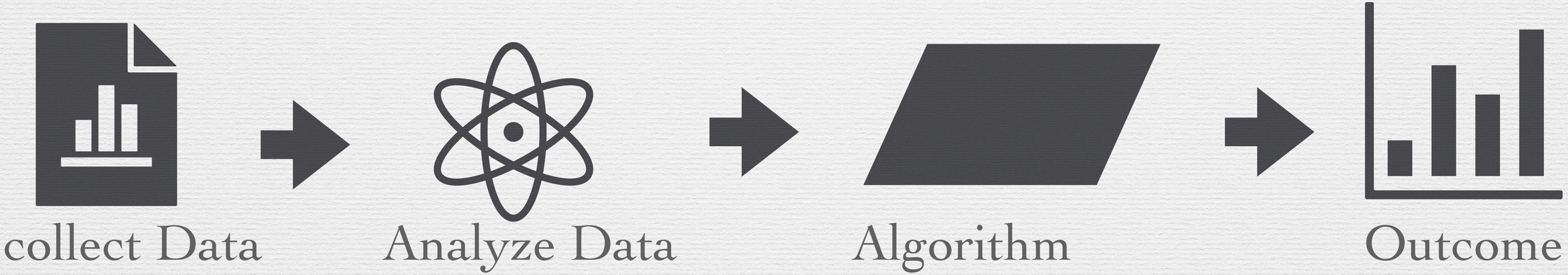
SENTIMENT ANALYSIS

Sentiment analysis

Discerption: contains sentences labelled with positive or negative sentiment; sentence come from three different websites/filed;
imdb.com; amazon.com ; yelp.com no neutral sentences to be selected.

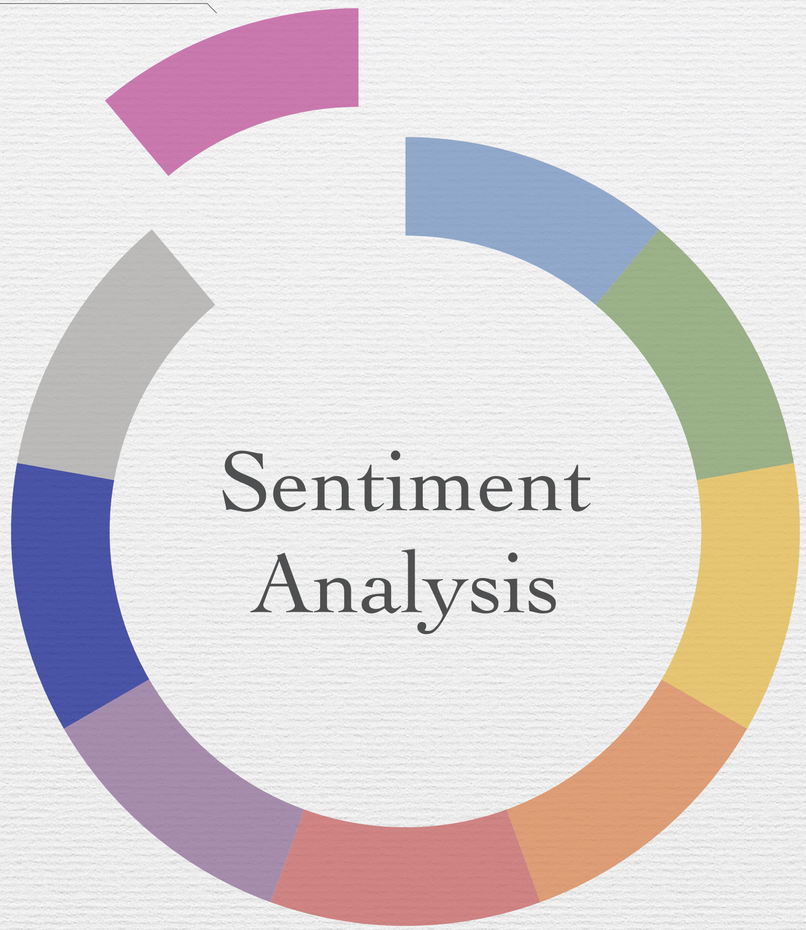
Task: Classify one sentences to be positive (Happy) or negative (Unhappy).

Sentiment analysis



Amazon
Twitter
IMBD

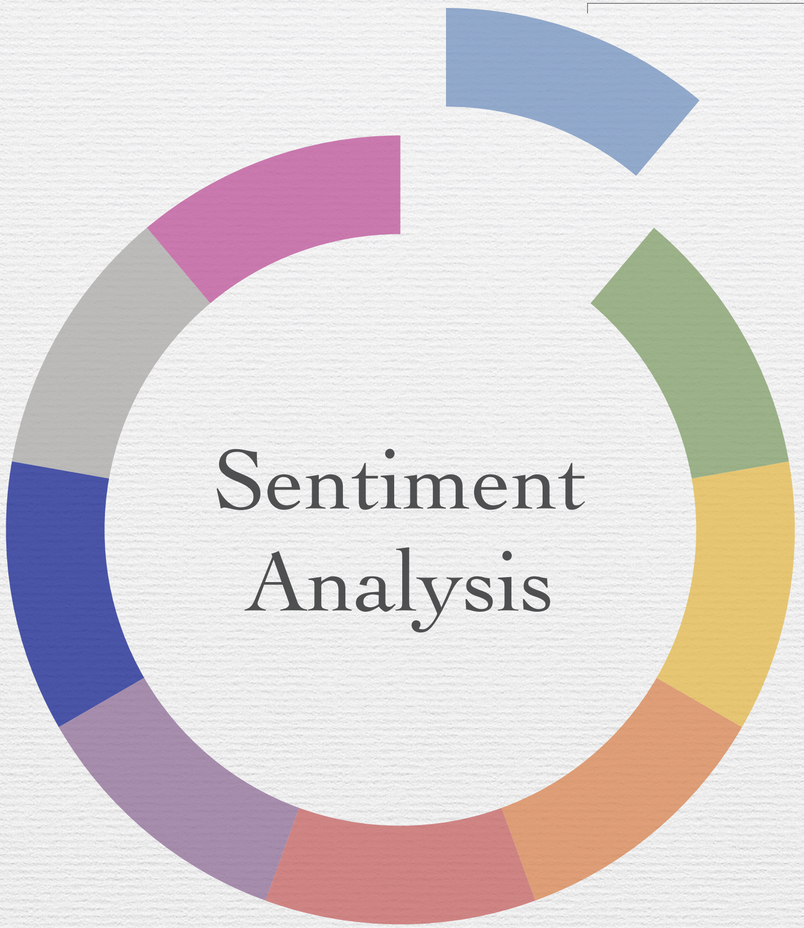
Libraries



Libraries

```
import collections
import glob
import mglearn                                                  # To use the visualizing tool
from networkx.drawing.tests.test_pylab import plt               # to visualize the cross-valid
from sklearn.model_selection import train_test_split            # To split our data
import csv                                                    # To read and split comma separated files or other delimiter
import numpy as np                                              # To use numpy arrays
from sklearn.feature_extraction.text import CountVectorizer     # To create bag of words
from sklearn.feature_extraction.text import TfidfVectorizer     # To use term frequency-inverse doc. freq. method
from sklearn.pipeline import make_pipeline                      # To use with tf-idf to ensure results are valid
from sklearn.model_selection import cross_val_score             # To get a cross valid score
from sklearn.linear_model import LogisticRegression             # To train our data
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
import spacy
import nltk
import re                                                       # use regexp-based tokenization
from sklearn.model_selection import StratifiedShuffleSplit      # build a grid search using only 1% of the data as the training
from sklearn.decomposition import LatentDirichletAllocation
from google.colab import drive
```

Data Processing



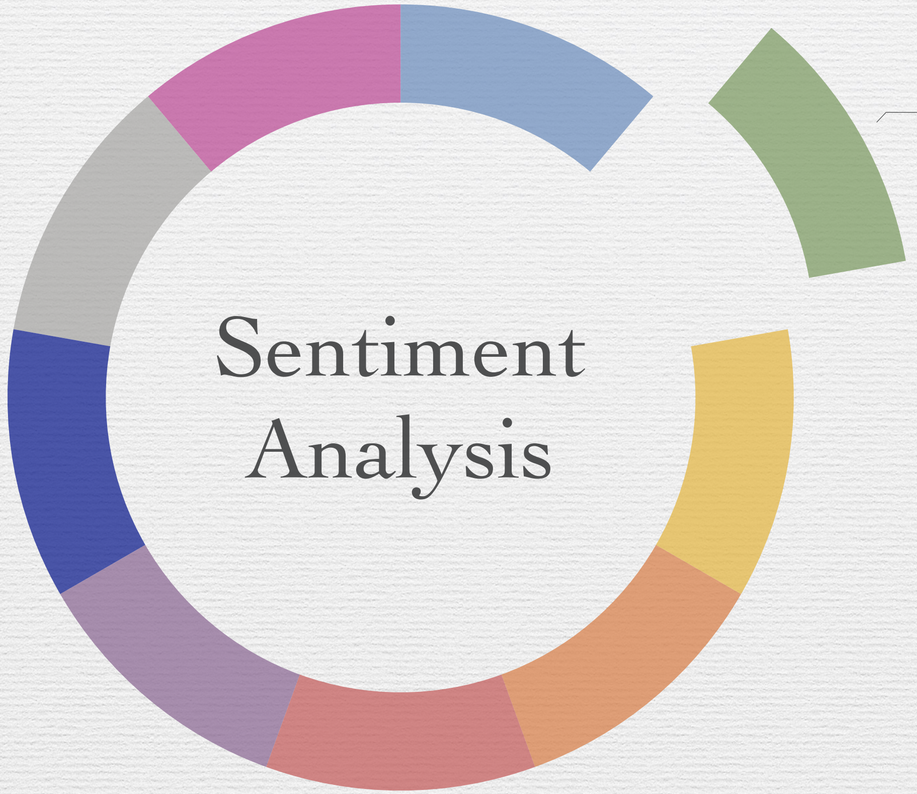
Data Processing

- Data from text files that contain reviews and the sentimental score
- Return: An input list of reviews and an output of an np array of the sentimental value (0 or 1)

```python
def process_data():
    # Get the files used for learning and testing. Should be three files:
    # amazon_cells_labelled.txt, yelp_labelled.txt and imdb_labelled.txt
    data_files = glob.glob("/content/gdrive/My Drive/Sentiment Analysis Data/*.txt")
    # populate input list and output list with data, separating sentences from the scores
    input_list = []   # Input list containing review sentences
    output_list = []  # Output list containing sentiment values of 0 or 1: 0 negative or 1 positive
    # Go through all the files and populate the input data along with its sentiment value in the output list
    for data_file in data_files:
        # For all text files we read one at a time
        with open(data_file, 'r') as file:
            # Read file and seperate review sentences from sentiment value
            text = csv.reader(file, delimiter='\t')  # since our files are separated by a tab space
            for line in text:
                # For each line we will get a sentence and a sentiment value of 0(negative) and 1(positive)
                sentence = line[0]                      # Get review sentence
                # The following 3 re.sub gets rid of any tabs and newlines for better extraction
                sentence = re.sub('\\t0', '', sentence)
                sentence = re.sub('\\t1', '', sentence)
                sentence = re.sub('\n', '', sentence)
                # Add sentence review and sentiment value to our lists
                input_list.append(sentence)
                output_list.append(int(line[1]))

    # Make our output an np array to be compatible with sklearn functions
    output_array = np.array(output_list)
    return input_list, output_array
```

Data: https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences#



Split Data

Spilt Data

Spilt Data: Spilt arrays or metrics into random train and test subsets

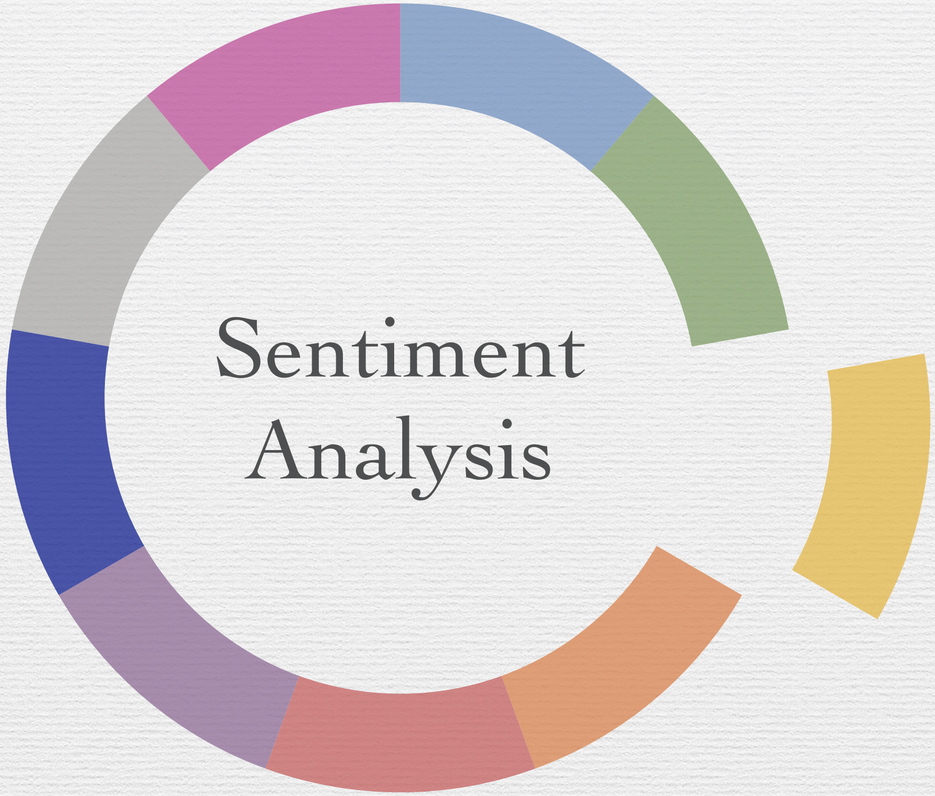
```
def split_data(X, y):

    # Constants for our arguments passed in sklearn.train_test_split
    SPLIT_FRACTION = 0.33 # The fraction size of our data
    RANDOM_STATE = 42     # The constant random state we will use to shuffle the data

    #Call sklearn.train_test_split to split our data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=SPLIT_FRACTION , random_state=42)

    return X_train, X_test, y_train, y_test
```

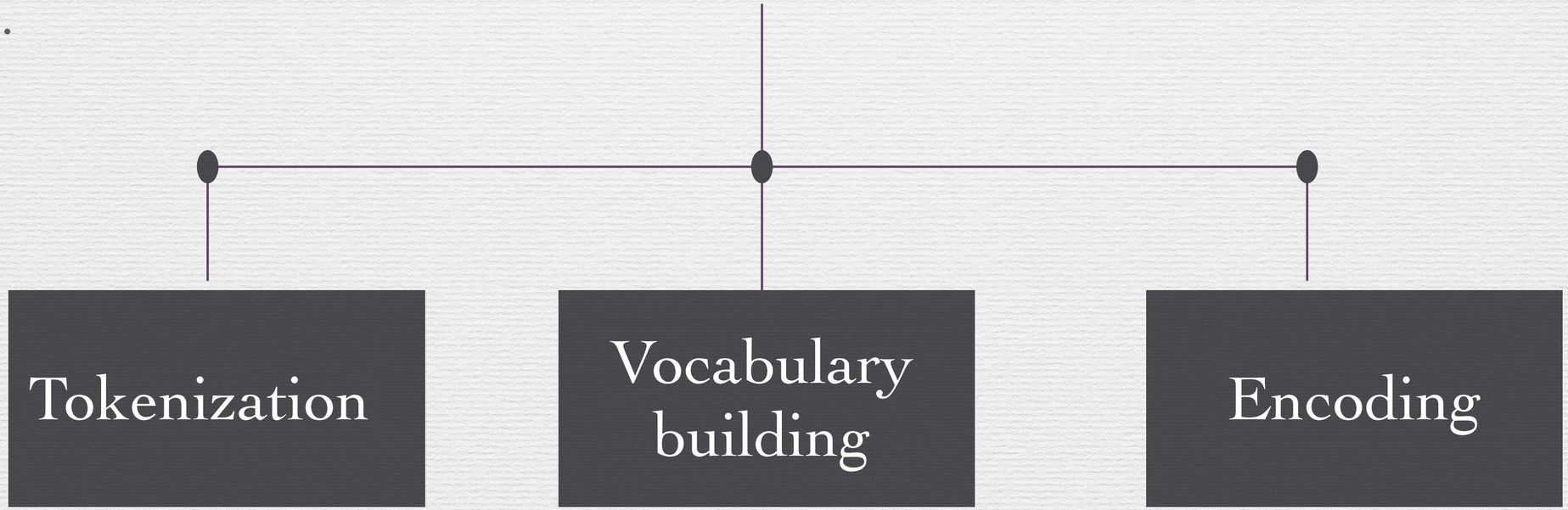
- X: The input list sentences of a review
- y: The output list/array of sentiment value based on a review



bag of words

Bag of Word

- Simple but effective and commonly used ways to represent text for machine learning.



Spilt each document into

the word

collect a vocabulary of all the words that appear in the documents

Count how often each of the word in the vocabulary appear in

document

Bag of Word

```
def get_bag_of_words(input_text_list):

    vect = CountVectorizer().fit(input_text_list)       # Create an instance of CountVectorizer
    transform_text_list = vect.transform(input_text_list)    # Transform the text list

    return transform_text_list
```

- The input_text_list that will have list bag_of_word

Data transformation

Function to compare lemmatization in Spacy with stemming in ntlk

```
# define function to compare lemmatization in spacy with stemming in nltk
def compare_normalization(doc):
    # tokenize document in spacy
    doc_spacy = en_nlp(doc)
    # print lemmas found by spacy
    print("Lemmatization:")
    print([token.lemma_ for token in doc_spacy])
    # print tokens found by Porter stemmer
    print("Stemming:")
    print([stemmer.stem(token.norm_.lower()) for token in doc_spacy])
```

Create a custom tokenizer using the spacy document processing pipeline

```
# regexp used in CountVectorizer
regexp = re.compile('(?u)\\b\\w\\w+\\b')
# load spacy language model
en_nlp = spacy.load('en', disable=['parser', 'ner'])
old_tokenizer = en_nlp.tokenizer
# replace the tokenizer with the preceding regexp
en_nlp.tokenizer = lambda string: old_tokenizer.tokens_from_list(
    regexp.findall(string))
# create a custom tokenizer using the spacy document processing pipeline
# (now using our own tokenizer)
def custom_tokenizer(document):
    doc_spacy = en_nlp(document)
    return [token.lemma_ for token in doc_spacy]
# define a count vectorizer with the custom tokenizer
lemma_vect = CountVectorizer(tokenizer=custom_tokenizer, min_df=5)
```

Data processing :

```
if __name__ == '__main__':
  # Process our data
  input_list, output_list = process_data()
```

Data splitting:

```
# Split our data
X_train, X_test, y_train, y_test = split_data(input_list, output_list)
```

Data visualization:

```
print('Number of sentences in our data: {}'.format(len(input_list)))
print('Samples per class...')
print('\tTraning set: ', len(X_train) , ',', len(y_train))
print('\tTest set: ', len(X_test),',', len(y_test))
print('\nTraining sentence of example 1:', X_train[0])
print('Training sentiment value of example 1:', y_train[0])
print('\nTraining sentence of example 2:', X_train[1])
print('Training sentiment value of example 2:', y_train[1])
```

```
Number of sentences in our data: 2748
Samples per class...
        Traning set:  1841 , 1841
        Test set:  907 , 907

Training sentence of example 1: This place is like Chipotle, but BETTER.
Training sentiment value of example 1: 1

Training sentence of example 2: This movie contained an all-star cast, and what I lov
Training sentiment value of example 2: 1
```

Bag of Word

Bag-of-Words Representation in our Sentiment Analysis Data

```
sent_bag_of_words = [ X_train[0] , X_train[1]]  # Since it take a whole list of sente
vect =  CountVectorizer()                       # Creates a vector class which we will pass c
vect.fit(sent_bag_of_words)                     # Vectorizes the words
sent_bag_of_words = vect.transform(sent_bag_of_words) # Transform representaion of ba
print('Vocabulary Size for X_train[0] and X_train[1]: {}'.format(len(vect.vocabulary_
print('\nContents in X_train[0] and X_train[1] along the index represented in the 2D
print(vect.vocabulary_)
print('\nRepresentation in bag of words:\n', sent_bag_of_words.toarray())
```

```
Vocabulary Size for X_train[0] and X_train[1]: 32

Contents in X_train[0] and X_train[1] along the index represented in the 2D array.
{'this': 27, 'place': 21, 'is': 12, 'like': 14, 'chipotle': 8, 'but': 6, 'better': 5,

Representation in bag of words:
 [[0 0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0]
 [1 1 1 1 1 0 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1]]
```

Training and Testing

```
# Set our input data into bag of words for feature extraction
transformed_X_train = get_bag_of_words(X_train) # Transform our input training data
clf = LogisticRegression() # Create class instance to train our data
scores = cross_val_score(LogisticRegression(), transformed_X_train, y_train, cv=5)
print('Mean cross-validation accuracy using bag of words {:.2f}'.format(np.mean(scor
```

```
Mean cross-validation accuracy using bag of words 0.80
```

Bag of Word

```
# In[12]
print('In[12]')
vect = CountVectorizer().fit(X_train)
X_train_transformed = vect.transform(X_train)
print('X_train:\n{}'.format(repr(X_train_transformed)))
# In[13]
print('In[13]')
feature_names = vect.get_feature_names()
print('Number of features: {}'.format(len(feature_names)))
print('First 20 features:\n{}'.format(feature_names[:20]))
print('Features 1000 to 1030:\n{}'.format((feature_names[1000:1030])))
print('Every 500th feature:\n{}'.format(feature_names[::500]))
```

Output:

```
In[12]
X_train:
<1841x4185 sparse matrix of type '<class 'numpy.int64'>'
        with 20840 stored elements in Compressed Sparse Row format>
In[13]
Number of features: 4185
First 20 features:
['00', '10', '100', '11', '12', '13', '15', '15g', '15pm', '17', '18th', '1948', '1949', '19
Features 1000 to 1030:
['deserved', 'deserves', 'deserving', 'design', 'designed', 'designer', 'desired', 'desperat
Every 500th feature:
['00', 'buffets', 'deserved', 'forgeries', 'journey', 'nutshell', 'reminded', 'strip', 'waaa
```

Bag of Word

```
# In[14]
print('In[14]')
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train_transformed, y_train)
print('Best cross-validation score: {:.2f}'.format(grid.best_score_))
print('Best parameters: ', grid.best_params_)
```

Output:

```
Best cross-validation score: 0.81
Best parameters:  {'C': 10}
```

```
# In[16]
print('In[16]')
X_test_transformed = vect.transform(X_test)
print('Test score: {:.2f}'.format(grid.score(X_test_transformed, y_test)))
# In[17]
print('In[17]')
vect = CountVectorizer(min_df=5).fit(X_train)
X_train_transformed = vect.transform(X_train)
print('X_train with min_dif: {}'.format(repr(X_train_transformed)))
```

Output:

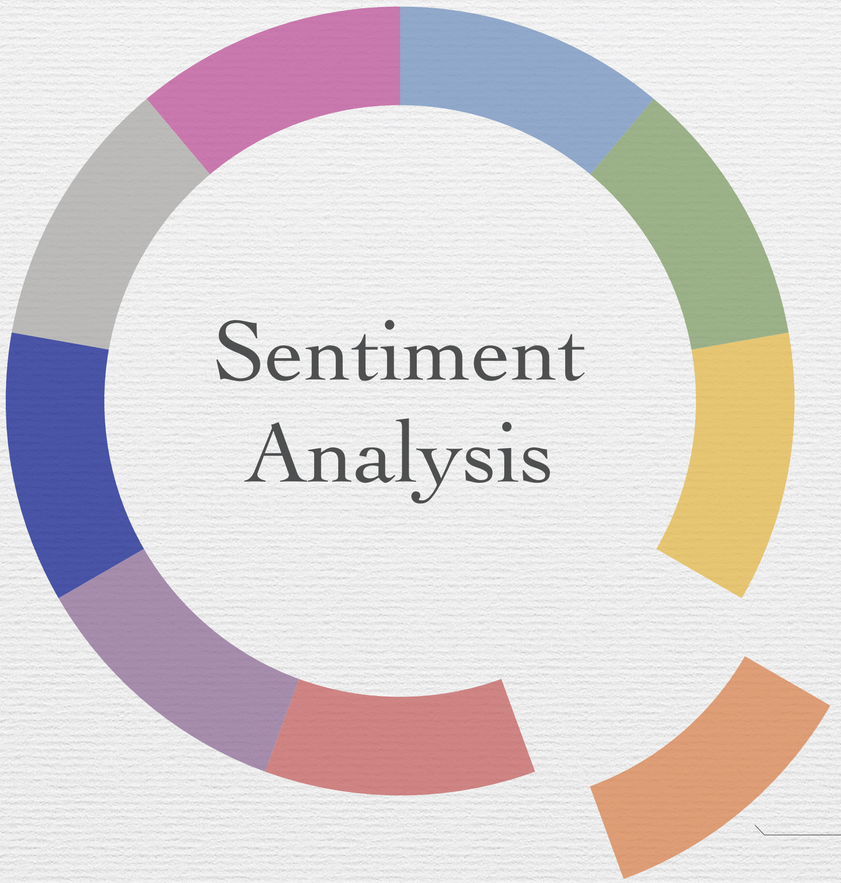
```
In[16]
Test score: 0.83
In[17]
X_train with min_dif: <1841x648 sparse matrix of type '<class 'numpy.int64'>'
        with 15654 stored elements in Compressed Sparse Row format>
```

Bag of Word

```
# In[18]
print('In[18]')
feature_names = vect.get_feature_names()
print('First 50 features:\n{}'.format(feature_names[:50]))
print('Features 900 to 1000:\n{}'.format(feature_names[900:1000]))
print('Every 70th feature:\n{}'.format(feature_names[::70]))
# In[19]
print('In[19]')
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train_transformed, y_train)
print('Best cross-validation score: {:.2f}'.format(grid.best_score_))
```

Output:

```
In[18]
First 50 features:
['10', '20', '90', 'ability', 'about', 'above', 'absolutely', 'acting', 'action', 'actor', '
Features 900 to 1000:
[]
Every 70th feature:
['10', 'book', 'device', 'fish', 'imagination', 'most', 'potato', 'since', 'top', 'word']
In[19]
Best cross-validation score: 0.77
```



Stopwords

StopWords

Stop word is another way to get rid of uninformative words is discarding words that are too frequent to be informative

There are two main approach

Using a language - specific list of stopWords

Discarding words that appear frequently

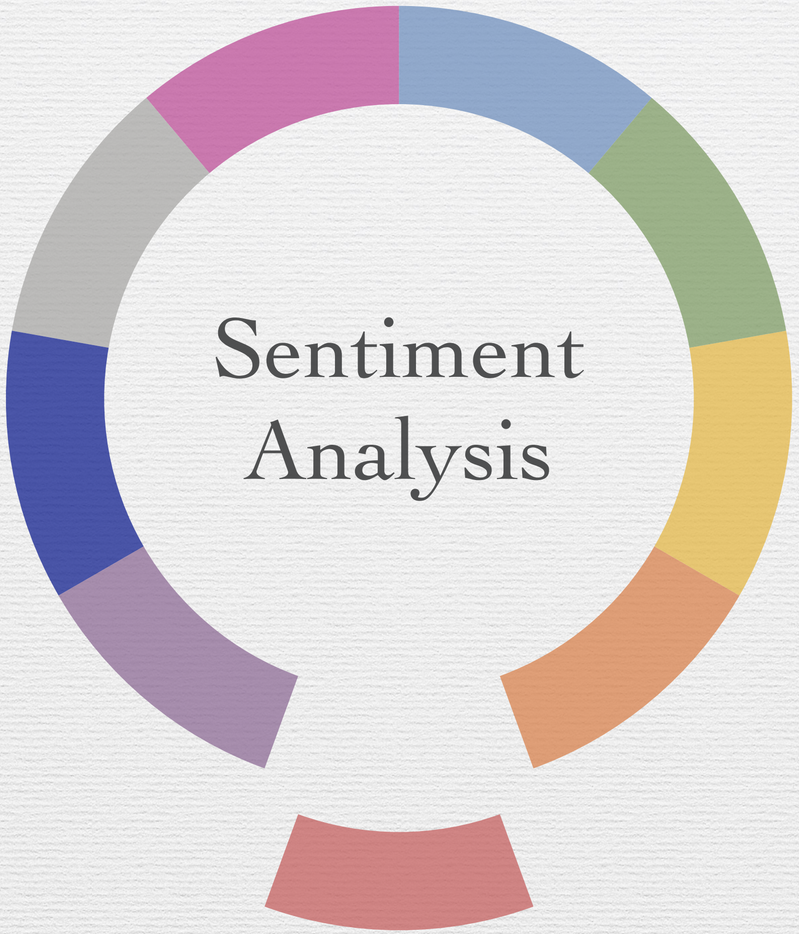
StopWords

Scikit-learn has a built-in list of English StopWords in the feature_extraction.text

```
# In[20]
print('In[20]')
print('Number of stop words: {}'.format(len(ENGLISH_STOP_WORDS)))
print('Every 10th stopword:\n{}'.format(list(ENGLISH_STOP_WORDS)[::10]))
# In[21]
print('In[21]')
vect = CountVectorizer(min_df=5, stop_words='english').fit(X_train)
X_train_transformed = vect.transform(X_train)
print('X_train with stop words:\n{}'.format(repr(X_train_transformed)))
# In[22]
print('In[22]')
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train_transformed, y_train)
print('Best cross-validation score: {:.2f}'.format(grid.best_score_))
```

Output:

```
Number of stop words: 318
Every 10th stopword:
['both', 'any', 'everyone', 'yourselves', 'move', 'twelve', 'meanwhile', 'sometime', 'yet',
In[21]
X_train with stop words:
<1841x471 sparse matrix of type '<class 'numpy.int64'>'
        with 5942 stored elements in Compressed Sparse Row format>
In[22]
Best cross-validation score: 0.74
```



Data with tf–idf

Data with tf–idf

The intuition of this method is to give high weight to any term that appears often in a particular document.

$$\text{tfidf}(w, d) = \text{tf} * \log\left(\frac{N+1}{N_w+1}\right) + 1$$

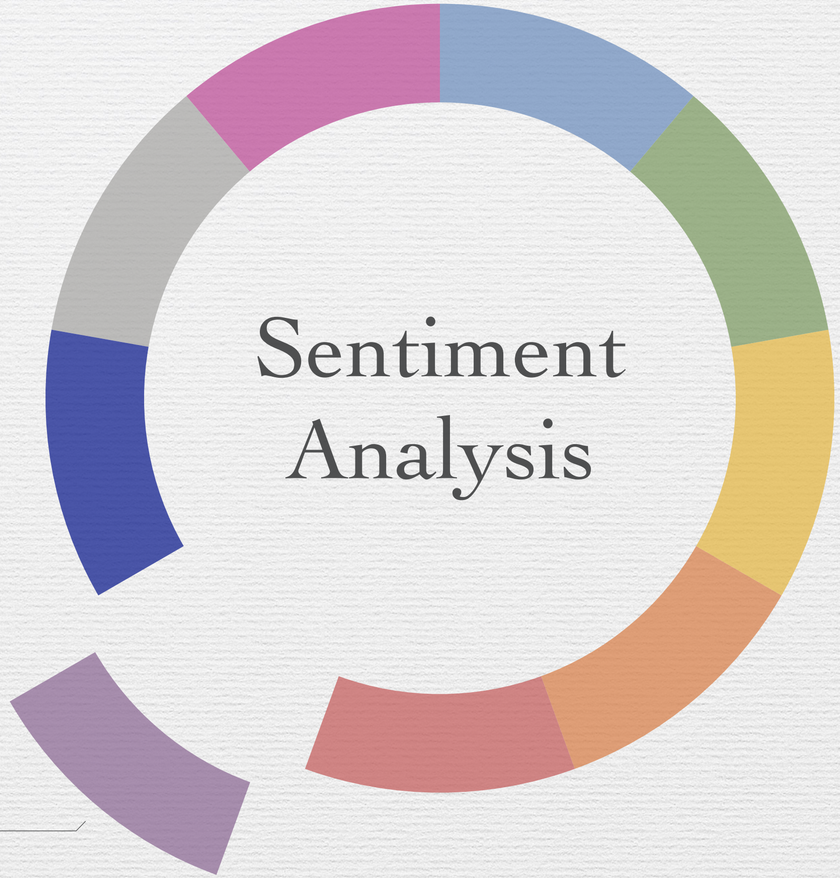
N : is number of document in the training set.
Nw: is the number of document in training set that the word w appears in.
tf: (the term frequency) is number of times that the word w appears in the query document d.

**each data point, represented as a single text, is called a *document*

Data with tf–idf

```python
# In[23]
print('In[23]')
pipe = make_pipeline(TfidfVectorizer(min_df=5), LogisticRegression())
param_grid = {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)
print('Best cross-validation score: {:.2f}'.format(grid.best_score_))
# In[24]
print('In[24]')
vectorizer = grid.best_estimator_.named_steps['tfidfvectorizer']
# transform the training dataset
X_train_transformed = vectorizer.transform(X_train)
# find maximum value for each of the features over the dataset
max_value = X_train_transformed.max(axis=0).toarray().ravel()
sorted_by_tfidf = max_value.argsort()
# get feature names
feature_names = np.array(vectorizer.get_feature_names())
print('Features with lowest tfidf:\n{}'.format(feature_names[sorted_by_tfidf[:20]]))
print('Features with highest tfidf: \n{}'.format(feature_names[sorted_by_tfidf[-20:]]))
# In[25]
print('In[25]')
sorted_by_idf = np.argsort(vectorizer.idf_)
print('Features with lowest idf:\n{}'.format(feature_names[sorted_by_tfidf[:100]]))
```

```
In[23]
Best cross-validation score: 0.77
In[24]
Features with lowest tfidf:
['considering' 'together' 'served' 'face' 'into' 'tell' 'put' 'comes'
 'when' 'took' 'understand' 'dead' 'tv' 'unless' 'years' 'other' 'told'
 'line' 'pay' 'before']
Features with highest tfidf:
['them' 'interesting' 'poorly' 'the' 'up' 'poor' 'is' 'very' 'super'
 'sucks' 'great' 'was' 'design' 'it' 'scene' 'its' 'bad' 'awful' 'good'
 '10']
In[25]
Features with lowest idf:
['considering' 'together' 'served' 'face' 'into' 'tell' 'put' 'comes'
 'when' 'took' 'understand' 'dead' 'tv' 'unless' 'years' 'other' 'told'
 'line' 'pay' 'before' 'cover' 'year' 'top' 'felt' 'certainly' 're'
 'another' 'fact' 'ago' 'feeling' 'couple' 'old' 'book' 'off' 'finally'
 'horror' 'days' 'comedy' 'kind' 'single' 'deal' 'verizon' 'if' 'writing'
 'history' '90' 'sick' 'show' 'perhaps' 'bring' 'child' 'actor'
 'something' 'call' 'running' 'seen' '20' 'hilarious' 'due' 'eaten'
 'characters' 'need' 'first' 'human' 'mess' 'action' 'watched' 'part'
 'suspense' 'totally' 'myself' 'lost' 'half' 'started' 'phones' 'vegas'
 'hear' 'given' 'until' 'things' 'wife' 'guess' 'broke' 'twice' 'waiting'
 'mind' 'throughout' 'inside' 'table' 'watching' 'outside' 'extra'
 'portrayal' 'let' 'care' 'many' 'where' 'especially' 'during' 'or']
```

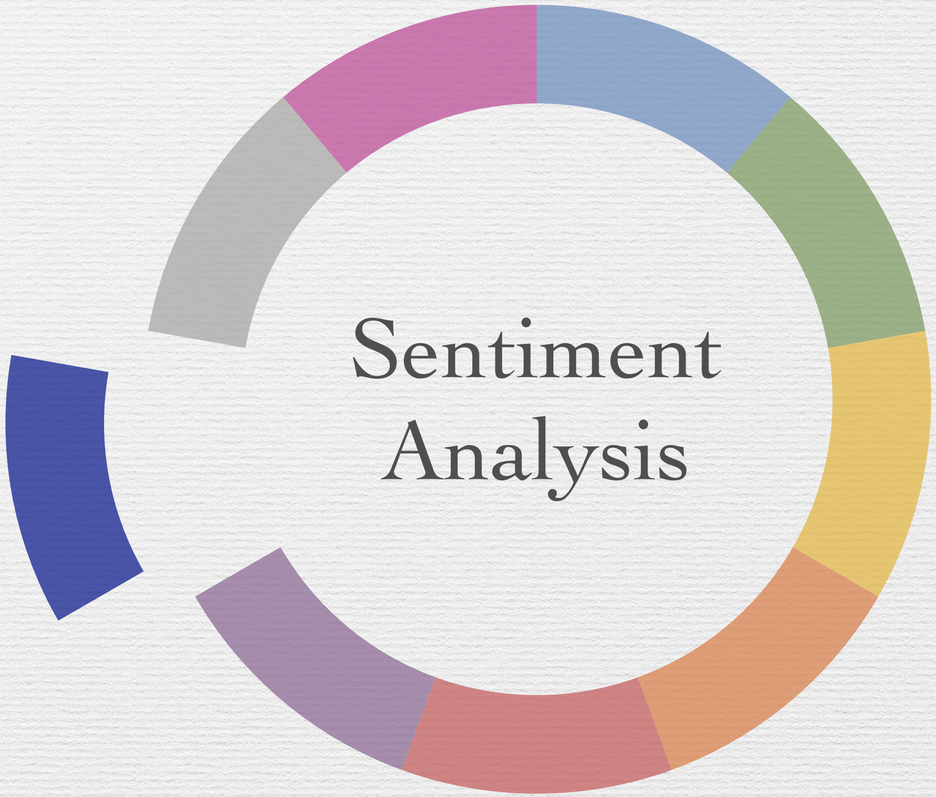


Investigating Model Coefficients

Investigating Model Coefficients

```
# In[26]
print('In[26]')
mglearn.tools.visualize_coefficients(grid.best_estimator_.named_steps['logisticregression'].coef_,
feature_names, n_top_features=40)
```

Bag-of-Words with More than one word(n-Grams)



Bag-of-Words with More than one word(n-Grams)

```
# In[32]
print('In[32]')
pipe = make_pipeline(TfidfVectorizer(min_df=5), LogisticRegression())
# running the grid search takes a long time because of the
# relatively large grid and the inclusion of trigrams
param_grid = {"logisticregression__C": [0.001, 0.01, 0.1, 1, 10, 100],
              "tfidfvectorizer__ngram_range": [(1, 1), (1, 2), (1, 3)]}
grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)
print('Best cross-validation score: {:.2f}'.format(grid.best_score_))
print('Best parameters:\n{}'.format(grid.best_params_))
```

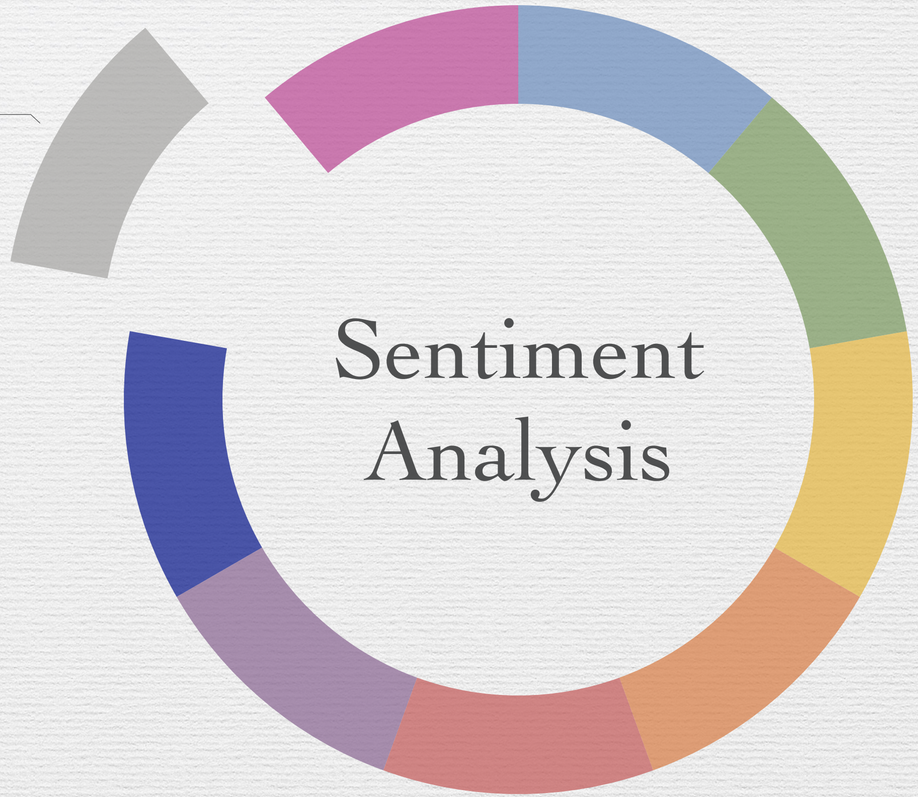
```
In[32]
Best cross-validation score: 0.77
Best parameters:
{'logisticregression__C': 1, 'tfidfvectorizer__ngram_range': (1, 2)}
```

```
# In[33]
print('In[33]')
# extract scores from grid_search
scores = grid.cv_results_['mean_test_score'].reshape(-1, 3).T
# visualize heat map
heatmap = mglearn.tools.heatmap(
    scores, xlabel="C", ylabel="ngram_range", cmap="viridis", fmt="%.3f",
    xticklabels=param_grid['logisticregression__C'],
    yticklabels=param_grid['tfidfvectorizer__ngram_range'])
plt.colorbar(heatmap)
```

Bag-of-Words with More than one word(n-Grams)

```
# In[34]
print('In[34]')
# extract feature names and coefficients
vect = grid.best_estimator_.named_steps['tfidfvectorizer']
feature_names = np.array(vect.get_feature_names())
coef = grid.best_estimator_.named_steps['logisticregression'].coef_
mglearn.tools.visualize_coefficients(coef, feature_names, n_top_features=40)
```

Advance Tokenization, stemming, Lemmatization



Advance Tokenization, stemming, Lemmatization

lemmatization and stemming, are forms of *normalization* that try to extract some normal form of a word. Another interesting case of normalization is spelling correction

Lemmatization seeks to address this issue. This process uses a data structure that relates all forms of a word back to its simplest form

stemming, a word is cut off at its **stem**, the smallest unit of that word from which you can create the descendant word

Advance Tokenization, stemming, Lemmatization

```
# In[36]
print('In[36]')
print('Spacy version: {}'.format(spacy.__version__))
print('nltk version: {}'.format(nltk.__version__))
```

```
In[36]
Spacy version: 2.2.4
nltk version: 3.2.5
```

```
# In[37]
print('In[37]')
# load spacy's English-language models
en_nlp = spacy.load('en')
# instantiate nltk's Porter stemmer
stemmer = nltk.stem.PorterStemmer()
# In[38]
print('In[38]')
compare_normalization(u'Our meeting today was worse than yesterday, ' 'I\'m scared
```

```
In[38]
Lemmatization:
['-PRON-', 'meeting', 'today', 'be', 'bad', 'than', 'yesterday', ',', '-PRON-', 'be',
Stemming:
['our', 'meet', 'today', 'wa', 'wors', 'than', 'yesterday', ',', 'i', 'am', 'scare',
```

```
# In[40]
print('In[40]')
# transform text_train using CountVectorizer with lemmatization
X_train_lemma = lemma_vect.fit_transform(X_train)
print("X_train_lemma.shape: {}".format(X_train_lemma.shape))

# standard CountVectorizer for reference
vect = CountVectorizer(min_df=5).fit(X_train)
X_train_transformed = vect.transform(X_train)
print("X_train.shape: {}".format(X_train_transformed.shape))
```

```
In[40]
X_train_lemma.shape: (1841, 626)
X_train.shape: (1841, 648)
```

Advance Tokenization, stemming, Lemmatization

```
# In[41]
   print('In[41]')
   param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
   cv = StratifiedShuffleSplit(n_splits=5, test_size=0.99, train_size=0.01, random_state=0)
   grid = GridSearchCV(LogisticRegression(), param_grid, cv=cv)
   # perform grid search with standard CountVectorizer
   grid.fit(X_train_transformed, y_train)
   print("Best cross-validation score " "(standard CountVectorizer):
         {:.3f}".format(grid.best_score_))
   # perform grid search with lemmatization
   grid.fit(X_train_lemma, y_train)
   print("Best cross-validation score " "(lemmatization): {:.3f}".format(grid.best_score_))
```

```
In[41]
Best cross-validation score (standard CountVectorizer): 0.549
Best cross-validation score (lemmatization): 0.563
```

```
# In[42]
print('In[42]')
vect = CountVectorizer(max_features=10000, max_df=.15)
X = vect.fit_transform(X_train)
```

```
# In[43]
    print('In[43]')
    lda = LatentDirichletAllocation(n_components=10, learning_method="batch",
                                    max_iter=25, random_state=0)
    # We build the model and transform the data in one step
    # Computing transform takes some time,
    # and we can save time by doing both at once
    document_topics = lda.fit_transform(X)
```

```python
# In[44]
print('In[44]')
print('lda.components_shape: {}'.format(lda.components_.shape))
# In[45]
print('In[45]')
# For each topic (a row in the components_), sort the features (ascending)
# Invert rows with [:, ::-1] to make sorting descending
sorting = np.argsort(lda.components_, axis=1)[:, ::-1]
# Get the feature names from the vectorizer
feature_names = np.array(vect.get_feature_names())
```

```python
# In[46]
    # Print out the 10 topics:
    mglearn.tools.print_topics(topics=range(10), feature_names=feature_names,
                               sorting=sorting, topics_per_chunk=5, n_words=10)
    # In 47
    print('In[47]')
    lda100 = LatentDirichletAllocation(n_components=100, learning_method="batch",
                                       max_iter=25, random_state=0)
    document_topics100 = lda100.fit_transform(X)
```

```
In[44]
lda.components_shape: (10, 4177)
In[45]
topic 0       topic 1       topic 2       topic 3       topic 4
--------      --------      --------      --------      --------
for           good          very          not           in
great         food          phone         have          film
but           quality       service       so            but
their         go            with          that          not
they          for           well          for           for
are           product       not           were          recommend
love          you           as            you           movie
my            so            nice          on            would
really        great         have          my            good
ve            excellent     battery       are           are


topic 5       topic 6       topic 7       topic 8       topic 9
--------      --------      --------      --------      --------
that          that          my            in            in
in            great         be            bad           all
movie         works         back          very          item
for           in            for           have          just
one           phone         in            good          are
with          as            on            no            with
you           good          your          with          one
film          my            that          had           up
there         at            phone         food          get
ever          only          all           not           not
```

Advance Tokenization, stemming, Lemmatization

```python
# In 48
print('In[48]')
topics = np.array([7, 16, 24, 25, 28, 36, 37, 45, 51, 53, 54, 63, 89, 97])

sorting = np.argsort(lda100.components_, axis=1)[:, ::-1]
feature_names = np.array(vect.get_feature_names())
mglearn.tools.print_topics(topics=topics, feature_names=feature_names,
                           sorting=sorting, topics_per_chunk=5, n_words=20)

# In[49]
print('In[49]')
# sort by weight of "music" topic 45
music = np.argsort(document_topics100[:, 45])[::-1]
```

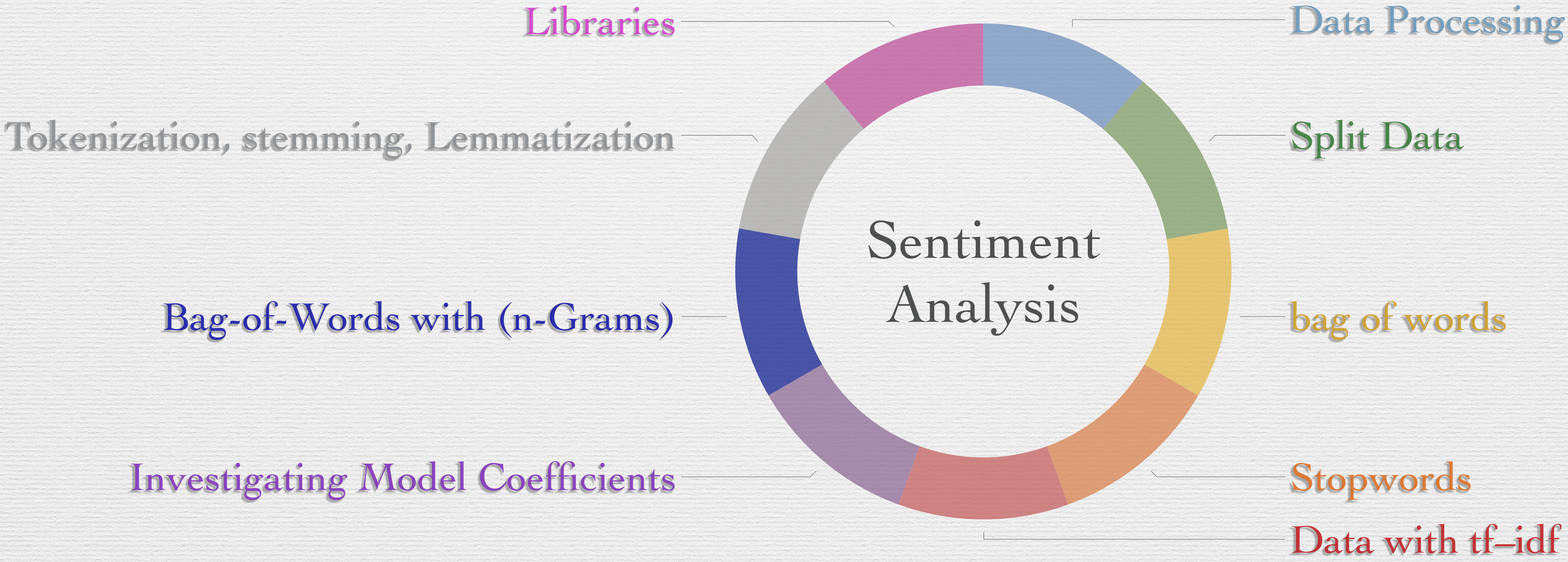
topic 7	topic 16	topic 24	topic 25	topic 28
in	just	they	small	an
my	about	that	are	bad
new	were	waited	that	isn
problems	film	charger	right	after
worked	looking	so	at	even
no	but	know	very	would
at	get	on	movie	so
none	headset	in	don	film
all	not	bad	in	were
money	are	don	their	at
not	at	beautiful	priced	thought
ponyo	my	just	all	rated
which	motorola	phone	from	over
that	got	as	after	their
bad	in	were	with	tool
but	for	we	terrible	excellent
script	you	with	amazed	25
out	so	my	any	actually
support	loose	film	minutes	timeless
up	quality	look	because	however

topic 36	topic 37	topic 45	topic 51
movie	restaurant	in	amazing
one	my	one	absolutely
in	family	film	little
into	very	on	that
that	with	but	an
astronaut	that	ear	restaurant
but	without	other	with
on	run	they	gross
could	have	different	simple
half	two	that	seems
an	days	best	excuse
for	charging	joe	flat
cheerless	battery	scene	grill
with	three	with	vegas
poor	yet	from	attack
lots	new	loads	heart
better	bars	are	lined
factory	below	well	for
judo	sitting	got	in
some	vehicle	who	downtown

topic 54	topic 63	topic 89	topic 97
my	food	won	you
in	service	going	if
come	for	as	are
shipping	terrible	disappointed	for
order	place	back	go
one	want	all	don
phone	have	did	in
not	everything	even	just
from	all	fresh	can
coverage	you	us	with
has	awful	described	see
also	mediocre	regular	have
bad	perfect	movie	again
vegas	or	be	back
sushi	authentic	one	any
wasting	if	are	think
menu	as	order	one
or	means	if	must
very	try	character	by
time	tasty	undoubtedly	as

Advance Tokenization, stemming, Lemmatization

```
# In[50]
print('In[50]')
fig, ax = plt.subplots(1, 2, figsize=(10, 10))
topic_names = ["{:>2} ".format(i) + " ".join(words)
               for i, words in enumerate(feature_names[sorting[:, :2]])]
# two column bar chart:
for col in [0, 1]:
    start = col * 50
    end = (col + 1) * 50
    ax[col].barh(np.arange(50), np.sum(document_topics100, axis=0)[start:end])
    ax[col].set_yticks(np.arange(50))
    ax[col].set_yticklabels(topic_names[start:end], ha="left", va="top")
    ax[col].invert_yaxis()
    ax[col].set_xlim(0, 2000)
    yax = ax[col].get_yaxis()
    yax.set_tick_params(pad=130)
plt.tight_layout()
```


Sentiment Analysis
Libraries
Data Processing
Tokenization, stemming, Lemmatization
Split Data
Bag-of-Words with (n-Grams)
bag of words
Investigating Model Coefficients
Stopwords
Data with tf–idf

Sentiment analysis

Profosser Xiao-Bai LIU

Ghozlan Fagerah
Amiel Nava
Phillip Tang