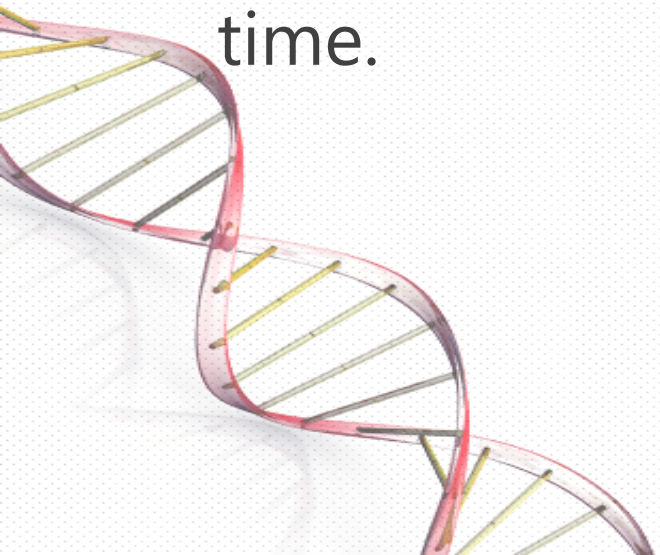
Indexing

Hammad Naveed
hammad.naveed@nu.edu.pk

Indexing a text (a genome, etc)

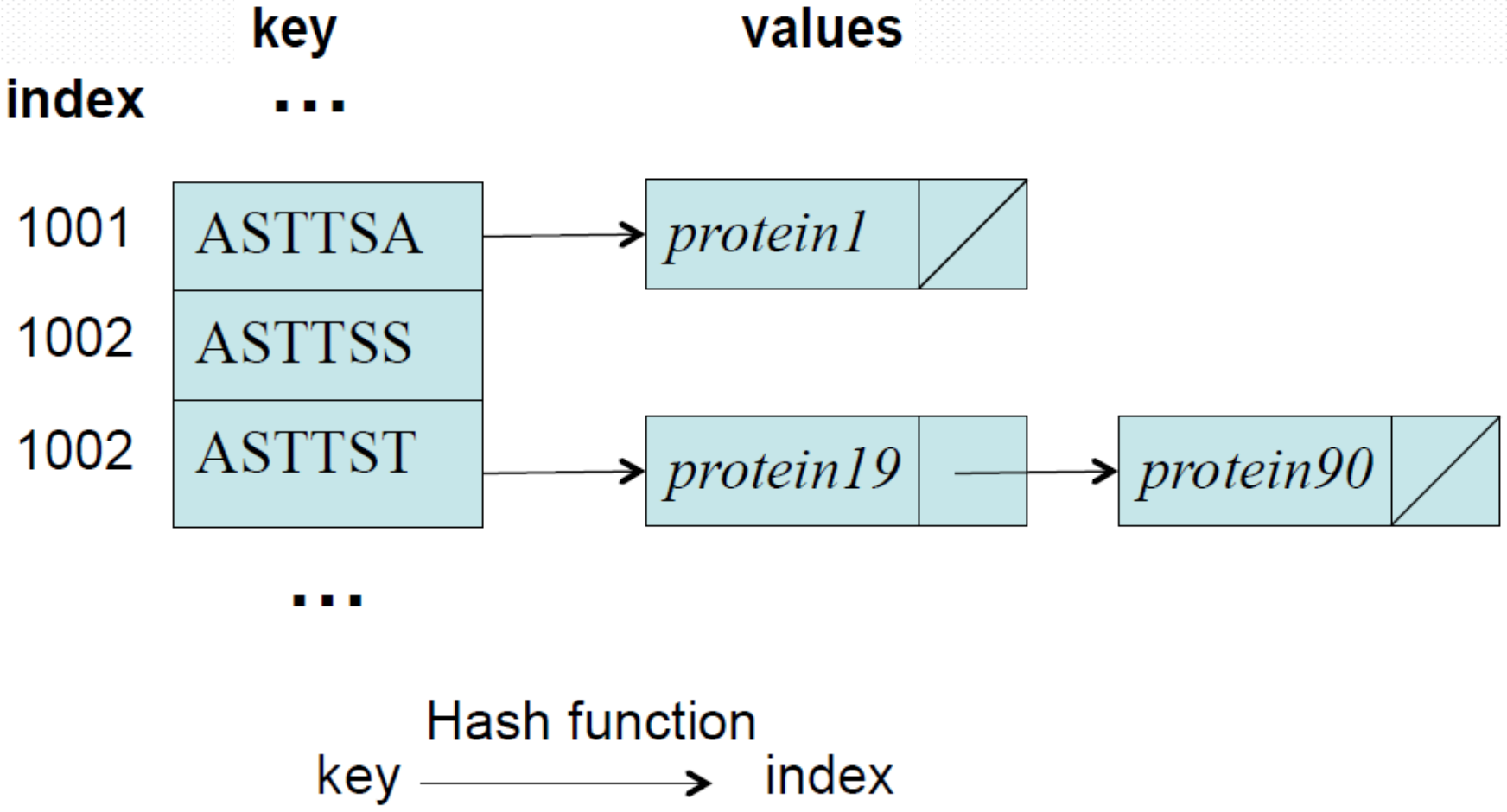
- Example 1: we want to index a genome such that we can look up any k-mer along the genome in O(1) time (without scanning the whole genome).

- Example 2: we want to index a protein database such that we can look up all the proteins containing a word (k-mer) in constant time.



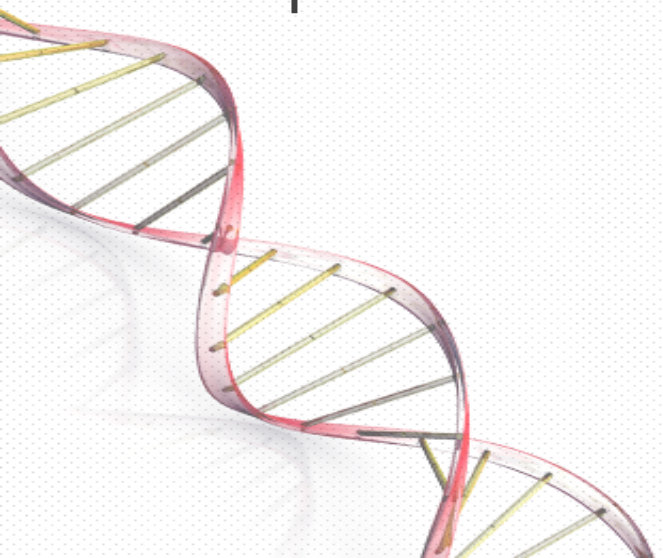
Hashing

- Hashing is an indexing technique that enables fast search by **computing index directly** based on the key



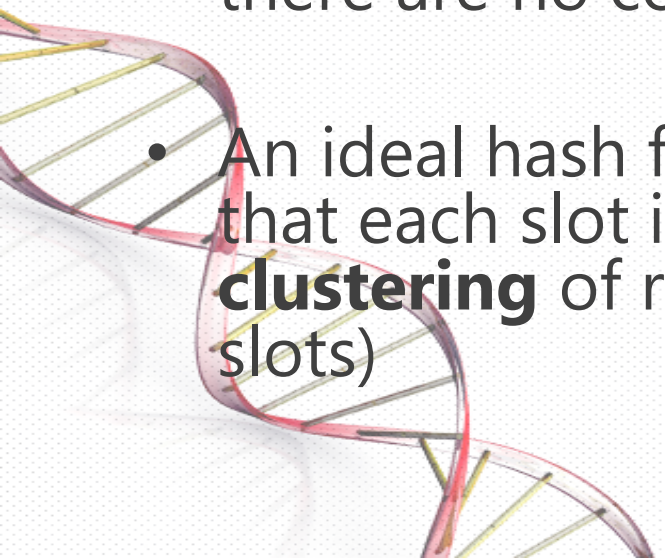
Terminologies

- The process of finding a record using some computation to map its key value to a position in the array is called **hashing**.
- The function that maps key values to positions is called a **hash function (h)**.
- The array that holds the hash table is called the **hash table (HT)**.
- A position in the hash table is also known as a **slot**.



Hash functions and collisions

- Typically there are many more values in the key range than there are slots in the hash table.
- Given a hash function h and two keys k1 and k2, if h(k1)=h(k2)=β, we say that k1 and k2 have a **collision** at slot β under hash function h.
- **Perfect hashing** is a system in which records are hashed such that there are no collisions (e.g., indexing k-mers when k is small).
- An ideal hash function stores the actual records in the collection such that each slot in the hash table has equal probability of being filled; but **clustering** of records happens (many records hash to only a few of the slots)

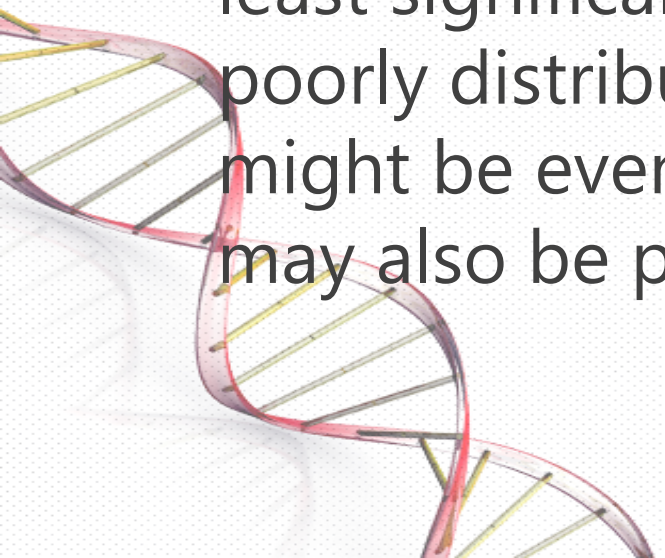


A simple hash function for integers

- A function used to hash integers to a table of 16 slots

```
int h(int x) {
        return (x % 16)
}
```

- The value returned by this hash function depends solely on the least significant four bits of the key. These bits are likely to be poorly distributed (as an example, a high percentage of the keys might be even numbers, so the low order bit is zero), so the result may also be poorly distributed.



A simple hash function for k-mers

- The strings are DNA sequences

```
int h(String x, int k){
        b2int = {'A':0, 'T':1, 'C':2, 'G':3}
        add = 1
        idx = 0
        for i = k to 1{
                b = x[i]
                idx = idx + base2int[b] x add
                add = add x 4
        }
        return idx }
```

k=1

index	key
0	**A**
1	T
2	C
3	G

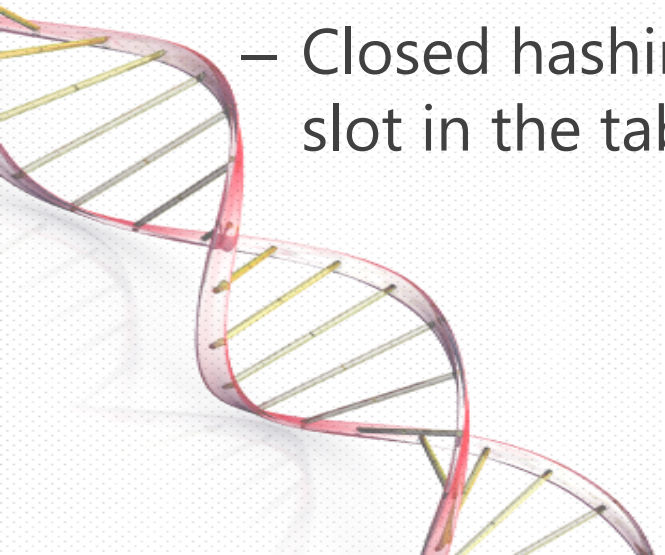
k=2

index	key
0	**AA**
1	**AT**
2	**AC**
3	**AG**
4	TA
5	TT
6	TC
7	TG
8	**CA**
9	**CT**
10	**CC**
11	**CG**
12	GA
13	GT
14	GC
15	GG

How many slots in the table for k = 3?
k = 40?

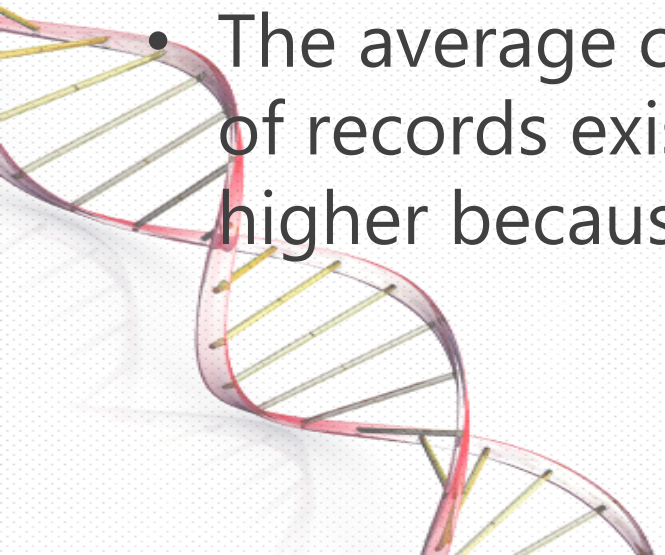
Collision resolution

- While the goal of a hash function is to minimize collisions, some collisions are unavoidable in practice.
- Hashing implementations must include some form of collision resolution policy.
- Two class of collision resolution techniques:
  - Open hashing (separate chaining)—collisions are stored outside the table
  - Closed hashing—collisions result in storing one of the records at another slot in the table



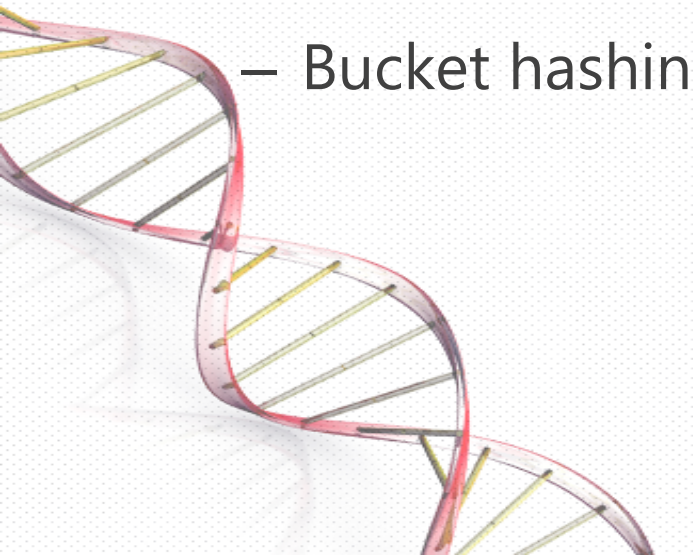
Open hashing

- The simplest form of open hashing defines each slot in the hash table to be the head of a linked list. All records that hash to a particular slot are placed on that slot's linked list
- Records within a slot's list can be ordered in several ways: by insertion order, by key value order, or by frequency-of-access order
- The average cost for hashing should be $\Theta(1)$; however, if clustering of records exists, then the cost to access a record can be much higher because many elements on the linked list must be searched



Closed hashing

- Closed hashing stores all records directly in the hash table.
- A collision resolution policy must be built to determine which slot to use when collision is detected.
- The same policy must be followed during search as during insertion.
- Some common closed hashing
  - Bucket hashing --- overflow goes to an overflow bucket



Example

Caragea *et al*. *Proteome Science* 2012, **10**(Suppl 1):S14
http://www.proteomesci.com/content/10/S1/S14

PROTEOME SCIENCE

PROCEEDINGS Open Access

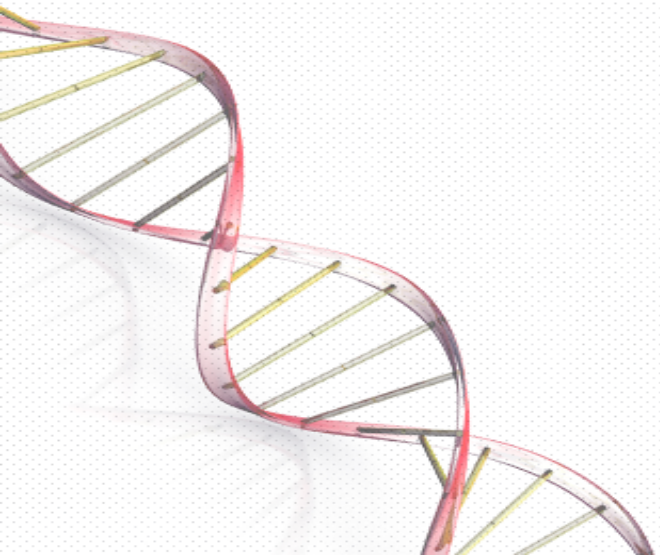
Protein sequence classification using feature hashing

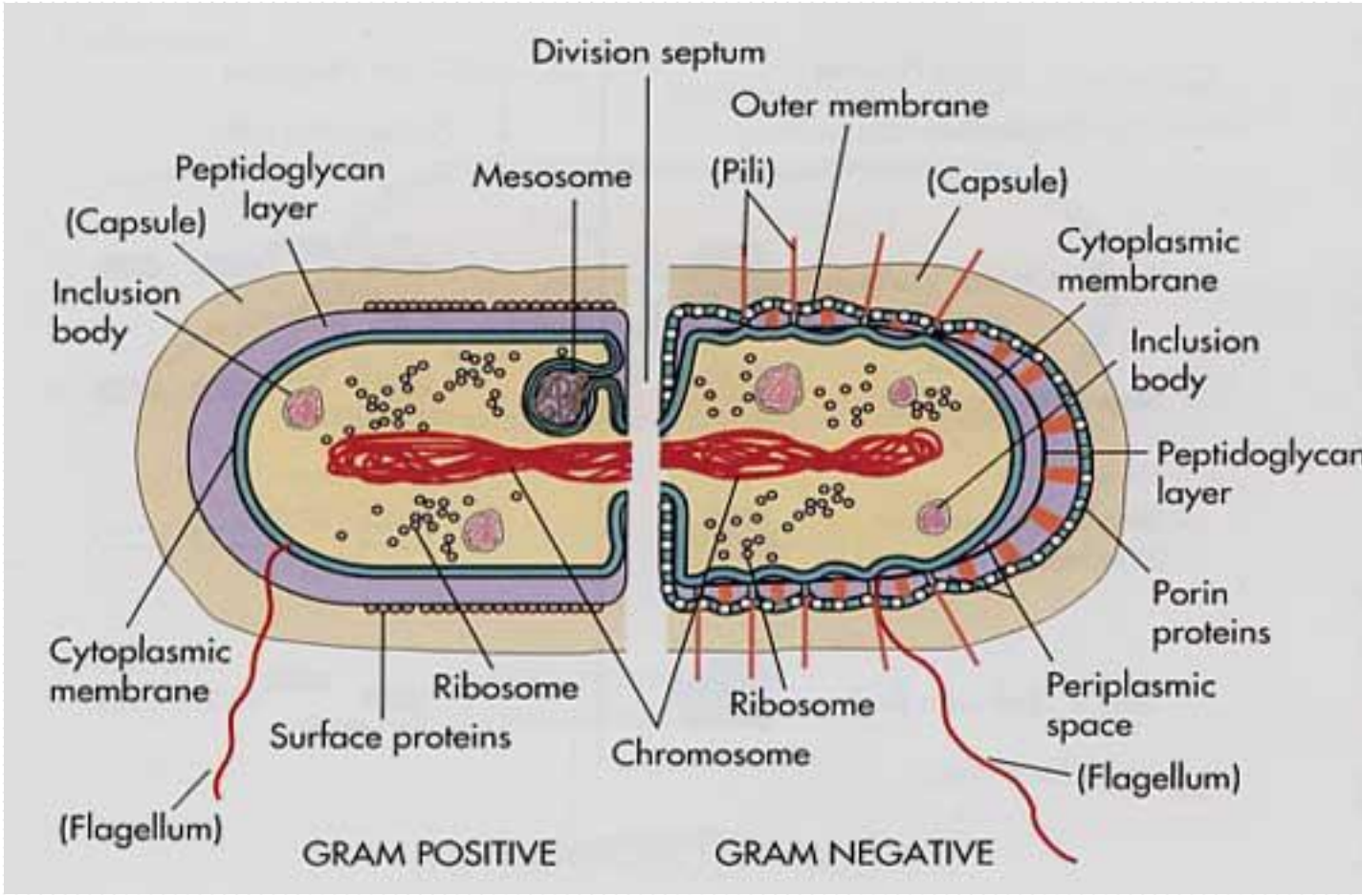
Cornelia Caragea[1*], Adrian Silvescu[2], Prasenjit Mitra[1,2]

From IEEE International Conference on Bioinformatics and Biomedicine 2011
Atlanta, GA, USA. 12-15 November 2011

Datasets

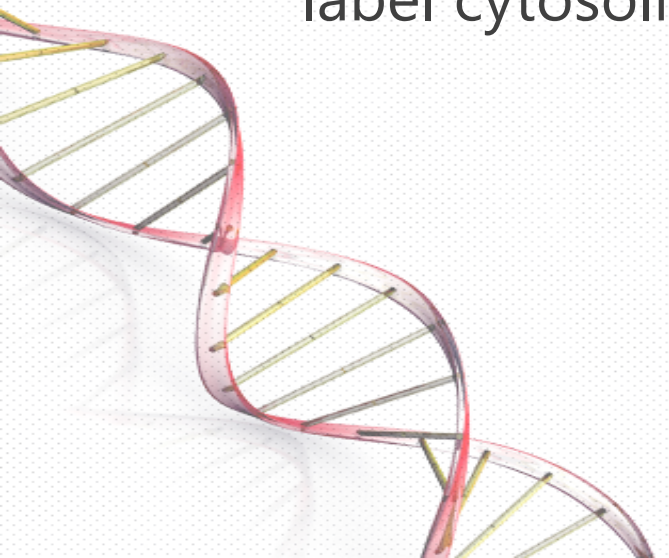
- psortNeg data set
  - cytoplasm(278)
  - cytoplasmic membrane (309)
  - periplasm(276)
  - outer membrane (391)
  - extracellular (190)

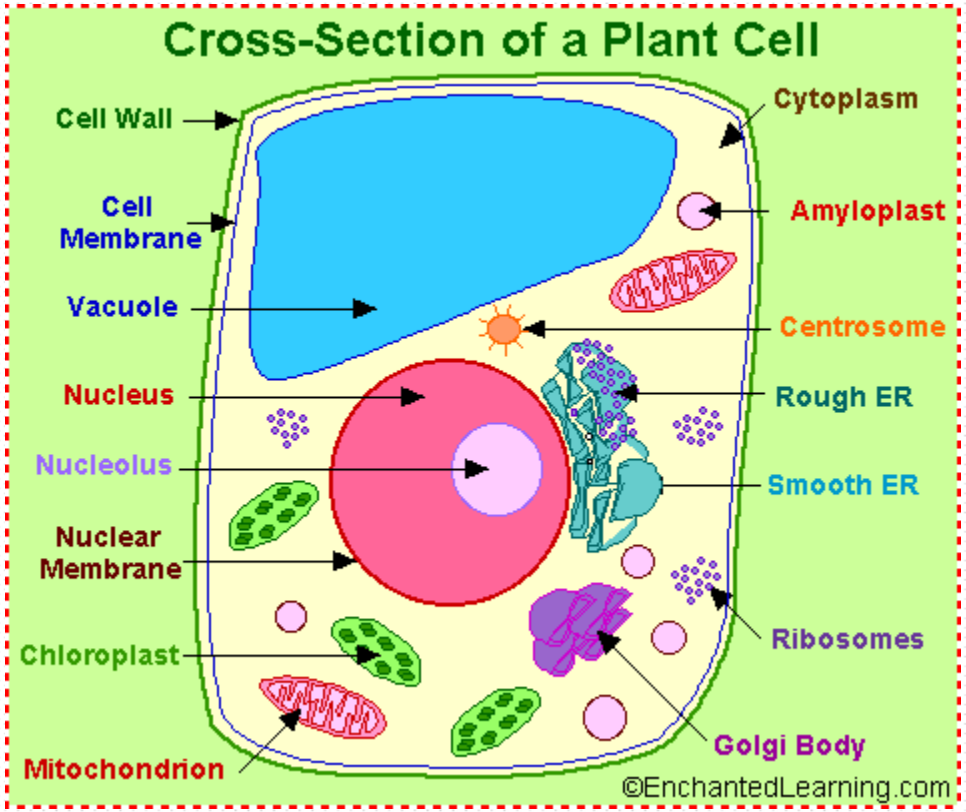




Datasets

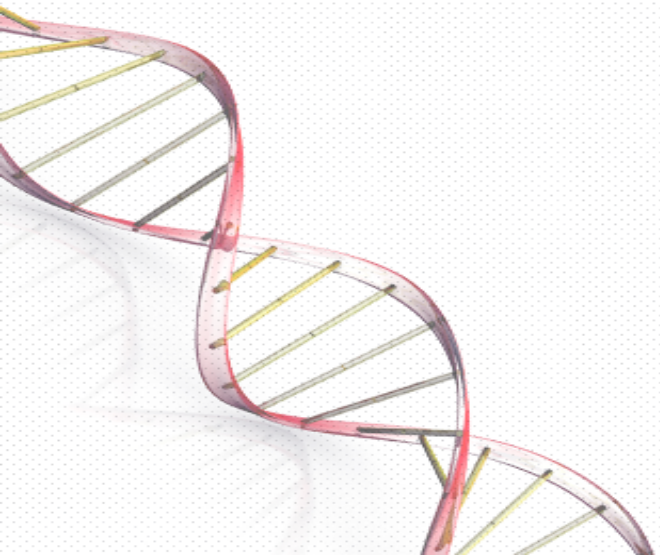
- Plant dataset
  - Chloroplast (141)
  - Mitochondrial (368)
  - Secretory pathway/signal peptide (269)
  - Other (consisting of 54 proteins with label nuclear and 108 examples with label cytosolic)

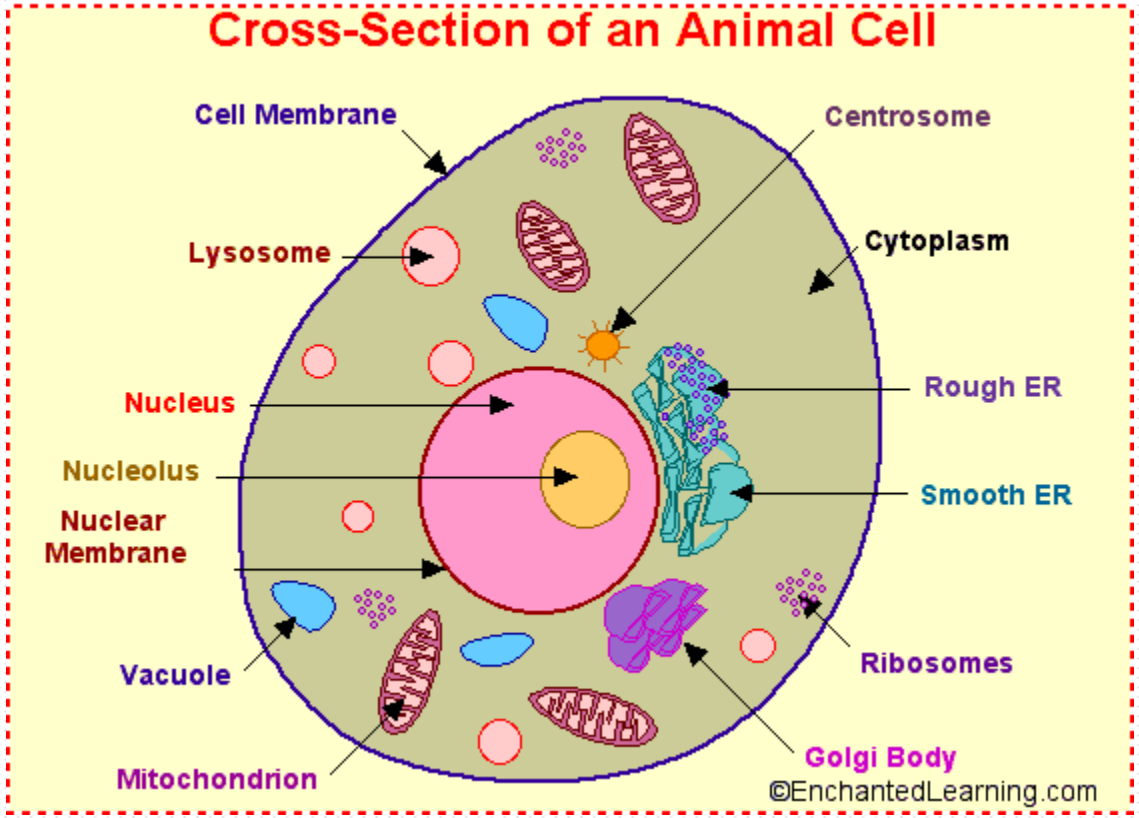




Datasets

- Non-plant
  - Mitochondrial (361)
  - Secretory pathway/signal peptide (715)
  - Other (consisting of 1224 proteins labeled nuclear and 438 proteins labeled cytosolic)





Feature Hashing

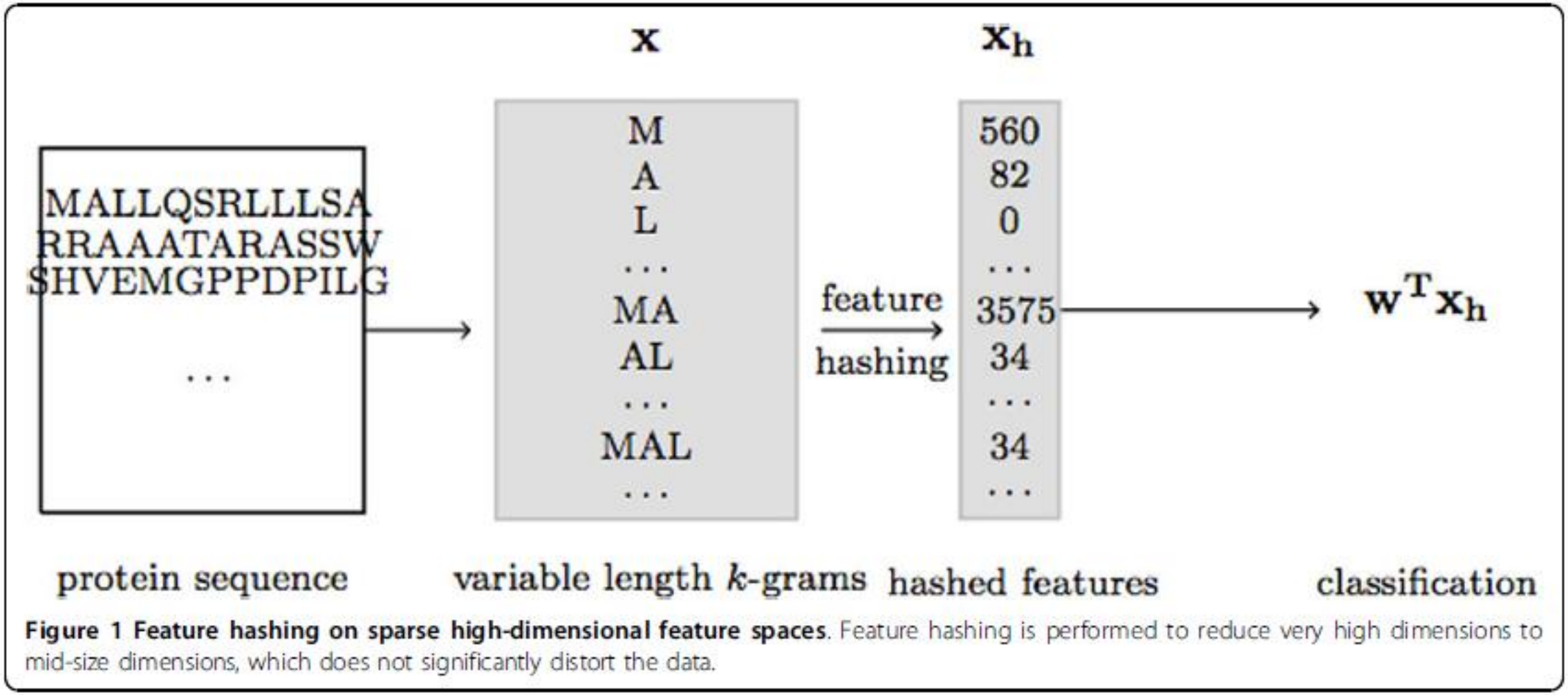


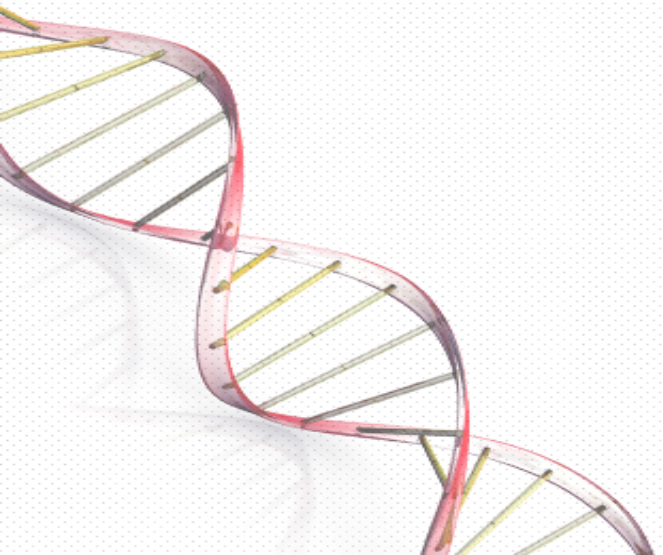
Figure 1 Feature hashing on sparse high-dimensional feature spaces. Feature hashing is performed to reduce very high dimensions to mid-size dimensions, which does not significantly distort the data.

Results

Table 1 Comparison of fixed-length with variable-length *k*-gram representations.

Bag of fixed or variable length *k*-grams	non-plant	
	Accuracy %	# features
1-grams	71.21	20
2-grams	70.85	400
3-grams	79.80	7999
4-grams	79.03	146598
(1-2)-grams	70.56	420
(1-3)-grams	79.69	8419
(1-4)-grams	**82.83**	**155017**
(1-5)-grams	80.09	950849

The performance of SVM classifiers trained using feature hashing on fixed length, 1-, 2-, 3-, 4-gram representations, as well as variable length, (1-2)-, (1-3)-, (1-4)-, (1-5)-grams representations, where the hash size is set to 2^{22}, on the non-plant data set.



Results

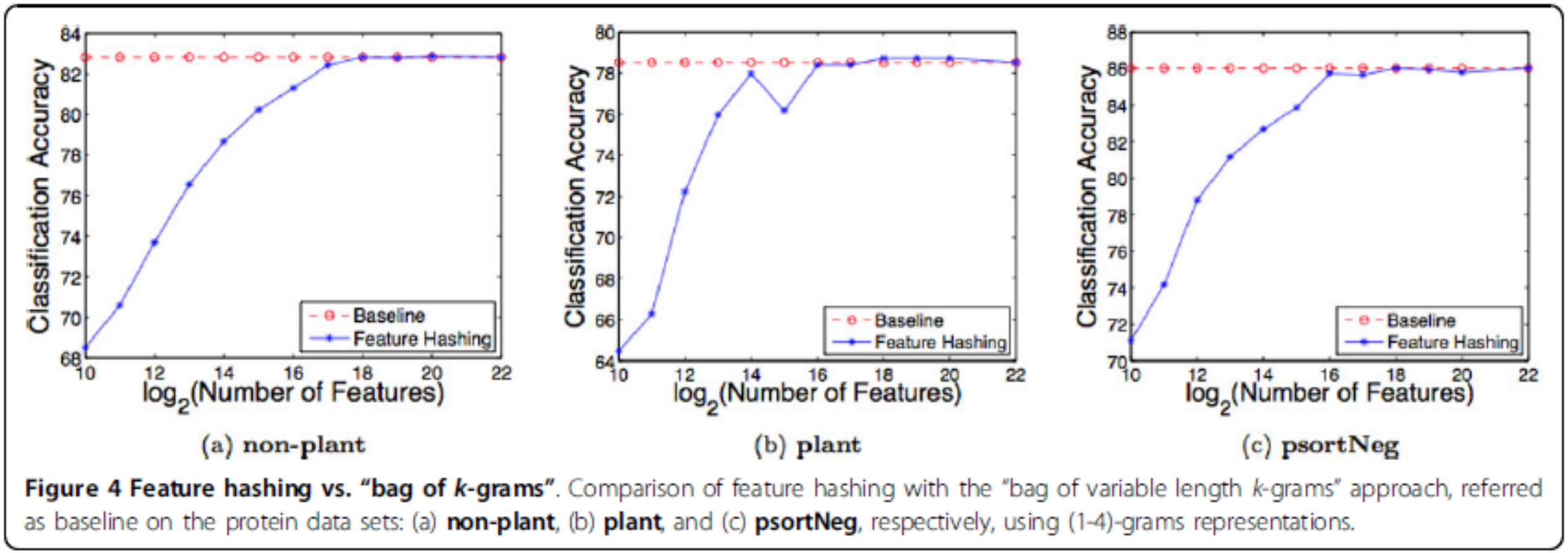


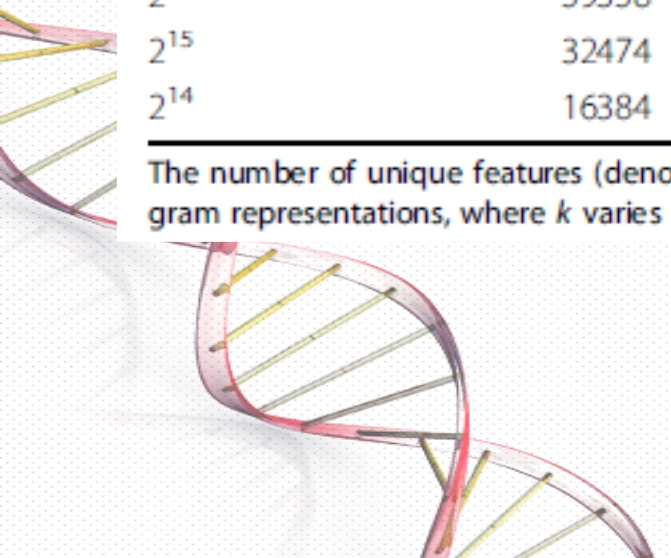
Figure 4 Feature hashing vs. "bag of *k*-grams". Comparison of feature hashing with the "bag of variable length *k*-grams" approach, referred as baseline on the protein data sets: (a) **non-plant**, (b) **plant**, and (c) **psortNeg**, respectively, using (1-4)-grams representations.

Results

Table 2 The number of variable-length *k*-grams and the rate of hash collisions for various hash sizes.

Value of *b*	non-plant		plant		psortNeg	
	# features	Collisions %	# features	Collisions %	# features	Collisions %
2^{22}	155017	0	111544	0	124389	0
2^{20}	153166	1.21	110236	1.18	122894	1.22
2^{19}	147223	5.29	107299	3.95	118871	4.64
2^{18}	132754	16.30	99913	11.43	109535	13.22
2^{17}	99764	45.04	82141	31.38	87618	35.66
2^{16}	59358	78.53	53616	64.29	55555	68.85
2^{15}	32474	95.80	31788	89.56	32075	92.02
2^{14}	16384	100	16384	100	16384	100

The number of unique features (denoted as # features) and the rate of collisions on non-plant, plant, and psortNeg data sets, respectively, for variable length *k*-gram representations, where *k* varies from 1 to 4.



References

- Lecture notes of Colin Dewey @ University of Wisconsin-Madison
- Lecture notes of Arne Elofsson @ Stockholm University
- Lecture notes of Yuzhen Ye @ Indiana University