Hashing for Machine Learning

Hashing Summer School, University of Copenhagen

To follow along:
git clone git://github.com/JohnLangford/vowpal_wabbit.git
wget http://hunch.net/~jl/rcv1.tar.gz

John Langford, Microsoft Resarch, NYC

July 17, 2014

Simple Machine Learning: Linear Learning

Features: a vector $x \in \mathbb{R}^n$
Label: $y \in \mathbb{R}$
Goal: Learn $w \in \mathbb{R}^n$ such that $\hat{y}_w(x) = \sum_i w_i x_i$ is close to y.

An Example: The RCV1 dataset

Pick whether a document is in category CCAT or not.

Dataset size:

781K examples

60M nonzero features

1.1G bytes

Format: label | sparse features ...

An Example: The RCV1 dataset

Pick whether a document is in category CCAT or not.

Dataset size:

781K examples

60M nonzero features

1.1G bytes

Format: label | sparse features ...

1 | 13:3.9656971e-02 24:3.4781646e-02 ...

An Example: The RCV1 dataset

Pick whether a document is in category CCAT or not.

Dataset size:

781K examples

60M nonzero features

1.1G bytes

Format: label | sparse features ...

1 | 13:3.9656971e-02 24:3.4781646e-02 ...

which corresponds to:

1 | tuesday year ...

with “Term Frequency Inverse Document Frequency” feature values.

An Example: The RCV1 dataset

Pick whether a document is in category CCAT or not.
Dataset size:
781K examples
60M nonzero features
1.1G bytes
Format: label | sparse features ...
1 | 13:3.9656971e-02 24:3.4781646e-02 ...
which corresponds to:
1 | tuesday year ...
with "Term Frequency Inverse Document Frequency" feature values.

run: vw rcv1.train.txt -c –binary
Progressive validation loss 0.0513 ($\simeq$ best) in 1 second. (best)

An Example: The RCV1 dataset

Pick whether a document is in category CCAT or not.
Dataset size:
781K examples
60M nonzero features
1.1G bytes
Format: label | sparse features ...
1 | 13:3.9656971e-02 24:3.4781646e-02 ...
which corresponds to:
1 | tuesday year ...
with "Term Frequency Inverse Document Frequency" feature values.

run: vw rcv1.train.txt -c –binary
Progressive validation loss 0.0513 ($\simeq$ best) in 1 second. (best)

Performance real, but timing not: TFIDF transform >2 minutes.

The unconventional story: Hashing on RCV1

Start with the raw data:

1 | tuesday year million short compan vehicl line stat financ commit exchang plan corp subsid credit issu debt pay gold bureau prelimin refin billion telephon time draw

...

The unconventional story: Hashing on RCV1

Start with the raw data:

1 | tuesday year million short compan vehicl line stat financ commit exchang plan corp subsid credit issu debt pay gold bureau prelimin refin billion telephon time draw

...

```
vw -b 24 -c rcv1.train.raw.txt --binary
```

Progressive validation loss 0.0572 in 1.3s

The unconventional story: Hashing on RCV1

Start with the raw data:

1 | tuesday year million short compan vehicl line stat financ commit exchang plan corp subsid credit issu debt pay gold bureau prelimin refin billion telephon time draw

...

vw -b 24 -c rcv1.train.raw.txt --binary

Progressive validation loss 0.0572 in 1.3s

vw -b 24 -c rcv1.train.raw.txt --binary --ngram 2

Progressive validation loss 0.0500 in 3.4s

The unconventional story: Hashing on RCV1

Start with the raw data:

1 | tuesday year million short compan vehicl line stat financ commit exchang plan corp subsid credit issu debt pay gold bureau prelimin refin billion telephon time draw

...

vw -b 24 -c rcv1.train.raw.txt --binary

Progressive validation loss 0.0572 in 1.3s

vw -b 24 -c rcv1.train.raw.txt --binary --ngram 2

Progressive validation loss 0.0500 in 3.4s

vw -b 24 -c rcv1.train.raw.txt --binary --ngram 2 --skips 4

Progressive validation loss 0.0454 in 11s

The unconventional story: Hashing on RCV1

Start with the raw data:

1 | tuesday year million short compan vehicl line stat financ commit exchang plan corp subsid credit issu debt pay gold bureau prelimin refin billion telephon time draw

...

vw -b 24 -c rcv1.train.raw.txt --binary

Progressive validation loss 0.0572 in 1.3s

vw -b 24 -c rcv1.train.raw.txt --binary --ngram 2

Progressive validation loss 0.0500 in 3.4s

vw -b 24 -c rcv1.train.raw.txt --binary --ngram 2 --skips 4

Progressive validation loss 0.0454 in 11s

vw -b 24 -c rcv1.train.raw.txt --binary --ngram 2 --skips 4 -l 0.25

Progressive validation loss 0.0449 in 11s

+Better error rate

+1/10th traing time!

+faster/easier testing

Outline

1. A bit more about machine Learning
2. Use of Hash
3. Hashing applications

Algorithm used: Online Linear Learning

Start with $\forall i : \quad w_i = 0$
Repeatedly:

1. Get features $x \in \mathbb{R}^n$.
2. Make linear prediction $\hat{y}_w(x) = \sum_i w_i x_i$.
3. Observe label y.
4. Record loss $\ell(y, \hat{y}_w(x))$.
5. Update weights so $\hat{y}_w(x)$ is closer to y.

Algorithm used: Online Linear Learning

Start with $\forall i: \quad w_i = 0$

Repeatedly:

1. Get features $x \in \mathbb{R}^n$.
2. Make linear prediction $\hat{y}_w(x) = \sum_i w_i x_i$.
3. Observe label y.
4. Record loss $\ell(y, \hat{y}_w(x))$.
5. Update weights so $\hat{y}_w(x)$ is closer to y.
   Example: $w_i \leftarrow w_i + \eta(y - \hat{y})x_i$.

Algorithm used: Online Linear Learning

Start with $\forall i : \quad w_i = 0$
Repeatedly:

1. Get features $x \in \mathbb{R}^n$.
2. Make linear prediction $\hat{y}_w(x) = \sum_i w_i x_i$.
3. Observe label y.
4. Record loss $\ell(y, \hat{y}_w(x))$.
5. Update weights so $\hat{y}_w(x)$ is closer to y.
   Example: $w_i \leftarrow w_i + \eta(y - \hat{y})x_i$.

(many more details that matter in practice)

Features in Practice: Engineered Features

Hand crafted features, built up iteratively over time, each new feature fixing a discovered problem.

1. +Good understanding of what's happening.
2. +Never fail to learn the obvious.
3. +Small RAM usage.
4. -Slow at test time. Intuitive features for humans can be hard
5. -Low Capacity. A poor fit for large datasets. (Boosted) Decision trees are a good compensation on smaller datasets.
6. -High persontime.

Features in Practice: Learned Features

Use a nonlinear/nonconvex possibly deep learning algorithm.

1. +Good results in Speech & Vision.
2. +Fast at test time.
3. +High capacity. Useful on large datasets.
4. -Slow training. Days to weeks are common.
5. -Wizardry may be required.

Features in Practice: sparse words

Generate a feature for every word.

1. +High capacity.
2. +fast test time. Lookup some numbers, then compute an easy prediction.
3. +fast learning Linear faster than decision tree, but parallel is tricky.
4. -High RAM usage

Features in Practice: sparse words

Generate a feature for every word.

1. +High capacity.
2. +fast test time. Lookup some numbers, then compute an easy prediction. This lecture.
3. +fast learning Linear faster than decision tree, but parallel is tricky. This lecture.
4. -High RAM usage This lecture.

Outline

1. A bit more about machine Learning
2. Use of Hash
3. Hashing applications

What is hashing?

Hash function: string $\rightarrow \{0, 1\}^b$

What is hashing?

Hash function: string $\rightarrow \{0, 1\}^b$

Hash table = Hash function + Array< Pair<string, int> > of length $\{0, 1\}^b$

What is hashing?

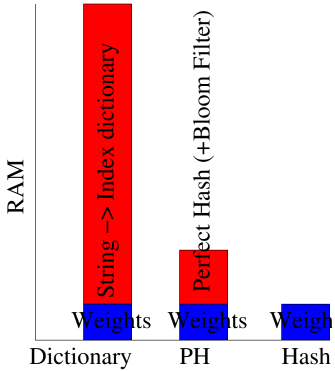
Hash function: string $\rightarrow \{0,1\}^b$

Hash table $=$ Hash function $+$ Array$<$ Pair$<$string, int$>$ $>$ of length $\{0,1\}^b$

Perfect hash $=$ mapping of n fixed (and known in advance) strings to integers $\{1,n\}$.

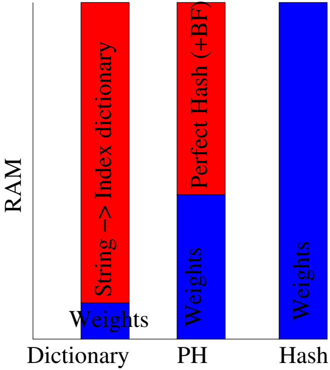
How does feature address parameter?

1. Hash Table (aka Dictionary): Store hash function + Every string + Index.
2. Perfect Hash (+Bloom Filter): Store Custom Hash function (+ bit array).
3. Hash function: Store Hash function.



How does feature address parameter?

1. Hash Table (aka Dictionary): Store hash function + Every string + Index.
2. Perfect Hash (+Bloom Filter): Store Custom Hash function (+ bit array).
3. Hash function: Store Hash function.



More weights is better!

A variation: unbiased hashing

Multiply feature value by $(-1)^s$ where s is a 1 bit hash.

Advantage: Feature values have expectation 0 for a random s $\Rightarrow$ better performance in the high collision regime.

Objection: Collisions!

Valid sometimes: particularly with low dimensional hand engineered features.

Objection: Collisions!

Valid sometimes: particularly with low dimensional hand engineered features.

Theorem: If a feature is duplicated $O(\log n)$ times when there are $O(n)$ features, at least one version of the feature is uncollided when hashing with $\log(n \log n)$ bits.
Proof: Essentially Bloom filter logic. See Michael Mitzenmacher's talk Monday.

Objection: Incomprehensible!

Objection: Incomprehensible!

Use --audit to decode.

Keep your own dictionary on the side --invert_hash if needed.

Outline

1. A bit more about machine Learning
2. Use of Hash
3. Hashing applications

Use of Hash: Ngrams

2-gram = a feature for every pair of adjacent words.
3-gram = a feature for every triple of adjacent words, etc...
n-gram = ...

Use of Hash: Ngrams

2-gram = a feature for every pair of adjacent words.
3-gram = a feature for every triple of adjacent words, etc...
n-gram = ...

Input:
$(\text{index}_1, \text{value}_1)$
$(\text{index}_2, \text{value}_2)$
Output:

Use of Hash: Ngrams

2-gram = a feature for every pair of adjacent words.
3-gram = a feature for every triple of adjacent words, etc...
n-gram = ...

Input:
$(\text{index}_1, \text{value}_1)$
$(\text{index}_2, \text{value}_2)$
Output:
$((\text{index}_1 \text{magic} + \text{index}_2)\&\text{mask}, \text{value}_1 \text{value}_2)$
(linear hash, value multiplication)

Use of Hash: Outer Products

Input:
Feature sets F_1, F_2
Output:
Outer product set $F_1 \times F_2$
Use linear hash again.

Use of Hash: Outer Products

Input:
Feature sets F_1, F_2
Output:
Outer product set $F_1 \times F_2$
Use linear hash again.
Implemented via -q in VW.

Example 2: Mass Personalized Spam Filtering

1. $3.2 * 10^6$ labeled emails.
2. 433167 users.
3. $\sim 40 * 10^6$ unique tokens.

How do we construct a spam filter which is personalized, yet uses global information?

Example 2: Mass Personalized Spam Filtering

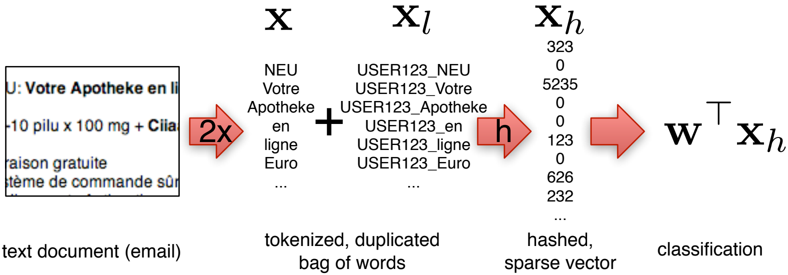
1. $3.2 * 10^6$ labeled emails.
2. 433167 users.
3. $\sim 40 * 10^6$ unique tokens.

How do we construct a spam filter which is personalized, yet uses global information?

Bad answer: Construct a global filter + 433167 personalized filters using a conventional hashmap to specify features. This might require $433167 * 40 * 10^6 * 4 \sim 70$Terabytes of RAM.

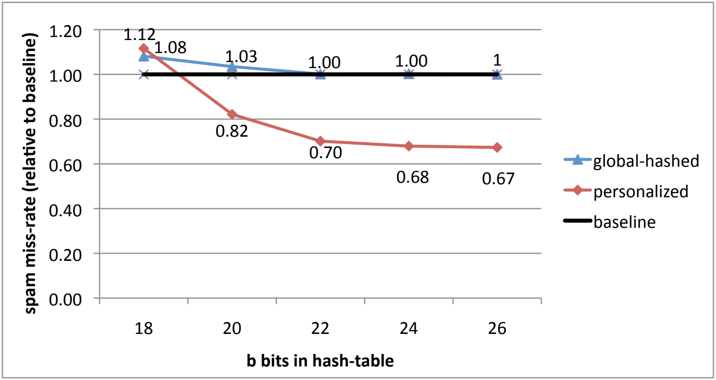
Using Hashing

Use hashing to predict according to: $\langle w, \phi(x) \rangle + \langle w, \phi_u(x) \rangle$



(in VW: specify the userid as a feature and use -q)

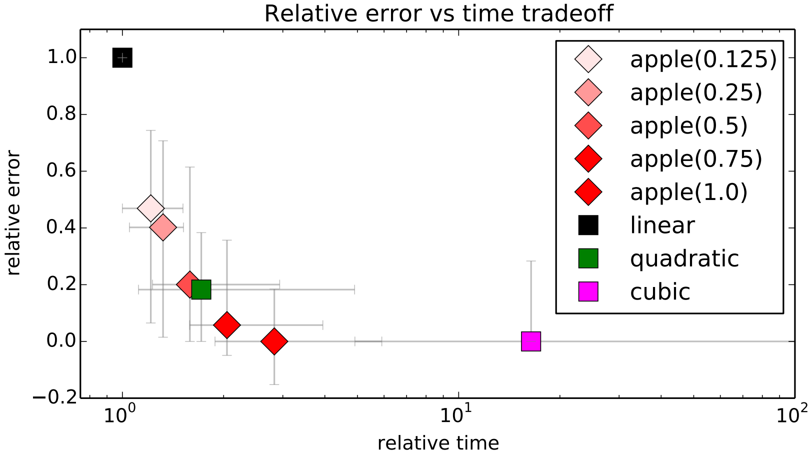
Results



2^{26} parameters $=$ 64M parameters $=$ 256MB of RAM.

An x270K savings in RAM requirements.

Another Application: Efficient Sparse Polynomial learning



Features sometimes collide, which is scary, but then you love it

Generate a feature for every word, ngram, skipgram, pair of (ad word, query word), etc... and use high dimensional representation.

1. +High capacity.
2. +Correlation effects nailed.
3. +Fast test time. Compute an easy prediction.
4. +Fast Learning (with Online + parallel techniques. See talks.)
5. +/-Variable RAM usage. Highly problem dependent but fully controlled.

Another cool observation: Online learning + Hashing = learning algorithm with fully controlled memory footprint $\Rightarrow$ Robustness.

References, prequels

1. Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, MIT Press, Cambridge, MA, 1998. Chapter 8.3.1 hashes states.
2. CRM114 http://crm114.sourceforge.net/, 2002. Uses hashing of grams for spam detection.
3. Apparently used by others as well, internally.
4. Many use hashtables which store the original item or a 64+ bit hash of the original item.

References, “modern” hashing trick

1. 2007, Langford, Li, Strehl, Vowpal Wabbit released.
2. 2008, Ganchev & Dredze, ACL workshop: A hash function is as good as a hashmap empirically.
3. 2008/2009, VW Reimplementation/Reimagination/Integration in Stream (James Patterson & Alex Smola) and Torch (Jason Weston, Olivier Chapelle, Kilian).
4. 2009, AIStat Qinfeng Shi et al, Hash kernel definition, Asymptopia Redundancy analysis
5. 2009, ICML Kilian et al, Unbiased Hash Kernel, Length Deviation Bound, Mass Personalization Example and Multiuse Bound.

Question 1

Hashing is applied to features before learning. Is it better to apply to parameters after learning?

Question 2

A commonly proposed alternative to hashing is random projection as per Johnson-Lindenstrauss (see Alex Andoni's lecture yesterday). Is that a better approach?

Question 3

How do you compute the n-grams for k elements in time $O(k)$?