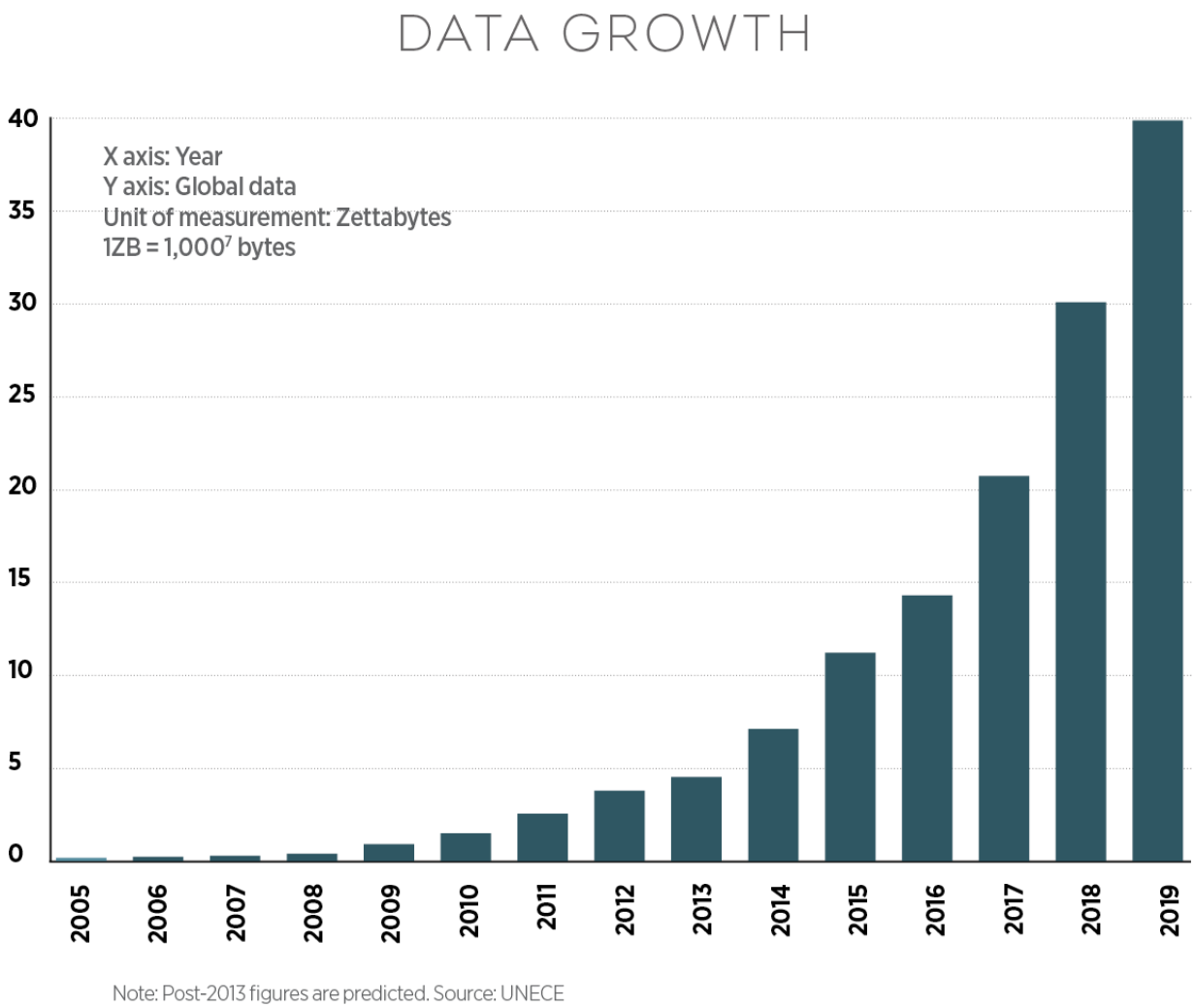
METHODS OF KNOWLEDGE ENGINEERING

PROJECT SUMMARY

Janusz Tomasik, AGH UST, summer 2016

Table of contents

1. Introduction



1 Zettabyte = 10^9 Terabytes

Big Data

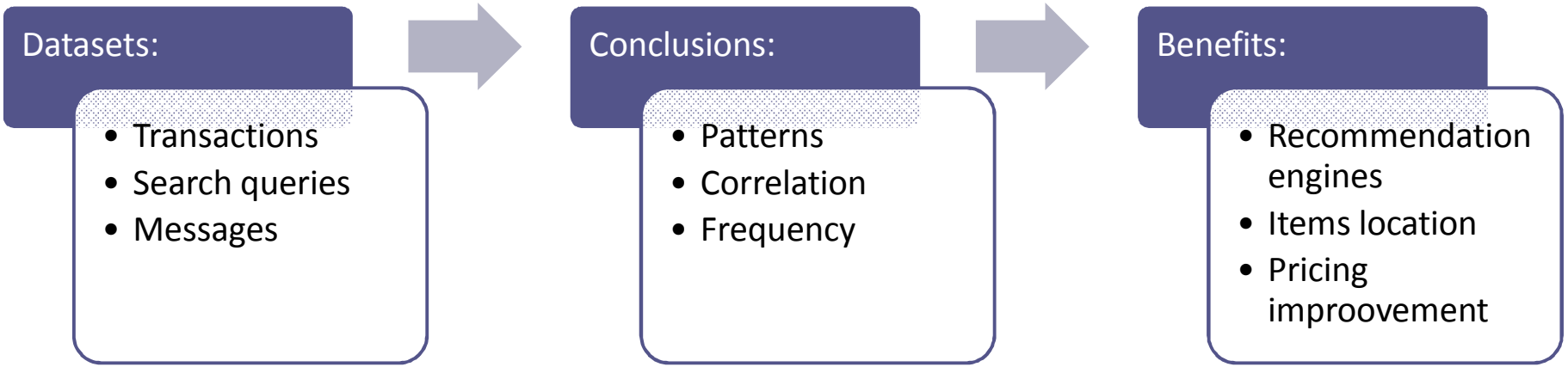
- **2.7 Zettabytes of data existed in the digital world in 2012.** [1]

- **Only 0.5% of the data was analyzed.** [2]

[1] https://www.marketingtechblog.com/ibm-big-data-marketing/

[2] http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf

2. Data Exploration



Definitions

Support - the frequency (in percentage) that an item occurs in the transactions

Example: milk - occurs in 8 transactions out of 10 => Support(milk) = 80%

Confidence - a conditional probability p(X|Y) (if a transaction has X, what is the probability that it has Y)

Example:

transaction 1: milk, coffee, cheese

transaction 2: milk, sugar

transaction 3: coffee, cheese

Confidence (milk|sugar) is 50%

Association rules - X -> Y (s,c) where s is support (X) and c is confidence(X|Y)

Simulation

How to run the simulation

Command line:

(your_directory) java –jar association_rules.jar

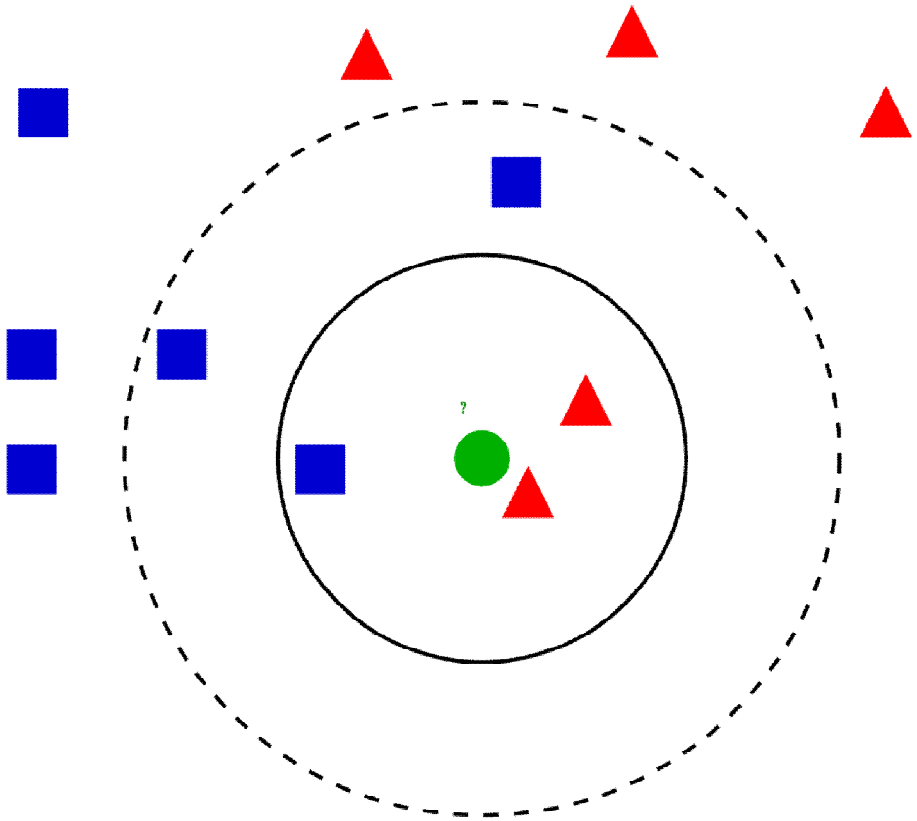
Print the dataset:

```
Choose an option from the menu. Enter 'q' to quit.
1 - Print the dataset
2 - Print support values for all products
3 - Rate the frequency of the products
4 - Get all associations above choosen support and confidence
1
ID        coffee    milk      sugar     nuts      eggs      bread     butter    honey     cereals   cheese
1         kawa      mleko     cukier    orzeszki
2         kawa                cukier              jajka
3         kawa                cukier                        chleb     maslo                         ser
4                             cukier    orzeszki  jajka                         miod      platki
5                   mleko                         jajka                maslo
6         kawa                          orzeszki                      chleb
7                   mleko                                                       miod      platki
8                                                   jajka               chleb     maslo               ser
9                   mleko                                             chleb                         ser
10        kawa      mleko               orzeszki
```

Print all association rules with support>= 55 and confidence >= 60:

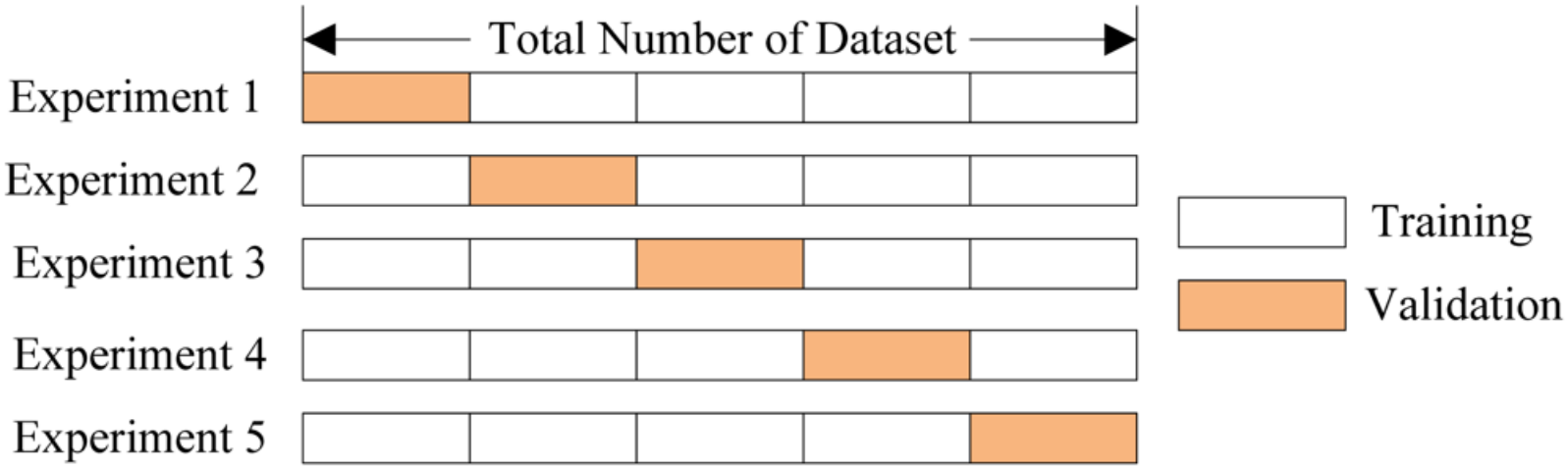
```
4
Type frequency level:
55
Type confidence level:
60
[Product1, Product2, Support, Confidence]
[coffee, sugar, 60.0, 60.0]
[coffee, nuts, 60.0, 60.0]
[sugar, coffee, 60.0, 75.0]
[nuts, coffee, 60.0, 75.0]
```

3. The kNN method with cross-validation



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Cross-validation

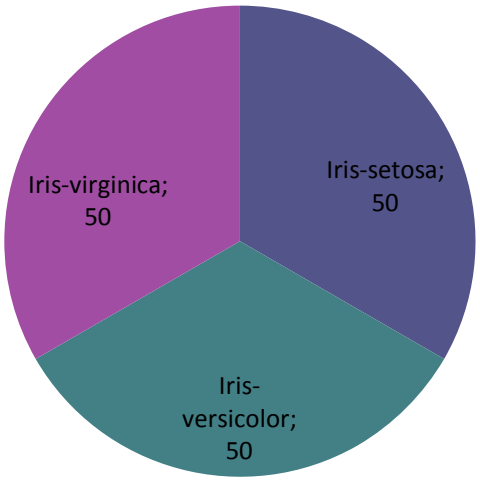


http://stackoverflow.com/questions/31947183/how-to-implement-walk-forward-testing-in-sklearn

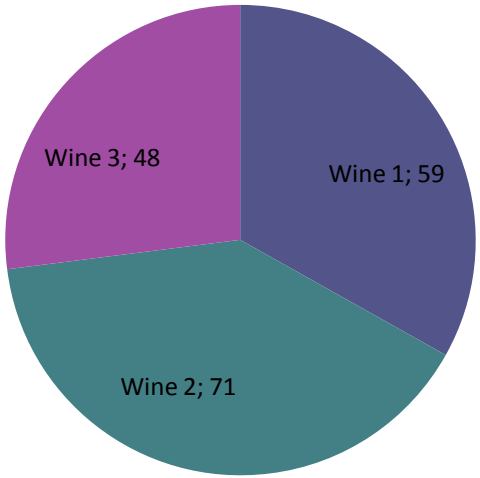
Datasets:

dataset:	records:	classes:	parameters:
IrisDataAll	150	3	4
Wine	178	3	13
YeastShort	309	10	8

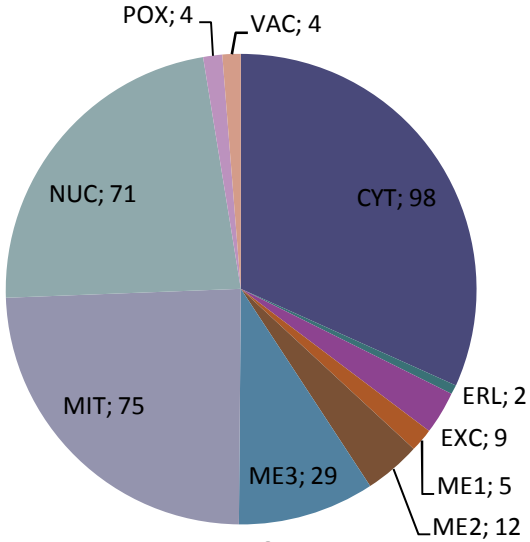
Classes distribution:



IrisDataAll



Wine



YeastShort

Simulation

How to run the simulation

Command line:

(your_directory) java –jar knn-classification-method.jar

(your_directory) java – jar knn_with_cross_validation.jar

Print the dataset:

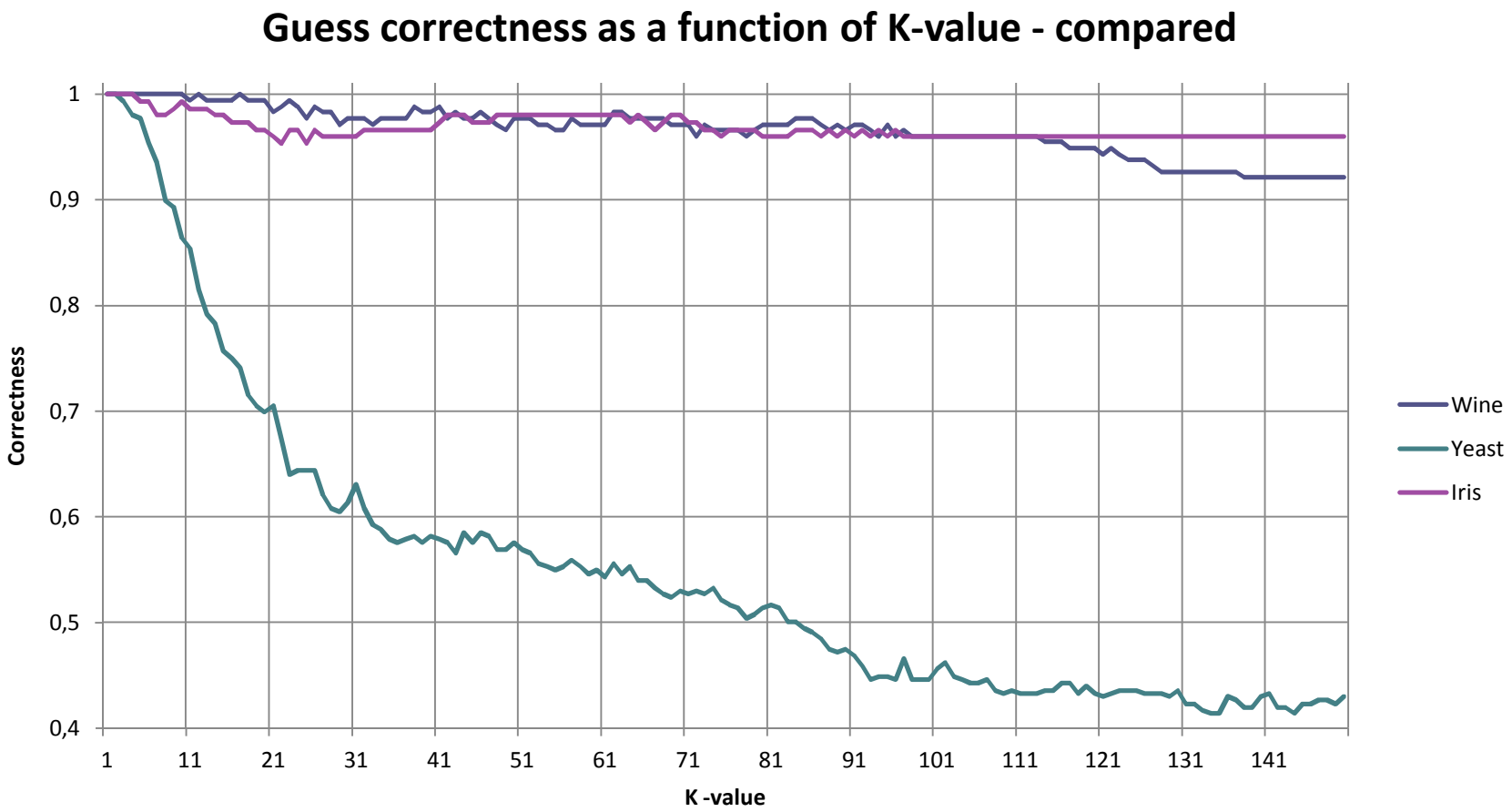
```
#---Dataset: IrisDataAll.csv---#
#---Size: 150 records---#

Choose an option from the menu. Enter 'q' to quit.
1 - Print the whole dataset (parameters normalized)
2 - Print the whole dataset - original
3 - Perform cross-validation
4 - Change dataset
1
leaf-length     leaf-width      petal-length    petal-width     class
0.6455          0.7954          0.2028          0.08            Iris-setosa
0.6202          0.6818          0.2028          0.08            Iris-setosa
0.5949          0.7272          0.1884          0.08            Iris-setosa
0.5822          0.7045          0.2173          0.08            Iris-setosa
0.6329          0.8181          0.2028          0.08            Iris-setosa
0.6835          0.8863          0.2463          0.16            Iris-setosa
0.5822          0.7727          0.2028          0.12            Iris-setosa
```

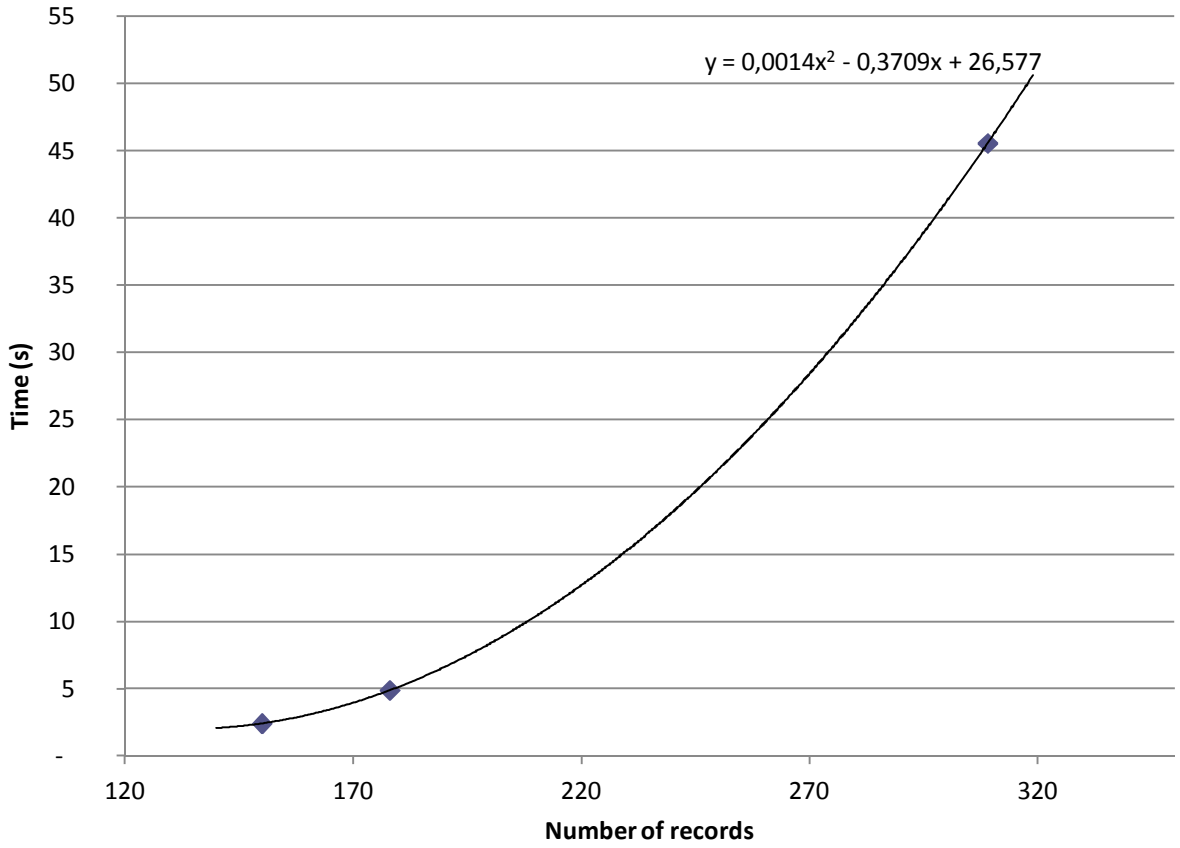
Perform cross-validation

```
Choose an option from the menu. Enter 'q' to quit.
1 - Print the whole dataset (parameters normalized)
2 - Print the whole dataset - original
3 - Perform cross-validation
4 - Change dataset
3
k       correct
1       1.0
2       1.0
3       1.0
4       1.0
5       1.0
6       0.986
7       1.0
8       0.98
9       0.986
```

Guess comparison:



Calculations performance:



Observations:

1. **Guess correctness generally decreases non-monotonically with an increasing K-value**

2. **Guess correctness gets worse if the classes are not even distributed**

3. **The performance of kNN method implementation is $O(n^2)$**

4. Self-Organizing Maps

- **Kohonen's SOM enable to represent multidimensional data in fewer dimensions, i.e. two-dimensional**
- **unsupervised learning method**
- **one node can map multiple objects**

Simulation

How to run the simulation

Command line:

(your_directory) java –jar SOM.jar

SOM after learning:

```
#---Dataset: IrisDataAll.csv---#
#---Size: 150 records---#

Choose an option from the menu. Enter 'q' to quit.
1 - Print the whole dataset - original
2 - Print the SOM before learning (one dimension)
3 - Print the SOM after learning (all dimensions)
4 - Change dataset
3
w[0]: 5.411 w[0]: 5.477 w[0]: 5.5   w[0]: 5.536
w[0]: 5.358 w[0]: 5.442 w[0]: 5.474 w[0]: 5.506
w[0]: 5.254 w[0]: 5.344 w[0]: 5.390 w[0]: 5.398
w[0]: 4.7   w[0]: 5.207 w[0]: 5.269 w[0]: 5.310
-------------------------------------
w[1]: 2.684 w[1]: 2.622 w[1]: 2.6   w[1]: 2.624
w[1]: 2.717 w[1]: 2.649 w[1]: 2.623 w[1]: 2.622
w[1]: 2.789 w[1]: 2.723 w[1]: 2.693 w[1]: 2.702
w[1]: 3.2   w[1]: 2.826 w[1]: 2.792 w[1]: 2.786
-------------------------------------
w[2]: 4.040 w[2]: 4.307 w[2]: 4.4   w[2]: 4.435
w[2]: 3.872 w[2]: 4.181 w[2]: 4.300 w[2]: 4.392
w[2]: 3.525 w[2]: 3.834 w[2]: 3.983 w[2]: 3.981
w[2]: 1.6   w[2]: 3.355 w[2]: 3.536 w[2]: 3.624
-------------------------------------
w[3]: 1.095 w[3]: 1.174 w[3]: 1.2   w[3]: 1.234
w[3]: 1.026 w[3]: 1.129 w[3]: 1.170 w[3]: 1.235
w[3]: 0.893 w[3]: 1.005 w[3]: 1.066 w[3]: 1.083
w[3]: 0.199 w[3]: 0.834 w[3]: 0.915 w[3]: 0.977
-------------------------------------
```

SOM before learning:

```
#---Dataset: IrisDataAll.csv---#
#---Size: 150 records---#

Choose an option from the menu. Enter 'q' to quit.
1 - Print the whole dataset - original
2 - Print the SOM before learning (one dimension)
3 - Print the SOM after learning (all dimensions)
4 - Change dataset
2
w[0]: 0.364 w[0]: 0.681 w[0]: 0.603 w[0]: 0.667
w[0]: 0.478 w[0]: 0.553 w[0]: 0.258 w[0]: 0.976
w[0]: 0.865 w[0]: 0.509 w[0]: 0.771 w[0]: 0.986
w[0]: 0.369 w[0]: 0.205 w[0]: 0.938 w[0]: 0.741
-------------------------------------
```

5. Associated Graph Data Structure

- **A passive data structure, which can substitute operations like: filtering, searching or ordering by providing them in O(1)**
- **No duplicates or excess data**
- **Faster data access**

Simulation

How to run the simulation

Command line:

(your_directory) java –jar AGDS.jar

(your_directory) java –jar AGDS_DB.jar

Finding the similar elements:

```
Choose an option from the menu. Enter 'q' to quit.
1 - Find similar based on given object
2 - Find similar based on given values
3 - Find objects with exact given values
4 - Find objects that have a value within given range
5 - Print all values of a parameter sorted
6 - Print the column names with indexes
7 - Change dataset
1
Type: leaf-length
4.4
Type: leaf-width
5
Type: petal-length
2
Type: petal-width
1

Most similar top3:
ObjID Wage
R45   3.138
R16   3.104
R44   3.098
Best guess: R45 Iris-setosa 5.1 3.8 1.9 0.4

Least similar top3:
ObjID Wage
R119  0.911
R123  1.136
R136  1.180
Worst guess: R119 Iris-virginica 7.7 2.6 6.9 2.3

 results in : 92ms
------------------------------------
```

Finding an element with exact values:

```
3
Type in the parameters indices (comma separated)
0,1
Type in the values (comma separated)
6.1,3

Most similar top3:
ObjID Wage
R128  2.0
R92   2.0
Best guess: R128 Iris-virginica 6.1 3 4.9 1.8

 results in : 7ms
------------------------------------
```

Tables vs Graphs

- **Tested database: US Baseball Players Season Statistics**
- **Number of records: 14 347**
- **Number of columns: 21**
- **Tested database structures:**
  - **Relational (MySQL)**
  - **Graph (AGDS)**

SELECT query

Full query:

SELECT * FROM `appearances` WHERE yearID = "1871„

Time performance:

MySQL AGH - online (ms)			AGDS with print (ms)			AGDS w/o print (ms)			MySQL - localhost (ms)		
37,3	0,7	0,8	22	23	19	0	0	1	250	125	250

Results – SQL 115 rows:

Results – AGDS 115 rows:

SELECT query with conjunction (AND)

Full query:

SELECT * FROM `appearances` WHERE yearID = "1871" AND teamID = "CH1" AND G_p = "0" AND G_defense = "26„

Time performance:

MySQL AGH - online (ms)			AGDS with print (ms)			AGDS w/o print (ms)			MySQL - localhost (ms)		
39,8	37,2	41,9	1	1	1	1	1	1	249	328	281

Results – SQL 2 rows:

yearID	teamID	lgID	playerID	G_all	GS	G_batting	G_defense	G_p	G_c	G_1b	G_2b	G_3b	G_ss	G_lf	G_cf	G_rf	G_of	G_dh	G_ph	G_pr
1871	CH1	NA	duffyed01	26		26	26	0	0	0	0	1	26	0	0	0	0			
1871	CH1	NA	mcatebu01	26		26	26	0	0	26	0	0	0	0	0	0	0			

Results – AGDS 2 rows:

```
1871    CH1 NA  duffyed01   26      26  26  0   0   0   0   1   26  0   0   0   0
1871    CH1 NA  mcatebu01   26      26  26  0   0   26  0   0   0   0   0   0   0
Time in ms: 1
-----------------------------------
```

SELECT query with conjunction (AND) 2nd test

Full query:

SELECT * FROM `appearances` WHERE yearID = "1908" AND lgID = "NL„

Time performance:

MySQL AGH - online (ms)			AGDS with print (ms)			AGDS w/o print (ms)			MySQL - localhost (ms)		
39,3	28,4	291	46	47	41	1	1	1	1232	47	296

Results – SQL 233 rows:

Results – AGDS 233 rows:

SELECT query with disjunction (OR)

Full query:

SELECT * FROM `appearances` WHERE yearID = "1871" OR teamID = "CH1" OR G_p = "0" OR G_defense = "26„

Time performance:

MySQL AGH - online (ms)			AGDS with print (ms)			AGDS w/o print (ms)			MySQL - localhost (ms)		
91,8	0,8	0,7	994	1331	1275	11	14	11	16	32	15

Results – SQL 9312 rows

Results – AGDS 9312 rows

Observations:

1. **The AGDS gives an edge over SQL in conjuction (AND) SELECT query cases**

2. **The performed tests have shown a correct AGDS queries implementation (not proved yet!)**

3. **Constant access time for simple AGDS SELECT queries**

6. Summary

- **Effectively handling Big Data will be the challenge of the next years**
- **Solutions: Both hardware (i.e. quantum computers) & software (better data structures and algorithms)**
- **Data exploration (data mining) and machine learning requires a sophisticated approach.**