



## 5장 단어와 타입 임베딩

---

윤 예 준

## INDEX

01 사전 훈련된 단어 임베딩

02 CBOW 임베딩 학습하기

03 문서 분류에 사전 훈련된  
임베딩을 사용한 전이 학습

04 요약

---

01

## 사전 훈련된 임베딩

## Word2Vec-GLoVe

### 임베딩

이산 타입과 벡터 공간의 포인트 사이에 매핑을 학습 하는 것

### 단어 임베딩

이산 타입이 단어일 때 밀집 벡터 표현

```
embeddings.get_embedding('dog').shape
```

(100,)

```
array([[ 0.30817,  0.30938,  0.52803, -0.92543, -0.73671,
         0.63475,  0.44197,  0.10262, -0.09142, -0.56607,
        -0.5327,  0.2013,  0.7704, -0.13983,  0.13727,
         1.1128,  0.89301, -0.17869, -0.0019722,  0.57289,
         0.59479,  0.50428, -0.28991, -1.3491,  0.42756,
         1.2748, -1.1613, -0.41084,  0.042804,  0.54866,
        ...])
```

```
class PreTrainedEmbeddings(object):
    """ 사전 훈련된 단어 벡터 사용을 위한 래퍼 클래스 """
    def __init__(self, word_to_index, word_vectors):
        """
        매개변수:
            word_to_index (dict): 단어에서 정수로 매핑
            word_vectors (numpy 배열의 리스트)
        """
        self.word_to_index = word_to_index
        self.word_vectors = word_vectors
        self.index_to_word = {v: k for k, v in self.word_to_index.items()}

        self.index = AnnoyIndex(len(word_vectors[0]), metric='euclidean')
        print("인덱스 만드는 중!")
        for i, v in self.word_to_index.items():
            self.index.add_item(i, self.word_vectors[i])
        self.index.build(50)
        print("완료!")

    @classmethod
    def from_embeddings_file(cls, embedding_file):
        """ 사전 훈련된 벡터 파일에서 객체를 만듭니다.

        벡터 파일은 다음과 같은 포맷입니다:
        word0 x0_0 x0_1 x0_2 x0_3 ... x0_M
        word1 x1_0 x1_1 x1_2 x1_3 ... x1_M

        매개변수:
            embedding_file (str): 파일 위치
        반환값:
            PreTrainedEmbeddings의 인스턴스
        """
        word_to_index = {}
        word_vectors = []

        with open(embedding_file) as fp:
            for line in fp.readlines():
                line = line.split(" ")
                word = line[0]
                vec = np.array([float(x) for x in line[1:]])

                word_to_index[word] = len(word_to_index)
                word_vectors.append(vec)

        return cls(word_to_index, word_vectors)
```

```
def get_embedding(self, word):
    """
    매개변수:
        word (str)
    반환값:
        임베딩 (numpy.ndarray)
    """
    return self.word_vectors[self.word_to_index[word]]

def get_closest_to_vector(self, vector, n=1):
    """ 벡터가 주어지면 n 개의 최근접 이웃을 반환합니다
    매개변수:
        vector (np.ndarray): Annoy 인덱스에 있는 벡터의 크기와 같아야 합니다
        n (int): 반환될 이웃의 개수
    반환값:
        [str, str, ...]: 주어진 벡터와 가장 가까운 단어
        단어는 거리순으로 정렬되어 있지 않습니다.
    """
    nn_indices = self.index.get_nns_by_vector(vector, n)
    return [self.index_to_word[neighbor] for neighbor in nn_indices]

def compute_and_print_analogy(self, word1, word2, word3):
    """ 단어 임베딩을 사용한 유사 결과를 출력합니다

    word1이 word2일 때 word3은 ... 입니다.
    이 예에서는 word1 : word2 :: word3 : word4를 출력합니다

    매개변수:
        word1 (str)
        word2 (str)
        word3 (str)
    """
    vec1 = self.get_embedding(word1)
    vec2 = self.get_embedding(word2)
    vec3 = self.get_embedding(word3)

    # 네 번째 단어 임베딩을 계산합니다
    spatial_relationship = vec2 - vec1
    vec4 = vec3 + spatial_relationship

    closest_words = self.get_closest_to_vector(vec4, n=4)
    existing_words = set([word1, word2, word3])
    closest_words = [word for word in closest_words
                     if word not in existing_words]

    if len(closest_words) == 0:
        print("계산된 벡터와 가장 가까운 이웃을 찾을 수 없습니다!")
        return

    for word4 in closest_words:
        print("{} : {} :: {} : {}".format(word1, word2, word3, word4))
```

## 언어 관계

### 단어 임베딩이 보여주는 언어 관계

```
embeddings.compute_and_print_analogy('man', 'he', 'woman')
```

```
man : he :: woman : she  
man : he :: woman : her
```

```
embeddings.compute_and_print_analogy('fly', 'plane', 'sail')
```

```
fly : plane :: sail : ship  
fly : plane :: sail : vessel  
fly : plane :: sail : boat
```

```
embeddings.compute_and_print_analogy('cat', 'kitten', 'dog')
```

```
cat : kitten :: dog : puppy  
cat : kitten :: dog : puppies  
cat : kitten :: dog : junkyard
```

```
embeddings.compute_and_print_analogy('blue', 'color', 'dog')
```

```
blue : color :: dog : animal  
blue : color :: dog : pet  
blue : color :: dog : taste  
blue : color :: dog : touch
```

### 단어 임베딩이 보여주는 편향

```
embeddings.compute_and_print_analogy('man', 'doctor', 'woman')
```

```
man : doctor :: woman : nurse
```

```
embeddings.compute_and_print_analogy('man', 'king', 'woman')
```

```
man : king :: woman : queen
```

02

## CBOW 임베딩 학습하기

## 프랑켄슈타인 데이터셋

### 전처리

1. 원시 데이터로 시작
2. NLTK의 Punk 토큰분할기 이용하여 개별 문장으로 분할
3. 각 문장을 소문자로 변환 및 구두점 제거 후 공백으로 문자열 분할하여 토큰 리스트 추출

```
1 args = Namespace(  
2     raw_dataset_txt="data/books/frankenstein.txt",  
3     window_size=5,  
4     train_proportion=0.7,  
5     val_proportion=0.15,  
6     test_proportion=0.15,  
7     output_munged_csv="data/books/frankenstein_with_splits.csv",  
8     seed=1337  
9 )  
  
1 # Split the raw text book into sentences  
2 tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')  
3 with open(args.raw_dataset_txt) as fp:  
4     book = fp.read()  
5 sentences = tokenizer.tokenize(book)
```

```
1 # Clean sentences  
2 def preprocess_text(text):  
3     text = ' '.join(word.lower() for word in text.split(" "))  
4     text = re.sub(r"([.,!?])", r" \1 ", text)  
5     text = re.sub(r"^[a-zA-Z.,!?]+", r" ", text)  
6     return text
```

## 프랑켄슈타인 데이터셋

전처리된 문장

원도 #1 i pitied frankenstein my pity amounted to horror i abhorred myself  
원도 #2 i pitied frankenstein my pity amounted to horror i abhorred myself  
원도 #3 i pitied frankenstein my pity amounted to horror i abhorred myself  
원도 #4 i pitied frankenstein my pity amounted to horror i abhorred myself

```
1 # Global vars
2 MASK_TOKEN = "<MASK>"
```

```
1 # Create windows
2 flatten = lambda outer_list: [item for inner_list in outer_list for item in inner_list]
3 windows = flatten([list(nltk.ngrams([MASK_TOKEN] * args.window_size + sentence.split(' ') + \
4 [MASK_TOKEN] * args.window_size, args.window_size * 2 + 1)) \
5 for sentence in tqdm_notebook(cleaned_sentences)])
6
7 # Create cbow data
8 data = []
9 for window in tqdm_notebook(windows):
10     target_token = window[args.window_size]
11     context = []
12     for i, token in enumerate(window):
13         if token == MASK_TOKEN or i == args.window_size:
14             continue
15         else:
16             context.append(token)
17     data.append([' '.join(token for token in context), target_token])
18
19 # Convert to dataframe
20 cbow_data = pd.DataFrame(data, columns=["context", "target"])
```

```
1 # Create split data
2 n = len(cbow_data)
3 def get_split(row_num):
4     if row_num <= n*args.train_proportion:
5         return 'train'
6     elif (row_num > n*args.train_proportion) and (row_num <= n*args.train_proportion + n*args.val_proportion):
7         return 'val'
8     else:
9         return 'test'
10 cbow_data['split'] = cbow_data.apply(lambda row: get_split(row.name), axis=1)
```

	context	target	split
0	, or the modern prometheus	frankenstein	train
1	frankenstein or the modern prometheus by	,	train
2	frankenstein , the modern prometheus by mary	or	train
3	frankenstein , or modern prometheus by mary wo...	the	train
4	frankenstein , or the prometheus by mary wolfs...	modern	train

CBOW\_Data



## Vocabulary

```

1 class Vocabulary(object):
2     """ 매핑을 위해 토큰들을 처리하고 여러 사전들을 만드는 클래스 """
3
4     def __init__(self, token_to_idx=None, mask_token="<MASK>", add_unk=True, unk_token="<UNK>"):
5         """
6         매개변수:
7             token_to_idx (dict): 기존 토큰-인덱스 매핑 딕셔너리
8             mask_token (str): Vocabulary에 추가할 MASK 토큰.
9                 모델 파라미터를 업데이트하는데 사용하지 않는 위치를 나타냅니다.
10             add_unk (bool): UNK 토큰을 추가할지 지정하는 플래그
11             unk_token (str): Vocabulary에 추가할 UNK 토큰
12
13         """
14         if token_to_idx is None:
15             token_to_idx = {}
16         self.token_to_idx = token_to_idx
17
18         self.idx_to_token = {idx: token
19                             for token, idx in self.token_to_idx.items()}
20
21         self.add_unk = add_unk
22         self.unk_token = unk_token
23         self.mask_token = mask_token
24
25         self.mask_index = self.add_token(self.mask_token)
26         self.unk_index = -1
27         if add_unk:
28             self.unk_index = self.add_token(unk_token)
29
30     def to_serializable(self):
31         """ 직렬화할 수 있는 딕셔너리를 반환합니다 """
32         return {'token_to_idx': self.token_to_idx,
33                 'add_unk': self.add_unk,
34                 'unk_token': self.unk_token,
35                 'mask_token': self.mask_token}
36

```

```

37 @classmethod
38 def from_serializable(cls, contents):
39     """ 직렬화된 딕셔너리에서 Vocabulary 객체를 만듭니다 """
40     return cls(**contents)
41
42 def add_token(self, token):
43     """ 토큰을 기반으로 매핑 딕셔너리를 업데이트합니다
44
45     매개변수:
46         token (str): Vocabulary에 추가할 토큰
47     반환값:
48         index (int): 토큰에 상응하는 점수
49     """
50     if token in self.token_to_idx:
51         index = self.token_to_idx[token]
52     else:
53         index = len(self.token_to_idx)
54         self.token_to_idx[token] = index
55         self.idx_to_token[index] = token
56     return index
57
58 def add_many(self, tokens):
59     """ 토큰 리스트를 Vocabulary에 추가합니다.
60
61     매개변수:
62         tokens (list): 문자열 토큰 리스트
63     반환값:
64         indices (list): 토큰 리스트에 상응되는 인덱스 리스트
65     """
66     return [self.add_token(token) for token in tokens]
67

```

```

68 def lookup_token(self, token):
69     """ 토큰에 대응하는 인덱스를 추출합니다.
70         토큰이 없으면 UNK 인덱스를 반환합니다.
71
72     매개변수:
73         token (str): 찾을 토큰
74     반환값:
75         index (int): 토큰에 해당하는 인덱스
76     노트:
77         UNK 토큰을 사용하려면 (Vocabulary에 추가하기 위해)
78         'unk_index'가 0보다 커야 합니다.
79     """
80     if self.unk_index >= 0:
81         return self.token_to_idx.get(token, self.unk_index)
82     else:
83         return self.token_to_idx[token]
84
85 def lookup_index(self, index):
86     """ 인덱스에 해당하는 토큰을 반환합니다.
87
88     매개변수:
89         index (int): 찾을 인덱스
90     반환값:
91         token (str): 인덱스에 해당하는 토큰
92     예외:
93         KeyError: 인덱스가 Vocabulary에 없을 때 발생합니다.
94     """
95     if index not in self.idx_to_token:
96         raise KeyError("the index (%d) is not in the Vocabulary" % index)
97     return self.idx_to_token[index]
98
99 def __str__(self):
100     return "<Vocabulary(size=%d)>" % len(self)
101
102 def __len__(self):
103     return len(self.token_to_idx)
104

```

## Vectorizer

```

1 class CBOWVectorizer(object):
2     """ 어휘 사전을 생성하고 관리합니다 """
3     def __init__(self, cbow_vocab):
4         """
5         매개변수:
6             cbow_vocab (Vocabulary): 단어를 정수에 매핑합니다
7         """
8         self.cbow_vocab = cbow_vocab
9
10    def vectorize(self, context, vector_length=-1):
11        """
12        매개변수:
13            context (str): 공백으로 나누어진 단어 문자열
14            vector_length (int): 인덱스 벡터의 길이 매개변수
15        """
16
17        indices = [self.cbow_vocab.lookup_token(token) for token in context.split(' ')]
18        if vector_length < 0:
19            vector_length = len(indices)
20
21        out_vector = np.zeros(vector_length, dtype=np.int64)
22        out_vector[:len(indices)] = indices
23        out_vector[len(indices):] = self.cbow_vocab.mask_index
24
25        return out_vector
26

```

```

27 @classmethod
28 def from_dataframe(cls, cbow_df):
29     """데이터셋 데이터프레임에서 Vectorizer 객체를 만듭니다
30
31     매개변수::
32         cbow_df (pandas.DataFrame): 타겟 데이터셋
33     반환값:
34         CBOWVectorizer 객체
35     """
36     cbow_vocab = Vocabulary()
37     for index, row in cbow_df.iterrows():
38         for token in row.context.split(' '):
39             cbow_vocab.add_token(token)
40         cbow_vocab.add_token(row.target)
41
42     return cls(cbow_vocab)
43
44 @classmethod
45 def from_serializable(cls, contents):
46     cbow_vocab = \
47         Vocabulary.from_serializable(contents['cbow_vocab'])
48     return cls(cbow_vocab=cbow_vocab)
49
50 def to_serializable(self):
51     return {'cbow_vocab': self.cbow_vocab.to_serializable()}

```

## DataLoader

```

1 class CBOWDataset(Dataset):
2     def __init__(self, cbow_df, vectorizer):
3         """
4         매개변수:
5             cbow_df (pandas.DataFrame): 데이터셋
6             vectorizer (CBOWVectorizer): 데이터셋에서 만든 CBOWVectorizer 객체
7         """
8         self.cbow_df = cbow_df
9         self.vectorizer = vectorizer
10
11     measure_len = lambda context: len(context.split(" "))
12     self._max_seq_length = max(map(measure_len, cbow_df.context))
13
14     self.train_df = self.cbow_df[self.cbow_df.split=="train"]
15     self.train_size = len(self.train_df)
16
17     self.val_df = self.cbow_df[self.cbow_df.split=="val"]
18     self.validation_size = len(self.val_df)
19
20     self.test_df = self.cbow_df[self.cbow_df.split=="test"]
21     self.test_size = len(self.test_df)
22
23     self.lookup_dict = {'train': (self.train_df, self.train_size),
24                          'val': (self.val_df, self.validation_size),
25                          'test': (self.test_df, self.test_size)}
26
27     self.set_split('train')
28 
```

```

29 @classmethod
30 def load_dataset_and_make_vectorizer(cls, cbow_csv):
31     """데이터셋을 로드하고 처음부터 새로운 Vectorizer 만들기
32
33     매개변수:
34         cbow_csv (str): 데이터셋의 위치
35     반환값:
36         CBOWDataset의 인스턴스
37     """
38     cbow_df = pd.read_csv(cbow_csv)
39     train_cbow_df = cbow_df[cbow_df.split=="train"]
40     return cls(cbow_df, CBOWVectorizer.from_dataframe(train_cbow_df))
41
42 @classmethod
43 def load_dataset_and_load_vectorizer(cls, cbow_csv, vectorizer_filepath):
44     """데이터셋을 로드하고 새로운 CBOWVectorizer 객체를 만듭니다.
45     캐시된 CBOWVectorizer 객체를 재사용할 때 사용합니다.
46
47     매개변수:
48         cbow_csv (str): 데이터셋의 위치
49         vectorizer_filepath (str): CBOWVectorizer 객체의 저장 위치
50     반환값:
51         CBOWVectorizer의 인스턴스
52     """
53     cbow_df = pd.read_csv(cbow_csv)
54     vectorizer = cls.load_vectorizer_only(vectorizer_filepath)
55     return cls(cbow_df, vectorizer)
56
57 @staticmethod
58 def load_vectorizer_only(vectorizer_filepath):
59     """파일에서 CBOWVectorizer 객체를 로드하는 정적 메서드
60
61     매개변수:
62         vectorizer_filepath (str): 직렬화된 CBOWVectorizer 객체의 위치
63     반환값:
64         CBOWVectorizer의 인스턴스
65     """
66     with open(vectorizer_filepath) as fp:
67         return CBOWVectorizer.from_serializable(json.load(fp))
68
69 def save_vectorizer(self, vectorizer_filepath):
70     """CBOWVectorizer 객체를 json 형태로 디스크에 저장합니다
71
72     매개변수:
73         vectorizer_filepath (str): CBOWVectorizer 객체의 저장 위치
74
75     with open(vectorizer_filepath, "w") as fp:
76         json.dump(self._vectorizer.to_serializable(), fp)
77
78 def get_vectorizer(self):
79     """백터 변환 객체를 반환합니다 """
80     return self._vectorizer
81
82 def set_split(self, split="train"):
83     """데이터프레임에 있는 열을 사용해 분할 세트를 선택합니다 """
84     self._target_split = split
85     self._target_df, self._target_size = self.lookup_dict[split]
86
87 def __len__(self):
88     return self._target_size

```

```

90 def __getitem__(self, index):
91     """파이토치 데이터셋의 주요 진입 메서드
92
93     매개변수:
94         index (int): 데이터 포인트의 인덱스
95     반환값:
96         데이터 포인트의 특성(x_data)과 레이블(y_target)로 이루어진 딕셔너리
97     """
98     row = self._target_df.iloc[index]
99
100     context_vector = \
101         self._vectorizer.vectorize(row.context, self._max_seq_length)
102     target_index = self._vectorizer.cbow_vocab.lookup_token(row.target)
103
104     return {'x_data': context_vector,
105            'y_target': target_index}
106
107 def get_num_batches(self, batch_size):
108     """배치 크기가 주어지면 데이터셋으로 만들 수 있는 배치 개수를 반환합니다
109
110     매개변수:
111         batch_size (int)
112     반환값:
113         배치 개수
114     """
115     return len(self) // batch_size
116
117 def generate_batches(dataset, batch_size, shuffle=True,
118                     drop_last=True, device="cpu"):
119     """
120     파이토치 DataLoader를 감싸고 있는 제너레이터 함수.
121     각 텐서를 지정된 장치로 이동합니다.
122     """
123     dataloader = DataLoader(dataset=dataset, batch_size=batch_size,
124                             shuffle=shuffle, drop_last=drop_last)
125
126     for data_dict in dataloader:
127         out_data_dict = {}
128         for name, tensor in data_dict.items():
129             out_data_dict[name] = data_dict[name].to(device)
130         yield out_data_dict

```

## CBOWClassifier 모델

```
class CBOWClassifier(nn.Module): # Simplified cbow Model
    def __init__(self, vocabulary_size, embedding_size, padding_idx=0):
        """
        매개변수:
            vocabulary_size (int): 어휘 사전 크기, 임베딩 개수와 예측 벡터 크기를 결정합니다
            embedding_size (int): 임베딩 크기
            padding_idx (int): 기본값 0: 임베딩은 이 인덱스를 사용하지 않습니다
        """
        super(CBOWClassifier, self).__init__()

        self.embedding = nn.Embedding(num_embeddings=vocabulary_size,
                                       embedding_dim=embedding_size,
                                       padding_idx=padding_idx)

        self.fc1 = nn.Linear(in_features=embedding_size,
                              out_features=vocabulary_size)

    def forward(self, x_in, apply_softmax=False):
        """분류기의 정방향 계산
        매개변수:
            x_in (torch.Tensor): 입력 데이터 텐서
            x_in.shape는 (batch, input_dim)입니다.
            apply_softmax (bool): 소프트맥스 활성화 함수를 위한 플래그
            크로스-엔트로피 손실을 사용하려면 False로 지정합니다
        반환값:
            결과 텐서. tensor.shape는 (batch, output_dim)입니다.
        """
        x_embedded_sum = F.dropout(self.embedding(x_in).sum(dim=1), 0.3)
        y_out = self.fc1(x_embedded_sum)

        if apply_softmax:
            y_out = F.softmax(y_out, dim=1)

        return y_out
```

## 모델 훈련 및 모델 평가

```
1 def make_train_state(args):
2     return {'stop_early': False,
3             'early_stopping_step': 0,
4             'early_stopping_best_val': 1e8,
5             'learning_rate': args.learning_rate,
6             'epoch_index': 0,
7             'train_loss': [],
8             'train_acc': [],
9             'val_loss': [],
10            'val_acc': [],
11            'test_loss': -1,
12            'test_acc': -1,
13            'model_filename': args.model_state_file}
```

CBOW 훈련 매개변수

테스트 손실: 7.6765486717224105;  
테스트 정확도: 13.191176470588225

03

## 문서 분류에 사전 훈련된 임베딩을 사용한 전이 학습

## AG뉴스 데이터셋

2005년 수집한 뉴스 기사 모음

스포츠, 과학/기술, 세계, 비즈니스 범주로  
균등하게 분할된 뉴스 기사 12만개 사용

```
1 class NewsDataset(Dataset):
2     @classmethod
3     def load_dataset_and_make_vectorizer(cls, news_csv):
4         """데이터셋을 로드하고 처음부터 새로운 Vectorizer 만들기
5
6         매개변수:
7             news_csv (str): 데이터셋의 위치
8         반환값:
9             NewsDataset의 인스턴스
10
11         """
12         news_df = pd.read_csv(news_csv)
13         train_news_df = news_df[news_df.split=='train']
14         return cls(news_df, NewsVectorizer.from_dataframe(train_news_df))
15
16     @classmethod
17     def load_dataset_and_load_vectorizer(cls, news_csv, vectorizer_filepath):
18         """ 데이터셋과 새로운 Vectorizer 객체를 로드합니다.
19         캐시된 Vectorizer 객체를 재사용할 때 사용합니다.
20
21         매개변수:
22             news_csv (str): 데이터셋의 위치
23             vectorizer_filepath (str): Vectorizer 객체의 저장 위치
24         반환값:
25             NewsDataset의 인스턴스
26
27         """
28         news_df = pd.read_csv(news_csv)
29         vectorizer = cls.load_vectorizer_only(vectorizer_filepath)
30         return cls(news_df, vectorizer)
31
32     def __getitem__(self, index):
33         """파이토치 데이터셋의 주요 진입 메서드
34
35         매개변수:
36             index (int): 데이터 포인트의 인덱스
37         반환값:
38             데이터 포인트의 특성(x_data)과 레이블(y_target)로 이루어진 딕셔너리
39
40         """
41         row = self._target_df.iloc[index]
42
43         title_vector = \
44             self._vectorizer.vectorize(row.title, self._max_seq_length)
45
46         category_index = \
47             self._vectorizer.category_vocab.lookup_token(row.category)
```

```
args = Namespace(
    raw_dataset_csv="data/ag_news/news.csv",
    train_proportion=0.7,
    val_proportion=0.15,
    test_proportion=0.15,
    output_munged_csv="data/ag_news/news_with_splits.csv",
    seed=1337
)
```

```
# Read raw data
news = pd.read_csv(args.raw_dataset_csv, header=0)
```

```
news.head()
```

	category	title
0	Business	Wall St. Bears Claw Back Into the Black (Reuters)
1	Business	Carlyle Looks Toward Commercial Aerospace (Reu...
2	Business	Oil and Economy Cloud Stocks' Outlook (Reuters)
3	Business	Iraq Halts Oil Exports from Main Southern Pipe...
4	Business	Oil prices soar to all-time record, posing new...

## Vocabulary

UNK : 모델이 드물게 등장하는 단어에 대한  
표현을 학습하도록 함

MASK : Embedding 층의 마스킹 역할 수행 및  
가변 길이의 시퀀스가 있을 때 손실 계산을 도움

BEGIN-OF-SEQUENCE, END-OF-SEQUENCE :  
시퀀스 경계에 관한 힌트를 신경망에 제공

```
1 class SequenceVocabulary(Vocabulary):
2     def __init__(self, token_to_idx=None, unk_token="<UNK>",
3                 mask_token="<MASK>", begin_seq_token="<BEGIN>",
4                 end_seq_token="<END>"):
5
6         super(SequenceVocabulary, self).__init__(token_to_idx)
7
8         self._mask_token = mask_token
9         self._unk_token = unk_token
10        self._begin_seq_token = begin_seq_token
11        self._end_seq_token = end_seq_token
12
13        self.mask_index = self.add_token(self._mask_token)
14        self.unk_index = self.add_token(self._unk_token)
15        self.begin_seq_index = self.add_token(self._begin_seq_token)
16        self.end_seq_index = self.add_token(self._end_seq_token)
17
18    def to_serializable(self):
19        contents = super(SequenceVocabulary, self).to_serializable()
20        contents.update({'unk_token': self._unk_token,
21                        'mask_token': self._mask_token,
22                        'begin_seq_token': self._begin_seq_token,
23                        'end_seq_token': self._end_seq_token})
24        return contents
25
26    def lookup_token(self, token):
27        """ 토큰에 대응하는 인덱스를 추출합니다.
28        토큰이 없으면 UNK 인덱스를 반환합니다.
29
30        매개변수:
31            token (str): 찾을 토큰
32        반환값:
33            index (int): 토큰에 해당하는 인덱스
34        노트:
35            UNK 토큰을 사용하려면 (Vocabulary에 추가하기 위해)
36            `unk_index`가 0보다 커야 합니다.
37
38        """
39        if self.unk_index >= 0:
40            return self._token_to_idx.get(token, self.unk_index)
41        else:
42            return self._token_to_idx[token]
```



## Vectorizer

```
SequenceVocabulary
{
  '<MASK>': 0,
  '<UNK>': 1,
  '<BEGIN-OF-SEQUENCE>': 2,
  '<END-OF-SEQUENCE>': 3,
  'is': 4,
  'happy': 5
}
```

"Jerry is happy"

0단계: 시퀀스 입력

1 4 5

1단계: 단어를 정수로 매핑합니다

2 1 4 5 3

2단계: 문장을 경계 토큰으로 감쌉니다

2 1 4 5 3 0 0

3단계: 모든 벡터의 길이가 같도록 0으로 패딩합니다

```
1 class NewsVectorizer(object):
2     """여러 사전을 생성하고 관리합니다"""
3     def __init__(self, title_vocab, category_vocab):
4         self.title_vocab = title_vocab
5         self.category_vocab = category_vocab
6
7     def vectorize(self, title, vector_length=-1):
8         """
9         매개변수:
10             title (str): 공백으로 나누어진 단어 문자열
11             vector_length (int): 인덱스 벡터의 길이 매개변수
12         반환값:
13             벡터로 변환된 제목 (numpy.array)
14         """
15         indices = [self.title_vocab.begin_seq_index]
16         indices.extend(self.title_vocab.lookup_token(token)
17                       for token in title.split(" "))
18         indices.append(self.title_vocab.end_seq_index)
19
20         if vector_length < 0:
21             vector_length = len(indices)
22
23         out_vector = np.zeros(vector_length, dtype=np.int64)
24         out_vector[:len(indices)] = indices
25         out_vector[len(indices):] = self.title_vocab.mask_index
26
27         return out_vector
28
```

```
29 @classmethod
30 def from_dataframe(cls, news_df, cutoff=25):
31     """데이터셋 데이터프레임에서 Vectorizer 객체를 만듭니다
32
33     매개변수:
34         news_df (pandas.DataFrame): 타깃 데이터셋
35         cutoff (int): Vocabulary에 포함할 빈도 임계값
36     반환값:
37         NewsVectorizer 객체
38     """
39     category_vocab = Vocabulary()
40     for category in sorted(set(news_df.category)):
41         category_vocab.add_token(category)
42
43     word_counts = Counter()
44     for title in news_df.title:
45         for token in title.split(" "):
46             if token not in string.punctuation:
47                 word_counts[token] += 1
48
49     title_vocab = SequenceVocabulary()
50     for word, word_count in word_counts.items():
51         if word_count >= cutoff:
52             title_vocab.add_token(word)
53
54     return cls(title_vocab, category_vocab)
55
56 @classmethod
57 def from_serializable(cls, contents):
58     title_vocab = \
59         SequenceVocabulary.from_serializable(contents['title_vocab'])
60     category_vocab = \
61         Vocabulary.from_serializable(contents['category_vocab'])
62
63     return cls(title_vocab=title_vocab, category_vocab=category_vocab)
64
65 def to_serializable(self):
66     return {'title_vocab': self.title_vocab.to_serializable(),
67           'category_vocab': self.category_vocab.to_serializable()}
```

## NewsClassifier 모델

```

11 def load_glove_from_file(glove_filepath):
12     """glove 임베딩 로드
13
14     매개변수:
15         glove_filepath (str): 임베딩 파일 경로
16     반환값:
17         word_to_index (dict), embeddings (numpy.ndarray)
18     """
19
20     word_to_index = {}
21     embeddings = []
22     with open(glove_filepath, "r") as fp:
23         for index, line in enumerate(fp):
24             line = line.split(" ") # each line: word num1 num2 ...
25             word_to_index[line[0]] = index # word = line[0]
26             embedding_i = np.array([float(val) for val in line[1:]])
27             embeddings.append(embedding_i)
28     return word_to_index, np.stack(embeddings)
29
30 def make_embedding_matrix(glove_filepath, words):
31     """
32     특정 단어 집합에 대한 임베딩 행렬을 만듭니다.
33
34     매개변수:
35         glove_filepath (str): 임베딩 파일 경로
36         words (list): 단어 리스트
37     """
38     word_to_idx, glove_embeddings = load_glove_from_file(glove_filepath)
39     embedding_size = glove_embeddings.shape[1]
40
41     final_embeddings = np.zeros((len(words), embedding_size))
42
43     for i, word in enumerate(words):
44         if word in word_to_idx:
45             final_embeddings[i, :] = glove_embeddings[word_to_idx[word]]
46         else:
47             embedding_i = torch.ones(1, embedding_size)
48             torch.nn.init.xavier_uniform_(embedding_i)
49             final_embeddings[i, :] = embedding_i
50
51     return final_embeddings

```

```

1 class NewsClassifier(nn.Module):
2     def __init__(self, embedding_size, num_embeddings, num_channels,
3                 hidden_dim, num_classes, dropout_p,
4                 pretrained_embeddings=None, padding_idx=0):
5
6         매개변수:
7             embedding_size (int): 임베딩 벡터의 크기
8             num_embeddings (int): 임베딩 벡터의 개수
9             num_channels (int): 합성곱 커널 개수
10            hidden_dim (int): 은닉 차원 크기
11            num_classes (int): 클래스 개수
12            dropout_p (float): 드롭아웃 확률
13            pretrained_embeddings (numpy.array): 사전에 훈련된 단어 임베딩
14            기문값은 None
15            padding_idx (int): 패딩 인덱스
16
17        super(NewsClassifier, self).__init__()
18
19        if pretrained_embeddings is None:
20
21            self.emb = nn.Embedding(embedding_dim=embedding_size,
22                                    num_embeddings=num_embeddings,
23                                    padding_idx=padding_idx)
24
25        else:
26            pretrained_embeddings = torch.from_numpy(pretrained_embeddings).float()
27            self.emb = nn.Embedding(embedding_dim=embedding_size,
28                                    num_embeddings=num_embeddings,
29                                    padding_idx=padding_idx,
30                                    _weight=pretrained_embeddings)
31
32        self.convnet = nn.Sequential(
33            nn.Conv1d(in_channels=embedding_size,
34                     out_channels=num_channels, kernel_size=3),
35            nn.ELU(),
36            nn.Conv1d(in_channels=num_channels, out_channels=num_channels,
37                     kernel_size=3, stride=2),
38            nn.ELU(),
39            nn.Conv1d(in_channels=num_channels, out_channels=num_channels,
40                     kernel_size=3, stride=2),
41            nn.ELU(),
42            nn.Conv1d(in_channels=num_channels, out_channels=num_channels,
43                     kernel_size=3),
44            nn.ELU()
45        )
46
47        self._dropout_p = dropout_p
48        self.fc1 = nn.Linear(num_channels, hidden_dim)
49        self.fc2 = nn.Linear(hidden_dim, num_classes)
50

```

```

51 def forward(self, x_in, apply_softmax=False):
52     """분류기의 정방향 계산
53
54     매개변수:
55         x_in (torch.Tensor): 입력 데이터 텐서
56         x_in.shape는 (batch, dataset.max_seq_length)입니다.
57         apply_softmax (bool): 소프트맥스 활성화 함수를 위한 플래그
58         크로스-엔트로피 손실을 사용하려면 False로 지정합니다
59
60     반환값:
61         결과 텐서. tensor.shape는 (batch, num_classes)입니다.
62     """
63
64     # 임베딩을 적용하고 특정 채널 차원을 바꿉니다
65     x_embedded = self.emb(x_in).permute(0, 2, 1)
66
67     features = self.convnet(x_embedded)
68
69     # 평균 풀링을 계산하여 부가적인 차원을 제거합니다
70     remaining_size = features.size(dim=2)
71     features = F.avg_pool1d(features, remaining_size).squeeze(dim=2)
72     features = F.dropout(features, p=self._dropout_p)
73
74     # MLP 분류기
75     intermediate_vector = F.relu(F.dropout(self.fc1(features), p=self._dropout_p))
76     prediction_vector = self.fc2(intermediate_vector)
77
78     if apply_softmax:
79         prediction_vector = F.softmax(prediction_vector, dim=1)
80
81     return prediction_vector

```

## 모델 훈련 및 평가

```
args = Namespace(  
    # 날짜와 경로 정보  
    news_csv="data/ag_news/news_with_splits.csv",  
    vectorizer_file="vectorizer.json",  
    model_state_file="model.pth",  
    save_dir="model_storage/ch5/document_classification",  
    # 모델 하이퍼파라미터  
    glove_filepath='data/glove/glove.6B.100d.txt',  
    use_glove=False,  
    embedding_size=100,  
    hidden_dim=100,  
    num_channels=100,  
    # 훈련 하이퍼파라미터  
    seed=1337,  
    learning_rate=0.001,  
    dropout_p=0.1,  
    batch_size=128,  
    num_epochs=100,  
    early_stopping_criteria=5,  
    # 실행 옵션  
    cuda=True,  
    catch_keyboard_interrupt=True,  
    reload_from_files=False,  
    expand_filepaths_to_save_dir=True  
)
```

테스트 손실: 0.6176266238093375;  
테스트 정확도: 79.6819196428571

## 새로운 뉴스 제목의 카테고리 예측

```
1 def predict_category(title, classifier, vectorizer, max_length):
2     """뉴스 제목을 기반으로 카테고리를 예측합니다
3
4     매개변수:
5         title (str): 원시 제목 문자열
6         classifier (NewsClassifier): 훈련된 분류기 객체
7         vectorizer (NewsVectorizer): 해당 Vectorizer
8         max_length (int): 최대 시퀀스 길이
9         노트: CNN은 입력 텐서 크기에 민감합니다.
10             훈련 데이터처럼 동일한 크기를 갖도록 만듭니다.
11
12     """
13     title = preprocess_text(title)
14     vectorized_title = \
15         torch.tensor(vectorizer.vectorize(title, vector_length=max_length))
16     result = classifier(vectorized_title.unsqueeze(0), apply_softmax=True)
17     probability_values, indices = result.max(dim=1)
18     predicted_category = vectorizer.category_vocab.lookup_index(indices.item())
19
20     return {'category': predicted_category,
21            'probability': probability_values.item()}
```

04

요약

1. 사전 훈련된 임베딩을 블랙박스처럼 사용하는 방법
2. CBOW를 포함해 데이터에서 이런 임베딩을 직접 훈련하는 방법(간략히)
3. 언어 모델링 측면에서 CBOW 모델을 훈련하는 방법
4. 문서 분류에 사전 훈련된 임베딩을 사용한 전이 학습

THE

END

감 사 합 니 다

---