Import necessary dependencies and settings

In [1]:
```
import pandas as pd
import numpy as np
import re
import nltk
```

Sample corpus of text documents

In [2]:
```
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          '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', 'animals', 'weather', 'animals']
corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus,
                          'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df
```

Out[2]:

	Document	Category
0	The sky is blue and beautiful.	weather
1	Love this blue and beautiful sky!	weather
2	The quick brown fox jumps over the lazy dog.	animals
3	The brown fox is quick and the blue dog is lazy!	animals
4	The sky is very blue and the sky is very beaut...	weather
5	The dog is lazy but the brown fox is quick!	animals

Simple text pre-processing

In [3]:
```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

In [4]:
```
norm_corpus = normalize_corpus(corpus)
norm_corpus
```

Out[4]:
```
array(['sky blue beautiful', 'love blue beautiful sky',
       'quick brown fox jumps lazy dog', 'brown fox quick blue dog lazy',
       'sky blue sky beautiful today', 'dog lazy brown fox quick'],
      dtype='<U30')
```

Bag of Words Model

In [5]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(norm_corpus)
cv_matrix = cv_matrix.toarray()
cv_matrix
```

Out[5]:
```
array([[1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
       [0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1],
       [0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0]], dtype=int64)
```

In [6]:
```
vocab = cv.get_feature_names()
pd.DataFrame(cv_matrix, columns=vocab)
```

Out[6]:

	beautiful	blue	brown	dog	fox	jumps	lazy	love	quick	sky	today
0	1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	1	0	1	0
2	0	0	1	1	1	1	1	0	1	0	0
3	0	1	1	1	1	0	1	0	1	0	0
4	1	1	0	0	0	0	0	0	0	2	1
5	0	0	1	1	1	0	1	0	1	0	0

Bag of N-Grams Model

In [7]:
```
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(norm_corpus)
bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)
```

Out[7]:

	beautiful sky	beautiful today	blue beautiful	blue dog	blue sky	brown fox	dog lazy	fox jumps	fox quick	jumps lazy	lazy brown	lazy dog
0	0	0	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	1	0	1	0	1
3	0	0	0	1	0	1	1	0	1	0	0	0
4	0	1	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	0	1	1	0	1	0	1	0

TF-IDF Model

In [8]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(norm_corpus)
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

Out[8]:

	beautiful	blue	brown	dog	fox	jumps	lazy	love	quick	sky	today
0	0.60	0.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00
1	0.46	0.39	0.00	0.00	0.00	0.00	0.00	0.66	0.00	0.46	0.00
2	0.00	0.00	0.38	0.38	0.38	0.54	0.38	0.00	0.38	0.00	0.00
3	0.00	0.36	0.42	0.42	0.42	0.00	0.42	0.00	0.42	0.00	0.00
4	0.36	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.72	0.52
5	0.00	0.00	0.45	0.45	0.45	0.00	0.45	0.00	0.45	0.00	0.00

Document Similarity

In [9]:
```
from sklearn.metrics.pairwise import cosine_similarity

similarity_matrix = cosine_similarity(tv_matrix)
similarity_df = pd.DataFrame(similarity_matrix)
similarity_df
```

Out[9]:

	0	1	2	3	4	5
0	1.000000	0.753128	0.000000	0.185447	0.807539	0.000000
1	0.753128	1.000000	0.000000	0.139665	0.608181	0.000000
2	0.000000	0.000000	1.000000	0.784362	0.000000	0.839987
3	0.185447	0.139665	0.784362	1.000000	0.109653	0.933779
4	0.807539	0.608181	0.000000	0.109653	1.000000	0.000000
5	0.000000	0.000000	0.839987	0.933779	0.000000	1.000000

Clustering documents using similarity features

In [10]:
```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=2)
km.fit_transform(similarity_df)
cluster_labels = km.labels_
cluster_labels = pd.DataFrame(cluster_labels, columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)
```

Out[10]:

	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	0
1	Love this blue and beautiful sky!	weather	0
2	The quick brown fox jumps over the lazy dog.	animals	1
3	The brown fox is quick and the blue dog is lazy!	animals	1
4	The sky is very blue and the sky is very beaut...	weather	0
5	The dog is lazy but the brown fox is quick!	animals	1

Topic models

In [12]:
```
from sklearn.decomposition import LatentDirichletAllocation

lda = LatentDirichletAllocation(2, max_iter=100, random_state=42)
dt_matrix = lda.fit_transform(tv_matrix)
features = pd.DataFrame(dt_matrix, columns=['T1', 'T2'])
features
```

Out[12]:

	T1	T2
0	0.190548	0.809452
1	0.176804	0.823196
2	0.846184	0.153816
3	0.814863	0.185137
4	0.180516	0.819484
5	0.839172	0.160828

Show topics and their weights

In [20]:
```
tt_matrix = lda.components_
for topic_weights in tt_matrix:
    topic = [(token, weight) for token, weight in zip(vocab, topic_weights)]
    topic = sorted(topic, key=lambda x: -x[1])
    topic = [item for item in topic if item[1] > 0.6]
    print(topic)
    print()
```

```
[('brown', 1.7273638692668465), ('dog', 1.7273638692668465), ('fox', 1.7273
638692668465), ('lazy', 1.7273638692668465), ('quick', 1.7273638692668465),
('jumps', 1.0328325272484777), ('blue', 0.7731573162915626)]

[('sky', 2.264386643135622), ('beautiful', 1.9068269319456903), ('blue', 1.
7996282104933266), ('love', 1.148127242397004), ('today', 1.006825116042993
5)]
```

Clustering documents using topic model features

In [21]:
```
km = KMeans(n_clusters=2)
km.fit_transform(features)
cluster_labels = km.labels_
cluster_labels = pd.DataFrame(cluster_labels, columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)
```

Out[21]:

	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	0
1	Love this blue and beautiful sky!	weather	0
2	The quick brown fox jumps over the lazy dog.	animals	1
3	The brown fox is quick and the blue dog is lazy!	animals	1
4	The sky is very blue and the sky is very beaut...	weather	0
5	The dog is lazy but the brown fox is quick!	animals	1

Word Embeddings

In [15]:
```
from gensim.models import word2vec

wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in norm_corpus]

# Set values for various parameters
feature_size = 10    # Word vector dimensionality
window_context = 10          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                          window=window_context, min_count = min_word_count,
                          sample=sample)
```

In [16]:
```
w2v_model.wv['sky']
```

Out[16]:
```
array([ 0.02500289, -0.00986266, -0.00674801, -0.04859276,  0.02821922,
       -0.02147278,  0.04886315, -0.00462028,  0.01516211, -0.04008733],
      dtype=float32)
```

In [17]:
```
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[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.index2word)
    features = [average_word_vectors(tokenized_sentence, model, vocabulary, r
                    for tokenized_sentence in corpus]
    return np.array(features)
```

In [18]:
```
w2v_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=w
                                             num_features=feature_size)
pd.DataFrame(w2v_feature_array)
```

```
C:\Users\seanx\anaconda3\lib\site-packages\ipykernel_launcher.py:9: Deprecation
Warning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  if __name__ == '__main__':
```

Out[18]:

	0	1	2	3	4	5	6	7	
0	0.022442	0.015491	-0.019546	-0.025809	0.000003	-0.021277	-0.005345	-0.003244	0.010
1	0.009457	0.005695	-0.024482	-0.009610	-0.001352	-0.006369	0.003238	0.000318	0.007
2	-0.019058	0.027038	0.010213	0.002015	-0.009461	-0.015911	-0.003053	0.000518	-0.008
3	-0.015681	0.023278	0.004512	-0.007184	-0.002491	-0.016447	-0.003670	0.004219	-0.009
4	0.018968	0.009214	-0.010911	-0.022844	0.004278	-0.020284	0.013690	-0.008404	0.003
5	-0.021541	0.022897	0.011099	-0.002553	-0.001841	-0.012693	0.004799	-0.002710	-0.009

In [19]:
```
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)
```

Out[19]:

	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	0
1	Love this blue and beautiful sky!	weather	0
2	The quick brown fox jumps over the lazy dog.	animals	1
3	The brown fox is quick and the blue dog is lazy!	animals	1
4	The sky is very blue and the sky is very beaut...	weather	0
5	The dog is lazy but the brown fox is quick!	animals	1