NLP-不同詞向量在文本分類上的成效表現

台灣人工智慧小聚(2020-02-07)

Tom Lin

目標

我們的目標為何？

手上的原始文本資料…

1	author	tweet
2	Neil deGrasse Tyson	A 50-yard field goal in MetLife stadium will deflect nearly 1/2 inch due to Earth's rotation — meet the Coriolis force.
3	Cristiano Ronaldo	RT @Thiaguinhooo14: Manda um abraço em português para seus fás no Brasil ! @Cristiano #Celebrate15M
4	Ellen DeGeneres	Today I'm talking about a topic that affects all of us. Man-spreading. https://t.co/fyBUEmHj5k
5	Sebastian Ruder	New blog post giving an overview of softmax approximations for learning better word embeddings https://t.co/I7lkb5ESu5 #deeplearning #NLProc

特徵值…

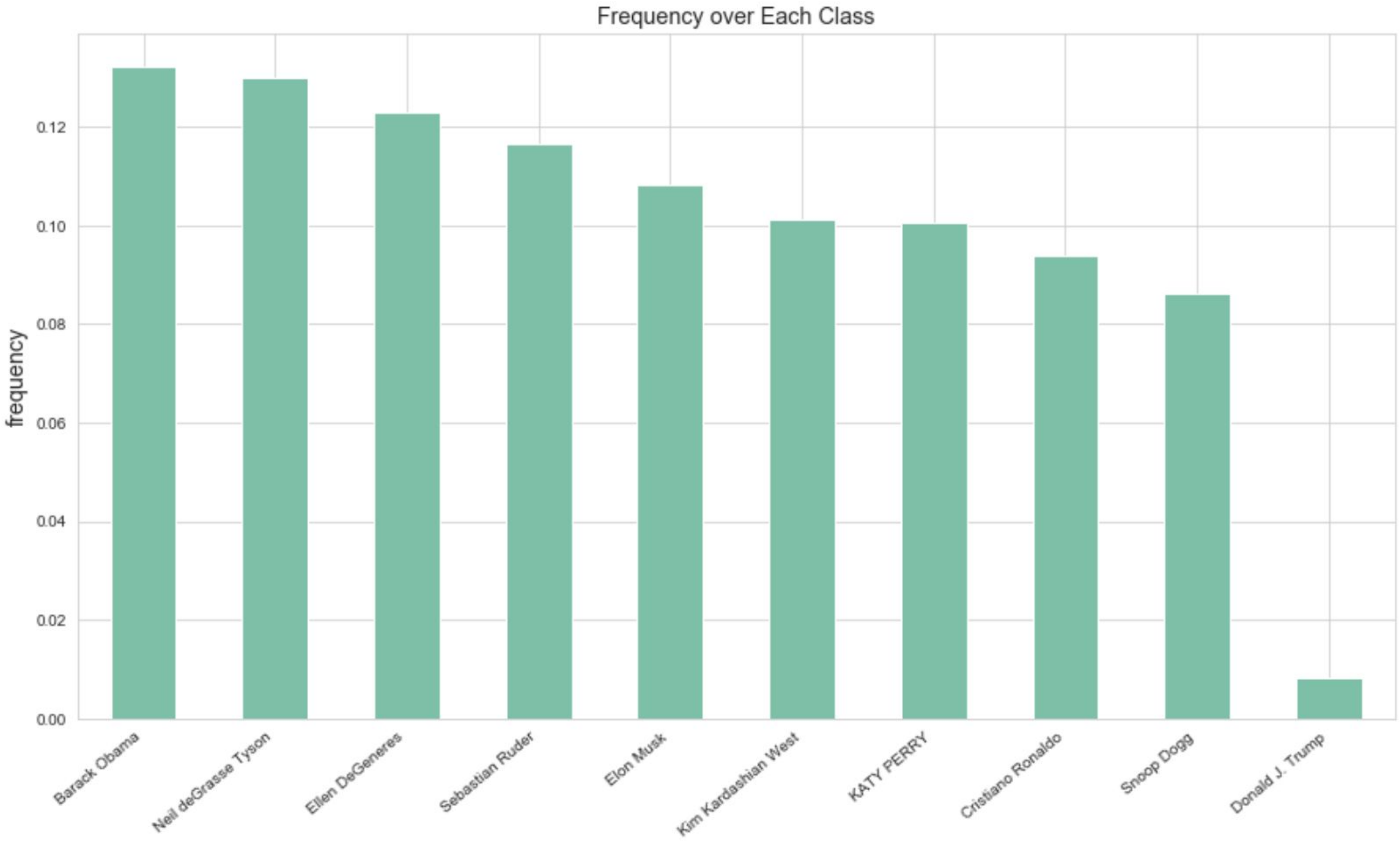
f1	f2	f3	f4	f5
0.1	0.5	0.3	0.6	0.7
0.2	0.4	0.3	0.1	0.2

將每一個文本(document)轉換成N維度的特徵值，再利用這些features進行文本分類器的訓練。

目標

Label的分佈

資料集中，各個名人的tweets數量分佈大致相同



- 每個名人tweets所佔的比率，從最高的13%到8%
- 唯一例外的是Donald Trump的tweets，所佔比率不及1%

方法

方法一

WHY word2vec?

1) 相較於one-hot-encode，使用word2vec可以進行降維，有助於分類模型學習

2) word embedding，可以將每個字以數值向量的方式表示，有助於比較每個字之間的相似度

方法一

利用word2vec進行轉換

1) 針對每一個文本，先將文本中的每一個字，轉換成N維的詞向量

2) 再將每個詞向量加總再平均，視作為那一個文本的文本向量

3) 我們暫且將這個方式稱作是 mean word embedding

示意圖…

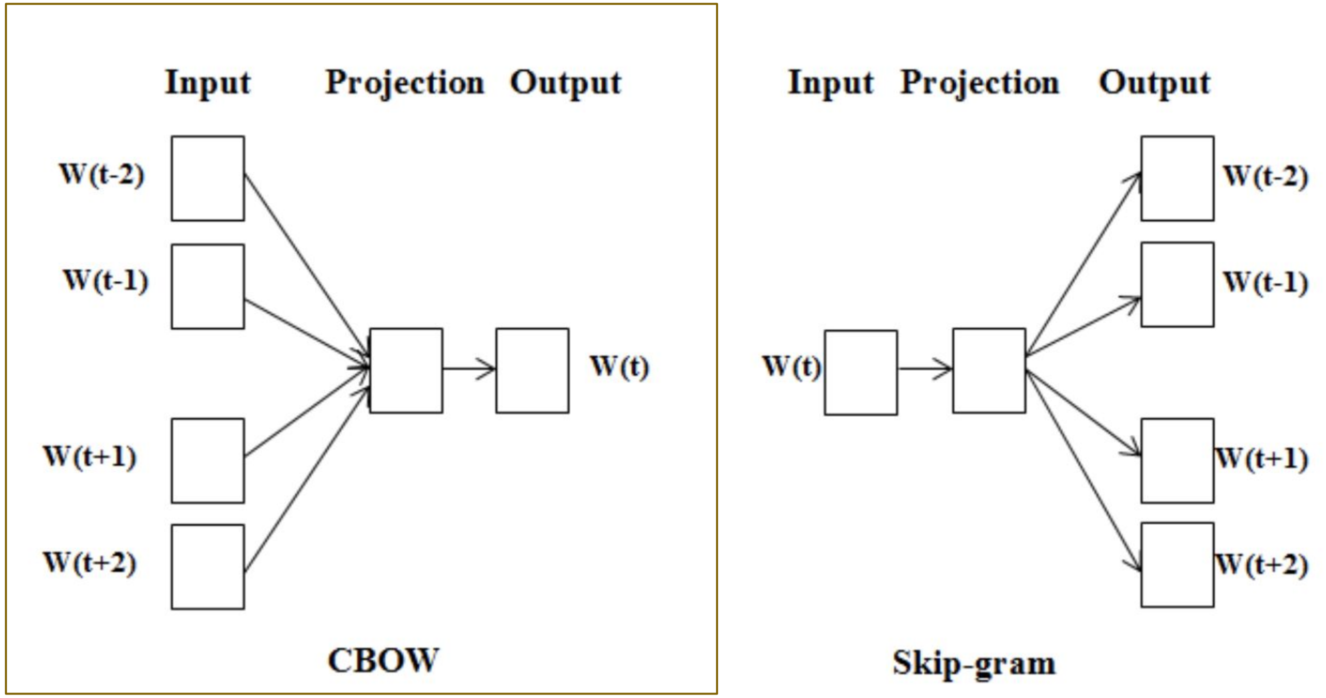
	f1	f2	f3	f4	f5
This	0.1	0.5	0.3	0.6	0.7
is	0.2	0.4	0.3	0.1	0.2
a	…				
book	…				
	…				
	0.1	0.4	0.3	0.3	0.4

訓練word2vec的兩種方法

1) CBOW (continuous bag of words)
   - cbow 在學習每個字的詞性和其在句構上的位置, 表現較好(syntatic)

2) SKIP-GRAM
   - skip-gram 在學習每個字的詞義, 和其在句子中的句義是否合理, 表現較好(semantic)

gensim套件預設的是CBOW模型



CBOW and Skip-gram models architecture [1]

REF:
https://medium.com/daai/%E5%BC%95%E8%B5%B7%E4%BD%A0%E5%B0%8D-word2vec-%E5%9F%BA%E6%9C%AC%E6%A6%82%E5%BF%B5-524c8b758f99

方法二

利用TF-IDF進行加權

1) 針對每一個文本，先將文本中的每一個字，轉換成N維的詞向量

2) 再將每個詞向量進行TF-IDF加權，後再加總平均，視作為那一個文本的文本向量

3) 我們暫且將這個方式稱作是 TF-IDF weighted mean embedding

示意圖…

	f1	f2	f3	f4	f5	
This ←	0.1	0.5	0.3	0.6	0.7	X TF-IDF
is ←	0.2	0.4	0.3	0.1	0.2	X TF-IDF
a			…			
book			…			
			…			
→	0.1	0.4	0.3	0.3	0.4	

善用TF-IDF

- 利用TF-IDF作為權重，將每個字進行加權。

- TF-IDF會給予只在侷部的文本中出現的高詞頻字彙，最高的權數，因此可以更強化不同文本在其專業領域上常用字的特徵值。

tf-idf簡要版公式，實際的公式，會再比這個更複雜一點。

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

REF: https://mropengate.blogspot.com/2016/04/tf-idf-in-r-language.html

方法三

利用預訓練的Word Embedding

1) 在這裡, 使用GloVe預訓練的詞向量
2) 並且將每個詞向量, 使用簡單的加總平均, 視作為那一個文本的文本向量
3) 在額外的試驗中, 也試過使用TF-IDF進行加權, 但是效果和單純使用mean GloVe embedding一樣

示意圖…

	f1	f2	f3	f4	f5
This	0.1	0.5	0.3	0.6	0.7
is	0.2	0.4	0.3	0.1	0.2
a			…		
book			…		
			…		
	0.1	0.4	0.3	0.3	0.4

預先訓練好, 給定的feature values

什麼是GloVe?(1/2)

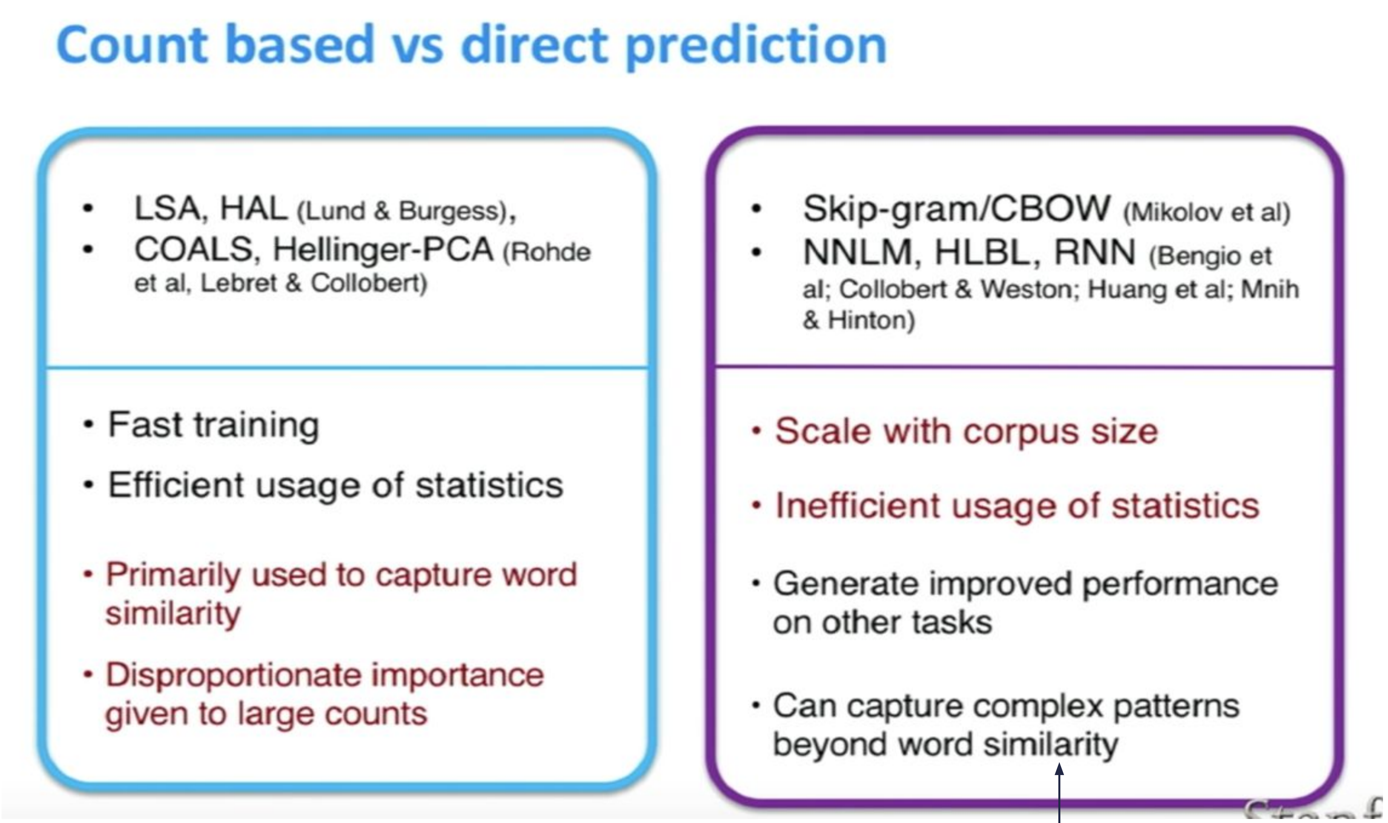
Global Vector for Word Representation

- 嘗試結合global matrix factorization method(SVD) 和 local context window based method(skip-gram).

- 右圖可以看到SVD和Skip-gram在演算法上的優缺點，紅色字為劣勢，黑色字為優勢。

Matrix Factorization

Local Context Window



像是Parts of Speech/Name Entity Recognition etc.

REF: https://youtu.be/ASn7ExxLZws?t=2248

什麼是GloVe?(2/2)

Global Vector for Word Representation

- 嘗試結合global matrix factorization method(SVD) 和 local context window based method(skip-gram).

- 這樣一方面可以考量每個字在它的context window的涵義，也可以將這個字在整個corpus當中與其它字的關係納入。

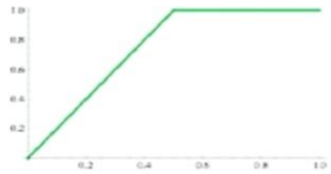
從GloVe的loss function當中來看，它如何同時納入 global statistics 和 local context info

Combining the best of both worlds: GloVe

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus, and small vectors
- By Pennington, Socher, Manning (2014)

$f \sim$



使用整體corpus的co-occurrence matrix

每個詞向量的內積，使其與co-occurrence的log機率誤差越小，這種詞向量內積，就是引自skip-gram的詞向量更新方法

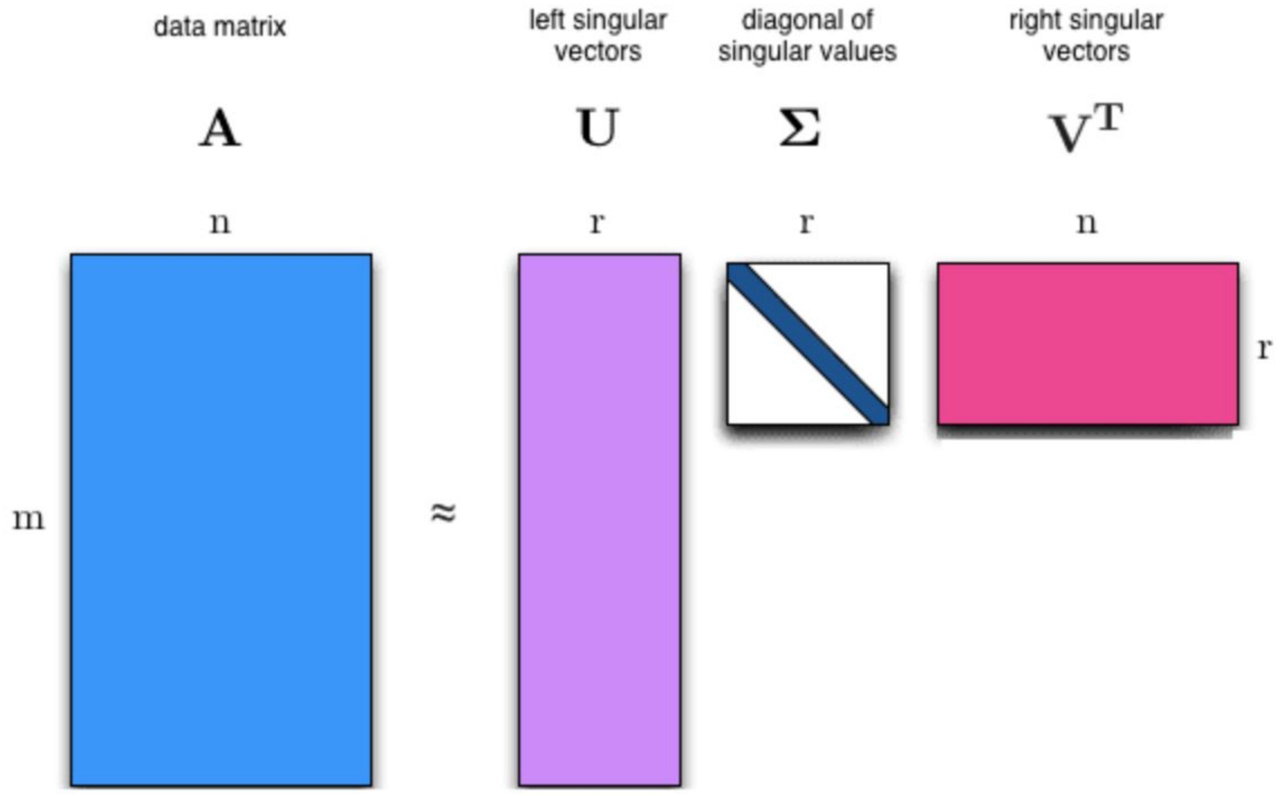
補充: SVD示意圖

Singular Value Decomposition

$$A = UDV^T$$

- A: original matrix
- U: left singular vectors
- D: diagonal matrix of singular values
- V: right singular vectors
- 參考網頁 (https://dev.to/mmithrakumar/singular-value-decomposition-with-tensorflow-2-0-4cnf)

SVD目的在讓原本的matrix，可以用比較小的U和V matrix來表示



方法四

使用doc2vec

直接計算每一個文本的feature values

1	**author**	**tweet**
2	Neil deGrasse Tyson	A 50-yard field goal in MetLife stadium will deflect nearly 1/2 inch due to Earth's rotation — meet the Coriolis force.
3	Cristiano Ronaldo	RT @Thiaguinhooo14: Manda um abraço em português para seus fás no Brasil ! @Cristiano #Celebrate15M
4	Ellen DeGeneres	Today I'm talking about a topic that affects all of us. Man-spreading. https://t.co/fyBUEmHj5k
5	Sebastian Ruder	New blog post giving an overview of softmax approximations for learning better word embeddings https://t.co/I7lkb5ESu5 #deeplearning #NLProc

示意圖…

f1	f2	f3	f4	f5
0.1	0.5	0.3	0.6	0.7
0.2	0.4	0.3	0.1	0.2
		…		
		…		
		…		

每一份文本, 直接產出一組feature values

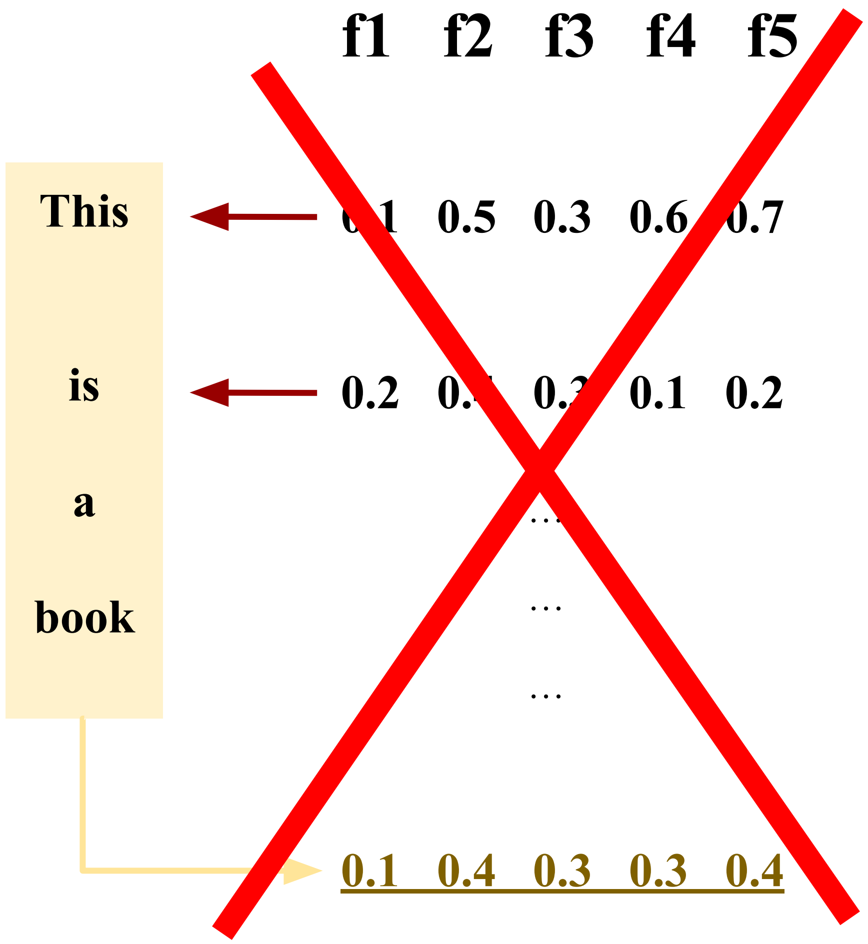
方法四

使用doc2vec

1) 針對每一個文本，直接訓練一個模型，將每個文本轉換成N維的文本向量

2) 因此在這裡，並不需要再求出文本的每個詞向量，並進行加總平均

示意圖…



如何訓練dov2vec?

- 如同word2vec，同樣有兩種方法，一種叫PV-DM(distributed memory model of paragraph vectors)，另一種是PV-DBOW(distributed bag of words version of paragraph vectors)

- 針對新的文本，運用doc2vec模型，有其特別的inference方法 - 參考這篇網頁(https://datascience.stackexchange.com/a/37501/75269)

PV-DM model和PV-DBOW model方法差異

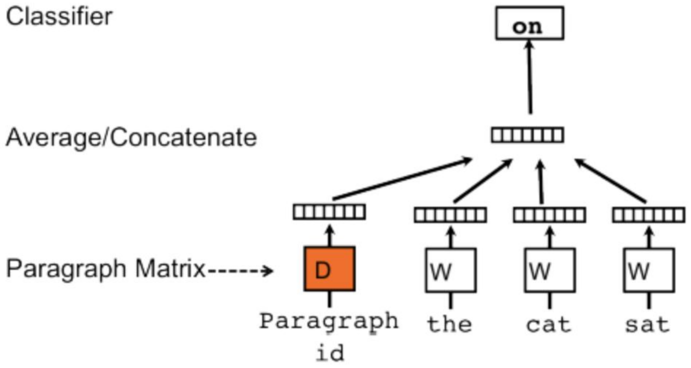


fig 3: PV-DM model

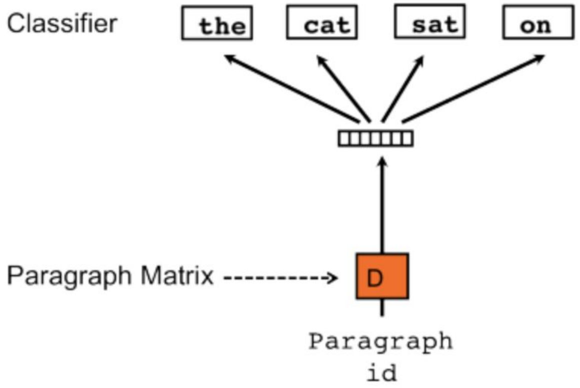


fig 4: PV-DBOW model

類似於前面word2vec當中的cbow方法，在訓練word embedding的同時，也訓練文本的vector

類似於word2vec中的skip-gram方法，直接訓練該文本的 vector

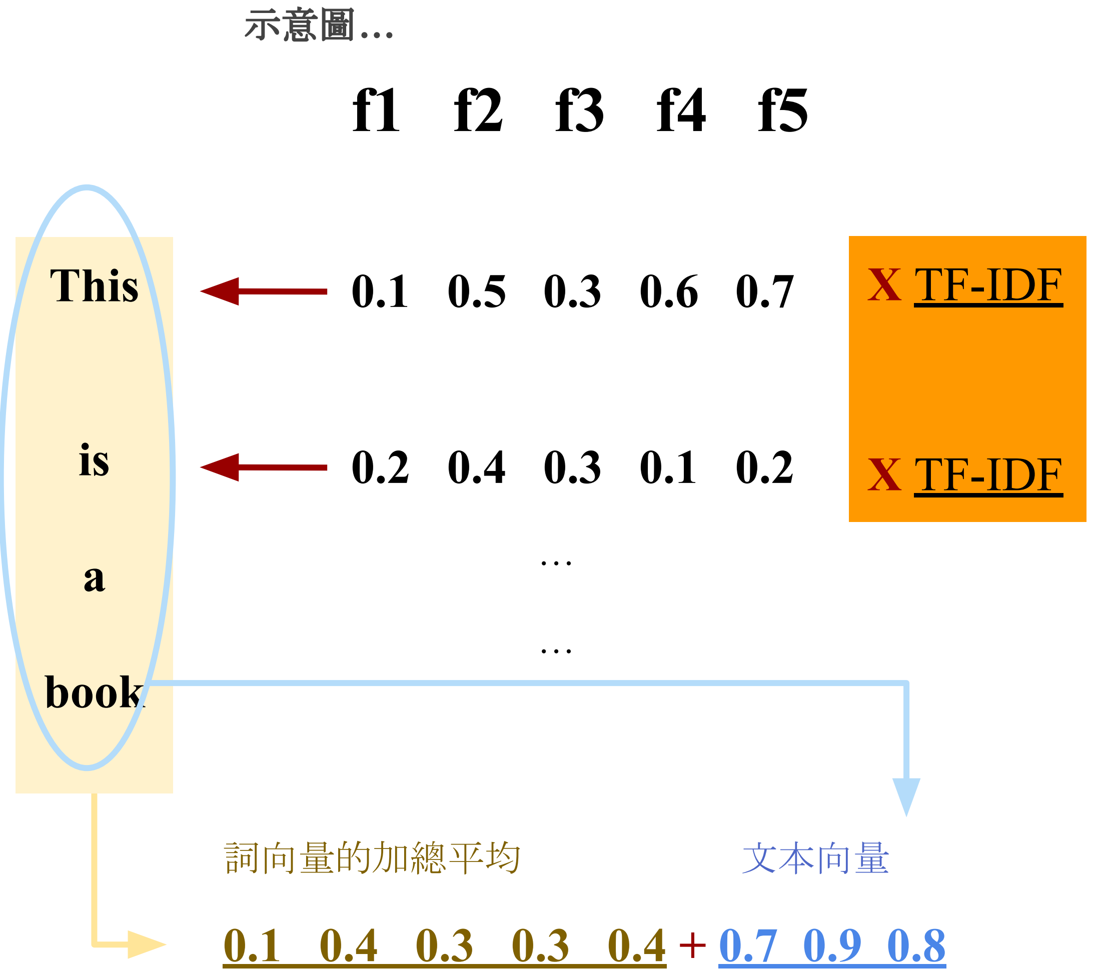
REF:
https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

方法五

合併使用word2vec與doc2vec

1) 針對每一個文本，找出它每個詞的word embedding，並且再透過tf-idf進行加權，最後進行加總平均，得到mean embedding

2) 同時再將此mean embedding和該文本的doc vector進行合併(concatenate)，再一併作為整個文本的feature values，進行分類器的訓練



小節

要進行比較的word embedding方法總共有五種:

-	Simple Averaging on Word Embedding
-	TF-IDF Weighted Averaging on Word Embedding
-	Pre-train GloVe Word Embedding
-	Doc2vec
-	TF-IDF Weighted Word Embedding and Doc2vec Combined

實作

Notebook: https://github.com/TomLin/MeetUp/blob/master/20200207-word-embedding-comparison.ipynb
Dataset: https://drive.google.com/drive/folders/1Fmhn4zwK-xE4vDYnAngAtUkfzQAVLd5j?usp=sharing

步驟一

預清理Tweet的文本

使用tweet-preprocessor套件，進行tweet文本資料清理

- 移除掉過短的文本
- 只保留英文的文本
- 將文本中的URL, Emoji, @, RT 等轉換成特殊token.
- 針對hashtag，移除掉#的字符

tweet
A 50-yard field goal in MetLife stadium will deflect nearly 1/2 inch due to Earth's rotation — meet the Coriolis force.
RT @Thiaguinhooo14: Manda um abraço em português para seus fás no Brasil ! @Cristiano #Celebrate15M
Today I'm talking about a topic that affects all of us. Man-spreading. https://t.co/fyBUEmHj5k
New blog post giving an overview of softmax approximations for learning better word embeddings https://t.co/I7lkb5ESu5 #deeplearning #NLProc
high of the day: 0 cavities 🙏🏼\nlow: @washingtonpost won't deliver physical newspapers to my zip code. Alas, I guess… https://t.co/tBDKO4naeW

↓

tweet
A 50-yard field goal in MetLife stadium will deflect nearly 1/2 inch due to Earth's rotation — meet the Coriolis force.
Today I'm talking about a topic that affects all of us. Man-spreading. *URL*
New blog post giving an overview of softmax approximations for learning better word embeddings *URL* deeplearning NLProc
high of the day: 0 cavities *EMOJIEMOJI* low: *MENTION* won't deliver physical newspapers to my zip code. Alas, I guess… *URL*
MENTION Yup. I occasionally repost after gaining more followers than the number I had the last time I posted the tweet

步驟二

將文本轉換成gensim接受的格式

TaggedDocument

- Gensim要求將每一個文本儲存在一個TaggedDocument
- 每一個TaggedDocument有兩個(key,value) pair. 分別是
  1) Words: 儲存已經過清理並且tokenized的word
  2) Tags: 每一份文件的編號

原始的文本資料

MENTION Yup. I occasionally repost after gaining more followers than the number I had the last time I posted the tweet

處理過，可進行建模的文本資料

```
TaggedDocument(words=['mention', 'occasionally', 'repost', 'gain', 'follower', 'number', 'time', 'post', 'tweet'], ta
gs=[4])
```

DocPreprocess
用來簡化資料轉換的步驟

這一連串的步驟包含了word tokenization, bi-gram detection, pos selection, and lemmatization etc..

此類會執行以下程序:

- 移除任何的標點符號，並且tokenize每一個詞
- 偵測是否有高頻率的bi-gram，並且獨立成詞
- 刪除掉stop words，並且依照自己的設定，只保留部份詞性的字(POS)
- 將每個字，還原成其原型(lemma)

```
class DocPreprocess(object):

    def __init__(self,
            nlp,
            stop_words,
            docs,
            labels,
            build_bi=False,
            min_count=5,
            threshold=10,
            allowed_postags=['ADV', 'VERB', 'ADJ', 'NOUN', 'PROPN', 'NUM']):

        self.nlp = nlp  # spacy nlp object
        self.stop_words = stop_words  # spacy.lang.en.stop_words.STOP_WORDS
        self.docs = docs  # docs must be either list or numpy array or series of docs
        self.labels = labels # labels must be list or or numpy array or series of labels
        self.doc_ids = np.arange(len(docs))
        self.simple_doc_tokens = [gensim.utils.simple_preprocess(doc, deacc=True) for doc in self.docs]

        if build_bi:
            self.bi_detector = self.build_bi_detect(self.simple_doc_tokens, min_count=min_count, threshold=threshold)
            self.new_docs = self.make_bigram_doc(self.bi_detector, self.simple_doc_tokens)
        else:
            self.new_docs = self.make_simple_doc(self.simple_doc_tokens)
        self.doc_words = [self.lemmatize(doc, allowed_postags=allowed_postags) for doc in self.new_docs]
        self.tagdocs = [TaggedDocument(words=words, tags=[tag]) for words, tag in zip(self.doc_words, self.doc_ids)]
```

步驟三

訓練word2vec模型

使用gensim預設的word2vec函數, 並且將參數設定為

- Word embedding的維度為100
- Local context window的size為5
- 設定loop整個corpus的iteration為100次

```
word_model = Word2Vec(all_docs.doc_words,
                      min_count=2,
                      size=100,
                      window=5,
                      workers=workers,
                      iter=100)
```

步驟四

進行詞向量的平均

利用簡單的加總平均，得出每個文本的特徵值

在此參考了兩篇範例，並予以統整，而以一個MeanEmbeddingVectorizer，進行包裝

- Text Classification With Word2Vec (http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/)
- Multi-Class Text Classification Model Comparison and Selection (https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568)

```
from UtilWordEmbedding import MeanEmbeddingVectorizer


mean_vec_tr = MeanEmbeddingVectorizer(word_model)
doc_vec = mean_vec_tr.transform(all_docs.doc_words)

# print('Demo of word averaging doc vector...')
# display(doc_vec[4])
```

MeanEmbedding Vectorizer

用來簡化計算Doc Vector的步驟

此類有以下方法:

- word_average: 將每個字的embedding進行平均
- word_average_list: 將每份文件stack起來
- transform: 學習sklearn的protocol, 用來啟動doc vec的計算

```python
class MeanEmbeddingVectorizer(object):

    def __init__(self, word_model):
        self.word_model = word_model
        self.vector_size = word_model.wv.vector_size

    def fit(self):  # comply with scikit-learn transformer requirement
        return self

    def transform(self, docs):  # comply with scikit-learn transformer requirement
        doc_word_vector = self.word_average_list(docs)
        return doc_word_vector

    def word_average(self, sent):
        """
        Compute average word vector for a single doc/sentence.


        :param sent: list of sentence tokens
        :return:
            mean: float of averaging word vectors
        """
        mean = []
        for word in sent:
            if word in self.word_model.wv.vocab:
                mean.append(self.word_model.wv.get_vector(word))

        if not mean:  # empty words
            # If a text is empty, return a vector of zeros.
            logging.warning("cannot compute average owing to no vector for {}".format(sent))
            return np.zeros(self.vector_size)
        else:
            mean = np.array(mean).mean(axis=0)
            return mean


    def word_average_list(self, docs):
        """
        Compute average word vector for multiple docs, where docs had been tokenized.

        :param docs: list of sentence in list of separated tokens
        :return:
            array of average word vector in shape (len(docs),)
        """
        return np.vstack([self.word_average(sent) for sent in docs])
```

步驟五

再微調，添入TF-IDF權數

將字進行TF-IDF加權，再得出每個文本的特徵值

以一個TfidfEmbeddingVectorizer，進行包裝

- 使用fit方法，計算每個字的IDF
- 再用transform方法，來得出每個doc vec

```
from UtilWordEmbedding import TfidfEmbeddingVectorizer


tfidf_vec_tr = TfidfEmbeddingVectorizer(word_model)
tfidf_vec_tr.fit(all_docs.doc_words)  # fit tfidf model first
tfidf_doc_vec = tfidf_vec_tr.transform(all_docs.doc_words)
```

TfidfEmbedding Vectorizer
用來簡化計算Doc Vector的步驟

此類有以下方法:

- fit: 得出每個字的idf
- word_average: 將每個字的embedding進行平均, 在計算word average時, 已經將tf納入考量
- transform: 仿照sklearn的protocol, 用來啟動doc vec的計算

```python
def fit(self, docs):  # comply with scikit-learn transformer requirement
        """
        Fit in a list of docs, which had been preprocessed and tokenized,
        such as word bi-grammed, stop-words removed, lemmatized, part of speech filtered.

        Then build up a tfidf model to compute each word's idf as its weight.
        Noted that tf weight is already involved when constructing average word vectors, and thus omitted.

        :param
                pre_processed_docs: list of docs, which are tokenized
        :return:
                self
        """

        text_docs = []
        for doc in docs:
                text_docs.append(" ".join(doc))

        tfidf = TfidfVectorizer()
        tfidf.fit(text_docs)  # must be list of text string

        # if a word was never seen - it must be at least as infrequent
        # as any of the known words - so the default idf is the max of
        # known idf's
        max_idf = max(tfidf.idf_)  # used as default value for defaultdict
        self.word_idf_weight = defaultdict(lambda: max_idf,
                                           [(word, tfidf.idf_[i]) for word, i in tfidf.vocabulary_.items()])
        return self

  def word_average(self, sent):
        """
        Compute average word vector for a single doc/sentence.


        :param sent: list of sentence tokens
        :return:
                mean: float of averaging word vectors
        """

        mean = []
        for word in sent:
                if word in self.word_model.wv.vocab:
                        mean.append(self.word_model.wv.get_vector(word) * self.word_idf_weight[word])  # idf weighted

        if not mean:  # empty words
                # If a text is empty, return a vector of zeros.
                logging.warning("cannot compute average owing to no vector for {}".format(sent))
                return np.zeros(self.vector_size)
        else:
                mean = np.array(mean).mean(axis=0)
                return mean
```

步驟六

使用GloVe Word Embedding

使用預訓練的Word Embedding，來計算每個文本的特徵值

```
from gensim.test.utils import get_tmpfile, datapath
from gensim.models import KeyedVectors
from gensim.scripts.glove2word2vec import glove2word2vec



# Load in GloVe vector.
glove_vec_fi = datapath('glove.twitter.27B/glove.twitter.27B.100d.txt')
tmp_word2vec_fi = get_tmpfile('tmp_glove2word2vec.txt')

glove2word2vec(glove_vec_fi, tmp_word2vec_fi)

glove_word_model = KeyedVectors.load_word2vec_format(tmp_word2vec_fi)
```

```
# Apply word averaging on GloVe word vector.
glove_mean_vec_tr = MeanEmbeddingVectorizer(glove_word_model)
glove_doc_vec = glove_mean_vec_tr.transform(all_docs.doc_words)
```

利用gensim讀進GloVe的資料檔，需要有幾個步驟

- 使用datapath，指向下載的資料檔
- 再用get_tmpfile，來設定暫時的路徑
- 利用glove2word2vec來串接datapath的檔案，和設定的暫時路徑
- 再利用KeyedVectors來讀進glove model
- 最後，再同樣使用MeanEmbeddingVectorizer來計算每個文本的doc vec

步驟七

改用doc2vec模型

直接計算出每一個文本的doc vec

有兩種演算法，一為PV-DM(為**預設方法**)，另一種為PV-DBOW，在此以DocModel類，進行包裝

```
from UtilWordEmbedding import DocModel

# Configure keyed arguments for Doc2Vec model.
dm_args = {
    'dm': 1,
    'dm_mean': 1,
    'vector_size': 100,
    'window': 5,
    'negative': 5,
    'hs': 0,
    'min_count': 2,
    'sample': 0,
    'workers': workers,
    'alpha': 0.025,
    'min_alpha': 0.025,
    'epochs': 100,
    'comment': 'alpha=0.025'
}

# Instantiate a pv-dm model.
dm = DocModel(docs=all_docs.tagdocs, **dm_args)

dm.custom_train()
```

- 先將模型訓練的參數，放入dm_args，再餵入DocModel
- 利用custom_train方法，進行模型訓練
- 每一個doc vec，可從dm.model.docvecs當中提取出來

```
# Save doc2vec as feature dataframe.
dm_doc_vec_ls = []
for i in range(len(dm.model.docvecs)):
    dm_doc_vec_ls.append(dm.model.docvecs[i])


dm_doc_vec = pd.DataFrame(dm_doc_vec_ls)
print('Shape of dm doc2vec...')
display(dm_doc_vec.shape)

print('Save dm doc2vec as csv file...')
dm_doc_vec.to_csv(os.path.join(dir_path, 'dm_doc_vec.csv'), index=False, header=False)
```

```
Shape of dm doc2vec...

(19418, 100)

Save dm doc2vec as csv file...
```

DocModel
用來簡化訓練doc2vec的模型

- 在初始化DocModel這個類時，需要放入參數1)List of Tagged Document, 2) arguments for Doc2vec模型訓練

- 之後再利用custom_train方法，來進行模型訓練，其中有固定的learning rate與不固定的learning rate(效果較好)，兩種訓練模式

```
class DocModel(object):

    def __init__(self, docs, **kwargs):
        """

        :param docs: list of TaggedDocument
        :param kwargs: dictionary of (key,value) for Doc2Vec arguments
        """
        self.model = Doc2Vec(**kwargs)
        self.docs = docs
        self.model.build_vocab([x for x in self.docs])

    def custom_train(self, fixed_lr=False, fixed_lr_epochs=None):
        """
        Train Doc2Vec with two options, without fixed learning rate(recommended) or with fixed learning rate.
        Fixed learning rate also includes implementation of shuffling training dataset.


        :param fixed_lr: boolean
        :param fixed_lr_epochs: num of epochs for fixed lr training
        """
        if not fixed_lr:
            self.model.train([x for x in self.docs],
                             total_examples=len(self.docs),
                             epochs=self.model.epochs)
        else:
            for _ in range(fixed_lr_epochs):
                self.model.train(utils.shuffle([x for x in self.docs]),
                                 total_examples=len(self.docs),
                                 epochs=1)
                self.model.alpha -= 0.002
                self.model.min_alpha = self.model.alpha  # fixed learning rate
```

步驟八

建立分類器模型

主要使用羅吉斯模型作為文本分類器

在此先將建模流程進行打包

- 先定義一個main函數，來打包所有建模的步驟
- 再定義一個sk_evaluate，來打包模型的評估指標

```python
def main(model, df, concate, concat_df):
    if concate:
        df = pd.concat([df, concat_df], axis=1, ignore_index=True)
    else:
        df = df

    # Specify train/valid/test size.
    train_size, valid_size, test_size = split_size(df, train=0.7, valid=0.)  # no need to use valid dataset here
    # Prepare test dataset.
    train_X, test_X, train_y, test_y = train_test_split(df,
                                                        target_labels,
                                                        test_size=test_size,
                                                        random_state=1,
                                                        stratify=target_labels)

    # Prepare valid dataset.
    if valid_size != 0:
        train_X, valid_X, train_y, valid_y = train_test_split(train_X,
                                                              train_y,
                                                              test_size=valid_size,
                                                              random_state=1,
                                                              stratify=train_y)

    print('Shape of train_X: {}'.format(train_X.shape))
    print('Shape of valid_X: {}'.format(valid_X.shape if 'valid_X' in vars() else (0,0)))
    print('Shape of text_X: {}'.format(test_X.shape))

    model.fit(train_X, train_y)

    if valid_size != 0:
        return model, train_X, valid_X, test_X, train_y, valid_y, test_y
    else:
        return model, train_X, None, test_X, train_y, None, test_y
```

```python
def sk_evaluate(model, feature, label, label_names):
        pred = model.predict(feature)
        true = np.array(label)

        print('Score on dataset...\n')
        print('Confusion Matrix:\n', confusion_matrix(true, pred))
        print('\nClassification Report:\n', classification_report(true, pred, target_names=label_names))
        print('\naccuracy: {:.3f}'.format(accuracy_score(true, pred)))
        print('f1 score: {:.3f}'.format(f1_score(true, pred, average='weighted')))

        return pred, true
```

結論

不同詞向量的表現(1/2)

以tf-idf與doc2vec的合併特徵值，有最好的表現，但是不明顯

- 可以看出以GloVe和Doc2vec兩種方法所造出來的feature values，在分類器上的表現最差
- 而使用單純的word2vec就有很好的表現，如果合併word2vec和doc2vec，會有最好的表現，但效果不明顯

Celebrity Tweets

WordEmbedding Method	F1 Score - Training	F1 Score - Testing	Accuracy - Training	Accuracy - Testing
Mean Word2vec	0.73	0.71	0.73	0.71
Tf-Idf Mean Word2vec	0.73	0.71	0.73	0.71
GloVe Mean Word2vec	0.65	0.63	0.66	0.64
PV-DM Doc2vec	0.59	0.57	0.59	0.57
Tf-Idf Word2vec + Doc2vec	0.76	0.72	0.76	0.73

結果

不同詞向量的表現(2/2)

讓我們測試另一組資料集，來看其表現

以下使用的資料集，是來自USA關於Customer Complaints on Financial Service的資料

- 同樣地GloVe和Doc2vec兩種方法所造出來的feature values，在分類器上的表現最差
- 同樣地也是合併word2vec和doc2vec，會有最好的表現，但效果不明顯

Customer Complaints

WordEmbedding Method	F1 Score - Training	F1 Score - Testing	Accuracy - Training	Accuracy - Testing
Mean Word2vec	0.82	0.81	0.82	0.81
Tf-Idf Mean Word2vec	0.82	0.81	0.82	0.81
GloVe Mean Word2vec	0.72	0.71	0.73	0.72
PV-DM Doc2vec	0.79	0.78	0.79	0.78
Tf-Idf Word2vec + Doc2vec	0.84	0.81	0.85	0.82

結果

反思

1 因為我設定的word vector 維度為100，有可能以更多的維度，例如 200或是300，不同word embedding的效果會有不同。

2 為了減少等待的時間，我訓練模型的 epoch iteration也不是很高，多設定為 100次左右，或許使用更多的epoch訓練次數，會有不同的效果。

3 目前使用的文本，都是短文本，例如 tweets，或許可以測試長文本，詞向量的表現會不一樣。

結果

探討主題回顧

Tweet Preprocessor	Tagged Document	Mean Embedding
Stop Word /Lemma	word2vec	CBOW
POS	TF-IDF	
Spacy	GloVe	
gensim	doc2vec	PV-DM

參考資料(1/2)

- [Medium Blog] Performance of Different Word Embeddings on Text Classification - https://towardsdatascience.com/nlp-performance-of-different-word-embeddings-on-text-classification-de648c6262b
- [Github] https://github.com/TomLin/Playground/blob/master/04-Model-Comparison-Word2vec-Doc2vec-TfIdfWeighted.ipynb
- An implementation guide to Word2Vec using NumPy and Google Sheets - https://towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281
- 引起你對 Word2Vec 基本概念 - https://medium.com/daai/%E5%BC%95%E8%B5%B7%E4%BD%A0%E5%B0%8D-word2vec-%E5%9F%BA%E6%9C%AC%E6%A6%82%E5%BF%B5-524c8b758f99
- 中文文本探勘初探: TF-IDF in R Language - https://mropengate.blogspot.com/2016/04/tf-idf-in-r-language.html
- Lecture 3 | GloVe: Global Vectors for Word Representation - https://youtu.be/ASn7ExxLZws?t=2248
- A gentle introduction to Doc2Vec - https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e
- An Intuitive Introduction to Document Vector(Doc2Vec) - https://medium.com/towards-artificial-intelligence/an-intuitive-introduction-of-document-vector-doc2vec-42c6205ca5a2

參考資料(2/2)

- Multi-Class Text Classification with Doc2Vec & Logistic Regression - https://towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4
- Text Classification With Word2Vec - http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/
- Multi-Class Text Classification Model Comparison and Selection - https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568
- Doc2vec tutorial - https://rare-technologies.com/doc2vec-tutorial/
- Word2Vec model Introduction (skip-gram & CBOW) - http://zongsoftwarenote.blogspot.com/2017/04/word2vec-model-introduction-skip-gram.html

Q&A