임베딩 코드 정리

1. word2vec 모델 생성 코드 (word2vec.py)

순서 : 크롤링(네이버 뉴스, 네이버 영화 리뷰 등등) -> 데이터 전처리 -> word2vec.py

from gensim.models import Word2Vec  
from gensim.models.callbacks import CallbackAny2Vec  
from tqdm import tqdm  
corpus\_fname = './data/corpus\_mecab.txt'  
model\_fname = './data/word2vec'  
  
  
class callback(CallbackAny2Vec):  
 *"""Callback to print loss after each epoch."""* def \_\_init\_\_(self):  
 self.epoch = 0  
 self.loss\_to\_be\_subed = 0  
  
 def on\_epoch\_end(self, model):  
 loss = model.get\_latest\_training\_loss()  
 loss\_now = loss - self.loss\_to\_be\_subed  
 self.loss\_to\_be\_subed = loss  
 print('Loss after epoch {}: {}'.format(self.epoch, loss\_now))  
 self.epoch += 1  
  
print('corpus 생성')  
corpus = [sent.strip().split(" ") for sent in tqdm(open(corpus\_fname, 'r', encoding='utf-8').readlines())]  
  
print("학습 중")  
model = Word2Vec(  
 corpus,  
 vector\_size=100,  
 workers=4,  
 sg=1,  
 compute\_loss=True,  
 min\_count=5,  
 callbacks=[callback()])  
model.wv.save\_word2vec\_format(model\_fname)  
print('완료')

결과)

epoch마다 출력되는 숫자는 손실율을 의미함.

텍스트, 스크린샷, 검은색, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

2. 학습된 word2vec 모델으로 유사한 단어 출력하는 코드(word2vec\_check.py)

from gensim.models import KeyedVectors  
  
# 모델을 로딩하여 가장 유사한 단어를 출력  
  
loaded\_model = KeyedVectors.load\_word2vec\_format("./data/word2vec")  
# 모델 로드  
print(loaded\_model.vectors.shape)  
print(loaded\_model.most\_similar("최민식", topn=5))  
print(loaded\_model.most\_similar("남대문", topn=5))  
print(loaded\_model.similarity("헐크", '아이언맨'))  
print(loaded\_model.most\_similar(positive=['어벤져스', '아이언맨'], negative=['스파이더맨'], topn=1))

결과)

텍스트이(가) 표시된 사진

자동 생성된 설명

<최민식>단어와 가장 관련성이 높은 단어 5개를 코사인 유사도가 높은 순으로 출력

<남대문>단어와 가장 관련성이 높은 단어 5개를 코사인 유사도가 높은 순으로 출력

<헐크>, <아이언맨> 두 단어의 코사인 유사도를 출력

<어벤져스>, <아이언맨> 단어와는 관련성이 높고 <스파이더맨>과는 관련성이 낮은 단어를 1개 출력

3. fasttext 모델 생성 코드 (fasttext.py)

from gensim.models import FastText  
from tqdm import tqdm  
  
corpus\_fname = './data/corpus\_mecab.txt'  
model\_fname = './data/fasttext'  
  
print('corpus 생성')  
#말뭉치 생성  
  
corpus = [sent.strip().split(" ") for sent in tqdm(open(corpus\_fname, 'r', encoding='utf-8').readlines())]  
  
print("학습 중")  
  
  
  
model = FastText(corpus, vector\_size=100, workers=4, sg=1, min\_count=6, word\_ngrams=1)  
  
model.save(model\_fname)  
# https://projector.tensorflow.org/ 에서 시각화 하기 위해 모델을 따로 저장  
model.wv.save\_word2vec\_format(model\_fname + "\_vis")  
print('완료')

4. 학습된 fasttext 모델을 통해 유사한 단어 추출하는 코드 (fasttext\_check.py)

from gensim.models import FastText  
  
# 모델을 로딩하여 가장 유사한 단어를 출력  
  
loaded\_model = FastText.load("./data/fasttext")  
print(loaded\_model.wv.vectors.shape)  
print(loaded\_model.wv.most\_similar("최민식", topn=5))  
print(loaded\_model.wv.most\_similar("남대문", topn=5))  
print(loaded\_model.wv.similarity("헐크", '아이언맨'))  
print(loaded\_model.wv.most\_similar(positive=['어벤져스', '아이언맨'], negative=['스파이더맨'], topn=1))

5. LDA – tokenizer.py

순서 : 크롤링(네이버 뉴스, 위키백과 등등) -> 콘솔창에 parser.arg에 들어갈 입력 데이터 작성 -> 실행

# -\*- coding: utf-8 -\*-  
*"""  
Author: Sumin Lim (KAIST)  
Desciption: This file tokenize document with nouns and save it as text file and modified excel file with new column, nouns   
Usage: python tokenizer.py -tk Okt -if blockchain\_patent.xlsx -of test -c 요약  
-tk: tokenizer name, it should be one of Hannanum, Komoran, Kkma, Okt  
-if: input file name with its extension   
-of: output file name without its extension  
-c: column name to analyze in the input file (ex. 요약 column in blockchain\_patent.xlsx)  
"""*import argparse  
import pandas as pd  
from itertools import chain  
from collections import Counter  
from tqdm import tqdm   
  
import gensim  
import gensim.corpora as corpora  
  
from konlpy.tag import Hannanum, Komoran, Kkma, Okt, Mecab  
from sklearn.feature\_extraction.text import TfidfVectorizer  
  
class Tokenize:  
 def \_\_init\_\_(self, tokenizer, input\_filename, output\_filename, colname):  
 self.tokenizer = tokenizer  
 self.input\_filename = input\_filename  
 self.output\_filename = output\_filename  
 self.colname = colname  
  
 def main(self):  
 if self.tokenizer == "Twitter" or self.tokenizer == "Okt":  
 tokenizer = Okt()  
 elif self.tokenizer == "Hannanum":  
 tokenizer = Hannanum()  
 elif self.tokenizer == "Komoran":  
 tokenizer = Komoran()  
 elif self.tokenizer == "Kkma":  
 tokenizer = Kkma()  
 elif self.tokenizer == "mecab":  
 tokenizer = Mecab(dicpath=r"C:\mecab\mecab-ko-dic")  
  
 # Read file  
 df = pd.read\_excel(self.input\_filename)  
  
 # tokenizing  
 self.nouns = []  
 for row in tqdm(df[self.colname]):  
 self.nouns.append(tokenizer.nouns(row))  
  
 # Save tokenized text into text file  
 filename = self.output\_filename + "\_" + self.tokenizer  
 with open(filename+".txt", "w", encoding='utf-8') as f:  
 for sublist in self.nouns:  
 line = " ".join(sublist) + "\n"  
 f.write(line)  
  
 df["nouns"] = [" ".join(x) for x in self.nouns]  
 return df  
  
  
if \_\_name\_\_=="\_\_main\_\_":  
 parser = argparse.ArgumentParser()  
 parser.add\_argument("-tk", "--tokenizer", default="mecab", required=True, nargs=1, type=str)  
 parser.add\_argument("-if", "--inputFilename", default="lda\_sample.xlsx", required=True, nargs=1, type=str)  
 parser.add\_argument("-of", "--outputFilename", default="test", required=True, nargs=1, type=str)  
 parser.add\_argument("-c", "--columnName", default="contents", required=True, nargs=1, type=str)  
 args = parser.parse\_args()  
  
 test = Tokenize(args.tokenizer[0],   
 args.inputFilename[0],  
 args.outputFilename[0],  
 args.columnName[0])  
 test.main()

6. LDA – lda.py

순서 : 앞에서 토큰화한 데이터를 가지고 LDA 토픽 모델링 실행

*"""  
Author: Sumin Lim (KAIST)  
Description: This file implements LDA model and saves result file. Users should prepare tokenized   
text file as input file of this program.  
Usage: python lda.py -tkf tokenized\_filename  
"""*import argparse  
import pandas as pd   
import pickle as pkl  
from tqdm import tqdm  
from itertools import chain  
from collections import Counter  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.metrics.pairwise import linear\_kernel  
  
import gensim  
import gensim.corpora as corpora  
from gensim.models.coherencemodel import CoherenceModel  
from gensim.models.ldamulticore import LdaMulticore  
  
import matplotlib.pyplot as plt  
  
class LDA:  
 def \_\_init\_\_(self, tokenized\_file):  
 self.tokenized = tokenized\_file  
  
 def read\_data(self):  
 print("Read Data ... ")  
 tokenized\_words = []  
 split\_lines = []  
 with open(self.tokenized, "r", encoding='utf-8') as f:  
 for line in f:  
 tokenized\_words.append(line.strip())  
 split\_lines.append([x for x in line.strip().split()])  
  
 return tokenized\_words, split\_lines  
  
 def get\_tf(self):  
 print("Get term frequency ... ")  
 term\_count = Counter(chain.from\_iterable(self.document\_split))  
 df\_idf = pd.DataFrame(term\_count.items(), columns=["Term", "Freq"])  
 return df\_idf  
  
 def get\_tfidf\_score(self):  
 print("Get Tf-idf Score ... ")  
 vectorizer = TfidfVectorizer()  
 X = vectorizer.fit\_transform(self.document)  
 self.X = X  
  
 # Tf-idf Full Matrix (sparse)  
 df\_full = pd.DataFrame(X.toarray(), columns=vectorizer.get\_feature\_names())  
  
 # Tf-idf matrix with row format of document\_id, token\_id, tf-idf score  
 doc\_size = X.shape[0]; term\_size = X.shape[1]  
 terms = vectorizer.get\_feature\_names()  
 doc\_id = []; term\_id = []; score = []  
  
 for doc in tqdm(range(doc\_size)):  
 for term in range(term\_size):  
 if X[doc, term] != 0:  
 doc\_id.append(doc)  
 term\_id.append(terms[term])  
 score.append(X[doc, term])  
  
 df\_tfidf = pd.DataFrame({"Document": doc\_id,   
 "Term": term\_id,  
 "Score": score})  
  
 return df\_full, df\_tfidf  
  
 def lda(self, is\_graph):  
 print("Analyze LDA ... ")  
 alpha = input("Please enter the alpha: ")  
 iterations = int(input("Please enter the number of iterations: "))  
 is\_tfidf = input("Using Tf-idf (y/n): ")  
  
 id2word = corpora.Dictionary(self.document\_split)  
 corpus = [id2word.doc2bow(text) for text in self.document\_split]  
  
 if is\_tfidf == "y":  
 tfidf = gensim.models.TfidfModel(corpus)  
 corpus = tfidf[corpus]  
  
  
 if is\_graph == "y":  
 if alpha == "auto":  
 alpha = "asymmetric"  
 perplexities = []; coherences = []  
  
 start = int(input("Please enter the starting number of topic: "))  
 end = int(input("Please enter the end number of topic: "))  
 step = int(input("Please enter the step to increase: "))  
  
 for num\_topic in tqdm(range(start, end+1, step)):  
 lda\_model = LdaMulticore(corpus=corpus,  
 num\_topics=num\_topic,  
 id2word=id2word,  
 chunksize=100,  
 alpha=alpha,  
 iterations=iterations,  
 per\_word\_topics=True)  
  
 perplexities.append(lda\_model.log\_perplexity(corpus))  
  
 coherence = CoherenceModel(model=lda\_model,  
 texts=self.document\_split,  
 dictionary=id2word,  
 coherence="c\_v")  
 coherences.append(coherence.get\_coherence())  
  
 x\_topic = range(start, end+1, step)  
 plt.plot(x\_topic, perplexities)  
 plt.xlabel("The Number of Topics")  
 plt.ylabel("Log Perplexity")  
 plt.savefig("log\_perplexities\_"+"from\_"+str(start)+"\_to\_"+str(end+1)+".png", bbox\_inches="tight")  
 plt.show()  
 plt.clf()  
  
 plt.plot(x\_topic, coherences)  
 plt.xlabel("The Number of Topics")  
 plt.ylabel("Coherence Score")  
 plt.savefig("coherence\_score\_"+"from\_"+str(start)+"\_to\_"+str(end+1)+".png", bbox\_inches="tight")  
 plt.show()  
 plt.close()  
  
 elif is\_graph == "n":  
 num\_topics = input("Please enter the number of topics: ")  
 lda\_model = gensim.models.ldamodel.LdaModel(corpus=corpus,  
 id2word=id2word,  
 num\_topics=num\_topics,  
 random\_state=100,  
 update\_every=1,  
 chunksize=100,  
 alpha=alpha,  
 iterations=iterations,  
 per\_word\_topics=True)  
  
 # Perplexity and Coherence Score  
 print("Get log perplexity and coherence score ... ")  
 print("\nPerplexity: ", lda\_model.log\_perplexity(corpus))  
  
 coherence\_model\_lda = CoherenceModel(model=lda\_model,   
 texts=self.document\_split,   
 dictionary=id2word,  
 coherence="c\_v")  
 coherence\_lda = coherence\_model\_lda.get\_coherence()  
 print("\nCoherence Score: ", coherence\_lda)  
  
 # Save LDA, corpus, dictionary for visualizing  
 print("Save LDA model, corpus, dictionary for future visualizing ... ")  
 with open("corpus", "wb") as f:  
 pkl.dump(corpus, f)  
  
 with open("dictionary", "wb") as f:  
 pkl.dump(id2word, f)  
  
 with open("lda\_model", "wb") as f:  
 pkl.dump(lda\_model, f)  
  
 doc\_lda = lda\_model[corpus]  
  
 # Get topic words   
 print("Get topic words ... ")  
 columns = []  
 for k, v in id2word.iteritems():  
 columns.append(v)  
  
 df\_topic = []  
 topics = lda\_model.get\_topics()  
 for idx, topic in enumerate(topics):  
 temp = list([v, topic[k]] for k, v in zip(id2word.keys(), id2word.values()))  
 df\_temp = pd.DataFrame(temp, columns=["Topic"+str(idx+1)+"\_term",  
 "Topic"+str(idx+1)+"\_weight"])  
 df\_temp = df\_temp.sort\_values(by=["Topic"+str(idx+1)+"\_weight"], ascending=False)  
 df\_temp = df\_temp.reset\_index(drop=True)  
 df\_topic.append(df\_temp)  
  
 df\_topic = pd.concat(df\_topic, axis=1)  
  
 # Get Document-Topic Distribution  
 print("Get Document-Topic Distribution ... ")  
 doc\_topic = lda\_model.get\_document\_topics(doc\_lda)  
 dict\_doc\_topic = {}  
 for idx, doc in enumerate(doc\_topic):  
 dict\_doc\_topic[idx] = [x[1] for x in doc]  
  
 df\_doc\_topic = pd.DataFrame(dict\_doc\_topic).transpose()  
 df\_doc\_topic.rename(columns={col:"Topic-"+str(col+1) for col in df\_doc\_topic.columns}, inplace=True)  
 df\_doc\_topic.rename(index={x:"Document"+str(x+1) for x in df\_doc\_topic.index}, inplace=True)  
  
 print("Get topic weight ... ")  
 df\_topic\_weight = pd.DataFrame(df\_doc\_topic.sum(axis=0)).reset\_index()  
 df\_topic\_weight.rename(columns={"index":"Topic",  
 0: "Weight (Sum)"}, inplace=True)  
 weight\_sum = sum(df\_topic\_weight["Weight (Sum)"])  
 df\_topic\_weight["Weight (%)"] = df\_topic\_weight["Weight (Sum)"].apply(lambda x: x/weight\_sum \* 100)  
 df\_topic\_weight["Rank"] = df\_topic\_weight["Weight (%)"].rank(axis=0, ascending=False)  
  
 return df\_topic, df\_doc\_topic, df\_topic\_weight  
  
 def similarity(self):  
 print("Get cosine similarity ... ")  
 cos\_sim = linear\_kernel(self.X, self.X)  
 df\_sim = pd.DataFrame(cos\_sim)  
 df\_sim.rename(columns={col:"Document"+str(col) for col in df\_sim.columns}, inplace=True)  
 df\_sim.rename(index={col:"Document"+str(col) for col in df\_sim.index}, inplace=True)  
 return df\_sim  
  
 def main(self):  
 self.document, self.document\_split = self.read\_data()  
 self.df\_tf = self.get\_tf()  
 self.tfidf\_sparse, self.tfidf\_dense = self.get\_tfidf\_score()  
 self.df\_sim = self.similarity()  
  
 is\_graph = input("Please enter whether you want perplexity and coherence graph or not (y/n): ")  
 if is\_graph == "n":  
 self.df\_topic, self.df\_doc\_topic, self.df\_topic\_weight = self.lda(is\_graph)  
  
 print("Save result file ... ")  
 with pd.ExcelWriter("LDA\_result.xlsx") as writer:  
 self.df\_tf.to\_excel(writer, sheet\_name="TF", index=False, encoding="utf-8")  
 self.tfidf\_sparse.to\_excel(writer, sheet\_name="TFIDF\_sparse", index=False, encoding="utf-8")  
 self.tfidf\_dense.to\_excel(writer, sheet\_name="TFIDF\_dense", index=False, encoding="utf-8")  
 self.df\_topic.to\_excel(writer, sheet\_name="Topic-Keyword", index=False, encoding="utf-8")  
 self.df\_doc\_topic.to\_excel(writer, sheet\_name="Topic-Document", encoding="utf-8")  
 self.df\_topic\_weight.to\_excel(writer, sheet\_name="Topic-Weight", index=False, encoding="utf-8")  
 self.df\_sim.to\_excel(writer, sheet\_name="Document-Similarity", encoding="utf-8")  
 elif is\_graph == "y":  
 self.lda(is\_graph)  
  
 print("All work is done. Bye!")  
  
if \_\_name\_\_=="\_\_main\_\_":  
 parser = argparse.ArgumentParser()  
 parser.add\_argument("-tkf", "--tokenizedFileName", required=True, nargs=1, type=str)  
 args = parser.parse\_args()  
 lda = LDA(args.tokenizedFileName[0])  
  
 lda.main()

7. LDA 시각화 – LDA\_visualize.ipynb (아나콘다에서만 실행가능)

앞에서 실행된 모델을 이용한다.

모델이름은 corpus, dictionary, lda\_model

세 개의 모델을 꼭 같은 폴더에 저장하고 아타콘다를 실행하여 위 코드를 컴파일 한다.