In [4]:

```
from sentence_transformers import SentenceTransformer
from tqdm import tqdm
import collections,heapq,functools
import numpy as np
import RAKE
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import f1_score

# load a pretrained phrase bert model
model = SentenceTransformer('whaleloops/phrase-bert')

# util function: load all keywords/labels
def load_keywords(datafolder):
    # input: string of datafolder name
    filename = "data/"+datafolder+"/keywords.txt"
    with open(filename,encoding="UTF-8") as file:
        lines = file.readlines()
        lines = [line.rstrip() for line in lines]
    category_to_keywords = collections.defaultdict(set)
    allwords = set()
    for line in lines:
        cate, keywords = line.split(":")
        keywords = keywords.split(",")
        for kw in keywords:
            allwords.add(kw)
        category_to_keywords[cate] = set(keywords)
    # output:hashmap of str(cate id) -> keywords set, list of all keywords
    return category_to_keywords, list(allwords)

# util function: load documents in a lines fashion (later was covered in load_document_phrase_from_d
def load_documents(datafolder):
    filename = "data/"+datafolder+"/documents.txt"
    with open(filename,encoding="UTF-8") as file:
        lines = file.readlines()
        lines = [line.rstrip() for line in lines]
    return lines

# util function: get euclidean distance between 2 word vectors
def distance(vec1,vec2):
    return np.sum((vec1-vec2)**2)

# util function: construct a fully connected graph between keywords/labels, dimensionality reduction
def keyword_graph(keywords):
    adjHash = collections.defaultdict(set)
    distHash = collections.defaultdict(float)
    vecHash = {}
    wordvecs = model.encode(keywords)
    pca = PCA(n_components = 5)
    pca.fit(wordvecs)
    wordvecs = pca.transform(wordvecs)
    for i in range(len(keywords)):
        vecHash[keywords[i]] = wordvecs[i]
    for i in range(len(keywords)):
        for j in range(len(keywords)):
            if i != j:
                adjHash[keywords[i]].add(keywords[j])
                distHash[(keywords[i],keywords[j])] = distance(wordvecs[i], wordvecs[j])
    return vecHash,adjHash,distHash,pca
```

```
# util function: mining all phrases with their standardized quality (not used)
def load_phrase_with_quality(datafolder):
    filename = "data/"+datafolder+"/"+datafolder+"_train.txt"
    with open(filename,encoding="UTF-8") as file:
        lines = file.readlines()
        lines = [line.rstrip().replace("-"," ").replace("/"," ") for line in lines]
    stop_dir = "data/stopwords.txt"
    rake = RAKE.Rake(stop_dir)
    phrase_quality = {}
    corpus = ""
    for line in lines:
        corpus += line
    res = rake.run(corpus)
    qualities = []
    for kw,quality in res:
        qualities.append(quality)
    mms = MinMaxScaler()
    qualities = mms.fit_transform([[i] for i in qualities])
    for i in range(len(qualities)):
        phrase_quality[res[i][0]] = qualities[i][0]
    return phrase_quality


# util function: mining all phrases from {dataset}_train.txt
def load_document_phrase_from_direct_mine(datafolder):
    # TODOS load what phrase each document have?
    filename = "data/"+datafolder+"/"+datafolder+"_train.txt"
    with open(filename,encoding="UTF-8") as file:
        lines = file.readlines()
        lines = [line.rstrip().replace("-"," ").replace("/"," ") for line in lines]
    stop_dir = "data/stopwords.txt"
    rake = RAKE.Rake(stop_dir)
    document_id_to_phrases = []
    phrases = set()
    print("mining document phrases...")
    overall_phrases = []
    for i in tqdm(range(len(lines))):
        topwords = sorted(rake.run(lines[i]),key=lambda x:x[1],reverse=True)[:10]
        for tup in topwords:
            heapq.heappush(overall_phrases,(-tup[1],tup[0]))
        res = [tup[0] for tup in topwords]
        document_id_to_phrases.append(res)
        for p in res:
            phrases.add(p)
    return (document_id_to_phrases, list(phrases),overall_phrases)

# util function: load phrases from autophrase segmentation.txt file (not used due to quality issue)
def load_document_phrase_from_segmentation(datafolder):
    # TODOS load what phrase each document have?
    filename = "data/"+datafolder+"/segmentation.txt"
    with open(filename,encoding="UTF-8") as file:
        lines = file.readlines()
        lines = [line.rstrip() for line in lines]
    def find_phrases(sentence):
        stemmer = PorterStemmer()
        idx = 0
        res = []
        while idx != -1:
            idx = sentence.find("<phrase_Q=",idx)
            idx = sentence.find(">",idx)
```

```
            start = idx + 1
            idx = sentence.find("</phrase>",idx)
            end = idx
            if end != -1:
                res.append(sentence[start:end])
        return [" ".join([stemmer.stem(a) for a in word_tokenize(r)]) for r in res]
    document_id_to_phrases = []
    phrases = set()
    for i in range(len(lines)):
        res = find_phrases(lines[i])
        document_id_to_phrases.append(res)
        for p in res:
            phrases.add(p)
    return (document_id_to_phrases, list(phrases))

def phrase_dist_to_cates(phrase,vecHash,adjHash,distHash,keywords,category_to_keywords,pca):
    phrasevec = pca.transform(model.encode([phrase]))
    for kw in keywords:
        distHash[(phrase,kw)] = distance(phrasevec, vecHash[kw])
        adjHash[phrase].add(kw)

    # dijkstra
    visited = {} # keywords:cost
    heap = [(0,phrase)]
    while heap:
#         print(visited)
        cost,cur = heapq.heappop(heap)
        if cur in visited:
            continue
        visited[cur] = cost
        for nei in adjHash[cur]:
            if nei not in visited:
                heapq.heappush(heap,(cost+distHash[(cur,nei)],nei))

    cate_dist = collections.defaultdict(float)
    for cate,kws in category_to_keywords.items():
        for kw in kws:
            cate_dist[cate] += visited[kw]
        cate_dist[cate] = cate_dist[cate]/len(category_to_keywords[cate])
    return cate_dist

# core function: classify and return labels
def classify(datafolder):
    print("preprocessing...")
    document_id_to_phrases,phrases,_ = load_document_phrase_from_direct_mine(datafolder)
    category_to_keywords,keywords = load_keywords(datafolder)
    print("preparing keyword graph...")
    vecHash,adjHash,distHash,pca = keyword_graph(keywords)
    phrase_to_score = {}
    cate_all = collections.defaultdict(list)
    print("caching all documents phrases score...")
    for i in tqdm(range(len(phrases))):
        p = phrases[i]
        phrase_to_score[p] = phrase_dist_to_cates(p,vecHash,adjHash,distHash,keywords,category_to_ke
        for cate in category_to_keywords:
            cate_all[cate].append(phrase_to_score[p][cate])

    # standardize all distances
    print("standardize distances...")
    for cate in cate_all:
        scaler = MinMaxScaler()
```

```
        cate_all[cate] = scaler.fit_transform(np.array(cate_all[cate]).reshape(-1,1))
        for i in range(len(phrases)):
            phrase_to_score[phrases[i]][cate] = cate_all[cate][i]

    labels = []
    print("generating labels...")
    for i in tqdm(range(len(document_id_to_phrases))):
        cate_score_tmp = {}
        for cate in category_to_keywords:
            cate_score_tmp[cate] = 0
        for p in document_id_to_phrases[i]:
            if p in phrase_to_score:
                score = phrase_to_score[p]
                for cate in category_to_keywords:
                    cate_score_tmp[cate] += score[cate]
        smallest_score = min(cate_score_tmp.values())

        for cate,v in cate_score_tmp.items():
                if v == smallest_score:
                    labels.append(cate)
                    break
    print("labeling finished!")
    return labels

# util function: save all the output labels in out.txt
def save_outputs(datafolder):
    filename = "data/"+datafolder+"/out.txt"
    with open(filename, 'w') as f:
        for line in labels:
            f.write(line)
            f.write('\n')

# testing code
labels = classify("news")
save_outputs("news")
filename = "news_train_labels.txt"
with open(filename,encoding="UTF-8") as file:
    true_labels = file.readlines()
    true_labels = [line.rstrip() for line in true_labels]
f1_score(true_labels[:100], labels[:100], average='micro')
```

```
preprocessing...
mining document phrases...

100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████| 10
0/100 [00:00<00:00, 814.21it/s]

preparing keyword graph...
caching all documents phrases score...


100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████|
902/902 [00:18<00:00, 49.35it/s]

standardize distances...
generating labels...

100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████| 100/10
0 [00:00<00:00, 17008.53it/s]
```

```
labeling finished!
```

Out[4]:

0.78

In [7]:

```
# sample phrase to category, "recession" to 2: business
phrase = "recession"
category_to_keywords,keywords = load_keywords("news")
vecHash,adjHash,distHash,pca = keyword_graph(keywords)
cate_dist = phrase_dist_to_cates(phrase,vecHash,adjHash,distHash,keywords,category_to_keywords,pca)
cate_dist
```

Out[7]:

```
defaultdict(float,
            {'0': 75.86842861175538,
             '1': 125.56239900588989,
             '2': 39.30691909790039,
             '3': 115.36514520645142})
```

In [ ]: