CASE STUDY - topic modeling and feature engineering

Feature engineering is the process of using domain knowledge of your data to create features that can be leveraged by machine learning. That is not a hard definition, because
where features are transformed for machine learning, but the inclusion of domain knowledge is not implied.

It is unfortunately common that for large datasets engineered features are not easy to create. When there are many features generally only a small number play an important rol
Furthermore, domain insight is even more difficult to fold into the model when there are hundreds or thousands of features to keep in mind. However, there is a middle ground--
locked up in language. In this case study we will use topic modeling to gather insight from text. Ideally, the result of these types of experiments would be shared with domain e
that are relevant when it comes to your business opportunity.

In [1]:
```
%%capture
pip install -U pip
```

In [2]:
```
%%capture
pip install pyLDAvis
```

In [3]:
```
%%capture
pip install ../data/en_core_web_sm-2.3.1.tar.gz --user
```

In [4]:
```
##IMPORTANT: Please restart the Kernel after running the above 3 cells
```

In [1]:
```
import os
import re
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from string import punctuation, printable
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

try:
    import pyLDAvis
    import pyLDAvis.sklearn
except:
    raise Exception("'pip install pyldavis' before running this notebook")

pyLDAvis.enable_notebook()
plt.style.use('seaborn')
%matplotlib inline

DATA_DIR = os.path.join("..","data")
```

Synopsis

> Goal: AAVAIL has recently enabled comments on the core streaming service. The data science team knows that this will be an incredibly important sour
> will be used inform customer retention, product quality, product market fit and more. Comments are going live next week and being the diligent data sci
> is to build a topic modeling pipeline that will consume the comments and create visualizations that can be used to communicate with domain experts.

Outline

1. EDA - summary tables, use tSNE to visualize the data
2. Create a transfomation pipelines for NMF and LDA
3. Use ldaviz and wordclouds to get insight into the clusters

Data

Even before receiving the first comment, we want to start building our Pipeline using a proxy dataset. In this study Case we will work with a dataset publicly ava

- Here is the web page that references all the public dataset that NLTK provide. In this Study Case we will work with the 'Sentiment Polarity Dataset Version 2
  been downloaded and is in the data folder of the working directory)
- For more examples of applications with these data see NLTK's book chapter that uses these data

In [2]:
```
filename = os.path.join(DATA_DIR, 'movie_reviews.csv')
df = pd.read_csv(filename)
X = df['review'].tolist()
print(X[4])
```

```
b"kolya is one of the richest films i've seen in some time . \nzdenek sverak plays a confirmed old bachelor ( who's lik
as a czech cellist increasingly impacted by the five-year old boy that he's taking care of . \nthough it ends rather ab
ed to spend more time with these characters-- the acting , writing , and production values are as high as , if not high
\nthis father-and-son delight-- sverak also wrote the script , while his son , jan , directed-- won a golden globe for
e days after i saw it , walked away an oscar . \nin czech and russian , with english subtitles . \n"
```

QUESTION 1

The main focus of this exercise is to enable visualization of topics, but these topics can be used as additional features for prediction tasks. The goal of this case
comfortable with natural language processing pipelines and topic modeling tools.

There are many ways to process tokens (words, dates, emojis etc). NLTK is often used to pre-process text data before the tokens are vectorized. Generally, the
lemmatization. The next code block provides a lemmatization function that makes use of the library spacy. You will need to install it and download the English la
Stopwords are words that are very common or otherwise irrelevant we use a default list here, but it is an important part of NLP pipelines that needs to be custor

If you prefer to use NLTK then you could use a simple lemmatizer like the WordLemmatizer.

In [3]:
```
import spacy
STOPLIST = ENGLISH_STOP_WORDS
STOPLIST = set(list(STOPLIST) + ["foo", "film", "movie", "make"])

if not 'nlp' in locals():
    print("Loading English Module...")
    nlp = spacy.load('en_core_web_sm')

def lemmatize_document(doc, stop_words=None):
    """
    takes a list of strings where each string is a document
    returns a processed list of strings
    """

    if not stop_words:
        stop_words = set([])

    ## ensure working with string
    doc = str(doc)
    doc = doc.replace('\\n','')
    doc = doc.replace('\\t','')

    # First remove punctuation form string
    if sys.version_info.major == 3:
        PUNCT_DICT = {ord(punc): None for punc in punctuation}
        doc = doc.translate(PUNCT_DICT)

    # remove unicode
    clean_doc = "".join([char for char in doc if char in printable])

    # Run the doc through spaCy
    doc = nlp(clean_doc)

    # Lemmatize and lower text
    tokens = [re.sub(r"\W+","",token.lemma_.lower()) for token in doc ]
    tokens = [t for t in tokens if len(t) > 1]

    return ' '.join(w for w in tokens if w not in stop_words)

## example usage
corpus = ['"You can fool some of the people all of the time, and all of the people some of the time, but you can not fo
processed = [lemmatize_document(doc, STOPLIST) for doc in corpus]
print(processed[0])
processed = [lemmatize_document(doc, None) for doc in corpus]
print("\n"+processed[0])
```

```
Loading English Module...
pron fool people time people time pron fool people time abraham lincoln

pron can fool some of the people all of the time and all of the people some of the time but pron can not fool all of th
```

```
In [4]: ## YOUR CODE HERE

        ## Preprocess all the reviews of the corpus with the lemmatize_document() function to create a list of cleaned reviews.

        ## Applying the lemmatize_document() function to all the documents of the corpus takes several minutes.
        ## In order to save you some time we preprocessed the texts with the line of code commented bellow and saved
        ## the processed documents in a .txt file. You can either re-preprocess the text by uncommenting the lines above
        ## or you can directly read the processed_text.txt file as shown bellow.

        from tqdm import tqdm
        tqdm.pandas()
        processed = df.progress_apply(lambda x : lemmatize_document(x['review'], STOPLIST), axis=1).tolist()

        processed = []
        with open(os.path.join(DATA_DIR, 'processed_text.txt'), 'r') as f :
            for line in f:
                processed.append(line)


        print("processing done.")
```

```
100%|██████████| 2000/2000 [06:20<00:00,  5.26it/s]

processing done.
```

QUESTION 2

Use the CountVectorizer from sklearn to vectorize the documents.

Additional resources:

- scikit-learn CountVectorizer
- scikit-learn working with text

Because this is an exercise in visualization set the `max_features` to something like 500. In the context of supervised learning it is reasonable to grid-search t

```
In [5]: ## YOUR CODE HERE

        max_features = 500

        # Create a CountVectorizer object
        tf_vectorizer = CountVectorizer(max_df=0.95, min_df=2,
                                        max_features=max_features,
                                        stop_words='english')

        # Fit and transform this object to the processed reviews
        tf = tf_vectorizer.fit_transform(processed)
        print("ready")
```

```
ready
```

QUESTION 3

Fit a LDA model to the corpus. For example, you could use something like the following.

```
n_topics = 10
lda_model = LatentDirichletAllocation(n_components=n_topics, max_iter=5,
                                      learning_method='online',
                                      learning_offset=50.,
                                      random_state=0)

lda_model.fit(tf)
```

- scikit-learn's LDA
- scikit-learn's user guide for LDA

In [6]:
```
## YOUR CODE HERE
n_topics = 10

# Create an LDA object
lda_model = LatentDirichletAllocation(n_components=n_topics, max_iter=5,
                                      learning_method='online',
                                      learning_offset=50.,
                                      random_state=0)

# Fit the model to the bag of word we created earlier
lda_model.fit(tf)
```

Out[6]:
```
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
                          evaluate_every=-1, learning_decay=0.7,
                          learning_method='online', learning_offset=50.0,
                          max_doc_update_iter=100, max_iter=5,
                          mean_change_tol=0.001, n_components=10, n_jobs=None,
                          perp_tol=0.1, random_state=0, topic_word_prior=None,
                          total_samples=1000000.0, verbose=0)
```

QUESTION 4

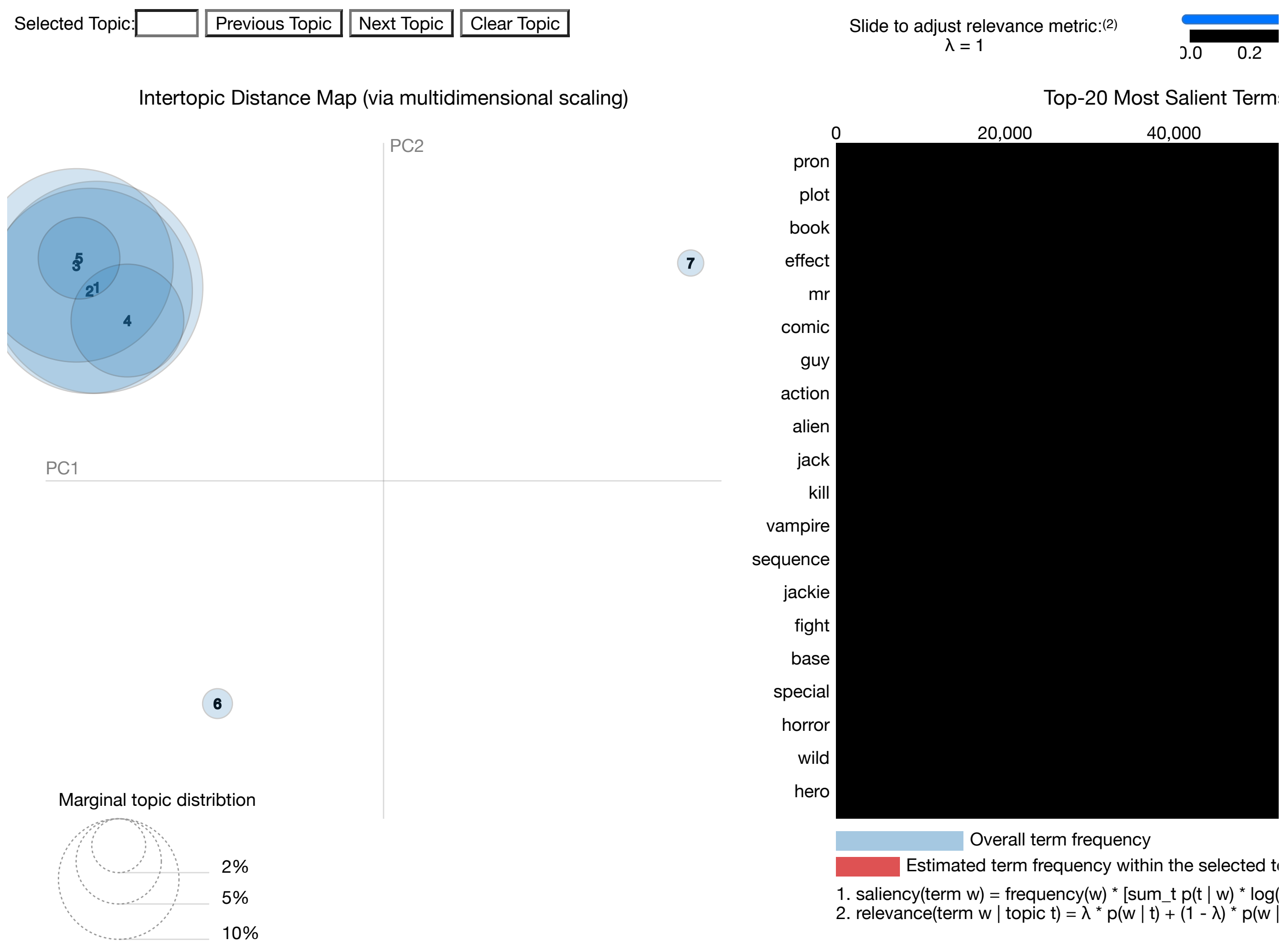
Visualize the corpus using pyldavis.

```
pyLDAvis.sklearn.prepare(lda_model,tf, tf_vectorizer, R=20)
```

- PyLDAViz documentation
- PyLDAViz demos

In [11]:
```
## YOUR CODE HERE
pyLDAvis.sklearn.prepare(lda_model, tf, tf_vectorizer, R=20)
```

Out[11]:

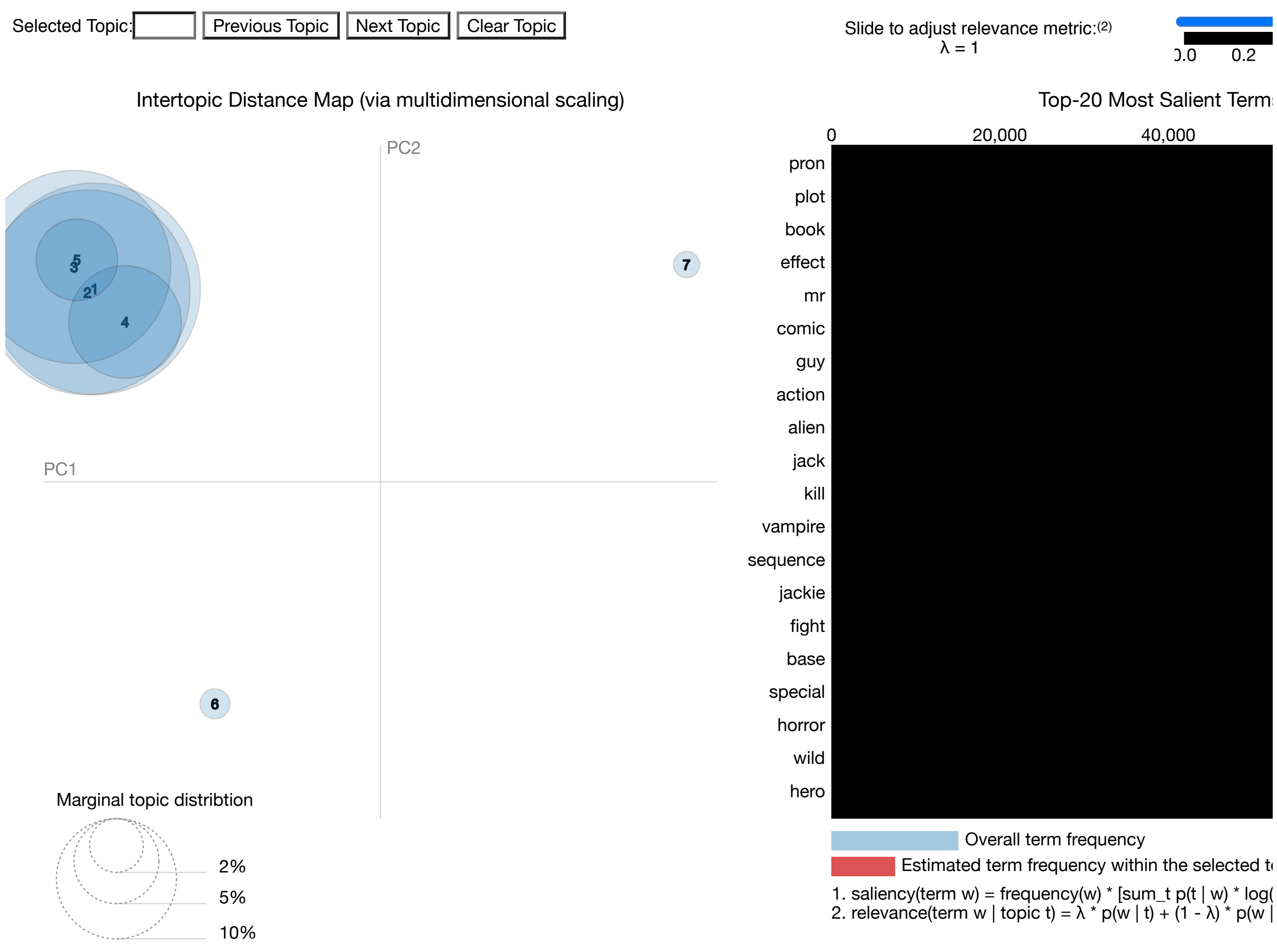


QUESTION 5

Try different numbers of clusters until there is decent separation in the visualization

In [12]:
```
## YOUR CODE HERE

n_topics = 7
lda_model = LatentDirichletAllocation(n_components=n_topics, max_iter=5,
                                      learning_method='online',
                                      learning_offset=50.,
                                      random_state=0)
lda_model.fit(tf)
lda_transformed = lda_model.transform(tf)
pyLDAvis.sklearn.prepare(lda_model, tf, tf_vectorizer, R=20)
```

Out[12]:



The visualization here can help determine a reasonable number of number of clusters and it can serve as a communication tool. If the goal was to find topics tha
profiles then you would likely work with folks in marketing to refine the clustering. There are a couple of parameters than can be used to modify the clustering ar
meaningful topics is a form of feature engineering.

QUESTION 6

If you were to use the topics from this model to inform clustering or supervised learning you would first need to be able to extract and represent them as a matri
populate a report with tabular descriptions of the data then you will need to be able to extract topic representations. Here is a starter function

In [13]:
```
def get_top_words(model, feature_names, n_top_words):
    """
    Get the top words defining the different topics of the LDA model
    INPUT : the LDA model, the names of the features of the bag of word (these are the actual words in the vocabulary)
    and the number of top words.
    RETURN : A dictionary where the keys are the topic's ID and the values are the lists of the n_top_words top words.

    """
    top_words = {}
    for topic_idx, topic in enumerate(model.components_):
        _top_words = [feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-1]]
        top_words[str(topic_idx)] = _top_words
    return top_words
```

Use the function to print the top k words for each topic

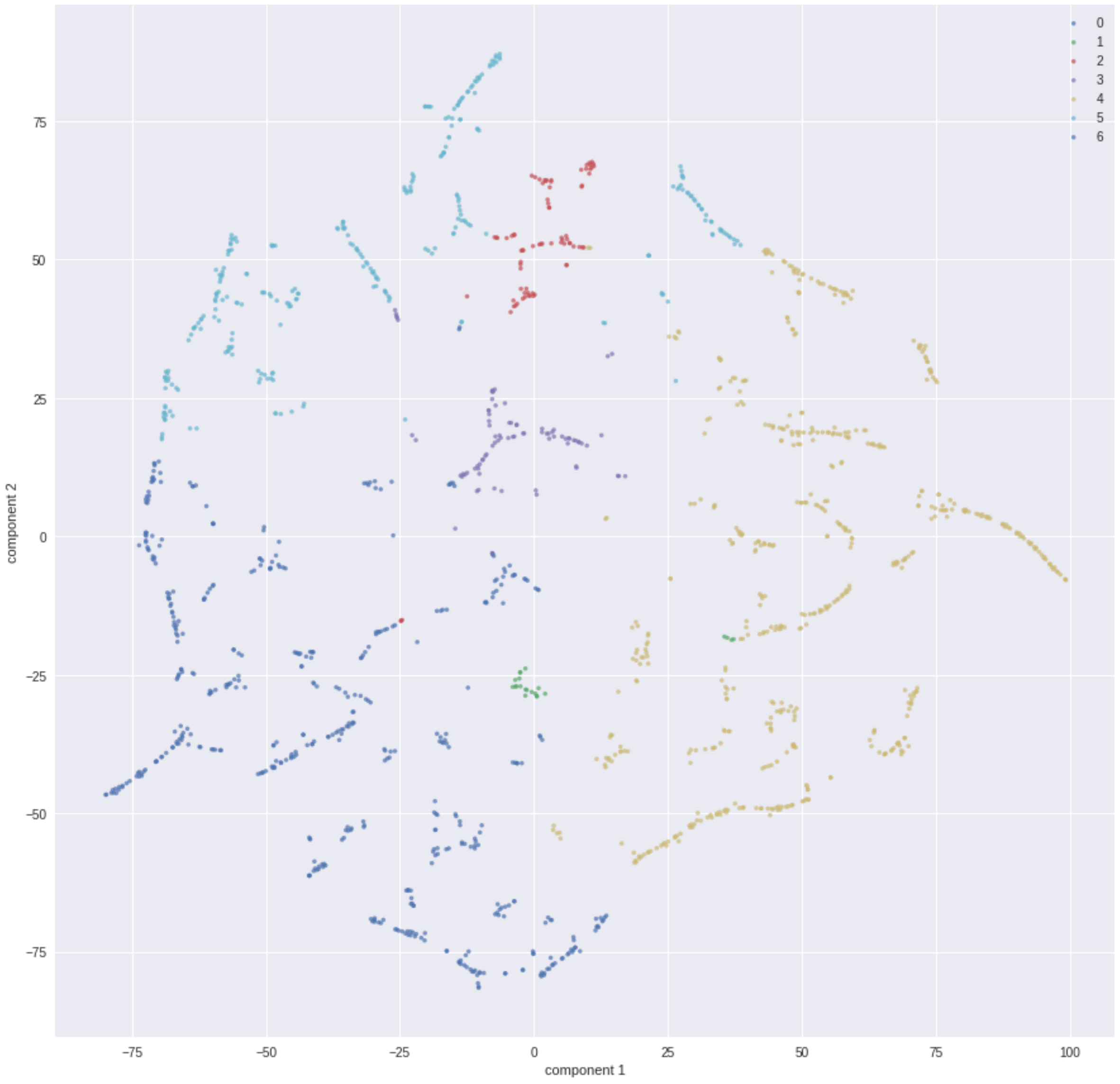
In [9]:
```
## YOUR CODE HERE

## set n_top_words
top_words = 15
## get the vectorizer's feature names
tf_feature_names = np.array(tf_vectorizer.get_feature_names())
## get the top words for each topic
top_words = get_top_words(lda_model, tf_feature_names, top_words)
all_top_words = np.array(list(set().union(*[v for v in top_words.values()])))

## print the topics and the top words of each topic
for key,vals in top_words.items():
    print(key, " ".join(vals))
print("total words: %s"%len(all_top_words))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-333cdbbbeba9> in <module>
      6 tf_feature_names = np.array(tf_vectorizer.get_feature_names())
      7 ## get the top words for each topic
----> 8 top_words = get_top_words(lda_model, tf_feature_names, top_words)
      9 all_top_words = np.array(list(set().union(*[v for v in top_words.values()])))
     10 

NameError: name 'get_top_words' is not defined
```

QUESTION (EXTRA CREDIT) 7

If you used `transform` on your original tokens you should have a `2000 x k` array where `k` is the number of topics you choose. Create a PCA or tSNE visu
into lower dimensional space then uses colors to indicate which documents belong to a topic (e.g. probability > 0.5).

```
In [15]: ## YOUR CODE HERE

         def make_plot(lda_mat):

             fig = plt.figure(figsize=(15,15), facecolor='white')
             ax = fig.add_subplot(111)

             tsne = TSNE(n_components=2, perplexity=10, init='pca')
             projected = tsne.fit_transform(lda_mat)

         #     pca = PCA(n_components=2)
         #     projected = pca.fit_transform(lda_mat)

             for class_num in np.arange(n_topics):
                 topic_inds = np.where(lda_mat[:, class_num] > 0.5)[0]
                 ax.scatter(projected[topic_inds, 0],
                            projected[topic_inds, 1],
                            edgecolor='none', marker='.', alpha=0.7, label=str(class_num))

             ax.set_xlabel('component 1')
             ax.set_ylabel('component 2')
             ax.legend()


         make_plot(lda_transformed)
```



In [10]:
```
with open(os.path.join(DATA_DIR, 'processed_text.txt'), 'w') as f:
    for item in processed:
        f.write("%s\n" % item)
```

In [ ]: