Contribution to "Systematic review generator"

Alexandru Pintea

Aim 1: Feature extraction / ranking

(03/02 - 17/02)

Initial feature retrieval (PubMed only)

- Using the PMID from the CSV files I had, I fetched all the data that PubMed gave me
- The object that PubMed returned had functions/properties in it, from which all the properties were put in a Pandas Dataframe

```
# Putting all the features that PubMed returns into a pandas dataframe ...

from metapub import PubMedFetcher
import json
fetch = PubMedFetcher()

def hasmethod(obj, name): # the functions returned should be stored separately .
    return hasattr(obj, name) and ( "method" in str(type(getattr(obj, name))) )

all_article_data=[]
column_names=[]
function_names=[]
for i in range( 0, len( pmids ) ):
  article = fetch.article_by_pmid(pmids[i])
  article_data = {}
  for attr in dir( article ):
    if ( i == 0 ):
      if ( not hasmethod( article, str( attr ) ) ):
          column_names.append( attr )
      else:
          function_names.append( attr )
    if ( not hasmethod( article, str( attr ) ) ):
      article_data[ attr ] = getattr(article, attr)
  all_article_data.append( article_data )

df = pd.DataFrame( all_article_data, columns=column_names )
```

Initial feature retrieval (PubMed only)

- Drop the features that have the same value for all rows
- This is done with the "frequency" and "count" rows of the df.describe() table

```
# Dropping all columns that only have NaN or fully identical values ...

count_row = df.shape[0]  # Gives number of rows
freq_df=df.describe().loc[['freq']]
count_df=df.describe().loc[['count']]
for column in freq_df:
    if ( freq_df[column].iloc[0] == count_df[column].iloc[0] ):
      df = df.drop(column, axis=1)
    if ( pd.isna(freq_df[column].iloc[0]) ):
      df = df.drop(column, axis=1)
```

Initial feature retrieval (PubMed only)

- Remove columns containing redundant information
- e.g. leave only "authors" from the "author_list", "authors_str" and "authors" columns

```
remove_columns=[ "__dict__", "pii", "author1_lastfm", "author_l
# pcim / doi are redundant as they are found in the urls too ..
# pii contains the issn and other things ... no way to retrieve

if "author1_last_fm" in df.columns:
  df["author_first"]=df["author1_last_fm"]

for i in remove_columns:
  if i in df.columns:
    df = df.drop(i, axis=1)
df.describe()
```

Initial feature retrieval (PubMed only)

- Retrieve and structure reference data in a JSON, if available in the XML object that PubMed provided
- Here I could attempt to retrieve references from somewhere else, if PubMed does not have them …

```python
def get_references(row):
  xml_tree = ET.ElementTree(ET.fromstring(row["xml"]))
  have_citation=0
  for elem in xml_tree.iter():
    if ( "'ReferenceList'" in str( elem )  ):
      have_citation=1
      break

  # <Reference>
  #     <Citation>REFERENCE ARTICLE TITLE</Citation>
  #     <ArticleIdList>
  #         <ArticleId IdType="pmc">PMC ID</ArticleId>
  #         <ArticleId IdType="pubmed">PUBMED ID</ArticleId>
  #     </ArticleIdList>
  # </Reference>
```

Initial feature retrieval (PubMed only)

- To generate keywords for the about 60% of articles that had none, the following method was used:
0. removing all the common words from the title (done using lists of common words from wiki)
1. going though all the keywords that were found (from the articles that had them), to make a list of non-generated keywords/phrases
2. adding non-generated keywords/phrases to an article without keywords if they appear in its title
3. adding the final non-common title words to the keywords (the ones that remained = were not common and were not in non-generated keywords/phrases)

Initial feature retrieval (PubMed only)

- 23629 stop words (common words) were taken from Wikipedia to help identify the medical terms of the title, by means of exclusion
- Then, all the existing (non-generated) citations were put into an array

```
count_empty=0
count_non_empty=0
non_generated_keywords=[]
def determine_keywords(row):
  global non_generated_keywords
  if ( row["keywords"] != [] ):
    row["keywords"]=[x.lower() for x in row["keywords"]]
    global count_non_empty
    count_non_empty = count_non_empty + 1
    non_generated_keywords.extend(row["keywords"])
    # print( row["title"] )
    # print( str( row["keywords"] ) + "\n\n" )
  non_generated_keywords = sorted(non_generated_keywords, key=len, reverse=True) # sorting by length
  non_generated_keywords = list(dict.fromkeys(non_generated_keywords)) # removing duplicate keywords
```

Initial feature retrieval (PubMed only)

- The titles of all the articles without keywords were parsed, in search of the identified keywords, from the previously mentioned array

```python
def generate_keywords(row):
  if ( row["keywords"] == [] ):
    global count_empty
    global non_generated_keywords
    count_empty = count_empty + 1
    # print( row["title"] )
    keywords=[]
    generated_keywords = get_keywords( row["title"] )
    for i in range(0, len( non_generated_keywords ) ):
      if ( non_generated_keywords[i] in generated_keywords ):
        generated_keywords = generated_keywords.replace( non_generated_keywords[i], "" )
        keywords.append( non_generated_keywords[i] )
    keywords.extend( generated_keywords.split( ' ' ) )
    keywords=[x.lower() for x in keywords]
    valid_characters = set('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ\\- /_0123456789')
    # as such: "!"#$%&'()*+,.:;<=>?@[\\]^`{|}~" are now allowed of keyword names
    for keyword in keywords:
      keyword = ''.join(c for c in keyword if c in valid_characters)
```

Initial feature retrieval (PubMed only)

- All the data that could've been retrieved through the ISSN was saved in a JSON, within a dataframe column

```python
def get_issn_data ( row ):
  soup = get_page( "https://portal.issn.org/resource/ISSN/" + row["issn"] )
  container = soup.find(attrs={"id" : "tab0"})
  info_container = container.find(attrs={"class" : "item-result-content-text"})
  spans = info_container.find_all('span')
  journal_data={}
  for span in spans:
    attr = span.text
    parent = span.parent
    span.extract()

    attr = attr.replace(":", "").strip().lower()
    value = parent.text.strip().lower()
    if attr not in journal_data:
      journal_data[ attr ]=value
    else:
      if ( type( journal_data[ attr ] ).__name__ == 'list' ):
        journal_data[ attr ].append( value )
      else:
        journal_data[ attr ]=[ journal_data[ attr ], value ]
  return json.dumps(journal_data)
```

Initial feature retrieval (PubMed only)

- All dataframe columns were converted to string, so as to remove duplicated without errors …
- Duplicate removal shall be done when PMIDs are selected in the beginning, not at the end like it is done now …
- Finally, 2 CSV files were saved:
    - one with the full contents of the dataframe
    - another with data for citation network building

```
# Remove duplicated (not sure if necessary)

df = df.astype(str)
df.drop_duplicates(keep=False, inplace=True)
```

```
# Saving only PMID/citation data to a CSV file. This file is meant for cre
df[["pmid", "references"]].to_csv("/content/drive/MyDrive/citations.csv",

# Saving the whole dataframe data to a CSV too ...
df.to_csv("/content/drive/MyDrive/extracted_features.csv", encoding='utf-8
```

Final "Aim 1" tasks (done until 17/02)

DONE: Merge initial CSV data with the Pandas dataframe
DONE: Input all the PMIDs from all the CSVs, remove duplicates, then retrieve features
- Use the API for retrieving full article texts though the article PMC number (maybe API for ISSN and others too …)
- Use feature ranking tools/feature correlation to rank all determined features

Input PMIDs from all the CSVs

```
for file in glob.glob("/content/drive/MyDrive/DTA/*/*.csv"):
  filename = os.path.basename(file)
  temp_df = pd.read_csv(file, index_col=0)
  temp_df[ "Filename" ]=filename
  if ( "'int'" in str(type(csv_df)) ):
    csv_df = temp_df.copy()
  else:
    csv_df = pd.concat([csv_df, temp_df], axis=0, ignore_index=True)
  print( "All " + str(temp_df.shape[0]) + " rows of \"" + file + "\" were added to \"csv_df\" .. " )
  print( "\"csv_df\" has " + str(csv_df.shape[0]) + " rows ..." )
  csv_df = csv_df.drop_duplicates()
  print( "\"csv_df\" has " + str(csv_df.shape[0]) + " rows after duplicate removal ..." )
  print( "===========" )
```

Add retrieved features to initial CSVs

```
def get_data_by_pmid(row):
  pmid=row["PMID"]
  row_iter = df.index[df['pmid'] == str(pmid)].tolist()[0]
  for column in df.columns.values:
    if ( column in [ "pmid", "abstract", "title" ] ):
      continue
    row[ column ] = df.loc[ row_iter ][ column ]
  return row

csv_df = csv_df.apply(get_data_by_pmid, axis=1)
```

```
# Save the newly aquired information to flies that match the source data filenames,

directory="/content/drive/MyDrive/Feature_extraction_results"
if not os.path.exists(directory):
    os.makedirs(directory)

filenames = csv_df['Filename'].unique()
for filename in filenames:
  df_filename = csv_df.loc[csv_df['Filename'] == filename]
  df_filename = df_filename.drop('Filename', axis=1)
  df_filename.to_csv("/content/drive/MyDrive/Feature_extraction_results/" + filename
```

Aim 2: LLM and query-based corpus creation (17/02 - 10/03)

Article generation (17/02 - 24/02)

- Prompts shall be formulated based on article screening questions, in order to generate several "positive"/"negative" articles using both LLMs and a query-based method
- Classifiers shall be trained on each of the types of articles generated/retrieved (LLM/query-based)
- The classifier that proves itself superior in performance shall determine which article generation/retrieval method is superior (LLM/query-based)
- The chosen article generation/retrieval method shall be used at a larger scale to provide a viable corpus for accurate classifier training

Article generation (24/02 - 03/03)

- Input the generated/retrieved articles into the feature extraction code from "Aim 1", to generate a CSV containing all article features
- Train a model on the features considered the most relevant
- See if any of the features extracted can help formulate a summary of the articles generated/retrieved

Article generation (03/03 - 10/03)

- Output synthetic literature reviews (complex tasks that needs to connect many application parts together)
- If "Aim 2" can be considered achieved, move on with the next week's tasks and write text meant for the final coursework

Aim 3: Citation network creation (10/03 - 24/03)

Citation network (10/03 - 17/03)

- use an embedding space to represent all the citation data gathered so far
- use the embedding space to make a citation network
- train a classifier to discern between a "positive"/"negative" article relying solely on citation networks (complex task)

Citation network (17/03 - 24/03)

- use the output of the citation network classifier along with the output of the rest of the features to decide whether an article is "positive"/negative (complex task)
- if "Aim 3" can be considered achieved, move on with the next week's tasks and write text meant for the final coursework

Aim 4: Document the project within the final coursework (24/03 - 19/04)

Final coursework (17/03 - 24/03)

- finish up any tasks that were left unfinished so far
- write text/figures for the final coursework (finish it)
- make a short presentation and application demo (coursework 2)
- discuss further personal involvement with the systematic revie generation project with my supervisor

Thank you for your attention!