MCZA017-13
Processamento de Linguagem Natural

Feature hashing

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

2Q-2019

Da aula anterior...

Matriz (esparsa): termo-contexto

Tamanho da matriz (sem stopwords):

Supondo 4 bytes para armazenar um inteiro

Nome	|V|	MBytes
Ufabc-bcc	96	0.035
notícias	1 580	9.523
Machado-db (4 obras mais conhecidas)	22 784	1 980.25
Machado-db (todas as obras)	62 933	15 108.347

Vetor esparso -> Vetor denso

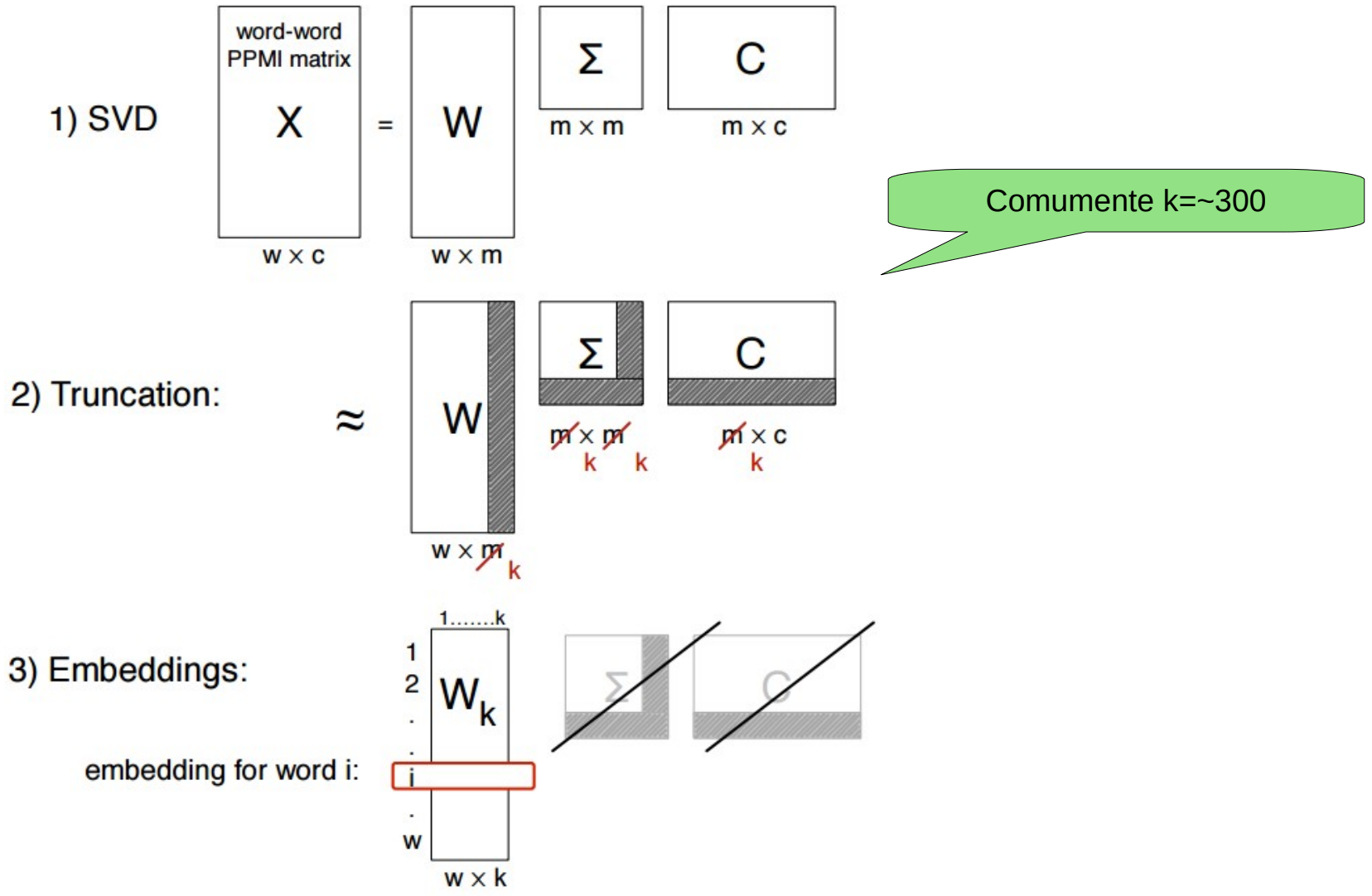
w_i 0 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 ... 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 ~60mil

w_i 1 0 3 4 2 1 4 2 8 5 6 2 9 13 ~1mil

Estrutura mais “fácil” de trabalhar nos algoritmos de aprendizado de máquina.

SVD na matriz PPMI

A mesma ideia pode ser considerada na matriz PPMI



Hashing
(tabelas de dispersão)

Busca em tabelas (vetores/arrays)

Para se resolver os problemas de busca, inserção e remoção em uma tabela com **n** elementos, há várias maneiras:

- **Busca sequencial:**
  acesso em O(n)
- **Busca Binária:**
  acesso em O(log2(n))
  ainda lento se **n** é grande (log2 (1000000) = 20)
- **Uso de árvores balanceadas:**
  acesso em O(log(n))
  acesso bem melhor do que na busca sequencial.

Busca em tabelas (vetores/arrays)

Tabelas de dispersão = **Tabelas de espalhamento** = **Hash tables**

- Busca com acesso médio igual a O(1).
- Isso significa “constante” na média.
  No pior caso é O(**n**).
  Na medida do possível uma constante pequena.

- São estruturas de dados eficientes para implementar um dicionário.

(1) Acesso/Endereçamento direto

- Considere que os valores das chaves sejam: 0, 1, … ,m-1:

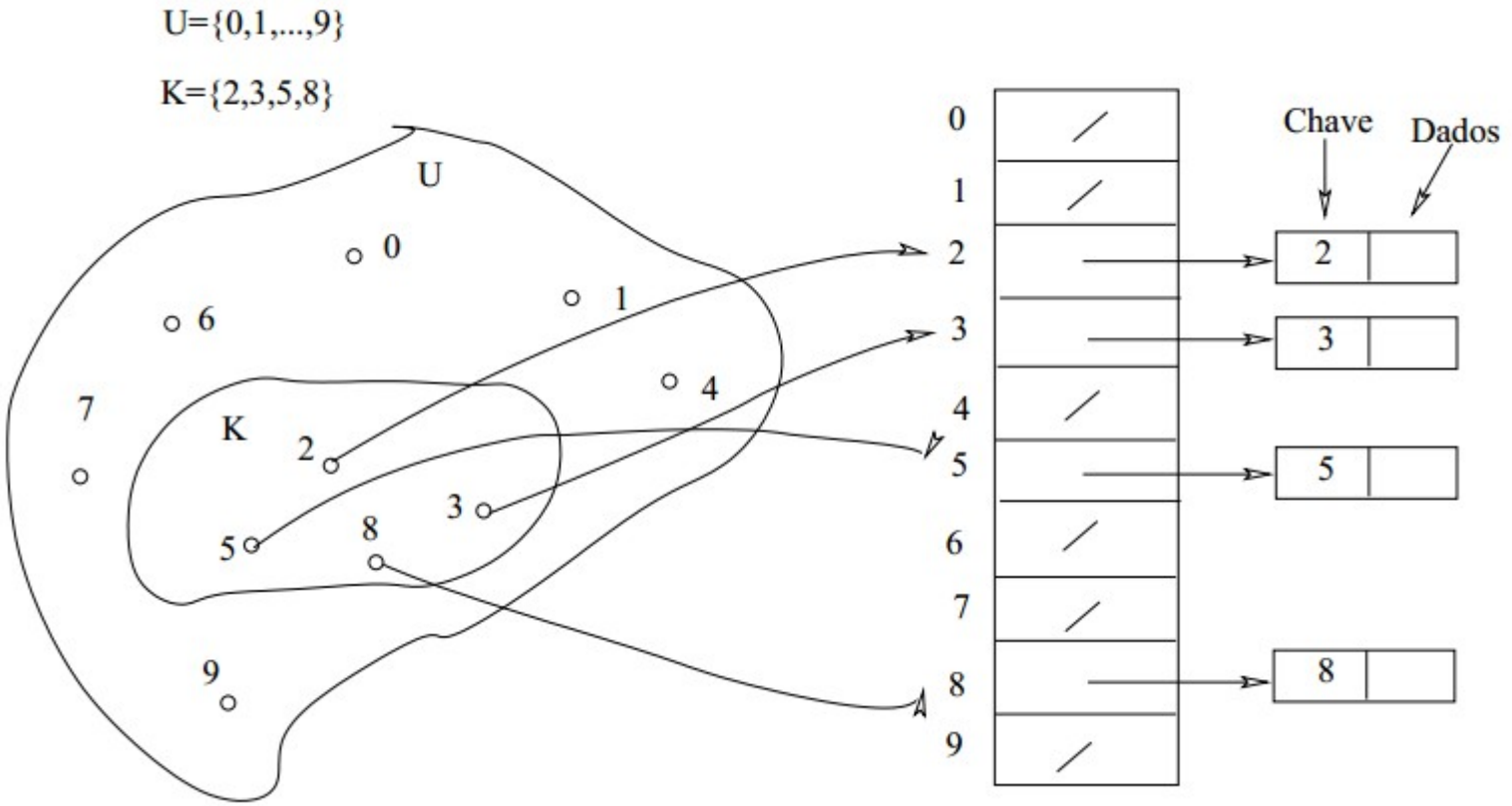
  Pode-se usar diretamente o valor de cada chave em seu índice de tabela (cada chave x armazenada no compartilhamento x)

- O endereçamento direto se baseia no fato de que podemos examinar uma posição qualquer em O(1).

  Isto é aplicável quando podemos alocar uma tabela com uma posição para cada chave possível.

(1) Acesso/Endereçamento direto

Técnica simples que funciona quando o universo de chaves **U** é razoavelmente pequeno.



Teriamos |U|-|k| compartimentos vazios ou espaços.

(1) Acesso/Endereçamento direto

Considerações

- A dificuldade de usar endereçamento direto é obvia:

  A alocação de uma tabela **T** de tamanho **|U|** pode ser inviável para um universo muito grande.

- O conjunto de chaves **K** armazenadas na tabela pode ser muito menor do que o universo de chaves **U:**
  - Espaço utilizado: **O(|U|)**
  - Tempo de busca: **O(1)**

(2) Tabelas de dispersão/espalhamento

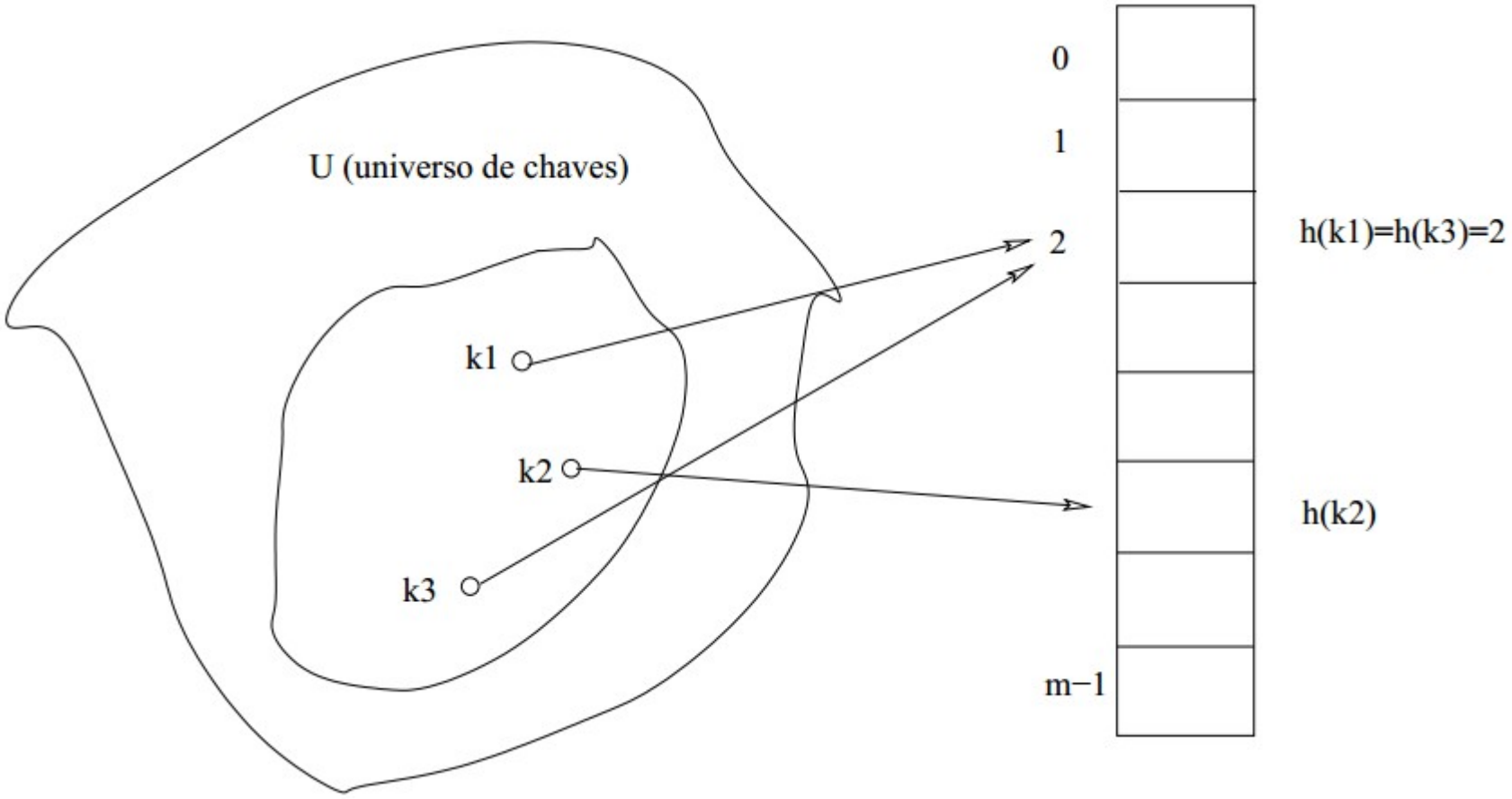
- Através da aplicação de uma função conveniente **(função hash)**, a chave é transformada em um endereço de tabela (endereço base).

hash(chave) → {0, 1, … ,m-1}

- **h(k)** é uma função **h** que mapeia o universo **U** de chaves para entradas da tabela de espalhamento T[0,...,m-1].

$$h : U \rightarrow \{0, 1, \ldots, m-1\}$$

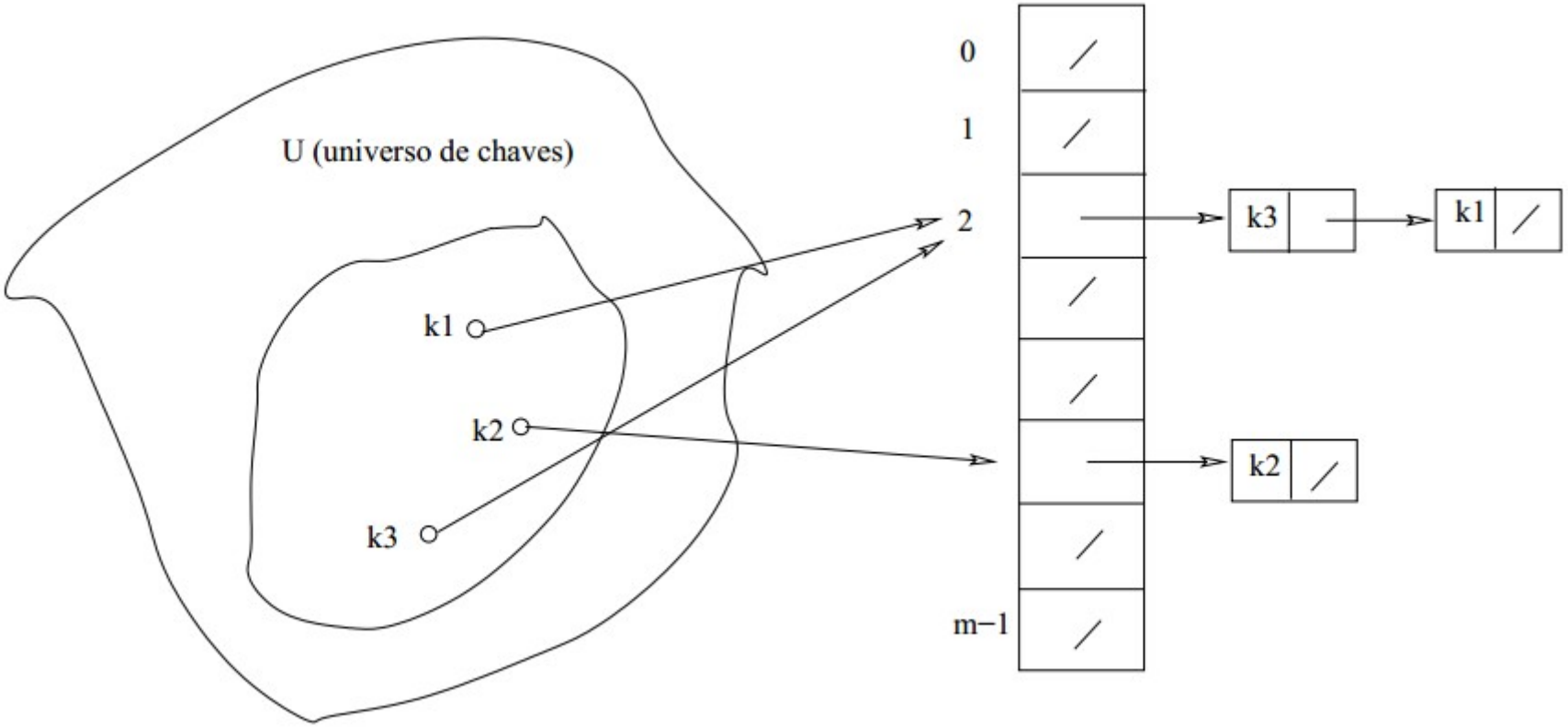
(2) Tabelas de dispersão/espalhamento



Uma dificuldade dessa técnica é a possibilidade de duas chaves k1 e k3 serem mapeadas para a mesma posição na Tabela de Espalhamento.

(2) Tabelas de dispersão/espalhamento

Uma forma de tratar colisões através de **encadeamento**.



Para evitar colisões usa-se uma função Hash que apresente **“comportamento randômico”**

Tabelas de espalhamento

Análise das operações

- Inserção é executada em tempo **O(1)**.
- Remoção de um elemento **x** é executada em tempo **O(1)**.
- Busca leva tempo proporcional ao comprimento da lista.

Veremos algumas funções de dispersão

Funções Hash (de espalhamento):

→ **Método da divisão**

Método da divisão

- Fácil, eficiente e largamente empregado.
- **A chave k é dividida pela dimensão da tabela m**: o resto da divisão é o endereço base:

$$h(k) = k \bmod m$$

Resulta em endereços no intervalo: [0, m-1].

Método da divisão

- Por exemplo, para **m=**12, e **k=**100:

h(k) = k mod m

h(100) = 100 mod 12 = 4

Método da divisão

Bons valores para **m** são números primos

- **O motivo não é óbvio!**
- Ver os seguintes links para uma discussão a respeito:
  - http://stackoverflow.com/questions/1145217/why-should-hash-functions-use-a-prime-number-modulus
  - https://computinglife.wordpress.com/2008/11/20/why-do-hash-functions-use-prime-numbers

Método da divisão

Suponha que desejamos alocar n=2000 elementos de 8 bits, e que não nos importamos em procurar em listas (ligadas) de tamanho médio 3.

Qual seria o tamanho apropriado para a tabela T?

Então, podemos fazer **m=701**, pois este é um número primo próximo de 2000/**3**, então

$$\alpha = \frac{n}{m} = \frac{2000}{701} \approx 3$$

Método da divisão

n=2000, **m=**701

h(k) = k mod 701

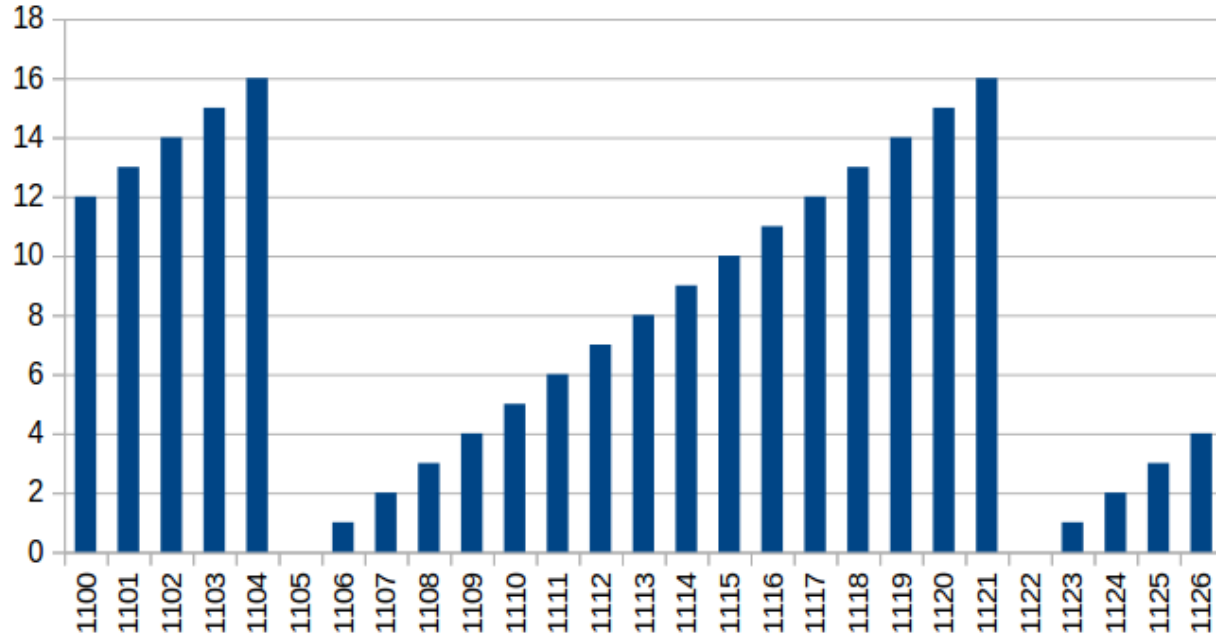
2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541
547	557	563	569	571	577	587	593	599	601
607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733
739	743	751	757	761	769	773	787	797	809
811	821	823	827	829	839	853	857	859	863
877	881	883	887	907	911	919	929	937	941
947	953	967	971	977	983	991	997		

Tabela de primos

Método da divisão

m= 17

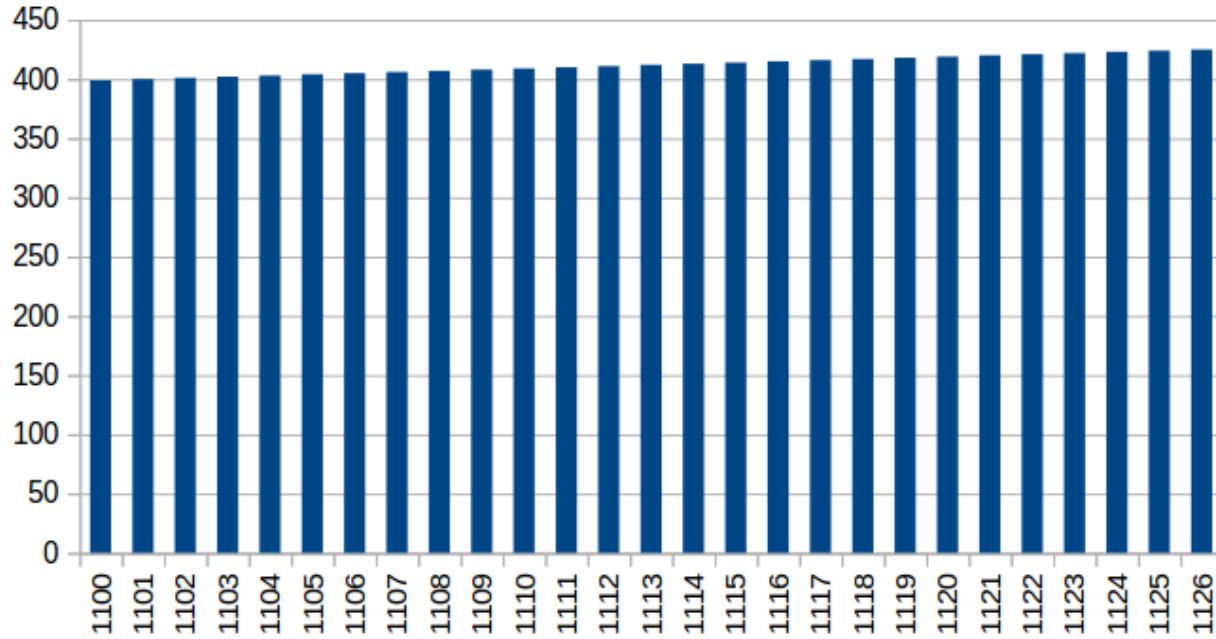
k	h(k)
1100	12
1101	13
1102	14
1103	15
1104	16
1105	0
1106	1
1107	2
1108	3
1109	4
1110	5
1111	6
1112	7
1113	8
1114	9
1115	10
1116	11
1117	12
1118	13
1119	14
1120	15
1121	16
1122	0
1123	1
1124	2
1125	3
1126	4



Método da divisão

m= 701

k	h(k)
1100	399
1101	400
1102	401
1103	402
1104	403
1105	404
1106	405
1107	406
1108	407
1109	408
1110	409
1111	410
1112	411
1113	412
1114	413
1115	414
1116	415
1117	416
1118	417
1119	418
1120	419
1121	420
1122	421
1123	422
1124	423
1125	424
1126	425



Funções Hash (de espalhamento):

→ Método da multiplicação

Método da multiplicação

O método utiliza uma constante **A** (0<A<1), sendo **h(k)** calculado como:

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

$$(kA \bmod 1) = kA - \lfloor kA \rfloor$$

A vantagem deste método é que o valor de **m** não é crítico como no método de divisão.

- Mas a escolha de uma constante **A** adequada é crítica.

Método da multiplicação

Alguns valores de **A** são melhores do que outros, em particular a razão áurea

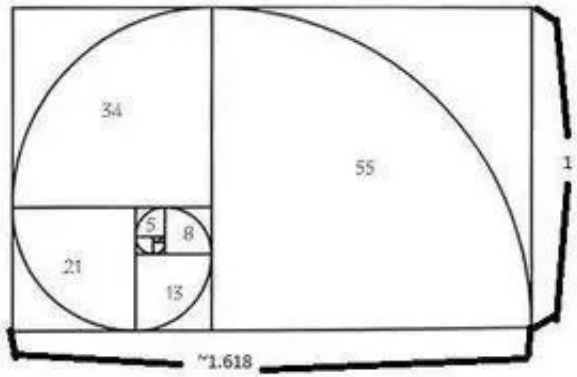
m = 1000
A = 0,6180339887...

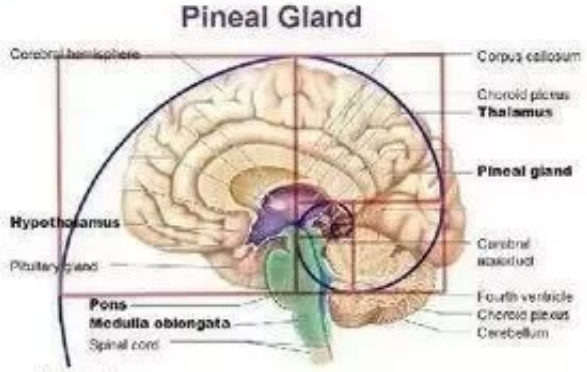
h(1100) = 837
h(1101) = 455
h(1102) = 73
h(1103) = 691
h(1104) = 309
h(1105) = 927
h(1106) = 545
h(1107) = 163
.
.
.
.
.

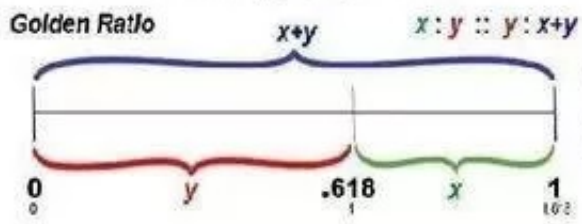
$$A \approx (\sqrt{5} - 1)/2 = 0,6180339887$$

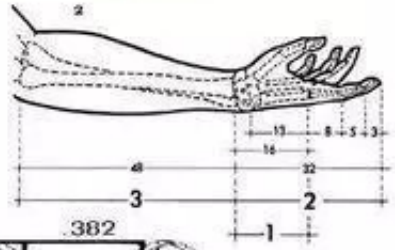
https://www.facebook.com/QuantumWorldAwakenYourMind

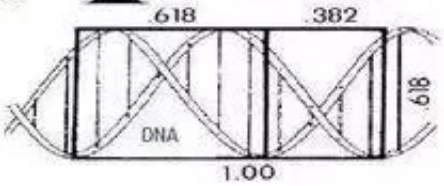
THE GOLDEN RATIO IN THE HUMAN BODY











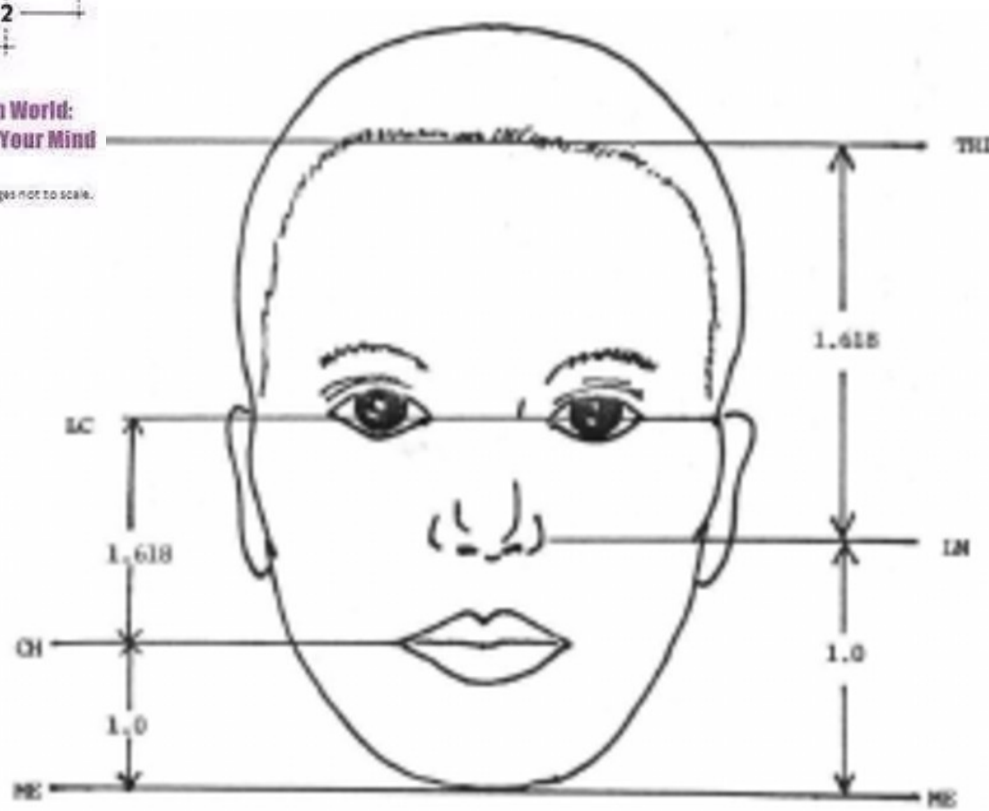
Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610...

3 x .618 ≈ 2	13 x .618 ≈ 8	55 x .618 ≈ 34
5 x .618 ≈ 3	21 x .618 ≈ 13	89 x .618 ≈ 55
8 x .618 ≈ 5	34 x .618 ≈ 21	144 x .618 ≈ 89

Quantum World:
Awaken Your Mind

*Some images not to scale.



Funções Hash:

→ **Método da dobra**

Método da dobra

Quebre a chave em partes e combine-as de algum modo

Exemplo:

Se as chaves são números de 8 dígitos, e a tabela hash tem 1000 entradas, quebre a chave em 3 números:

- Número 1: 3 dígitos
- Número 2: 3 dígitos
- Número 3: 2 dígitos

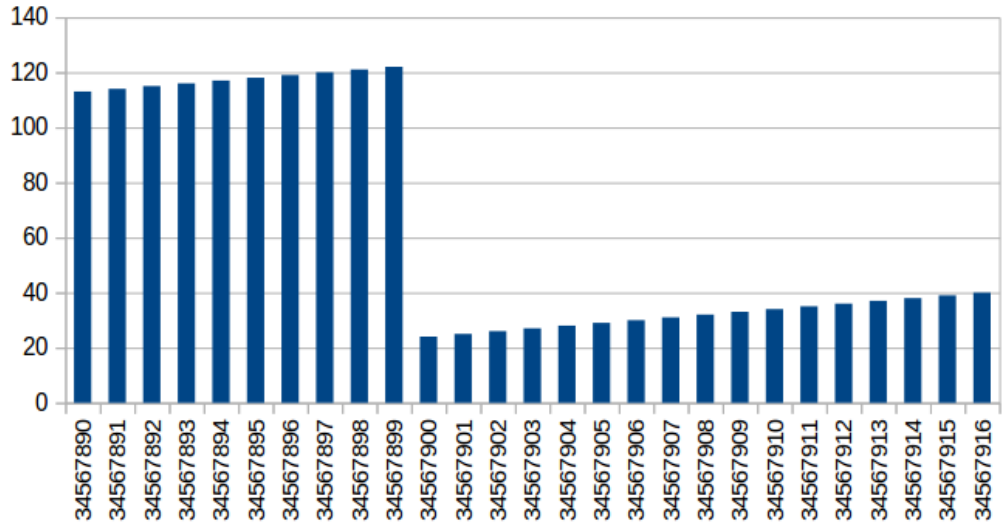
Some-os, considerando os 3 últimos dígitos da soma para compor a chave.

73419**883** → **734** + **198** + **83** = 1015 → 015 → h(k) = 15

Método da dobra

n= 1000

k	d1	d2	d3	soma	h(k)
34567890	345	678	90	1113	113
34567891	345	678	91	1114	114
34567892	345	678	92	1115	115
34567893	345	678	93	1116	116
34567894	345	678	94	1117	117
34567895	345	678	95	1118	118
34567896	345	678	96	1119	119
34567897	345	678	97	1120	120
34567898	345	678	98	1121	121
34567899	345	678	99	1122	122
34567900	345	679	00	1024	24
34567901	345	679	01	1025	25
34567902	345	679	02	1026	26
34567903	345	679	03	1027	27
34567904	345	679	04	1028	28
34567905	345	679	05	1029	29
34567906	345	679	06	1030	30
34567907	345	679	07	1031	31
34567908	345	679	08	1032	32
34567909	345	679	09	1033	33
34567910	345	679	10	1034	34
34567911	345	679	11	1035	35
34567912	345	679	12	1036	36
34567913	345	679	13	1037	37
34567914	345	679	14	1038	38
34567915	345	679	15	1039	39
34567916	345	679	16	1040	40



Hash para strings

- Até aqui consideramos o caso da chave ser um número natural.

  k = 'gato'

  h(k) = ?

Hash para strings

Método da divisão

```
/*********************************
  função de espalhamento exemplo
  devolve um inteiro entre 0 e 50
*********************************/
int f(char *s){
  int k = strlen(s);
  int i, j = 0;
  for(i=0; i<k; i++) j+=s[i];
  return j % 51;
}
```

Hash para strings

Uma alternativa

Por exemplo, se a chave é uma cadeia de caracteres podemos interpretá-la como um natural em uma base conveniente, por exemplo:

$$gato = 7 \cdot 26^3 + 1 \cdot 26^2 + 20 \cdot 26^1 + 15 = 124243.$$

- Palavras mais longas representam um desafio maior, pois o valor calculado pode ser muito grande.

Sobre hash

Vantagens:

- Algoritmo simples e eficiente para inserção, remoção e busca.

Desvantagens:

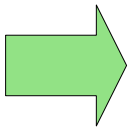
- Nenhuma garantia de balanceamento.
- Espaço sub-utilizado nas tabelas.
- O grau de espalhamento é sensível à função de hashing utilizada e ao tipo de informação usada como chave.

Desafio:

- Pense na criação de uma função hash “universal”.

Feature hashing (*"Hashing trick"*)

- *John likes to watch movies.*
- *Mary likes movies too.*
- *John also likes football.*



John	likes	to	watch	movies	Mary	too	also	football
1	1	1	1	1	0	0	0	0
0	1	0	0	1	1	1	0	0
1	1	0	0	0	0	0	1	1

Feature hashing

w_i 0 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 ... 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 ~60mil

w_i 1 0 3 4 2 1 4 2 8 5 6 2 9 13 ~1mil

Nome	|V|	MBytes
Ufabc-bcc	96	0.035
notícias	1 580	9.523
Machado-db (4 obras mais conhecidas)	22 784	1 980.25
Machado-db (todas as obras)	62 933	15 108.347
Novo Corpus	40 milhões	6400 teras

Feature hashing

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

Abstract

Empirical evidence suggests that hashing is an effective strategy for dimensionality reduction and practical nonparametric estimation. In this paper we provide exponential tail bounds for feature hashing and show that the interaction between random subspaces is negligible with high probability. We demonstrate the feasibility of this approach with experimental results for a new use case — multitask learning with hundreds of thousands of tasks.

Spam filtering proprietary dataset

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

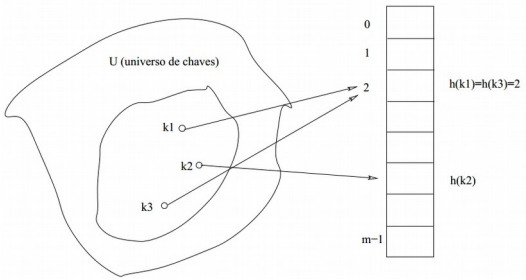
Muitas palavras novas ou com erros de escrita

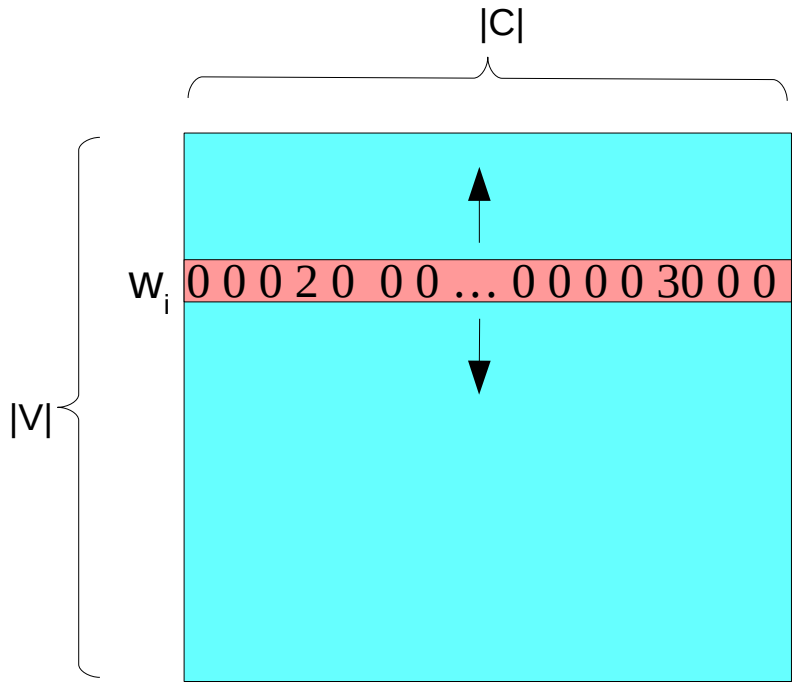
Weinberger, K., Dasgupta, A., Langford, J., Smola, A., & Attenberg, J. (2009, June). **Feature hashing for large scale multitask learning**. In Proceedings of the 26th Annual International Conference on Machine Learning (pp. 1113-1120). ACM.

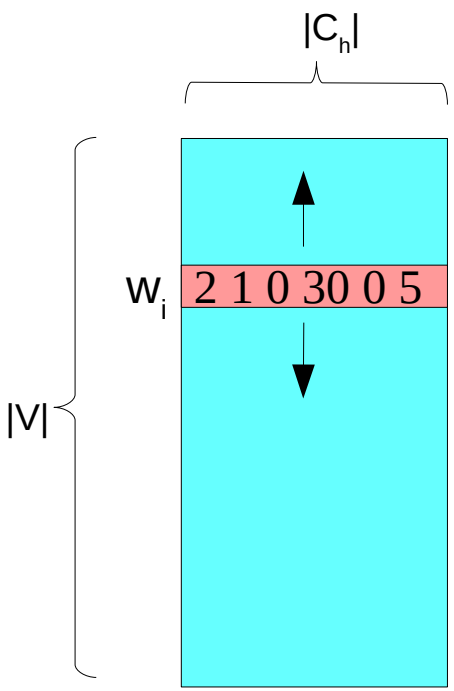
Feature hashing

- É uma maneira rápida e eficiente para tratar vetores com muitas características.
- No lugar de utilizar índices (de uma matriz, por exemplo termo-contexto), será utilizado um valor de hash.
- Assim, essa estratégia permite reduzir dimensionalidade (número de características – e.g., colunas)

Feature hashing: na matriz termo-contexto







Cada palavra é representado por um vetor cumprido mas com muitos valores nulos.

Cada palavra é representado por um vetor 'menor'.

teste1.py

Não temos controle do contradomínio

```
def hashF(word):
    k = 0
    for i in range(0, len(word)):
        k += ord(w[i])
    return k

if __name__ == '__main__':

    while True:
        w = input("\nDigite uma palavra: ")
        print(" h = {}".format(hashF(w)) )
```

teste1.py

```
python3 teste1.py

Digite uma palavra: casa
 h = 408

Digite uma palavra: casas
 h = 523

Digite uma palavra: cassa
 h = 523

Digite uma palavra: multidisciplinaridade
 h = 2228

Digite uma palavra: programação
 h = 1426

Digite uma palavra: programacao
 h = 1164

Digite uma palavra: programações
 h = 1549
```

teste1.py

```
python3 teste1.py

Digite uma palavra: algoritmos
 h = 1089

Digite uma palavra: algoritmico
 h = 1178

Digite uma palavra: logaritmo
 h = 974

Digite uma palavra: logotipo
 h = 877

Digite uma palavra: alkoresmi
 h = 967

Digite uma palavra: sapato
 h = 648

Digite uma palavra: cadeado
 h = 705
```

teste2.py

```
def hashF(word):
    k = 0
    for i in range(0, len(word)):
        k += ord(w[i])
    return k%100
```

Contradomínio: [0:99]

teste2.py

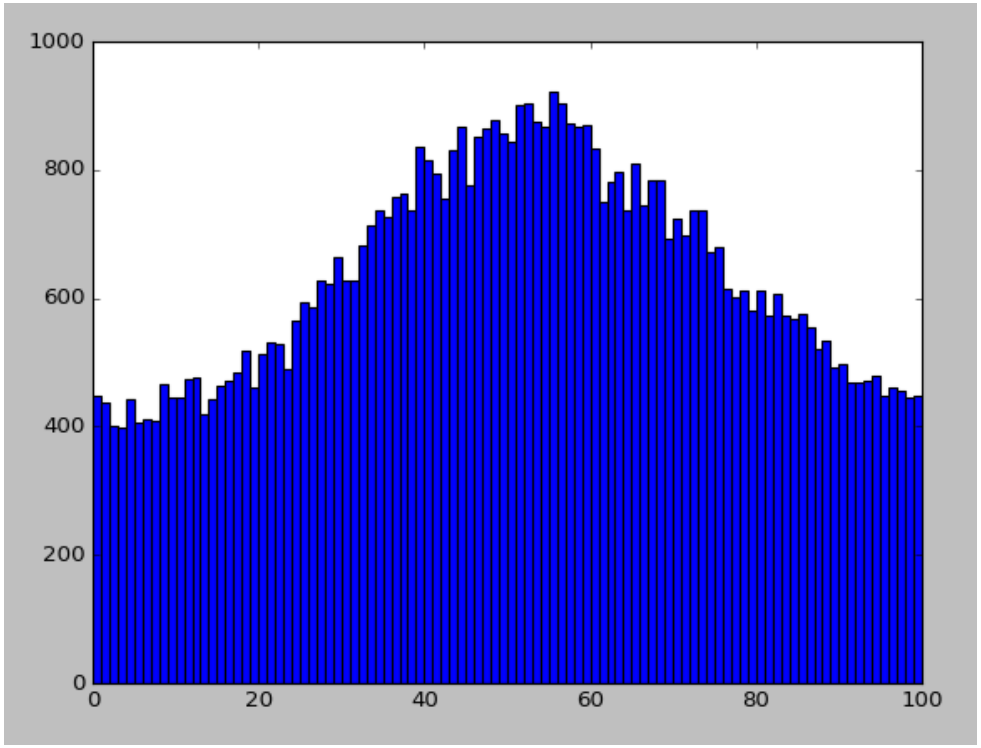
```
python3 teste2.py

Digite uma palavra: casa
 h = 8

Digite uma palavra: casas
 h = 23

Digite uma palavra: multidisciplinaridade
 h = 28

Digite uma palavra: programação
 h = 26

Digite uma palavra: programacao
 h = 64

Digite uma palavra: programações
 h = 49
```

teste3.py

```
M = 100

def hashF(word):
    k = 0
    for i in range(0, len(word)):
        k += ord(word[i])
    return k%M

if __name__ == '__main__':
    dataset = sys.argv[1]

    Mcollision = numpy.zeros(M)

    for s in open(dataset,'r').readlines():
        index = hashF(s.strip().lower())
        Mcollision[index] += 1


    plt.bar(range(M), Mcollision, 1, color="blue")
    plt.show()
```

teste3.py

```
python3 teste3.py Machado-palavras-unicas.txt
```

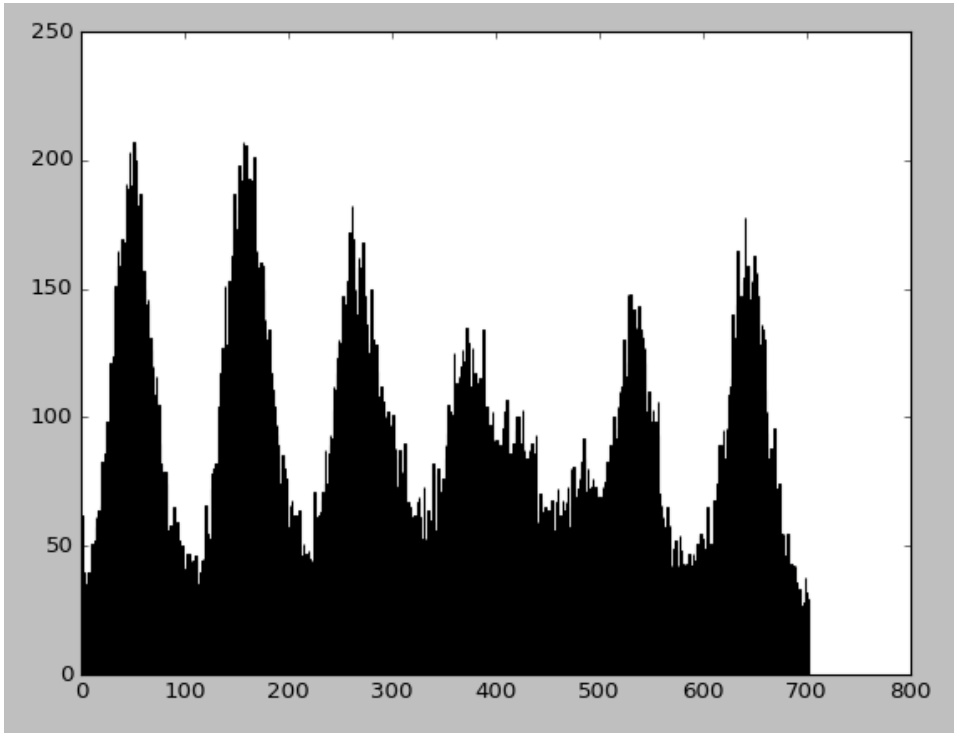


M=100

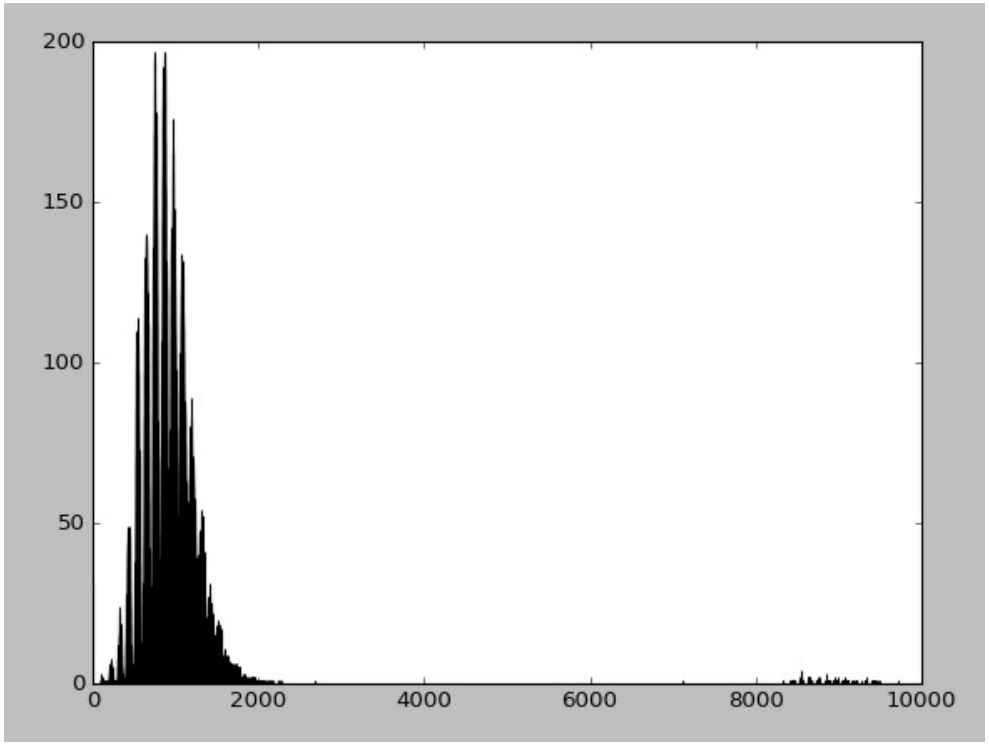
Palavas únicas em Machado = 63573

teste3.py

```
python3 teste3.py Machado-palavras-unicas.txt
```



M=701

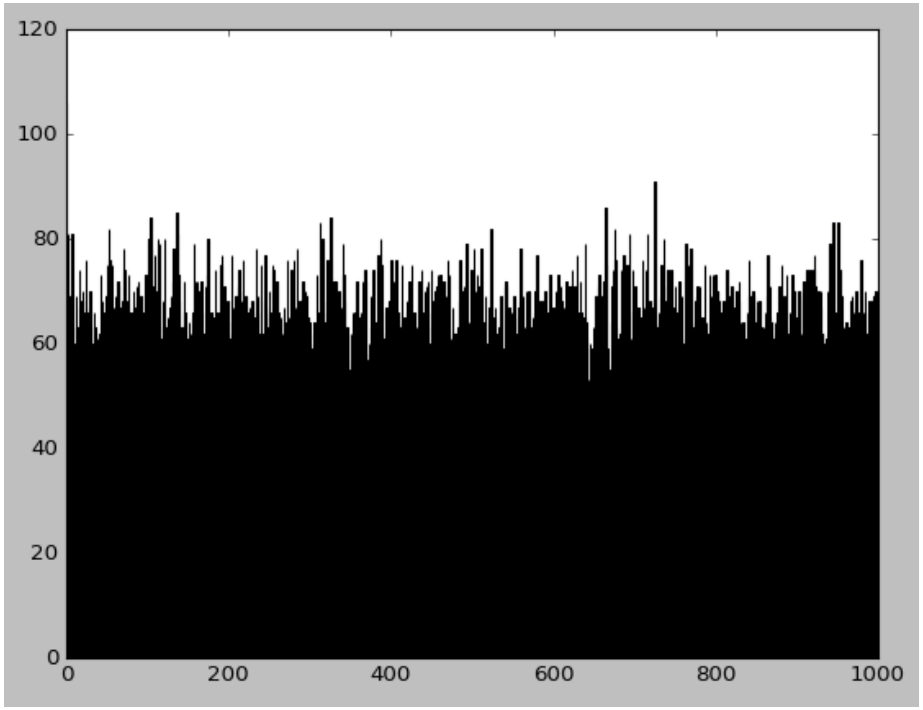


M=10000

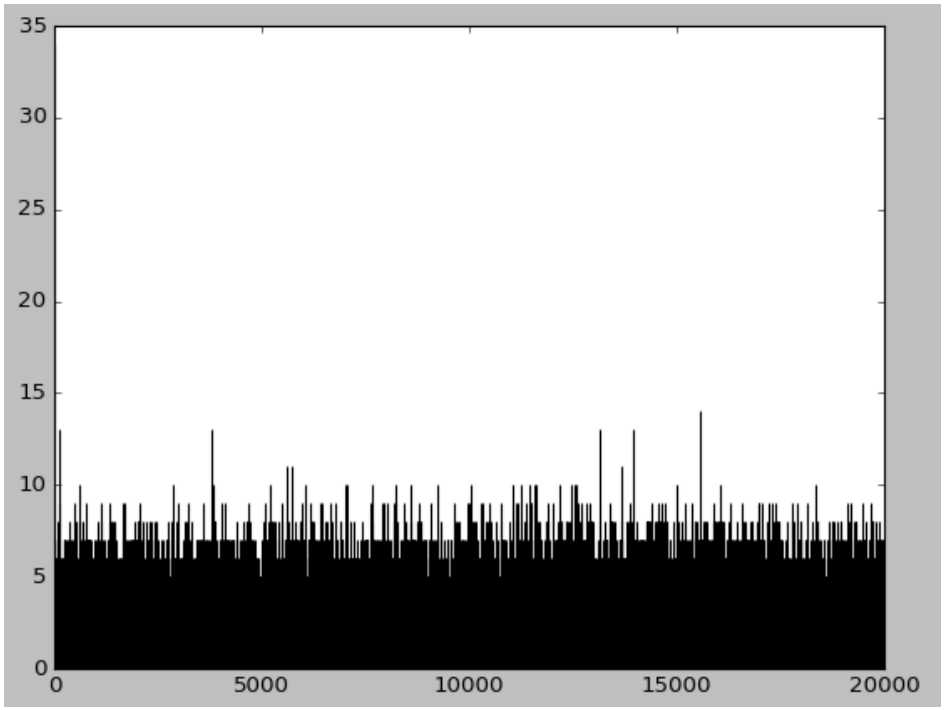
teste4.py

```
M = 20000

def hashF(word):
    k = 0
    for i in range(0, len(word)):
        k += (17**i) * ord(word[i])
    return k%M
```

Palavras únicas em Machado = 63573



M=1000



M=20000

Feature hashing (*"Hashing trick"*) para matrizes termo-contexto

testePLN.py

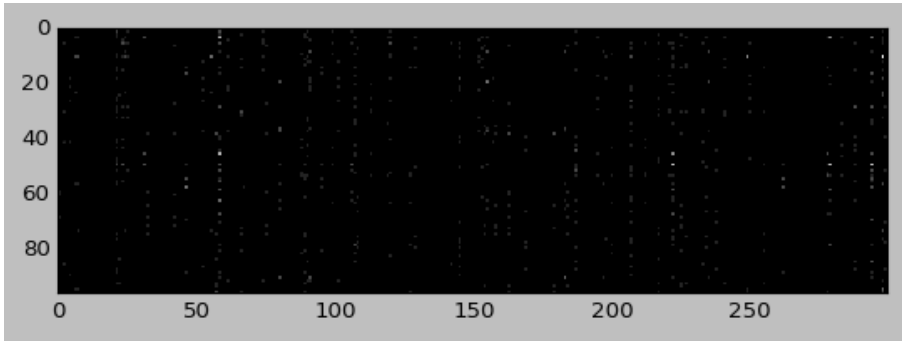
```
M = 300

def hashF(word):
    k = 0
    for i in range(0, len(word)):
        k += (17**i) * ord(word[i])
    return k%M
```

```
dirDB = sys.argv[1]

Document   = dict([])
Vocabulary = set([])

# leitura das stopwords
Stopwords        = set([])
for s in open("stopwords-pt.txt",'r').readlines():
    Stopwords.add(s.strip().lower())

# leitura dos documentos
for fileName in os.listdir(dirDB):
    Document[fileName] = []
    document = open(dirDB+"/"+fileName,'r')
    content  = document.read().lower()

    for w in re.findall(regex, content):
        if w not in Stopwords and len(w)>=3:
            Document[fileName].append(w)
    Vocabulary.update( Document[fileName] )


D = len(Document)
V = len(Vocabulary)
S = len(Stopwords)
```

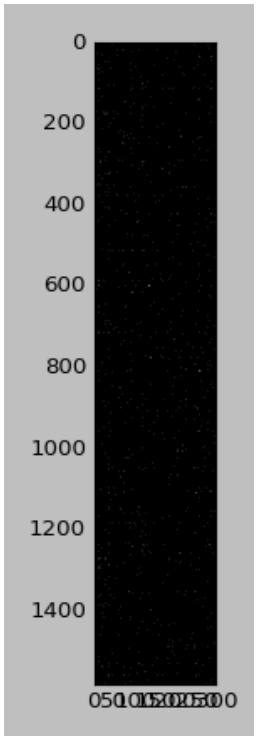
```
# contabilizando os pares de palavras
k           = 3
Mcontext    = numpy.ones((V, M))   # colunas: 0 .. M-1
Vocabulary  = list(Vocabulary)
iVocabulary = dict([])

for (i,w) in enumerate(Vocabulary):
    iVocabulary[w] = i

for d in Document.keys():
    print (d)
    for (i,w) in enumerate(Document[d]):
        context = []
        if i>k:
            context += Document[d][i-k:i]
        if i<len(Document[d])-k:
            context += Document[d][i+1:i+k+1]

        iw = iVocabulary[w]
        for wc in context:
            #Mcontext[iw, iVocabulary[wc]] += 1
            Mcontext[iw, hashF(wc) ] += 1
```

```
# Calculando para cada palavra do vocabulario: PPMI
N    = numpy.sum(Mcontext)
PPMI = numpy.zeros((V, M))

Fw   = numpy.sum(Mcontext, axis=1) # somatoria de cada linha
Fc   = numpy.sum(Mcontext, axis=0) # somatoria de cada coluna

for i in range(0, V):
    for j in range(0, M):
        PPMI[i,j] = max(0, math.log2((Mcontext[i,j]/N)/(Fw[i]/N*Fc[j]/N)) )

import matplotlib.image as img
import matplotlib.pyplot as plt
plt.imshow(Mcontext[:,:], cmap=plt.cm.gray, interpolation='none')
plt.show()
```

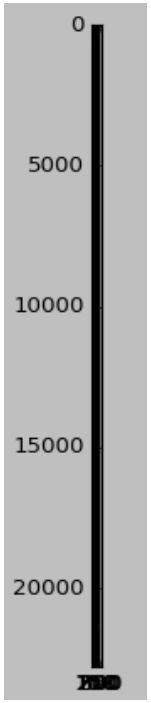
testePLN.py



ufabc-bcc



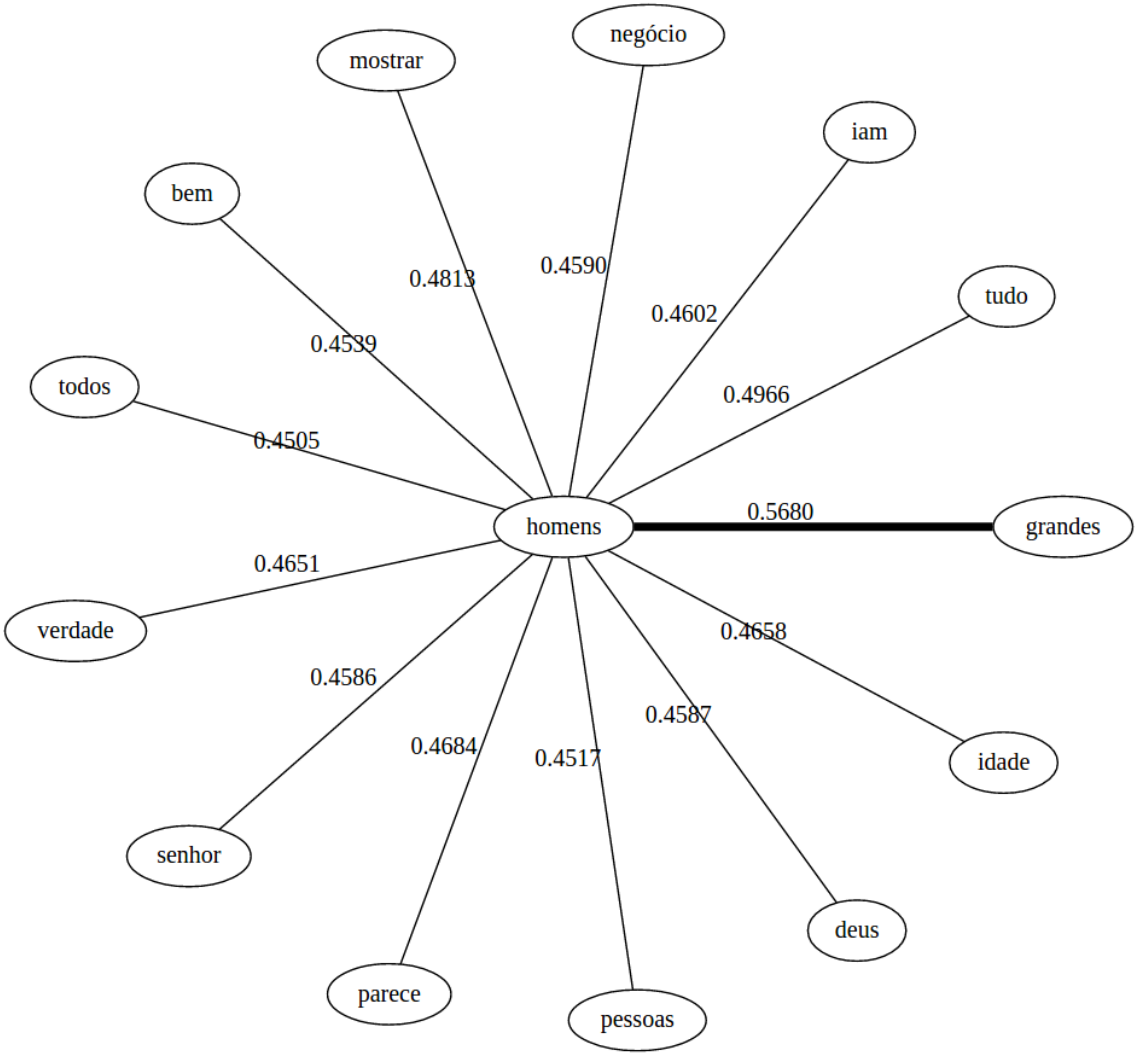
noticias



machado-db

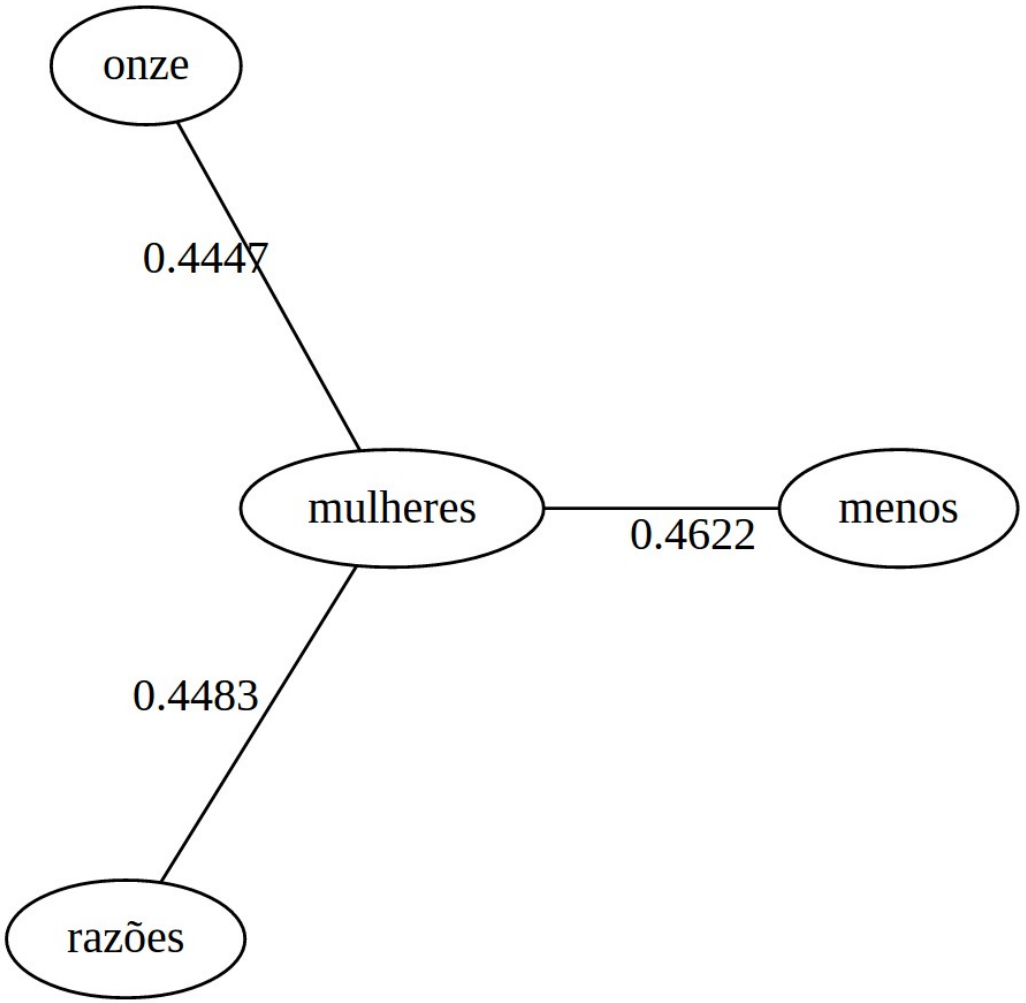
testePLN2.py

```
python3 testePLN2.py  machado-db/
```



testePLN2.py

```
python3 testePLN2.py  machado-db/
```



testePLN2.py

```
python3 testePLN2.py  machado-db/
```