실습 2

Logistic Regression with Doc2vec

목표(sklearn의 linear_model사용) logistic regression

- MBTI CSV데이터를 사용하여 75%:25% 의 train, test set을 만들고
- 테스트셋의 classfication accuracy 80% 이상 성능 목표
- doc2vec 사용하여 size=100, min_count=2, iter=55 으로 학습
- 중간중간 단어들의 벡터화가 어떻게 진행되는지 확인하여 이해하자.
- 제공 : 기본전처리코드, csv파일 제공
- 결과:

```
In [23]: sum(preds == test_y) / len(test_y)
Out[23]: 0.8142000922083911
```

입력

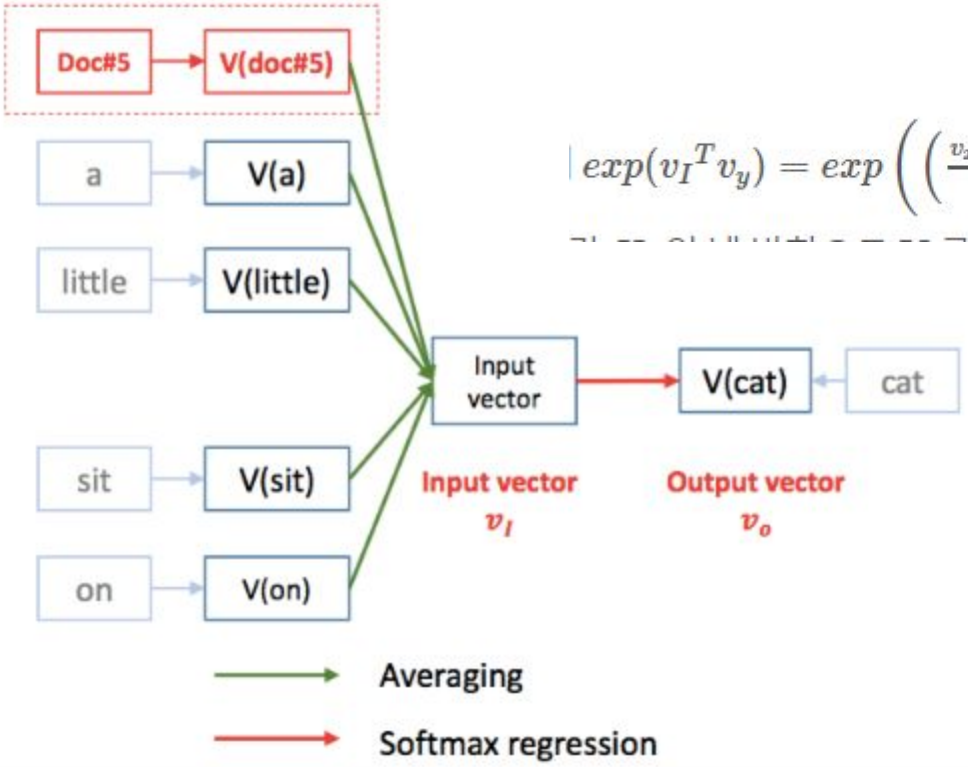
doc2vec은 word2vec의 확장이기 때문에 사용 패턴이 유사하다. 표현의 크기, 슬라이딩 윈도우의 크기, 작업자 수 또는 word2vec 모델로 변경할 수 있는 거의 모든 다른 매개변수를 쉽게 조정할 수 있다.

이 규칙의 한가지 예외는 모델에서 사용한 교육방법과 관련된 매개변수이다. word2vec 아키텍쳐에서는 두개의 중요 알고리즘이 있는데 “continuous bag of words”(CBOW)와 “skip-gram”(sg) 입니다. **doc2vec** 아키텍처에서는 “distributed memory”(dm)과 “distributed bag of words”(dbow)이다. 분산메모리 모델이 논문에서 훨씬 좋은 수행을 보였기에 이를 기본으로 사용한다. 원한다면 dm=0 플래그를 설정하여 dbow모델을 강제 실행할 수 있다.

doc2vec의 입력은 LabeledSentence 객체의 iterator이다. 각 객체는 하나의 문장을 나타내며 단어 목록과 레이블 목록으로 구성된다.

```
sentence = LabeledSentence(words=[u'some', u'words', u'here'], labels=[u'SENT_1'])
```

그런다음 알고리즘은 sentence iterator를 두번 수행한다.: 한번은 vocab를 빌드하고, 한번은 입력 데이터에서 모델을 학습하고, 각 단어 및 데이터 세트의 각 레이블에 대한 벡터 표현을 학습한다. 이 아키텍처에서는 문장당 둘 이상의 레이블을 사용할 수 있지만 가장 많이 사용되는 경우는 문장의 고유 식별자인 문장당 하나의 레이블을 갖는 것이다. 트레이닝 데이터로 다음의 클래스를 사용하여 한 줄에 한 문장의 파일에 대해 이러한 종류의 사용 사례를 구현할 수 있다.



$$exp(v_I{}^T v_y) = exp\left(\left(\frac{v_{x1}+\cdots+v_{x4}}{4}\right)^T v_y\right)$$

기본 전처리 코드 제공

```
def read_corpus(fname, tokens_only=False):
    with smart_open.smart_open(fname, encoding="iso-8859-1") as f:
        for i, line in enumerate(f):
            if tokens_only:
                yield gensim.utils.simple_preprocess(line)
            else:
                # For training data, add tags
                if i==0:
                    pass
                else:
                    yield gensim.models.doc2vec.TaggedDocument(gensim.utils.simple_preprocess(line)[1:],
                                                               gensim.utils.simple_preprocess(line)[0])
```

word2vec , doc2vec 참고 사이트

https://lovit.github.io/nlp/representation/2018/03/26/word_doc_embedding/

doc2vec의 함수들을 알아 볼 수 있다.

https://radimrehurek.com/gensim/models/doc2vec.html

모델 generate 예시코드. 이후 train과정도 필요하다.

```
gensim.models.doc2vec.Doc2Vec
```

두가지의 과정 build vocab과 train과정이 수행되어져야 한다.

```
model.build_vocab(train_corpus)
```

```
model.train(train_corpus, total_examples=model.corpus_count, epochs=model.iter)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

sklearn에 있는 linear모델 패키지를 이용하여 LogisticRegression을 generate시킨다.

- 임의의 단어들을 넣어서 어떠한 백터를 갖는지 확인 해볼수 있다.

- 랜덤으로 숫자들이 generate 된다는 사실도 알 수 있다.

```
model.infer_vector(['I', 'feel', 'sad'])
```

```
array([-0.00278599, -0.01557731,  0.01890842, -0.00029549,  0.05383271,
        0.01940338, -0.03221119,  0.07604899,  0.00588031,  0.02700568,
        0.02806578, -0.0245112 , -0.06045406, -0.03911825,  0.04883436,
       -0.00359876, -0.056864  ,  0.06742074,  0.00261859, -0.04436981,
        0.00619907, -0.04728865, -0.02438512,  0.02999322, -0.0115072 ,
        0.06164995, -0.08784495,  0.02438634,  0.05194131,  0.01070534,
       -0.02468438, -0.01552269,  0.00382666,  0.02027941,  0.04277623,
        0.04443385,  0.01638766, -0.03593103,  0.00191042,  0.01339457,
        0.05903789,  0.03614008,  0.04582254, -0.03801802, -0.06010791,
ll output; double click to hide
        0.06831823, -0.04069351,  0.007586  ,  0.03664215, -0.02017786,
       -0.03989157,  0.0602704 , -0.02118911,  0.01739586, -0.01377531,
        0.00274627,  0.02612195, -0.01770055, -0.02832794,  0.03917153,
        0.01341582, -0.04190777, -0.01466462,  0.11753285,  0.06130242,
        0.00543751,  0.08337265, -0.01711578,  0.00432177,  0.00232403,
        0.01341729,  0.01654916,  0.01585295,  0.04662458, -0.03394553,
       -0.04544606,  0.07339423,  0.01653387,  0.0040966 , -0.05253183,
        0.05121448,  0.01543063, -0.05268038, -0.00979281, -0.02280125,
        0.01509547, -0.01311491, -0.00952421, -0.01051938, -0.08485222,
       -0.0272574 ,  0.00565611,  0.00629376,  0.00313089,  0.06422194],
      dtype=float32)
```

- 최종 예측 predict 값이다

```
sum(preds == test_y) / len(test_y)
```

```
0.8123559243891194
```

- 79~81사이로 백터가 랜덤으로 generate되면서

  조금씩의 오차는 생길수 있다.

- 추가옵션) 임의의 test 파일을 만들어 predict해보자