kwe-comparison

June 1, 2021

1 Keyword Extraction Comparison

This notebook contains a comparison of various Keyword Extraction (KWE) strategies applied to the scotch dataset. Each strategy is timed and its extracted keywords are retained.

A pdf of this notebook is saved - I have too frequently accidentally lost results of time intensive cells!

[1]:

```python
import pandas as pd
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from whiskynlp.Vectorizer import ListFeatureVectorizer
from whiskynlp.GraphKeywordExtraction import GraphKE
from whiskynlp.WhiskyLemmatizer import WhiskyLemmatizer
from nltk import pos_tag
import math
import time
from operator import itemgetter
```

[2]:

```python
t1 = time.time()
```

[3]:

```python
t2 = time.time()
```

[4]:

```python
t2 - t1
```

[4]:

```
0.019999742507934570
```

[5]:

```python
df = pd.read_csv("scotch-no-dupes.csv")
# Making corpus list to operate on
df["All"] = df["Nose"] + " " + df["Palate"] + " " + df["Finish"]
lst = GraphKE().makeCorpusList(df, "All")
corp = GraphKE().makeCorpus(lst)
whisky_stopwords = WhiskyLemmatizer().swords
```

[6]:

```python
results = {}
```

1.1 TF-IDF

https://www.analyticsvidhya.com/blog/2020/11/words-that-matter-a-simple-guide-to-keyword-extraction-in-python/

[7]:

```
# Based on implementation from analyticsvidhya, adapted
def wordTFScore(corpus, docs):
    tf = {}
    words = corpus.split()
    n_words = len(words)
    for word in words:
        word.replace('.','')
        if word not in whisky_stopwords:
            if word in tf:
                tf[word] += 1
            else:
                tf[word] = 1
    tf.update(
        (x, y/int(n_words)) for x, y in tf.items()
    )
    return tf

def countDocs(word, docs):
    final = [all([w in x for w in word]) for x in docs]
    return int(len([docs[i] for i in range(0, len(final)) if final[i]]))

def wordIDFScore(corpus, docs):
    idf = {}
    words = corpus.split()
    n_words = len(words)
    n_docs = len(docs)
    for word in words:
        word = word.replace('.','')
        if word not in whisky_stopwords:
            if word in idf:
                idf[word] = countDocs(word, docs)
            else:
                idf[word] = 1
    idf.update(
        (x, math.log(int(n_docs)/y)) for x, y in idf.items()
    )
    return idf

def tf_idf(corpus, docs):
    tf = wordTFScore(corpus, docs)
    idf = wordIDFScore(corpus, docs)
    tf_idf = [
        (word, tf[word]*idf[word]) for word in tf.keys()
    ]
    tf_idf.sort(key=itemgetter(1), reverse=True)
    return tf_idf
```

1.1.1 No Lemmatizing

[8]:

```
t1 = time.time()

tf_idf_wrds = tf_idf(corp, lst)

t2 = time.time()
tf_idf_time = t2 - t1
print(f"Time taken: {tf_idf_time} seconds")

results["tf_idf"] = {}
results["tf_idf"]["kws"] = tf_idf_wrds
results["tf_idf"]["time"] = tf_idf_time
```

```
Time taken: 1148.445481300354 seconds
```

1.1.2 WordNetLemmatizer

[9]:

```
wordnet = WordNetLemmatizer()
# Wordnet doesn't have an implementation of a cache, meaning it needs to query␣
 ↪wordnet
# each time.  Adding in a cache to save time.
wordnet_cache = {}
def replaceWithLemmas(txt):
    split = txt.split()
    lemma_txt = ''
    for word in split:
        if word not in whisky_stopwords:
            if word in wordnet_cache:
                lemma = wordnet_cache[word]
            else:
                # Getting POS tag
                tag = pos_tag([word])[0][1][0].lower()
                if tag == "v":
                    tag =  "v"
                if tag == "j":
                    tag =  "a"
                else:
                    tag = "n"
                lemma = wordnet.lemmatize(word, tag)
            lemma_txt = lemma_txt + ' ' + lemma
    return txt
```

[10]:
```
t1 = time.time()
wordnet_lst = [replaceWithLemmas(txt) for txt in lst]
wordnet_corp = GraphKE().makeCorpus(wordnet_lst)
t2 = time.time()
wordnet_lematizing = t2-t1
results["wordnet_lemmatizing_time"] = wordnet_lematizing
print(f"Time taken: {wordnet_lematizing} seconds")
```

```
Time taken: 125.97502851486206 seconds
```

[11]:
```
t1 = time.time()


wordnet_tf_idf = tf_idf(wordnet_corp, wordnet_lst)

t2 = time.time()
wordnet_tf_idf_time = t2 - t1
print(f"Time taken: {wordnet_tf_idf_time} seconds")

results["wordnet_tf_idf"] = {}
results["wordnet_tf_idf"]["kws"] = wordnet_tf_idf
results["wordnet_tf_idf"]["time"] = wordnet_tf_idf_time
```

```
Time taken: 1204.9405109882355 seconds
```

1.1.3 Custom Whisky Lemmatizer

[12]:
```
t1 = time.time()
whiskylemmatizer = WhiskyLemmatizer()
whisky_lst = [
    " ".join(whiskylemmatizer.tokenFilter(text)) for text in lst
]
whisky_corp = GraphKE().makeCorpus(whisky_lst)
t2 = time.time()
whisky_lemmatizing_time = t2 - t1
results["whisky_lemmatizing_time"] = whisky_lemmatizing_time
print(f"Time taken : {whisky_lemmatizing_time} seconds")
```

```
Time taken : 4.4477245807647705 seconds
```

[13]:
```
t1 = time.time()

whisky_tf_idf = tf_idf(whisky_corp, whisky_lst)

t2 = time.time()
whisky_tf_idf_time = t2 - t1
print(f"Time taken: {whisky_tf_idf_time} seconds")

results["whisky_tf_idf"] = {}
```

```
results["whisky_tf_idf"]["kws"] = whisky_tf_idf
results["whisky_tf_idf"]["time"] = whisky_tf_idf_time
```

```
Time taken: 1166.1536271572113 seconds
```

1.2 RAKE

[52]:

```
from rake_nltk import Rake, Metric
def rakeAsList(corpus):
    raker = Rake(corpus, min_length=1, max_length=1)
    raker.extract_keywords_from_text(corpus)
    return raker.get_ranked_phrases_with_scores()
```

1.2.1 Unlemmatized

[15]:

```
t1 = time.time()

unlemma_rake = rakeAsList(corp)

t2 = time.time()
unlemma_rake_time = t2 - t1
print(f"Time taken: {unlemma_rake_time} seconds")

results["unlemma_rake"] = {}
results["unlemma_rake"]["kws"] = unlemma_rake
results["unlemma_rake"]["time"] = unlemma_rake_time
```

```
Time taken: 0.44502973556518555 seconds
```

1.2.2 Wordnet Lemmatized

[16]:

```
t1 = time.time()

wordnet_rake = rakeAsList(wordnet_corp)

t2 = time.time()
wordnet_rake_time = t2 - t1
print(f"Time taken: {wordnet_rake_time} seconds")

results["wordnet_rake"] = {}
results["wordnet_rake"]["kws"] = wordnet_rake
results["wordnet_rake"]["time"] = wordnet_rake_time
```

```
Time taken: 0.46202850341796875 seconds
```

1.2.3 Whisky Lemmatized

[17]:

```
t1 = time.time()

whisky_rake = rakeAsList(whisky_corp)

t2 = time.time()
whisky_rake_time = t2 - t1
print(f"Time taken: {whisky_rake_time} seconds")

results["whisky_rake"] = {}
results["whisky_rake"]["kws"] = whisky_rake
results["whisky_rake"]["time"] = whisky_rake_time
```

```
Time taken: 0.2689990997314453 seconds
```

1.3 Eigencentrality RAKE

[18]:

```
t1 = time.time()

erake = GraphKE().keywordExtract(df, "All")

t2 = time.time()
erake_time = t2 - t1
print(f"Time taken: {erake_time} seconds")

results["erake"] = {}
results["erake"]["kws"] = erake
results["erake"]["time"] = erake_time
```

```
Building Corpus
Building Graph
Candidate Keywords Selected
Edges Created
Ranking Nodes
Time taken: 44.06533336639404 seconds
```

1.4 Saving Results to JSON

[19]:

```
import json
with open("results.json", "w") as f:
    json.dump(results, f)
```