Proof-of-Concept Experiments

- **dataset #1:** 20-newsgroups
  - 3 (one vs. all) class. tasks for comp, sci, talk
  - 70% train / 30% test
  - feature dim.: 700K (all 2-grams)

- **dataset #2:** RCV1
  - hashes learned on 100K examples
  - Vowpal Wabbit ran on full train set
  - feature dim.: 40K

Classical Random Feature Hashing

Consider classification task (for simplicity):

- sparse high-dimensional labeled data: \((\phi, y_n) \in \mathbb{R}^D \times \{-1, +1\}, D \gg 1\)
- linear scoring \(f(\phi; w) = \langle w, \phi \rangle\)
- high dimension \(D\) \Rightarrow approximate separability

But we need to access \(w\) rapidly:
- must keep \(w\) in RAM \(\Rightarrow\) may not fit
- storage model matters:
  - (un)ordered associative tables need RAM
  - and/or are slow
- linear arrays are fastest \(\Rightarrow\) need integer feature indexes

Common solution:

- **feature hashing trick** (alphabet elimination/random feature mixing)
- Idea: per-coordinate mapping into a lower-dimension feature space
  - with data-independent pseudo-random function \(HASH:\)
    \[
    \phi'_d = \sum_{d: HASH(d) = d'} \phi_d, \text{ where } d \text{ is a feature key (usually a string)}
    \]

\[||\phi'|| \approx ||\phi||\text{ with high probability}
\]
- works surprisingly well: little or no sacrifice in quality!
- implemented in many learning kits (Vowpal Wabbit, etc.)

Can we do even better with data-dependent hashing? 😊

Learning Feature Hashes by Optimizing Hinge Loss with Boosting

1: greedy learning of Hamming representation \(H\)

- will look for \(w_d\) representable as \(\sum_{t \leq T} \alpha_t h_t(\nu(d))\)
- close \(H(\nu(d)) = [h_1(\nu(d)), \ldots, h_T(\nu(d))]\) for different \(d\) \Rightarrow close values of respective \(w_d\)

Hinge loss: \(L = \sum_n (1 - y_n \sum_{t \leq T} \alpha_t h_t(\nu(d)) \phi_{\nu,d})_+\)

Learning \(H(\nu)\) in a boosting fashion:

- \(H_{1-}^{t-1}(\nu) = [h_1(\nu), \ldots, h_{t-1}(\nu)] \in \{0,1\}^{t-1}\)
- \(H(\nu)\) is obtained by appending a bit-function – a per-coordinate decision stump \(h_t = [\nu_d > \theta_t]^{+}\)

\[k^*, \theta^* = \arg \max_{k, \theta} \left| \sum_{n, y_n \phi_{\nu,d} < 1} y_n \sum_d \phi_{\nu,d} [\nu_d > \theta] \right|\]

2: distance-sensitive projection \(B\)

- **Task:** compress \(H(\nu)\) into short codes that have a high collision probability for close \(H(\nu)\).
- we use the **KOR random traces** [Kushilevitz et al., 1998]:
  - for a bit-vector \(h = [h_1, \ldots, h_T]\)
  - and random Bernoulli vectors \(r_m = [r_{m,1}, \ldots, r_{m,T}]\)
  - the trace is \(t = [t_1, \ldots, t_M]\), s.t. \(t_m \equiv (h, r_m) \mod 2\)
  - \(t_m\) has a bias towards closer \(h\), amplified by repeating \(M\) times
- **collision probability decays** with \(D_H(h_1, h_2)\) and \(M\):

\[P[t_1 = t_2 | D_H(h_1, h_2) \leq \Delta] \geq \left( \frac{1}{2} + \frac{1}{2}(1 - 2p)^{\Delta} \right)^M\]

Conclusions

For the considered classification task:

- the less RAM is available (the smaller \(b\))
- or the higher dim. input feature space has

... the more sense it makes to learn hashes

Future work:

- ranking objectives
- tasks with very large dimension:
  - information retrieval
  - collaborative filtering
References


