Announcements: projects

- 805 students: Project proposals are due **Sun 10/1.** If you'd like to work with 605 students then indicate this on your proposal.

- 605 students: the week after 10/1 I will post the proposals on the wiki and you will have time to contact 805 students and join teams.

- 805 students: let me know your team by 10/8: I will approve the teams.

The course so far

- Doing i/o bound tasks in parallel
  - map-reduce
  - dataflow languages: PIG, Spark, GuineaPig
- A little bit about iteration and parallelism
- Next up:
  - this years favorite big ML trick!
    - don't be disappointed…
  - step back and look at perceptrons and fancy ML tricks: margins, ranking, kernels, ….
  - step forward and talk about parallel learning
  - midterm

LEARNING AS OPTIMIZATION: MOTIVATION

Learning as optimization: warmup

Goal: Learn the parameter θ of a binomial

Dataset: $D=\{x_1,...,x_n\}$, x_i is 0 or 1, k of them are 1

$$P(X = x_i | \theta) = \theta^{x_i}(1-\theta)^{1-x_i}$$

➜ $P(D/\theta) = c\, \theta^k(1-\theta)^{n-k}$

➜ $d/d\theta\ P(D/\theta) = k\theta^{k-1}(1-\theta)^{n-k} + \theta^k(n-k)(1-\theta)^{n-k-1}$

$$\frac{\partial}{\partial \theta}P(D) = \theta^{k-1}(1-\theta)^{n-k-1}\left(k(1-\theta) - \theta(n-k)\right)$$

Learning as optimization: warmup

Goal: Learn the parameter θ of a binomial

Dataset: $D=\{x_1,...,x_n\}$, x_i is 0 or 1, k of them are 1

$$\frac{\partial}{\partial\theta}P(D) = \theta^{k-1}(1-\theta)^{n-k-1}\left(k(1-\theta)-\theta(n-k)\right) = 0$$

θ= 0

θ= 1

$k- k\theta - n\theta + k\theta = 0$
➜ $n\theta = k$
➜ $\theta = k/n$

Learning as optimization: general procedure

- Goal: Learn parameter θ (or weight vector **w**)

- Dataset: $D=\{(x_1,y_1)...,(x_n,y_n)\}$

- Write down loss function: how well **w** fits the data D as a function of **w**

  – Common choice: $\log \Pr(D|\mathbf{w})$

- Maximize by differentiating

  – Then **gradient descent**: repeatedly take a small step in the direction of the gradient

Learning as optimization: general procedure for **SGD** (stochastic gradient descent)

- **Big-data** problem: we *don't* want to load all the data *D* into memory, and the gradient depends on all the data
- **Solution:**
  - pick a small subset of examples B<<D
  - **approximate** the gradient using them
    - "on average" this is the right direction
  - take a step in that direction
  - repeat….
- Math: find gradient of **w** for a *single* example, not a dataset

B = **one** example is a very popular choice

SGD vs streaming

- Streaming:
  - pass through the data *once*
  - hold model + one example in memory
  - update model for each example
- Stochastic gradient:
  - pass through the data *multiple times*
    - stream through a disk file repeatedly
  - hold model + B examples in memory
  - update model *via gradient step*
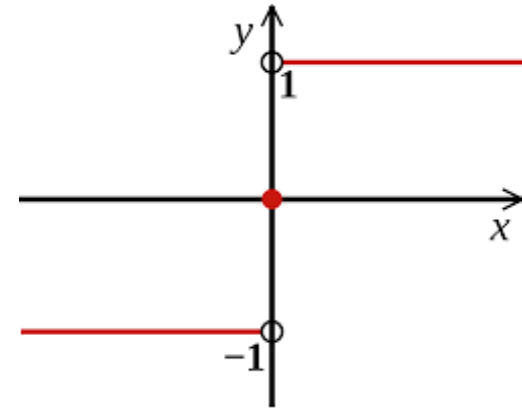
B = **one** example is a very popular choice

its simple ☺

sometimes its cheaper to evaluate 100 examples at once than one example 100 times ☹

Logistic Regression vs Rocchio

- Rocchio looks like:

$$f(d) = \operatorname{argmax}_y \mathbf{v}(d) \cdot \mathbf{v}(y)$$
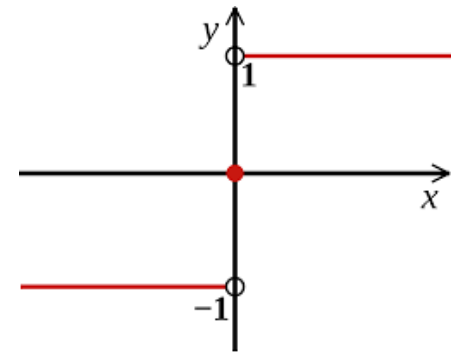
- Two classes, *y=+1* or *y=-1*:

$$f(d) = \operatorname{sign}\big([\mathbf{v}(d) \cdot \mathbf{v}(+1)] - [\mathbf{v}(d) \cdot \mathbf{v}(-1)]\big)$$

$$= \operatorname{sign}\big(\mathbf{v}(d) \cdot [\mathbf{v}(+1) - \mathbf{v}(-1)]\big)$$

$$f(\mathbf{x}) = \operatorname{sign}(\mathbf{x} \cdot \mathbf{w}) \qquad \begin{array}{l} \boldsymbol{w = v(+1) - v(-1)} \\ \boldsymbol{x = v(d)} \end{array}$$

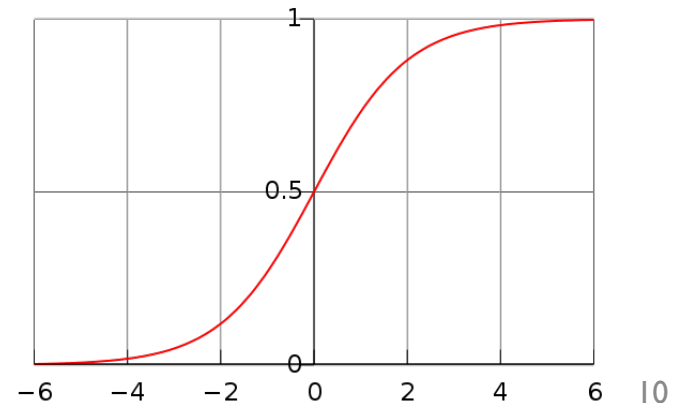# Logistic Regression vs Naïve Bayes

- Naïve Bayes for two classes can also be written as:

$$f(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{w})$$

- Since we can't differentiate $\text{sign}(x)$, a convenient variant is a logistic function:

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

Efficient Logistic Regression with Stochastic Gradient Descent

William Cohen

Learning as optimization for logistic regression

- Goal: Learn the parameter **w** of the classifier

$$P(Y = y | X = \mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{x} \cdot \mathbf{w}}}$$

- Probability of a single example P(y|**x,**w) would be

$$P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{1+e^{-\mathbf{x} \cdot \mathbf{w}}} & \text{if } y = 1 \\ 1 - \frac{1}{1+e^{-\mathbf{x} \cdot \mathbf{w}}} & \text{if } y = 0 \end{cases}$$

- Or with logs:

$$\log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases}$$

$$p \equiv \frac{1}{1 + e^{-\mathbf{x} \cdot \mathbf{w}}} = \frac{1}{1 + \exp(-\sum_j x^j w^j)}$$

$$\log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases}$$

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{p} \boxed{\frac{\partial}{\partial w^j} p} & \text{if } y = 1 \\ \frac{1}{1-p}(-\frac{\partial}{\partial w^j} p) & \text{if } y = 0 \end{cases}$$

$$(\log f)' = \frac{1}{f} f'$$

$$p \equiv \frac{1}{1 + e^{-\mathbf{x} \cdot \mathbf{w}}} = \frac{1}{1 + \exp(-\sum_j x^j w^j)}$$

$$p \equiv \frac{1}{1 + e^{-\mathbf{x} \cdot \mathbf{w}}} = \frac{1}{1 + \exp(-\sum_j x^j w^j)}$$

$$1 - p = \frac{1 + \exp(-\sum_j x^j w^j)}{1 + \exp(-\sum_j x^j w^j)} - \frac{1}{1 + \exp(-\sum_j x^j w^j)} = \boxed{\frac{\exp(-\sum_j x^j w^j)}{1 + \exp(-\sum_j x^j w^j)}}$$
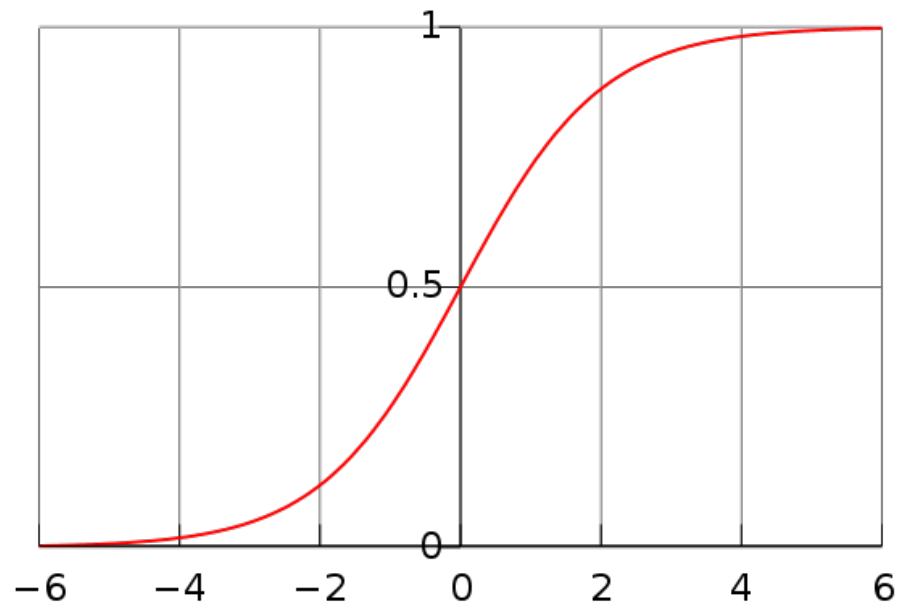
$$\boxed{\frac{\partial}{\partial w^j} p} = \frac{\partial}{\partial w^j} \left(1 + \exp(-\sum_j x^j w^j)\right)^{-1} \qquad (e^f)' = e^f f'$$

$$= (-1)\left(1 + \exp(-\sum_j x^j w^j)\right)^{-2} \frac{\partial}{\partial w^j} \exp(-\sum_j x^j w^j)$$

$$= (-1)\left(1 + \exp(-\sum_j x^j w^j)\right)^{-2} \exp(-\sum_j x^j w^j)(-x^j)$$

$$= \underset{\text{P}}{\boxed{\frac{1}{1 + \exp(-\sum_j x^j w^j)}}} \boxed{\frac{\exp(-\sum_j x^j w^j)}{1 + \exp(-\sum_j x^j w^j)}} x^j$$

$$\frac{\partial}{\partial w^j} p = p(1 - p)x^j$$

14

$$\frac{\partial}{\partial w^j} p = p(1-p)x^j$$

$$\log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases}$$

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{p}\boxed{\frac{\partial}{\partial w^j} p} & \text{if } y = 1 \\ \frac{1}{1-p}(-\frac{\partial}{\partial w^j} p) & \text{if } y = 0 \end{cases}$$

$$\frac{\partial}{\partial w^j} p = p(1 - p)x^j$$

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \frac{1}{p}p(1 - p)x^j = (1 - p)x^j & \text{if } y = 1 \\ \frac{1}{1-p}(-1)p(1 - p)x^j = -px^j & \text{if } y = 0 \end{cases}$$

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = (y - p)x^j$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda(y - p)\mathbf{x}$$

update with learning rate lambda

16

Again: Logistic regression

- Start with Rocchio-like linear classifier:

$$\hat{y} = sign(\mathbf{x} \cdot \mathbf{w})$$

- Replace sign(...) with something differentiable:
  - Also scale from 0-1 not -1 to +1

$$\hat{y} = \sigma(\mathbf{x} \cdot \mathbf{w}) = p$$

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$
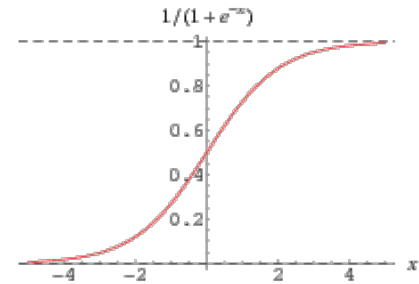
- Define a loss function:

$$L(\mathbf{w} \mid y, \mathbf{x}) = \begin{cases} \log \sigma(\mathbf{w} \cdot \mathbf{x}) & y = 1 \\ \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x})) & y = 0 \end{cases}$$

- Differentiate....

$$= \log\left(\sigma(\mathbf{w} \cdot \mathbf{x})^y (1 - \sigma(\mathbf{w} \cdot \mathbf{x}))^{1-y}\right)$$

$$\log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases}$$

$$p = \sigma(\mathbf{x} \cdot \mathbf{w})$$

Magically, when we differentiate, we end up with something very simple and elegant…..

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w} | y, \mathbf{x}) = (y - p)\mathbf{x}$$

$$\frac{\partial}{\partial w^j} L(\mathbf{w} | y, \mathbf{x}) = (y - p)x^j$$

The update for gradient descent is just:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda(y - p)\mathbf{x}$$

Logistic regression has a sparse update

An observation: sparsity!

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = (y - p)x^j$$

Key computational point:

- if $x^j = 0$ then the gradient of w^j is zero
- so when processing an example you only need to update weights for the **non-zero** features of an example.

Learning as optimization for logistic regression

- The algorithm: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda(y - p)\mathbf{x}$

1. Initialize a hashtable W

2. For $t = 1, \ldots, T$

   - For each example $\mathbf{x}_i, y_i$:   *-- do this in random order*
     - Compute the prediction for $\mathbf{x}_i$:

$$p_i = \frac{1}{1 + \exp(-\sum_{j:x_i^j > 0} x_i^j w^j)}$$

     - For each non-zero feature of $\mathbf{x_i}$ with index j and value x^j:
       * If j is not in W, set $W[j] = 0$.
       * Set $W[j] = W[j] + \lambda(y_i - p_i)x^j$

3. Output the hash table W.

Another observation

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = (y - p)x^j$$

- Consider averaging the gradient over all the examples $D=\{(x_1, y_1)..., (x_n, y_n)\}$

$$\frac{\partial}{\partial w^j} \log P(D | \mathbf{w}) = \frac{1}{n} \sum_i (y_i - p_i)x_i^j = \boxed{\frac{1}{n} \sum_{i:x_i^j=1} y_i} - \boxed{\frac{1}{n} \sum_{i:x_i^j=1} p_i}$$

- This will overfit badly with sparse features
  - Consider any word that appears only in positive examples!

Learning as optimization for logistic regression

$$P(Y = y | X = \mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{\mathbf{x} \cdot \mathbf{w}}}$$

- Practical problem: this overfits badly with sparse features
  - e.g., if w^j is only in positive examples, its gradient is **always** positive !

$$\frac{\partial}{\partial w^j} \log P(D | \mathbf{w}) = \frac{1}{n} \sum_i (y_i - p_i) x_i^j = \frac{1}{n} \sum_{i:x_i^j=1} y_i - \frac{1}{n} \sum_{i:x_i^j=1} p_i$$

REGULARIZED LOGISTIC REGRESSION

Regularized logistic regression

- Replace LCL

$$\log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases}$$

- with LCL + penalty for large weights, eg

$$LCL - \mu \sum_{j=1}^{d} (w^j)^2$$

- So:

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = (y - p)x^j$$

- becomes:

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) - \mu \sum_{j=1}^{d} (w^j)^2 = (y - p)x^j - 2\mu w^j$$

Regularized logistic regression

- Replace LCL

$$\log P(Y = y | X = \mathbf{x}, \mathbf{w}) = \begin{cases} \log p & \text{if } y = 1 \\ \log(1 - p) & \text{if } y = 0 \end{cases}$$

- with LCL + penalty for large weights, eg

$$LCL - \mu \sum_{j=1}^{d} (w^j)^2$$

- So the update for wj becomes:

$$w^j = w^j + \lambda((y - p)x^j - 2\mu w^j)$$

- Or

$$w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$$

Learning as optimization for logistic regression

- Algorithm: $$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda(y - p)\mathbf{x}$$

1. Initialize a hashtable W

2. For $t = 1, \ldots, T$

   - For each example $\mathbf{x}_i, y_i$:   *-- do this in random order*

     - Compute the prediction for $\mathbf{x}_i$:

       $$p_i = \frac{1}{1 + \exp(-\sum_{j:x_i^j > 0} x_i^j w^j)}$$

     - For each non-zero feature of $\mathbf{x_i}$ with index j and value x^j:

       * If j is not in W, set $W[j] = 0$.
       * Set $W[j] = W[j] + \lambda(y_i - p_i)x^j$

3. Output the hash table W.

Learning as optimization for regularized logistic regression

- Algorithm: $\qquad w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$

1. Initialize a hashtable W

2. For $t = 1, \ldots, T$

   - For each example $\mathbf{x}_i, y_i$:

     − Compute the prediction for $\mathbf{x}_i$:

$$p_i = \frac{1}{1 + \exp(-\sum_{j:x_i^j > 0} x_i^j w^j)}$$

     − For each non-zero feature of $\mathbf{x_i}$ with index j and value x^j:

       ∗ If j is not in W, set $W[j] = 0$.
       ∗ Set $W[j] = W[j] \boxed{+ \lambda(y - p)x^j - \lambda 2\mu w^j}$

3. Output the hash table W.

Time goes from O(nT) to O(mVT) where
- n = number of non-zero entries,
- m = number of examples
- V = number of features
- T = number of passes over data

This change is very important for large datasets

- We've lost the ability to do *sparse* updates
- This makes learning *much much* more expensive
  - $2*10^6$ examples
  - $2*10^8$ non-zero entries
  - $2*10^6$ + features
  - 10,000x slower (!)

Time goes from $O(nT)$ to $O(mVT)$ where
- n = number of non-zero entries,
- m = number of examples
- V = number of features
- T = number of passes over data

SPARSE UPDATES FOR REGULARIZED LOGISTIC REGRESSION

Learning as optimization for regularized logistic regression

- Final algorithm: $$w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$$

- Initialize hashtable *W*

- For each iteration t=1,…T
  - For each example $(\mathbf{x}_i, y_i)$
    - $p_i = \ldots$
    - For each feature *W[j]*
      - $W[j] = W[j] - \lambda 2\mu W[j]$
      - If $x_i^j > 0$ then
        - » $W[j] = W[j] + \lambda(y_i - p^i)x_j$

Learning as optimization for regularized logistic regression

- Final algorithm: $w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$

- Initialize hashtable W

- For each iteration t=1,...T

  – For each example $(\mathbf{x}_i, y_i)$

    - $p_i = \ldots$

    - For each feature $W[j]$

      – $W[j]\ \ *= (1\ -\ \lambda 2\mu)$

      – If $x_i^j > 0$ then

        » $W[j] =\ W[j]\ + \lambda(y_i - p^i)x_j$

Learning as optimization for regularized logistic regression

- Final algorithm:
  $$w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$$

- Initialize hashtable W

- For each iteration t=1,...T
  - For each example $(\mathbf{x}_i, y_i)$
    - $p_i = \ldots$
    - For each feature $W[j]$
      - If $x_i^j > 0$ then
        » $W[j] \; *= (1 - \lambda 2\mu)^A$
        » $W[j] = W[j] + \lambda(y_i - p^i)x_j$

A is number of examples seen since the **last** time we did an **x>0 update** on $W[j]$

33

Learning as optimization for regularized logistic regression

- Final algorithm: $w^j = w^j + \lambda(y-p)x^j - \lambda 2\mu w^j$
- Initialize hashtables $W, \boxed{A}$ and set $\boxed{k=0}$
- For each iteration t=1,...T
  - For each example $(\mathbf{x}_i, y_i)$
    - $p_i = \ldots ; \boxed{k{+}{+}}$
    - For each feature $W[j]$
      - If $x_i^j > 0$ then
        - » $\boxed{W[j] \ {*}{=} \ (1 - \lambda 2\mu)^{k-A[j]}}$
        - » $W[j] = W[j] + \lambda(y_i - p^i)x_j$
        - » $\boxed{A[j] = k}$

> *k-A[j]* is number of examples seen since the **last** time we did an **x>0 update** on W[j]

34

Learning as optimization for regularized logistic regression

- Final algorithm: $$w^j = w^j + \lambda(y-p)x^j - \lambda 2\mu w^j$$

- Initialize hashtables $W, \boxed{A}$ and $\boxed{\text{set } k=0}$

- For each iteration t=1,...T
  - For each example $(\mathbf{x}_i, y_i)$
    - $p_i = \ldots ; \boxed{k\text{++}}$
    - For each feature $W[j]$
      - If $x_i^j > 0$ then
        - $\boxed{W[j] \;\;*= (1 \; - \; \lambda 2\mu)^{k-A[j]}}$
        - $W[j] = \; W[j] \; + \lambda(y_i - p^i)x_j$
        - $\boxed{A[j] = k}$

> - k = "clock" reading
> - A[j] = clock reading last time feature j was "active"
> - we implement the "weight decay" update using a "lazy" strategy: weights are decayed in one shot when a feature is "active"

Learning as optimization for regularized logistic regression

- Final algorithm: $w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$

- Initialize hashtables $W, \boxed{A}$ and set $\boxed{k=0}$

- For each iteration t=1,…T
  - For each example $(\mathbf{x}_i, y_i)$
    - $p_i = \ldots ; \boxed{k{+}{+}}$
    - For each feature $W[j]$
      - If $x_i^j > 0$ then
        » $\boxed{W[j] \ \ {*}{=} \ (1 \ - \ \lambda 2\mu)^{k-A[j]}}$
        » $W[j] = \ W[j] \ + \lambda(y_i - p^i)x_j$
        » $\boxed{A[j] = k}$

> Time goes from O(nT) to O(mVT) where
> - n = number of non-zero entries,
> - m = number of examples
> - V = number of features
> - T = number of passes over data
>
> Memory use doubles.

Comments

- What's happened here:
  - Our update involves a *sparse part* and a *dense part*
    - Sparse: empirical loss on this example
    - Dense: regularization loss – not affected by the example
  - We remove the *dense part* of the update
    - Old example update:
      - for each feature { do something example-independent}
      - For each active feature { do something example-dependent}
    - New example update:
      - For each active feature :
        - » {simulate the prior example-independent updates}
        - » {do something example-dependent}

Comments

- Same trick can be applied in other contexts
  - Other regularizers (eg L1, ...)
  - Conjugate gradient (Langford)
  - FTRL (Follow the regularized leader)
  - Voted perceptron averaging
  - ...?

BOUNDED-MEMORY LOGISTIC REGRESSION

Question

- In text classification most words are
  a. rare
  b. not correlated with any class
  c. given low weights in the LR classifier
  d. unlikely to affect classification
  e. not very interesting

Question

- In text classification most bigrams are

  a. rare

  b. not correlated with any class

  c. given low weights in the LR classifier

  d. unlikely to affect classification

  e. not very interesting

Question

- Most of the weights in a classifier are
  - important
  - not important

How can we exploit this?

- One idea: combine uncommon words together *randomly*
- Examples:
  - replace all occurrances of "humanitarianism" or "biopsy" with "humanitarianismOrBiopsy"
  - replace all occurrances of "schizoid" or "duchy" with "schizoidOrDuchy"
  - replace all occurrances of "gynecologist" or "constrictor" with "gynecologistOrConstrictor"
  - …
- For Naïve Bayes this breaks independence assumptions
  - it's not obviously a problem for logistic regression, though
- I could combine
  - two low-weight words (won't matter much)
  - a low-weight and a high-weight word (won't matter much)
  - two high-weight words (not very likely to happen)
- How much of this can I get away with?
  - certainly a little
  - is it enough to make a difference? how much memory does it save?

How can we exploit this?

- Another observation:
  - the values in my hash table are *weights*
  - the keys in my hash table are *strings* for the feature names
    - We need them to avoid collisions

- But maybe we don't care about collisions?
  - Allowing "schizoid" & "duchy" to collide is equivalent to replacing all occurrences of "schizoid" or "duchy" with "schizoidOrDuchy"

Learning as optimization for regularized logistic regression

- Algorithm: $$w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$$

- Initialize hashtables W, A and set $k=0$

- For each iteration t=1,…T

  – For each example $(\mathbf{x}_i, y_i)$

    - $p_i = \dots$; $k++$

    - For each feature $j: x_i^j > 0$:
      - » $W[j]$ *= $(1 - \lambda 2\mu)^{k-A[j]}$
      - » $W[j] = W[j] + \lambda(y_i - p^i)x_j$
      - » $A[j] = k$

Learning as optimization for regularized logistic regression

- Algorithm:

$$w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$$

- Initialize arrays W, A of size R and set $k=0$

- For each iteration t=1,...T

  - For each example $(\mathbf{x}_i, y_i)$

    - Let V be hash table so that $V[h] = \sum_{j:hash(x_i^j)\%R=h} x_i^j$

    - $p_i = ... ; k++$

    - For each hash value $h: V[h]>0$:

      » $W[h]$ *= $(1 - \lambda 2\mu)^{k-A[j]}$

      » $W[h] = W[h] + \lambda(y_i - p^i)V[h]$

      » $A[h] = k$

Learning as optimization for regularized logistic regression

- Algorithm:
  $$w^j = w^j + \lambda(y - p)x^j - \lambda 2\mu w^j$$

- Initialize arrays W, A of size R and set $k=0$

- For each iteration t=1,...T

  – For each example $(\mathbf{x}_i, y_i)$

    - Let V be hash table so that
      $$V[h] = \sum_{j:hash(j)\%R==h} x_i^j$$

    ??? • $p_i = ...$; $k++$

$$p \equiv \frac{1}{1 + e^{-\mathbf{x}\cdot\mathbf{w}}} \quad \Longrightarrow \quad p \equiv \frac{1}{1 + e^{-\mathbf{V}\cdot\mathbf{w}}}$$

IMPLEMENTATION DETAILS

Fixes and optimizations

- This is the basic idea but
  - we need to apply "weight decay" to features in an example before we compute the prediction
  - we need to apply "weight decay" before we save the learned classifier
  - my suggestion:
    - an abstraction for a logistic regression classifier

A possible SGD implementation

```
class SGDLogistic Regression {
    /** Predict using current weights **/
    double predict(Map features);
    /** Apply weight decay to a single feature and record when in A[ ]**/
    void regularize(string feature, int currentK);
    /** Regularize all features then save to disk **/
    void save(string fileName,int currentK);
    /** Load a saved classifier **/
    static SGDClassifier load(String fileName);
    /** Train on one example **/
    void train1(Map features, double trueLabel, int k) {
        // regularize each feature
        // predict and apply update
    }
}
// main 'train' program assumes a stream of randomly-ordered examples and
    outputs classifier to disk; main 'test' program prints predictions for each
    test case in input.
```

A possible SGD implementation

class SGDLogistic Regression {

...

}

// main 'train' program assumes a stream of **randomly-ordered** examples and outputs classifier to disk; main 'test' program prints predictions for each test case in input.

<100 lines (in python)

Other mains:

- A "shuffler:"
  - stream thru a training file T times and output instances
  - output is randomly ordered, as much as possible, given a buffer of size B
- Something to collect predictions + true labels and produce error rates, etc.

A possible SGD implementation

- Parameter settings:
  - $W[j]\ \ *= (1\ -\ \lambda 2\mu)^{k-A[j]}$
  - $W[j] =\ \ W[j]\ + \lambda(y_i - p^i)x_j$

- I didn't tune especially but used
  - $\mu = 0.1$
  - $\lambda = \eta * E^{-2}$ where E is "epoch", $\eta = \frac{1}{2}$
    - epoch: number of times you've iterated over the dataset, starting at E=1

Feature Hashing for Large Scale Multitask Learning

Kilian Weinberger   KILIAN@YAHOO-INC.COM
Anirban Dasgupta   ANIRBAN@YAHOO-INC.COM
John Langford   JL@HUNCH.NET
Alex Smola   ALEX@SMOLA.ORG
Josh Attenberg   JOSH@CIS.POLY.EDU

Yahoo! Research, 2821 Mission College Blvd., Santa Clara, CA 95051 USA

ICML 2009

An interesting example

- Spam filtering for Yahoo mail
  - Lots of examples and lots of users
  - Two options:
    - one filter for everyone—but users disagree
    - one filter for each user—but some users are lazy and don't label anything
  - Third option:
    - classify *(msg,user)* pairs
    - features of message i are words $w_{i,1},...,w_{i,ki}$
    - feature of user is his/her id u
    - features of **pair** are: $w_{i,1},...,w_{i,ki}$ and $u \bullet w_{i,1},...,u \bullet w_{i,ki}$
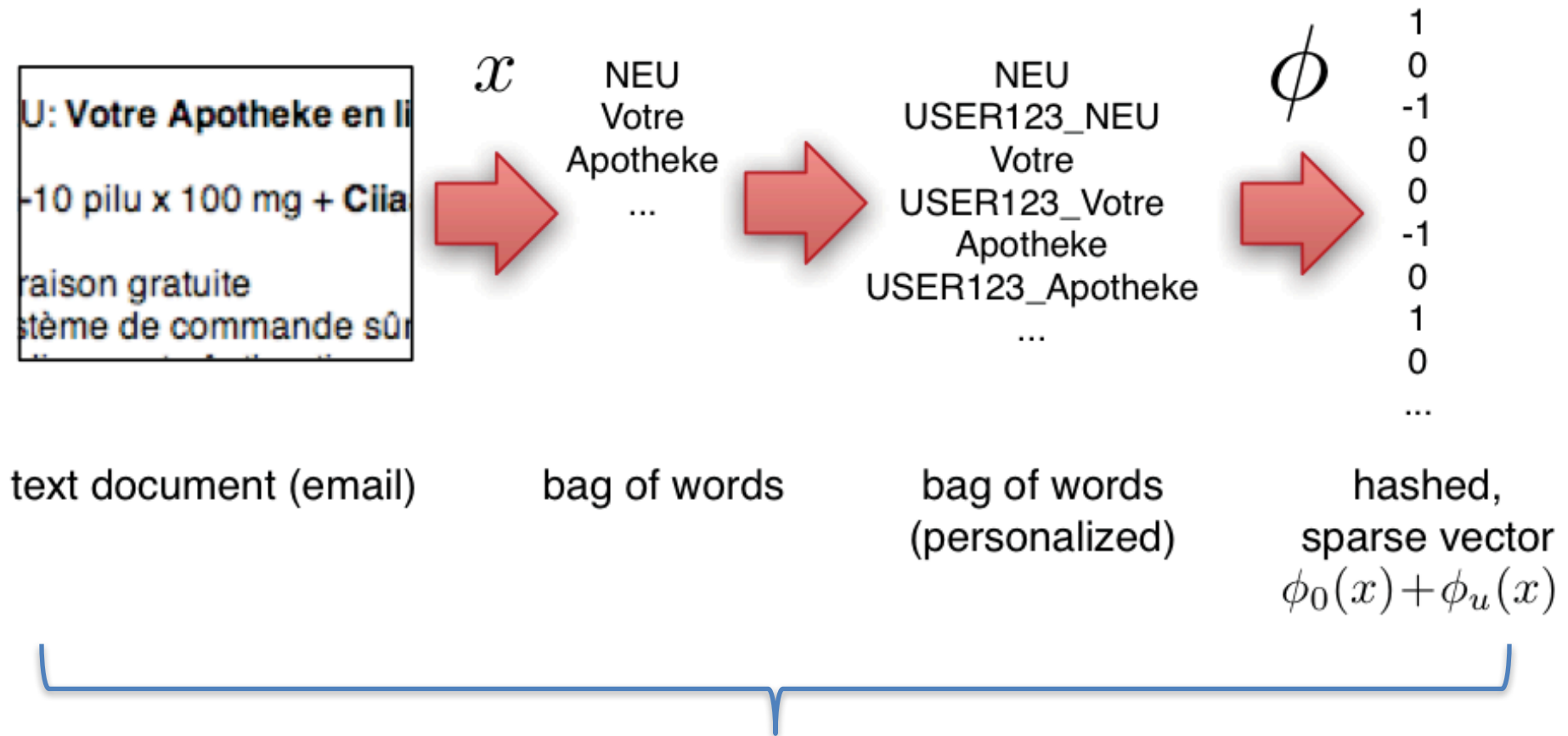   - based on an idea by Hal Daumé

An example

- E.g., this email to wcohen

> Dear Madam/Sir,
>
> My name is Mohammed Azziz an investment Broker with SouthCoast Plc a company based in London United Kingdom our major activity is in the area of managing customers funds with targetted interest rates through provision and acquisition of loans to interested borrowers with the basic requisite. Our periodic checks on people and Companies located

- features:
  - dear, madam, sir,…. investment, broker,…, wcohen•dear, wcohen•madam, wcohen,…,
- idea: the learner will figure out how to personalize my spam filter by using the wcohen•X features

An example

text document (email) → bag of words → bag of words (personalized) → hashed, sparse vector $\phi_0(x)+\phi_u(x)$

Compute personalized features and multiple hashes on-the-fly: a great opportunity to use several processors and speed up i/o

Experiments

- 3.2M emails
- 40M tokens
- 430k users
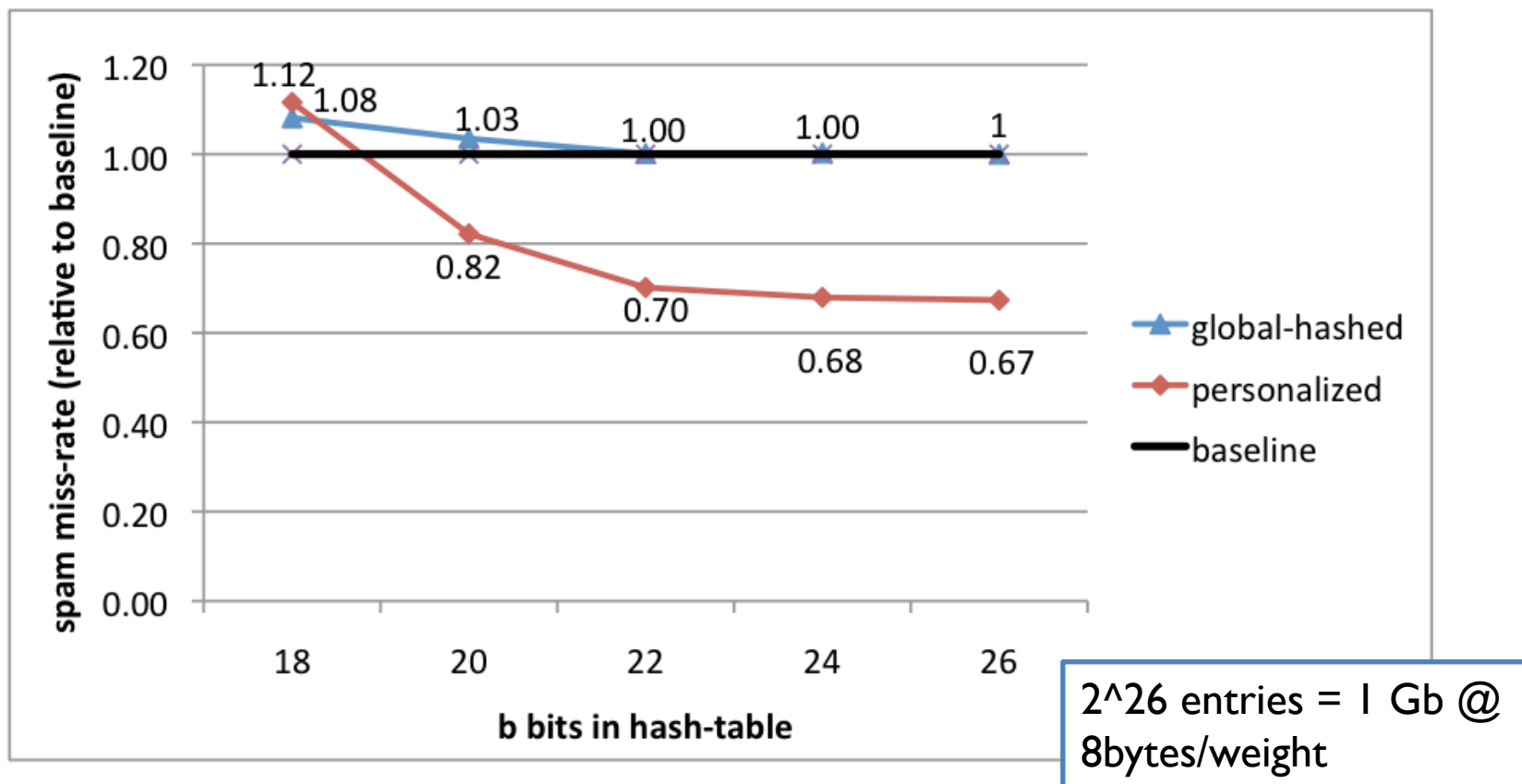- 16T unique features – after personalization

Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (*global-hashed*) converges relatively soon, showing that the distortion error ϵ_d vanishes. The personalized classifier results in an average improvement of up to 30%.
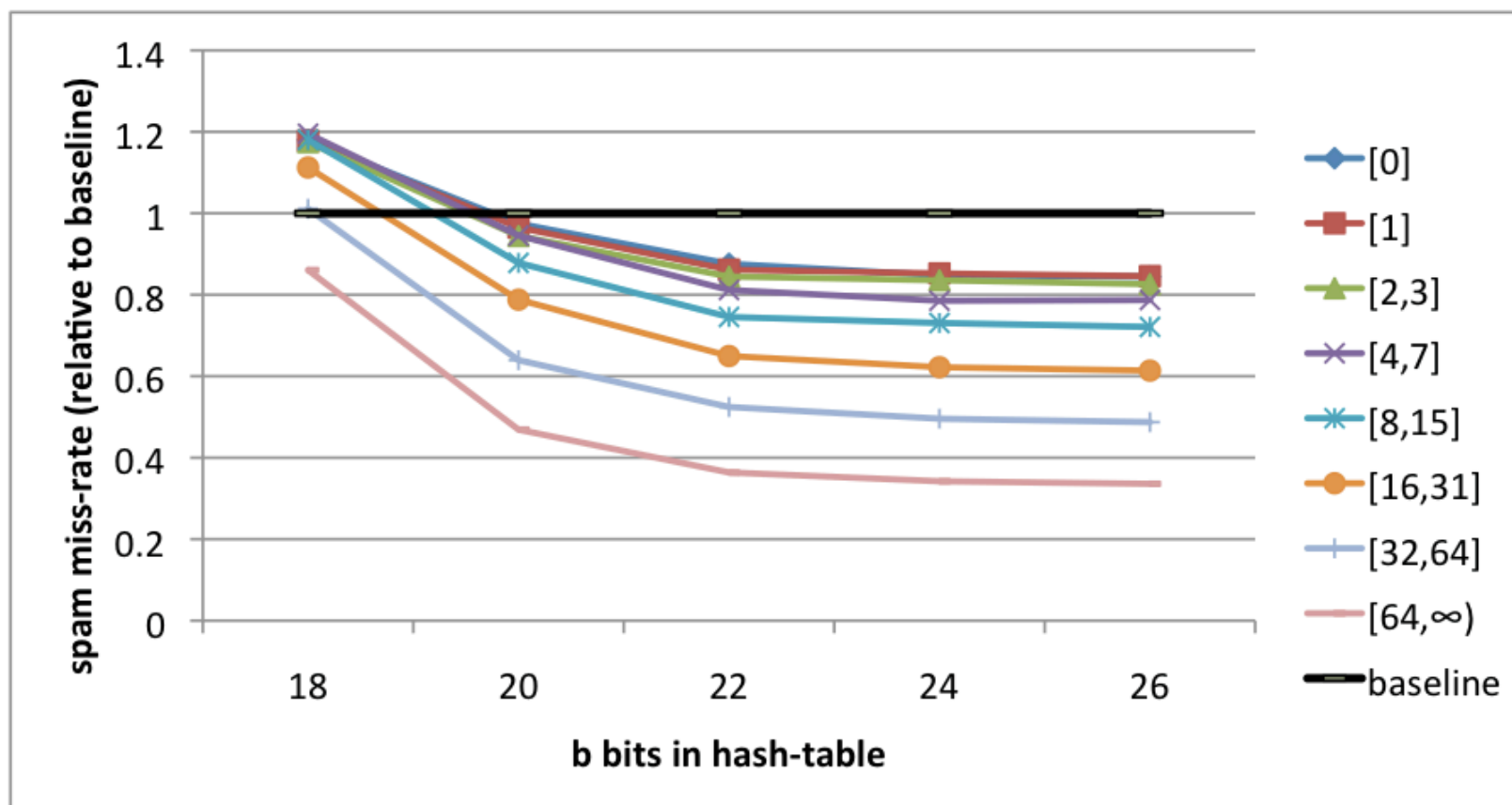
Figure 3. Results for users clustered by training emails. For example, the bucket [8, 15] consists of all users with eight to fifteen training emails. Although users in buckets with large amounts of training data do benefit more from the personalized classifier (up-to 65% reduction in spam), even users that did not contribute to the training corpus at all obtain almost 20% spam-reduction.