Spam filtering task

Mapping n-grams to feature indices

If your dataset is small you can store
{n-gram → feature index} in hash map.

But if you have a huge dataset that can be a problem

- Let's say we have 1 TB of texts distributed on 10 computers
- You need to vectorize each text
- You will have to maintain {n-gram → feature index} mapping
  - May not fit in memory on one machine
  - Hard to synchronize
- An easier way is hashing: {n-gram → hash(n-gram) % 2^{20}}
  - Has collisions but works in practice
  - sklearn.feature_extraction.text.**HashingVectorizer**
  - Implemented in **vowpal wabbit** library

Spam filtering is a huge task

Spam filtering proprietary dataset

- https://arxiv.org/pdf/0902.2206.pdf
- 0.4 million users
- 3.2 million letters
- 40 million unique words

Let's say we map each token to index using hash function ϕ

- $\phi(x) = \text{hash}(x) \,\%\, 2^b$
- For $b = 22$ we have 4 million features
- That is a huge improvement over 40 million features
- It turns out it doesn't hurt the quality of the model

Hashing example

- $\phi(good) = 0$
- $\phi(movie) = 1$
- $\phi(not) = 2$
- $\boldsymbol{\phi(a) = 3}$
- $\boldsymbol{\phi(did) = 3}$
- $\phi(like) = 4$

Hash collision

$$\mathbf{hash}(\boldsymbol{s}) = \boldsymbol{s}[0] + \boldsymbol{s}[1]\boldsymbol{p}^1 + \cdots + \boldsymbol{s}[\boldsymbol{n}]\boldsymbol{p}^{\boldsymbol{n}}$$

s – string

p – fixed prime number

$s[i]$ – character code

good movie
not a good movie
did not like

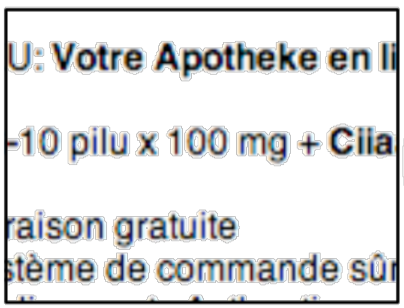
0	1	2	3	4
1	**1**	0	0	0
1	**1**	**1**	**1**	0
0	0	**1**	**1**	**1**

Trillion features with hashing

Personalized tokens trick

- $\phi_o(token) = \text{hash}(token) \ \% \ 2^b$
- $\phi_u(token) = \text{hash}(u + \text{"\_"} + token) \ \% \ 2^b$
- We obtain 16 trillion pairs (user, word) but still 2^b features

text document (email)


U: Votre Apotheke en li
-10 pilu x 100 mg + Ciia
raison gratuite
stème de commande sûr

bag of words

NEU
Votre
Apotheke
…

bag of words (personalized)

NEU
USER123_NEU
Votre
USER123_Votre
Apotheke
USER123_Apotheke
…

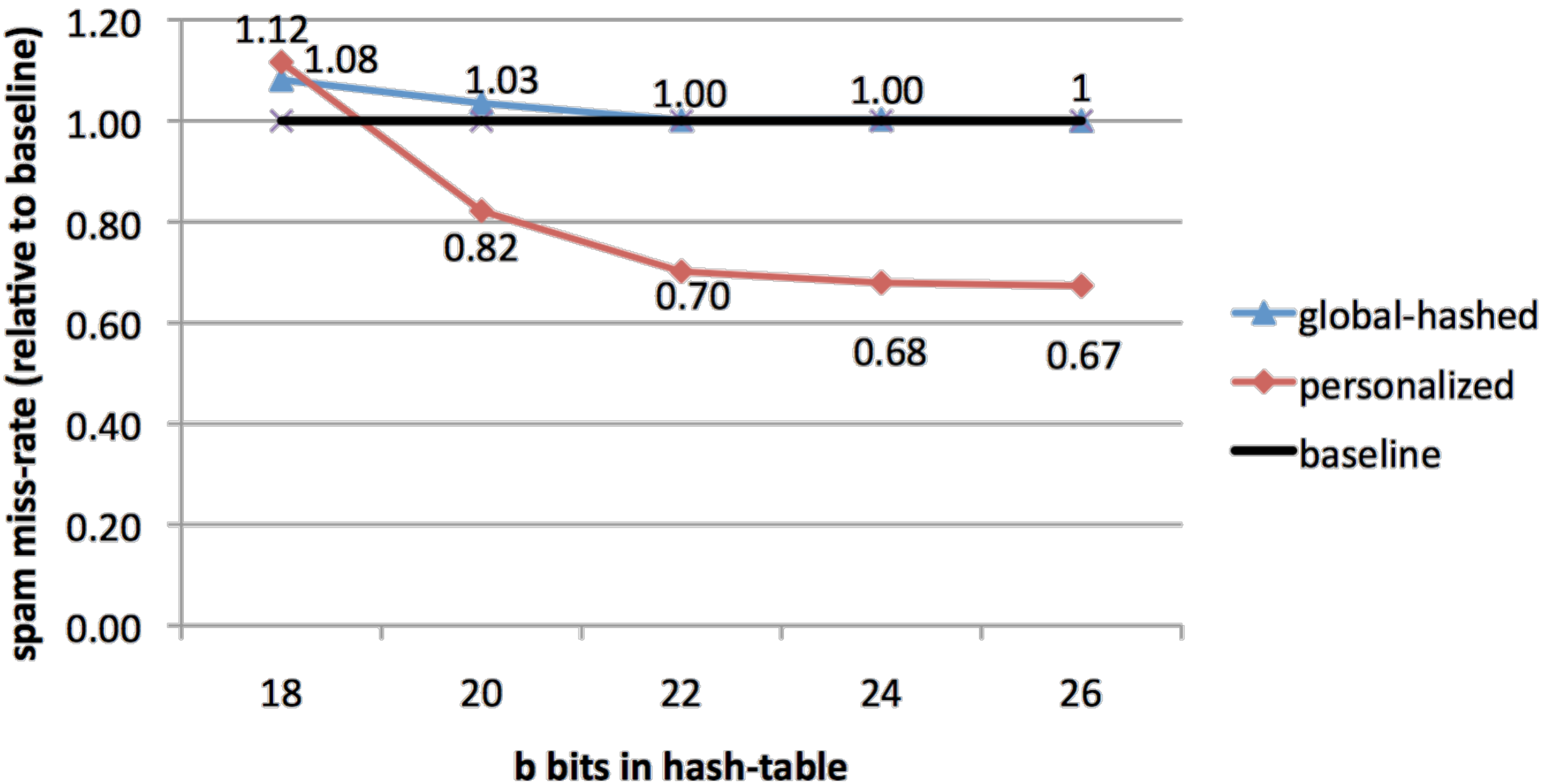
$\phi_0(x) + \phi_u(x)$

1
0
-1
0
0
-1
0
1
0
…

https://arxiv.org/pdf/0902.2206.pdf

Experimental results

- For $b = 22$ it performs just like a linear model on original tokens
- We observe that personalized tokens give a huge improvement in miss-rate!

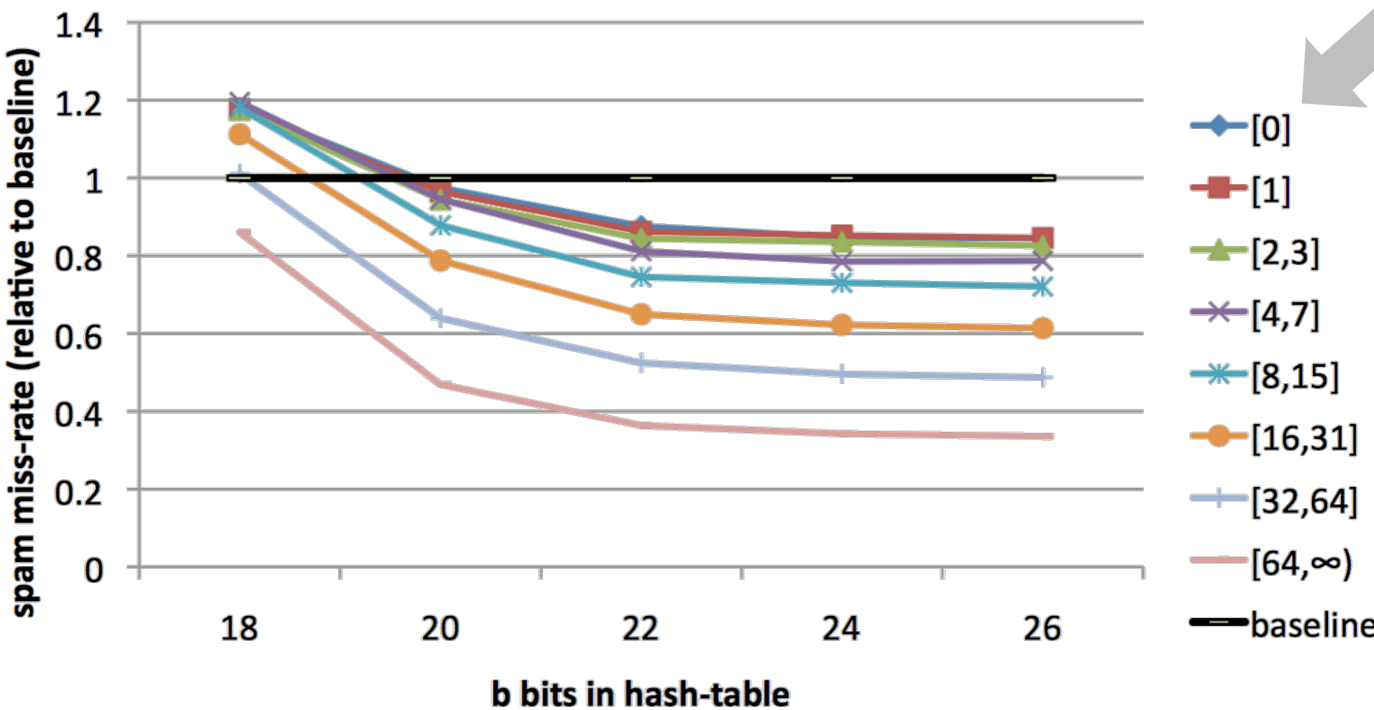


Why personalized features work

Personalized features capture "local" user-specific preference

- Some users might consider newsletters a spam but for the majority of the people they are fine

How will it work for new users?



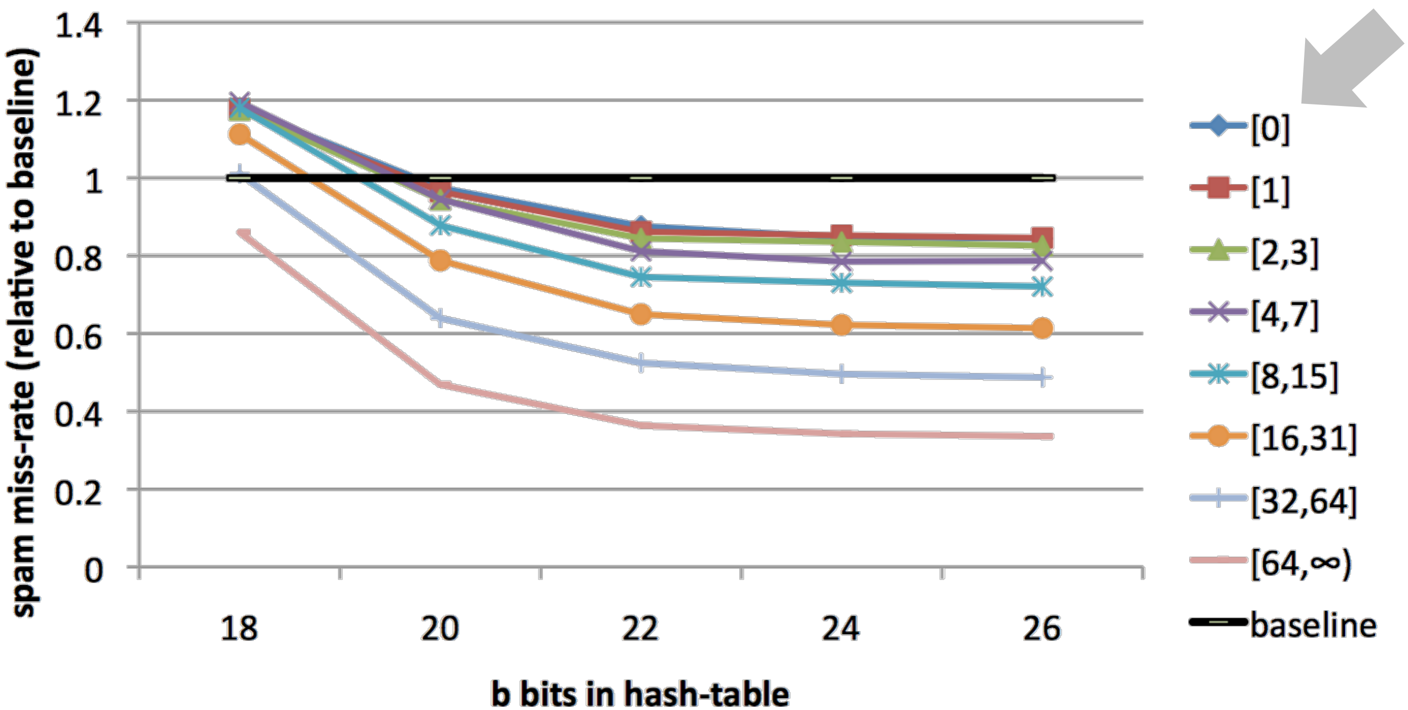
What?

Number of emails in training

Why personalized features work

It turns out we learn better "global" preference having personalized features which learn "local" user preference

- You can think of it as a more universal definition of spam



Number of emails in training

Why the size matters

Why do we need such huge datasets?

- It turns out you can learn better models using the same simple linear classifier

Ad click prediction

- https://arxiv.org/pdf/1110.4198.pdf
- Trillions of features, billions of training examples
- Data sampling hurts the model

	1%	10%	100%
auROC	0.8178	0.8301	0.8344
auPRC	0.4505	0.4753	0.4856
NLL	0.2654	0.2582	0.2554

Sampling rate

Vowpal Wabbit

- A popular machine learning library for training linear models
- Uses feature hashing internally
- Has lots of features
- Really fast and scales well

Format: label | sparse features ...

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

which corresponds to:

1 | tuesday year ...

command: time vw –sgd rcv1.train.txt -c

https://github.com/JohnLangford/vowpal_wabbit/wiki

Summary

- We’ve taken a look on applications of feature hashing
- Personalized features is a nice trick
- Linear models over bag of words scale well for production
- In the next video we’ll take a look at text classification problem using deep learning