

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

❑ Load dataset

```
In [2]: dataset = pd.read_csv('Data.csv')
```

```
In [3]: 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:

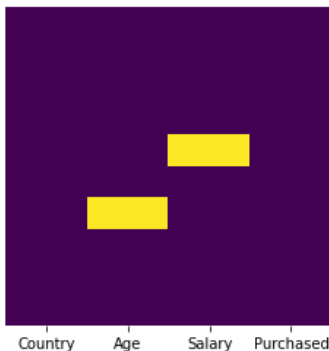
```
In [4]: for i in range(len(dataset.columns)):
        missing_data = dataset[dataset.columns[i]].isna().sum()
        perc = missing_data / len(dataset) * 100
        print('>%d, missing entries: %d, percentage %.2f' % (i, missing_data, perc))

>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 xuất 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	One-Hot code 	Index	Dog	Cat	Sheep	Lion	Horse
0	Dog		0	1	0	0	0	0
1	Cat		1	0	1	0	0	0
2	Sheep		2	0	0	1	0	0
3	Horse		3	0	0	0	0	1
4	Lion		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 63777.777777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.777777777777778 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:

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

- Khi đó b_0, b_1, b_2, b_3, b_n là các hệ số để bù lại cho việc chênh lệch do vậy không cần FS

Feature Scaling

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

[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]
[1.0 0.0 0.0]	38.0	61000.0]
[0.0 0.0 1.0]	40.0	63777.77777777778]
[0.0 1.0 0.0]	35.0	58000.0]
[1.0 0.0 0.0]	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]

FS NOT required for Dummy Variables

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 lệch 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

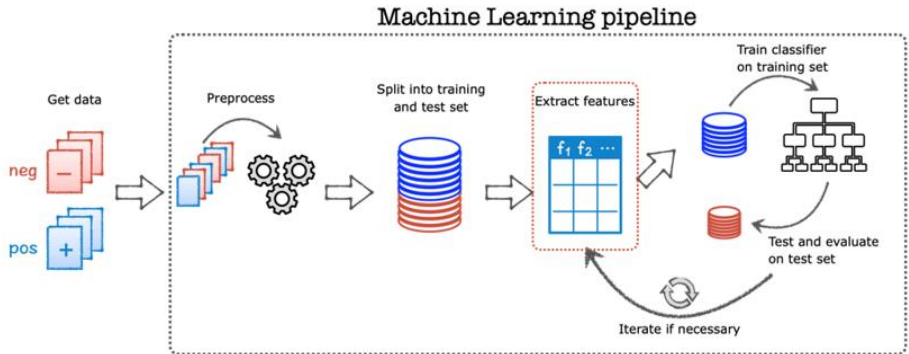
- Text tokenization
- Text normalization
- Text parsing and filtering

❖ 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']
```

Tokenization

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

Tokenization

```
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']
```

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

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

```
# 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'])
```

```
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 -> append
        #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 stopwords 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

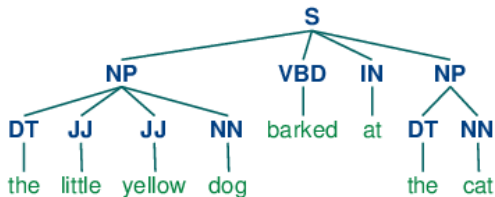
```
lemmatizer = nltk.stem.WordNetLemmatizer()
# Lemmatize(): first argument is word, second is Part-of-Speech tag
print('books →', lemmatizer.lemmatize('books', 'n'))    # n stands for noun
print('booked →', lemmatizer.lemmatize('booked', 'v'))  # v stands for verb
print('employees →', lemmatizer.lemmatize('employees',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

```
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

$$C = \{D_1, D_2, D_3\}$$

$$D_1 = \{simple, yet, beautiful, example\}$$

$$D_2 = \{beautiful, beautiful, flowers\}$$

$$D_3 = \{example, after, example\}$$

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

<i>document</i>	<i>simple</i>	<i>yet</i>	<i>beautiful</i>	<i>example</i>	<i>flowers</i>	<i>after</i>
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', 'VPPP'),
            ('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', 'VFIN'),
          ('nordkorea', 'NE')],
'xyz': [('eu', 'NE'),
        ('bleiben', 'VFIN'),
        ('eu', 'NE'),
        ('eu', 'NE'),
        ('us', 'NE'),
        ('beziehung', 'NN'),
        ('trump', 'NE'),
        ('angespannt', 'VPP')]]
```

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 later
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, column
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 doc
    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']
```

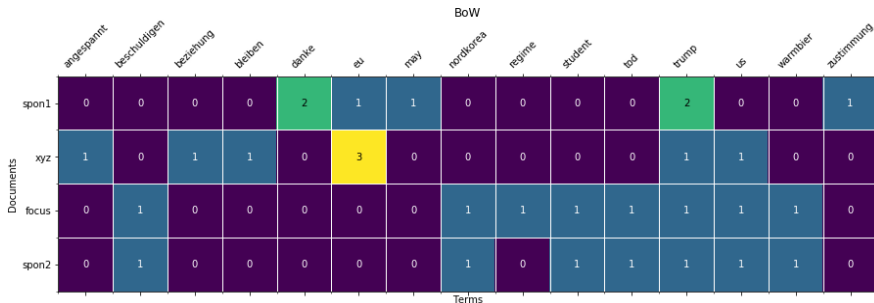
```
from utils import plot_heatmap
```

```
# 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 conjunction 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

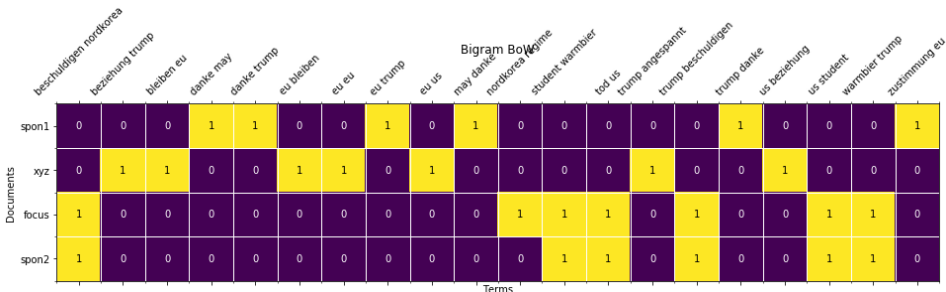
```
documents_bigrams = {doc_label: create_ngrams(doc_tok, n=2)
                      for doc_label, doc_tok in documents.items()}
documents_bigrams['spon1']
```

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

Bag-of-Words (BoW) model

```
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

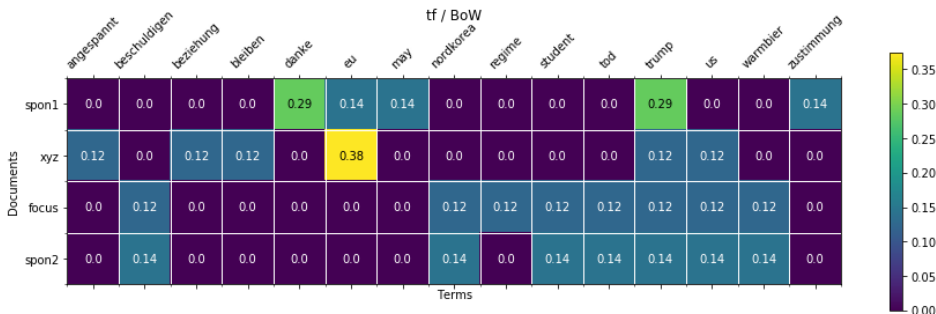
$$tf(t, d) = \frac{f(t, d)}{\max\{f(w, d) : w \in d\}}$$

- 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
 - $\max(\{f(w, d) : w \in d\})$: Số lần xuất hiện của từ có số lần xuất hiện nhiều nhất trong văn bản d

Tf-idf (Term Frequency – Inverse Document Frequency)

```
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

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

▪ Trong đó:

- $\text{idf}(t, D)$: 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)

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) * \text{idf}(t, D)$$

- 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)
```

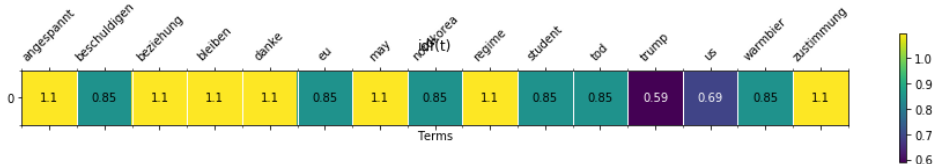
0.8472978603872034

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>
```

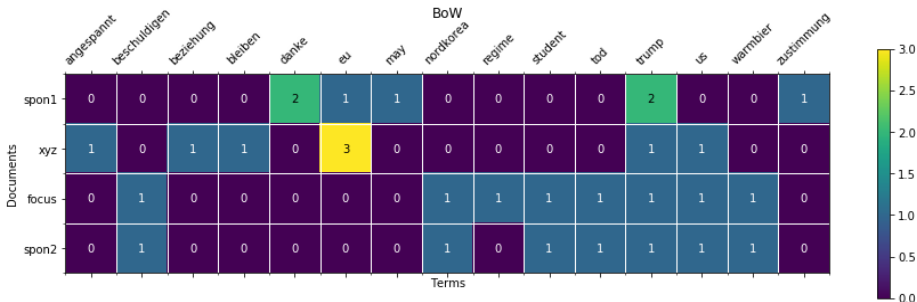


Tf-idf (Term Frequency – Inverse Document Frequency)

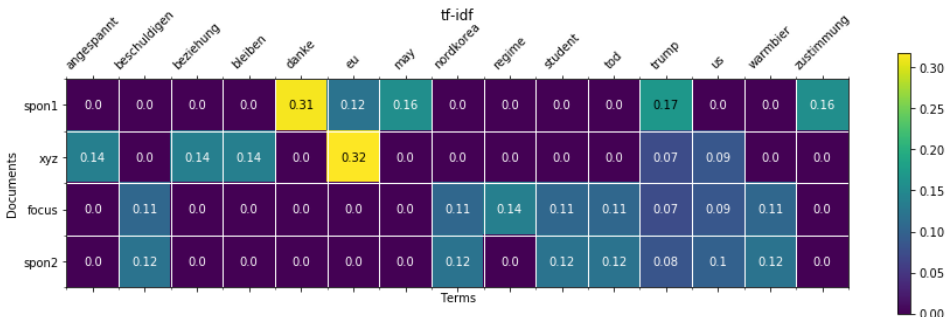
```
idf_mat = np.mat(np.diag(idf_row))
tfidf = tf * idf_mat
```

```
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

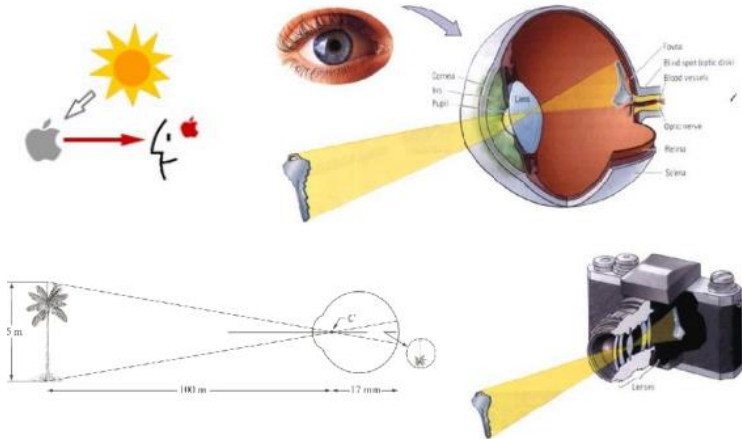
- doc2bow
- TfIdfModel

❖ scikit-learn

- CountVectorizer
- TfidfVectorizer

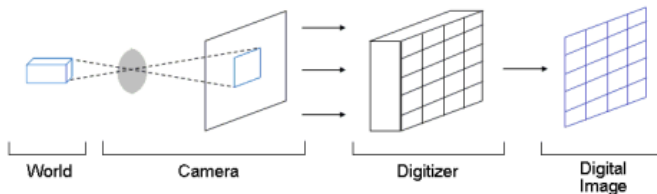
Images data preprocessing

❖ Cơ bản về ảnh số: Quá trình thu nhận ảnh



Images data preprocessing

❖ Quá trình thu nhận ảnh



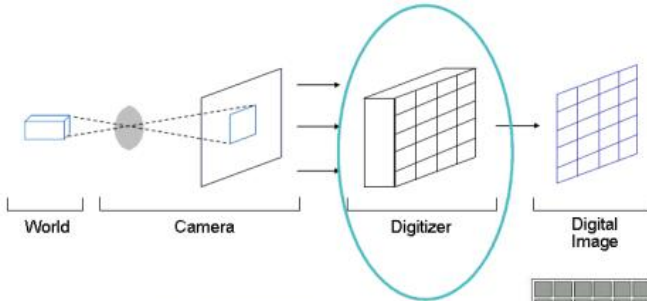
Máy ảnh

Bộ số hoá

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

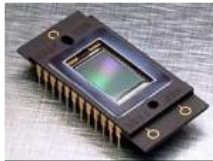
Images data preprocessing

❖ Quá trình thu nhận ảnh

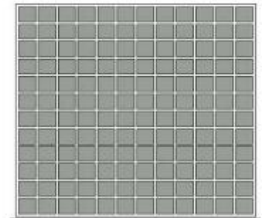


Có 2 loại:

- CMOS
- CCD

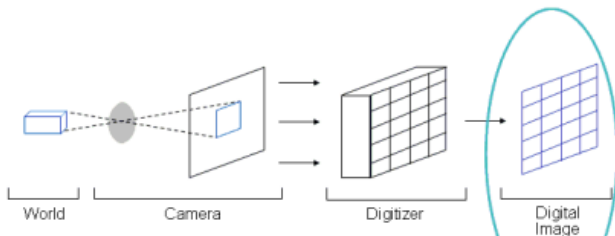


KAF-1600 - Kodak



Images data preprocessing

❖ Quá trình thu nhận ảnh



0	10	10	15	50	70	80
0	0	100	120	125	130	130
0	35	100	150	150	80	50
0	15	70	100	10	20	20
0	15	70	0	0	0	15
5	15	50	120	110	130	110
5	10	20	50	50	20	250

PIXEL
(picture element)

Images data preprocessing


❖ Biểu diễn ảnh số



64	60	69	100	149	151	176	182	179
65	62	68	97	145	148	175	183	181
65	66	70	95	142	146	176	185	184
66	66	68	90	135	140	172	184	184
66	64	64	84	129	134	168	181	182
59	63	62	88	130	128	166	185	180
60	62	60	85	127	125	163	183	178
62	62	58	81	122	120	160	181	176
63	64	58	78	118	117	159	180	176

Images data preprocessing

❖ Biểu diễn ảnh số

		x =																												
		58	59	60	61	62	63	64	65	66																				
y =	41	210	209	204	202	197	247	143	71	64																				
	42	206	196	203	197	195	210	207	56	63																				
	43	201	207	192	201	198	213	156	69	66																				
	44	216	206	211	193	202	207	208	57	69																				
	45	221	206	211	194	196	197	220	56	63																				
	46	209	214	224	199	194	193	204	173	64																				
	47	204	212	213	208	191	190	191	214	60																				
	48	214	215	215	207	208	180	172	188	69																				
	49	209	205	214	205	204	196	187	196	86	62	66	87	57	60	48														
	50	208	209	205	203	202	186	174	185	149	71	63	55	55	45	56														
	51	207	210	211	199	217	194	183	177	209	90	62	64	52	93	52														
	52	208	205	209	209	197	194	183	187	187	239	58	68	61	51	56														
	53	204	206	203	209	195	203	188	185	183	221	75	61	58	60	60														
	54	200	203	199	236	188	197	183	190	183	196	122	63	58	64	66														
	55	205	210	202	203	199	197	196	181	173	186	105	62	57	64	63														



Images data preprocessing

❖ Biểu diễn ảnh số

- Được biểu diễn bởi một ma trận 2 chiều kích thước $M \times N$
 - Mỗi phần tử của ma trận biểu diễn 1 phần tử ảnh (pixel, hay còn gọi là điểm ảnh)
 - Giá trị (số nguyên) của phần tử biểu diễn “màu” của điểm ảnh, có giá trị trong khoảng $[0, L_{\max}]$
- Số bit (K) cần thiết để biểu diễn 1 “màu”:

$$2^K \geq L_{\max} \rightarrow K = \text{ceil} (\log_2 L_{\max})$$

- Số bit cần thiết để biểu diễn 1 ảnh: $b = M \times N \times K$
- Công thức để truy xuất 1 điểm ảnh ?

Images data preprocessing

❖ Độ phân giải (resolution)

- Độ phân giải **không gian**
 - Chi tiết nhỏ nhất có thể phân biệt được
- Độ phân giải **màu**
 - Sự thay đổi “màu” nhỏ nhất có thể phân biệt được
- Một ảnh sẽ có độ phân giải không gian là $M \times N$ pixels và có độ phân giải “màu” là K bit, hoặc L_{\max} màu.

Images data preprocessing

❖ Độ phân giải không gian



200 X 278



50 X 70



12 X 18

Images data preprocessing

❖ Độ phân giải không gian



128 x 128



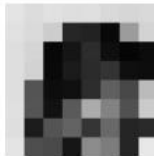
64 x 64



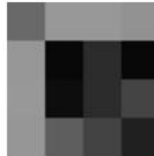
32 x 32



16 x 16



8 x 8



4 x 4

Images data preprocessing

❖ Độ phân giải màu



8 bits



4 bits



2 bits

Images data preprocessing

❖ Độ phân giải màu



256 mức xám



16 mức xám



8 mức xám



4 mức xám



2 mức xám
Ảnh nhị phân

Images data preprocessing

❖ 3 loại ảnh số

- Dựa vào “màu” của điểm ảnh, người ta phân biệt 3 loại ảnh số sau:



Ảnh mức xám



Ảnh nhị phân



Ảnh màu

- Ảnh nhị phân: chỉ có 2 mức “màu” (0: đen, 1: trắng)
- Ảnh mức xám: có nhiều mức “màu” (0: đen, 255: trắng)
- Ảnh màu: được tổng hợp từ 3 màu cơ bản: R, G, B

Images data preprocessing

❖ Ánh sáng (Light) & Màu sắc (Color)

- Về mặt vật lý, ánh sáng là một sóng điện từ:
 - Bản chất hạt: photon
 - Bản chất sóng: tần số, độ dài bước sóng
- Mắt người chỉ cảm nhận được các ánh sáng có bước sóng từ **380nm** (tím) đến **740nm** (đỏ).
- Ánh sáng với độ dài sóng khác nhau khi tác động lên mắt người sẽ tạo nên các ý niệm về màu sắc khác nhau.

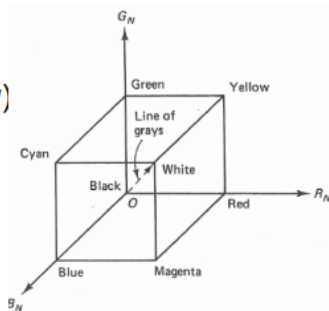
Images data preprocessing

❖ Mô hình màu (Color model)

➢ Mô hình màu là một mô hình toán học trừu tượng dùng để mô tả các màu sắc thông qua các bộ số.

➢ Các mô hình màu thông dụng

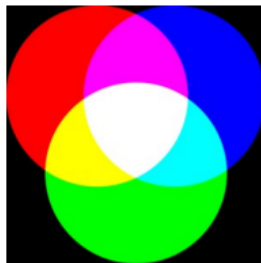
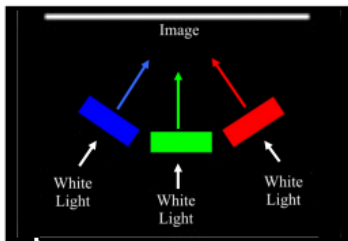
- **RGB** (Red – Green – Blue)
- **CMY** (Cyan – Magenta – Yellow)
- **HSV** (Hue – Saturation – Value)
- **YC_bC_r**
- **CIE**



Images data preprocessing

❖ Mô hình màu RGB


- Các màu sắc khác nhau được tạo ra từ 3 thành phần màu cơ bản: **R**, **G** và **B**.
- Sự pha màu mang tính chất “**cộng**” (additive)



Images data preprocessing

❖ Mô hình màu RGB

➢ Mỗi màu được biểu diễn bởi một bộ ba số (R,G,B). Mỗi thành phần R,G,B là 1 số thực có giá trị từ 0 đến 1.

- Màu đỏ  (1.0, 0.0, 0.0)
- Màu xanh lá  (0.0, 1.0, 0.0)
- Màu xanh dương  (0.0, 0.0, 1.0)
- Màu vàng  (1.0, 1.0, 0.0)
- Màu xanh lơ  (0.0, 1.0, 1.0)
- Màu tím  (1.0, 0.0, 1.0)
- Màu đen  (0.0, 0.0, 0.0) , màu trắng (1.0, 1.0, 1.0)

Images data preprocessing

❖ Độ sâu màu (Color depth)

- **Chế độ 24 bit**: mỗi thành phần R, G, B được biểu diễn bởi một số nguyên 8 bit
 - Biểu diễn được: $256^3 = 16.777.216$ màu
 - True color (mắt người phân biệt được 10 triệu màu)
 - Bộ nhớ cần thiết để lưu giữ hình ảnh cho 1 ảnh có độ phân giải 1024×768 ở chế độ 24 bit màu:

$$1024 \times 768 \times 3 = 2.359.296 \text{ bytes} > 2\text{MB}$$

- **Chế độ 32 bit**: (RGBA) thêm 8 bit cho độ “trong suốt” (transparent)

Images data preprocessing

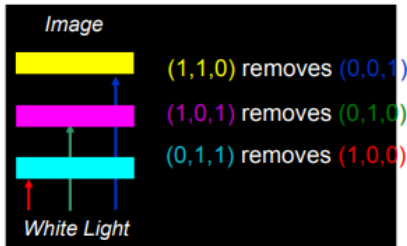
❖ Độ sâu màu (Color depth)

- Chế độ 16 bit: mỗi thành phần R, G, B được biểu diễn bởi một số nguyên 5 bit (555 mode = 15 bit), hoặc riêng thành phần G được biểu diễn bởi 6 bit (565 mode = 16bit)
 - Biểu diễn được: $256^2 = 65.536$ màu
 - Hi color (hiển thị hình ảnh đủ tốt)
- Chế độ 256 màu: hay ta có 8 bit để biểu diễn một màu
 - R, G, B: bao nhiêu bit ? 332 mode?

Images data preprocessing

❖ Không gian màu CMY (CMYK)

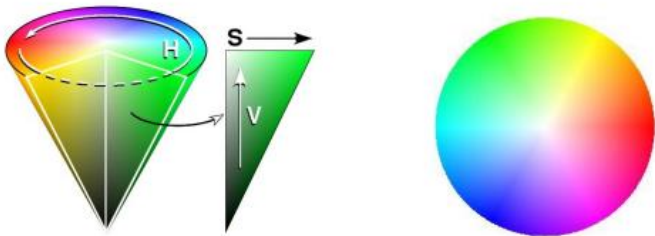
- Sử dụng 3 màu cơ bản: **Cyan**, **Magenta**, **Yellow**.
- Sự pha màu mang tính “trừ” (subtractive)
- Thường được sử dụng trong các máy in
- Để tiết kiệm mực → thêm thành phần K (mức xám)



Images data preprocessing

❖ Không gian màu HSV

- Dùng 3 thành phần
 - H (hue): sắc thái màu (vàng, xanh, đỏ, ...)
 - S (saturation): độ bão hòa màu (đỏ, đỏ tươi, ...)
 - V (value): độ sáng (brightness) của màu



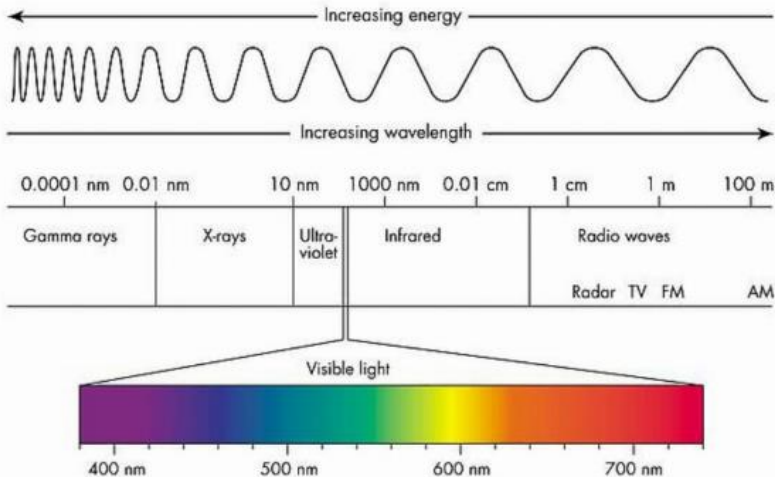
Images data preprocessing

❖ Không gian màu YC_bC_r

- Được dùng trong ảnh JPEG, các đoạn video...
- Thích hợp cho việc nén và giải nén dữ liệu theo tần số
- Y: thành phần độ sáng
- Cb và Cr: thành phần sắc thái màu
- Mô hình màu tương tự: YUV

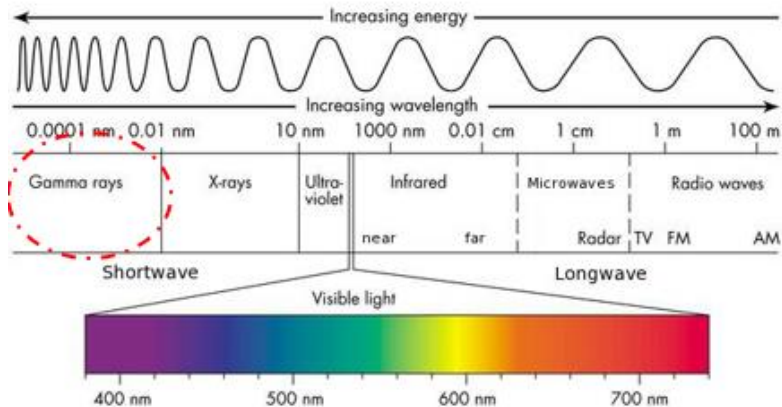
Images data preprocessing

❖ Ánh sáng & Màu sắc



Images data preprocessing

Gamma rays

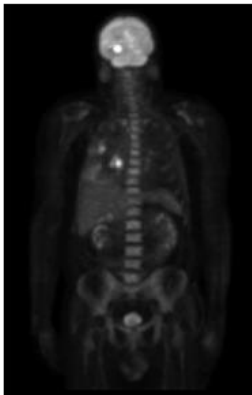


Images data preprocessing

❖ Gamma rays



Gamma-Ray
Imaging
Cherenkov
Telescope



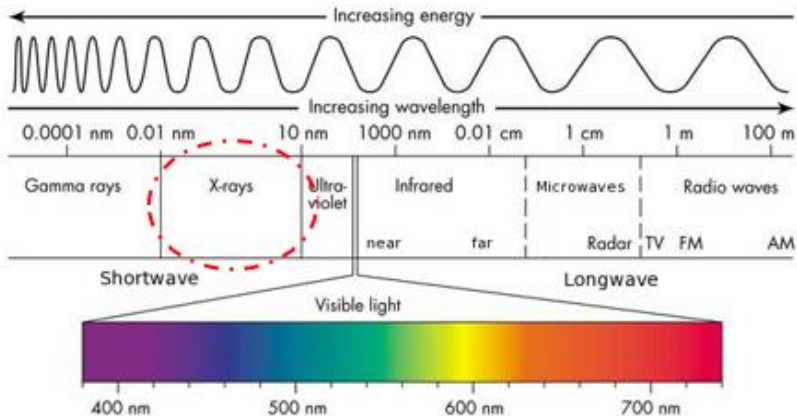
Gamma-Ray Imaging
In nuclear medicine



Gamma-Ray imaging of
A starburst galaxy about
12 million light-years
away

Images data preprocessing

X- rays



Images data preprocessing

X- rays

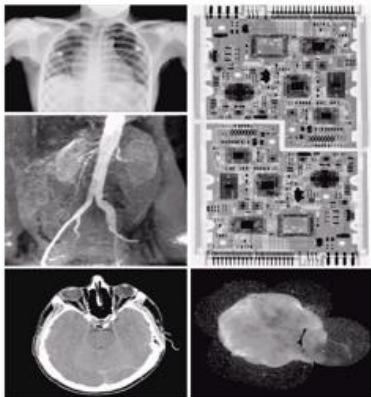
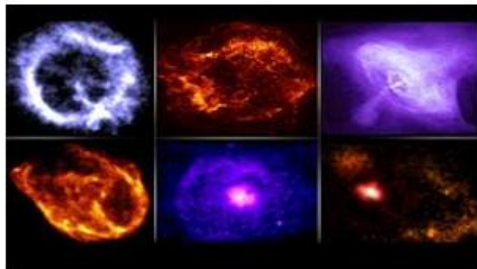


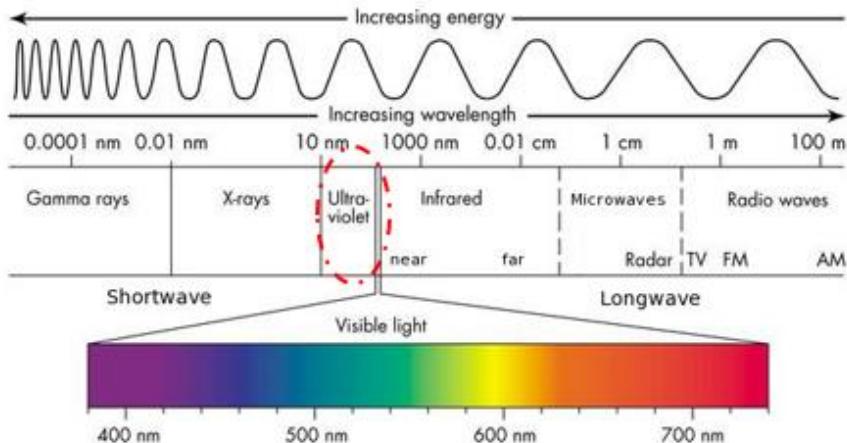
FIGURE 1.7 Examples of X-ray imaging. (a) Chest X-ray. (b) Aortic angiogram. (c) Head CT. (d) Circuit boards. (e) Cygnus Loop. (Images courtesy of (a) and (c) Dr. David R. Pickens, Dept. of Radiology & Radiological Sciences, Vanderbilt University Medical Center; (b) Dr. Thomas R. Gent, Division of Anatomical Sciences, University of Michigan Medical School; (d) Mr. Joseph E. Pascente, Lick, Inc.; and (e) NASA.)



**X-ray images from the space
The Chandra X-Ray Observatory**

Images data preprocessing

Ultra-violet

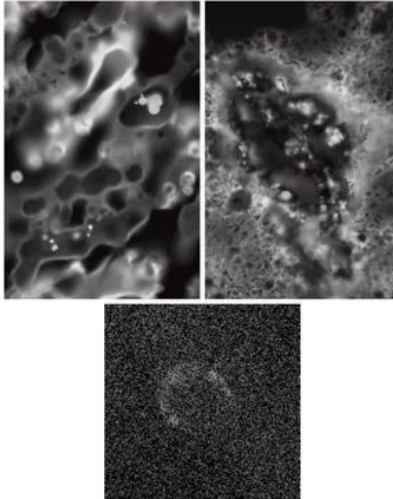


Images data preprocessing

Ultra-violet

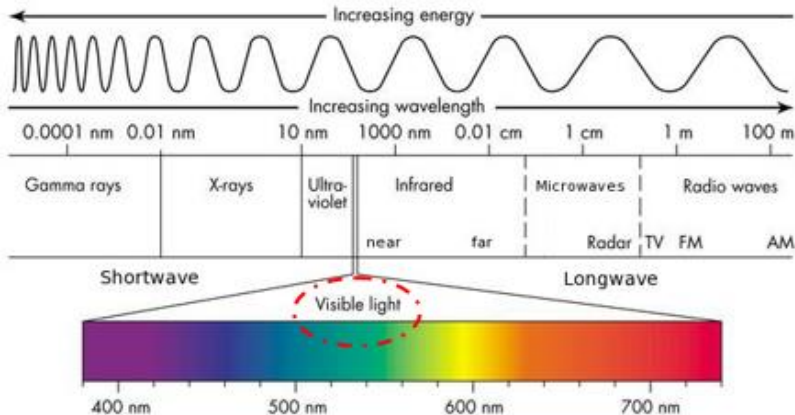
a b
c

FIGURE 1.8
Examples of
ultraviolet
imaging.
(a) Normal corn.
(b) Smut corn.
(c) Cygnus Loop.
(Images courtesy
of (a) and
(b) Dr. Michael
W. Davidson,
Florida State
University,
(c) NASA.)



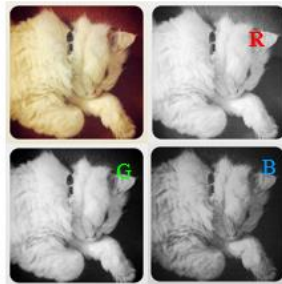
Images data preprocessing

Visible light



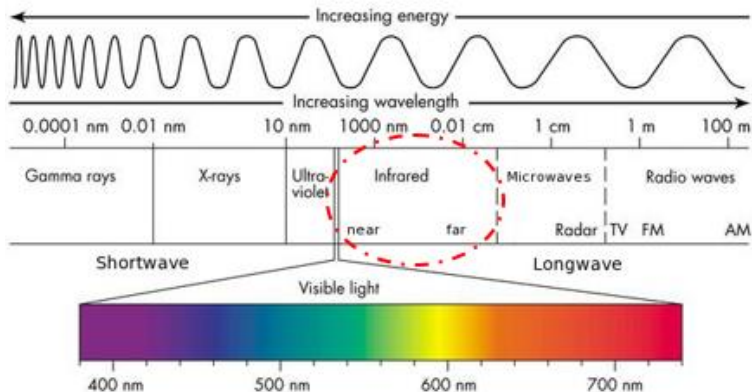
Images data preprocessing

Visible light



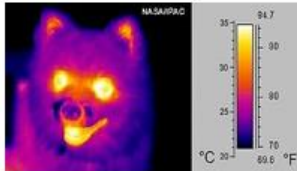
Images data preprocessing

Infrared

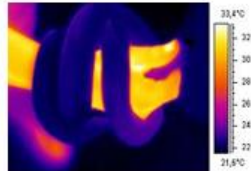


Images data preprocessing

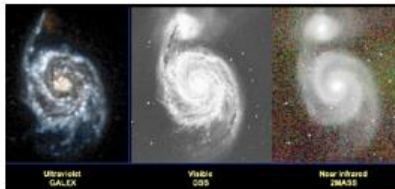
Infrared



infrared ("thermal") image



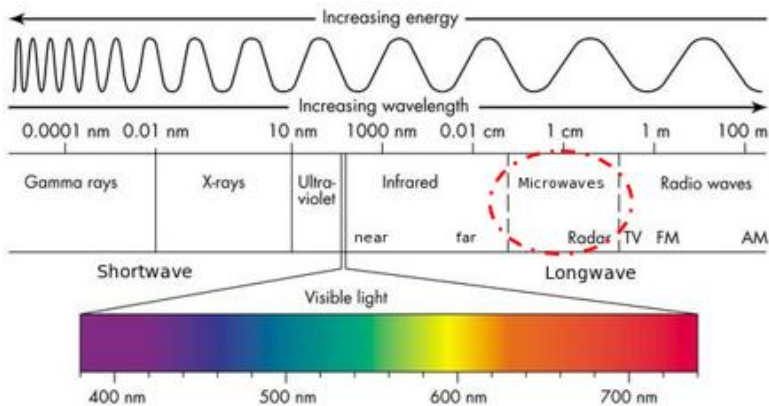
Snake around the arm



Messier 51 in ultraviolet (GALEX), visible (DSS), and **near infrared** (2MASS). Courtesy of James Fanon.

Images data preprocessing

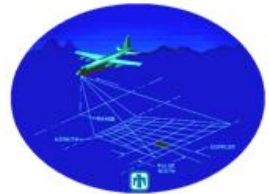
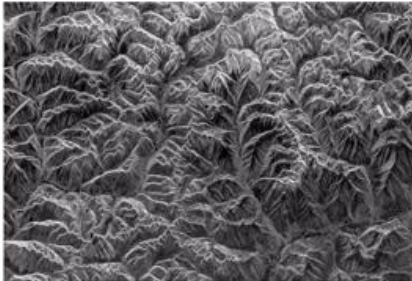
Microwaves



Images data preprocessing

Microwaves

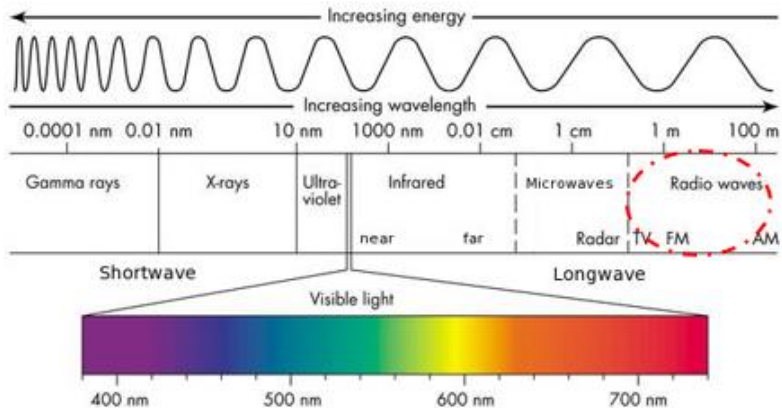
FIGURE 1.16
Spaceborne radar
image of
mountains in
southeast Tibet.
(Courtesy of
NASA.)



**Synthetic Aperture Radar
System**

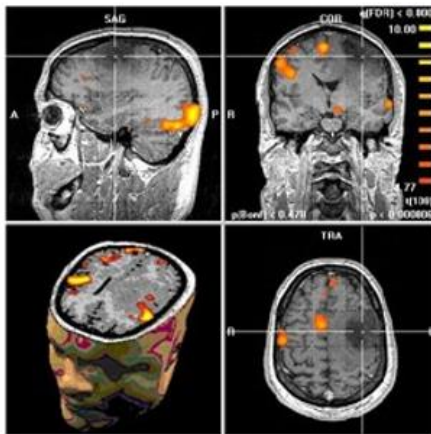
Images data preprocessing

Radio Waves



Images data preprocessing

Radio Waves



MRI image slices from the brain

Images data preprocessing

Digital Images based on the EM Spectrum

An example showing Imaging in all of the bands

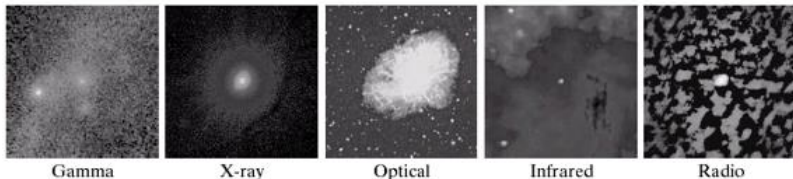
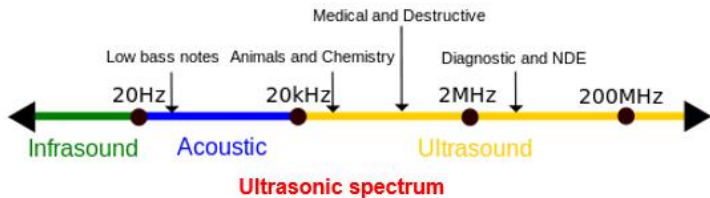


FIGURE 1.18 Images of the Crab Pulsar (in the center of images) covering the electromagnetic spectrum. (Courtesy of NASA.)

Visible light

Images data preprocessing

Ultrasound Imaging



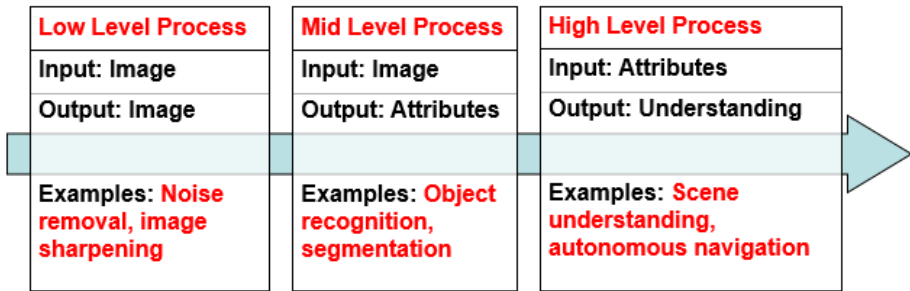
Ultrasonic Baby image during pregnancy



Ultrasound image acquisition device

Images data preprocessing

The continuum from image processing to computer vision can be broken up into low-, mid- and high-level processes



Techniques for Image Preprocessing

1. Image Resizing
2. Noise Reduction
3. Color Correction
4. Contrast Enhancement
5. Segmentation
6. Lighting Adjustment
7. Visualization
8. Data Split
9. Data Augmentation
10. Normalization
11. Feature Extraction

Techniques for Image Preprocessing

❖ Image Resizing:

- Image resizing techniques are used to adjust the size of an image.
- Resizing can be done to make an image smaller or larger or to change its aspect ratio.
- Some typical image resizing techniques include nearest neighbor interpolation, bilinear interpolation, and bicubic interpolation

Techniques for Image Preprocessing

❖ **Noise Reduction:**

- Noise in an image can be caused by various factors such as low light, sensor noise, and compression artifacts.
- Noise reduction techniques aim to remove noise from the image while preserving its essential features.
- Gaussian smoothing, median filtering, and wavelet denoising

Techniques for Image Preprocessing

❖ **Color Correction:**

- Color correction techniques are used to adjust the color balance of an image.
- Color correction is important in applications such as photography, where the color accuracy of an image is critical.
- Techniques include gray world assumption, white balance, and color transfer.

Techniques for Image Preprocessing

❖ Contrast Enhancement:

- Contrast enhancement techniques aim to increase the contrast of an image, making it easier to distinguish between different image features.
- These techniques can be helpful in applications such as medical imaging and surveillance.
- Techniques include histogram equalization, adaptive histogram equalization, and contrast stretching

Techniques for Image Preprocessing

❖ Segmentation:

- Segmentation techniques are used to divide an image into regions based on its content.
- Segmentation can be helpful in applications such as medical imaging, where specific structures or organs must be isolated from the image.
- Techniques include thresholding, edge detection, and region growing

Techniques for Image Preprocessing

❖ Data Augmentation:

- Techniques include horizontal & vertical flipping, rotation, cropping, shearing, etc.

❖ Normalization

- Normalize pixel values to a common scale (e.g., 0 to 1 or -1 to 1) to make the data suitable for machine learning algorithms.

Techniques for Image Preprocessing

❖ Feature Extraction:

- Feature extraction techniques are used to identify and extract relevant features from an image.
- These features can be used in object recognition and image classification applications.
- Some standard feature extraction techniques include edge detection, corner detection, and texture analysis, feature descriptors like SIFT or HOG

Images data preprocessing

❖ Image processing tools:

- OpenCV.
- Scikit-image.
- PIL/pillow

Images data preprocessing



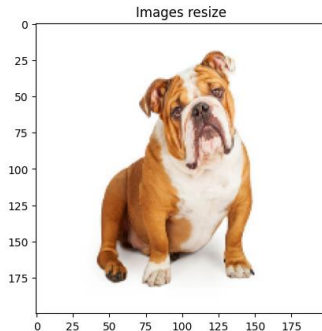
```
pic1 = plt.imread('/content/dog.jpg')  
plt.imshow(pic1)
```

<matplotlib.image.AxesImage at 0x7db9a2ac0ee0>



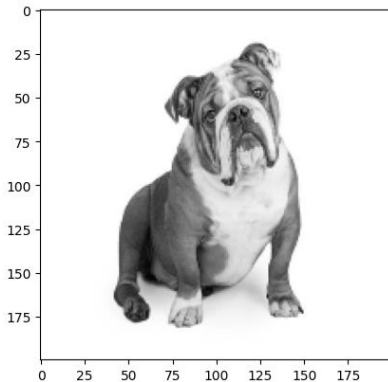
Images data preprocessing

```
# setting dim of the resize  
height = 200  
width = 200  
dim = (width, height)  
res_img = cv2.resize(pic1, dim, interpolation=cv2.INTER_LINEAR)  
plt.imshow(res_img), plt.title("Images resize")  
plt.show()
```



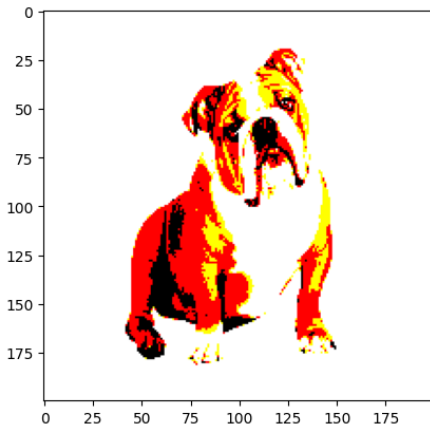
Images data preprocessing

```
#Converting the sample image to grayscale  
img = cv2.cvtColor(res_img, cv2.COLOR_BGR2GRAY)  
plt.imshow(img, cmap='gray')
```



Images data preprocessing

```
#Thresholding: try playing with the threshold value (144 here) to see the changes  
ret, thresh1 = cv2.threshold(res_img, 140, 255, cv2.THRESH_BINARY)  
plt.imshow(thresh1, cmap='gray')
```

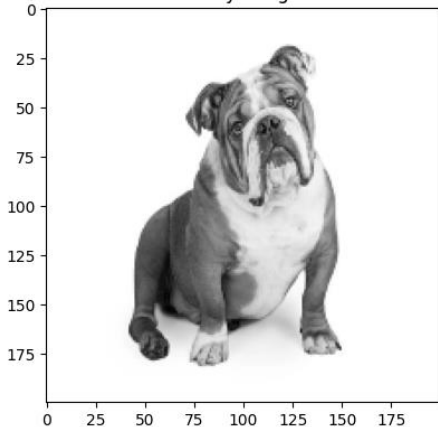


Images data preprocessing

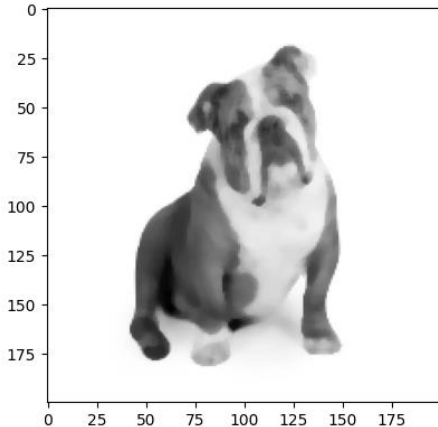
```
#Noise Reduction
#Median filter
img = cv2.cvtColor(res_img, cv2.COLOR_BGR2GRAY)
median = cv2.medianBlur(img,5)
plt.figure(figsize=(10, 10))
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Noisy Image')
plt.subplot(122),
plt.imshow(median,cmap = 'gray')
plt.title('Median filter')
plt.show()
```

Images data preprocessing

Noisy Image



Median filter



Images data preprocessing

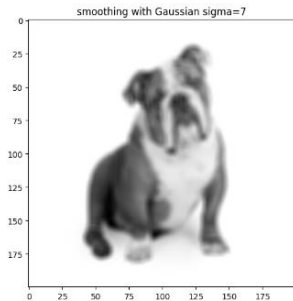
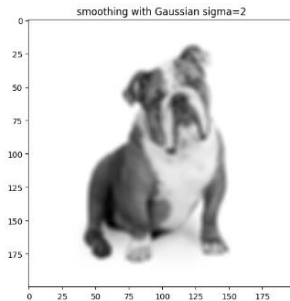
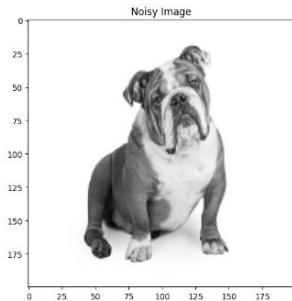
```
gaussian_blur1 = cv2.GaussianBlur(img,(5,5),2,cv2.BORDER_DEFAULT)
gaussian_blur2 = cv2.GaussianBlur(img,(5,5),7,cv2.BORDER_DEFAULT)

plt.figure(figsize=(20, 20))
plt.subplot(1,3,1),plt.imshow(img,cmap = 'gray')
plt.title('Noisy Image')

plt.subplot(1,3,2),
plt.imshow(gaussian_blur1,cmap = 'gray')
plt.title('smoothing with Gaussian sigma=2')

plt.subplot(1,3,3),
plt.imshow(gaussian_blur2,cmap = 'gray')
plt.title('smoothing with Gaussian sigma=7')
```

Images data preprocessing



Images data preprocessing - Otsu's Thresholding

#Otsu's thresholding before and after Gaussian filtering

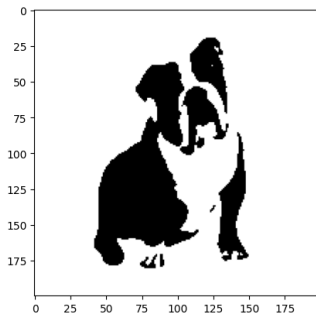
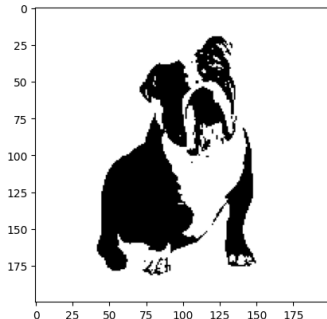
```
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

```
blur = cv2.GaussianBlur(img,(5,5),0)
```

```
ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

```
plt.imshow(th2,cmap='gray')
```

```
plt.imshow(th3,cmap='gray')
```



Images data preprocessing - Otsu's Thresholding

- Histogram is a visual representation of the number of pixels of each image's intensity value.
- The changes in histograms before and after applying thresholding on original and filtered images are shown below.

Images data preprocessing

```
plt.figure(figsize=(16,16))
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
# Otsu's thresholding
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
# Otsu's thresholding after Gaussian filtering
blur = cv2.GaussianBlur(img,(5,5),0)
ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([], plt.yticks([]))
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([], plt.yticks([]))
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([], plt.yticks([]))
plt.show()
```


Images data preprocessing

Original Noisy Image



Histogram



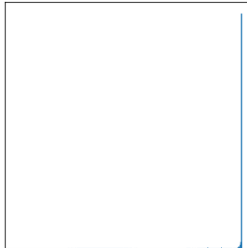
Global Thresholding ($v=127$)



Original Noisy Image



Histogram



Otsu's Thresholding

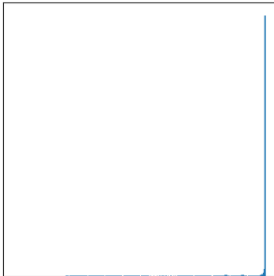


Images data preprocessing

Gaussian filtered Image



Histogram



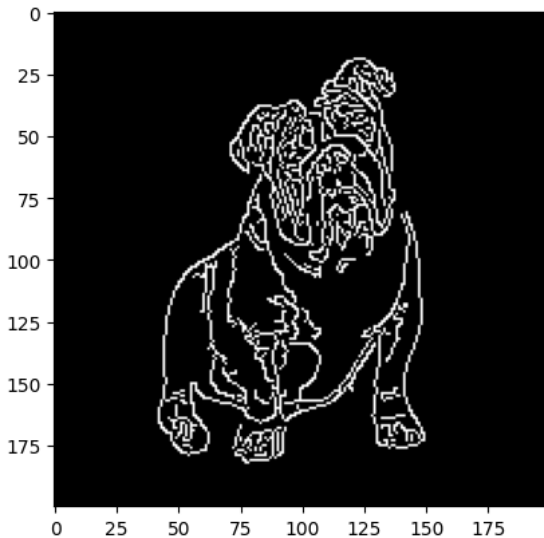
Otsu's Thresholding



Images data preprocessing - Canny Edge Detection

```
#Hough Line Transform
dst = cv2.Canny(img, 50, 200, None, 3)
lines = cv2.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)
# Draw the lines
if lines is not None:
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = math.cos(theta)
        b = math.sin(theta)
        x0 = a * rho
        y0 = b * rho
        pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
        pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
        cv2.line(dst, pt1, pt2, (0,0,255), 3, cv2.LINE_AA)
cdst = cv2.cvtColor(dst, cv2.COLOR_GRAY2BGR)
plt.imshow(cdst)
```

Images data preprocessing



Images data preprocessing - Segmentation

```
# Segmentation
gray = cv2.cvtColor(res_img, cv2.COLOR_RGB2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Displaying segmented images
display(res_img, thresh, 'Original', 'Segmented')
# Further noise removal (Morphology)
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)

# sure background area
sure_bg = cv2.dilate(opening, kernel, iterations=3)

# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)

# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

#Displaying segmented back ground
display(res_img, sure_bg, 'Original', 'Segmented Background')
```

Images data preprocessing - Segmentation

Original



Segmented



Original



Segmented Background

