Ling575 Summarization System

D3: Initial System

Rachel Hantz, Yi-Chien Lin, Yian Wang, Chenxi Li, Tashi Tsering

Overview

System Architecture

System Implementation

- Method 1: Term Frequency - Inverse Document Frequency (TF-IDF)
- Method 2: Integer Linear Programming (ILP)

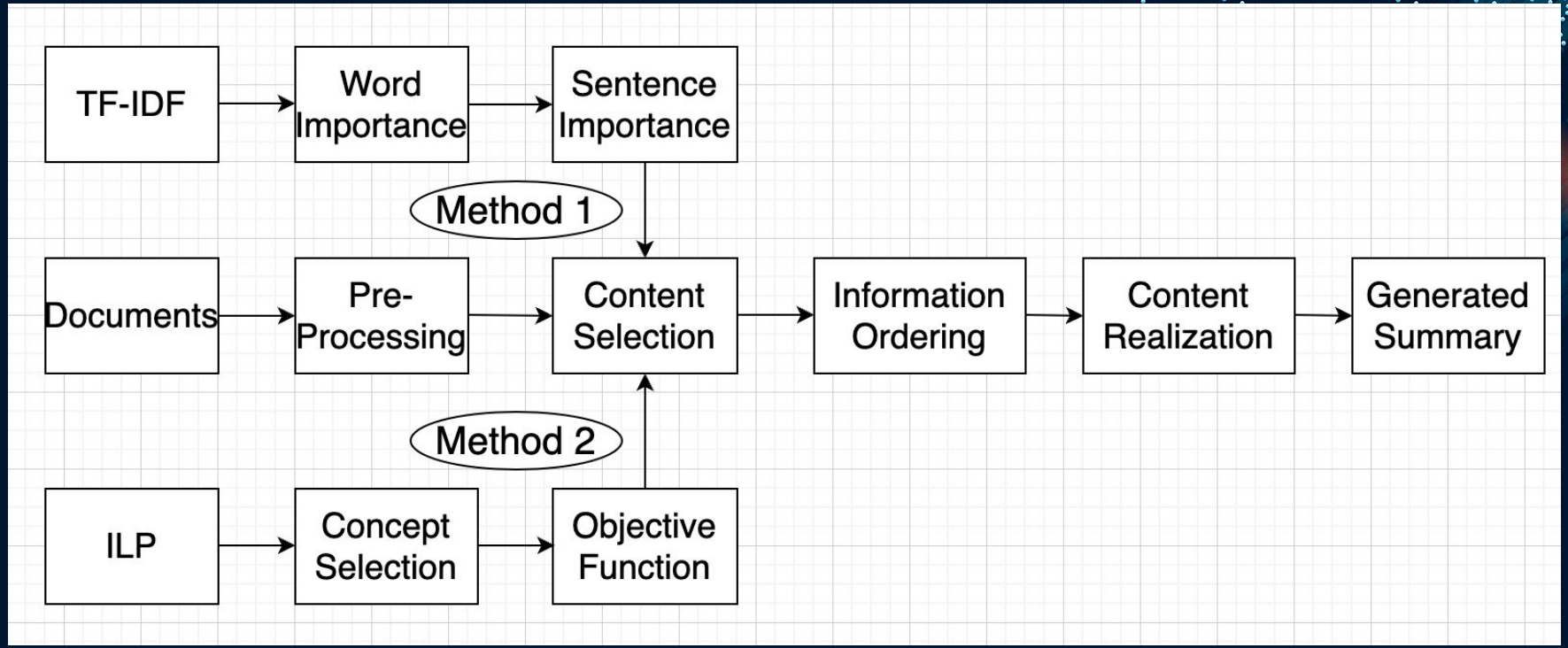
Results

Demo

WorkSplit

- TF-IDF - Tashi & Chenxi
- ILP - Rachel
- Evaluation & Pipeline - Yi-Chien
- Report & Slides - Everyone

System Architecture



Method 1: TF-IDF

Build TF-IDF model

- tf(t, d) * (log [(1+n) / (1+df(t))] + 1)
  - Background: all documents in devtest directory
  - Foreground: ten documents under one topic
- Exclude stop words and punctuation

Sentence Selection

- Store data (index, topic_id, length) in a matrix
- Calculate sentence TF-IDF score
- Pick sentences
  - Token number in a sentence
  - Cos_similarity between sentences

Method 1: TF-IDF

Utilize TfidfVectorizer to build the TF-IDF matrix

- Token pattern defined as "r'\b[^\s]+\b'", to match one or more characters that are not white spaces, and are surrounded by word boundaries
- Create a dictionary to map the score to each token

```
vectorizer = TfidfVectorizer(stop_words = stop_word, token_pattern=r'\b[^\s]+\b') #, ngram_range=(2, 2))
tfidf_matrix = vectorizer.fit_transform(all_set)
feature_names = vectorizer.get_feature_names_out()
tfidf_dict1 = dict(zip(feature_names, tfidf_matrix.toarray().T))
```

Store information for each tokenized sentence into a Panda dataframe

- The DataFrame consists of 5 columns
- Sorted first by the document set, and then by the tf-idf scores within each document set.

```
df = pd.DataFrame({'Topic_ID':A, 'sentence': B, 'word_count_no_stop': C, 'word_count': D, 'tfidf_scores': E})
df = df.sort_values(by = ['Doc_set', 'tfidf_scores'], ascending = [True, False])
```

Method 1: TF-IDF

Selecting sentences

- $S_w = \Sigma(w_i) / n$
- Skip short sentences
  - Exclude stopwords
  - Tested w/o punctuation
- Skip similar sentences
  - Tested diff cos_similarity
- No more than 100

```
v_curr = vectorizer.transform([row[1]])
total_cos_simi = 0
for sent in my_tuple[1]:
    v_selected = vectorizer.transform([sent])
    total_cos_simi += cosine_similarity(v_curr, v_selected)
avg_cos_simi = total_cos_simi / len(my_tuple[1])
if  avg_cos_simi > cosine_sim:
    continue
```

```
if row[3] < word_limit:
    pass
else:
    if len(my_tuple[1]) == 0:
        first_string = row[1].strip('\n')
        my_tuple[1].append(first_string)
        w_c += row[3]
        continue
```

Method 1: TF-IDF

TF-IDF table

Punct	WB	Cos_simi	R1-R	R1-P	R1-F	R2-R	R2-P	R2-F
YES	10	NA	0.29292	0.34907	0.31728	0.06990	0.08310	0.07562
YES	10	0.3	0.30895	0.36750	0.33475	0.07160	0.08529	0.07761
N0	10	0.3	0.33809	0.35275	0.34346	0.07848	0.08122	0.07950
YES	10	0.4	0.30130	0.35664	0.32573	0.07026	0.08243	0.07564
YES	10	0.5	0.29866	0.35613	0.32383	0.07016	0.08355	0.07601
YES	10	0.6	0.29705	0.35435	0.32207	0.07066	0.08403	0.07649

Method 2: Integer Linear Programming

Get Concepts

- Stemmed bigrams
- No punctuation
- No stop-word only bigrams

```
# Stems all but stop words
stemmer = SnowballStemmer("english", ignore_stopwords=True)
# tokenizes and removes punctuation (does not remove _ )
tokenizer = RegexpTokenizer(r'\w+')
```

```
sent_stemmed = [stemmer.stem(word) for word in tokenizer.tokenize(sent)]
sent_concepts = {bigram for bigram in bigrams(sent_stemmed) if bigram[0] not in stop_words and bigram[1] not in stop_words}
```

- Weights = # of articles concept is found in.

```
return Counter({concept: weight for concept, weight in concepts.items() if weight >= 3})
```

Build Occurrence Matrix →

```
occurence = {}
for i, concept_i in enumerate(concepts):
    for j, sentence_j in enumerate(sentences):
        if concept_i in sentence_j["concepts"]:
            occurence[(i, j)] = 1
        else:
            occurence[(i, j)] = 0
```

Method 2: Integer Linear Programming

Use PuLP with GLPK solver - (tutorial)

- Define Model and Decision Variables

```
# Implement ILP model

model = LpProblem(name="content-selector", sense=LpMaximize)

# Define the decision variables

s = {j: LpVariable(name=f"s{j}", cat="Binary") for j in range(0, len(sentences))}
c = {i: LpVariable(name=f"c{i}", cat="Binary") for i in range(0, len(concepts))}
```

- Add Objective Function

```
objective_function = []
for i, concept in enumerate(concepts):
    objective_function.append(concept_weights[concept] * c[i])
model += lpSum(objective_function)
```

Constraints

- Included Concept in at least 1 Included Sentence

$$\sum_j s_j o_{ij} \geq c_i \forall i$$

- Included Sentences don't have not Included Concepts

$$s_j o_{ij} \leq c_i \forall i, j$$

- Length is no more than 100 tokens

```
# Length Constraint: All sentences chosen do not exceed 100 tokens
length_constraint = []
for j, sentence in enumerate(sentences):
    length_constraint.append(len(sentence["text"].strip().split()) * s[j])
model += (lpSum(length_constraint) <= max_length, "length_constraint")
```

Method 2: Integer Linear Programming

Solve and Export Summary!

```
# Solve the model -- decide which s_j's should be included
status = model.solve(solver=GLPK(msg=False))

# Collect the index (j) of each s_j that should be included (i.e. s_j = 1 in optimal solution)
sentences_in_summary = []
for var in s.values():
    if var.value() == 1:
        sentences_in_summary.append(sentences[int(var.name[1:])]["text"])

# Print to file
export_summary.export_summary(sentences_in_summary, topic_id, "2", "../outputs/D3")
```

Information Ordering

For now:

Print in order of appearance in article + in order of article date

Content Realization

For now:

Leave sentences as they are

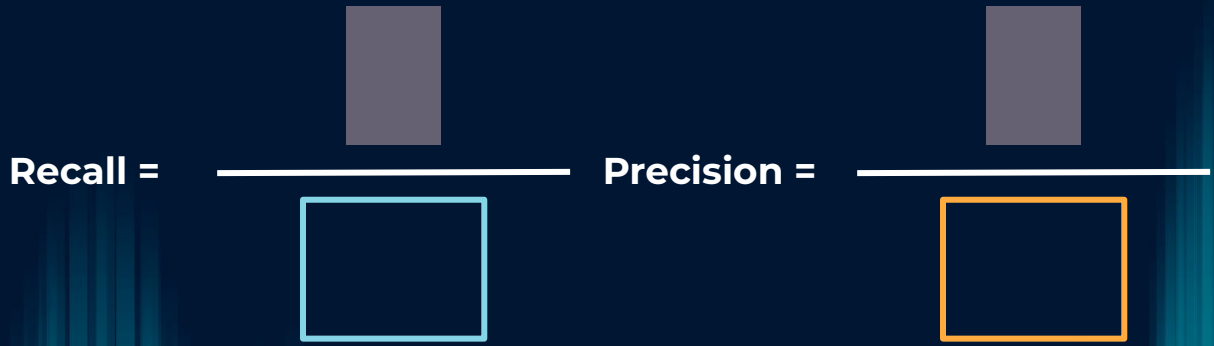
Demo

Results

- ROUGE evaluation metrics: ROUGE-N
  - Recall
  - Precision
  - F1-Score
- Metrics used for the current system:
  - ROUGE-1: Co-occurrence of unigrams in both system-generated summaries and human summaries
  - ROUGE-2: Co-occurrence of bigrams in both system-generated summaries and human summaries

System summary

Human summary



Results

- ROUGE python implementation:
  - rouge_score package
  - Allow stemming

Results

ROUGE-N	Metric	TF-IDF	ILP
ROUGE-1	Recall	0.30895	0.30750
	Precision	**0.36750**	0.35340
	F1-Score	**0.33475**	0.32734
ROUGE-2	Recall	0.07160	**0.07539**
	Precision	0.08529	**0.08723**
	F1-Score	0.07761	**0.08049**

Issues and Successes

Problem 1:

Description: Anaconda environment with multiple dependencies became tricky to export and re-install from others' branches

Solution: Create new environment file from scratch with clear dependencies

Problem 2:

Description: ROUGE implementation on patas was inaccessible due to missing perl file.

Solution: Use python rouge-score package instead

Related Reading

TFiDF -

- sklearn.feature_extraction.text.TfidfVectorizer
- sklearn.metrics.pairwise.cosine_similarity
- NLP — Text Summarization using NLTK: TF-IDF Algorithm

ILP -

- Hands-On Linear Programming: Optimization With Python
- Dan Gillick and Benoit Favre. (2009) A Scalable Global Model for Summarization., in Proceedings of the Workshop on ILP for NLP 2009.