Natural Language processing

FastText and Word2vec example

Pre treatment and modelling

1. Import libraries

```
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['figure.dpi'] = 300
pd.options.display.max_colwidth = 200
```

2. Sample corpus: a small example

```
corpus = [
    'The sky is blue and beautiful.', 'Love this blue and beautiful sky!',
    'The quick brown fox jumps over the lazy dog.',
    "A king's breakfast has sausages, ham, bacon, eggs, toast and beans",
    'I love green eggs, ham, sausages and bacon!',
    'The brown fox is quick and the blue dog is lazy!',
    'The sky is very blue and the sky is very beautiful today',
    'The dog is lazy but the brown fox is quick!'
]
labels = [
    'weather', 'weather', 'animals', 'food', 'food', 'animals', 'weather',
    'animals'
]

corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus, 'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df
```

corpus = [
'The sky is blue and beautiful.', 'Love this blue and beautiful sky!',
'The quick brown fox jumps over the lazy dog.',
"A king's breakfast has sausages, ham, bacon, eggs, toast and beans",
'I love green eggs, ham, sausages and bacon!',
'The brown fox is quick and the blue dog is lazy!',
'The sky is very blue and the sky is very beautiful today',
'The dog is lazy but the brown fox is quick!'
]

3. Pre processing

```
wpt = nltk.WordPunctTokenizer()
# stop_words = nltk.corpus.stopwords.words('english')
def preprocess_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I | re.A)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    doc = ' '.join(tokens)
    return doc

corpus_norm = [preprocess_document(text) for text in corpus]
corpus_tokens = [preprocess_document(text).split(' ') for text in corpus]
```

4. Print corpus

```
print(corpus_norm)
print(corpus_tokens)
```

5. Training embeddings using word2vec

```
%%time

from gensim.models import word2vec

# Set values for various parameters
feature_size = 10
window_context = 5
min_word_count = 1

w2v_model = word2vec.Word2Vec(
    corpus_tokens,
    #size=feature_size,         # Word embeddings dimensionality
    window=window_context,     # Context window size
    min_count=min_word_count, # Minimum word count
    sg=1,                      # `1` for skip-gram; otherwise CBOW.
    seed = 123,                # random seed
    workers=1,                 # number of cores to use
    negative = 5,              # how many negative samples should be drawn
    cbow_mean = 1,             # whether to use the average of context word embeddings or sum(concat)
    #iter=10000,                # number of epochs for the entire corpus
    batch_words=10000,         # batch size
)
```

6. Visualize embeddings in a 2D plot

```
from sklearn.manifold import TSNE

words = w2v_model.wv.index_to_key ## get the word forms of voculary
wvs = w2v_model.wv[words] ## get embeddings of all word forms

tsne = TSNE(n_components=2, random_state=0, n_iter=5000, perplexity=5)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words

plt.figure(figsize=(12, 6))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label,
                 xy=(x + 1, y + 1),
                 xytext=(0, 0),
                 textcoords='offset points')
```

7. We can easily extract embeddings for any specific words

```
w2v_model.wv.index_to_key[:5]
```

8. We can inspect also the final vector for a word

```
[w2v_model.wv[w] for w in w2v_model.wv.index_to_key[:5]]
```

9. Compute the abveage embeddings of a document: doc2vec

```
def average_word_vectors(words, model, vocabulary, num_features):


    feature_vector = np.zeros((num_features, ), dtype="float64")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector, model.wv[word])
            #feature_vector = np.add(feature_vector, model.wv.get_vector(word))



    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector


def averaged_word_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index_to_key)
    features = [
        average_word_vectors(tokenized_sentence, model, vocabulary,
                             num_features) for tokenized_sentence in corpus
    ]
    return np.array(features)
```

10. Shaping up final result

```
feature_size = 100

w2v_feature_array = averaged_word_vectorizer(corpus=corpus_tokens,
                                             model=w2v_model,
                                             num_features=feature_size)
pd.DataFrame(w2v_feature_array, index=corpus_norm)
```

Clustering

11. Let's cluster by document

```
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

similarity_doc_matrix = cosine_similarity(w2v_feature_array)
similarity_doc_df = pd.DataFrame(similarity_doc_matrix)
similarity_doc_df
```

12. Plot a dendogram with the clusters

```
from scipy.cluster.hierarchy import dendrogram, linkage

Z = linkage(similarity_doc_matrix, 'ward')
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data point')
plt.ylabel('Distance')
dendrogram(Z,
           labels=corpus_norm,
           leaf_rotation=0,
           leaf_font_size=8,
           orientation='right',
           color_threshold=0.5)
plt.axvline(x=0.5, c='k', ls='--', lw=0.5)
```

13. Other clustering methods

```
from sklearn.cluster import AffinityPropagation

ap = AffinityPropagation()
ap.fit(w2v_feature_array)
cluster_labels = ap.labels_
cluster_labels = pd.DataFrame(cluster_labels, columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)

## PCA Plotting
from sklearn.decomposition import PCA

pca = PCA(n_components=2, random_state=0)
pcs = pca.fit_transform(w2v_feature_array)
labels = ap.labels_
categories = list(corpus_df['Category'])
plt.figure(figsize=(8, 6))

for i in range(len(labels)):
    label = labels[i]
    color = 'orange' if label == 0 else 'blue' if label == 1 else 'green'
    annotation_label = categories[i]
    x, y = pcs[i]
    plt.scatter(x, y, c=color, edgecolors='k')
    plt.annotate(annotation_label,
                 xy=(x + 1e-4, y + 1e-3),
                 xytext=(0, 0),
                 textcoords='offset points')
```

Modelling with GLOVE

14. Using pre trained embeddings available in spaCy

```
import spacy


nlp = spacy.load('en_core_web_sm',disable=['parse','entity'])

total_vectors = len(nlp.vocab.vectors)
print('Total word vectors:', total_vectors)
```

15. Extract words form the pre trained model

```
# get vocab of the corpus
unique_words = set(sum(corpus_tokens,[]))

# extract pre-trained embeddings of all words
word_glove_vectors = np.array([nlp(word).vector for word in unique_words])

word_glove_vectors = pd.DataFrame(word_glove_vectors)
word_glove_vectors.index = list(unique_words)
```

16. Plotting in 2D words

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=0, n_iter=5000, perplexity=5)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(word_glove_vectors)
labels = unique_words

plt.figure(figsize=(12, 6))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label,
                 xy=(x + 1, y + 1),
                 xytext=(0, 0),
                 textcoords='offset points')
```

Question: what is the roile of the perplexity hyperprameter on the TSNE function?

Clustering with GLOVE

17. Run clustering using GLOVE embeddings

```
doc_glove_vectors = np.array([nlp(str(doc)).vector for doc in corpus_norm])

import sklearn
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3, random_state=0)
km.fit_transform(doc_glove_vectors)
cluster_labels = km.labels_
cluster_labels = pd.DataFrame(cluster_labels, columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)
```

Question: Perform other type of clustering of your choice and plot the result

Using FastText

18. Import libraries and downloading a corpus from NLTK

```
from gensim.models.fasttext import FastText

import nltk
nltk.download('brown')

brown_tokens = [brown.words(fileids=f) for f in brown.fileids()]
```

19. Training a FastText model

```
%%time
# Set values for various parameters
feature_size = 100  # Word vector dimensionality
window_context = 5  # Context window size
min_word_count = 5  # Minimum word count

ft_model = FastText(brown_tokens,
                    #size=feature_size,
                    window=window_context,
                    min_count=min_word_count,
                    sg=1,
                    #iter=50
                    )
```

20. Print some similar words as an example.

```
# view similar words based on gensim's model
similar_words = {
    search_term:
    [item[0] for item in ft_model.wv.most_similar([search_term], topn=5)]
    for search_term in
    ['think', 'say','news', 'report','nation', 'democracy']
}
similar_words
```

21. Plotting with PCA

```
from sklearn.decomposition import PCA

words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = ft_model.wv[words]

pca = PCA(n_components=2)
np.set_printoptions(suppress=True)
P = pca.fit_transform(wvs)
labels = words

plt.figure(figsize=(12, 10))
plt.scatter(P[:, 0], P[:, 1], c='lightgreen', edgecolors='g')
for label, x, y in zip(labels, P[:, 0], P[:, 1]):
    plt.annotate(label,
                 xy=(x + 0.03, y + 0.03),
                 xytext=(0, 0),
                 textcoords='offset points')
```

Question: What are the differences on projection into lower dimentions a dataset with PCA Vs TSNE?

22. We can visualize a vector, given a word

```
ft_model.wv['democracy']
```

23. We can compare similarity of 2 or more words

```
print(ft_model.wv.similarity(w1='taiwan', w2='freedom'))
print(ft_model.wv.similarity(w1='china', w2='freedom'))
```