SE 953482 Natural Language Processing for SE
66/2
Text Extractions and representation

Asst. Prof. Pree Thiengburanathum

Announcement

- Midterm exam date 19^{th} Jan 8am
- Open book (you can bring slide, text book, notes)
- No laptop and cellphone
- 3 hrs
- 25% of your final grade

Midterm coverage

1. NLP Overview
2. Data science methodology
3. Word Tokenization, Text preprocessing
4. Text extraction methods

Where we are now

Regular expression

- Known as **regex**" or "**regexp**", are a powerful tool used in computing for pattern matching within strings.
- Benefits
  - *Searching and Extracting* – emails, URLs, etc.
  - *Data Validation* – checking input format
  - *String Manipulation*- split string

Finding patterns with regular expressions

- Callouts are more than just tokens starting with '@'
- @username @UK_Spokesperon
- Match something after '@'
  - Alphabets
  - Numbers
  - Special symbols such as '_'

Finding patterns with regular expressions

Callouts

```
>>> w for w in txt10.split() if w.startswith('@')]
['@GreggJarrett:', '@']
```

Import regular expressions first

```
import re
[w for w in txt10.split(' ') if re.search('@[A-Za-z0-9_]+', w)]
['@GreggJarrett:']
```

Parsing the callout regular expression

- @[A-Za-z0-9_]+
- Starts with @
- Followed by any alphabet (upper or lower case), digit, underscore
- That repeats at least once, but any number of times

Metacharacters

Metacharacters are characters with a special meaning:

<table>
<tr><th>Character</th><th>Description</th><th>Example</th></tr>
<tr><td>[]</td><td>A set of characters</td><td>“[a-m]”</td></tr>
<tr><td>\</td><td>Signals a special sequence</td><td>“\d”</td></tr>
<tr><td>.</td><td>Any character(except newline character)</td><td>“he..o”</td></tr>
<tr><td>^</td><td>Starts with</td><td>“^hello”</td></tr>
<tr><td>$</td><td>Ends with</td><td>“world$”</td></tr>
<tr><td>* (repetitions)</td><td>Zero or more occurrences</td><td>“aix*”</td></tr>
<tr><td>+ (repetition)
? (repetition)</td><td>One or more occurrences
Zero or one occurrences</td><td>“aix+”</td></tr>
<tr><td>{}</td><td>Exactly the specified number of occurrences</td><td>“al{2}”</td></tr>
<tr><td>|</td><td>Either or</td><td>“falls|stays”</td></tr>
</table>

Metacharacters

- \d – any digit
- \D – any non-digit
- \s – any white space char, [\t\n\r\f\v]
- \S – opposite the above
- \w – Alphanumeric character , [a-zA-Z0-9_]
- \W – [^ a-zA-Z0-9_]
- \b - word boundery

Sets

A set is a set of characters inside a pair of square brackets

Set	Description
[arn]	Returns a match where one of the specified characters (a, r, or n) are present
[a-n]	Returns a match for any lower-case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a, r, and n
[0123]	Returns a match where any of specified digits(0, 1, 2, or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit number from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case

Example 1 – search() function

The search() function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

```
import re

#Check if the string starts with "The" and ends with " ChiangMai":

txt = "The rain in ChaingMai"
x = re.search("^The.*ChiangMai$", txt)

if (x):
  print("YES! We have a match!")
else:
  print("No match")
```

Example 2 – findall() function

- The findall() function returns a list containing all matches.
- import re

```
#Return a list containing every occurrence of "ai":

str = "The rain in Chaing Mai"
x = re.findall("ai", str)
print(x)
['ai', 'ai']
```

Example 2.1 findall() function

- Finding specific characters

```
>>> txt12 = 'ouagadougou'
>>> re.findall(r'[aeiou]', txt12)
['o', 'u', 'a', 'a', 'o', 'u', 'o', 'u’]
>>> re.findall(r'[^aeiou]', txt1)
['g', 'd', 'g']
```

Example 3 – sub() function

- The sub() function replaces the matches with the text of your choice:

```
import re

str = "The rain in Chaing Mai"
x = re.sub("\s", "9", str)
print(x)
```

```
The9rain9in9Chiang Mai
```

Email validation

```python
import re

# Sample string
text = "Please contact us at support@example.com or sales@example.net."

# Regular expression for matching email addresses
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

# Find all matches
emails = re.findall(email_pattern, text)

print(emails)
```

Email validation

- **\b** at the beginning of the pattern ensures that the email address is not preceded by another word character.

- text = "The cat chased the mouse."
- pattern = r'\bcat\b'

- matches = re.findall(pattern, text)
- print(matches) # Output: ['cat']

URL validation

```python
import re

# Regular expression pattern for a URL
url_pattern = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'

# Sample string
text = "Check out this website: https://www.example.com or http://example.net."

# Find all matches
urls = re.findall(url_pattern, text)

print(urls)

```

Class exercise *Regular expression practice*

- Write a regular expression to find the word "Python" in a string.
- Write a regular expression to match postal codes like 50300

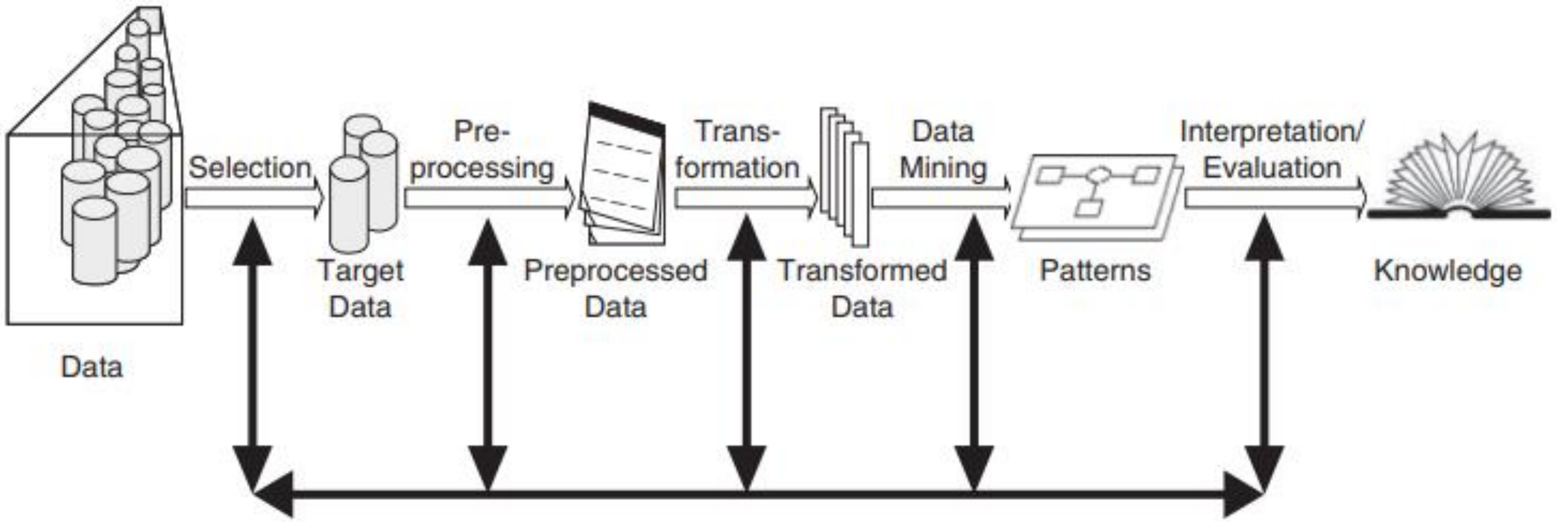


Figure 5 Overview of the steps constituting the knowledge discovery in databases (KDD) process (Fayyad *et al.*, 1996b)

Recall Processes of KDD

1. Learning application domain (initial selection)
2. Data Cleaning
3. Data Integration – where multiple data sources may be combined (heterogenous info. sources)
4. Data transformation
5. Data reduction / feature selection
6. Selecting function of data mining/ml
    1. Prediction/ classification/ associate / clustering

Recall Processes of KDD (Cont.)

7. Selecting the mining / machine learning algorithms

    Depends on the 6 step

8. Evaluation of the data mining/ml algorithm

9. Result interpretation – visualization of the model, main finding, etc.

10. Action (use of discover knowledge -> public policies, intelligent systems)

Data cleaning

- 60-70% of the time spending on cleaning data in the Data Mining processes
- “57% of data scientists regard cleaning and organizing data as the least enjoyable part of their work and 19% say this about collecting data sets” (Forbes, 2016)

Source: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#1852a116f637, (Accessed, August 2018)

Data cleaning – Common error

- Interpretation error: person age >=300
- Inconsistencies : Gender female = [Female, F, Fe]

Error pointing to false value within one dataset

Error	Solution
Redundant white space	Use string functions
Impossible values	Manual overrules
Mistakes during data entry	Manual overrules
Missing values	Remove observation or value
Outliers	Validate and, if erroneous, treat as missing value (remove or insert)

Data cleaning - Common error

Error pointing to inconsistencies between data sets

Error	solution
Deviations from a code book	Match on keys or else use manual overrules
Different units of measurement	Recalculate

- Ignored the sample/case and variables/features
  - case that contain more than 15 % of miss values should be ignored.
  - Variables missing at least 10 % of data were candidates for deletion
  - Ignore the sample, usually perform when target class is missing

Data Cleaning – (cont.)

Value	Count
Male	156
Female	140
Femalw	12
Malw	10
Malee	5
F	42
M	45

If-else rule
S1 = “Female”
S2 = “Male”
If x == “F”:
X==“Female”

White space: “ Male”, M ale”, Male “

Capital mismatch: “mAle”,

Impossible value:
Check = 0<=age<=120

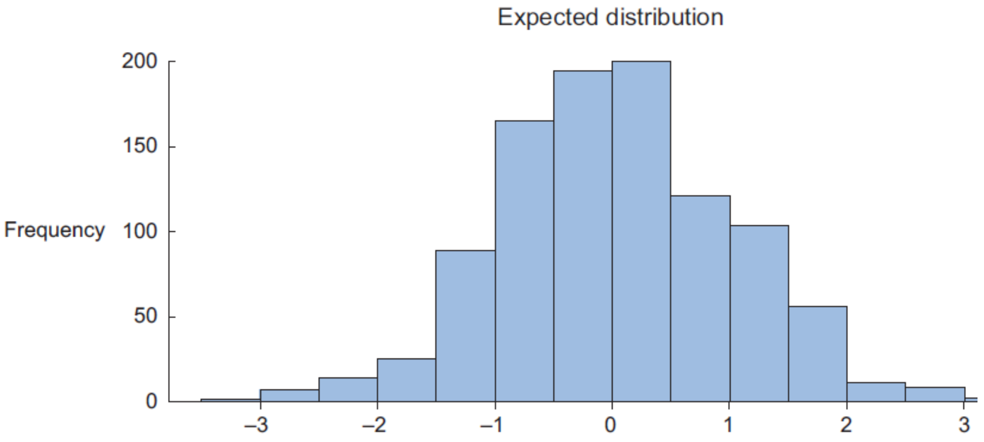
Data Cleaning- Handling missing value

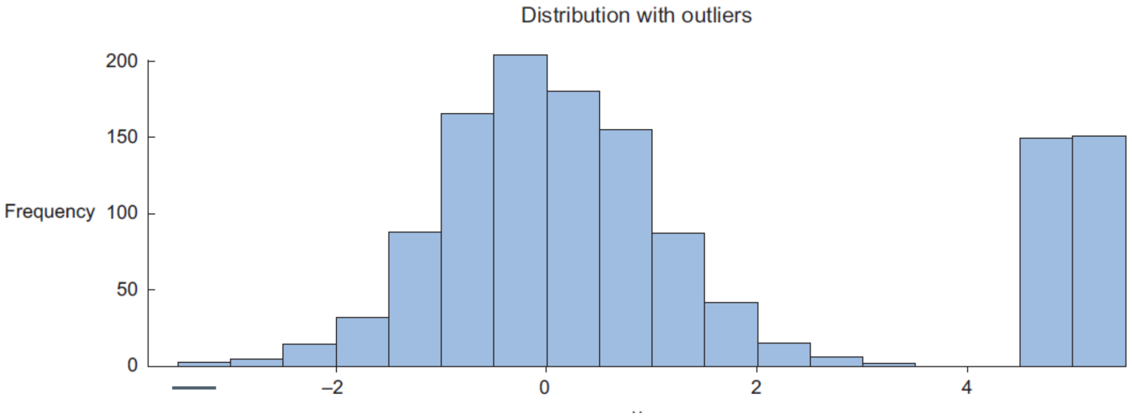
Technique	Advantage	Disadvantage
Omit the values	Easy to perform	Lose the information
Use NULL	Easy to perform	Not many algorithms can handle null values
Impute a static value such as 0 or mean	Easy to perform	Lead to false estimation from a model
Model the value	Doesn’t disturb the model too much	Hard to execute Make data assumption

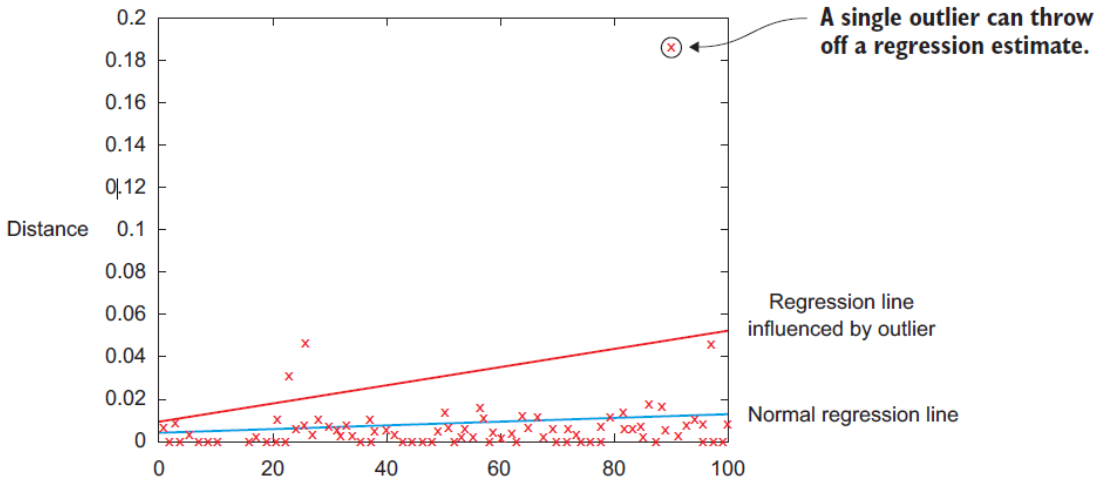
Data Cleaning – (cont.)

- Variables that are Missing At Random (MAR)
  - Use imputation methods
    - Mean or mode substitution (easy to implement)
- Identify outlier and extreme values
    - Binning approach – the most basic technique (sort data to equal bin, then smooth by mean or median
    - Semi-Automated approach - Automate script and domain expert to correct inconsistent data.
    - Clustering approach – using clustering algorithm to group common values,
- Use domain knowledge expert to correct the missing value
  - E.g. Let the weather climate expert comes to check those values.

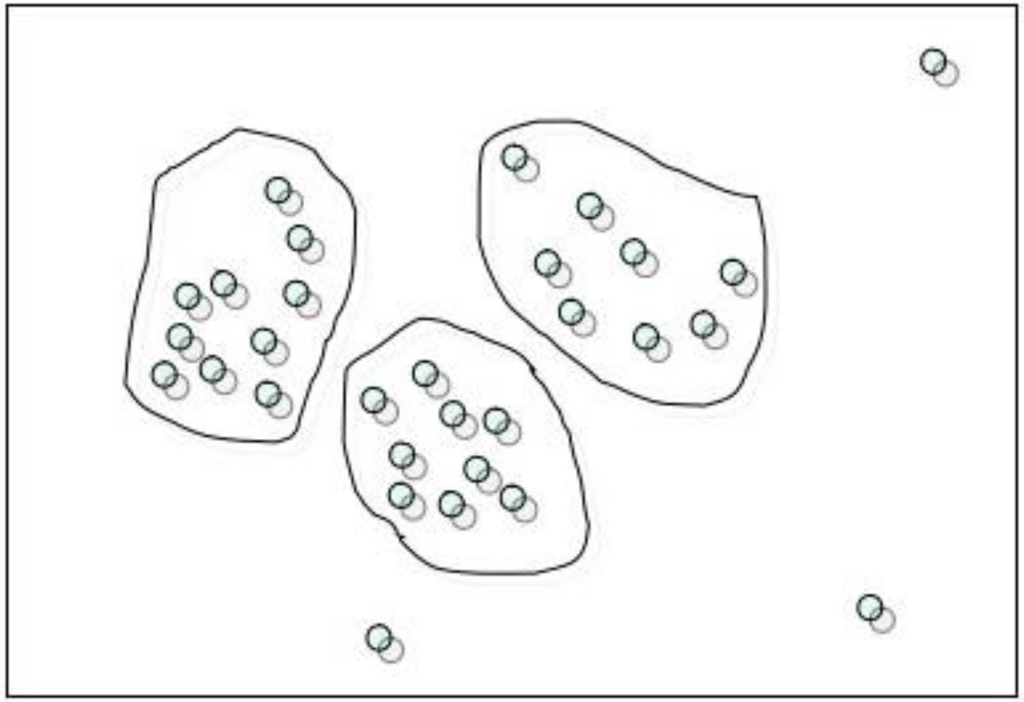
Data cleaning - outlier



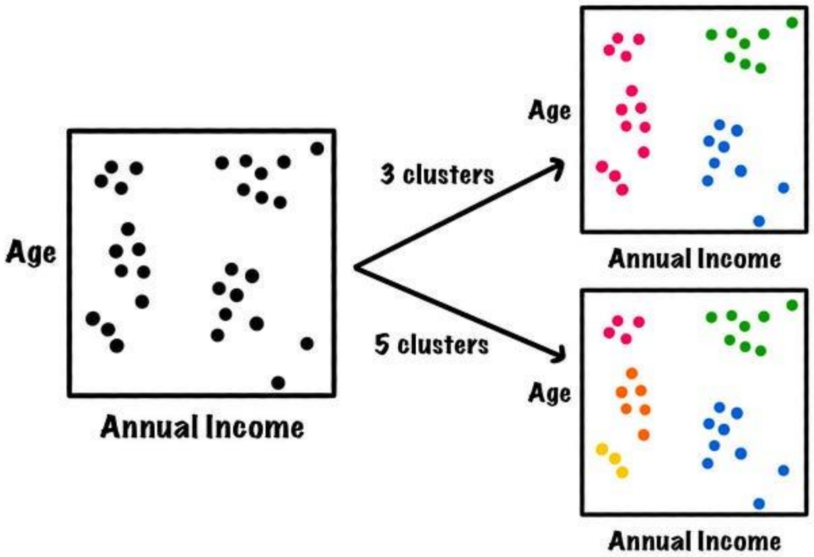


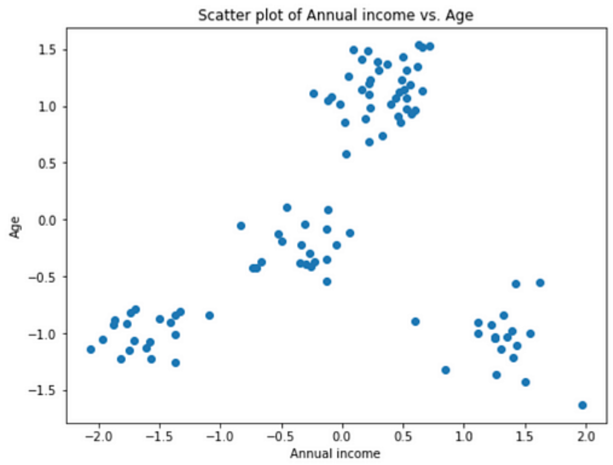


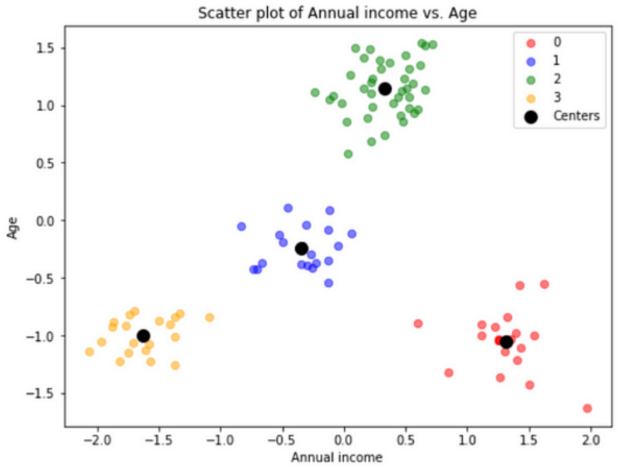
Data Cleaning – using Cluster analysis (Identify outliers/extreme values)

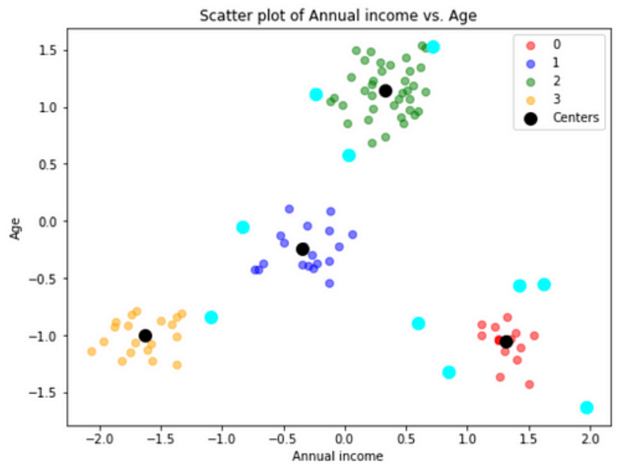


Outlier Detection Using K-means Clustering









Binning approach – Smooth data

- Sort data : age = [4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34]
- Partition into equal-depth / equal frequency bins: bin_number = 3
  - Bin1 = [4, 8, 9, 15]
  - Bin 2 = [21, 21, 24, 25]
  - Bin 3 = [26, 28, 29, 34]
- Smooth by mean: bin1= [9, 9, 9, 9] , bin 2 = [23, 23, 23, 23], bin3 = [29, 29, 29, 29]
- Smooth by bin boundaries: bin1 = [4, 4, 4, 15], bin2 = [21, 21, 25, 25], bin3= [26, 26, 26, 34]

Basic Text processing (Stop word)

- Remove most common words: and", "the", "is", "in", "on", "that", and "with".
- Noise and irreverent

Basic Text processing (Stop words)

- from nltk.corpus import stopwords
- from nltk.tokenize import word_tokenize
-
- sentence = ‘Machine learning is cool!
-
- stop_words = set(stopwords.words('english'))
- word_tokens = word_tokenize(sentence)
-
- filtered_sentence = [w for w in word_tokens if not w in stop_words]
- print(filtered_sentence)

Basic Text processing (Stemming)

- a process of transforming a word to its root form
  - Improve computing process
  - Reduce complexity

Original	**Stemming**	**Lemmatization**
New	New	New
York	York	York
is	is	**be**
the	the	the
most	most	most
densely	**dens**	densely
populated	**popul**	populated
city	**citi**	city
in	in	in
the	the	the
United	**Unite**	United
States	**State**	States

Basic Text processing (Stemming)

- import nltk
- from nltk.stem import PorterStemmer
- ps = PorterStemmer()

- sentence = "Machine Learning is cool"

- for word in sentence.split():
-   print(ps.stem(word)

Basic Text processing (Lemmatizing)

- import nltk
- from nltk.stem import WordNetLemmatizer

- lemmatizer = WordNetLemmatizer()

- print(lemmatizer.lemmatize("Machine", pos='n'))
- # pos: parts of speech tag, verb
- print(lemmatizer.lemmatize("caring", pos='v'))

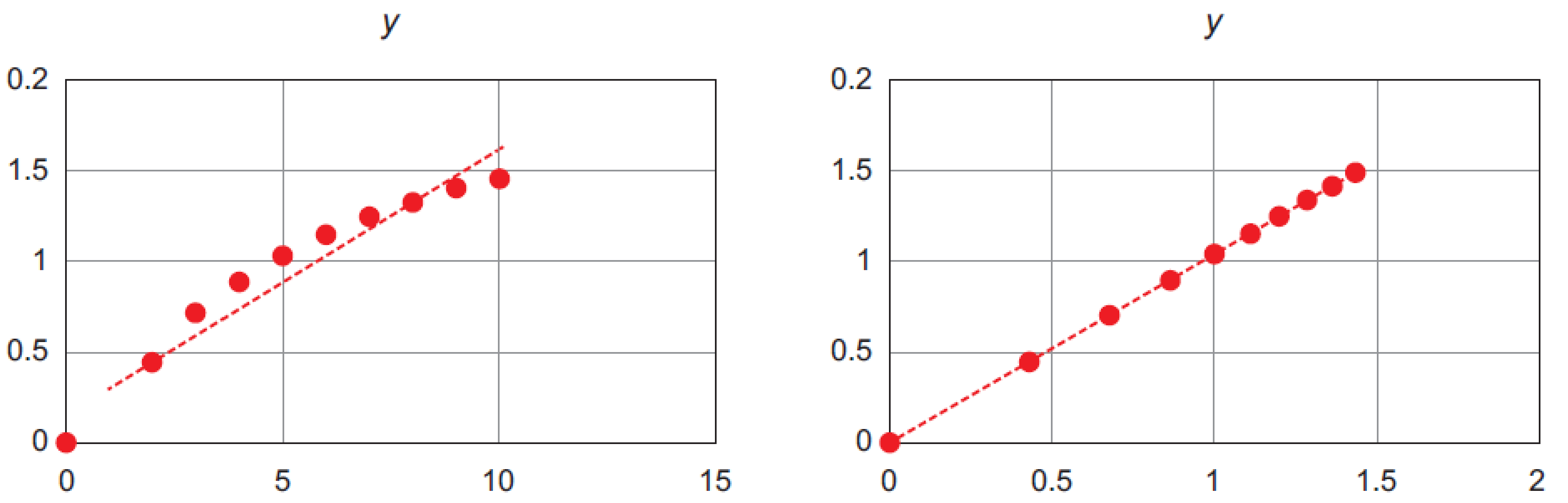
Data transformation

- Normalization
  - Scaling attribute values to fall with in a specified range (binning again)
  - Contract or replace with new attribute
    - e.g. measure 3 times, we construct average of the three column
    - e.g. we replace Celsius to Fahrenheit
    - Replace target variable majority vote
  - Dealing with derived attribute (e.g. date of employee)

Data transformation – continuous value

Log normalization

x	1	2	3	4	5	6	7	8	9	10
log(x)	0.00	0.43	0.68	0.86	1.00	1.11	1.21	1.29	1.37	1.43
y	0.00	0.44	0.69	0.87	1.02	1.11	1.24	1.32	1.38	1.46



Data transformation (cont.)

- Normalization using
  - Decimal Scaling (for NN, SVM) – move the decimal point of value of attribute
  - Min-max function (for NN, SVM) – move the attribute value in the specific ran
  - Select normalization techniques depends on machine learning algorithm and nature of data set (try and try till you get good results)

$$s' = \frac{s - Min}{Max - Min}$$

$$z = \frac{x - \mu}{\sigma}$$

Decimal scaling normalization

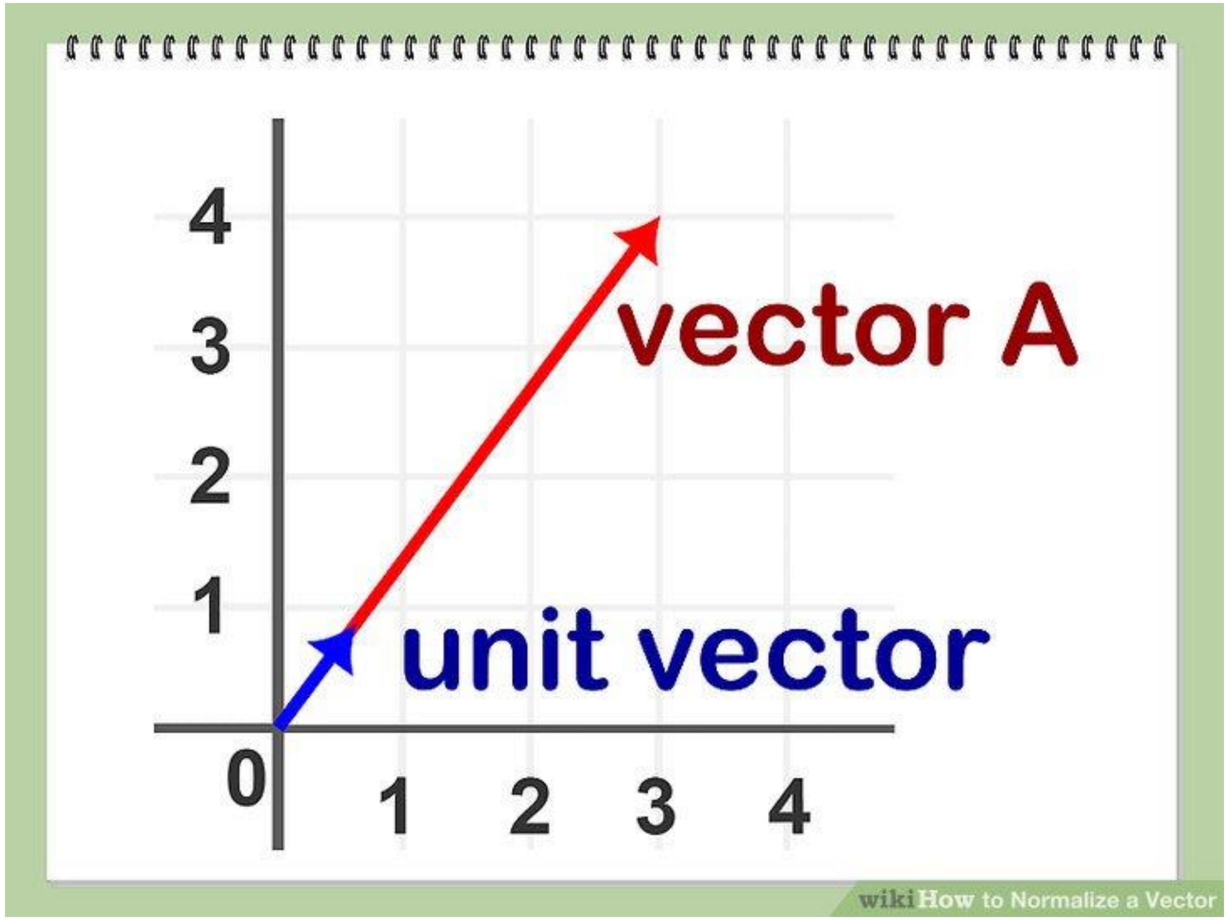
- Suppose that the recorded values of x range from − 986 to 917.
- The maximum absolute value of x is 986.
- To normalize by decimal scaling, we therefore divide each value by 1,000
- so that −986 normalizes to −0.986 and 917 normalizes to 0.917.

Min-max normalization

- Suppose that the minimum and maximum values for the feature income are 12,000 and 98,000, respectively. We would like to map income to the range 0.0,1.0 . By min-max normalization function. a value of $73,600 for income is transformed to:
-
- $\frac{73600-12000}{98000-12000}(1.0-0)+0=0.716$

$$s' = \frac{s - Min}{Max - Min}$$

Unit Vector normalize



When your feature have large range,
e.g. dot product can return and overflow,
So, scaling the vector can be benefit.

$$\vec{u} = \frac{\vec{v}}{|\vec{v}|} = \frac{(3,4)}{\sqrt{3^2+4^2}} = \frac{(3,4)}{5} = \left(\frac{3}{5}, \frac{4}{5}\right).$$

Data transformation-Discrete value

Data transformation- encoding

- Nominal features (one-of-n encoding)
- Ordinal features (Thermometer encoding)

	1 d4_23	2 a3_5	3 e1_2_5	4 d2
1	0	0	3	8
2	0	0	3	1
3	0	0	2	8
4	0	0	2	1
5	0	0	3	8
6	0	0	2	8
7	0	0	2	8
8	0	0	1	8
9	1	0	3	8
10	0	0	1	8
11	1	0	3	1
12	1	0	1	8
13	0	1	3	1
14	1	1	3	8
15	0	NaN	3	1

	1 d4_23	2 a3_5	3 e1_2_5_v1	4 e1_2_5_v2	5 e1_2_5_v3
1	0	0	0	0	1
2	0	0	0	0	1
3	0	0	0	1	0
4	0	0	0	1	0
5	0	0	0	0	1
6	0	0	0	1	0
7	0	0	0	1	0
8	0	0	1	0	0
9	1	0	0	0	1
10	0	0	1	0	0

One-of n/one-hot encoding

- Categorical variables need to be converted into forma that could provided machine learning algorithms to perform better

Compay name	Type	Price
BMW	1	220,000
FORD	2	780,000
Toyoya	3	670,000
Toyoya	3	640,000

CN_1	CN_2	CN_3	Price
1	0	0	220,000
0	1	0	780,000
0	0	1	670,000
0	0	1	640,000

Thermometer Encoding

Compay name	Type	Price
BMW	1	220,000
FORD	2	780,000
Toyoya	3	670,000
Toyoya	3	640,000

CN_1	CN_2	CN_3	Price
0	0	1	220,000
0	1	1	780,000
1	1	1	670,000
1	1	1	640,000

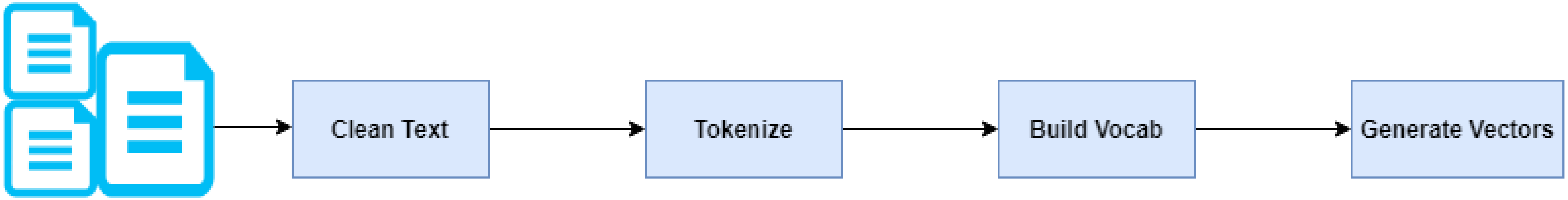
Data transformation – LibSVM format

- Every data mining software require specific data format in order to use in data mining processes.
- Transform data to the LibSVM format.
- <target> <index 1>:<value 1> <index 2>:<value 2>…<index n>.

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 40:1 54:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 2:1 6:1 14:1 20:1 37:1 41:1 47:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 14:1 22:1 36:1 42:1 49:1 64:1 67:1 72:1 74:1 77:1 80:1 83:1
-1 1:1 6:1 17:1 19:1 39:1 42:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 20:1 37:1 42:1 48:1 64:1 71:1 73:1 74:1 76:1 81:1 83:1
+1 5:1 11:1 15:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
```

Text pre-processing pipeline with BOW

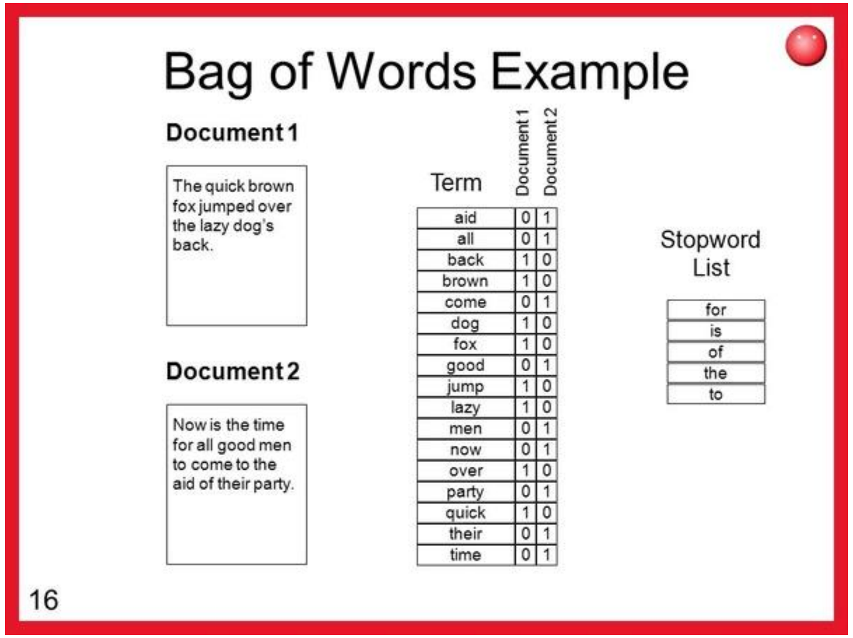
- Clean the text (white space, upper case, etc.)
- Tokenizing
- Build dictionary
- Filter is also useful (sorting histogram and take top 50)
- The vectors will be use in ML algorithms for document classification or clustering.



Data transformation-text value

Bag of Words (BOW)

- Machine learning can’t work with raw text
- Text must be convert to numbers, vectors of numbers
- Usually this is called “feature extraction”
- Bag of words is the most simple approach.
- Represent of string based on frequency of entities


Bag of Words Example

Document 1

The quick brown fox jumped over the lazy dog's back.

Document 2

Now is the time for all good men to come to the aid of their party.

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Stopword List
for
is
of
the
to

16

BOW (cont.)

- Sentence 1: "I love apples."
- Sentence 2: "I do not love oranges.“
- Vocab = ["I", "love", "apples", "do", "not", "oranges"]
- Vector for Sentence 1: [1, 1, 1, 0, 0, 0]
- Vector for Sentence 2: [1, 1, 0, 1, 1, 1]

BOW in action

```python
sentences = ['sky is nice', 'clouds are nice', 'Sky is nice and Clouds are nice']

cleaned_sentence = []

for sentence in sentences:
    word = sentence.lower()
    ##lowering all the letters becaz we dont want it to treat uppercase and lower case words differently

    word = word.split()    ##splitting our sentence into words

    ##removing stop words
    word = [i for i in word if i not in set(stopwords.words('english'))]
    word = " ".join(word)                  ##joining our words back to sentences
    cleaned_sentence.append(word)          ##appending our preprocessed sentence into a new list


## printing our new list
print(cleaned_sentence)
```

BOW in action

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features = 10)
Bagofwords = cv.fit_transform(cleaned_sentence).toarray()

print(Bagofwords)
```

BOW modeling in action

```python
wordfreq = {} # create dict and iterate through each sentence
for sentence in corpus:
    tokens = nltk.word_tokenize(sentence)
    for token in tokens:
        if token not in wordfreq.keys():
            wordfreq[token] = 1
        else:
            wordfreq[token] += 1

# filter dimension to 300
import heapq
most_freq = heapq.nlargest(300, wordfreq, key=wordfreq.get)

sentence_vectors = [] # create sentence list and iterate throgh each sentence
for sentence in corpus:
    sentence_tokens = nltk.word_tokenize(sentence)
    sent_vec = []
    for token in most_freq:
        if token in sentence_tokens:
            sent_vec.append(1)
        else:
            sent_vec.append(0)
```

```python
from sklearn.feature_extraction.text import CountVectorizer
matrix = CountVectorizer(max_features=300)
X = matrix.fit_transform(data).toarray()
```

Limitation of BOW

- **High Dimensionality:** With a large vocabulary, the feature space becomes very large, leading to issues like the curse of dimensionality.
- **Context Ignorance:** doesn't look at the order or context of words, so lines with different meanings but the same words would be represented the same way.
- **Sparsity:** The vectors tend to be sparse (mostly zeros), which can be inefficient for computation, especially with large vocabularies.

Bag of Ngrams (BoN)

- N-grams - contiguous sequences of 'n' items from a given sample of text or speech
- Unigram – “The quick brown fox"
- Bi-gram - "The quick", "quick brown", "brown fox"
- Tri-gram - "The quick brown", "quick brown fox"

BoN in action

```python
def generate_ngrams(text, n):
    # Split the text into words
    words = text.split()
    # Create n-grams
    ngrams = zip(*[words[i:] for i in range(n)])
    return [" ".join(ngram) for ngram in ngrams]

# Example usage
text = "The quick brown fox jumps over the lazy dog"
bigrams = generate_ngrams(text, 2)
trigrams = generate_ngrams(text, 3)

print("Bigrams:", bigrams)
print("Trigrams:", trigrams
```

Workshop 2 (Basic Processing and Representation)

- 1 Use the previous dataset “spam dictation”
- 2 Preprocess text including:
  - Remove white space
  - Remove anything that is not English
  - Calculate word length and added with column name “length”
- 3 Create new column name “text2”

	label	text	length
1463	ham	ok good later come lucky told earlier later pp...	114
2313	ham	guys	23
3290	ham	smoking people use wylie smokes justify ruinin...	85
2604	ham	times job today ok umma ask speed	57
4018	spam	ve selected stay british hotels holiday valued...	159

Workshop 2 (Basic Processing and Representation)

- Todays Voda numbers ending 1225 are selected to receive a å£50award. \
- If you have a match please call 08712300220 quoting claim code 3100 standard rates app")
- 'todays voda numbers ending selected receive award match quoting claim code standard rates app'

- 4. Use labelEncoder method to convert class target
- 5. Use CountVectorize to perform BOW
- 6. List Top 5 and bottom 5 of transform sample to show the results and submit your works to MS team