James Gray
graymatter@u.northwestern.edu

CIS-435 Summer 2015
Assignment #5 – August 24, 2014

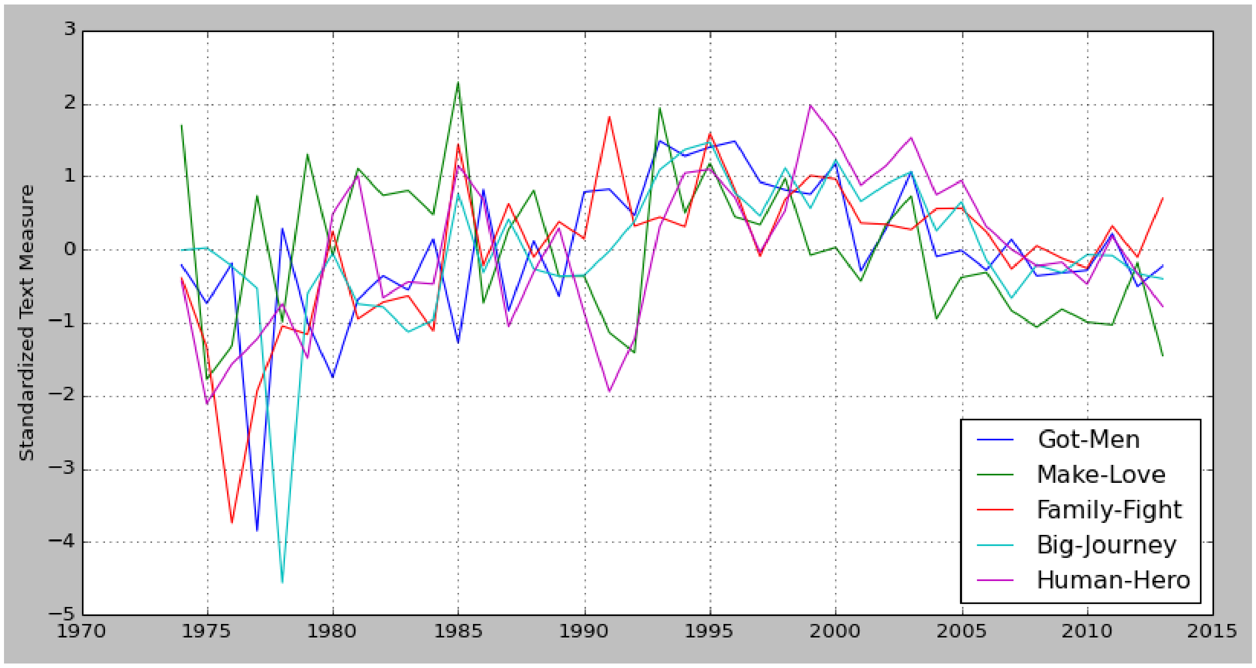
Introduction

The purpose of this assignment is cluster movie taglines over the last 40 years to identify trends based the top 200 words in these taglines.

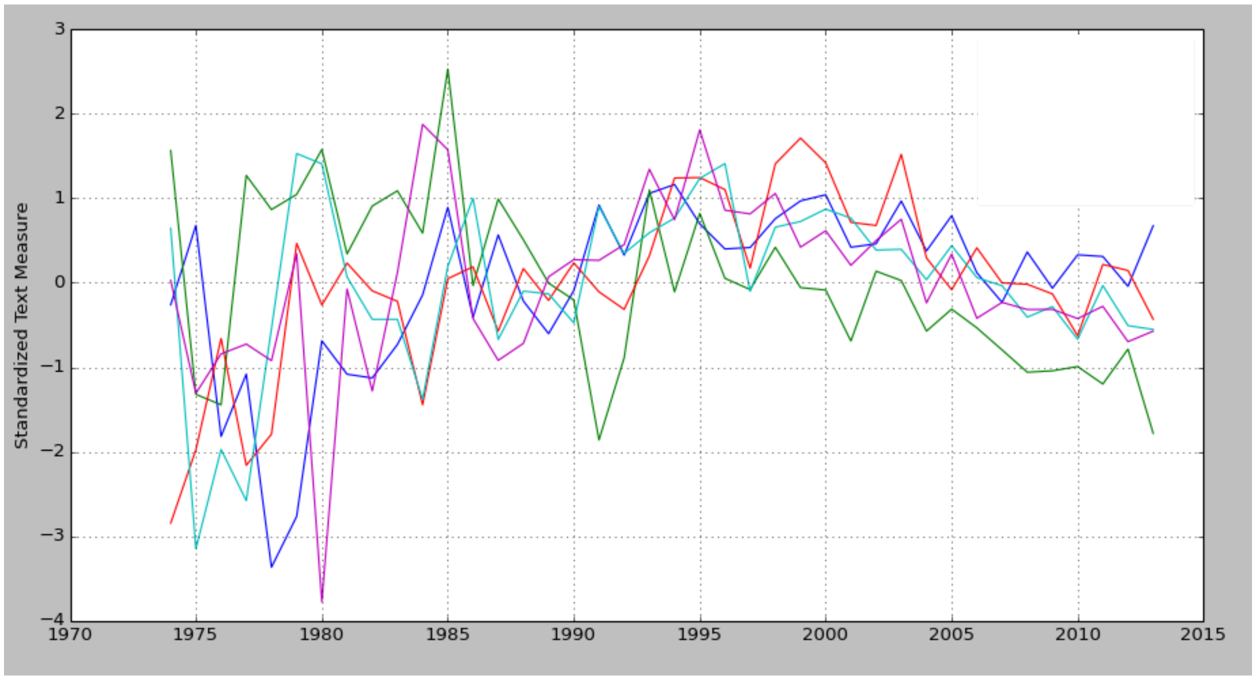
Results

I first ran the model that created 5 clusters that I named. I noticed that Cluster 0 was mainly shaped by two words (got and scream) and the remaining 8 of the top 10 words were more similar to Cluster 1.

```
 Top Words in Cluster : 0 ------------------------------
     cluster  dist_to_0  dist_to_1  dist_to_2  dist_to_3  dist_to_4    word
79         0   0.828109   1.434530   1.733250   1.639360   1.788392     got
155        0   1.009503   1.715042   1.807571   1.808545   1.710444  scream
99         1   1.049967   0.240573   1.193821   1.163687   1.320531    know
119        1   1.049967   0.240573   1.193821   1.163687   1.320531     men
115        1   1.049967   0.240573   1.193821   1.163687   1.320531     man
114        1   1.049967   0.240573   1.193821   1.163687   1.320531    make
110        1   1.049967   0.240573   1.193821   1.163687   1.320531    love
109        1   1.049967   0.240573   1.193821   1.163687   1.320531    look
108        1   1.049967   0.240573   1.193821   1.163687   1.320531    live
106        1   1.049967   0.240573   1.193821   1.163687   1.320531    like
```



I continued the analysis by removing the word "got" as it was not extremely descriptive. I then also removed "could" as it also showed up as top word in Cluster 0. This improved a longer list of worlds in Cluster 0 and the performance of the other clusters.



New cluster names were specified after executing the model with these word exclusions.

Blue = big-crime
Green = dead-men
Orange = dark-night
Cyan = human-hero
Purple = high-survival

A few conclusions can be drawn from the trends of these 5 clusters:

- Crime movies have trended slightly down from 2000 but are showing a steep increase over the last few years. This may be a market opportunity.
- Movies about death have trended down since 2000 and are showing a steep decline over the last few years.
- Movies related to darkness or night have declined since the high of 1998 but have remained relatively flat over the last ten years.
- Movies about human heroes have declined since 2000 and have remained relatively flat over the last few years.
- Movies about high survival have trended significantly downward since the high of 1995.

Code

```
# Text Analysis of Movie Taglines (Python)
# James Gray - CIS 435 Assignment #5
# Extended and adapted from Professor Thomas Miller's base Python code

# prepare for Python version 3x features and functions
from __future__ import division, print_function

# import packages for text processing and multivariate analysis
import re # regular expressions
import nltk # draw on the Python natural language toolkit
import pandas as pd # DataFrame structure and operations
import numpy as np # arrays and numerical processing
import scipy
import matplotlib.pyplot as plt # 2D plotting

# terms-by-documents matrix
# http://scikit-learn.org/stable/modules/feature_extraction.html
# CountVectorizer converts a collection of text documents to a matrix of word
# tokens (rows) and documents (columns)
from sklearn.feature_extraction.text import CountVectorizer

# alternative distance metrics for multidimensional scaling
from sklearn.metrics import euclidean_distances
from sklearn.metrics.pairwise import linear_kernel as cosine_distances
from sklearn.metrics.pairwise import manhattan_distances as manhattan_distances

from sklearn import manifold # multidimensional scaling
from sklearn.cluster import KMeans # cluster analysis by partitioning
from sklearn.cluster import MeanShift # cluster analysis by partitioning
from sklearn.decomposition import PCA # principal component analysis

# define list of codes to be dropped from documents
# carriage-returns (\r), line-feeds (\n), tabs (\t)
codelist = ['\r', '\n', '\t']

# ==============================================================================
# DATA PREPARATION - Remove words that do not add context such as stop words
#
# Corpus of stopwords are from NTLK http://www.nltk.org/book/ch02.html
# These words will be removed after the data is processed
# ==============================================================================
english_stopwords = ['i',
'me',
'my',
'myself',
'we',
'our',
'ours',
'ourselves',
'you',
'your',
'yours',
'yourself',
'yourselves',
'he',
'him',
'his',
'himself',
'she',
'her',
'hers',
'herself',
'it',
'its',
'itself',
'they',
'them',
'their',
'theirs',
'themselves',
'what',
'which',
'who',
'whom',
'this',
'that',
'these',
'those',
'am',
'is',
'are',
'was',
'were',
'be',
'been',
'being',
'have',
'has',
'had',
'having',
'do',
'does',
'did',
'doing',
'a',
'an',
'the',
'and',
'but',
'if',
'or',
'because',
'as',
'until',
'while',
'of',
'at',
'by',
'for',
'with',
'about',
'against',
'between',
'into',
'through',
'during',
'before',
'after',
'above',
'below',
'to',
'from',
'up',
'down',
'in',
'out',
'on',
'off',
'over',
'under',
'again',
'further',
'then',
'once',
'here',
'there',
'when',
'where',
'why',
'how',
'all',
'any',
'both',
'each',
'few',
'more',
'most',
'other',
'some',
'such',
'no',
'nor',
'not',
'only',
'own',
'same',
'so',
'than',
'too',
'very',
's',
't',
'can',
'will',
'just',
'don',
'should',
'now']

# contractions and other word strings to drop from further analysis, adding
# to the usual English stopwords to be dropped from the document collection
more_stop_words = ['cant','didnt','doesnt','dont','goes','isnt','hes',\
'shes','thats','theres','theyre','wont','youll','youre','youve',\
're','tv','g','us','en','ve','vg','didn','pg','gp','our','we',
'll','film','video','name','years','days','one','two','three',\
'four','five','six','seven','eight','nine','ten','eleven','twelve', 'got',
'could']
# start with the initial list and add to it for movie text work
stoplist = english_stopwords + more_stop_words

# =============================================================================
# text parsing function for creating text documents
# =============================================================================
def text_parse(string):
# replace non-alphanumeric with space
# ^ matches the start of the string
# a-z matches any lower case letter, A-Z any upper case letter
temp_string = re.sub('[^a-zA-Z]', ' ', string)
# replace codes with space
for i in range(len(codelist)):
```

```
stopstring = ' ' + codelist[i] + ' '
temp_string = re.sub(stopstring, ' ', temp_string)
# replace single-character words with space
temp_string = re.sub('\s.\s', ' ', temp_string)
# convert uppercase to lowercase
temp_string = temp_string.lower()
# replace selected character strings/stop-words with space
for i in range(len(stoplist)):
stopstring = ' ' + str(stoplist[i]) + ' '
temp_string = re.sub(stopstring, ' ', temp_string)
# replace multiple blank characters with one blank character
temp_string = re.sub('\s+', ' ', temp_string)
return(temp_string)

# =======================================================================
# DATA INPUT - read in CSV data file of parsed movie taglines

# read in the comma-delimited text file to create the movies data frame for analysis
# 5 columns Pandas DataFrame
# movie - movie name
# year - year movie was published
# title - movie title
# tagline - movie tagline
# text - long description
# =======================================================================
movies = pd.read_csv('movie_tagline_data_parsed.csv')

# =======================================================================
# EDA - plot frequency of movies by year... histogram
# =======================================================================

plt.figure()
plt.hist(movies['year'], bins= 100)
plt.xlabel('Year')
plt.ylabel('Number of Movies in Database')
plt.show()
plt.savefig('fig_text_movies_by_year_histogram.pdf',
bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
orientation='landscape', papertype=None, format=None,
transparent=True, pad_inches=0.25, frameon=None)

# =======================================================================
# DATA TRANSFORMATION

# we work with movies from 1974 to 2013
# create aggregate tagline_text collection for each year of interest
# =======================================================================
year = [] # initialize year list
```

```
tagline_text = [] # initialize aggregate tagline text
parsed_text = [] # parsed tagline text for subsequent analysis
aggregate_document = '' # intialize aggregate taglines document

for iyear in range(1974,2014):
year.append(iyear)
gather_taglines = '' # initialize aggregate tagline text
this_year_data_frame = movies[movies['year'] == iyear]
for this_record_index in this_year_data_frame.index.values:
this_record = this_year_data_frame.ix[this_record_index]
gather_taglines = gather_taglines + this_record['tagline']
tagline_text.append(gather_taglines)
parsed_text.append(text_parse(gather_taglines))
aggregate_document = aggregate_document + gather_taglines
big_bag_of_words = text_parse(aggregate_document)

# ===============================================================================
# DATA TRANSFORMATION - "Stemming"

# This will attempt to find "word stems" by dropping suffixes
# This will reduce the word/document matrix for words that have same stem
# Uses Porter algorithm in NLTK
# http://www.nltk.org/api/nltk.stem.html?highlight=porter#module-nltk.stem.porter
# =================================================================================

stemmer = nltk.PorterStemmer() #instantiate stemming function
# the stemmer function evaulates a single word to determine the word stem
# stem_parsed_text is a list of word stems
stem_parsed_text = [[stemmer.stem(word)
for word in sentence.split(" ")] # iterate through words in sentence
for sentence in parsed_text] # iterate through sentences of text doc
# join all of the word stems back together into a text string
stemmed = [" ".join(sentence) for sentence in stem_parsed_text]

# ================================================================================
# DATA TRANSFORMATION - Create Document Collection
# 40 years of data = 40 documents
# ================================================================================
tagline_data = {'year': year, 'tagline_text':tagline_text,\
'stemmed':parsed_text}
# 'parsed_text':parsed_text}
tagline_data_frame = pd.DataFrame(tagline_data)

# ================================================================================
# DATA TRANSFORMATION - Create Word x Document matrix

# We need to create a vector of word counts by document for analysis
# create terms-by-documents matrix from the parsed text
# extracting the top 200 words in the tagline corpus
# binary = True sets all non-zero counts to 1.
# CountVectorizer creates tokens and occurence counting
# http://scikit-learn.org/stable/modules/feature_extraction.html#common-vectorizer-usage
# ===============================================================================

tdm_method = CountVectorizer(max_features = 200, binary = True)
#examine_movies_tdm = tdm_method.fit(parsed_text)
examine_movies_tdm = tdm_method.fit(stemmed)
top_words = examine_movies_tdm.get_feature_names()

# get clean printing of the top words
print('\nTop 200 words in movie taglines database\n')
print(map(lambda t: t.encode('ascii'), top_words)) # print sans unicode

# ===============================================================================
# DATA TRANSFORMATION
# extract the terms-by-documents matrix
# in scipy compressed sparse column format
# ===============================================================================
sparse_movies_tdm = tdm_method.fit_transform(stemmed)
# convert sparse matrix into regular terms-by-documents matrix
movies_tdm = sparse_movies_tdm.todense()
# define the documents-by-terms matrix
movies_dtm = movies_tdm.transpose()
# dissimilarity measures and multidimensional scaling
# consider alternative pairwise distance metrics from sklearn modules
# euclidean_distances, cosine_distances, manhattan_distances (city-block)
# note that different metrics provide different solutions
# movies_distance_matrix = euclidean_distances(movies_tdm)
# movies_distance_matrix = manhattan_distances(movies_tdm)

movies_distance_matrix = cosine_distances(movies_tdm)
#movies_distance_matrix = euclidean_distances(movies_tdm)

mds_method = manifold.MDS(n_components = 2, random_state = 9999,\
dissimilarity = 'precomputed')
mds_fit = mds_method.fit(movies_distance_matrix)
mds_coordinates = mds_method.fit_transform(movies_distance_matrix)

# plot tagline text for years in two dimensions
# defined by multidimensional scaling
plt.figure()
plt.scatter(mds_coordinates[:,0],mds_coordinates[:,1],\
facecolors = 'none', edgecolors = 'none') # plots points in white (invisible)
labels = []
for iyear in range(1974,2014):
labels.append(str(iyear))
```

```
for label, x, y in zip(labels, mds_coordinates[:,0], mds_coordinates[:,1]):
plt.annotate(label, (x,y), xycoords = 'data')
plt.xlabel('First Dimension')
plt.ylabel('Second Dimension')
plt.show()
plt.savefig('fig_text_mds_1974_2013.pdf',
bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
orientation='landscape', papertype=None, format=None,
transparent=True, pad_inches=0.25, frameon=None)

# =====================================================================
# CLUSTER ANALYSIS - K-MEANS
# K-Means requires number of clusters to be predefined
# classification of words into groups for further analysis
# use transpose of the terms-by-document matrix and cluster analysis
# try five clusters/groups of words
# =====================================================================

# instantiate KMeans clustering algorithm
clustering_method = KMeans(n_clusters = 5, random_state = 9999)
#clustering_method = MeanShift(cluster_all = False)

# Compute k-means clustering
clustering_solution = clustering_method.fit(movies_dtm)

# Compute cluster centers and predict cluster index for each sample
cluster_membership = clustering_method.predict(movies_dtm)

# Transform movies to a cluster-distance space
word_distance_to_center = clustering_method.transform(movies_dtm)

# top words data frame for reporting k-means clustering results
top_words_data = {'word': top_words, 'cluster': cluster_membership,\
'dist_to_0': word_distance_to_center[0:,0],\
'dist_to_1': word_distance_to_center[0:,1],\
'dist_to_2': word_distance_to_center[0:,2],\
'dist_to_3': word_distance_to_center[0:,3],\
'dist_to_4': word_distance_to_center[0:,4]}

distance_name_list = ['dist_to_0','dist_to_1','dist_to_2','dist_to_3','dist_to_4']

top_words_data_frame = pd.DataFrame(top_words_data)

for cluster in range(5):
words_in_cluster =\
top_words_data_frame[top_words_data_frame['cluster'] == cluster]
sorted_data_frame =\
top_words_data_frame.sort_index(by = distance_name_list[cluster],\
ascending = True)
print('\n Top Words in Cluster :',cluster,'------------------------------')
print(sorted_data_frame.head(10))
# could examine possible clustering solutions with partitioning
# by changing the n_clusters setting for KMeans

# a five-cluster solution seems to make sense with words
# toward the center of each cluster fitting together
# let's use pairs of top words from each cluster to name the clusters
# cluster index 0: got-men
# cluster index 1: make-love
# cluster index 2: family-fight
# cluster index 3: big-journey
# cluster index 4: human-hero

# name the clusters in the top words data frame
cluster_to_name = {0:'got-men',\
1:'make-love', 2:'family-fight',\
3:'big-journey', 4:'human-hero'}
top_words_data_frame['cluster_name'] = top_words_data_frame['cluster'].map(cluster_to_name)
print(top_words_data_frame.head())

# use word clusters to define text measures...
# in particular, let the raw score for a cluster for a year be the percentage
# of words in that year's tagline documents that fall within the cluster
# then to examine movies in time, standardize cluster scores across the
# forty years of the study and plot as a multiple time series
got_men_df =\
top_words_data_frame[top_words_data_frame['cluster'] == 0]
got_men_word_list = str(got_men_df['word'])

make_love_df =\
top_words_data_frame[top_words_data_frame['cluster'] == 1]
make_love_word_list = str(make_love_df['word'])

family_fight_df =\
top_words_data_frame[top_words_data_frame['cluster'] == 2]
family_fight_word_list = str(family_fight_df['word'])

big_journey_df =\
top_words_data_frame[top_words_data_frame['cluster'] == 3]
big_journey_word_list = str(big_journey_df['word'])

human_hero_df =\
top_words_data_frame[top_words_data_frame['cluster'] == 4]
human_hero_word_list = str(human_hero_df['word'])

# cluster scores as percentage of total words
```

```
def cluster_scoring(cluster_count, total_count):
return (100 * (cluster_count/total_count))
# initialize word counts
got_men_words = []; make_love_words = []; family_fight_words = [];
big_journey_words = []; human_hero_words = []; total_words = []
# initialize cluster scores
got_men = []; make_love = []; family_fight = [];
big_journey = []; human_hero = []
# compute text measures for each year
for iyear in range(len(year)):
text = stemmed[iyear] # this year's text for scoring
total_count = len([w for w in text.split()])
total_words.append(total_count)
got_men_count =\
len([w for w in text.split() if w in got_men_word_list])
got_men_words.append(got_men_count)
got_men.append(cluster_scoring(got_men_count, total_count))
make_love_count =\
len([w for w in text.split() if w in make_love_word_list])
make_love_words.append(make_love_count)
make_love.append(cluster_scoring(make_love_count, total_count))
family_fight_count =\
len([w for w in text.split() if w in family_fight_word_list])
family_fight_words.append(family_fight_count)
family_fight.append(cluster_scoring(family_fight_count, total_count))
big_journey_count =\
len([w for w in text.split() if w in big_journey_word_list])
big_journey_words.append(big_journey_count)
big_journey.append\
(cluster_scoring(big_journey_count, total_count))
human_hero_count =\
len([w for w in text.split() if w in human_hero_word_list])
human_hero_words.append(human_hero_count)
human_hero.append(cluster_scoring(human_hero_count, total_count))

add_cluster_data = {'total_words':total_words,\
'got_men_words':got_men_words,\
'got_men':got_men,\
'make_love_words':make_love_words,\
'make_love':make_love,\
'family_fight_words':family_fight_words,\
'family_fight':family_fight,\
'big_journey_words':big_journey_words,\
'big_journey':big_journey,\
'human_hero_words':human_hero_words,\
'human_hero':human_hero}
add_cluster_data_frame = pd.DataFrame(add_cluster_data)
tagline_data_frame =\
pd.concat([tagline_data_frame,add_cluster_data_frame],axis=1)

# check text measure calculations
print(tagline_data_frame.describe())
print(tagline_data_frame.head())
print(tagline_data_frame.tail())

# compute text measure standard scores across years
tagline_data_frame['z_got_men'] =\
tagline_data_frame['got_men'].\
apply(lambda d: (d - tagline_data_frame['got_men'].mean())/\
tagline_data_frame['got_men'].std())
tagline_data_frame['z_make_love'] =\
tagline_data_frame['make_love'].\
apply(lambda d: (d - tagline_data_frame['make_love'].mean())/\
tagline_data_frame['make_love'].std())
tagline_data_frame['z_family_fight'] =\
tagline_data_frame['family_fight'].\
apply(lambda d: (d - tagline_data_frame['family_fight'].mean())/\
tagline_data_frame['family_fight'].std())

tagline_data_frame['z_big_journey'] =\
tagline_data_frame['big_journey'].\
apply(lambda d: (d - tagline_data_frame['big_journey'].mean())/\
tagline_data_frame['big_journey'].std())
tagline_data_frame['z_human_hero'] =\
tagline_data_frame['human_hero'].\
apply(lambda d: (d - tagline_data_frame['human_hero'].mean())/\
tagline_data_frame['human_hero'].std())
# prepare data frame for multiple time series plot
prelim_mts = pd.DataFrame(tagline_data_frame, columns =\
['year', 'z_got_men', 'z_make_love', 'z_family_fight',\
'z_big_journey', 'z_human_hero'])
prelim_mts.rename(columns = {'z_got_men':'Got-Men',\
'z_make_love':'Make-Love', 'z_family_fight':'Family-Fight',\
'z_big_journey':'Big-Journey',\
'z_human_hero':'Human-Hero'}, inplace = True)
mts = prelim_mts.set_index('year')

# generate the plot
mts.plot()
plt.xlabel('')
plt.ylabel('Standardized Text Measure')
plt.show()
plt.savefig('fig_text_mts_1974_2013.pdf',
bbox_inches = 'tight', dpi=None,
orientation='portrait', papertype=None, format=None,
transparent=True, pad_inches=0.25, frameon=None)
```

```
# Suggestions for the student:
# Try word stemming prior to the definition of a
# terms-by-documents matrix. Try longer lists of words
# for the identified clusters. See if there are ways to utilize
# information from wordnet to guide further analyses.
# Text features within text classification problems may be defined
# on term document frequency alone or on measures of term
# document frequency adjusted by term corpus frequency.
# Try alternative feature selection and text measures, and
# try alternative cluster analysis methods. See if you can
# use methods of latent semantic analysis to identify
# themes or common topics across the corpus.
```