|  |  |
| --- | --- |
| 교육제목 | 데이터 기반 인공지능 시스템 엔지니어 양성 과정 |
| 교육일시 | 2021-12-03 |
| 교육장소 | YGL 학과장 및 자택(디스코드 사용 온라인 학습) |
| **교육내용** | |
| 1. 네이버 영화 리뷰 감성분석(word2Vec)   !pip install konlpy  import pandas as pd  import urllib.request  import matplotlib.pyplot as plt  import re  from konlpy.tag import Okt  from tensorflow import keras  from tensorflow.keras.preprocessing.text import Tokenizer  import numpy as np  from tensorflow.keras.preprocessing.sequence import pad\_sequences  from collections import Counter   * 데이터 준비   urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ratings\_train.txt", filename="ratings\_train.txt")  urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ratings\_test.txt", filename="ratings\_test.txt")  train\_data = pd.read\_table('ratings\_train.txt')  test\_data = pd.read\_table('ratings\_test.txt')  train\_data. head()  from konlpy.tag import Mecab  !git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git  %cd Mecab-ko-for-Google-Colab/  !bash install\_mecab-ko\_on\_colab190912.sh  tokenizer= Mecab()  def tokenize\_and\_remove\_stopwords(data, stopwords, tokenizer):  result = []  for sentence in data:  curr\_data = []  curr\_data = tokenizer.morphs(sentence) # 형태소기반으로한 토큰화  curr\_data = [word for word in curr\_data if not word in stopwords] # 불용어 제거  result.append(curr\_data)  return result  # https://www.ranks.nl/stopwords/korean  stopwords = ['의', '가', '이', '은', '들', '는', '좀', '잘', '걍', '과', '도', '를', '으로', '자', '에', '와', '한', '하다']  def load\_data(train\_data, test\_data, num\_words= 10000):  # num\_words : 등장 빈도 순위로 몇 번째에 해당하는 단어까지 사용할 것인가?  # 10000을 입력하면, 등장 빈도 순위가 1~10000에 해당하는 단어만 사용. --> 단어집합의 크기 10,000  train\_data.drop\_duplicates(subset=['document'], inplace=True)  test\_data.drop\_duplicates(subset=['document'], inplace=True)  train\_data = train\_data.dropna(how='any')  test\_data = test\_data.dropna(how='any')  x\_train = tokenize\_and\_remove\_stopwords(train\_data['document'], stopwords, tokenizer)  x\_test = tokenize\_and\_remove\_stopwords(test\_data['document'], stopwords, tokenizer)  words = np.concatenate(x\_train).tolist()  counter = Counter(words)  counter = counter.most\_common(num\_words-4)  vocab = ['<PAD>', '<BOS>', '<UNK>', '<UNUSED>'] + [key for key, \_ in counter]  word\_to\_index = {word:index for index, word in enumerate(vocab)}  def wordlist\_to\_Indexlist(wordlist):  return [word\_to\_index[word] if word in word\_to\_index else word\_to\_index['<UNK>'] for word in wordlist]  x\_train = list(map(wordlist\_to\_Indexlist, x\_train))  x\_test = list(map(wordlist\_to\_Indexlist, x\_test))  return x\_train, np.array(list(train\_data['label'])), x\_test, np.array(list(test\_data['label'])), word\_to\_index  x\_train, y\_train, x\_test, y\_test, word\_to\_index = load\_data(train\_data, test\_data)  print(x\_train[0])  index\_to\_word = {index: word for word, index in word\_to\_index.items()}  def get\_encoded\_sentence(sentece, word\_to\_index): # 한 문장  return [word\_to\_index['<BOS>']]+ [word\_to\_index[word] if word in word\_to\_index else word\_to\_index['<UNK'] for word in sentence.split()]  def get\_encoded\_sentences(sentences, word\_to\_index): #여러 문장  return [get\_encoded\_sentence(sentence, word\_to\_index) for sentence in sentences]  def get\_decoded\_sentence(encoded\_sentence, index\_to\_word):  return ' '.join(index\_to\_word[index] if index in index\_to\_word else '<UNK>' for index in encoded\_sentence[1:])  def get\_decoded\_sentences(encoded\_sentences, index\_to\_word):  return [get\_decoded\_sentence(encoded\_sentence, index\_to\_word) for encoded\_sentence in encoded\_sentences]  get\_decoded\_sentence(x\_train[10], index\_to\_word)   * 모델 구성을 위한 데이터 분석 및 가공   total\_data\_text = list(x\_train) + list(x\_test)  num\_tokens = [len(tokens) for tokens in total\_data\_text]  num\_tokens = np.array(num\_tokens)  print('문장길이 평균: ', np.mean(num\_tokens))  print('문장길이 최대: ', np.max(num\_tokens))  print('문장길의 표준편차: ', np.std(num\_tokens))  # 최대길이 (평균 + 2 \* 표준편차)  max\_tokens = np.mean(num\_tokens) + 2 \* np.std(num\_tokens)  maxlen = int(max\_tokens)  print('pad sequences maxlen :', maxlen)  print('전체 문장의 {}%가 maxlen설정값 이내에 포함됩니다.'.format(np.sum(num\_tokens < max\_tokens)/len(num\_tokens)\*100))  x\_train = keras.preprocessing.sequence.pad\_sequences(x\_train, value = word\_to\_index['<PAD>'], padding='pre', maxlen = maxlen)  x\_test = keras.preprocessing.sequence.pad\_sequences(x\_test, value = word\_to\_index['<PAD>'], padding='pre', maxlen = maxlen)  print(x\_train.shape)  print(x\_test.shape)   * 모델 구성 및 validation 구성   vocab\_size = 10000  word\_vector\_dim = 256 # 워드 벡터의 차원 수  # 1. RNN버전  model\_rnn = keras.Sequential()  model\_rnn.add(keras.layers.Embedding(vocab\_size, word\_vector\_dim, input\_shape=(None,)))  model\_rnn.add(keras.layers.LSTM(16, activation='relu'))  model\_rnn.add(keras.layers.Dense(16, activation='relu'))  model\_rnn.add(keras.layers.Dense(1, activation='sigmoid'))  # 2. 1D-CNN  model\_cnn = keras.Sequential()  model\_cnn.add(keras.layers.Embedding(vocab\_size, word\_vector\_dim, input\_shape=(None,)))  model\_cnn.add(keras.layers.Conv1D(16, 3, activation='relu'))  model\_cnn.add(keras.layers.MaxPool1D(2))  model\_cnn.add(keras.layers.Conv1D(16, 3, activation='relu'))  model\_cnn.add(keras.layers.GlobalAveragePooling1D())  model\_cnn.add(keras.layers.Dense(8, activation='relu'))  model\_cnn.add(keras.layers.Dense(1, activation='sigmoid'))  #각 모델을 각각 다른 변수에 저장해주세요!  model\_rnn.summary()  model\_cnn.summary()  x\_val = x\_train[:50000]  y\_val = y\_train[:50000]  partial\_x\_train = x\_train[50000:]  partial\_y\_train = y\_train[50000:]  model\_rnn.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  epochs = 15  history\_rnn = model\_rnn.fit(partial\_x\_train, partial\_y\_train, epochs = epochs, batch\_size=512, validation\_data =(x\_val, y\_val), verbose=1)  # CNN1D학습  model\_cnn.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  history\_cnn = model\_cnn.fit(partial\_x\_train, partial\_y\_train, epochs = epochs, batch\_size=512, validation\_data =(x\_val, y\_val), verbose=1)  result\_rnn = model\_rnn.evaluate(x\_test, y\_test, verbose=2)  result\_cnn = model\_cnn.evaluate(x\_test, y\_test, verbose=2)  history\_rnn\_dic = history\_rnn.history  history\_cnn\_dic = history\_cnn.history  acc = history\_rnn\_dic['accuracy']  val\_acc = history\_rnn\_dic['val\_accuracy']  loss = history\_rnn\_dic['loss']  val\_loss = history\_rnn\_dic['val\_loss']  epochs = range(1, len(acc)+1)  plt.plot(epochs, loss, 'bo', label='Training loss')  plt.plot(epochs, val\_loss, 'b', label='Validation loss')  plt.title('Training and validation loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()  plt.clf()  plt.plot(epochs, acc, 'bo', label='Training acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  plt.title('Training and validation accuracy')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()  acc = history\_cnn\_dic['accuracy']  val\_acc = history\_cnn\_dic['val\_accuracy']  loss = history\_cnn\_dic['loss']  val\_loss = history\_cnn\_dic['val\_loss']  epochs = range(1, len(acc)+1)  plt.plot(epochs, loss, 'r\*', label='Training loss')  plt.plot(epochs, val\_loss, 'b^', label='Validation loss')  plt.title('CNN Training and validation loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()   1. 학습된 embedding 레이어 분석   import os  word2vec\_file\_path = 'word2vec.txt'  f = open(word2vec\_file\_path, 'w')  f.write('{} {} \n'.format(vocab\_size-4, word\_vector\_dim))  vectors = model\_rnn.get\_weights()[0]  for i in range(4, vocab\_size):  f.write('{} {}\n'.format(index\_to\_word[i], ' '.join(map(str, list(vectors[i, :])))))  f.close()  from gensim.models.keyedvectors import Word2VecKeyedVectors  word\_vector = Word2VecKeyedVectors.load\_word2vec\_format(word2vec\_file\_path, binary=False)  vector = word\_vector['짜증']  vector  word\_vector.similar\_by\_word("짜증")   1. 한국어 word2vec 임베딩을 활용해서 성능 개선   import gensim  word2vec\_path = '/content/drive/MyDrive/Colab Notebooks/영우4기\_자연어 (10일완성)/dataset/ko.bin'  word2vec = gensim.models.Word2Vec.load(word2vec\_path)  vector = word2vec['감동']  vector  word2vec.similar\_by\_word('재미')  mecab = Mecab()  def sentiment\_predict(new\_sentence):  import re  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences  t = Tokenizer()  new\_sentence = re.sub(r'[^ㄱ-ㅎㅏ-ㅣ가-힣]','',new\_sentence)  new\_sentence = mecab.morphs(new\_sentence)  new\_sentence = [word for word in new\_sentence if not word in stopwords]  encoded = t.texts\_to\_sequences([new\_sentence])  pad\_new = pad\_sequences(encoded, maxlen=max\_len)  score = float(model\_rnn.predict(pad\_new))  if (score > 0.5): # 긍정  print("{:.2f}% 확률로 긍정 리뷰 입니다. \n".format(score\*100))  else:  print("{:.2f}% 확률로 부정 리뷰 입니다. \n".format((1-score)\*100))  sentiment\_predict('이 영화 꿀잼 ㅋㅋㅋㅋ짱짱짱')   1. 네이버 쇼핑 리뷰 감성 분류하기  * 총 200,000개 리뷰로 구성 * 평점이 5점 만점에 1, 2, 4, 5인 리뷰들로 구성된 데이터 * 3점인 리뷰는 긍부정 유무가 애매해서 제외 * 평점이 4, 5인 리뷰에 긍정 ---> 1 * 평점이 1, 2인 리뷰에 부정 ---> 0   from konlpy.tag import Mecab  !git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git  %cd Mecab-ko-for-Google-Colab/  !bash install\_mecab-ko\_on\_colab190912.sh  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import urllib.request  from collections import Counter  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences  urllib.request.urlretrieve("https://raw.githubusercontent.com/bab2min/corpus/master/sentiment/naver\_shopping.txt", filename="ratings\_total.txt")  total\_data = pd.read\_table('ratings\_total.txt', names=['ratings','reviews'])  print('전체 리뷰 갯수 :', len(total\_data))  total\_data[:5]   * 훈련데이터와 테스트데이터를 분리   total\_data['label'] = np.select([total\_data.ratings >3], [1], default=0)  total\_data[:5]  total\_data['ratings'].nunique()  total\_data['reviews'].nunique() # 특이값/ 고유 값 갯수 확인  total\_data['label'].nunique()  total\_data.drop\_duplicates(subset=['reviews'], inplace=True) # 삭제  print('샘플의 수 :', len(total\_data)) # 삭제 후 갯수 확인  print(total\_data.isnull().values.any())  train\_data, test\_data = train\_test\_split(total\_data, test\_size=0.25, random\_state=42)  print('훈련용 리뷰의 갯수 :', len(train\_data))  print('테스트용 리뷰의 갯수 :', len(test\_data))   * 레이블의 분포 확인   train\_data['label'].value\_counts().plot(kind='bar')  print(train\_data.groupby('label').size().reset\_index(name='count'))   * 데이터 정제하기   train\_data['reviews'] = train\_data['reviews'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣]","")  train\_data['reviews'].replace('', np.nan, inplace=True)  print(train\_data.isnull().sum())  # test data  # 중복 제거  # 정규표현식을 이용하여 한글 외 문자 제거  # 공백을 null 변경  # Null값 제거  # test\_data 갯수 반환  test\_data.drop\_duplicates(subset=['reviews'], inplace=True)  test\_data['reviews'] = test\_data['reviews'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣]","")  test\_data['reviews'].replace('', np.nan, inplace=True)  test\_data = test\_data.dropna(how='any')  print('전처리 후 테스트용 샘플의 갯수 :', len(test\_data))   * 토큰화   mecab= Mecab()  print(mecab.morphs('이런 상품도 상품인가요? 허허허'))   * 불용어 제거   stopwords = ['의', '가', '이', '은', '들', '는', '좀', '잘', '걍', '과', '도', '를', '으로', '자', '에', '와', '한', '하다']  train\_data['tokenized'] = train\_data['reviews'].apply(mecab.morphs)  train\_data['tokenized'] = train\_data['tokenized'].apply(lambda x: [item for item in x if item not in stopwords])  test\_data['tokenized'] = test\_data['reviews'].apply(mecab.morphs)  test\_data['tokenized'] = test\_data['tokenized'].apply(lambda x: [item for item in x if item not in stopwords])   * 단어 길이와 분포 확인하기   negative\_words = np.hstack(train\_data[train\_data.label==0]['tokenized'].values)  positive\_words = np.hstack(train\_data[train\_data.label==1]['tokenized'].values)  negative\_word\_count = Counter(negative\_words)  print(negative\_word\_count.most\_common(20))  positive\_words\_count = Counter(positive\_words)  print(positive\_words\_count.most\_common(20))  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))  text\_len = train\_data[train\_data['label']==1]['tokenized'].map(lambda x: len(x))  ax1.hist(text\_len, color='red')  ax1.set\_title('Positive Reviews')  ax1.set\_xlabel('length of samples')  ax1.set\_ylabel('number of samples')  print('긍정 리뷰의 평균 길이 ;', np.mean(text\_len))  text\_len = train\_data[train\_data['label']==0]['tokenized'].map(lambda x: len(x))  ax2.hist(text\_len, color='blue')  ax2.set\_title('Negative Reviews')  ax2.set\_xlabel('length of samples')  ax2.set\_ylabel('number of samples')  print('부정 리뷰의 평균 길이 ;', np.mean(text\_len))  train\_data.head()  x\_train = train\_data['tokenized'].values  y\_train = train\_data['label'].values  x\_test = test\_data['tokenized'].values  y\_test = test\_data['label'].values   * 정수 인코딩   t = Tokenizer()  t.fit\_on\_texts(x\_train)  threshold = 2  total\_cnt = len(t.word\_index)  rare\_cnt = 0  total\_freq = 0  rare\_freq = 0  for key, value in t.word\_counts.items():  total\_freq = total\_freq + value  if (value < threshold):  rare\_cnt = rare\_cnt + 1  rare\_freq = rare\_freq + value  print('단어 집한 (vocabulary)의 크기 :', total\_cnt)  print('등장 빈도가 %s번 이하인 희귀단어의 수 : %s'%(threshold-1, rare\_cnt))  print('단어 집합에서 희귀단어의 비율 :', (rare\_cnt/total\_cnt)\*100)  print('전체 등장 빈도에서 희귀단어 등장 빈도 비율 :', (rare\_freq/total\_freq)\* 100)  vocab\_size = total\_cnt - rare\_cnt +2  print('단어 집합의 크기 :', vocab\_size)  original\_vocab\_size = vocab\_size + rare\_cnt -2  print('원래 vocab size :', original\_vocab\_size)  tokenizer = Tokenizer(vocab\_size, oov\_token='OOV')  tokenizer.fit\_on\_texts(x\_train)  x\_train = tokenizer.texts\_to\_sequences(x\_train)  x\_test = tokenizer.texts\_to\_sequences(x\_test)  print(x\_train[:3])  print(x\_test[:3])   * 패딩   print('리뷰의 최대 길이:', max(len(l) for l in x\_train))  print('리뷰의 평균 길이 :', sum(map(len, x\_train))/len(x\_train))  plt.hist([len(s) for s in x\_train], bins=50)  plt.xlabel('length of samples')  plt.xlabel('number of samples')  plt.show()  def below\_threshold\_len(max\_len, nested\_list):  cnt = 0  for s in nested\_list:  if (len(s) <= max\_len):  cnt = cnt +1  print('전체 샘플 중 길이가 %s 이하인 샘플의 비율 : %s'%(max\_len, (cnt/len(nested\_list))\*100))  max\_len = 80  below\_threshold\_len(max\_len, x\_train)  x\_train = pad\_sequences(x\_train, maxlen=max\_len)  x\_test = pad\_sequences(x\_test, maxlen=max\_len)  print(x\_train.shape)  print(x\_test.shape)  from tensorflow.keras.layers import Embedding, Dense, GRU  from tensorflow.keras.models import Sequential  from tensorflow.keras.models import load\_model  from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint  # 모델 만들기  embedding\_dim = 100  hidden\_size = 128  model\_gru = Sequential()  model\_gru.add(Embedding(vocab\_size, 100))  model\_gru.add(GRU(hidden\_size))  model\_gru.add(Dense(1, activation='sigmoid'))  es = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=4)  mc = ModelCheckpoint('best\_model.h5', monitor='val\_acc', mode='max', verbose=1, save\_best\_only=True)  model\_gru.compile(optimizer='adam', loss = 'binary\_crossentropy', metrics=['acc'])  history\_gru = model\_gru.fit(x\_train, y\_train, epochs=1, callbacks=[es, mc], batch\_size= 60, validation\_split=0.2)  model\_gru.evaluate(x\_test, y\_test)[1]   * 리뷰 예측하기   def sentiment\_predict(new\_sentence):  #new\_sentence = re.sub(r'[^ㄱ-ㅎㅏ-ㅣ가-힣]','',new\_sentence)  new\_sentence = mecab.morphs(new\_sentence)  new\_sentence = [word for word in new\_sentence if not word in stopwords]  encoded = tokenizer.texts\_to\_sequences([new\_sentence])  pad\_new = pad\_sequences(encoded, maxlen=max\_len)  score = float(model\_gru.predict(pad\_new))  if (score > 0.5): # 긍정  print("{:.2f}% 확률로 긍정 리뷰 입니다. \n".format(score\*100))  else:  print("{:.2f}% 확률로 부정 리뷰 입니다. \n".format((1-score)\*100))  sentiment\_predict('이 상품은 진짜 너무너무 좋아요!')  sentiment\_predict('이 상품은 진짜 너무너무 별로예요!') | |