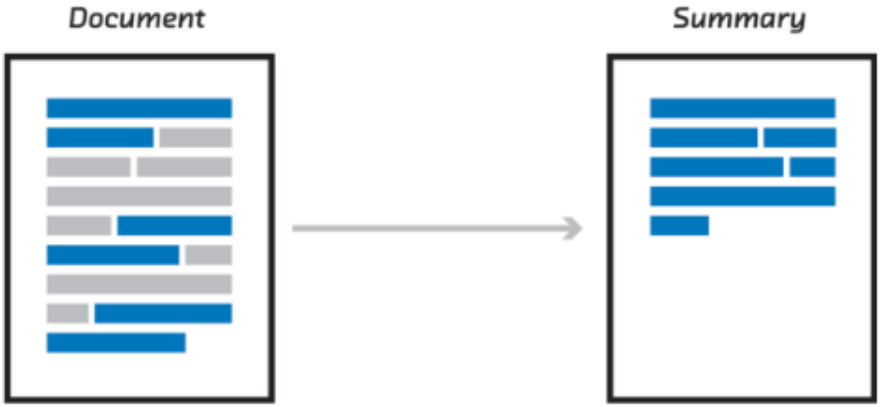
Xử lý ngôn ngữ tự nhiên

Chương 5: Information Retrieval

Khoa CNTT, Đại học Kỹ thuật - Công nghệ Cần Thơ
Lưu hành nội bộ

Nội dung


Document
Summary

Text Summarization xuất phát từ đâu?

- Truyền thống:
  - Sách, báo số theo ngày, tạp chí số theo tuần
  - Dữ liệu theo chuẩn để tóm tắt văn bản ở nhiều mức chi tiết
- Thời đại mạng xã hội:
  - Sách, báo, tạp chí tính theo phút
  - Dữ liệu văn bản mạng xã hội
  - Dữ liệu không theo chuẩn, dữ liệu loãng, dữ liệu nhiễu
  - Vượt quá khả năng đọc của con người

Text Summarization là gì?

- Gọi corpus là một tập hợp các documents
- Bài toán đặt ra: hiểu ý nghĩa của các documents
  - Trích lọc từ khóa: keyphrase extraction
  - Trích lọc chủ đề: topic modeling
  - Tóm tắt văn bản theo đoạn, theo documents, theo corpus
  - Tự động hóa tác vụ tóm tắt văn bản: automated document summarization

Text Summarization và toán xác suất

- Khi nói document i thuộc về topic (chủ đề) j là bao nhiêu %
- Khi nói word/token i giống hay khác word/token j là bao nhiêu %

Các khái niệm căn bản

- Document
- Tokenization
- Text normalization
- Feature extraction
- Feature matrix

Các khái niệm căn bản: Feature extraction

- Binary term occurrence-based features
- Frequency bag of words-based features
- TFIDF-weighted features

Các khái niệm căn bản

- Cho dữ liệu: sample_text = "The brown fox wasn't that quick and he couldn't win the race."
- Hãy trình bày các khái niệm căn bản ở slide trước

Keyphrase extraction

- Kỹ thuật căn bản đối với dữ liệu không cấu trúc
- Mục tiêu: tìm ra thông tin quan trọng
- Thể hiện qua các key words, repeated words, nouns, relevant phrases
  - Semantic web
  - Search engines
  - Recommendation systems
  - Tagging systems

Text summarization dựa trên frequency

```
from nltk.tokenize import sent_tokenize,word_tokenize
from nltk.corpus import stopwords
from collections import defaultdict
from string import punctuation
from heapq import nlargest

class Summarize_Frequency:
    def __init__(self, cut_min=0.2, cut_max=0.8):
        """ Initilize the text summarizer.
        Words that have a frequency term lower than cut_min
        or higer than cut_max will be ignored. """
        self._cut_min = cut_min
        self._cut_max = cut_max
        self._stopwords = set(stopwords.words('english') + list(
    punctuation))
```

```
15
16      def _compute_frequencies(self, word_sent):
17        """ Compute the frequency of each of word.
18        Input: word_sent, a list of sentences already tokenized.
19        Output: freq, a dictionary where freq[w] is the frequency of w.
20        """
21        freq = defaultdict(int)
22        for s in word_sent:
23            for word in s:
24                if word not in self._stopwords:
25                    freq[word] += 1
26        # frequencies normalization and fitering
27        m = float(max(freq.values()))
28        for w in list(freq.keys()):
29            freq[w] = freq[w]/m
30            if freq[w] >= self._cut_max or freq[w] <= self._cut_min:
31                del freq[w]
32        return freq
```

```
33
34      def summarize(self, text, n):
35          """ list of (n) sentences are returned. summary of text is
        returned. """
36          sents = sent_tokenize(text)
37          assert n <= len(sents)
38          word_sent = [word_tokenize(s.lower()) for s in sents]
39          self._freq = self._compute_frequencies(word_sent)
40          ranking = defaultdict(int)
41          for i,sent in enumerate(word_sent):
42              for w in sent:
43                  if w in self._freq:
44                      ranking[i] += self._freq[w]
45          sents_idx = self._rank(ranking, n)
46          return [sents[j] for j in sents_idx]
```

```
47
48      def _rank(self, ranking, n):
49          """ return the first n sentences with highest ranking """
50          return nlargest(n, ranking, key=ranking.get)
```

```
51
52 toy_text = "Elephants are large mammals of the family Elephantidae and
       the order Proboscidea. Two species are traditionally recognised,
       the African elephant and the Asian elephant. Elephants are
       scattered throughout sub-Saharan Africa, South Asia, and Southeast
       Asia. Male African elephants are the largest extant terrestrial
       animals. All elephants have a long trunk used for many purposes,
       particularly breathing, lifting water and grasping objects. Their
       incisors grow into tusks, which can serve as weapons and as tools
       for moving objects and digging. Elephants' large ear flaps help to
       control their body temperature. Their pillar-like legs can carry
       their great weight. African elephants have larger ears and concave
       backs while Asian elephants have smaller ears and convex or level
       backs."
```

```
53
54 sumFre = Summarize_Frequency()
55
56 #summarize input text by 2 most important sentences that summarize the
       text's content
57 print(sumFre.summarize(toy_text,2))
58
59 #['African elephants have larger ears and concave backs while Asian
       elephants have smaller ears and convex or #level backs.', 'Two
       species are traditionally recognised, the African elephant and the
       Asian elephant.']
```

- Phần codes trình bày ở trên có thể áp dụng cho tiếng việt
- Tuy nhiên, phần codes text normalization không được thực hiện. Tại sao?
- Ví dụ thử đoạn text ở `https://www.thongtincongnghe.com/article/75136`

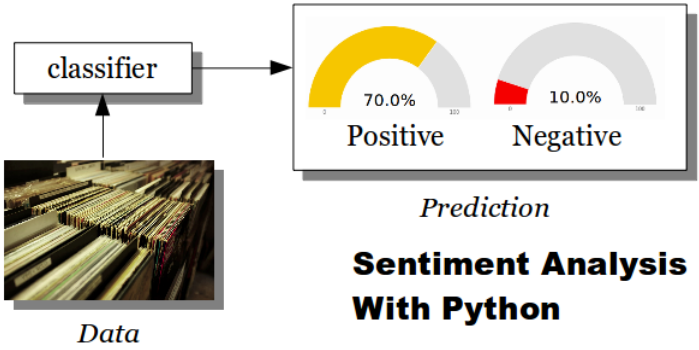
Question-answering system

- Hệ thống hỏi-trả lời là hệ thống thông minh có khả năng trả lời những câu hỏi do người dùng nhập vào
- Tác vụ đầu tiên trong hệ thống chính là tác vụ XLP: làm sạch văn bản
  - Bao gồm?

```
import nltk
from nltk import *
import string

print("enter your question")
question = input()

question = question.lower()
stopwords = nltk.corpus.stopwords.words("english")
cont = nltk.word_tokenize(question)
analysis_keywords = list(set(cont) - set(stopwords))
print(analysis_keywords)
```

Sentiment analysis - phân tích cảm tính

- Biểu diễn cảm tính là một dạng của bài toán phân loại
  - Đầu vào: văn bản
  - Đầu ra: phân loại thành các categories. Ví dụ: positive, negative



Sentiment analysis - ví dụ 1

- Classification bao gồm 2 bước: training và prediction
- Đinh nghĩa các categories cần phân loại

```
positive_vocab = [ 'awesome', 'outstanding', 'fantastic', 'terrific', '
    good', 'nice', 'great', ':)' ]
negative_vocab = [ 'bad', 'terrible','useless', 'hate', ':(' ]
neutral_vocab = [ 'movie','the','sound','was','is','actors','did','know
    ','words','not' ]
```

- Chuyển đổi mỗi word thành feature sử dụng mô hình bag of word

```
def word_feats(words):
    return dict([(word, True) for word in words])

positive_features = [(word_feats(pos), 'pos') for pos in positive_vocab]
negative_features = [(word_feats(neg), 'neg') for neg in negative_vocab]
neutral_features = [(word_feats(neu), 'neu') for neu in neutral_vocab]
```

- Thực hiện tác vụ phân loại

Codes ví dụ 1

```
import nltk.classify.util
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import names

def word_feats(words):
    return dict([(word, True) for word in words])

positive_vocab = [ 'awesome', 'outstanding', 'fantastic', 'terrific', 'good', 'nice', 'great', ':)' ]
negative_vocab = [ 'bad', 'terrible','useless', 'hate', ':(' ]
neutral_vocab = [ 'movie','the','sound','was','is','actors','did','know','words','not' ]
```

```
positive_features = [(word_feats(pos), 'pos') for pos in positive_vocab]
negative_features = [(word_feats(neg), 'neg') for neg in negative_vocab]
neutral_features = [(word_feats(neu), 'neu') for neu in neutral_vocab]

train_set = negative_features + positive_features + neutral_features

classifier = NaiveBayesClassifier.train(train_set)

# Predict
neg = 0
pos = 0
sentence = "Awesome movie, I liked it"
sentence = sentence.lower()
words = sentence.split(' ')
```

```
26
27 for word in words:
28     classResult = classifier.classify( word_feats(word))
29     if classResult == 'neg':
30         neg = neg + 1
31     if classResult == 'pos':
32         pos = pos + 1
33
34 print('Positive: ' + str(float(pos)/len(words)))
35 print('Negative: ' + str(float(neg)/len(words)))
36
37 #Positive: 0.6
38 #Negative: 0.2
```

Sentiment analysis - ví dụ 2

- Cài đặt thư viện wordcloud tại `https://anaconda.org/conda-forge/wordcloud`

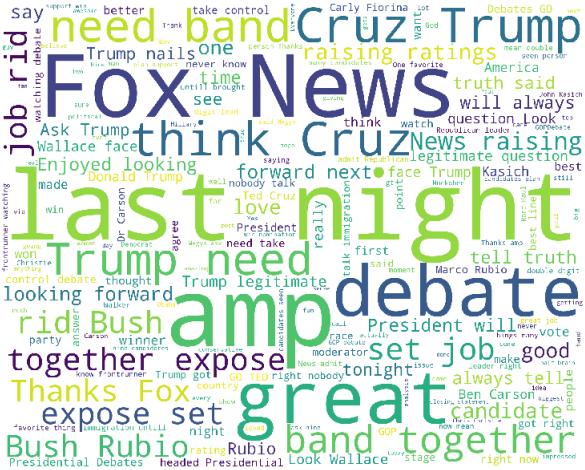
```
import numpy as np
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split

import nltk
from nltk.corpus import stopwords
from nltk.classify import SklearnClassifier

from wordcloud import WordCloud,STOPWORDS
import matplotlib.pyplot as plt
%matplotlib inline
```

```
12
13 data = pd.read_csv('Sentiment.csv')
14 # Keeping only the neccessary columns
15 data = data[['text','sentiment']]
16 print(data.head())
17
18 #                                                text sentiment
19 #0  RT @NancyLeeGrahn: How did everyone feel about...   Neutral
20 #1  RT @ScottWalker: Didn't catch the full #GOPdeb...  Positive
21 #2  RT @TJMShow: No mention of Tamir Rice and the ...   Neutral
22 #3  RT @RobGeorge: That Carly Fiorina is trending ...  Positive
23 #4  RT @DanScavino: #GOPDebate w/ @realDonaldTrump...  Positive
```

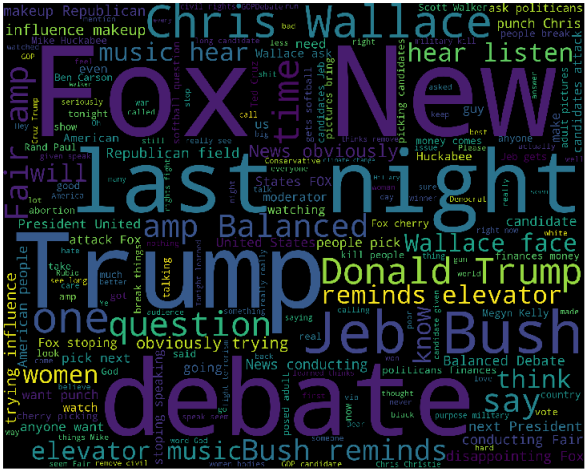
```
24
25 # Splitting the dataset into train and test set
26 train, test = train_test_split(data,test_size = 0.25)
27 # Removing neutral sentiments
28 train = train[train.sentiment != "Neutral"]
29
30 train_pos = train[ train['sentiment'] == 'Positive']
31 train_pos = train_pos['text']
32 train_neg = train[ train['sentiment'] == 'Negative']
33 train_neg = train_neg['text']
```

```
34
35 def wordcloud_draw(data, color = 'black'):
36     words = ' '.join(data)
37     cleaned_word = " ".join([word for word in words.split()
38                 if 'http' not in word
39                     and not word.startswith('@')
40                     and not word.startswith('#')
41                     and word != 'RT'
42                 ])
43     wordcloud = WordCloud(stopwords=STOPWORDS,
44                 background_color=color,
45                 width=2500,
46                 height=2000
47                 ).generate(cleaned_word)
48     plt.figure(1,figsize=(13, 13))
49     plt.imshow(wordcloud)
50     plt.axis('off')
51     plt.show()
```

```
print("Positive words")
wordcloud_draw(train_pos,'white')
#Positive words
```



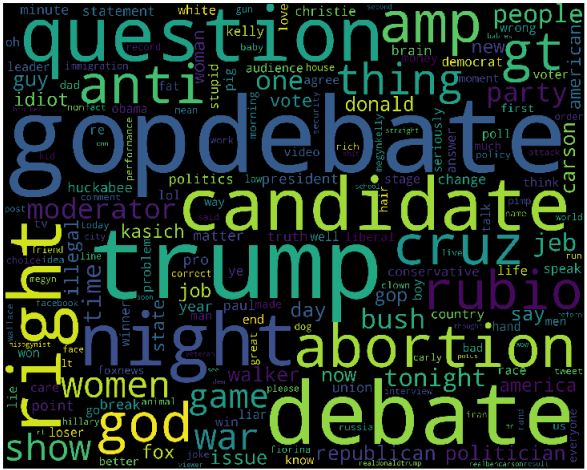
```
55 print("Negative words")
56 wordcloud_draw(train_neg)
57 #Negative words
```



```
58 tweets = []
59 stopwords_set = set(stopwords.words("english"))
60
61 for index, row in train.iterrows():
62     words_filtered = [e.lower() for e in row.text.split() if len(e) >=
       3]
63     words_cleaned = [word for word in words_filtered
64         if 'http' not in word
65         and not word.startswith('@')
66         and not word.startswith('#')
67         and word != 'RT']
68     words_without_stopwords = [word for word in words_cleaned if not
       word in stopwords_set]
69     tweets.append((words_cleaned,row.sentiment))
70
71 test_pos = test[ test['sentiment'] == 'Positive']
72 test_pos = test_pos['text']
73 test_neg = test[ test['sentiment'] == 'Negative']
74 test_neg = test_neg['text']
```

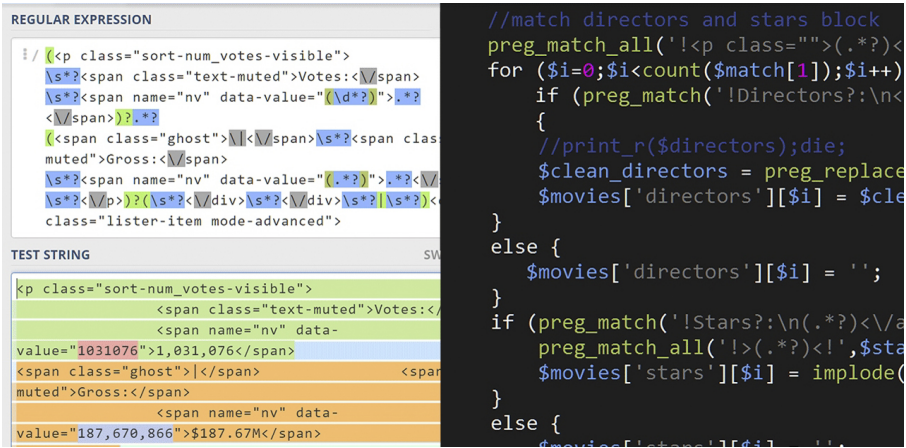
```
75
76 # Extracting word features
77 def get_words_in_tweets(tweets):
78     all = []
79     for (words, sentiment) in tweets:
80         all.extend(words)
81     return all
82
83 def get_word_features(wordlist):
84     wordlist = nltk.FreqDist(wordlist)
85     features = wordlist.keys()
86     return features
87
88 w_features = get_word_features(get_words_in_tweets(tweets))
```

```
89
90 wordcloud_draw(w_features)
```



```
91
92 def extract_features(document):
93     document_words = set(document)
94     features = {}
95     for word in w_features:
96         features['containts(%s)' % word] = (word in document_words)
97     return features
98
99 # Training the Naive Bayes classifier
100 training_set = nltk.classify.apply_features(extract_features,tweets)
101 classifier = nltk.NaiveBayesClassifier.train(training_set)
```

```
102
103 neg_cnt = 0
104 pos_cnt = 0
105 for obj in test_neg:
106     res =  classifier.classify(extract_features(obj.split()))
107     if(res == 'Negative'):
108         neg_cnt = neg_cnt + 1
109 for obj in test_pos:
110     res =  classifier.classify(extract_features(obj.split()))
111     if(res == 'Positive'):
112         pos_cnt = pos_cnt + 1
113
114 print('[Negative]: %s/%s '  % (len(test_neg),neg_cnt))
115 print('[Positive]: %s/%s '  % (len(test_pos),pos_cnt))
116
117 #[Negative]: 2114/1995
118 #[Positive]: 584/205
```


REGULAR EXPRESSION
/ (<p class="sort-num_votes-visible">
\s*?Votes:<\/span>
\s*?.*?
<\/span>)?.*?
(\|<\/span>\s*?<span clas
muted">Gross:<\/span>
\s*?.*?<\/
\s*?<\/p>)?(\s*?<\/div>\s*?<\/div>\s*?|\s*?)<
class="lister-item mode-advanced">
TEST STRING
<p class="sort-num_votes-visible">
Votes:</
<span name="nv" data-
value="1031076">1,031,076
|
muted">Gross:
<span name="nv" data-
value="187,670,866">$187.67M
//match directors and stars block
preg_match_all('!<p class="">(.*?)<
for ($i=0;$i<count($match[1]);$i++)
if (preg_match('!Directors?:\n<
{
//print_r($directors);die;
$clean_directors = preg_replace
$movies['directors'][$i] = $cle
}
else {
$movies['directors'][$i] = '';
}
if (preg_match('!Stars?:\n(.*?)<\/a
preg_match_all('!>(.*?)<!',$sta
$movies['stars'][$i] = implode(
}
else {

Web scraping and parsing

- Thu thập nội dung chữ từ trang web
- Áp dụng mọi tác vụ NLP
- Thu thập data
- Thư viện `Beautiful Soup` `https://www.crummy.com/software/BeautifulSoup/bs4/doc/#`
- `https://anaconda.org/anaconda/beautifulsoup4`