Chap 2: Data Pre-processing (tt)

Nguyễn Tấn Phú
ntanphu@ctuet.edu.vn

Bộ môn HTTT
Khoa CNTT – Đại học Kỹ Thuật Công Nghệ Cần Thơ

Outline

- Structure data preprocessing
- Text Processing and Feature Extraction
- Images data preprocessing
- Audio, Video data preprocessing

Structure data preprocessing

❑ Importing the libraries

In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

❑ Load dataset

In [2]:

```python
dataset = pd.read_csv('Data.csv')
```

In [3]:

```python
print(dataset)
# print(X)
# print(y)
```

Structure data preprocessing

❑ Output

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Structure data preprocessing

- Xác định các feature
- X = dataset.iloc[:3, :-1] // cắt từ 3 hàng đầu và bỏ cột cuối.

```
     Country   Age    Salary
0
   France  44.0  72000.0
1     Spain   27.0   48000.0
2  Germany   30.0   54000.0
```

- Để xử lý dữ liệu thì bạn phải chuyển về numpy array với hàm X = dataset.iloc[:3, :-1].values.

Structure data preprocessing

❑ Tiền xử lý dữ liệu

- Xử lý Missing Data
- Standardization (Phân phối chuẩn)
- Handling Catogrical Variables
- One-hot Encoding
- Multicollinearity

Structure data preprocessing

❑ TH1:

In [4]:

```
for i in range(len(dataset.columns)):
    missing_data = dataset[dataset.columns[i]].isna().sum()
    perc = missing_data / len(dataset) * 100
    print('>%d,  missing entries: %d, percentage %.2f' % (i, missing_data, perc))
```

```
>0,  missing entries: 0, percentage 0.00
>1,  missing entries: 1, percentage 10.00
>2,  missing entries: 1, percentage 10.00
>3,  missing entries: 0, percentage 0.00
```

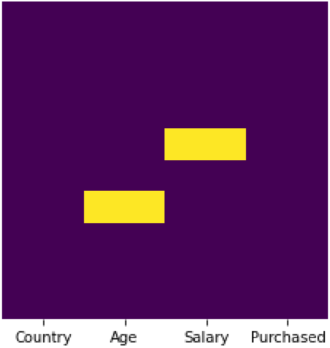
In [5]:

```
plt.figure(figsize = (4,4)) #is to create a figure object with a given size
sns.heatmap(dataset.isna(), cbar=False, cmap='viridis', yticklabels=False)
```

Data Imputation (Missing Data Replacement)

❑ TH1:

Out[5]: <AxesSubplot:>



In [6]:

```
#convert the dataframe into a numpy array by calling values
X= dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Data Imputation (Missing Data Replacement)

❑ TH2:

- Sử dụng thư viện Sklearn xử lý các missing data.
- **SimpleImputer** là một class của Sklearn hỗ trợ xử lý các missing data là số và thay chúng là một giá trị trung bình của cột, tần suất của dữ liệu suất hiện nhiều nhất …

Data Imputation (Missing Data Replacement)

❑ TH2:

In [7]:

```
from sklearn.impute import SimpleImputer

#Create an instance of Class SimpleImputer: np.nan is the empty value in the dataset
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

#Replace missing value from numerical Col 1 'Age', Col 2 'Salary'
#fit on the dataset to calculate the statistic for each column
imputer.fit(X[:, 1:3])

#The fit imputer is then applied to the dataset
# to create a copy of the dataset with all the missing values
# for each column replaced with the calculated mean statistic.
#transform will replace & return the new updated columns
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

Data Imputation (Missing Data Replacement)

❑ TH2:

In [8]:
```
print(X)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

Encode Categorical Data

❑ **Encode Independent Variables:**

- Convert một cột chứa các String thành vector 0 & 1.
- Sử dụng ColumnTransformer class OneHotEncoder của sklearn.

Encode Categorical Data

❑ Encode Independent Variables:

Index	Animal
0	Dog
1	Cat
2	Sheep
3	Horse
4	Lion

One-Hot code

Index	Dog	Cat	Sheep	Lion	Horse
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	0	1
4	0	0	0	1	0

Encode Categorical Data

❑ Encode Independent Variables:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

- Tạo một tuple (‘encoder’ encoding transformation, instance của class OneHotEncoder, [cols muốn transform) và các cols khác không muốn làm gì tới nó thì có thể dùng remainder=”passthrough” để bỏ qua.

```
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])] , remainder="passthrough" )
```

Encode Categorical Data

- Fit và transform với input = X và instance ct của class ColumnTransformer

```
#fit and transform with input = X
#np.array: need to convert output of fit_transform() from matrix to np.array
X = np.array(ct.fit_transform(X))
```

Encode Categorical Data

- **Encode Independent variable (X)**

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
#transformers: specify what kind of transformation, and which cols
#Tuple ('encoder' encoding transformation, instance of Class OneHotEncoder, [col to transform])
#remainder ="passthrough" > to keep the cols which not be transformed. Otherwise, the remaining cols
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])] , remainder="passthrough" )
#fit and transform with input = X
#np.array: need to convert output of fit_transform() from matrix to np.array
X = np.array(ct.fit_transform(X))
```

```
print(X)
```

```
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

Encode Categorical Data

- **Encode Dependent Variables:** Sử dụng Label Encoder để mã hóa các nhãn

In [11]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
#output of fit_transform of Label Encoder is already a Numpy Array
y = le.fit_transform(y)
```

In [12]:
```
print(y)
```

```
[0 1 0 0 1 1 0 1 0 1]
```

Splitting Training set and Test set

- **train_test_split**: của Sklearn-Model Selection để cắt dữ liệu train và test.
- **test_size**: để chia dữ liệu tập test trên toàn bộ dữ liệu.
- **random_state = 1**: Giúp sử dụng bộ random có sẵn của python.

Splitting Training set and Test set

- **train_test_split**: của Sklearn-Model Selection để cắt dữ liệu train và test.
- **test_size**: để chia dữ liệu tập test trên toàn bộ dữ liệu.
- **random_state = 1**: Giúp sử dụng bộ random có sẵn của python.

Splitting Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

```
print(X_train)
```

```
[[0.0 0.0 1.0 38.77777777777778 52000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 35.0 58000.0]]
```

```
print(X_test)
```

```
[[0.0 1.0 0.0 30.0 54000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

Splitting Training set and Test set

In [16]:
```
print(y_train)
```

```
[0 1 0 0 1 1 0 1]
```

In [17]:
```
print(y_test)
```

```
[0 1]
```

Feature Scaling (chuẩn hóa đặc trưng)

- Tại sao lại xảy ra Feature Scaling ?

✓ Khi khai phá dữ liệu thì có thể có một số feature có độ lớn hơn hẳn các feature khác do vậy features nhỏ hơn chắc chắn sẽ bị bỏ qua khi chúng ta thực hiện ML Model.

Feature Scaling

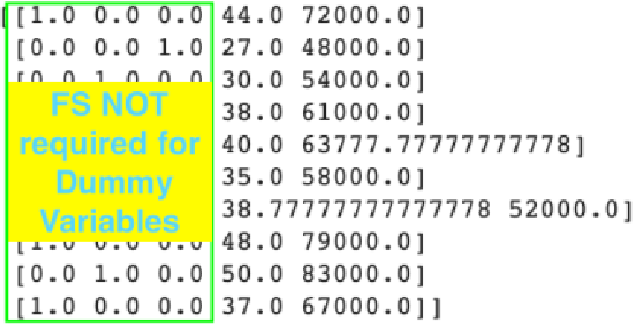
- Note #1: FS không cần áp dụng cho Multi-Regression Model:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \ldots + b_nx_n$$

- Khi đó $b_0, b_1, b_{2,} b_3, b_n$ là các hệ số để bù lại cho việc chênh lệnh do vậy không cần FS

Feature Scaling

- Note #2: Đối với các Categorical Features Encoding cũng không cần áp dụng FS.



Feature Scaling.

Feature Scaling

- Note #3: FS phải được thực hiện sau khi splitting Training và Test sets. Do nếu chúng ta sử dụng FS trước khi splitting training & test sets thì dữ liệu sẽ bị mất đi tính đúng.

➢ **Vậy làm sao để Feature Scaling ?**

Standardisation & Normalisation

- **Standardisation:** Biến đổi dữ liệu sao cho giá trị ***trung bình*** là 0 và ***standard deviation*** là 1.

	Country	Age	Salary	Purchased
0	France	44.0	72000.000000	No
1	Spain	27.0	48000.000000	Yes
2	Germany	30.0	54000.000000	No
3	Spain	38.0	61000.000000	No
4	Germany	40.0	63777.777778	Yes

Standardisation & Normalisation

- Từ tập dữ liệu có thể thấy số Age và Salary có độ chênh lệnh nhau khá nhiều do vậy dữ liệu của Age có thể không được sử dụng trong model. Cho nên cần chuẩn hóa dữ liệu đưa chúng về số nhỏ hơn và vẫn đảm bảo tính tương quan của dữ liệu.

$$x_{std}^{[i]} = \frac{x^{[i]} - \mu_x}{\sigma_x}$$

Standardisation & Normalisation

In [18]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:,3:] = sc.fit_transform(X_train[:,3:])
#only use Transform to use the SAME scaler as the Training Set
X_test[:,3:] = sc.transform(X_test[:,3:])
```

In [19]:
```
print(X_train)
```

```
[[0.0 0.0 1.0 -0.19159184384578545 -1.0781259408412425]
 [0.0 1.0 0.0 -0.014117293757057777 -0.07013167641635372]
 [1.0 0.0 0.0 0.566708506533324 0.633562432710455]
 [0.0 0.0 1.0 -0.30453019390224867 -0.30786617274297867]
 [0.0 0.0 1.0 -1.9018011447007988 -1.420463615551582]
 [1.0 0.0 0.0 1.1475343068237058 1.232653363453549]
 [0.0 1.0 0.0 1.4379472069688968 1.5749910381638885]
 [1.0 0.0 0.0 -0.7401495441200351 -0.5646194287757332]]
```

In [20]:
```
print(X_test)
```

```
[[0.0 1.0 0.0 -1.4661817944830124 -0.9069571034860727]
 [1.0 0.0 0.0 -0.44973664397484414 0.2056403393225306]]
```

Standardisation & Normalisation

- **Normalisation:** [0, 1]

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

```
x_norm = (x - x.min()) / (x.max() - x.min())
x_norm
```

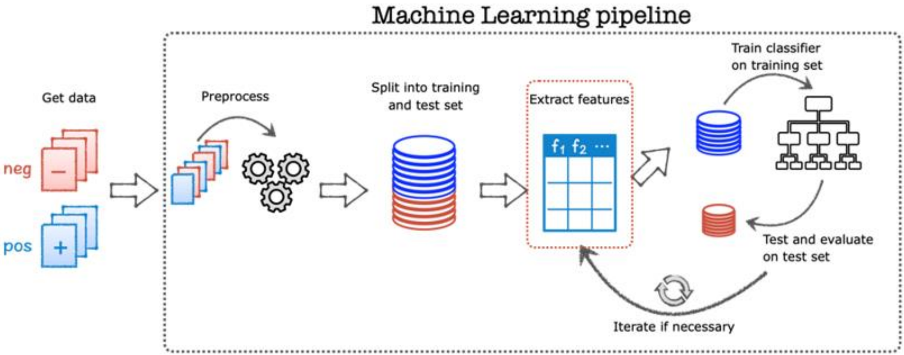
```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

Text Processing and Feature Extraction

- **Introduction**
- **Text preprocessing**
  - Text tokenization
  - Text normalization
  - Text parsing and filtering
- **Feature Extraction**
  - Bag-of-Words model
  - tf-idf model

Introduction

- **Text analysis pipeline**



Text preprocessing

Goal: Transform raw text input into normalized sequence of tokens. Prepare for feature extraction

"Hi. This is an example sentence in an Example Document."

→ [hi, example, sentence, example, document] → [1, 2, 1, 1]

The document corpus

- A corpus contains the documents that we want to process. Each document can be accessed by a unique document label or document ID.
- The document itself is usually a (very long) character string (Python type: str) that may contain line breaks.

The document corpus

In [1]:
```
# a small toy corpus with some (adapted) German newspaper headlines from June 20th
corpus = {   # document label: document text
    'spon1': 'Mehr Zustimmung zur EU auch wegen Trump - Danke May, danke Trump',
    'spon2': 'Nach Tod von US-Student Warmbier: Trump beschuldigt Nordkorea',
    'focus': 'Tod von US-Student Warmbier - Trump beschuldigt Nordkorea-Regime',
    'xyz': 'EU bleibt EU, aber EU-US-Beziehungen unter Trump weiter angespannt',
}
```

In [2]:
```
# access by document label
corpus['spon1']
```

Out[2]:
```
'Mehr Zustimmung zur EU auch wegen Trump - Danke May, danke Trump'
```

Tokenization

- Goal: Break down document text into smaller, meaningful components (***paragraphs, sentences, words***) → from a document, form ***a list of tokens***.

- With plain Python: calling split() on a string splits it by whitespace.

```
print(corpus['xyz'].split())
```

```
['EU', 'bleibt', 'EU,', 'aber', 'EU-US-Beziehungen', 'unter', 'Trump', 'weiter', 'angespannt']
```

```
print(corpus['spon2'].split())
```

```
['Nach', 'Tod', 'von', 'US-Student', 'Warmbier:', 'Trump', 'beschuldigt', 'Nordkorea']
```

al

Tokenization

- **str.split()** might not be optimal.
- NLTK
  - ✓ TreebankWordTokenizer
  - ✓ RegExpTokenizer

Tokenization

```python
import nltk

# word_tokenize uses TreebankWordTokenizer by default
# set language to "german" to use German punctuation
print(nltk.word_tokenize(corpus['spon2'], language="german"))
```

```
['Nach', 'Tod', 'von', 'US-Student', 'Warmbier', ':', 'Trump', 'beschuldigt', 'Nordkorea']
```

```python
nltk.word_tokenize("I wasn't there.")    # default language is English
```

```
['I', 'was', "n't", 'there', '.']
```

```python
# tokenize whole corpus
tokens = {doc_label: nltk.word_tokenize(text, language="german")
          for doc_label, text in corpus.items()}
tokens.keys()
```

```
dict_keys(['spon1', 'xyz', 'focus', 'spon2'])
```

```python
print(tokens['spon1'])
```

```
['Mehr', 'Zustimmung', 'zur', 'EU', 'auch', 'wegen', 'Trump', '-', 'Danke', 'May', ',', 'danke', 'Trump']
```

Text normalization

❖ **Can involve:**

- expanding contractions
- expanding hyphenated compound words
- removing special characters
- case conversion
- removing stopwords
- correct spelling
- stemming / lemmatization

➢ ***The order is important!***

Text normalization

- Expanding contractions
  - strategy: make list of all possible contractions and their expanded replacement
  - search & replace with Python using regular expressions
  - see "correct spelling" later

Expanding hyphenated compound words

- How to handle words like "**US-Student**"?
  - leave as is
  - strip hyphens (see "removing special characters" later)
  - split by hyphens

```
# example to split by hyphen
split_tokens = []
for t in tokens['focus']:
    split_tokens.extend(t.split('-'))
print(split_tokens)
```

```
['Tod', 'von', 'US', 'Student', 'Warmbier', '-', 'Trump', 'beschuldigt', 'Nordkorea', 'Regime']
```

Removing special characters

```
import string
string.punctuation
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
del_chars = str.maketrans('', '', string.punctuation + '–')   # add another character "–"
print([t.translate(del_chars) for t in tokens['focus']])   # apply table "del_chars"
```

```
['Tod', 'von', 'USStudent', 'Warmbier', '', 'Trump', 'beschuldigt', 'NordkoreaRegime']
```

❖ Especially Part-of-Speech tagging

Removing special characters

- Our strategy: Split only if first compound word is possibly longer than one character.

```
def expand_compound_token(t, split_chars="-"):
    parts = []
    add = False   # signals if current part should be appended to previous part
    for p in t.split(split_chars):  # for each part p in compound token t
        if not p: continue  # skip empty part
        if add and parts:   # append current part p to previous part
            parts[-1] += p
        else:               # add p as separate token
            parts.append(p)
        add = len(p) <= 1   # if p only consists of a single character -> appen
        #add = p.isupper()   # alt. strategy: if p is all uppercase ("US", "E",

    return parts

print(expand_compound_token("US-Student"))
print(expand_compound_token("Nordkorea-Regime"))
print(expand_compound_token("E-Mail-Provider"))
```

```
['US', 'Student']
['Nordkorea', 'Regime']
['EMail', 'Provider']
```

Removing special characters

```
tmp_tokens = {}
for doc_label, doc_tok in tokens.items():
    tmp_tokens[doc_label] = []
    for t in doc_tok:
        t_parts = expand_compound_token(t)
        tmp_tokens[doc_label].extend(t_parts)

print('Old:', tokens['focus'])
print('New:', tmp_tokens['focus'])
tokens = tmp_tokens
```

```
Old: ['Tod', 'von', 'US-Student', 'Warmbier', '-', 'Trump', 'beschuldigt', 'Nordkorea-Regime']
New: ['Tod', 'von', 'US', 'Student', 'Warmbier', '-', 'Trump', 'beschuldigt', 'Nordkorea', 'Regime']
```

Case conversion

❖ Usually: convert all words to lowercase.

```
print([t.lower() for t in tokens['focus']])
```

```
['tod', 'von', 'us', 'student', 'warmbier', '-', 'trump'
```

➢ `str.lower(),str.upper()`

❖ ***Proper Part-of-Speech tagging might not be possible afterwards!***

Removing stopwords

- Stopwords are words that are removed before doing further text analysis. Usually: Very common words for a certain language that transport little information.

- Stopword list depends on:
  - Language
  - Your data / research scenario (filter out too common words)
  - Later text analysis method, e.g:
    - ✓ *tf-idf* automatically reduces importance of very common words (as opposed to *Bag-of-Words*)
    - ✓ sentiment analysis: bad idea to have words like "not" in the stopword list!

Removing stopwords

```
print('English:', nltk.corpus.stopwords.words('english')[:5], '...')
print('German:', nltk.corpus.stopwords.words('german')[:5], '...')
```

```
English: ['i', 'me', 'my', 'myself', 'we'] ...
German: ['aber', 'alle', 'allem', 'allen', 'aller'] ...
```

```
# usage example (will remove "von" tokens):
stopwords = nltk.corpus.stopwords.words('german')
[t for t in tokens['focus'] if t.lower() not in stopwords]
```

```
['Tod',
 'US',
 'Student',
 'Warmbier',
 '-',
 'Trump',
 'beschuldigt',
 'Nordkorea',
 'Regime']
```

Correct spelling

- Depends on your data → especially necessary when working with social media data, surveys, etc.
- Available packages for automatic spell correction:
  - PyEnchant
  - aspell-python
  - pattern (suggest() function)

Stemming or Lemmatization

❖ **Goal:** Reduce inflected words to a common form so that they're counted as one.

❖ **Stemming:**

- Remove affixes from a word to get base form *(stem)* of a word → stem might not be a lexicographically correct word

❖ **Example:**

- books → book
- booked → book
- **employees → employ**
- **argued → argu**

Stemming or Lemmatization

❖ NLTK implements several stemming algorithms:

- PorterStemmer, LancasterStemmer (English only)
- SnowballStemmer (supports 13 languages)

```
stemmer = nltk.stem.LancasterStemmer()
stemmer.stem('employees')
```

```
'employ'
```

```
stemmer = nltk.stem.SnowballStemmer('german')
print('Bücher →', stemmer.stem("Bücher"))
print('gebuchte →', stemmer.stem("gebuchte"))
print('sahen →', stemmer.stem("sahen"))
```

```
Bücher → buch
gebuchte → gebucht
sahen → sah
```

Stemming or Lemmatization

❖ Lemmatization

- Find *lemma* (dictionary form) of a inflected word → a lemma is always a lexicographically correct word

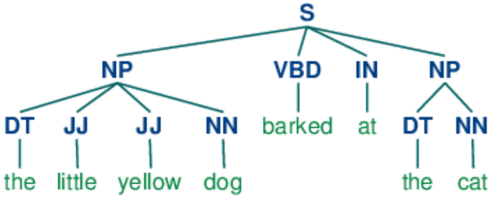
```
lemmatizer = nltk.stem.WordNetLemmatizer()
# lemmatize(): first argument is word, second is Part-of-Speech tag
print('books →', lemmatizer.lemmatize('books', 'n'))    # n stands for noun
print('booked →', lemmatizer.lemmatize('booked', 'v'))  # v stands for verb
print('employees →', lemmatizer.lemmatize('employees', 'n'))
print('argued →', lemmatizer.lemmatize('argued', 'v'))
```

```
books → book
booked → book
employees → employee
argued → argue
```

Text parsing

❖ To understand **text syntax and structure**

- *Part-of-Speech (POS) tagging* → annotate words with lexical categories
- *Shallow parsing / chunking* → split sentences into phrases
- Dependency-based parsing
- Constituency-based parsing



POS tagging

❖ **Goal:** assign a lexical category such as noun, verb, adjective, etc. to each word.

- *Needed for lemmatization*
- *Optionally needed for filtering (e.g. nouns only)*

```
example = ['The', 'little', 'yellow', 'dog', 'barked', 'loudly', 'at', 'the', 'cat', '.']
nltk.pos_tag(example)    # with default tagset (Penn Treebank)
```

```
[('The', 'DT'),
 ('little', 'JJ'),
 ('yellow', 'JJ'),
 ('dog', 'NN'),
 ('barked', 'VBD'),
 ('loudly', 'RB'),
 ('at', 'IN'),
 ('the', 'DT'),
 ('cat', 'NN'),
 ('.', '.')]
```

POS tagging

```python
nltk.pos_tag(example, tagset='universal')    # with universal tagset
```

```
[('The', 'DET'),
 ('little', 'ADJ'),
 ('yellow', 'ADJ'),
 ('dog', 'NOUN'),
 ('barked', 'VERB'),
 ('loudly', 'ADV'),
 ('at', 'ADP'),
 ('the', 'DET'),
 ('cat', 'NOUN'),
 ('.', '.')]
```

Text normalization summary

- **Steps from raw input text to normalized tokens:**
  1. *tokenization*
  2. *expand compound words*
  3. *POS tagging*
  4. *lemmatization*
  5. *lower-case transformation*
  6. *removing special characters*
  7. *removing stopwords </small>*
- **Each step involves decisions that highly effect further analyses.**

Recommended Python packages for text preprocessing

- **NLTK**: stable but slow
- **Pattern**: many language models but some of them only with low accuracy, Python 2.7 only
- **Spacy**: language models for English and partly for German and French
- **SyntaxNet**: many language models but difficult to install, Python 2.7 only
- **Stanford CoreNLP:** many language models but requires Java

Feature Extraction

❖ Bag-of-Words (BoW) model

- Simple but powerful model
- Features are absolute term counts
- Basis for:
  - ✓ Topic Modeling with *Latent Dirichlet Allocation (LDA)* via *Gibbs sampling*
  - ✓ Text classification with *Naive Bayes*, *Support Vector Machines*
  - ✓ Document similarity
  - ✓ Document clustering

Bag-of-Words (BoW) model

$$C = \{D_1, D_2, D_3\}$$
$$D_1 = \{simple, yet, beautiful, example\}$$
$$D_2 = \{beautiful, beautiful, flowers\}$$
$$D_3 = \{example, after, example\}$$

$$vocab = \{simple, yet, beautiful, example, flowers, after\}$$

document	simple	yet	beautiful	example	flowers	after
D_1	1	1	1	1	0	0
D_2	0	0	2	0	1	0
D_3	0	0	0	2	0	1

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 1 \end{pmatrix}$$

Bag-of-Words (BoW) model

❖ Example

```
from collections import Counter

example_data = ['a', 'b', 'c', 'b', 'b', 'a']
example_counter = Counter(example_data)
example_counter
```

```
Counter({'a': 2, 'b': 3, 'c': 1})
```

```
example_counter.update(['c', 'a', 'a', 'a'])
example_counter
```

```
Counter({'a': 5, 'b': 3, 'c': 2})
```

Bag-of-Words (BoW) model

- Normalized tokens with their POS tags are still in the variable tagged_tokens

```
pprint(tagged_tokens)
```

```
{'focus': [('tod', 'NN'),
           ('us', 'NE'),
           ('student', 'NN'),
           ('warmbier', 'NE'),
           ('trump', 'FM'),
           ('beschuldigen', 'VVPP'),
           ('nordkorea', 'NE'),
           ('regime', 'NN')],
 'spon1': [('zustimmung', 'NN'),
           ('eu', 'NE'),
           ('trump', 'NN'),
           ('danke', 'NE'),
           ('may', 'NE'),
           ('danke', 'PRELS'),
           ('trump', 'NE')],
```

Bag-of-Words (BoW) model

❖ Normalized tokens with their POS tags are still in the variable tagged_tokens

```
'spon2': [('tod', 'NN'),
          ('us', 'NE'),
          ('student', 'NN'),
          ('warmbier', 'NE'),
          ('trump', 'NE'),
          ('beschuldigen', 'VVFIN'),
          ('nordkorea', 'NE')],
'xyz': [('eu', 'NE'),
        ('bleiben', 'VVFIN'),
        ('eu', 'NE'),
        ('eu', 'NE'),
        ('us', 'NE'),
        ('beziehung', 'NN'),
        ('trump', 'NE'),
        ('angespannt', 'VVPP')]}
```

Bag-of-Words (BoW) model

```
documents = {doc_label: [t for t, _ in tok_pos]    # dismiss the POS tag
             for doc_label, tok_pos in tagged_tokens.items()}
```

1. Count the tokens for each document:

```
counts = {doc_label: Counter(tok) for doc_label, tok in documents.items()}
print('tokens:', documents['spon1'])
print('counts:', list(counts['spon1'].items()))
```

```
tokens: ['zustimmung', 'eu', 'trump', 'danke', 'may', 'danke', 'trump']
counts: [('may', 1), ('trump', 2), ('zustimmung', 1), ('eu', 1), ('danke', 2)]
```

Bag-of-Words (BoW) model

2. extract the vocabulary (set of unique terms in all documents):

```
vocab = set()
for counter in counts.values():
    vocab |= set(counter.keys())   # set union of unique tokens per doc

vocab = sorted(list(vocab))  # sorting here only for better display lat
vocab   # => becomes columns of BoW matrix
```

```
['angespannt',
 'beschuldigen',
 'beziehung',
 'bleiben',
 'danke',
 'eu',
 'may',
 'nordkorea',
 'regime',
 'student',
 'tod',
 'trump',
 'us',
 'warmbier',
 'zustimmung']
```

Bag-of-Words (BoW) model

3. Create the BoW matrix:

```
# create Bag of Words matrix: rows are documents, colum
bow = []
for counter in counts.values():  # iterate through each
    # make a list that contains the term count of each
    # if a term of the vocab. does not exist in this do
    bow_row = [counter.get(term, 0) for term in vocab]
    bow.append(bow_row)
bow
```

```
[[0, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 2, 0, 0, 1],
 [1, 0, 1, 1, 0, 3, 0, 0, 0, 0, 0, 1, 1, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0]]
```

Bag-of-Words (BoW) model

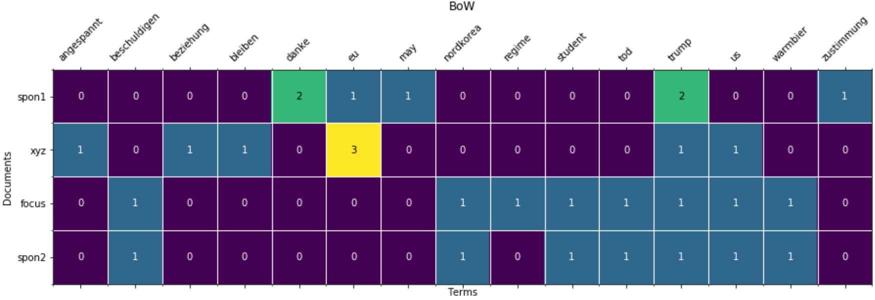
```
doc_labels = list(counts.keys())    # => becomes rows of BoW matrix
doc_labels
```

```
['spon1', 'xyz', 'focus', 'spon2']
```

```
from utils import plot_heatmap

# show a heatmap of the BoW model
print('spon1:', documents['spon1'])
plot_heatmap(bow, xticklabels=vocab, yticklabels=doc_labels, title='BoW', save_to='img/bow.png')
```

```
spon1: ['zustimmung', 'eu', 'trump', 'danke', 'may', 'danke', 'trump']
```



Bag-of-Words (BoW) model

❖ BoW can be used in conjuction with n-grams

```
# 1-grams (unigrams):
documents['spon1']
```

```
['zustimmung', 'eu', 'trump', 'danke', 'may', 'danke', 'trump']
```

```
from utils import create_ngrams

# 2-grams (bigrams):
print(create_ngrams(documents['spon1'], n=2))
```

```
['zustimmung eu', 'eu trump', 'trump danke', 'danke may', 'may danke', 'danke trump']
```

Bag-of-Words (BoW) model

❖ Bigrams of our tokens

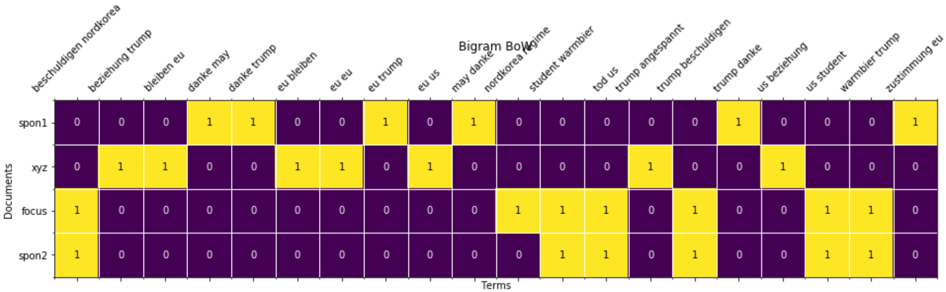
```
documents_bigrams = {doc_label: create_ngrams(doc_tok, n=2)
                     for doc_label, doc_tok in documents.items()}
documents_bigrams['spon1']
```

```
['zustimmung eu',
 'eu trump',
 'trump danke',
 'danke may',
 'may danke',
 'danke trump']
```

Bag-of-Words (BoW) model

```python
from utils import create_bow

bow_bi, doc_labels_bi, vocab_bi = create_bow(documents_bigrams)
plot_heatmap(bow_bi, xticklabels=vocab_bi, yticklabels=doc_labels_bi, title='Bigram BoW');
```



Tf-idf (Term Frequency – Inverse Document Frequency)

❖ **TF**: Term Frequency(Tần suất xuất hiện của từ) là số lần từ xuất hiện trong văn bản

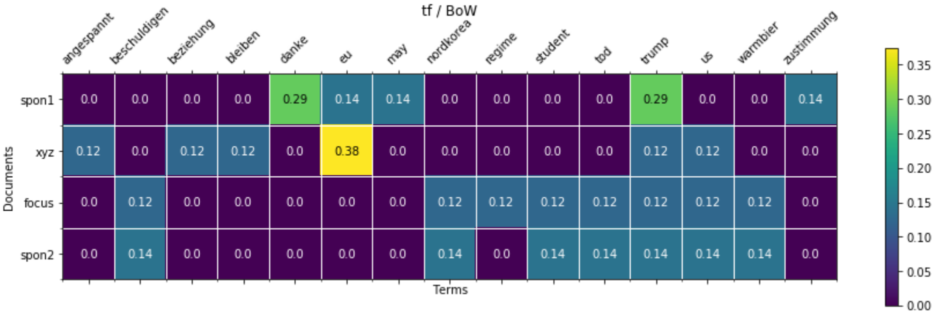
$$\mathrm{tf}(t, d) = \frac{\mathrm{f}(t, d)}{\max\{\mathrm{f}(w, d) : w \in d\}}$$

- Trong đó:
  - tf(t, d): tần suất xuất hiện của từ t trong văn bản d
  - f(t, d): Số lần xuất hiện của từ t trong văn bản d
  - max({f(w, d) : w ∈ d}): Số lần xuất hiện của từ có số lần xuất hiện nhiều nhất trong văn bản d

Tf-idf (Term Frequency – Inverse Document Frequency)

```python
import numpy as np
raw_counts = np.mat(bow, dtype=float)          # raw counts converted to NumPy matrix
tf = raw_counts / np.sum(raw_counts, axis=1)   # divide by row-wise sums (document lengths) -> proportions
plot_heatmap(tf, xticklabels=vocab, yticklabels=doc_labels, title='tf / BoW', legend=True, save_to='img/tf.png');
```

```
<matplotlib.figure.Figure at 0x7fae9773eac8>
```



Tf-idf (Term Frequency – Inverse Document Frequency)

❖ IDF: Inverse Document Frequency

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

- Trong đó:
  - idf(t, D): giá trị idf của từ t trong tập văn bản
  - |D|: Tổng số văn bản trong tập D
  - $|\{d \in D : t \in d\}|$: thể hiện số văn bản trong tập D có chứa từ t.

Tf-idf (Term Frequency – Inverse Document Frequency)

$$tfidf(t, d, D) = tf(t, d) * idf(t, D)$$

- Giá trị TF-IDF cao là những từ xuất hiện nhiều trong văn bản này, và xuất hiện ít trong các văn bản khác.
- Giúp lọc ra những từ phổ biến và giữ lại những từ có giá trị cao (từ khoá của văn bản đó).

Tf-idf (Term Frequency – Inverse Document Frequency)

```
def num_term_in_docs(t, docs):
    return sum(t in d for d in docs.values())

num_term_in_docs('eu', documents)
```

```
2
```

```
from math import log

# define a function that calculates the inverse document frequency
def idf(t, docs):
    return log(1 + len(docs) / (1+num_term_in_docs(t, docs)))

idf('eu', documents)
```

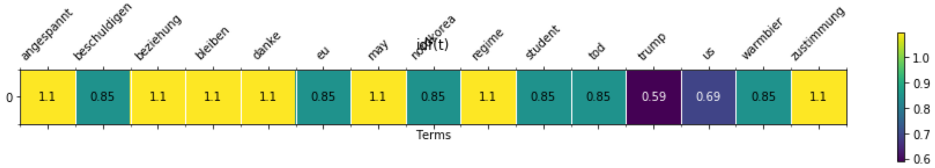
```
0.8472978603872034
```

Tf-idf (Term Frequency – Inverse Document Frequency)

```
idf_row = [idf(t, documents) for t in vocab]
```

```
plot_heatmap(np.mat(idf_row), vocab, title="idf(t)", ylabel=None, legend=True);
```

```
<matplotlib.figure.Figure at 0x7fae97699240>
```

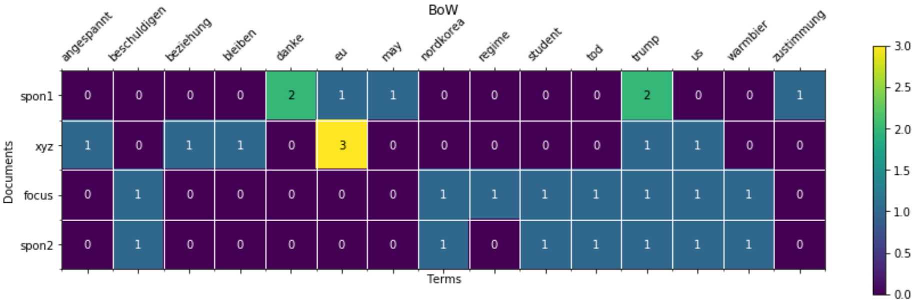


Tf-idf (Term Frequency – Inverse Document Frequency)

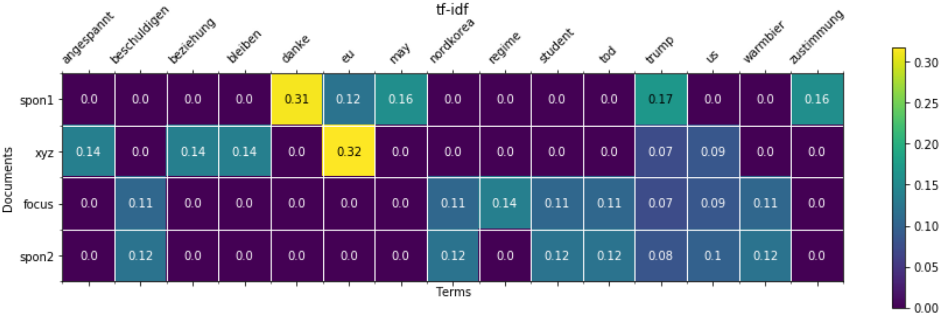
```
idf_mat = np.mat(np.diag(idf_row))
tfidf = tf * idf_mat
```

```
plot_heatmap(bow, xticklabels=vocab, yticklabels=doc_labels, title='BoW', legend=True, save_to='img/bow.png')
plot_heatmap(tfidf, xticklabels=vocab, yticklabels=doc_labels, title='tf-idf', legend=True, save_to='img/tfidf.png');
```

```
<matplotlib.figure.Figure at 0x7fae97477668>
```



Tf-idf (Term Frequency – Inverse Document Frequency)



$$tf(danke, spon1) = 2/7 = 0.2857$$

$$idf(danke) = log(1 + 4/2) = 1.0986$$

$$tfidf(danke, spon1) = 0.2857 \cdot 1.0986 = 0.3139$$

Recommended Python packages for BoW and tf-idf

- **Gensim**
  - doc2bow
  - TfIdfModel
- **scikit-learn**
  - CountVectorizer
  - TfidfVectorizer