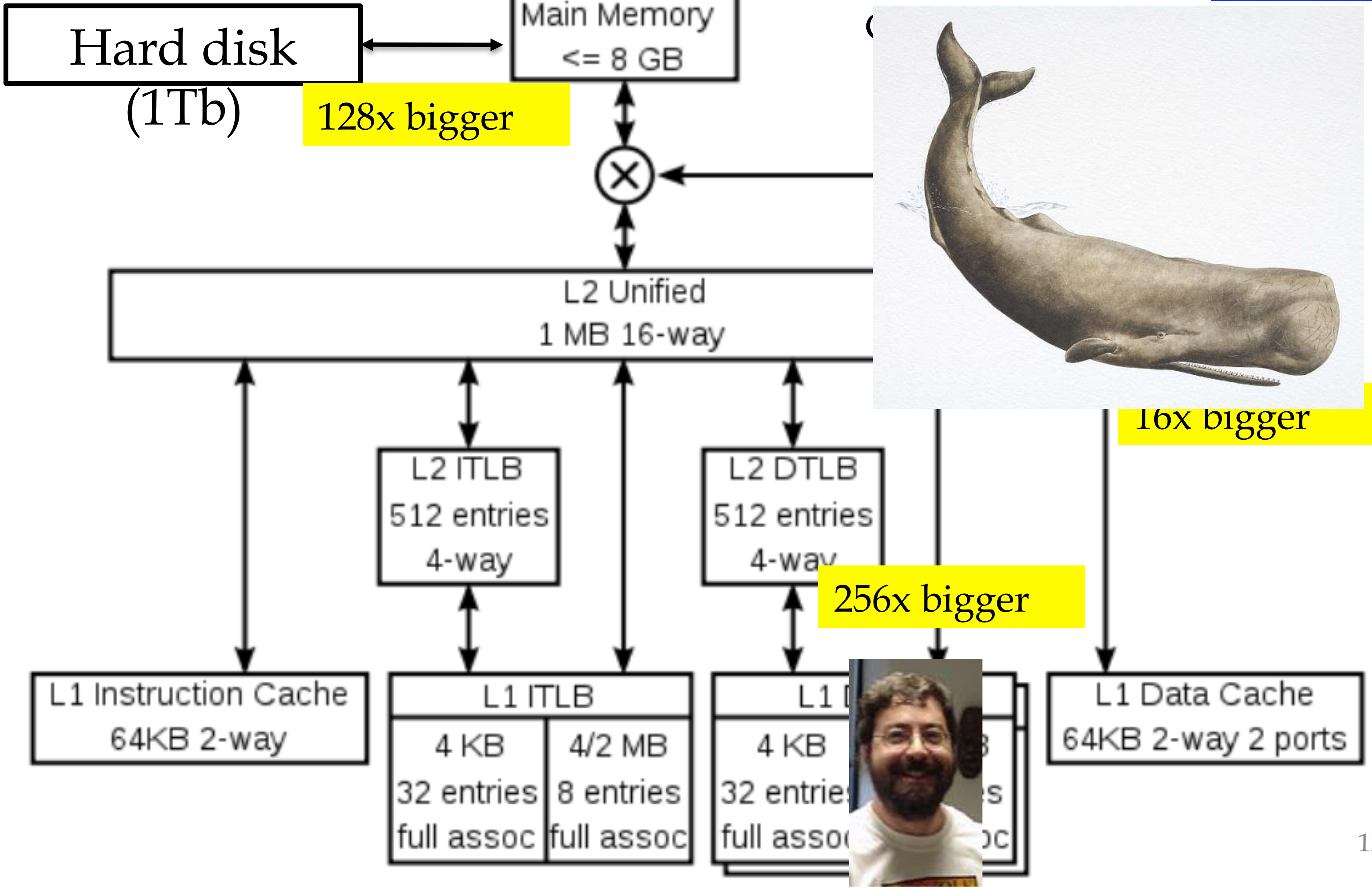
A typical CPU (not to scale)

K8 core in the AMD Athlon 64 CPU



A typical CPU (not to scale)

K8 core in the AMD Athlon 64



A typical disk

Numbers (Jeff Dean says) Everyone Should Know

L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	
Mutex lock/unlock	100 ns	
Main memory reference	100 ns	
Compress 1K bytes with Zippy	10,000 ns	
Send 2K bytes over 1 Gbps network	20,000 ns	
Read 1 MB sequentially from memory	250,000 ns	
Round trip within same datacenter	500,000 ns	
Disk seek	10,000,000 ns	40x
Read 1 MB sequentially from network	10,000,000 ns	~= 100,000x
Read 1 MB sequentially from disk	30,000,000 ns	
Send packet CA->Netherlands->CA	150,000,000 ns	

What do we count?

- Compilers don’t warn Jeff Dean. Jeff Dean warns compilers.
- ….
- Memory access/instructions are *qualitatively different* from disk access
- Seeks are *qualitatively different* from sequential reads on disk
- Cache, disk fetches, etc work best when you stream through data *sequentially*
- Best case for data processing: stream through the data *once* in *sequential order,* as it’s found on disk.

Other lessons -?

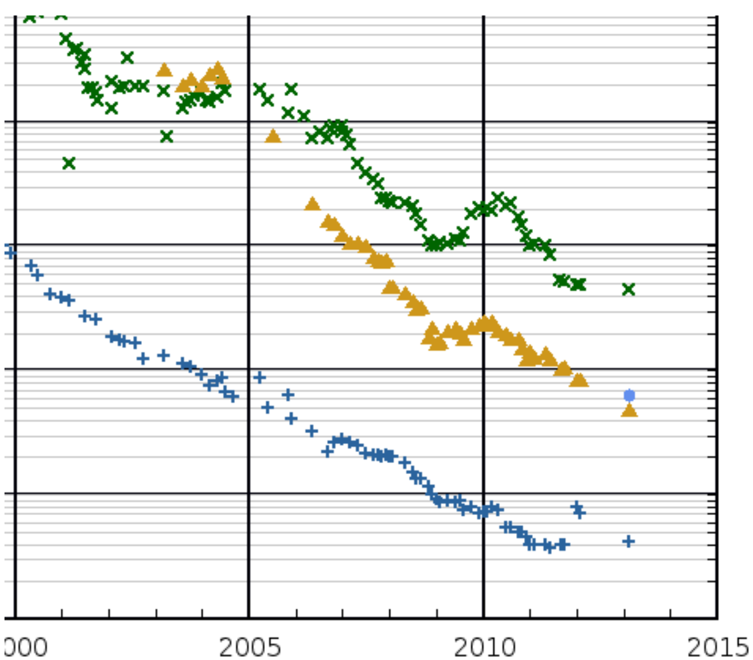
Encoding Your Data

- CPUs are fast, memory/bandwidth are precious, ergo…
  - Variable-length encodings
  - Compression
  - Compact in-memory representations
- Compression very important* aspect of many systems
  - inverted index posting list formats
  - storage systems for persistent data

* but not important enough for this class's assignments….

What to count

Operation	~ Time	x/100ns	x/10M ns
random access, RAM	100 ns	1	
read 1 Mb sequentially – RAM	250,000 ns	2,500	
random access, disk (seek)	10,000,000 ns	100,000	1
read 1Mb sequentially - net	10,000,000 ns		1
read 1Mb sequentially - disk	30,000,000 ns		3



First lecture: review

- Admin stuff
- Review – **Why** to scale, **how** to count and **what** to count

- What sort of computations do we want to *do* in (large-scale) machine learning programs?
  - *Probability*

PROBABILITY AND SCALABILITY: LEARNING AND COUNTING

Big ML *c.* 2001 (Banko & Brill, "Scaling to Very Very Large…", ACL 2001)

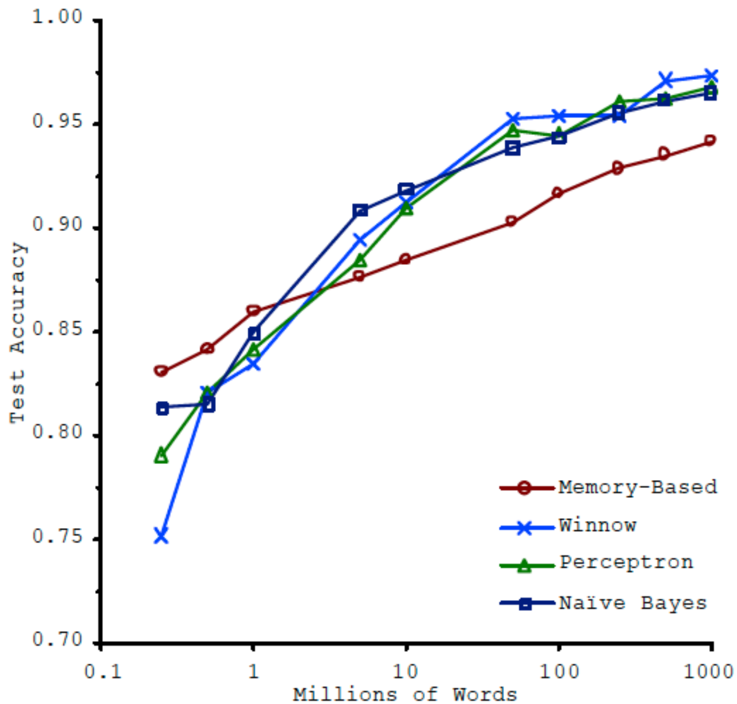


Figure 1. Learning Curves for Confusion Set Disambiguation

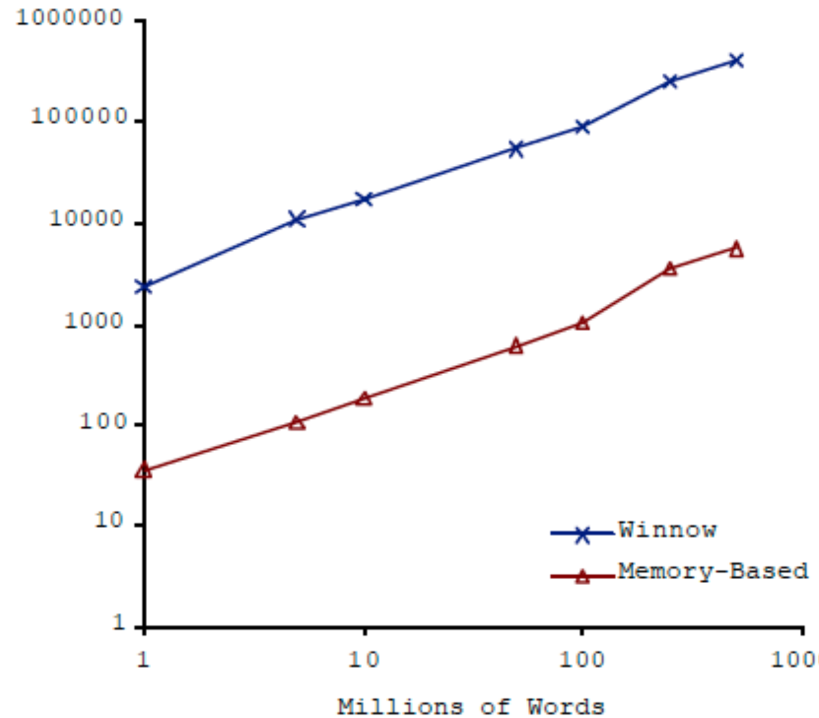


Figure 2. Representation Size vs. Training Corpus Size

Task: distinguish pairs of easily-confused words ("affect" vs "effect") in context

Why More Data Helps: A Demo

- Data:
  - All 5-grams that appear >= 40 times in a corpus of 1M English books
    - approx 80B words
    - 5-grams: 30Gb compressed, 250-300Gb uncompressed
    - Each 5-gram contains frequency distribution over *years*
  - Wrote code to compute
    - Pr(A,B,C,D,E|C=affect or C=effect)
    - Pr(any subset of A,…,E|any other fixed values of A,…,E with C=affect V effect)
  - Demo:
    - /Users/wcohen/Documents/code/pyhack/bigml
    - eg: python ngram-query.py data/aeffect-train.txt _ _B effect _ _

Why More Data Helps

- Data:
  - All 5-grams that appear >= 40 times in a corpus of 1M English books
    - approx 80B words
    - 5-grams: 30Gb compressed, 250-300Gb uncompressed
    - Each 5-gram contains frequency distribution over *years*
  - Wrote code to compute
    - Pr(A,B,C,D,E | C=affect or C=effect)
    - Pr(any subset of A,…,E | any other fixed values of A,…,E with C=affect V effect)
- Observations [from playing with data]:
  - Mostly effect not affect
  - Most common word before affect is not
  - After not effect most common word is a
  - …

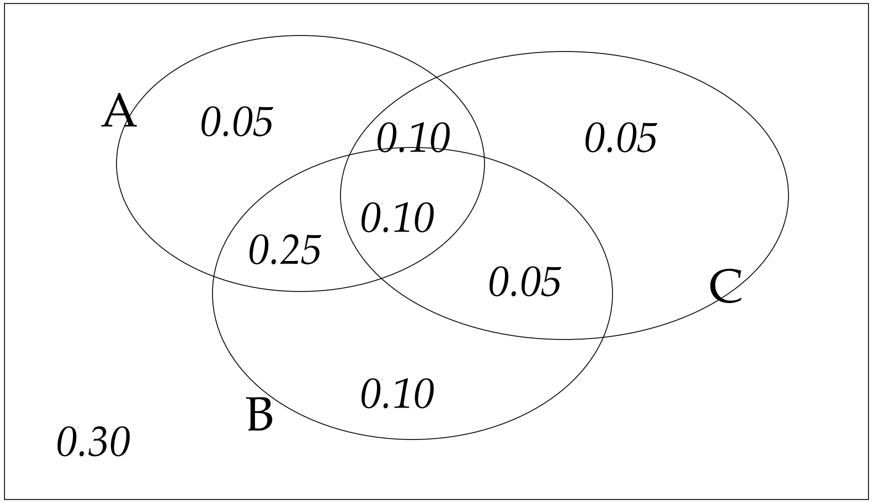
The Joint Distribution

Example: Boolean variables A, B, C

Recipe for making a joint distribution of M variables:

1. Make a truth table listing all combinations of values of your variables (if there are M Boolean variables then the table will have 2^M rows).
2. For each combination of values, say how probable it is.
3. If you subscribe to the axioms of probability, those numbers must sum to 1.

A	B	C	Prob
0	0	0	0.30
0	0	1	0.05
0	1	0	0.10
0	1	1	0.05
1	0	0	0.05
1	0	1	0.10
1	1	0	0.25
1	1	1	0.10



Some of the Joint Distribution

A	B	C	D	E	p
is	the	effect	of	the	0.00036
is	the	effect	of	a	0.00034
.	The	effect	of	this	0.00034
to	this	effect	:	“	0.00034
be	the	effect	of	the	…
	…	…	…	…	…
not	the	effect	of	any	0.00024
	…	…	…	…	…
does	not	affect	the	general	0.00020
does	not	affect	the	question	0.00020
any	manner	affect	the	principle	0.00018

An experiment: how useful is the brute-force joint classifier?

- Extracted all affect/effect 5-grams from an old Reuters corpus
  - about 20k documents
  - about 723 n-grams, 661 distinct
  - Financial news, not novels or textbooks
- Tried to predict center word with:
  - $\Pr(C \mid A=a, B=b, D=d, E=e)$
  - then $P(C \mid A,B,D)$
  - then $P(C \mid B,D)$
  - then $P(C \mid B)$
  - then $P(C)$

EXAMPLES

- "The cumulative _ of the" → effect (1.0)
- "Go into _ on January" → effect (1.0)
- "From cumulative _ of accounting" not present in train data
  - Nor is ""From cumulative _ of _"
  - But "_ cumulative _ of _" → effect (1.0)
- "Would not _ Finance Minister" not present
  - But "_ not _ _ _" → affect (0.9625)

Performance summary

Pattern	Used	Errors
P(C | A,B,D,E)	101	1
P(C | A,B,D)	157	6
P(C | B,D)	163	13
P(C | B)	244	78
P(C)	58	31

3% error

5% error

15% error

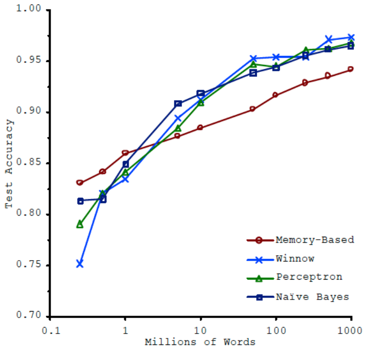


Figure 1. Learning Curves for Confusion Set Disambiguation

Is this a useful density estimate?

5% error

What Have We Learned?

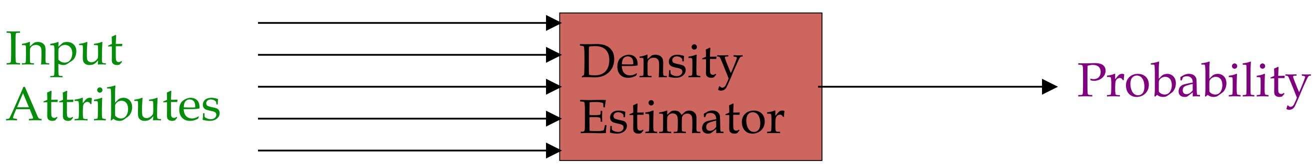
- Counting's not enough -?
- Counting goes a long way with big data -?

- Big data can sometimes be made small
  - For a specific task, like this one
  - It's all in the data preparation -?

- Often *density estimation* is more important than *classification*
- Counts are a good? density estimator

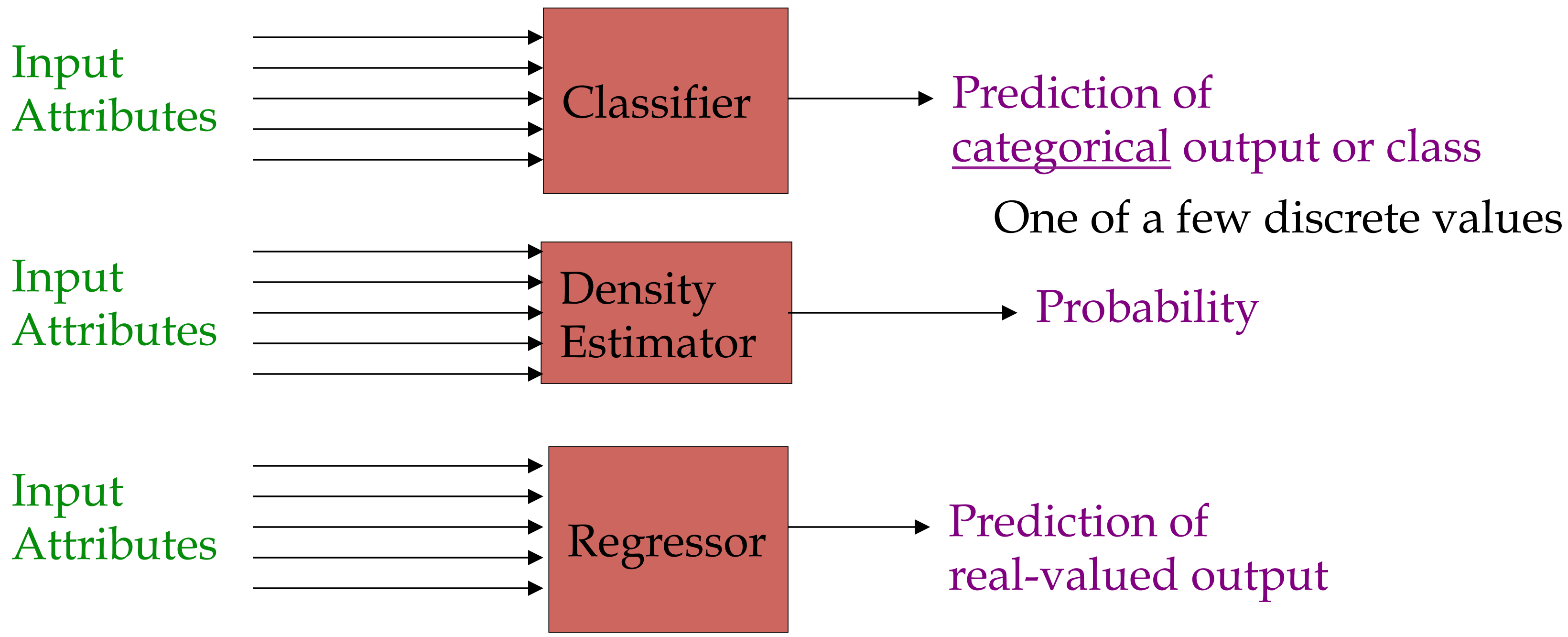
Density Estimation

- Our Joint Distribution learner is our first example of something called Density Estimation
- A Density Estimator learns a mapping from a set of attributes values to a Probability

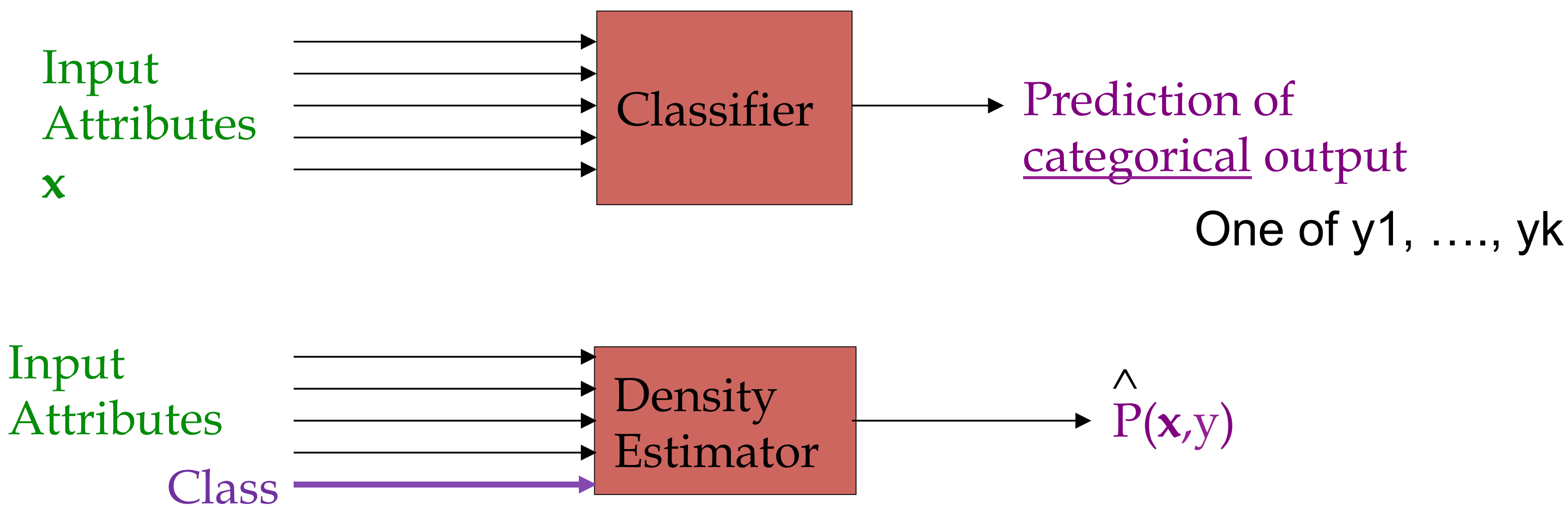


Density Estimation

- Compare it against the two other major kinds of models:



Density Estimation → Classification



To classify **x**

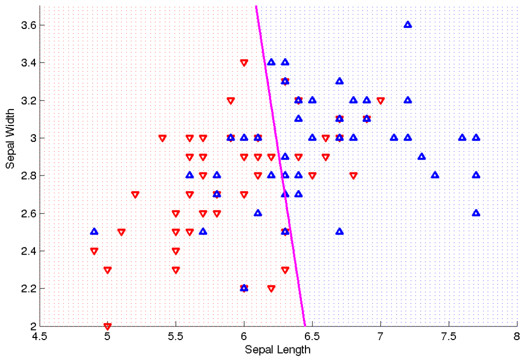
1. Use your estimator to compute $\hat{P}(\mathbf{x},y1)$, …., $\hat{P}(\mathbf{x},yk)$
2. Return the class y* with the highest predicted probability

Ideally is correct with $\hat{P}(x,y^*) = \hat{P}(x,y^*)/(\hat{P}(\mathbf{x},y1) + \ldots + \hat{P}(\mathbf{x},yk))$

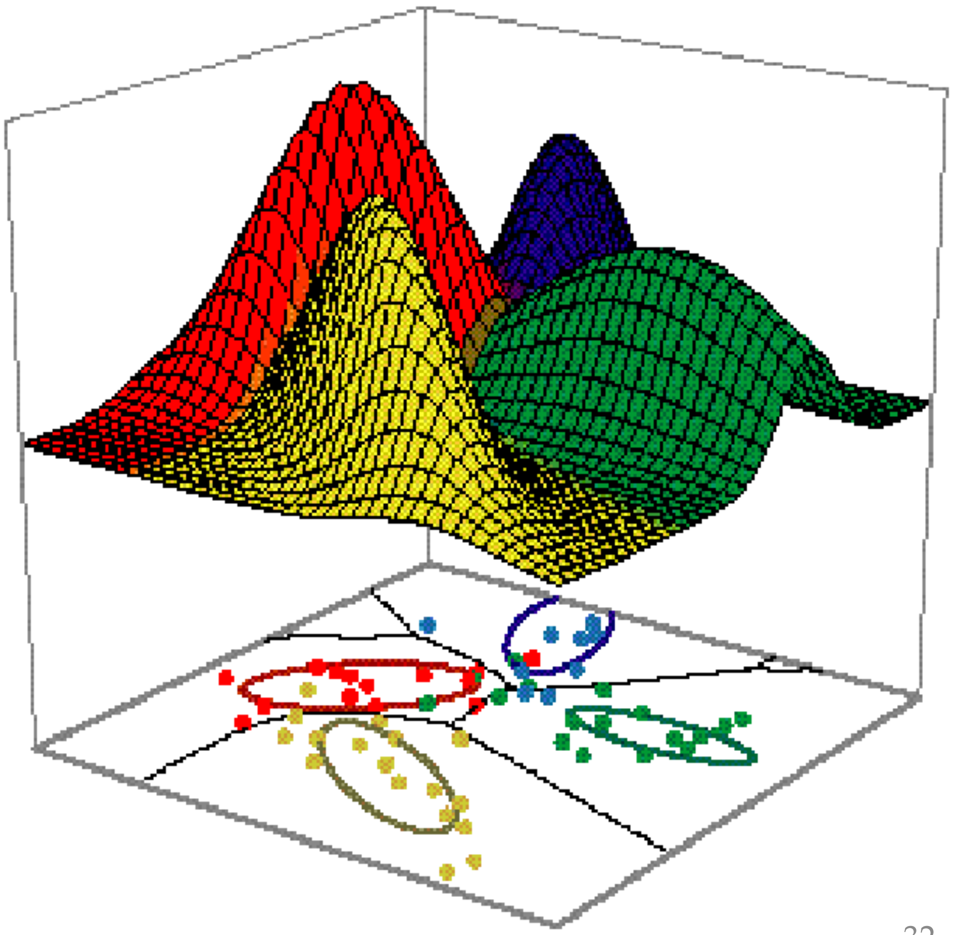
Binary case:
predict POS
if $\hat{P}(\mathbf{x})>0.5$

Classification vs Density Estimation

Classification



Density Estimation



Classification vs density estimation

PROBABILITY AND SCALABILITY: NAÏVE BAYES

Second most scalable learning method in the world?

Performance …

Pattern	Used	Errors
P(C | A,B,D,E)	101	1
P(C | A,B,D)	157	6
P(C | B,D)	163	13
P(C | B)	244	78
P(C)	58	31

- Is this good performance?
- If we care about recall, what should we do?

Naïve Density Estimation

What's an alternative to the joint distribution?

The naïve model generalizes strongly:

Assume that each attribute is distributed independently of any of the other attributes.

Using the Naïve Distribution

- Once you have a Naïve Distribution you can easily compute any row of the joint distribution.
- Suppose *A*, *B*, *C* and *D* are independently distributed. What is $P(A \wedge \sim B \wedge C \wedge \sim D)$?

Using the Naïve Distribution

- Once you have a Naïve Distribution you can easily compute any row of the joint distribution.
- Suppose A, B, C and D are independently distributed. What is P(A ^ ~B ^ C ^ ~D)?

P(A) P(~B) P(C) P(~D)

Naïve Distribution General Case

- Suppose $X_1, X_2, \ldots, X_d$ are independently distributed.

$$\Pr(X_1 = x_1, \ldots, X_d = x_d) = \Pr(X_1 = x_1) \cdot \ldots \cdot \Pr(X_d = x_d)$$

- So if we have a Naïve Distribution we can construct any row of the implied Joint Distribution on demand.

- How do we learn this?

Learning a Naïve Density Estimator

$$P(X_i = x_i) = \frac{\#\text{records with } X_i = x_i}{\#\text{records}}$$ MLE

$$P(X_i = x_i) = \frac{\#\text{records with } X_i = x_i + mq}{\#\text{records} + m}$$ Dirichlet (MAP)

Another trivial learning algorithm!

Is this an interesting learning algorithm? No

- For n-grams, what is $\hat{P}(C=\text{effect} \mid A=\text{will})$?
- In joint: $\hat{P}(C=\text{effect} \mid A=\text{will}) = 0.38$
- In naïve: $\hat{P}(C=\text{effect} \mid A=\text{will}) = \hat{P}(C=\text{effect}) = \#[C=\text{effect}] / \#\text{totalNgrams} = 0.94$ (!)

- What is $\hat{P}(C=\text{effect} \mid B=\text{no})$?
- In joint: $\hat{P}(C=\text{effect} \mid B=\text{no}) = 0.999$
- In naïve: $\hat{P}(C=\text{effect} \mid B=\text{no}) = \hat{P}(C=\text{effect}) = 0.94$

Can we make this interesting? Yes!

- Key ideas:
  - Pick the class variable Y
  - Instead of estimating $P(X_1,\ldots,X_n,Y) = P(X_1)*\ldots*P(X_n)*Y$, estimate $P(X_1,\ldots,X_n \mid Y) = P(X_1 \mid Y)*\ldots*P(X_n \mid Y)$
  - Or, assume $P(X_i \mid Y)=Pr(X_i \mid X_1,\ldots,X_{i-1},X_{i+1},\ldots X_n,Y)$
  - Or, that X_i is conditionally independent of every X_j, j!=i, given Y.

  - How to estimate?

MLE or MAP

The Naïve Bayes classifier – v1

- Dataset: each example has
  - A unique id *id*
    - Why? For debugging the feature extractor
  - d attributes $X_1,\ldots,X_d$
    - Each X_i takes a discrete value in $dom(X_i)$
  - One class label Y in $dom(Y)$
- You have a *train* dataset and a *test* dataset
- Assume:
  - the dataset doesn't fit in memory
  - the model does

stream through it

The Naïve Bayes classifier – v0

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example *id, y,* $x_1,\ldots,x_d$ in *train:*
  - C("*Y*=ANY") ++; C("*Y*=*y*") ++
  - For *j* in *1..d*:
    - C("$Y=y \wedge X_j=x_j$") ++
- For each example *id, y,* $x_1,\ldots,x_d$ in *test:*
  - For each *y′* in *dom(Y):*
    - Compute $\Pr(y',x_1,\ldots,x_d) = \left(\prod_{j=1}^{d}\Pr(X_j = x_j \mid Y = y')\right)\Pr(Y = y')$

$$= \left(\prod_{j=1}^{d}\frac{\Pr(X_j = x_j, Y = y')}{\Pr(Y = y')}\right)\Pr(Y = y')$$

  - Return the best *y′*

The Naïve Bayes classifier – v0

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example *id, y,* $x_1,\ldots,x_d$ in *train:*
  - C("*Y*=ANY") ++;   C("*Y*=*y*") ++
  - For *j* in *1..d*:
    - C("$Y=y \wedge X_j=x_j$") ++
- For each example *id, y,* $x_1,\ldots,x_d$ in *test:*
  - For each *y′* in *dom(Y):*
    - Compute $\Pr(y',x_1,\ldots,x_d) = \left(\prod_{j=1}^{d}\Pr(X_j = x_j \mid Y = y')\right)\Pr(Y = y')$

$$= \left(\prod_{j=1}^{d}\frac{C(X_j = x_j \wedge Y = y')}{C(Y = y')}\right)\frac{C(Y = y')}{C(Y = ANY)}$$

  - Return the best *y′*

The Naïve Bayes classifier – v0

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example $id, y, x_1, \ldots, x_d$ in *train:*
  - C("Y=ANY") ++;   C("Y=y") ++
  - For j in $1..d$:
    - C("Y=y ^ X_j=x_j") ++
- For each example $id, y, x_1, \ldots, x_d$ in *test:*
  - For each y' in $dom(Y)$:
    - Compute $\Pr(y', x_1, \ldots, x_d) = \left(\prod_{j=1}^{d} \Pr(X_j = x_j \mid Y = y')\right) \Pr(Y = y')$

$$= \left(\prod_{j=1}^{d} \frac{C(X_j = x_j \wedge Y = y')}{C(Y = y')}\right) \frac{C(Y = y')}{C(Y = ANY)}$$

This may overfit, so …

  - Return the best y'

The Naïve Bayes classifier – v1

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example $id, y, x_1,\ldots,x_d$ in *train:*
  - C("Y=ANY") ++; C("$Y=y$") ++
  - For j in $1..d$:
    - C("$Y=y \wedge X_j=x_j$") ++
- For each example $id, y, x_1,\ldots,x_d$ in *test:*
  - For each y' in $dom(Y)$:
    - Compute $\Pr(y',x_1,\ldots,x_d) = \left(\prod_{j=1}^{d} \Pr(X_j = x_j \mid Y = y')\right)\Pr(Y = y')$

$$= \left(\prod_{j=1}^{d} \frac{C(X_j = x_j \wedge Y = y') + mq_x}{C(Y = y') + m}\right)\frac{C(Y = y') + mq_y}{C(Y = ANY) + m}$$

where:
$q_j = 1/|dom(X_j)|$
$q_y = 1/|dom(Y)|$
$mq_x=1$

  - Return the best y'

This may underflow, so …

The Naïve Bayes classifier – v1

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example *id*, *y*, $x_1,\ldots,x_d$ in *train:*
  - C("*Y*=ANY") ++; C("*Y*=*y*") ++
  - For *j* in *1..d*:
    - C("*Y*=*y* ^ X_j=x_j") ++
- For each example *id*, *y*, $x_1,\ldots,x_d$ in *test:*
  - For each *y′* in *dom(Y):*
    - Compute log Pr($y',x_1,\ldots,x_d$) =

$$= \left(\sum_j \log \frac{C(X_j = x_j \wedge Y = y') + mq_j}{C(Y = y') + m} \right) + \log \frac{C(Y = y') + mq_j}{C(Y = ANY) + m}$$

  - Return the best *y′*

where:
$q_j = 1/|dom(X_j)|$
$q_y = 1/|dom(Y)|$
$mq_x = 1$

The Naïve Bayes classifier – v2

- For text documents, what features do you use?
- One common choice:
  - X_1 = first word in the document
  - X_2 = second word in the document
  - X_3 = third …
  - X_4 = …
  - …
- But: $\Pr(X_{13}=hockey \mid Y=sports)$ is probably not that different from $\Pr(X_{11}=hockey \mid Y=sports)$…so instead of treating them as different variables, treat them as different copies of the same variable

The Naïve Bayes classifier – v1

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example *id, y,* $x_1,\ldots,x_d$ in *train:*
  - C("*Y*=ANY") ++;   C("*Y*=*y*") ++
  - For *j* in *1..d*:
    - C("$Y=y \wedge X_j=x_j$") ++
- For each example *id, y,* $x_1,\ldots,x_d$ in *test:*
  - For each *y′* in *dom(Y):*
    - Compute $\Pr(y',x_1,\ldots,x_d) = \left(\prod_{j=1}^{d}\Pr(X_j = x_j \mid Y = y')\right)\Pr(Y = y')$

$$= \left(\prod_{j=1}^{d}\frac{\Pr(X_j = x_j, Y = y')}{\Pr(Y = y')}\right)\Pr(Y = y')$$

  - Return the best *y′*

The Naïve Bayes classifier – v2

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example $id, y, x_1, \ldots, x_d$ in *train:*
  - C("*Y*=ANY") ++;   C("*Y*=*y*") ++
  - For *j* in *1..d*:
    - C("$Y=y \wedge X_{\cancel{j}}=x_j$") ++
- For each example $id, y, x_1, \ldots, x_d$ in *test:*
  - For each y' in *dom(Y):*
    - Compute $\Pr(y', x_1, \ldots, x_d) = \left(\prod_{j=1}^{d} \Pr(X_{\cancel{j}} = x_j \mid Y = y')\right) \Pr(Y = y')$

$$= \left(\prod_{j=1}^{d} \frac{\Pr(X_{\cancel{j}} = x_j, Y = y')}{\Pr(Y = y')}\right) \Pr(Y = y')$$

  - Return the best y'

The Naïve Bayes classifier – v2

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example *id, y,* $x_1, \ldots, x_d$ in *train:*
  - C("*Y*=ANY") ++;   C("*Y*=*y*") ++
  - For *j* in *1..d*:
    - C("*Y*=*y* ^ X=x_j") ++
- For each example *id, y,* $x_1, \ldots, x_d$ in *test:*
  - For each *y′* in *dom(Y):*
    - Compute $\Pr(y', x_1, \ldots, x_d) = \left(\prod_{j=1}^{d} \Pr(X = x_j \mid Y = y') \right) \Pr(Y = y')$

$$= \left(\prod_{j=1}^{d} \frac{\Pr(X = x_j, Y = y')}{\Pr(Y = y')} \right) \Pr(Y = y')$$

  - Return the best *y′*

The Naïve Bayes classifier – v2

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example *id, y,* $x_1,\ldots,x_d$ in *train:*
  - C("*Y*=ANY") ++;   C("*Y*=*y*") ++
  - For *j* in *1..d*:
    - C("*Y*=*y* ^ X=x_j") ++
- For each example *id, y,* $x_1,\ldots,x_d$ in *test:*
  - For each *y′* in *dom(Y):*
    - Compute log Pr($y',x_1,\ldots,x_d$) =

$$= \left(\sum_j \log \frac{C(X = x_j \wedge Y = y') + mq_x}{C(X = ANY \wedge Y = y') + m} \right) + \log \frac{C(Y = y') + mq_y}{C(Y = ANY) + m}$$

where:
$q_j = 1/|V|$
$q_y = 1/|dom(Y)|$
$mq_x=1$

  - Return the best *y′*

The Naïve Bayes classifier – v2

- You have a *train* dataset and a *test* dataset
- To classify documents, these might be:
  - http://wcohen.com academic,FacultyHome William W. Cohen Research Professor Machine Learning Department Carnegie Mellon University Member of the Language Technology Institute the joint CMU-Pitt Program in Computational Biology the Lane Center for Computational Biology and the Center for Bioimage Informatics Director of the Undergraduate Minor in Machine Learning Bio Teaching Projects Publications recent all Software Datasets Talks Students Colleagues Blog Contact Info Other Stuff …
  - http://google.com commercial Search Images Videos ….
  - …
- How about for n-grams?

The Naïve Bayes classifier – v2

- You have a *train* dataset and a *test* dataset
- To do C-S spelling correction these might be
  - ng1223 effect a_the b_main d_of e_the
  - ng1224 affect a_shows b_not d_mice e_in
  - ….
- I.e., encode event X_i=*w* with another event X=*i_w*

- Question: are there any differences in behavior from using A,B,C,D ?

Assume hashtable holding all counts fits in memory

Complexity of Naïve Bayes

- You have a *train* dataset and a *test* dataset
- Initialize an "event counter" (hashtable) C
- For each example $id, y, x_1,\ldots,x_d$ in *train:*
  - C("Y=ANY") ++; C("$Y=y$") ++
  - For j in $1..d$:
    - C("$Y=y \wedge X=x_j$") ++
- For each example $id, y, x_1,\ldots,x_d$ in *test:*
  - For each y' in $dom(Y)$:
    - Compute log Pr$(y',x_1,\ldots,x_d)$ =

$$= \left(\sum_j \log \frac{C(X = x_j \wedge Y = y') + mq_x}{C(X = ANY \wedge Y = y') + m} \right) + \log \frac{C(Y = y') + mq_y}{C(Y = ANY) + m}$$

  - Return the best y'

Sequential reads

Complexity: O(n), n=size of *train*

where:
$q_j = 1/|V|$
$q_y = 1/|dom(Y)|$
$mq_x=1$

Sequential reads

Complexity: O($|dom(Y)|$ *n'), n'=size of *test*

Complexity of Naïve Bayes

- You have a *train* dataset and a *test* dataset
- Process:
  - Count events in the *train* dataset
    - $O(n)$, where n is total size of *train*
  - Write the counts to disk
    - $O(\min(|dom(X)| * |dom(Y)|, n))$
    - $O(|V|)$, if V is vocabulary and $dom(Y)$ is small
  - Classify the *test* dataset
    - $O(|V|+n')$
  - Worst-case memory usage:
    - $O(\min(|dom(X)| * |dom(Y)|, n))$

Naïve Bayes v2

- This is one example of a *streaming classifier*
  - Each example is only read only once
  - You can create a classifier and perform classifications at any point
  - Memory is minimal (<< O(n))
    - Ideally it would be constant
    - Traditionally less than O(sqrt(N))
  - Order doesn't matter
    - Nice because we may not control the order of examples in real life
    - This is a hard one to get a learning system to have!
- There are few competitive learning methods that as stream-y as naïve Bayes…

Rocchio's Algorithm

Motivation

- Naïve Bayes is unusual as a learner:
  - Only one pass through data
  - Order doesn't matter

Rocchio's algorithm

- Relevance Feedback in Information Retrieval, SMART Retrieval System Experiments in Automatic Document Processing, 1971, Prentice Hall Inc.

Rocchio's algorithm

Many variants of these formulae

$$DF(w) = \#\text{different docs } w \text{ occurs in}$$

$$TF(w,d) = \#\text{different times } w \text{ occurs in doc } d$$

$$IDF(w) = \frac{|D|}{DF(w)}$$

$$u(w,d) = \log(TF(w,d)+1)\cdot\log(IDF(w))$$

...as long as $u(w,d)=0$ for words not in d!

$$\mathbf{u}(d) = \langle u(w_1,d),....,u(w_{|V|},d)\rangle$$

Store only non-zeros in $\mathbf{u}(d)$, so size is $O(|d|)$

$$\mathbf{u}(y) = \alpha\frac{1}{|C_y|}\sum_{d\in C_y}\frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2} - \beta\frac{1}{|D-C_y|}\sum_{d'\in D-C_y}\frac{\mathbf{u}(d')}{\|\mathbf{u}(d')\|_2}$$

But size of $\mathbf{u}(y)$ is $O(|n_V|)$

$$f(d) = \arg\max_y \frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2}\cdot\frac{\mathbf{u}(y)}{\|\mathbf{u}(y)\|_2}$$

$$\|\mathbf{u}\|_2 = \sqrt{\sum_i u_i^2}$$

Rocchio results…

Joacchim '98, "A Probabilistic Analysis of the Rocchio Algorithm…"

	PrTFIDF	BAYES	TFIDF
Newsgroups	91.8	89.6	86.3
"acq"	88.9	88.5	84.5
"wheat"	93.9	94.8	90.9
"crude"	90.2	95.5	85.4
"earn"	90.5	90.9	90.6
"cbond"	91.9	90.9	87.7

Table 2: Maximum accuracy in percentages.

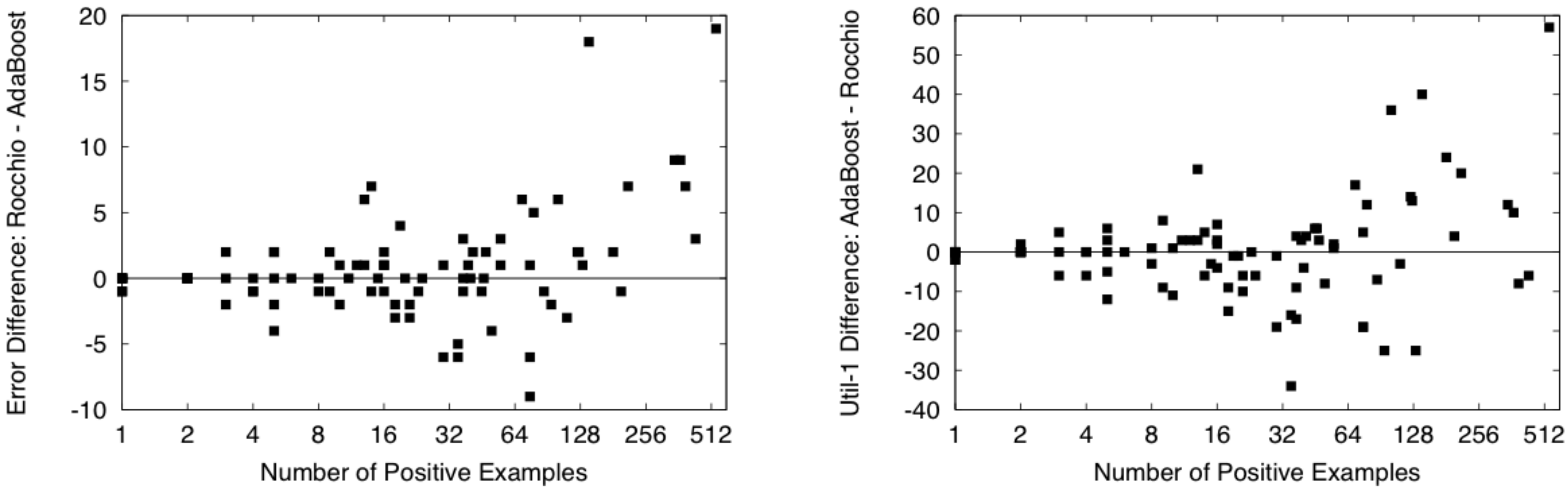
Variant TF and IDF formulas

Rocchio's method (w/ linear TF)

			#1	#2	#3	#4	#5
		# of documents	21,450	14,347	13,272	12,902	12,902
		# of training documents	14,704	10,667	9,610	9,603	9,603
		# of test documents	6,746	3,680	3,662	3,299	3,299
		# of categories	135	93	92	90	10
System	**Type**	**Results reported by**					
Word	(non-learning)	Yang [1999]	.150	.310	.290		
	probabilistic	[Dumais et al. 1998]				.752	.815
	probabilistic	[Joachims 1998]				.720	
	probabilistic	[Lam et al. 1997]	.443 (MF_1)				
PropBayes	probabilistic	[Lewis 1992a]	.650				
Bim	probabilistic	[Li and Yamanishi 1999]				.747	
	probabilistic	[Li and Yamanishi 1999]				.773	
Nb	probabilistic	[Yang and Liu 1999]				.795	
	decision trees	[Dumais et al. 1998]					.884
C4.5	decision trees	[Joachims 1998]				.794	
Ind	decision trees	[Lewis and Ringuette 1994]	.670				
Swap-1	decision rules	[Apté et al. 1994]		.805			
Ripper	decision rules	[Cohen and Singer 1999]	.683	.811		.820	
SleepingExperts	decision rules	[Cohen and Singer 1999]	**.753**	.759		.827	
Dl-Esc	decision rules	[Li and Yamanishi 1999]				.820	
Charade	decision rules	[Moulinier and Ganascia 1996]		.738			
Charade	decision rules	[Moulinier et al. 1996]		.783 (F_1)			
Llsf	regression	[Yang 1999]		.855	.810		
Llsf	regression	[Yang and Liu 1999]				.849	
BalancedWinnow	on-line linear	[Dagan et al. 1997]	.747 (M)	.833 (M)			
Widrow-Hoff	on-line linear	[Lam and Ho 1998]				.822	
Rocchio	batch linear	[Cohen and Singer 1999]	.660	.748		.776	
FindSim	batch linear	[Dumais et al. 1998]				.617	.646
Rocchio	batch linear	[Joachims 1998]				.799	
Rocchio	batch linear	[Lam and Ho 1998]				.781	
Rocchio	batch linear	[Li and Yamanishi 1999]				.625	
Classi	neural network	[Ng et al. 1997]		.802			
Nnet	neural network	Yang and Liu 1999]				.838	
	neural network	[Wiener et al. 1995]			**.820**		
Gis-W	example-based	[Lam and Ho 1998]				.860	
k-NN	example-based	[Joachims 1998]				.823	
k-NN	example-based	[Lam and Ho 1998]				.820	
k-NN	example-based	[Yang 1999]	.690	.852	**.820**		
k-NN	example-based	[Yang and Liu 1999]				.856	
	SVM	[Dumais et al. 1998]				.870	**.920**
SvmLight	SVM	[Joachims 1998]				.864	
SvmLight	SVM	[Li Yamanishi 1999]				.841	
SvmLight	SVM	[Yang and Liu 1999]				.859	
AdaBoost.MH	committee	[Schapire and Singer 2000]		**.860**			
	committee	[Weiss et al. 1999]				**.878**	
	Bayesian net	[Dumais et al. 1998]				.800	.850
	Bayesian net	[Lam et al. 1997]	.542 (MF_1)				

Rocchio results…

Schapire, Singer, Singhal, "Boosting and Rocchio Applied to Text Filtering", SIGIR 98



Reuters 21578 – all classes (not just the frequent ones)

A hidden agenda

- Part of machine learning is good grasp of theory
- Part of ML is a good grasp of what hacks tend to work
- These are not always the same
  - Especially in big-data situations

- Catalog of useful tricks so far
  - Brute-force estimation of a joint distribution
  - Naive Bayes
  - Stream-and-sort, request-and-answer patterns
  - BLRT and KL-divergence (and when to use them)
  - **TF-IDF weighting – especially IDF**
    - it's often useful even when we don't understand why

One more Rocchio observation

Rennie et al, ICML 2003, "Tackling the Poor Assumptions of Naïve Bayes Text Classifiers"

	MNB	TWCNB	SVM
Industry Sector	0.582	0.923	0.934
20 Newsgroups	0.848	0.861	0.862
Reuters (micro)	0.739	0.844	0.887
Reuters (macro)	0.270	0.647	0.694

NB + cascade of hacks

One more Rocchio observation

Rennie et al, ICML 2003, "Tackling the Poor Assumptions of Naïve Bayes Text Classifiers"

- TWCNB($\vec{d}, \vec{y}$)

1. $d_{ij} = \log(d_{ij} + 1)$ (TF transform § 4.1)
2. $d_{ij} = d_{ij} \log \frac{\sum_k 1}{\sum_k \delta_{ik}}$ (IDF transform § 4.2)
3. $d_{ij} = \frac{d_{ij}}{\sqrt{\sum_k (d_{kj})^2}}$ (length norm. § 4.3)
4. $\hat{\theta}_{ci} = \frac{\sum_{j:y_j \neq c} d_{ij} + \alpha_i}{\sum_{j:y_j \neq c} \sum_k d_{kj} + \alpha}$ (complement § 3.1)
5. $w_{ci} = \log \hat{\theta}_{ci}$
6. $w_{ci} = \frac{w_{ci}}{\sum_i w_{ci}}$ (weight normalization § 3.2)
7. Let $t = (t_1, \ldots, t_n)$ be a test document; let t_i be the count of word i.
8. Label the document according to

$$l(t) = \arg\min_c \sum_i t_i w_{ci}$$

"In tests, we found the length normalization to be most useful, followed by the log transform… these transforms were also applied to the input of SVM".