IIT KHARAGPUR

IIT GANDHINAGAR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Scalable Data Science

Lecture 16c: Feature Hashing

Anirban Dasgupta

Computer Science and Engineering

IIT GANDHINAGAR

RandNLA in ML

- Studied applications of sampling + random projection techniques on linear algebraic problems
  - random projection to approximate PCA
  - QB decomposition
  - random projection for efficient L2 regression

- This lecture we explore the effects of random projection in supervised ML

Outline

- Explore uses of hashing / random projection when the feature space is large

- Two use cases
  1. Text classification: large feature space to capture correlations
  2. Mail classification: feature space is large due to personalization

Text Classification

- Nonlinear/non-convex classifiers
  - excellent results in speech in speech and vision
  - reasonably fast at test time
  - slow training, require wizardry to ensure not stuck in bad local minima
  - not so effective for sparse, high dimensional, datasets of text
    - Although techniques e.g. Word2Vec are changing this

In practice: count based features

- In practice, lot of features are based on “normalized counts” of tokens
  - easy to train, at least on single machine
  - fast at test time: calculate some statistics and then predict

- Works great for settings e.g. text classification

- A common practice, in order to capture higher level correlations is to take various combinations of n-grams and skip-grams
  - Captures high level correlations, still efficient to calculate

Example

“the rain in Spain falls mainly on the plain”

<u>2-grams:</u>

the rain, rain in, in Spain, Spain falls, falls mainly, mainly on, on the, the plain.

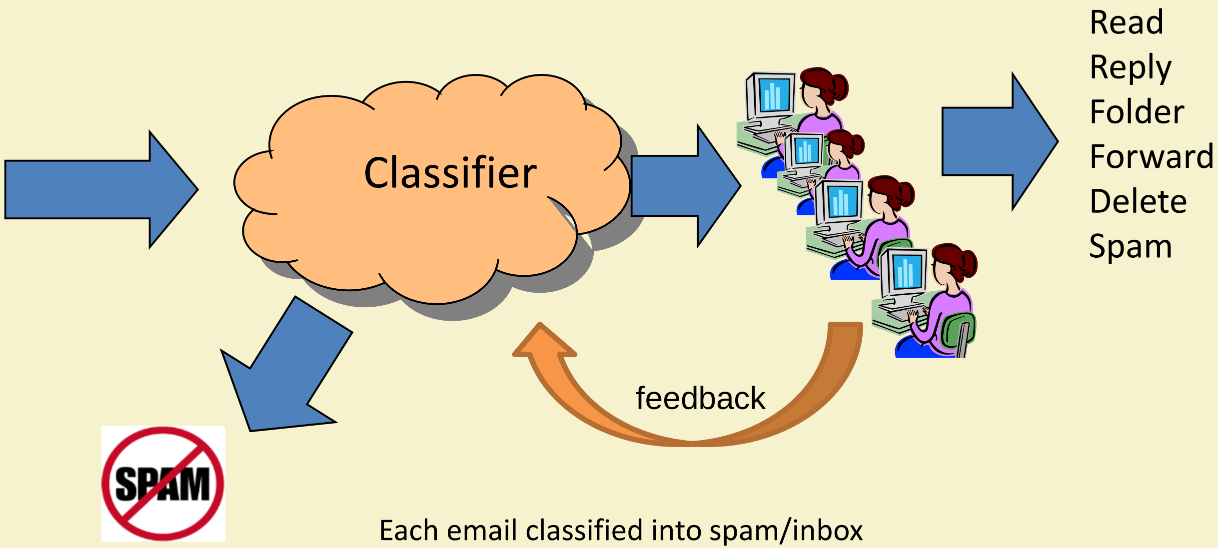
<u>1-skip-2-grams:</u>

the in, *rain Spain*, *in falls*, *Spain mainly*, *falls on*, *mainly the*, *on plain*.

→ Can also look at this as a kernel representation

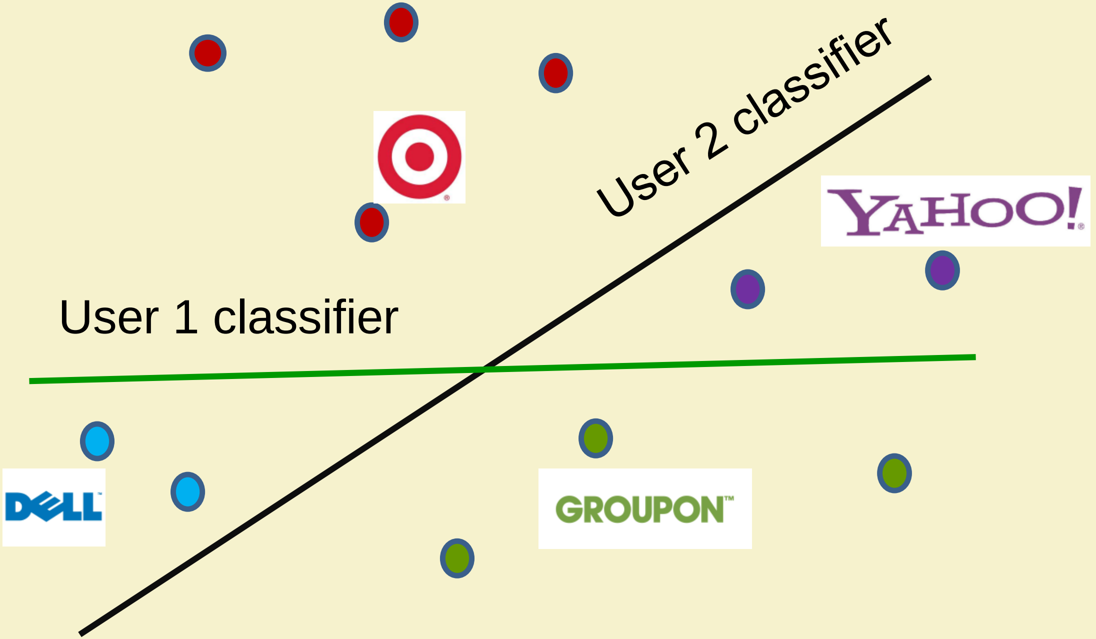
→ Feature space become very high dimensional, how to deal with it?

Second example: Mail Classification



Each email classified into spam/inbox
Classifier updated by user feedback
Users don’t always agree about what is spam

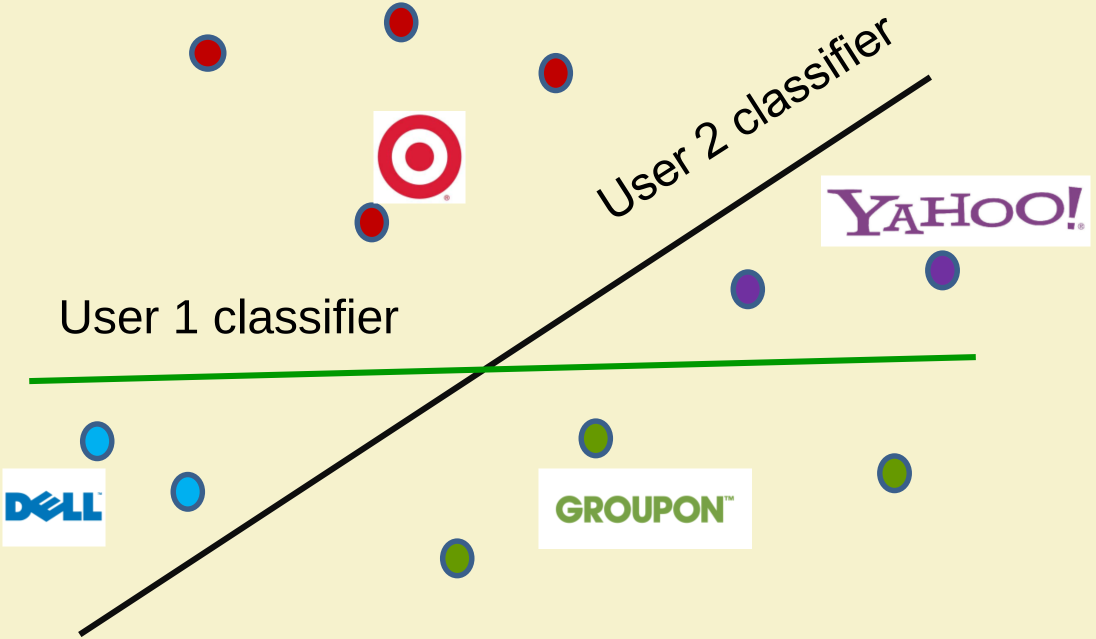
Personalization



Users have different spam preferences

Need to assign different classifiers

Personalization



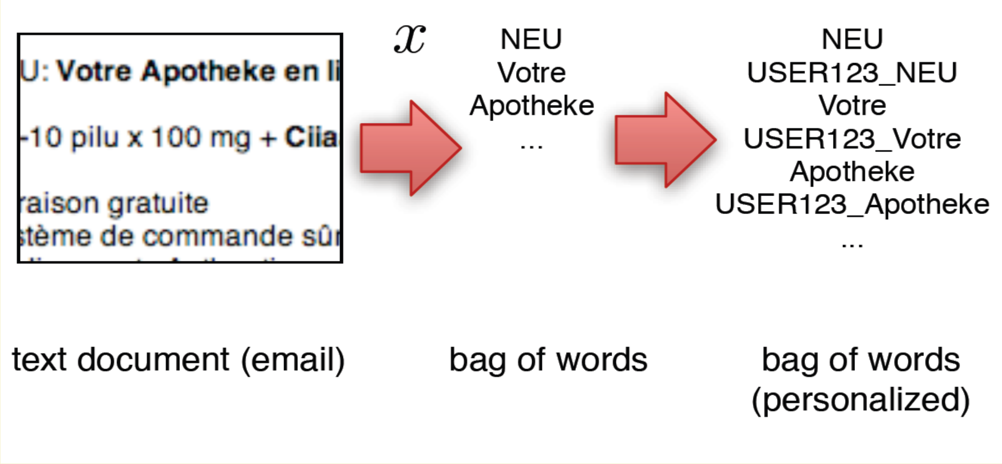
Users have different spam preferences

Need to assign different classifiers

Also should ensure new users get good classifier

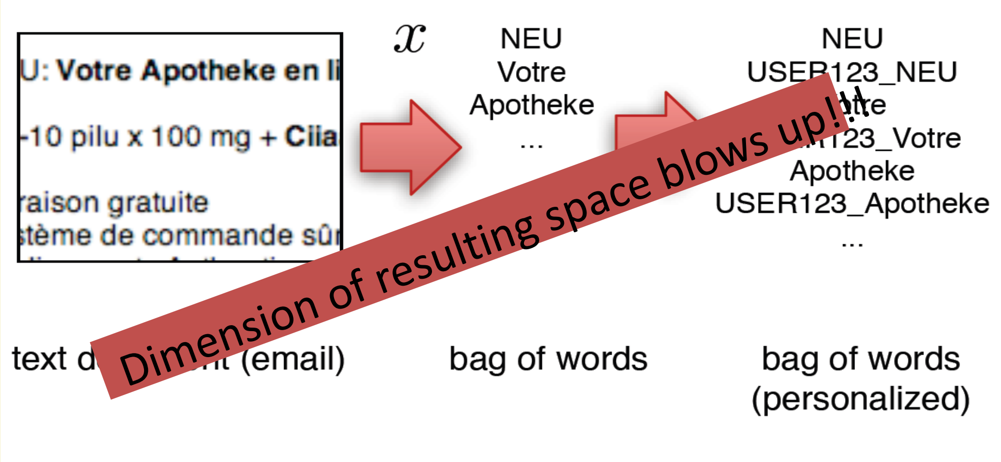
Should be updated frequently using new feedback

Multi-task learning via feature address space



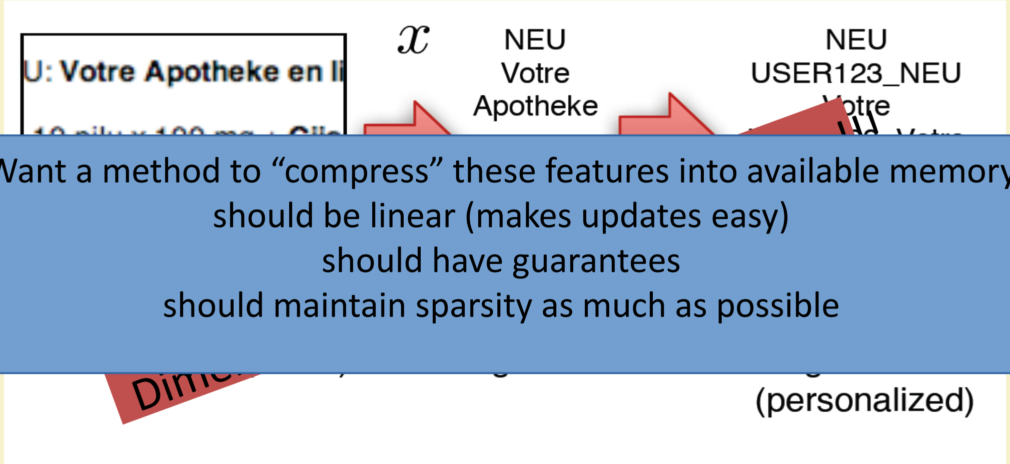
Combination of local and global classifiers: $w_{\text{global}}^T x_{\text{global}} + w_{\text{user}}^T x_{\text{user}}$

Multi-task learning via feature address space



Combination of local and global classifiers: $w_{\text{global}}^T x_{\text{global}} + w_{\text{user}}^T x_{\text{user}}$

Multi-task learning via feature address space



Want a method to "compress" these features into available memory
should be linear (makes updates easy)
should have guarantees
should maintain sparsity as much as possible

Combination of local and global classifiers: $w_{\text{global}}^T x_{\text{global}} + w_{\text{user}}^T x_{\text{user}}$

Hashing

Create a hash-table that hashes features into buckets

Are there principled ways of thinking about this?

Hashing

Create a hash-table that hashes features into buckets

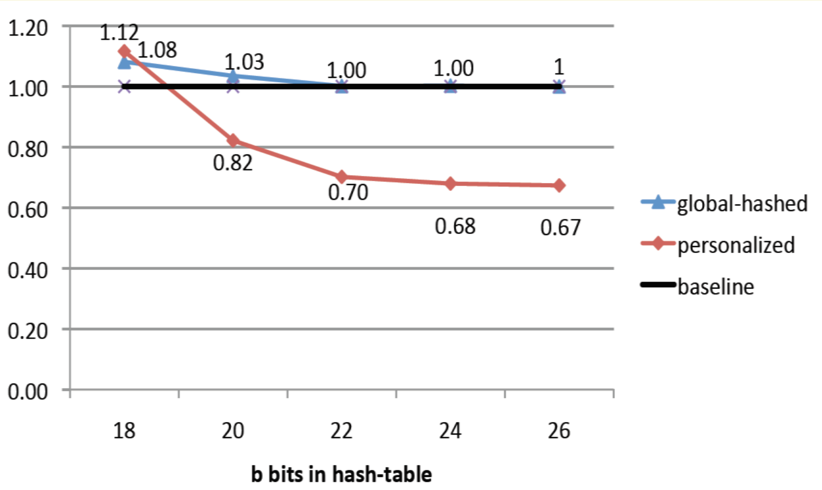
Are there principled ways of thinking about this?

- This is essentially SparseJL, with only one hash being used!
- We should take guidance from principles of designing it, as well as draw from theoretical guarantees
- In particular, should use a sign hash to make this unbiased

Using in practice

- Theoretical guarantees assume a k-wise independent hash function
- Feature hashing now a standard api in many machine learning packages
- Use a standard, fast, hash function
  - Vowpal Wabbit uses MurmurHash3
- Standard procedure:
  - Hash input data, learning classifier over hashed space
  - When testing, hash test point first and then apply classifier
  - Collision is fine, that’s the point
  - Beyond this, use domain knowledge if we want to ensure specific “super important” features or groups of features

Experimental results on spam



- 3.2 million emails, 433K users → ~70 TB if standard hash-map + user personalization
- $\sim 40 \times 10^6$ unique tokens
- Used linear classifier in hashed space
- 2^{26} floats = 256 MB

Summary

- Hashing has proven to be an important practical component of scaling supervised ML to large feature spaces
  - some amount of background theory provided by the random projection literatures
  - More open questions, e.g. how does the margin play a role, generalizability, etc…

References:

- Primary reference
  - *Kilian Weinberger; Anirban Dasgupta; John Langford; Alex Smola; Josh Attenberg (2009). Feature Hashing for Large Scale Multitask Learning (PDF). Proc. ICML.*
  - *Shi, Q.; Petterson J.; Dror G.; Langford J.; Smola A.; Strehl A.; Vishwanathan V. (2009). Hash Kernels. AISTATS.*

Thank You!!