Text Similarity and Clustering

Dr. M. Brindha
Assistant Professor
Department of CSE
NIT, Trichy-15

Concepts-Information Retrieval

- Information retrieval (IR) is the process of retrieving or fetching relevant sources of information from a corpus or set of entities that hold information based on some demand.
- For example, it could be a query or search that users enter in a search engine and then get relevant search items pertaining to their query.
- In fact, search engines are the most popular use-case or application of IR.

- The relevancy of documents with information compared to the demand can be measured in several ways.
- It can include looking for specific keywords from the search text or using some similarity measures to see the similarity rank or score of the documents with respect to the entered query.
- This makes is quite different from string matching or matching regular expressions because more than often the words in a search string can have different order, context, and semantics in the collection of documents (entities), and these words can even have multiple different resolutions or possibilities based on synonyms, antonyms, and negation modifiers.

Concepts-Feature Engineering

- Feature engineering or feature extraction
- Methods like Bag of Words, TF-IDF, and word vectorization models are typically used to represent or model documents in the form of numeric vectors so that applying mathematical or machine learning techniques become much easier.

Concepts-Similarity Measures

- Similarity measures are used frequently in text similarity analysis and clustering.
- Any similarity or distance measure usually measures the degree of closeness between two entities, which can be any text format like documents, sentences, or even terms.
- This measure of similarity can be useful in identifying similar entities and distinguishing clearly different entities from each other.
- Similarity measures are very effective, and sometimes choosing the right measure can make a lot of difference in the performance of the final analytics system.
- Various scoring or ranking algorithms have also been invented based on these distance measures.
- Two main factors determine the degree of similarity between entities:
  - Inherent properties or features of the entities
  - Measure formula and properties

Concepts-Similarity Measures

- All distance measures of similarity are not distance metrics of similarity.
- Consider a distance measure d and two entities (say they are documents in our context) x and y.
- The distance between x and y, which is used to determine the degree of similarity between them, can be represented as d(x, y), but the measure d can be called as a distance metric of similarity if and only if it satisfies the following four conditions:

1. The distance measured between any two entities, say x and y, must be always non-negative, that is, d(x, y) >= 0 .
2. The distance between two entities should always be zero if and only if they are both identical, that is, d(x, y) >= 0 iff x = y .
3. This distance measure should always be symmetric, which means that the distance from x to y is always the same as the distance from y to x. Mathematically this is represented as d(x, y) = d(y, x) .
4. This distance measure should satisfy the triangle inequality property, which can be mathematically represented d(x, z) <= d(x, y)+ d(y, z) .

Concepts-Unsupervised Machine Learning Algorithms

- Unsupervised machine learning algorithms are the family of ML algorithms that try to discover latent hidden structures and patterns in data from their various attributes and features.
- Besides this, several unsupervised learning algorithms are also used to reduce the feature space, which is often of a higher dimension to one with a lower dimension.
- The data on which these algorithms operate is essentially unlabeled data that does not have any pre-determined category or class.
- We apply these algorithms with the intent of finding patterns and distinguishing features that might help us in grouping various data points into groups or clusters.
- These algorithms are popularly known as clustering algorithms.

Text Normalization

```
    stopword_list = nltk.corpus.stopwords.words('english')
    stopword_list = stopword_list + ['mr', 'mrs', 'come', 'go', 'get', 'tell',
    'listen', 'one', 'two', 'three', 'four', 'five',
    'six', 'seven', 'eight',
    'nine', 'zero', 'join', 'find', 'make', 'say', 'ask',
    'tell', 'see', 'try', 'back', 'also']

def normalize_corpus(corpus, lemmatize=True,
                     only_text_chars=False,
                     tokenize=False):

    normalized_corpus = []
    for text in corpus:
        text = html_parser.unescape(text)
        text = expand_contractions(text, CONTRACTION_MAP)
        if lemmatize:
            text = lemmatize_text(text)
        else:
            text = text.lower()
        text = remove_special_characters(text)
        text = remove_stopwords(text)
        if only_text_chars:
            text = keep_text_characters(text)

        if tokenize:
            text = tokenize_text(text)
            normalized_corpus.append(text)
        else:
            normalized_corpus.append(text)

    return normalized_corpus
```

Expanding contractions, unescaping HTML, tokenization, removing stopwords, special characters, and lemmatization.

Feature Extraction

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

def build_feature_matrix(documents, feature_type='frequency',
                         ngram_range=(1, 1), min_df=0.0, max_df=1.0):

    feature_type = feature_type.lower().strip()

    if feature_type == 'binary':
        vectorizer = CountVectorizer(binary=True, min_df=min_df,
                                     max_df=max_df, ngram_range=ngram_range)
    elif feature_type == 'frequency':
        vectorizer = CountVectorizer(binary=False, min_df=min_df,
                                     max_df=max_df, ngram_range=ngram_range)
    elif feature_type == 'tfidf':
        vectorizer = TfidfVectorizer(min_df=min_df, max_df=max_df,
                                     ngram_range=ngram_range)
    else:
        raise Exception("Wrong feature type entered. Possible values:
'binary', 'frequency',
                        'tfidf'")

    feature_matrix = vectorizer.fit_transform(documents).astype(float)

    return vectorizer, feature_matrix
```

Feature Extraction

- The new additions in this function include the addition of the min_df, max_df and ngram_range parameters and also accepting them as optional arguments.
- The ngram_range is useful when we want to add bigrams, trigrams, and so on as additional features.
- The min_df parameter can be expressed by a threshold value within a range of [0.0, 1.0] and it will ignore terms as features that will have a document frequency strictly lower than the input threshold value.
- The max_df parameter can also be expressed by a threshold value within a range of [0.0, 1.0] and it will ignore terms as features that will have a document frequency strictly higher than the input threshold value.

Text Similarity

- The main objective of text similarity is to analyze and measure how two entities of text are close or far apart from each other.
- These entities of text can be simple tokens or terms, like words, or whole documents, which may include sentences or paragraphs of text.
- There are various ways of analyzing text similarity, and we can classify the intent of text similarity broadly into the following two areas:
  - Lexical similarity: This involves observing the contents of the text documents with regard to syntax, structure, and content and measuring their similarity based on these parameters.
  - Semantic similarity: This involves trying to find out the semantics, meaning, and context of the documents and then trying to see how close they are to each other. Dependency grammars and entity recognition are handy tools that can help in this.

Text Similarity

- The most popular area is lexical similarity, because the techniques are more straightforward, easy to implement.
- Usually distance metrics will be used to measure similarity scores between text entities.
  - Term similarity: Here we will measure similarity between individual tokens or words.
  - Document similarity: Here we will be measuring similarity between entire text documents.

Analyzing Term Similarity

- The word representations
  - Character vectorization- it is an extremely simple process of just mapping each character of the term to a corresponding unique number.

```
import numpy as np

def vectorize_terms(terms):
    terms = [term.lower() for term in terms]
    terms = [np.array(list(term)) for term in terms]
    terms = [np.array([ord(char) for char in term])
                for term in terms]
    return terms
```

- The function takes input a list of words or terms and returns the corresponding character vectors for the words.
  - Bag of Characters vectorization
- Bag of Characters vectorization is very similar to the Bag of Words model except here we compute the frequency of each character in the word.
- Sequence or word orders are not taken into account.

```
from scipy.stats import itemfreq

def boc_term_vectors(word_list):
    word_list = [word.lower() for word in word_list]
    unique_chars = np.unique(
                        np.hstack([list(word)
                        for word in word_list]))
    word_list_term_counts = [{char: count for char, count in
itemfreq(list(word))}
                                for word in word_list]

    boc_vectors = [np.array([int(word_term_counts.get(char, 0))
                                for char in unique_chars])
                    for word_term_counts in word_list_term_counts]
    return list(unique_chars), boc_vectors
```

Analyzing Term Similarity

- There are a lot of distance metrics out there that you can use to compute and measure similarities.
- Hamming distance
- Manhattan distance
- Euclidean distance
- Levenshtein edit distance
- Cosine distance and similarity

```
root_term = root
root_vector = vec_root
root_boc_vector = boc_root

terms = [term1, term2, term3]
vector_terms = [vec_term1, vec_term2, vec_term3]
boc_vector_terms = [boc_term1, boc_term2, boc_term3]
```

Analyzing Term Similarity

Hamming Distance

- It is distance measured between two strings under the assumption that they are of equal length.
- Formally, it is defined as the number of positions that have different characters or symbols between two strings of equal length.
- Considering two terms u and v of length n, we can mathematically denote Hamming distance as. $hd(u,v)=\sum_{i=1}^{n}(u_i \neq v_i)$
- Normalize it by dividing the number of mismatches by the total length of the terms to give the normalized hamming distance, which is represented as $norm\_hd(u,v)=\frac{\sum_{i=1}^{n}(u_i \neq v_i)}{n}$

```
def hamming_distance(u, v, norm=False):
    if u.shape != v.shape:
        raise ValueError('The vectors must have equal lengths.')
    return (u != v).sum() if not norm else (u != v).mean()
```

Analyzing Term Similarity

Hamming Distance

```
# compute Hamming distance
In [115]: for term, vector_term in zip(terms, vector_terms):
     ...:     print 'Hamming distance between root: {} and term: {} is {}'.
format(root_term,
     ...:                         term, hamming_distance(root_vector, vector_
term, norm=False))

Hamming distance between root: Believe and term: believe is 2
Hamming distance between root: Believe and term: bargain is 6
Traceback (most recent call last):
  File "<ipython-input-115-3391bd2c4b7e>", line 4, in <module>
    hamming_distance(root_vector, vector_term, norm=False))
ValueError: The vectors must have equal lengths.

# compute normalized Hamming distance
In [117]: for term, vector_term in zip(terms, vector_terms):
     ...:     print 'Normalized Hamming distance between root: {} and term:
{} is
     ...:                                                 {}'.format(root_term,
                                                          term,
     ...:             round(hamming_distance(root_vector, vector_term,
                      norm=True), 2))

Normalized Hamming distance between root: Believe and term: believe is 0.29
Normalized Hamming distance between root: Believe and term: bargain is 0.86
Traceback (most recent call last):
  File "<ipython-input-117-7dfc67d08c3f>", line 4, in <module>
    round(hamming_distance(root_vector, vector_term, norm=True), 2))
ValueError: The vectors must have equal lengths
```

Analyzing Term Similarity

Manhattan Distance

- The Manhattan distance metric is similar to the Hamming distance conceptually, where instead of counting the number of mismatches, we subtract the difference between each pair of characters at each position of the two strings.
- Formally, Manhattan distance is also known as city block distance, L1 norm, taxicab metric and is defined as the distance between two points in a grid based on strictly horizontal or vertical paths instead of the diagonal distance conventionally calculated by the Euclidean distance metric.

$$md(u,v) = \|u - v\|_1 = \sum_{i=1}^{n} |u_i - v_i|$$

- where u and v are the two terms of length n.
- Compute the normalized Manhattan distance by dividing the sum of the absolute differences by the term length.

$$norm\_md(u,v) = \frac{\|u - v\|_1}{n} = \frac{\sum_{i=1}^{n} |u_i - v_i|}{n}$$

where *n* is the length of each of the terms *u* and *v*.

Analyzing Term Similarity

Manhattan Distance

```
def manhattan_distance(u, v, norm=False):
    if u.shape != v.shape:
        raise ValueError('The vectors must have equal lengths.')
    return abs(u - v).sum() if not norm else abs(u - v).mean()
```

```
# compute Manhattan distance
In [120]: for term, vector_term in zip(terms, vector_terms):
     ...:     print 'Manhattan distance between root: {} and term: {} is
     ...:     {}'.format(root_term,
     ...:                term, manhattan_distance(root_vector,
                         vector_term, norm=False))

Manhattan distance between root: Believe and term: believe is 8
Manhattan distance between root: Believe and term: bargain is 38
Traceback (most recent call last):
  File "<ipython-input-120-b228f24ad6a2>", line 4, in <module>
    manhattan_distance(root_vector, vector_term, norm=False))
ValueError: The vectors must have equal lengths.

# compute normalized Manhattan distance
In [122]: for term, vector_term in zip(terms, vector_terms):
     ...:     print 'Normalized Manhattan distance between root: {} and
              term: {} is {}'.format(root_term,


     ...:             term,
     ...:             round(manhattan_distance(root_vector, vector_term,
                      norm=True),2))
     ...:
     ...:
Normalized Manhattan distance between root: Believe and term: believe is 1.14
Normalized Manhattan distance between root: Believe and term: bargain is 5.43
Traceback (most recent call last):
  File "<ipython-input-122-d13a48d56a22>", line 4, in <module>
    round(manhattan_distance(root_vector, vector_term, norm=True),2))
ValueError: The vectors must have equal lengths.
```

Analyzing Term Similarity

Euclidean Distance

- Euclidean distance is also known as the Euclidean norm, L2 norm, or L2 distance and is defined as the shortest straight-line distance between two points:

$$ed(u,v) = \|u - v\|_2 = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2}$$

- where the two points u and v are vectorized text terms in our scenario, each having length n.

```
def euclidean_distance(u, v):
    if u.shape != v.shape:
        raise ValueError('The vectors must have equal lengths.')
    distance = np.sqrt(np.sum(np.square(u - v)))
    return distance


# compute Euclidean distance
In [132]: for term, vector_term in zip(terms, vector_terms):
     ...:     print 'Euclidean distance between root: {} and term: {} is
{}'.format(root_term,
     ...:                           term, round(euclidean_distance(root_
                                    vector, vector_term),2))
Euclidean distance between root: Believe and term: believe is 5.66
Euclidean distance between root: Believe and term: bargain is 17.94
Traceback (most recent call last):
  File "<ipython-input-132-90a4dbe8ce60>", line 4, in <module>
    round(euclidean_distance(root_vector, vector_term),2))
ValueError: The vectors must have equal lengths.
```

Analyzing Term Similarity

Levenshtein Edit Distance

- The Levenshtein edit distance between two terms can be defined as the minimum number of edits needed in the form of additions, deletions, or substitutions to change or convert one term to the other.
- These substitutions are character-based substitutions, where a single character can be edited in a single operation.
- Also, the length of the two terms need not be equal here.
- Mathematically, we can represent the Levenshtein edit distance between two terms as *ldu, v*(|*u*|, |*v*|) such that *u* and *v* are our two terms where |*u*| and |*v*| are their lengths.

$$ld_{u,v}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} ld_{u,v}(i-1,j)+1 \\ ld_{u,v}(i,j-1)+1 \\ ld_{u,v}(i-1,j-1)+C_{u_i \neq v_j} \end{cases} & \text{otherwise} \end{cases}$$

- where i and j are basically indices for the terms u and v. The third equation in the minimum above has a cost function denoted by $C_{u_i \neq v_j}$ such that it has the following conditions

$$C_{u_i \neq v_j} = \begin{cases} 1 & \text{if } u_i \neq v_j \\ 0 & \text{if } u_i = v_j \end{cases}$$

and this denotes the indicator function, which depicts the cost associated with two characters being matched for the two terms

Analyzing Term Similarity

Levenshtein Edit Distance

- Considering the root term 'believe' and another term 'beleive' (we ignore case in our computations).
- The edit distance would be 2 because we would need the following two operations:
  - 'beleive' → 'beliive' (substitution of e to i)
  - 'beliive' → 'believe' (substitution of i to e)

	b	e	l	i	e	v	e
b	0	1	2	3	4	5	6
e	1	0	1	2	3	4	5
l	2	1	0	1	2	3	4
e	3	2	1	1	1	2	3
i	4	3	2	1	2	2	3
v	5	4	3	2	2	2	3
e	6	5	4	3	2	3	2

```
function levenshtein_distance(char u[1..m], char v[1..n]):
# for all i and j, d[i,j] will hold the Levenshtein distance between the
first i characters of
# u and the first j characters of v, note that d has (m+1)*(n+1) values
int d[0..m, 0..n]

# set each element in d to zero
d[0..m, 0..n] := 0

# source prefixes can be transformed into empty string by dropping all
characters
for i from 1 to m:
   d[i, 0] := i

# target prefixes can be reached from empty source prefix by inserting every
character
for j from 1 to n:
    d[0, j] := j

# build the edit distance matrix
for j from 1 to n:
    for i from 1 to m:
        if s[i] = t[j]:
            substitutionCost := 0
        else:
            substitutionCost := 1
            d[i, j] := minimum(d[i-1, j] + 1,                    # deletion
                               d[i, j-1] + 1,                    # insertion
                               d[i-1, j-1] + substitutionCost)   # substitution

# the final value of the matrix is the edit distance between the terms
return d[m, n]
```

Analyzing Term Similarity

Levenshtein Edit Distance

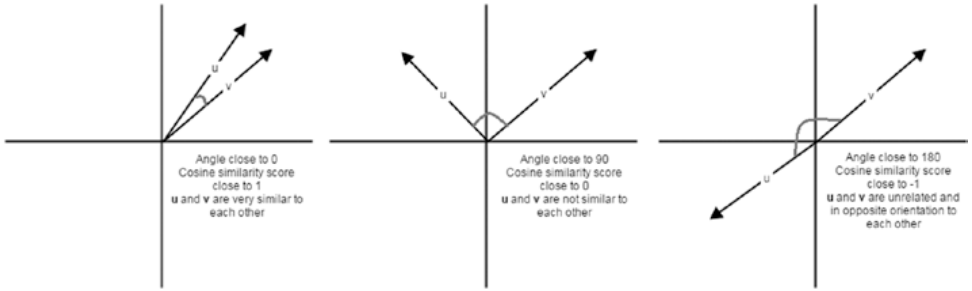
```
import copy
import pandas as pd

def levenshtein_edit_distance(u, v):
    # convert to lower case
    u = u.lower()
    v = v.lower()
    # base cases
    if u == v: return 0
    elif len(u) == 0: return len(v)
    elif len(v) == 0: return len(u)
    # initialize edit distance matrix
    edit_matrix = []
    # initialize two distance matrices
    du = [0] * (len(v) + 1)
    dv = [0] * (len(v) + 1)
    # du: the previous row of edit distances
    for i in range(len(du)):
        du[i] = i
    # dv : the current row of edit distances
    for i in range(len(u)):
        dv[0] = i + 1
        # compute cost as per algorithm
        for j in range(len(v)):
            cost = 0 if u[i] == v[j] else 1
            dv[j + 1] = min(dv[j] + 1, du[j + 1] + 1, du[j] + cost)
        # assign dv to du for next iteration
        for j in range(len(du)):
            du[j] = dv[j]
        # copy dv to the edit matrix
        edit_matrix.append(copy.copy(dv))
    # compute the final edit distance and edit matrix
    distance = dv[len(v)]
    edit_matrix = np.array(edit_matrix)
    edit_matrix = edit_matrix.T
    edit_matrix = edit_matrix[1:,]
    edit_matrix = pd.DataFrame(data=edit_matrix,
                               index=list(v),
                               columns=list(u))
    return distance, edit_matrix
```

```
In [223]: for term in terms:
     ...:     edit_d, edit_m = levenshtein_edit_distance(root_term, term)
     ...:     print 'Computing distance between root: {} and term: {}'.
              format(root_term,
     ...:                                                              term)
     ...:     print 'Levenshtein edit distance is {}'.format(edit_d)
     ...:     print 'The complete edit distance matrix is depicted below'
     ...:     print edit_m
     ...:     print '-'*30

Computing distance between root: Believe and term: beleive
Levenshtein edit distance is 2
The complete edit distance matrix is depicted below
   b  e  l  i  e  v  e
b  0  1  2  3  4  5  6
e  1  0  1  2  3  4  5
l  2  1  0  1  2  3  4
e  3  2  1  1  1  2  3
i  4  3  2  1  2  2  3
v  5  4  3  2  2  2  3
e  6  5  4  3  2  3  2
------------------------------
Computing distance between root: Believe and term: bargain
Levenshtein edit distance is 6
The complete edit distance matrix is depicted below
   b  e  l  i  e  v  e
b  0  1  2  3  4  5  6
a  1  1  2  3  4  5  6
r  2  2  2  3  4  5  6
g  3  3  3  3  4  5  6
a  4  4  4  4  4  5  6
i  5  5  5  4  5  5  6
n  6  6  6  5  5  6  6
------------------------------
Computing distance between root: Believe and term: Elephant
Levenshtein edit distance is 7
The complete edit distance matrix is depicted below
   b  e  l  i  e  v  e
e  1  1  2  3  4  5  6
l  2  2  1  2  3  4  5
e  3  2  2  2  2  3  4
p  4  3  3  3  3  3  4
h  5  4  4  4  4  4  4
a  6  5  5  5  5  5  5
n  7  6  6  6  6  6  6
t  8  7  7  7  7  7  7
------------------------------
```

Analyzing Term Similarity

Cosine Distance and Similarity

- The Cosine distance is a metric that can be actually derived from the Cosine similarity and vice versa.
- Considering we have two terms such that they are represented in their vectorized forms, Cosine similarity gives us the measure of the cosine of the angle between them when they are represented as non-zero positive vectors in an inner product space.
- Thus term vectors having similar orientation will have scores closer to 1 ($\cos 0^\circ$) indicating the vectors are very close to each other in the same direction (near to zero degree angle between them).
- Term vectors having a similarity score close to 0 ($\cos 90^\circ$) indicate unrelated terms with a near orthogonal angle between then.
- Term vectors with a similarity score close to –1 ($\cos 180^\circ$) indicate terms that are completely oppositely oriented to each other.



Analyzing Term Similarity

Cosine Distance and Similarity

- Cosine similarity as the dot product of the two term vectors u and v, divided by the product of their L2 norms.
- Mathematically, represent the dot product between two vectors $u \cdot v = \|u\| \|v\| \cos(\theta)$
- where θ is the angle between u and v and u represents the L2 norm for vector u and v is the L2 norm for vector v

$$cs(u,v) = \cos(\theta) = \frac{u \cdot v}{\|u\| \|v\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \cdot \sqrt{\sum_{i=1}^{n} v_i^2}}$$

- where cs(u, v) is the Cosine similarity score between u and v. Here ui and vi are the various features or components of the two vectors, and the total number of these features or components is n.
- In our case, we will be using the Bag of Characters vectorization to build these term vectors, and n will be the number of unique characters across the terms under analysis.
- An important thing to note here is that the Cosine similarity score usually ranges from –1 to +1, but if we use the Bag of Characters–based character frequencies for terms or Bag of Words–based word frequencies for documents, the score will range from 0 to 1 because the frequency vectors can never be negative, and hence the angle between the two vectors cannot exceed 90° .

Analyzing Term Similarity

Cosine Distance and Similarity

The Cosine distance is complimentary to the similarity score can be computed by the formula

$$cd(u,v) = 1 - cs(u,v) = 1 - \cos(\theta) = 1 - \frac{u \cdot v}{\|u\| \, \|v\|} = 1 - \frac{\sum_{i=1}^{n} u_i \, v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}}$$

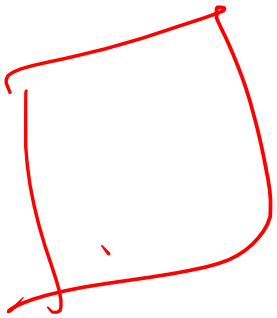
```
def cosine_distance(u, v):
    distance = 1.0 - (np.dot(u, v) /
                      (np.sqrt(sum(np.square(u))) * np.sqrt(sum(np.
                      square(v))))
                     )
    return distance
```

```
In [235]: for term, boc_term in zip(terms, boc_vector_terms):
     ...:     print 'Analyzing similarity between root: {} and term: {}'.
format(root_term,
     ...:                                                          term)
     ...:     distance = round(cosine_distance(root_boc_vector, boc_term),2)
     ...:     similarity = 1 - distance
     ...:     print 'Cosine distance  is {}'.format(distance)
     ...:     print 'Cosine similarity  is {}'.format(similarity)
     ...:     print '-'*40
```

```
Analyzing similarity between root: Believe and term: believe
Cosine distance  is -0.0
Cosine similarity  is 1.0
----------------------------------------
Analyzing similarity between root: Believe and term: bargain
Cosine distance  is 0.82
Cosine similarity  is 0.18
----------------------------------------
Analyzing similarity between root: Believe and term: Elephant
Cosine distance  is 0.39
Cosine similarity  is 0.61
----------------------------------------
```

Analyzing Document Similarity

- Cosine similarity
- Hellinger-Bhattacharya distance
- Okapi BM25 ranking



- The metrics on a toy corpus here with nine documents and a separate corpus with three documents, which will be the query documents.
- For each of these three documents, we will try to find out the most similar documents from the corpus of nine documents, which will act as our index.

Analyzing Document Similarity

- Cosine similarity
- Hellinger-Bhattacharya distance
- Okapi BM25 ranking

- our metrics on a toy corpus here with nine documents and a separate corpus with three documents, which will be our query documents.
- For each of these three documents, we will try to find out the most similar documents from the corpus of nine documents, which will act as our index.

```
from normalization import normalize_corpus
from utils import build_feature_matrix
import numpy as np

# load the toy corpus index
toy_corpus = ['The sky is blue',
'The sky is blue and beautiful',
'Look at the bright blue sky!',
'Python is a great Programming language',
'Python and Java are popular Programming languages',
'Among Programming languages, both Python and Java are the most used in
Analytics',
'The fox is quicker than the lazy dog',
'The dog is smarter than the fox',
'The dog, fox and cat are good friends']

# load the docs for which we will be measuring similarities
query_docs = ['The fox is definitely smarter than the dog',
            'Java is a static typed programming language unlike Python',
            'I love to relax under the beautiful blue sky!']
```

```
# normalize and extract features from the toy corpus
norm_corpus = normalize_corpus(toy_corpus, lemmatize=True)
tfidf_vectorizer, tfidf_features = build_feature_matrix(norm_corpus,
                                                        feature_
                                                        type='tfidf',
                                                        ngram_range=(1, 1),
                                                        min_df=0.0, max_
                                                        df=1.0)
```

```
# normalize and extract features from the query corpus
norm_query_docs =  normalize_corpus(query_docs, lemmatize=True)
query_docs_tfidf = tfidf_vectorizer.transform(norm_query_docs)
```

Analyzing Document Similarity

Cosine similarity

- The document vectors will be the Bag of Words model–based vectors with TF-IDF values instead of term frequencies. We have also taken only unigrams here, but you can experiment with bigrams and so on as document features during the vectorization process.
- For each of the three query documents, we will compute its similarity with the nine documents in toy_corpus and return the n most similar documents where n is a user input parameter.

```
def compute_cosine_similarity(doc_features, corpus_features,
                              top_n=3):
    # get document vectors
    doc_features = doc_features.toarray()[0]
    corpus_features = corpus_features.toarray()
    # compute similarities
    similarity = np.dot(doc_features,
                        corpus_features.T)
    # get docs with highest similarity scores
    top_docs = similarity.argsort()[::-1][:top_n]
    top_docs_with_score = [(index, round(similarity[index], 3))
                           for index in top_docs]
    return top_docs_with_score
```

Analyzing Document Similarity

Cosine similarity

- These documents will be retrieved on the basis of their similarity score with doc_features, which basically represents the vectorized document belonging to each of the query_docs,

```
# get Cosine similarity results for our example documents
In [243]: print 'Document Similarity Analysis using Cosine Similarity'
     ...: print '='*60
     ...: for index, doc in enumerate(query_docs):
     ...:
     ...:     doc_tfidf = query_docs_tfidf[index]
     ...:     top_similar_docs = compute_cosine_similarity(doc_tfidf,
     ...:                                                  tfidf_features,
     ...:                                                  top_n=2)
     ...:     print 'Document',index+1 ,':', doc
     ...:     print 'Top', len(top_similar_docs), 'similar docs:'
     ...:     print '-'*40
     ...:     for doc_index, sim_score in top_similar_docs:
     ...:         print 'Doc num: {} Similarity Score: {}\nDoc: {}'.
                  format(doc_index+1,
     ...:
sim_score, toy_corpus[doc_index])
     ...:         print '-'*40
     ...:     print

Document Similarity Analysis using Cosine Similarity
============================================================
Document 1 : The fox is definitely smarter than the dog
Top 2 similar docs:
----------------------------------------
Doc num: 8 Similarity Score: 1.0
Doc: The dog is smarter than the fox
----------------------------------------
Doc num: 7 Similarity Score: 0.426
Doc: The fox is quicker than the lazy dog
----------------------------------------

Document 2 : Java is a static typed programming language unlike Python
Top 2 similar docs:
----------------------------------------
Doc num: 5 Similarity Score: 0.837
Doc: Python and Java are popular Programming languages
----------------------------------------
Doc num: 6 Similarity Score: 0.661
Doc: Among Programming languages, both Python and Java are the most used in
Analytics
----------------------------------------

Document 3 : I love to relax under the beautiful blue sky!
Top 2 similar docs:
----------------------------------------
Doc num: 2 Similarity Score: 1.0
Doc: The sky is blue and beautiful
----------------------------------------
Doc num: 1 Similarity Score: 0.72
Doc: The sky is blue
```

Analyzing Document Similarity

Hellinger-Bhattacharya Distance

- The Hellinger-Bhattacharya distance (HB-distance) is also called the Hellinger distance or the Bhattacharya distance.
- The Bhattacharya distance is used to measure the similarity between two discrete or continuous probability distributions.
- The Hellinger-Bhattacharya distance is an f-divergence, which in the theory of probability is defined as a function $Df(P||Q)$, which can be used to measure the difference between P and Q probability distributions.
- There are many instances of f-HB-distance including KL-divergence and HB-distance.
- KL-divergence is not a distance metric because it violates the symmetric condition from the four conditions necessary for a distance measure to be a metric.
- HB-distance is computable for both continuous and discrete probability distributions.
- We will be using the TF-IDF–based vectors as our document distributions. This makes it discrete distributions because we have specific TF-IDF values for specific feature terms, unlike continuous distributions.

Analyzing Document Similarity

Hellinger-Bhattacharya Distance

$$hbd(u,v)=\frac{1}{\sqrt{2}}\left\|\sqrt{u}-\sqrt{v}\right\|_2$$

- where hbd(u, v) denotes the Hellinger-Bhattacharya distance between the document vectors u and v, and it is equal to the Euclidean or L2 norm of the difference of the square root of the vectors divided by the square root of 2.
- Considering the document vectors u and v to be discrete with n number of features, we can further expand the above formula into

$$hbd(u,v)=\frac{1}{\sqrt{2}}\sqrt{\sum_{i=1}^{n}\left(\sqrt{u_i}-\sqrt{v_i}\right)^2}$$

- such that $u=(u_1,u_2,\ldots,u_n)$ and $v=(v_1,v_2,\ldots,v_n)$ are the document vectors having length n indicating n features, which are the TF-IDF weights of the various terms in the documents.

```
def compute_hellinger_bhattacharya_distance(doc_features, corpus_features,
                                            top_n=3):
    # get document vectors
    doc_features = doc_features.toarray()[0]
    corpus_features = corpus_features.toarray()
    # compute hb distances
    distance = np.hstack(
                    np.sqrt(0.5 *
                            np.sum(
                                np.square(np.sqrt(doc_features) -
                                          np.sqrt(corpus_features)),
                                axis=1)))
    # get docs with lowest distance scores
    top_docs = distance.argsort()[:top_n]
    top_docs_with_score = [(index, round(distance[index], 3))
                           for index in top_docs]
    return top_docs_with_score
```

Analyzing Document Similarity

Hellinger-Bhattacharya Distance

```
# get Hellinger-Bhattacharya distance based similarities for our example
documents
In [246]: print 'Document Similarity Analysis using Hellinger-Bhattacharya
distance'
     ...: print '='*60
     ...: for index, doc in enumerate(query_docs):
     ...:
     ...:     doc_tfidf = query_docs_tfidf[index]
     ...:     top_similar_docs = compute_hellinger_bhattacharya_
              distance(doc_tfidf,
     ...:                                                  tfidf_features,
     ...:                                                  top_n=2)
     ...:     print 'Document',index+1 ,':', doc
     ...:     print 'Top', len(top_similar_docs), 'similar docs:'
     ...:     print '-'*40
     ...:     for doc_index, sim_score in top_similar_docs:
     ...:         print 'Doc num: {} Distance Score: {}\nDoc: {}'.
              format(doc_index+1,
     ...:                                          sim_score, toy_corpus[doc_
                                                   index])
     ...:         print '-'*40
     ...:     print
     ...:
     ...:
Document Similarity Analysis using Hellinger-Bhattacharya distance
============================================================
Document 1 : The fox is definitely smarter than the dog
Top 2 similar docs:
----------------------------------------
Doc num: 8 Distance Score: 0.0
Doc: The dog is smarter than the fox
----------------------------------------
Doc num: 7 Distance Score: 0.96
Doc: The fox is quicker than the lazy dog
----------------------------------------

Document 2 : Java is a static typed programming language unlike Python
Top 2 similar docs:
----------------------------------------
Doc num: 5 Distance Score: 0.53
Doc: Python and Java are popular Programming languages
----------------------------------------
Doc num: 4 Distance Score: 0.766
Doc: Python is a great Programming language
----------------------------------------

Document 3 : I love to relax under the beautiful blue sky!
Top 2 similar docs:
----------------------------------------
Doc num: 2 Distance Score: 0.0
Doc: The sky is blue and beautiful
----------------------------------------
Doc num: 1 Distance Score: 0.602
Doc: The sky is blue
----------------------------------------
```

Analyzing Document Similarity

Okapi BM25 Ranking

- There are several techniques that are quite popular in information retrieval and search engines, including PageRank and Okapi BM25.
- The acronym BM stands for best Matching.
- The Okapi BM25 can be formally defined as a document ranking and retrieval function based on a Bag of Words–based model for retrieving relevant documents based on a user input query.
- This query can be itself a document containing a sentence or collection of sentences, or it can even be a couple of words.
- The Okapi BM25 is actually not just a single function but is a framework consisting of a whole collection of scoring functions combined together.

Analyzing Document Similarity

Okapi BM25 Ranking

- Say we have a query document QD such that $QD=(q_1, q_2, \ldots, q_n)$ containing n terms or keywords and we have a corpus document CD in the corpus of documents from which we want to get the most relevant documents to the query document based on similarity scores.
- BM25 score between these two documents:

$$bm25(CD, QD) = \sum_{i=1}^{n} idf(q_i) \cdot \frac{f(q_i, CD) \cdot (k_1 + 1)}{f(q_i, CD) + k_1 \cdot \left(1 - b + b \cdot \frac{|CD|}{avgdl}\right)}$$

- where the function bm25(CD, QD) computes the BM25 rank or score of the document CD based on the query document QD.
- The function idf(qi) gives us the inverse document frequency (IDF) of the term qi in the corpus that contains CD and from which we want to retrieve the relevant documents.
- IDF using TF-IDF feature extractor

$$idf(t) = 1 + \log \frac{C}{1 + df(t)}$$

Analyzing Document Similarity

Okapi BM25 Ranking

- Say we have a query document QD such that $QD=(q_1, q_2, \ldots, q_n)$ containing n terms or keywords and we have a corpus document CD in the corpus of documents from which we want to get the most relevant documents to the query document based on similarity scores.
- BM25 score between these two documents:

$$bm25(CD, QD)=\sum_{i=1}^{n} idf(q_i)\cdot\frac{f(q_i, CD)\cdot(k_1+1)}{f(q_i, CD)+k_1\cdot\left(1-b+b\cdot\frac{|CD|}{avgdl}\right)}$$

- where the function bm25(CD, QD) computes the BM25 rank or score of the document CD based on the query document QD.
- The function idf(qi) gives us the inverse document frequency (IDF) of the term qi in the corpus that contains CD and from which we want to retrieve the relevant documents.
- IDF using TF-IDF feature extractor

$$idf(t)=1+\log\frac{C}{1+df(t)}$$

- where idf(t) represents the idf for the term t and C represents the count of the total number of documents in our corpus and df(t) represents the frequency of the number of documents in which the term t is present.

Analyzing Document Similarity

Okapi BM25 Ranking

- There are various other methods of implementing IDF, but we will be using this one, and on a side note the end outcome from the different implementations is very similar.
- The function f(qi, CD) gives us the frequency of the term qi in the corpus document CD.
- The expression |CD| indicates the total length of the document CD which is measured by its number of words, and the term avgdl represents the average document length of the corpus from which we will be retrieving documents.
- Besides that, you will also observe there are two free parameters, k1, which is usually in the range of [1.2, 2.0], and b, which is usually taken as 0.75.

Analyzing Document Similarity

Okapi BM25 Ranking

- There are several steps we must go through to successfully implement and compute BM25 scores for documents:

1. Build a function to get inverse document frequency (IDF) values for terms in corpus.

2. Build a function for computing BM25 scores for query document and corpus documents.

3. Get Bag of Words–based features for corpus documents and query documents.

4. Compute average length of corpus documents and IDFs of the terms in the corpus documents using function from point 1.

5. Compute BM25 scores, rank relevant documents, and fetch the n most relevant documents for each query document using the function in point 2.

Analyzing Document Similarity

Okapi BM25 Ranking

```
import scipy.sparse as sp

def compute_corpus_term_idfs(corpus_features, norm_corpus):

    dfs = np.diff(sp.csc_matrix(corpus_features, copy=True).indptr)
    dfs = 1 + dfs # to smoothen idf later
    total_docs = 1 + len(norm_corpus)
    idfs = 1.0 + np.log(float(total_docs) / dfs)
    return idfs
```

```
    # build bag of words based features first
    vectorizer, corpus_features = build_feature_matrix(norm_corpus,
                                                       feature_type='frequency')
    query_docs_features = vectorizer.transform(norm_query_docs)

    # get average document length of the corpus (avgdl)
    doc_lengths = [len(doc.split()) for doc in norm_corpus]
    avg_dl = np.average(doc_lengths)
```

```
def compute_bm25_similarity(doc_features, corpus_features,
                            corpus_doc_lengths, avg_doc_length,
                            term_idfs, k1=1.5, b=0.75, top_n=3):
    # get corpus bag of words features
    corpus_features = corpus_features.toarray()
    # convert query document features to binary features
    # this is to keep a note of which terms exist per document
    doc_features = doc_features.toarray()[0]
    doc_features[doc_features >= 1] = 1

    # compute the document idf scores for present terms
    doc_idfs = doc_features * term_idfs
    # compute numerator expression in BM25 equation
    numerator_coeff = corpus_features * (k1 + 1)
    numerator = np.multiply(doc_idfs, numerator_coeff)
    # compute denominator expression in BM25 equation
    denominator_coeff =  k1 * (1 - b +
                               (b * (corpus_doc_lengths /
                                     avg_doc_length)))
    denominator_coeff = np.vstack(denominator_coeff)
    denominator = corpus_features + denominator_coeff
    # compute the BM25 score combining the above equations
    bm25_scores = np.sum(np.divide(numerator,
                                   denominator),
                         axis=1)
    # get top n relevant docs with highest BM25 score
    top_docs = bm25_scores.argsort()[::-1][:top_n]
    top_docs_with_score = [(index, round(bm25_scores[index], 3))
                           for index in top_docs]
    return top_docs_with_score
```

Analyzing Document Similarity

Okapi BM25 Ranking

```
# Get the corpus term idfs
corpus_term_idfs = compute_corpus_term_idfs(corpus_features,
                                            norm_corpus)

# analyze document similarity using BM25 framework
In [253]: print 'Document Similarity Analysis using BM25'
     ...: print '='*60
     ...: for index, doc in enumerate(query_docs):
     ...:
     ...:     doc_features = query_docs_features[index]
     ...:     top_similar_docs = compute_bm25_similarity(doc_features,
     ...:                                                corpus_features,
     ...:                                                doc_lengths,
     ...:                                                avg_dl,
     ...:                                                corpus_term_idfs
     ...:                                                k1=1.5, b=0.75,
     ...:                                                top_n=2)
     ...:     print 'Document',index+1 ,':', doc
     ...:     print 'Top', len(top_similar_docs), 'similar docs:'
     ...:     print '-'*40
     ...:     for doc_index, sim_score in top_similar_docs:
     ...:         print 'Doc num: {} BM25 Score: {}\nDoc: {}'.format(doc_
                  index+1,
     ...:                                         sim_score, toy_corpus[doc_
                                                  index])
     ...:         print '-'*40
     ...:     print
```

```
Document Similarity Analysis using BM25
============================================================
Document 1 : The fox is definitely smarter than the dog
Top 2 similar docs:
----------------------------------------
Doc num: 8 BM25 Score: 7.334
Doc: The dog is smarter than the fox
----------------------------------------
Doc num: 7 BM25 Score: 3.88
Doc: The fox is quicker than the lazy dog
----------------------------------------

Document 2 : Java is a static typed programming language unlike Python
Top 2 similar docs:
----------------------------------------
Doc num: 5 BM25 Score: 7.248
Doc: Python and Java are popular Programming languages
----------------------------------------
Doc num: 6 BM25 Score: 6.042
Doc: Among Programming languages, both Python and Java are the most used in
Analytics
----------------------------------------

Document 3 : I love to relax under the beautiful blue sky!
Top 2 similar docs:
----------------------------------------
Doc num: 2 BM25 Score: 7.334
Doc: The sky is blue and beautiful
----------------------------------------
Doc num: 1 BM25 Score: 4.984
Doc: The sky is blue
----------------------------------------
```

Document Clustering

- Document clustering or cluster analysis is an interesting area in NLP and text analytics that applies unsupervised ML concepts and techniques.
- Document clustering uses unsupervised ML algorithms to group the documents into various clusters
- 1. Build a function to get inverse document frequency (IDF) values for terms in corpus.

2. Build a function for computing BM25 scores for query document and corpus documents.

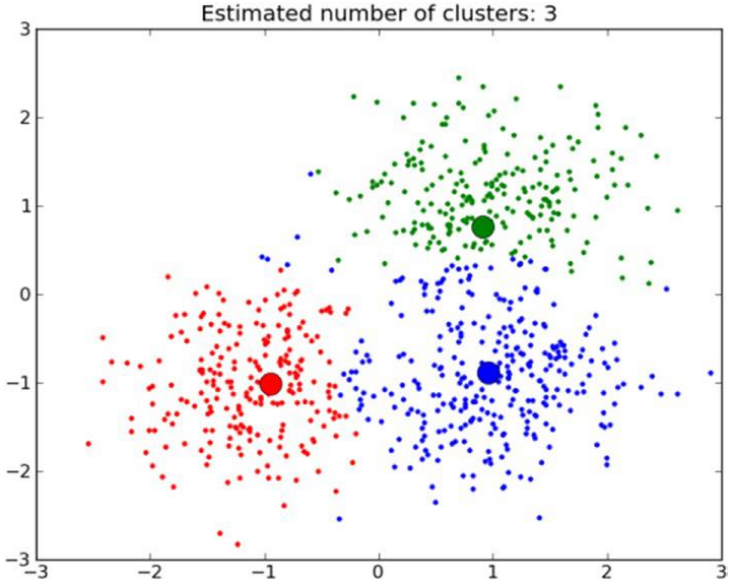
3. Get Bag of Words–based features for corpus documents and query documents.

4. Compute average length of corpus documents and IDFs of the terms in the corpus documents using function from point 1.

5. Compute BM25 scores, rank relevant documents, and fetch the n most relevant documents for each query document using the function in point 2.

Document Clustering

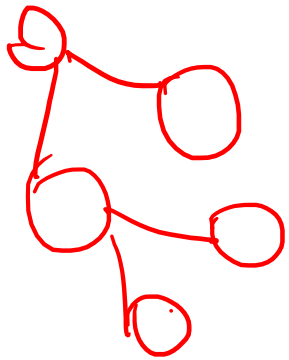
- Courtesy of scikit-learn, visualizes an example of clustering data points into three clusters based on its features.



- It is pretty clear that there will always be some overlap among the clusters because there is no such definition of a perfect cluster.
- All the techniques are based on math, heuristics, and some inherent attributes toward generating clusters, and they are never a 100 percent perfect.

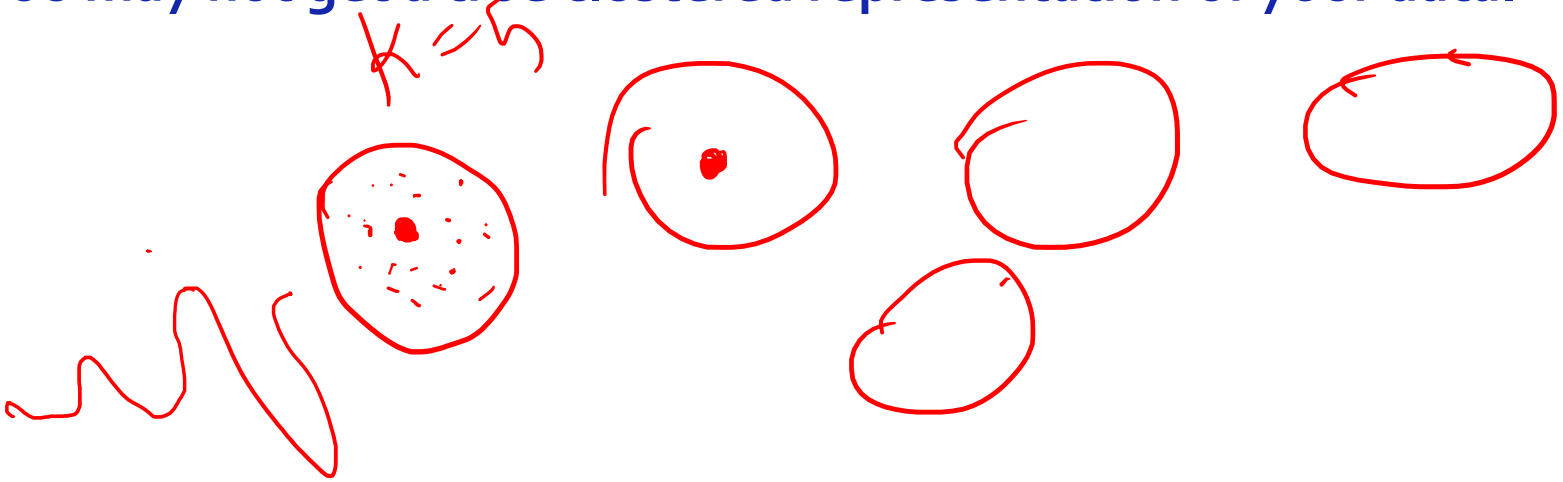
Document Clustering

- **Hierarchical clustering models:** These clustering models are also known as connectivity-based clustering methods and are based on the concept that similar objects will be closer to related objects in the vector space than unrelated objects, which will be farther away from them.
- Clusters are formed by connecting objects based on their distance and they can be visualized using a dendrogram.
- The output of these models is a complete, exhaustive hierarchy of clusters.
- They are mainly subdivided into agglomerative and divisive clustering models.



Document Clustering

- **Centroid-based clustering models:** These models build clusters in such a way that each cluster has a central representative member that represents each cluster and has the features that distinguish that particular cluster from the rest.
- There are various algorithms in this, like k-means, k-medoids, and so on, where we need to set the number of clusters 'k' in advance, and distance metrics like squares of distances from each data point to the centroid need to be minimized.
- The disadvantage of these models is that you need to specify the 'k' number of clusters in advance, which may lead to local minima, and you may not get a true clustered representation of your data.



Document Clustering

- **Distribution-based clustering models:** These models make use of concepts from probability distributions when clustering data points.
- The idea is that objects having similar distributions can be clustered into the same group or cluster.
- Gaussian mixture models (GMM) use algorithms like the Expectation-Maximization algorithm for building these clusters.
- Feature and attribute correlations and dependencies can also be captured using these models, but it is prone to overfitting.
- **Density-based clustering models:** These clustering models generate clusters from data points that are grouped together at areas of high density compared to the rest of the data points, which may occur randomly across the vector space in sparsely populated areas.
- These sparse areas are treated as noise and are used as border points to separate clusters.
- Two popular algorithms in this area include DBSCAN and OPTICS.

Several other clustering models have been recently introduced, including algorithms like BIRCH and CLARANS

Clustering Greatest Movies of All Time

- We will be clustering a total of 100 different popular movies based on their IMDb synopses as our raw data.
- IMDb, also known as the Internet Movie Database (www.imdb.com), is an online database that hosts extensive detailed information about movies, video games, and television shows.
- It also aggregates reviews and synopses for movies and shows and has several curated lists.
- The list we are interested in is available at www.imdb.com/list/ls055592025/, titled Top 100 Greatest Movies of All Time (The Ultimate List).

```
import pandas as pd
import numpy as np

# load movie data
movie_data = pd.read_csv('movie_data.csv')

# view movie data
In [256]: print movie_data.head()

                    Title                                      Synopsis
0            The Godfather  In late summer 1945, guests are gathered...
1  The Shawshank Redemption  In 1947, Andy Dufresne (Tim Robbins),...
2         Schindler's List  The relocation of Polish Jews from...
3              Raging Bull  The film opens in 1964, where an older...
4               Casablanca  In the early years of World War II...

# print sample movie and its synopsis
In [268]: print 'Movie:', movie_titles[0]
     ...: print 'Movie Synopsis:', movie_synopses[0][:1000]
     ...:
Movie: The Godfather
```

Clustering Greatest Movies of All Time

Movie Synopsis: In late summer 1945, guests are gathered for the wedding reception of Don Vito Corleone's daughter Connie (Talia Shire) and Carlo Rizzi (Gianni Russo). Vito (Marlon Brando), the head of the Corleone Mafia family, is known to friends and associates as "Godfather." He and Tom Hagen (Robert Duvall), the Corleone family lawyer, are hearing requests for favors because, according to Italian tradition, "no Sicilian can refuse a request on his daughter's wedding day." One of the men who asks the Don for a favor is Amerigo Bonasera, a successful mortician and acquaintance of the Don, whose daughter was brutally beaten by two young men because she refused their advances; the men received minimal punishment. The Don is disappointed in Bonasera, who'd avoided most contact with the Don due to Corleone's nefarious business dealings. The Don's wife is godmother to Bonasera's shamed daughter, a relationship the Don uses to extract new loyalty from the undertaker. The Don agrees to have his men punish

We will extract features from these synopses and use unsupervised learning algorithms on them to cluster them together.

```
from normalization import normalize_corpus
from utils import build_feature_matrix

# normalize corpus
norm_movie_synopses = normalize_corpus(movie_synopses,
                                       lemmatize=True,
                                       only_text_chars=True)

# extract tf-idf features
vectorizer, feature_matrix = build_feature_matrix(norm_movie_synopses,
                                                  feature_type='tfidf',
                                                  min_df=0.24, max_df=0.85,
                                                  ngram_range=(1, 2))
# view number of features
In [275]: print feature_matrix.shape
(100, 307)

# get feature names
feature_names = vectorizer.get_feature_names()
# print sample features
In [277]: print feature_names[:20]
```

K- Means Clustering

- The k-means clustering algorithm is a centroid-based clustering model that tries to cluster data into groups or clusters of equal variance.
- The criteria or measure that this algorithm tries to minimize is inertia, also known as within-cluster sum-of-squares.
- Perhaps the one main disadvantage of this algorithm is that the number of clusters k need to be specified in advance, as is the case with all other centroid-based clustering models.
- This algorithm is perhaps the most popular clustering algorithm out there and is frequently used due to its ease of use as well as the fact that it is scalable with large amounts of data.

K- Means Clustering

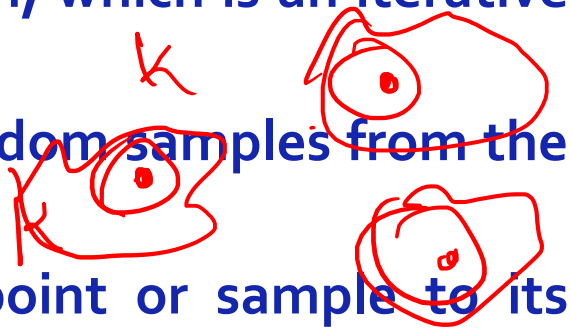
- Consider that we have a dataset X with N data points or samples and we want to group them into K clusters where K is a user-specified parameter.
- The k-means clustering algorithm will segregate the N data points into K disjoint separate clusters Ck, and each of these clusters can be described by the means of the cluster samples.
- These means become the cluster centroids μk such that these centroids are not bound by the condition that they have to be actual data points from the N samples in X.
- The algorithm chooses these centroids and builds the clusters in such a way that the inertia or within-cluster sums of squares are minimized.

$$min \sum_{i=1}^{k} \sum_{x_n \in C_i} \|x_n - \mu_i\|^2$$

- with regard to clusters Ci and centroids μi such that $i \in \{1, 2, \ldots, k\}$.
- This optimization is an NP hard problem for all you algorithm enthusiasts out there.

K- Means Clustering

- Lloyd's algorithm is a solution to this problem, which is an iterative procedure consisting of the following steps.



1. Choose initial k centroids μk by taking k random samples from the dataset X.

2. Update clusters by assigning each data point or sample to its nearest centroid point. Mathematically, we can represent this as $C_k = \{x_n : \|x_n - \mu_k\| \leq all \|x_n - \mu_l\|\}$ where Ck denotes the clusters.

3. Recalculate and update clusters based on the new cluster data points for each cluster obtained from step 2. Mathematically, this can be represented as $\mu_k = \frac{1}{C_k} \sum_{x_n \in C_k} x_n$ where μk denotes the centroids.

- One caveat of this method is that even though the optimization is guaranteed to converge, it might lead to a local minimum, hence in reality, this algorithm is run multiple times with several epochs and iterations, and the results might be averaged from them if needed.
- The convergence and occurrence of local minimum are highly dependent on the initialization of the initial centroids in step 1.
- One way is to make multiple iterations with multiple random initializations and take the average.

K- Means Clustering

```
from sklearn.cluster import KMeans
# define the k-means clustering function
def k_means(feature_matrix, num_clusters=5):
    km = KMeans(n_clusters=num_clusters,
                max_iter=10000)
    km.fit(feature_matrix)
    clusters = km.labels_
    return km, clusters
# set k = 5, lets say we want 5 clusters from the 100 movie
num_clusters = 5

# get clusters and assigned the cluster labels to the movie
km_obj, clusters = k_means(feature_matrix=feature_matrix,
                           num_clusters=num_clusters)
movie_data['Cluster'] = clusters
```

```
In [284]: from collections import Counter
     ...: # get the total number of movies per cluster
     ...: c = Counter(clusters)
     ...: print c.items()
[(0, 29), (1, 5), (2, 21), (3, 15), (4, 30)]
```

```
def get_cluster_data(clustering_obj, movie_data,
                     feature_names, num_clusters,
                     topn_features=10):

    cluster_details = {}
    # get cluster centroids
    ordered_centroids = clustering_obj.cluster_centers_.argsort()[:, ::-1]
    # get key features for each cluster
    # get movies belonging to each cluster
    for cluster_num in range(num_clusters):
        cluster_details[cluster_num] = {}
        cluster_details[cluster_num]['cluster_num'] = cluster_num
        key_features = [feature_names[index]
                        for index
                        in ordered_centroids[cluster_num, :topn_features]]
        cluster_details[cluster_num]['key_features'] = key_features

        movies = movie_data[movie_data['Cluster'] == cluster_num]['Title'].
        values.tolist()
        cluster_details[cluster_num]['movies'] = movies

    return cluster_details
```

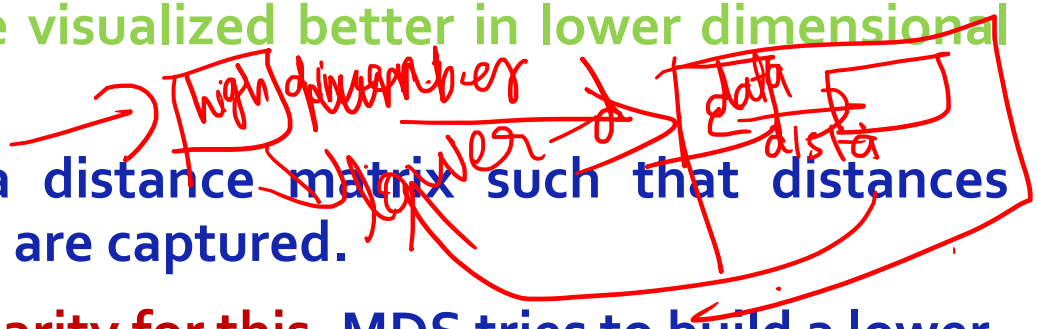
K- Means Clustering

```
def print_cluster_data(cluster_data):
    # print cluster details
    for cluster_num, cluster_details in cluster_data.items():
        print 'Cluster {} details:'.format(cluster_num)
        print '-'*20
        print 'Key features:', cluster_details['key_features']
        print 'Movies in this cluster:'
        print ', '.join(cluster_details['movies'])
        print '='*40
```

- multidimensional feature spaces and unstructured text data.
- Numeric feature vectors themselves may not make any sense to readers if they were visualized directly.
- So, there are some techniques like principal component analysis (PCA) or multidimensional scaling (MDS) to reduce the dimensionality such that we can visualize these clusters in 2- or 3-dimensional plots.

K- Means Clustering

- MDS is an approach towards non-linear dimensionality reduction such that the results can be visualized better in lower dimensional systems.
- The main idea is having a distance matrix such that distances between various data points are captured.
- We will be using Cosine similarity for this. MDS tries to build a lower-dimensional representation of our data with higher numbers of features in the vector space such that the distances between the various data points obtained using Cosine similarity in the higher dimensional feature space is still similar in this lower-dimensional representation.
- The scikit-learn implementation for MDS has two types of algorithms: metric and non-metric.
- We will be using the metric approach because we will use the Cosine similarity–based distance metric to build the input similarity matrix between the various movies.



K- Means Clustering

- Mathematically, MDS can be defined as follows: Let S be our similarity matrix between the various data points (movies) obtained using Cosine similarity on the feature matrix and X be the coordinates of the n input data points (movies).
- Disparities are represented by $\hat{d}_{ij} = t(S_{ij})$, which is usually some optimal transformation of the similarity values or could even be the raw similarity values themselves.
- The objective function for MDS, called stress, is defined as sum

$$sum_{i<j} d_{ij}(X) - \hat{d}_{ij}(X)$$

```
import matplotlib.pyplot as plt
from sklearn.manifold import MDS
from sklearn.metrics.pairwise import cosine_similarity
import random
from matplotlib.font_manager import FontProperties

def plot_clusters(num_clusters, feature_matrix,
                  cluster_data, movie_data,
                  plot_size=(16,8)):
    # generate random color for clusters
    def generate_random_color():
        color = '#%06x' % random.randint(0, 0xFFFFFF)
        return color
    # define markers for clusters
    markers = ['o', 'v', '^', '<', '>', '8', 's', 'p', '*', 'h', 'H', 'D', 'd']
    # build cosine distance matrix
    cosine_distance = 1 - cosine_similarity(feature_matrix)
    # dimensionality reduction using MDS
    mds = MDS(n_components=2, dissimilarity="precomputed",
              random_state=1)
    # get coordinates of clusters in new low-dimensional space
    plot_positions = mds.fit_transform(cosine_distance)
    x_pos, y_pos = plot_positions[:, 0], plot_positions[:, 1]
    # build cluster plotting data
```

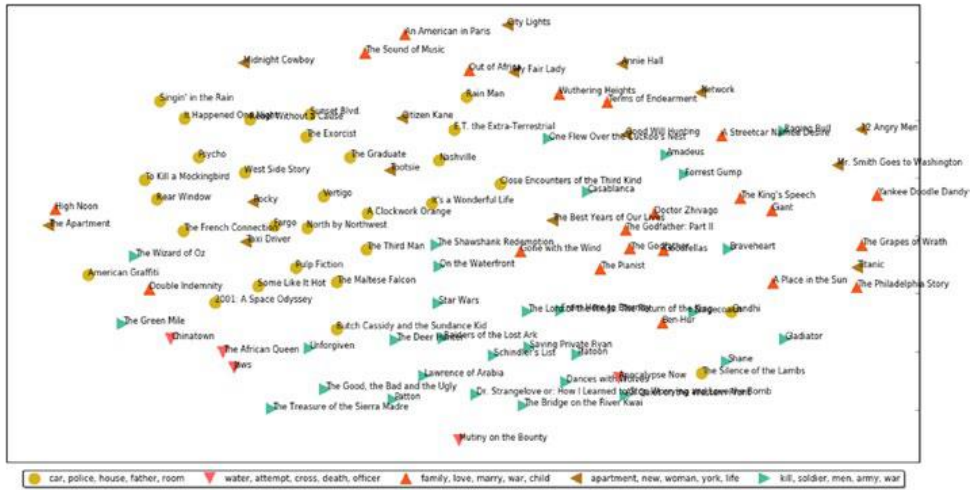
K- Means Clustering

```
cluster_color_map = {}
cluster_name_map = {}
for cluster_num, cluster_details in cluster_data.items():
    # assign cluster features to unique label
    cluster_color_map[cluster_num] = generate_random_color()
    cluster_name_map[cluster_num] = ', '.join(cluster_details['key_
    features'][:5]).strip()
# map each unique cluster label with its coordinates and movies
cluster_plot_frame = pd.DataFrame({'x': x_pos,
                                   'y': y_pos,
                                   'label': movie_data['Cluster'].
                                   values.tolist(),
                                   'title': movie_data['Title'].values.
                                   tolist()
                                    })
grouped_plot_frame = cluster_plot_frame.groupby('label')
# set plot figure size and axes
fig, ax = plt.subplots(figsize=plot_size)
ax.margins(0.05)
# plot each cluster using co-ordinates and movie titles
for cluster_num, cluster_frame in grouped_plot_frame:
     marker = markers[cluster_num] if cluster_num < len(markers) \
              else np.random.choice(markers, size=1)[0]
     ax.plot(cluster_frame['x'], cluster_frame['y'],
             marker=marker, linestyle='', ms=12,
             label=cluster_name_map[cluster_num],
             color=cluster_color_map[cluster_num], mec='none')
     ax.set_aspect('auto')
     ax.tick_params(axis= 'x', which='both', bottom='off', top='off',
                    labelbottom='off')
     ax.tick_params(axis= 'y', which='both', left='off', top='off',
                    labelleft='off')
fontP = FontProperties()
fontP.set_size('small')
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.01),
fancybox=True,
          shadow=True, ncol=5, numpoints=1, prop=fontP)
#add labels as the film titles
for index in range(len(cluster_plot_frame)):
    ax.text(cluster_plot_frame.ix[index]['x'],
            cluster_plot_frame.ix[index]['y'],
            cluster_plot_frame.ix[index]['title'], size=8)
# show the plot
plt.show()
```

```
# get clustering analysis data
cluster_data =  get_cluster_data(clustering_obj=km_obj, movie_data=movie_
data,
                                 feature_names=feature_names, num_
                                 clusters=num_clusters,
                                 topn_features=5)

# print clustering analysis results
In [294]: print_cluster_data(cluster_data)

Cluster 0 details:
------------------
Key features: [u'car', u'police', u'house', u'father', u'room']
Movies in this cluster:
Psycho, Sunset Blvd., Vertigo, West Side Story, E.T. the Extra-Terrestrial,
2001: A Space Odyssey, The Silence of the Lambs, Singin' in the Rain, It's
a Wonderful Life, Some Like It Hot, Gandhi, To Kill a Mockingbird, Butch
Cassidy and the Sundance Kid, The Exorcist, The French Connection, It
Happened One Night, Rain Man, Fargo, Close Encounters of the Third Kind,
Nashville, The Graduate, American Graffiti, Pulp Fiction, The Maltese
Falcon, A Clockwork Orange, Rebel Without a Cause, Rear Window, The Third
Man, North by Northwest
====================================
Cluster 1 details:
------------------
Key features: [u'water', u'attempt', u'cross', u'death', u'officer']
Movies in this cluster:
Chinatown, Apocalypse Now, Jaws, The African Queen, Mutiny on the Bounty
====================================
Cluster 2 details:
------------------
Key features: [u'family', u'love', u'marry', u'war', u'child']
Movies in this cluster:
The Godfather, Gone with the Wind, The Godfather: Part II, The Sound of
Music, A Streetcar Named Desire, The Philadelphia Story, An American in
Paris, Ben-Hur, Doctor Zhivago, High Noon, The Pianist, Goodfellas, The
King's Speech, A Place in the Sun, Out of Africa, Terms of Endearment,
Giant, The Grapes of Wrath, Wuthering Heights, Double Indemnity, Yankee
Doodle Dandy
```

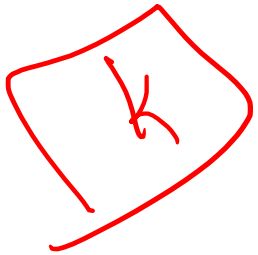
K- Means Clustering

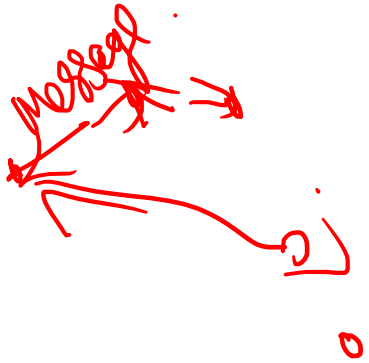
```
Cluster 3 details:
--------------------
Key features: [u'apartment', u'new', u'woman', u'york', u'life']
Movies in this cluster:
Citizen Kane, Titanic, 12 Angry Men, Rocky, The Best Years of Our Lives, My
Fair Lady, The Apartment, City Lights, Midnight Cowboy, Mr. Smith Goes to
Washington, Annie Hall, Good Will Hunting, Tootsie, Network, Taxi Driver
====================================
Cluster 4 details:
--------------------
Key features: [u'kill', u'soldier', u'men', u'army', u'war']
Movies in this cluster:
The Shawshank Redemption, Schindler's List, Raging Bull, Casablanca, One
Flew Over the Cuckoo's Nest, The Wizard of Oz, Lawrence of Arabia, On the
Waterfront, Forrest Gump, Star Wars, The Bridge on the River Kwai, Dr.
Strangelove or: How I Learned to Stop Worrying and Love the Bomb, Amadeus,
The Lord of the Rings: The Return of the King, Gladiator, From Here to
Eternity, Saving Private Ryan, Unforgiven, Raiders of the Lost Ark, Patton,
Braveheart, The Good, the Bad and the Ugly, The Treasure of the Sierra
Madre, Platoon, Dances with Wolves, The Deer Hunter, All Quiet on the
Western Front, Shane, The Green Mile, Stagecoach
====================================

# visualize the clusters
In [295]: plot_clusters(num_clusters=num_clusters,
     ...:               feature_matrix=feature_matrix,
     ...:               cluster_data=cluster_data,
     ...:               movie_data=movie_data,
     ...:               plot_size=(16,8))
```



Affinity Propagation

- The k-means algorithm, although very popular, has the drawback that the user has to predefine the number of clusters.
- What if in reality there are more clusters or lesser clusters?
- There are some ways of checking the cluster quality and seeing what the value of the optimum k might be.
- The silhouette coefficient, which are popular methods of determining the optimum k.
- The affinity propagation (AP) algorithm is based on the concept of "message passing" among the various data points to be clustered, and no pre-assumption is needed about the number of possible clusters.





Affinity Propagation

- AP creates these clusters from the data points by passing messages between pairs of data points until convergence is achieved.
- The entire dataset is then represented by a small number of exemplars that act as representatives for samples.
- These exemplars are analogous to the centroids you obtain from k-means or k-medoids.
- The messages that are sent between pairs represent how suitable one of the points might be in being the exemplar or representative of the other data point.
- This keeps getting updated in every iteration until convergence is achieved, with the final exemplars being the representatives of each cluster.
- Remember, one drawback of this method is that it is computationally intensive because messages are passed between each pair of data points across the entire dataset and can take substantial time to converge for large datasets.

Affinity Propagation

- Consider that we have a dataset X with n data points such that $X=\{x_1, x_2, \ldots, x_n\}$, let sim(x, y) be the similarity function that quantifies the similarity between two points x and y.
- The AP algorithm iteratively proceeds by executing two message-passing steps as follows:

1. **Responsibility updates** are sent around, which can be mathematically represented as $r(i,k) \leftarrow sim(i,k) - \max_{k' \neq k}\{a(i,k') + sim(i,k')\}$

- where the responsibility matrix is R and r(i, k) is a measure which quantifies how well xk can serve as being the representative or exemplar for xi in comparison to the other candidates.

2. **Availability updates** are then sent around which can be mathematically represented as $a(i,k) \leftarrow \min\left(0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k))\right)$ for $i \neq k$ and

availability for $i = k$ is represented as

$$a(k,k) \leftarrow \sum_{i' \neq k} \max(0, r(i',k))$$

where the availability matrix is *A* and *a*(*i*, *k*) represents how appropriate it would be for x_i to pick x_k as its exemplar, considering all the other points' preference to pick x_k as an exemplar.

Affinity Propagation

```
from sklearn.cluster import AffinityPropagation

def affinity_propagation(feature_matrix):
    sim = feature_matrix * feature_matrix.T
    sim = sim.todense()
    ap = AffinityPropagation()
    ap.fit(sim)
    clusters = ap.labels_
    return ap, clusters
```

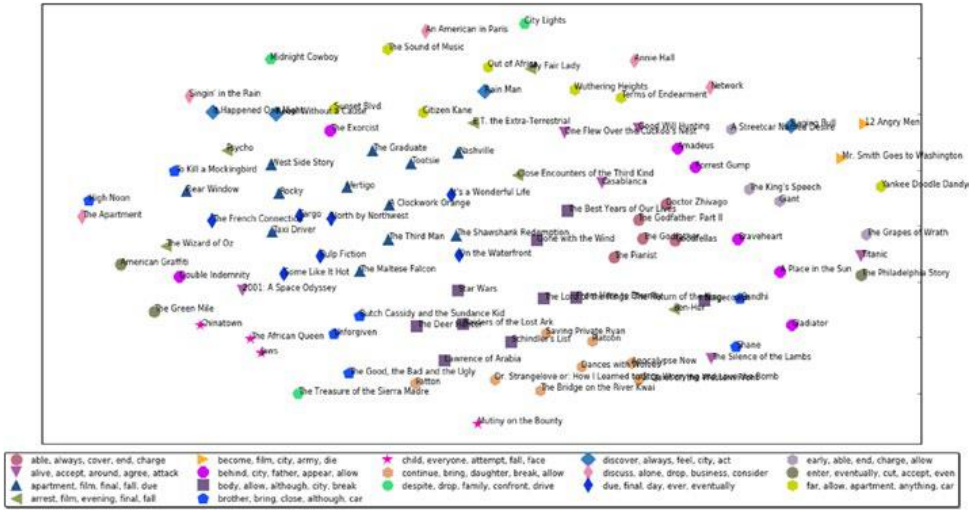
We will now use this function to cluster our movies based on their synopses and then we will print the number of movies in each cluster and the total number of clusters formed by this algorithm:

```
# get clusters using affinity propagation
ap_obj, clusters = affinity_propagation(feature_matrix=feature_matrix)
movie_data['Cluster'] = clusters

# get the total number of movies per cluster
In [299]: c = Counter(clusters)
     ...: print c.items()
[(0, 5), (1, 6), (2, 12), (3, 6), (4, 2), (5, 7), (6, 10), (7, 7), (8, 4),
(9, 8), (10, 3), (11, 4), (12, 5), (13, 7), (14, 4), (15, 3), (16, 7)]

# get total clusters
In [300]: total_clusters = len(c)
     ...: print 'Total Clusters:', total_clusters
Total Clusters: 17
```

```
# get clustering analysis data
cluster_data =  get_cluster_data(clustering_obj=ap_obj, movie_data=movie_
data,
                                 feature_names=feature_names, num_
                                 clusters=total_clusters,
                                 topn_features=5)

# print clustering analysis results
In [302]: print_cluster_data(cluster_data)
     ...:
Cluster 0 details:
------------------
Key features: [u'able', u'always', u'cover', u'end', u'charge']
Movies in this cluster:
The Godfather, The Godfather: Part II, Doctor Zhivago, The Pianist,
Goodfellas
====================================
Cluster 1 details:
------------------
Key features: [u'alive', u'accept', u'around', u'agree', u'attack']
Movies in this cluster:
Casablanca, One Flew Over the Cuckoo's Nest, Titanic, 2001: A Space Odyssey,
The Silence of the Lambs, Good Will Hunting
====================================
Cluster 2 details:
------------------
Key features: [u'apartment', u'film', u'final', u'fall', u'due']
Movies in this cluster:
The Shawshank Redemption, Vertigo, West Side Story, Rocky, Tootsie,
Nashville, The Graduate, The Maltese Falcon, A Clockwork Orange, Taxi
Driver, Rear Window, The Third Man
====================================
Cluster 3 details:
------------------
Key features: [u'arrest', u'film', u'evening', u'final', u'fall']
Movies in this cluster:
The Wizard of Oz, Psycho, E.T. the Extra-Terrestrial, My Fair Lady, Ben-Hur,
Close Encounters of the Third Kind
====================================
```

Affinity Propagation

```
====================================
Cluster 12 details:
--------------------
Key features: [u'discuss', u'alone', u'drop', u'business', u'consider']
Movies in this cluster:
Singin' in the Rain, An American in Paris, The Apartment, Annie Hall,
Network
====================================
Cluster 13 details:
--------------------
Key features: [u'due', u'final', u'day', u'ever', u'eventually']
Movies in this cluster:
On the Waterfront, It's a Wonderful Life, Some Like It Hot, The French
Connection, Fargo, Pulp Fiction, North by Northwest
====================================
Cluster 14 details:
--------------------
Key features: [u'early', u'able', u'end', u'charge', u'allow']
Movies in this cluster:
A Streetcar Named Desire, The King's Speech, Giant, The Grapes of Wrath
====================================
Cluster 15 details:
--------------------
Key features: [u'enter', u'eventually', u'cut', u'accept', u'even']
Movies in this cluster:
The Philadelphia Story, The Green Mile, American Graffiti
====================================
Cluster 16 details:
--------------------
Key features: [u'far', u'allow', u'apartment', u'anything', u'car']
Movies in this cluster:
Citizen Kane, Sunset Blvd., The Sound of Music, Out of Africa, Terms of
Endearment, Wuthering Heights, Yankee Doodle Dandy
====================================

# visualize the clusters
In [304]: plot_clusters(num_clusters=num_clusters, feature_matrix=feature_
matrix,
     ...:               cluster_data=cluster_data, movie_data=movie_data,
     ...:               plot_size=(16,8))
```



Ward's Agglomerative Hierarchical Clustering

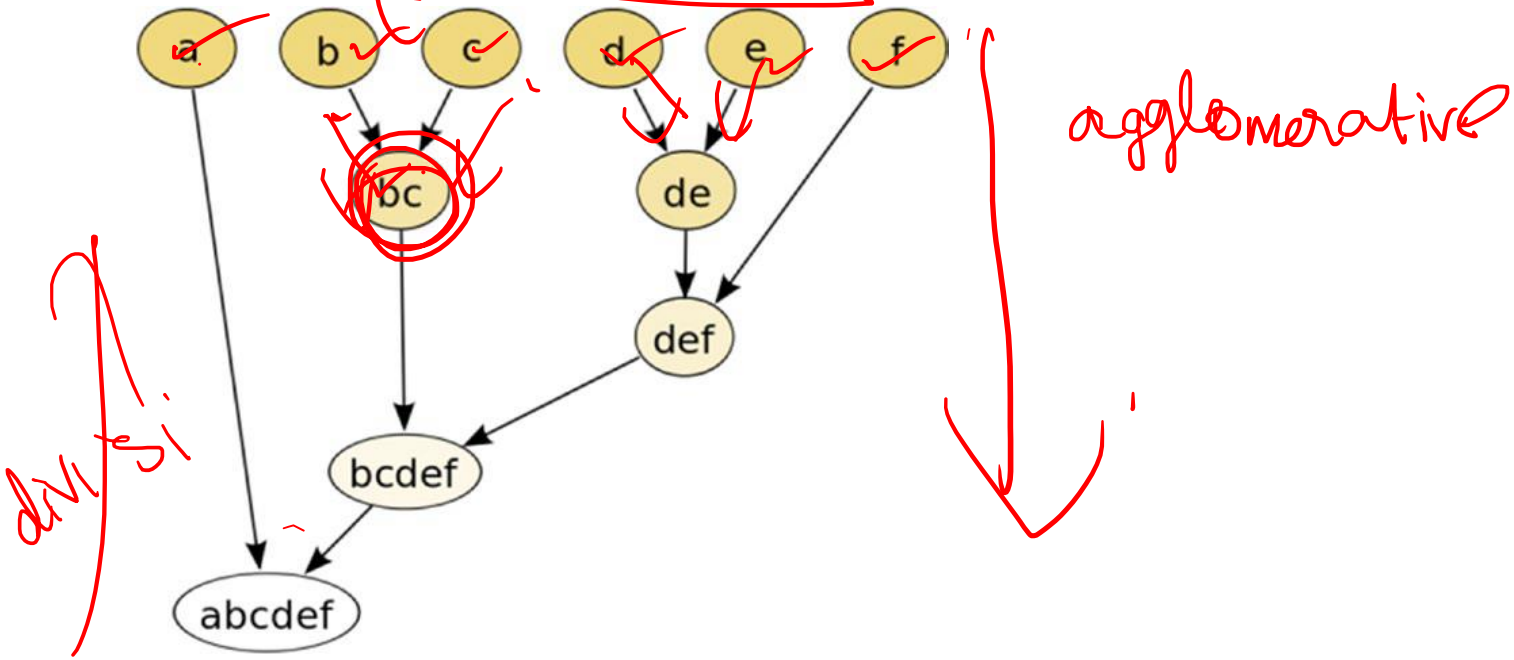
- Hierarchical clustering tries to build a nested hierarchy of clusters by either merging or splitting them in succession.
- There are two main strategies for Hierarchical clustering:

- **Agglomerative:** These algorithms follow a bottom-up approach where initially all data points belong to their own individual cluster, and then from this bottom layer, we start merging clusters together, building a hierarchy of clusters as we go up.

Divisive: These algorithms follow a top-down approach where initially all the data points belong to a single huge cluster and then we start recursively dividing them up as we move down gradually, and this produces a hierarchy of clusters going from the top-down.

Ward's Agglomerative Hierarchical Clustering

- Merges and splits normally happen using a greedy algorithm, and the end result of the hierarchy of clusters can be visualized as a tree structure, called a dendrogram.



highlights how six separate data points start off as six clusters, and then we slowly start grouping them in each step following a bottom-up approach.

Ward's Agglomerative Hierarchical Clustering

- For deciding which clusters we should combine when starting from the individual data point clusters, we need two things:

- A distance metric to measure the similarity or dissimilarity degree between data points

- A linkage criterion that determines the metric to be used for the merging strategy of clusters. We will be using Ward's method here.

- The Ward's linkage criterion minimizes the sum of squared differences within all the clusters and is a variance minimizing approach.

- The idea is to minimize the variances within each cluster using an objective function like the L2 norm distance between two points.

- We can start with computing the initial cluster distances between each pair of points using the formula $d_{ij} = d(\{C_i, C_j\}) = \|C_i - C_j\|^2$

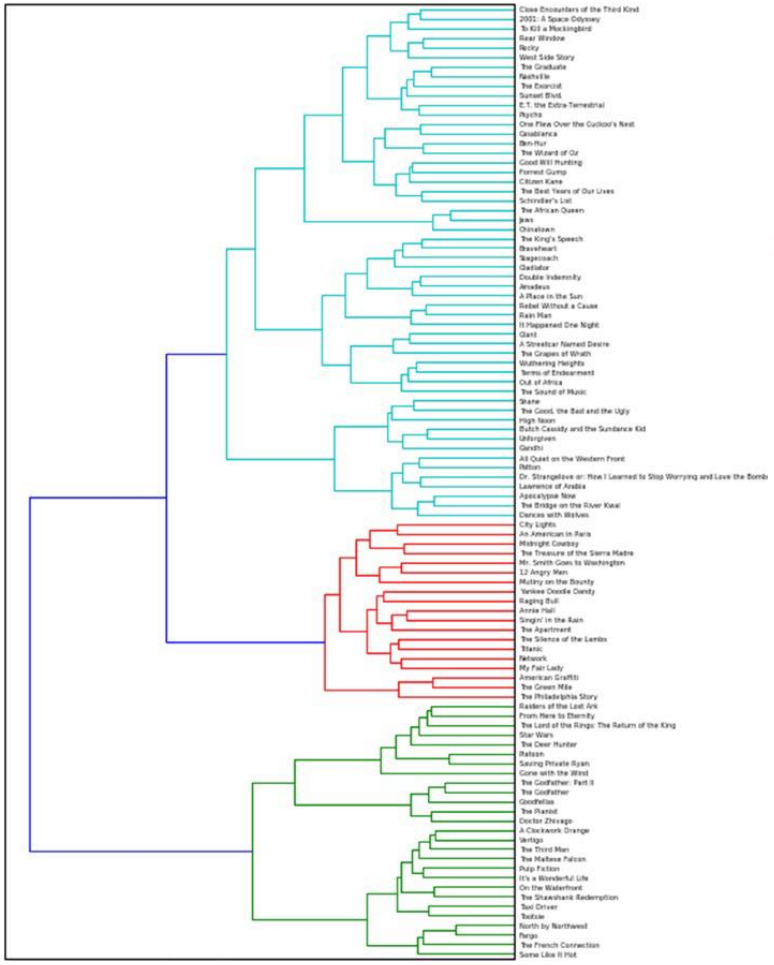
- where initially Ci indicates cluster i with one document, and at each iteration, we find the pairs of clusters that lead to the least increase in variance for that cluster once merged.

- A weighted squared Euclidean distance or L2 norm as depicted in the preceding formula would suffice for this algorithm.

- We use Cosine similarity to compute the cosine distances between each pair of movies for our dataset

Ward's Agglomerative Hierarchical Clustering

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

def ward_hierarchical_clustering(feature_matrix):

    cosine_distance = 1 - cosine_similarity(feature_matrix)
    linkage_matrix = ward(cosine_distance)
    return linkage_matrix
```



```
def plot_hierarchical_clusters(linkage_matrix, movie_data, figure_
size=(8,12)):
    # set size
    fig, ax = plt.subplots(figsize=figure_size)
    movie_titles = movie_data['Title'].values.tolist()
    # plot dendrogram
    ax = dendrogram(linkage_matrix, orientation="left", labels=movie_titles)
    plt.tick_params(axis= 'x',
                    which='both',
                    bottom='off',
                    top='off',
                    labelbottom='off')
    plt.tight_layout()
    plt.savefig('ward_hierachical_clusters.png', dpi=200)
```

```
In [307]:# build ward's linkage matrix
    ...:linkage_matrix = ward_hierarchical_clustering(feature_matrix)
    ...: # plot the dendrogram
    ...: plot_hierarchical_clusters(linkage_matrix=linkage_matrix,
    ...:                            movie_data=movie_data,
    ...:                            figure_size=(8,10))
```

Semantic Analysis

- Text semantics specifically deals with understanding the meaning of text or language.
- When combined into sentences, words have lexical relations and contextual relations between them lead to various types of relationships and hierarchies, and semantics sits at the heart of all this in trying to analyze and understand these relationships and infer meaning from them.
- Semantics is purely concerned with context and meaning, and the structure or format of text holds little significance here.
- But sometimes even the syntax or arrangement of words helps us in inferring the context of words and helps us differentiate things like lead as a metal from lead as in the lead of a movie.

Semantic Analysis

- Exploring WordNet and synsets
- Analyzing lexical semantic relations
- Word sense disambiguation
- Named entity recognition
- Analyzing semantic representations

Exploring WordNet

- WordNet is a huge lexical database for the English Language.
- The database is a part of Princeton University, https://wordnet.princeton.edu .
- This lexical database consists of nouns, adjective, verbs, and adverbs, and related lexical terms are grouped together based on some common concepts into sets, known as cognitive synonym sets or synsets.
- Each synset expresses a unique, distinct concept. At a high level, WordNet can be compared to a thesaurus or a dictionary that provides words and their synonyms.
- On a lower level, it is much more than that, with synsets and their corresponding terms having detailed relationships and hierarchies based on their semantic meaning and similar concepts.
- WordNet is used extensively as a lexical database, in text analytics, NLP, and artificial intelligence (AI)-based applications.
- The WordNet database consists of over 155,000 words, represented in more than 117,000 synsets, and contains over 206,000 word-sense pairs.
- The database is roughly 12 MB in size and can be accessed through various interfaces and APIs.

Exploring WordNet-Understanding Synsets

- A synset is a collection or set of data entities that are considered to be semantically similar.
- Specifically in the context of WordNet, a synset is a set or collection of synonyms that are interchangeable and revolve around a specific concept.
- Synsets not only consist of simple words, but also collocations.
- Polysemous word forms (words that sound and look the same but have different but relatable meanings) are assigned to different synsets based on their meaning.
- Typically each synset has the term, a definition explaining the meaning of the term, and some optional examples and related lemmas (collection of synonyms) to the term.
- Some terms may have multiple synsets associated with them, where each synset has a particular context.

Exploring WordNet-Understanding Synsets

- nltk 's WordNet interface to explore synsets associated with the term, 'fruit'

The output shows us details pertaining to each synset associated with the term 'fruit', and the definitions give us the sense of each synset and the lemma associated with it.

The part of speech for each synset is also mentioned, which includes nouns and verbs.

```
from nltk.corpus import wordnet as wn
import pandas as pd

term = 'fruit'
synsets = wn.synsets(term)
# display total synsets
In [75]: print 'Total Synsets:', len(synsets)
Total Synsets: 5
```

We can see that there are a total of five synsets associated with the term 'fruit'. What can these synsets indicate? We can dig deeper into each synset and its components using the following code snippet:

```
In [76]: for synset in synsets:
    ...:     print 'Synset:', synset
    ...:     print 'Part of speech:', synset.lexname()
    ...:     print 'Definition:', synset.definition()
    ...:     print 'Lemmas:', synset.lemma_names()
    ...:     print 'Examples:', synset.examples()
    ...:     print
    ...:
    ...:
Synset: Synset('fruit.n.01')
Part of speech: noun.plant
Definition: the ripened reproductive body of a seed plant
Lemmas: [u'fruit']
Examples: []

Synset: Synset('yield.n.03')
Part of speech: noun.artifact
Definition: an amount of a product
Lemmas: [u'yield', u'fruit']
Examples: []

Synset: Synset('fruit.n.03')
Part of speech: noun.event
Definition: the consequence of some effort or action
Lemmas: [u'fruit']
Examples: [u'he lived long enough to see the fruit of his policies']

Synset: Synset('fruit.v.01')
Part of speech: verb.creation
Definition: cause to bear fruit
Lemmas: [u'fruit']
Examples: []

Synset: Synset('fruit.v.02')
Part of speech: verb.creation
Definition: bear fruit
Lemmas: [u'fruit']
Examples: [u'the trees fruited early this year']
```

Exploring WordNet-Analyzing Lexical Semantic Relations

- Text semantics refers to the study of meaning and context.
- Synsets give a nice abstraction over various terms and provide useful information like definition, examples, POS, and lemmas.

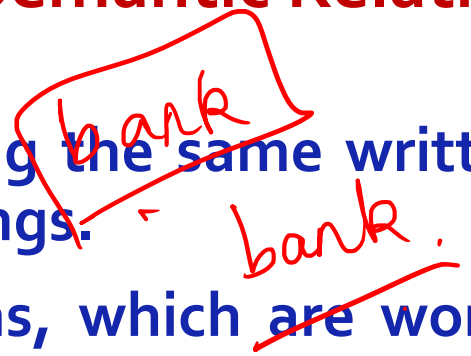
Entailments

- *Entailment* usually refers to some event or action that logically involves or is associated with some other action or event that has taken place or will take place.
- Ideally this applies very well to verbs indicating some specific action.
- Related actions are depicted in entailment, where actions like walking involve or entail stepping , and eating entails chewing and swallowing .

```
# entailments
In [80]: for action in ['walk', 'eat', 'digest']:
    ...:     action_syn = wn.synsets(action, pos='v')[0]
    ...:     print action_syn, '-- entails -->', action_syn.entailments()
Synset('walk.v.01') -- entails --> [Synset('step.v.01')]
Synset('eat.v.01') -- entails --> [Synset('chew.v.01'),
Synset('swallow.v.01')]
Synset('digest.v.01') -- entails --> [Synset('consume.v.02')]
```

Exploring WordNet-Analyzing Lexical Semantic Relations

Homonyms and Homographs



- Homonyms refer to words or terms having the same written form or pronunciation but different meanings.
- Homonyms are a superset of homographs, which are words with same spelling but may have different pronunciation and meaning.
- You can see that there are various different meanings associated with the word 'bank'.

```
In [81]: for synset in wn.synsets('bank'):
    ...:     print synset.name(),'-',synset.definition()
    ...:
    ...:
bank.n.01 - sloping land (especially the slope beside a body of water)
depository_financial_institution.n.01 - a financial institution that accepts
deposits and channels the money into lending activities
bank.n.03 - a long ridge or pile
bank.n.04 - an arrangement of similar objects in a row or in tiers
...
...
deposit.v.02 - put into a bank account
bank.v.07 - cover with ashes so to control the rate of burning
trust.v.01 - have confidence or faith in
```

Exploring WordNet-Analyzing Lexical Semantic Relations

Synonyms and Antonyms

- Synonyms are words having similar meaning and context, and antonyms are words having opposite or contrasting meaning, as you may know already.
- The preceding outputs show sample synonyms and antonyms for the term 'large' and the term 'rich' .
- Additionally, we explore several synsets associated with the term or concept 'rich' , which rightly give us distinct synonyms and their corresponding antonyms.'

```
In [82]: term = 'large'
    ...: synsets = wn.synsets(term)
    ...: adj_large = synsets[1]
    ...: adj_large = adj_large.lemmas()[0]
    ...: adj_large_synonym = adj_large.synset()
    ...: adj_large_antonym = adj_large.antonyms()[0].synset()
    ...: # print synonym and antonym
    ...: print 'Synonym:', adj_large_synonym.name()
    ...: print 'Definition:', adj_large_synonym.definition()
    ...: print 'Antonym:', adj_large_antonym.name()
    ...: print 'Definition:', adj_large_antonym.definition()
Synonym: large.a.01
Definition: above average in size or number or quantity or magnitude or
extent
Antonym: small.a.01
Definition: limited or below average in number or quantity or magnitude or
extent
```

```
In [83]: term = 'rich'
    ...: synsets = wn.synsets(term)[:3]
    ...: # print synonym and antonym for different synsets
    ...: for synset in synsets:
    ...:     rich = synset.lemmas()[0]
    ...:     rich_synonym = rich.synset()
    ...:     rich_antonym = rich.antonyms()[0].synset()
    ...:     print 'Synonym:', rich_synonym.name()
    ...:     print 'Definition:', rich_synonym.definition()
    ...:     print 'Antonym:', rich_antonym.name()
    ...:     print 'Definition:', rich_antonym.definition()
Synonym: rich_people.n.01
Definition: people who have possessions and wealth (considered as a group)
Antonym: poor_people.n.01
Definition: people without possessions or wealth (considered as a group)

Synonym: rich.a.01
Definition: possessing material wealth
Antonym: poor.a.02
Definition: having little money or few possessions

Synonym: rich.a.02
Definition: having an abundant supply of desirable qualities or substances
(especially natural resources)
Antonym: poor.a.04
Definition: lacking in specific resources, qualities or substances
```

Exploring WordNet-Analyzing Lexical Semantic Relations

Hyponyms and Hypernyms

- Synsets represent terms with unique semantics and concepts and are linked or related to each other based on some similarity and context.
- Several of these synsets represent abstract and generic concepts also besides concrete entities.
- Usually they are interlinked together in the form of a hierarchical structure representing is-a relationships.
- Hyponyms and hypernyms help us explore related concepts by navigating through this hierarchy.
- To be more specific, hyponyms refer to entities or concepts that are a subclass of a higher order concept or entity and have very specific sense or context compared to its superclass.

Exploring WordNet-Analyzing Lexical Semantic Relations

Hyponyms and Hypernyms

- The following snippet shows the hyponyms for the entity 'tree' :
- The preceding output tells us that there are a total of 180 hyponyms for 'tree' , and we see some of the sample hyponyms and their definitions.

```
term = 'tree'
synsets = wn.synsets(term)
tree = synsets[0]
# print the entity and its meaning
In [86]: print 'Name:', tree.name()
    ...: print 'Definition:', tree.definition()
Name: tree.n.01
Definition: a tall perennial woody plant having a main trunk and branches
forming a distinct elevated crown; includes both gymnosperms and angiosperms
# print total hyponyms and some sample hyponyms for 'tree'
In [87]: hyponyms = tree.hyponyms()
    ...: print 'Total Hyponyms:', len(hyponyms)
    ...: print 'Sample Hyponyms'
    ...: for hyponym in hyponyms[:10]:
    ...:     print hyponym.name(), '-', hyponym.definition()

Total Hyponyms: 180
Sample Hyponyms
aalii.n.01 - a small Hawaiian tree with hard dark wood
acacia.n.01 - any of various spiny trees or shrubs of the genus Acacia
african_walnut.n.01 - tropical African timber tree with wood that resembles
mahogany
albizzia.n.01 - any of numerous trees of the genus Albizia
alder.n.02 - north temperate shrubs or trees having toothed leaves and
conelike fruit; bark is used in tanning and dyeing and the wood is rot-
resistant
angelim.n.01 - any of several tropical American trees of the genus Andira
angiospermous_tree.n.01 - any tree having seeds and ovules contained in the
ovary
anise_tree.n.01 - any of several evergreen shrubs and small trees of the
genus Illicium
arbor.n.01 - tree (as opposed to shrub)
aroeira_blanca.n.01 - small resinous tree or shrub of Brazil
```

Exploring WordNet-Analyzing Lexical Semantic Relations

Hyponyms and Hypernyms

- Hyperyms are entities or concepts that act as the superclass to hyponyms and have a more generic sense or context.

```
In [88]: hypernyms = tree.hypernyms()
    ...: print hypernyms
[Synset('woody_plant.n.01')]
```

- You can even navigate up the entire entity/concept hierarchy depicting all the hyponyms or parent classes for 'tree' using the following code snippet:

```
# get total hierarchy pathways for 'tree'
In [91]: hypernym_paths = tree.hypernym_paths()

    ...: print 'Total Hypernym paths:', len(hypernym_paths)
Total Hypernym paths: 1

# print the entire hypernym hierarchy
In [92]: print 'Hypernym Hierarchy'
    ...: print ' -> '.join(synset.name() for synset in hypernym_paths[0])
Hypernym Hierarchy
entity.n.01 -> physical_entity.n.01 -> object.n.01 -> whole.n.02 -> living_
thing.n.01 -> organism.n.01 -> plant.n.02 -> vascular_plant.n.01 -> woody_
plant.n.01 -> tree.n.01
```

- 'entity' is the most generic concept in which 'tree' is present, and the complete hypernym hierarchy showing the corresponding hypernym or superclass at each level is shown

Exploring WordNet-Analyzing Lexical Semantic Relations

Holonyms and Meronyms

- Holonyms are entities that contain a specific entity of our interest.
- Basically holonym refers to the relationship between a term or entity that denotes the whole and a term denoting a specific part of the whole.
- The following snippet shows the holonyms for 'tree' :
- From the output, we can see that 'forest' is a holonym for 'tree' , which is semantically correct because, of course, a forest is a collection of trees.

```
In [94]: member_holonyms = tree.member_holonyms()
    ...: print 'Total Member Holonyms:', len(member_holonyms)
    ...: print 'Member Holonyms for [tree]:-'
    ...: for holonym in member_holonyms:
    ...:     print holonym.name(), '-', holonym.definition()
Total Member Holonyms: 1
Member Holonyms for [tree]:-
forest.n.01 - the trees and other plants in a large densely wooded area
```

Exploring WordNet-Analyzing Lexical Semantic Relations

Holonyms and Meronyms

- Meronyms are semantic relationships that relate a term or entity as a part or constituent of another term or entity.
- The following snippet depicts different types of meronyms for 'tree' :

```
# part based meronyms for tree
In [95]: part_meronyms = tree.part_meronyms()
    ...: print 'Total Part Meronyms:', len(part_meronyms)
    ...: print 'Part Meronyms for [tree]:-'
    ...: for meronym in part_meronyms:
    ...:     print meronym.name(), '-', meronym.definition()
Total Part Meronyms: 5
Part Meronyms for [tree]:-
burl.n.02 - a large rounded outgrowth on the trunk or branch of a tree
crown.n.07 - the upper branches and leaves of a tree or other plant
limb.n.02 - any of the main branches arising from the trunk or a bough of a tree
stump.n.01 - the base part of a tree that remains standing after the tree has been felled
trunk.n.01 - the main stem of a tree; usually covered with bark; the bole is usually the part that is commercially useful for lumber

# substance based meronyms for tree
In [96]: substance_meronyms = tree.substance_meronyms()
    ...: print 'Total Substance Meronyms:', len(substance_meronyms)
    ...: print 'Substance Meronyms for [tree]:-'
    ...: for meronym in substance_meronyms:
    ...:     print meronym.name(), '-', meronym.definition()
Total Substance Meronyms: 2
Substance Meronyms for [tree]:-
heartwood.n.01 - the older inactive central wood of a tree or woody plant; usually darker and denser than the surrounding sapwood
sapwood.n.01 - newly formed outer wood lying between the cambium and the heartwood of a tree or woody plant; usually light colored; active in water conduction
```

- The preceding output shows various meronyms that include various constituents of trees like stump and trunk and also various derived substances from trees like heartwood and sapwood

Exploring WordNet-Analyzing Lexical Semantic Relations

Semantic Relationships and Similarity

- Ways to connect similar entities based on their semantic relationships and also measure semantic similarity between them.
- Semantic similarity is different from the conventional similarity metrics.

```
tree = wn.synset('tree.n.01')
lion = wn.synset('lion.n.01')
tiger = wn.synset('tiger.n.02')
cat = wn.synset('cat.n.01')
dog = wn.synset('dog.n.01')
# create entities and extract names and definitions
entities = [tree, lion, tiger, cat, dog]
entity_names = [entity.name().split('.')[0] for entity in entities]
entity_definitions = [entity.definition() for entity in entities]

# print entities and their definitions
In [99]: for entity, definition in zip(entity_names, entity_definitions):
    ...:     print entity, '-', definition
```

```
tree - a tall perennial woody plant having a main trunk and branches forming
a distinct elevated crown; includes both gymnosperms and angiosperms
lion - large gregarious predatory feline of Africa and India having a tawny
coat with a shaggy mane in the male
tiger - large feline of forests in most of Asia having a tawny coat with
black stripes; endangered
cat - feline mammal usually having thick soft fur and no ability to roar:
domestic cats; wildcats
dog - a member of the genus Canis (probably descended from the common wolf)
that has been domesticated by man since prehistoric times; occurs in many
breeds
```

- We will try to correlate the entities based on common hypernyms.
- For each pair of entities, we will try to find the lowest common hypernym in the relationship hierarchy tree.
- Correlated entities are expected to have very specific hypernyms, and unrelated entities should have very abstract or generic hypernyms.

Exploring WordNet-Analyzing Lexical Semantic Relations

Semantic Relationships and Similarity

```
common_hypernyms = []
for entity in entities:
    # get pairwise lowest common hypernyms
    common_hypernyms.append([entity.lowest_common_hypernyms(compared_entity)[0]
                                    .name().split('.')[0]
                             for compared_entity in entities])
# build pairwise lower common hypernym matrix
common_hypernym_frame = pd.DataFrame(common_hypernyms,
                                     index=entity_names,
                                     columns=entity_names)

# print the matrix
In [101]: print common_hypernym_frame
     ...:
          tree      lion      tiger       cat         dog
tree      tree  organism  organism  organism    organism
lion  organism      lion    big_cat    feline   carnivore
tiger organism    big_cat      tiger    feline   carnivore
cat   organism    feline     feline       cat   carnivore
dog   organism  carnivore  carnivore  carnivore         dog
```

- Ignoring the main diagonal of the matrix, for each pair of entities, we can see their lowest common hypernym which depicts the nature of relationship between them.
- Trees are unrelated to the other animals except that they are all living organisms. Hence we get the 'organism' relationship amongst them.
- Cats are related to lions and tigers with respect to being feline creatures, and we can see the same in the preceding output.
- Tigers and lions are connected to each other with the 'big cat' relationship.
- Finally, we can see dogs having the relationship of 'carnivore' with the other animals since they all typically eat meat.

Exploring WordNet-Analyzing Lexical Semantic Relations

Semantic Relationships and Similarity

- We will use 'path similarity' , which returns a value between [0, 1] based on the shortest path connecting two terms based on their hypernym/hyponym based taxonomy.
- The following snippet shows us how to generate this similarity matrix:

```
similarities = []
for entity in entities:
    # get pairwise similarities
    similarities.append([round(entity.path_similarity(compared_entity), 2)
                         for compared_entity in entities])
# build pairwise similarity matrix
similarity_frame = pd.DataFrame(similarities,
                                index=entity_names,
                                columns=entity_names)
# print the matrix
print similarity_frame
```

```
       tree  lion  tiger   cat   dog
tree   1.00  0.07   0.07  0.08  0.13
lion   0.07  1.00   0.33  0.25  0.17
tiger  0.07  0.33   1.00  0.25  0.17
cat    0.08  0.25   0.25  1.00  0.20
dog    0.13  0.17   0.17  0.20  1.00
```

- From the preceding output, as expected, lion and tiger are the most similar with a value of 0.33, followed by their semantic similarity with cat having a value of 0.25.
- And tree has the lowest semantic similarity values when compared with other animals.

Exploring WordNet- Word Sense Disambiguation

- Homographs and homonyms, which are basically words that look or sound similar but have very different meanings.
- This meaning is contextual based on how it has been used and also depends on the word semantics, also called word sense .
- Identifying the correct sense or semantics of a word based on its usage is called word sense disambiguation with the assumption that the word has multiple meanings based on its context.
- This is a very popular problem in NLP and is used in various applications, such as improving the relevance of search engine results, coherence, and so on.
- There are various ways to solve this problem, including lexical and dictionary-based methods and supervised and unsupervised ML methods.

Exploring WordNet- Word Sense Disambiguation

- The basic principle behind Word Sense Disambiguation is to leverage dictionary or vocabulary definitions for a word we want to disambiguate in a body of text and compare the words in these definitions with a section of text surrounding our word of interest.
- We will be using the WordNet definitions for words instead of a dictionary.
- The main objective for us would be to return the synset with the maximum number of overlapping words or terms between the context sentence and the different definitions from each synset for the word we target for disambiguation.

Exploring WordNet- Word Sense Disambiguation

```
from nltk.wsd import lesk
from nltk import word_tokenize

# sample text and word to disambiguate
samples = [('The fruits on that plant have ripened', 'n'),
           ('He finally reaped the fruit of his hard work as he won the
           race', 'n')]
word = 'fruit'
# perform word sense disambiguation
In [106]: for sentence, pos_tag in samples:
     ...:     word_syn = lesk(word_tokenize(sentence.lower()), word, pos_tag)
     ...:     print 'Sentence:', sentence
     ...:     print 'Word synset:', word_syn
     ...:     print 'Corresponding definition:', word_syn.definition()
     ...:     print
Sentence: The fruits on that plant have ripened
Word synset: Synset('fruit.n.01')
Corresponding definition: the ripened reproductive body of a seed plant

Sentence: He finally reaped the fruit of his hard work as he won the race
Word synset: Synset('fruit.n.03')
Corresponding definition: the consequence of some effort or action

# sample text and word to disambiguate
samples = [('Lead is a very soft, malleable metal', 'n'),
           ('John is the actor who plays the lead in that movie', 'n'),
           ('This road leads to nowhere', 'v')]
word = 'lead'
# perform word sense disambiguation
In [108]: for sentence, pos_tag in samples:
     ...:     word_syn = lesk(word_tokenize(sentence.lower()), word,
              pos_tag)
     ...:     print 'Sentence:', sentence
     ...:     print 'Word synset:', word_syn
     ...:     print 'Corresponding definition:', word_syn.definition()
     ...:     print
Sentence: Lead is a very soft, malleable metal
Word synset: Synset('lead.n.02')
Corresponding definition: a soft heavy toxic malleable metallic element;
bluish white when freshly cut but tarnishes readily to dull grey

Sentence: John is the actor who plays the lead in that movie
Word synset: Synset('star.n.04')
Corresponding definition: an actor who plays a principal role

Sentence: This road leads to nowhere
Word synset: Synset('run.v.23')
Corresponding definition: cause something to pass or lead somewhere
```

Disambiguate two words, 'fruit' and 'lead' in various text documents in the preceding examples.

Exploring WordNet- Named Entity Recognition

- In any text document, there are particular terms that represent entities that are more informative and have a unique context compared to the rest of the text.
- These entities are known as named entities , which more specifically refers to terms that represent real-world objects like people, places, organizations, and so on, which are usually denoted by proper names.
- Named entity recognition , also known as entity chunking/extraction , is a popular technique used in information extraction to identify and segment named entities and classify or categorize them under various predefined classes.

Exploring WordNet- Named Entity Recognition

- There is some overlap between GPE and LOCATION . The GPE entities are usually more generic and represent geo-political entities like cities, states, countries, and continents.
- LOCATION can also refer to these entities (it varies across different NER systems) along with very specific locations like a mountain, river, or hill-station.
- FACILITY on the other hand refers to popular monuments or artifacts that are usually man-made.
- The remaining categories are pretty self-explanatory from their names and the examples

Named Entity Type	Examples
PERSON	President Obama, Franz Beckenbauer
ORGANIZATION	WHO, ISRO, FC Bayern
LOCATION	Germany, India, USA, Mt. Everest
DATE	December, 2016-12-25
TIME	12:30:00 AM, one thirty pm
MONEY	Twenty dollars, Rs. 50, 100 GBP
PERCENT	20%, forty five percent
FACILITY	Stonehenge, Taj Mahal, Washington Monument
GPE	Asia, Europe, Germany, North America

Exploring WordNet- Named Entity Recognition

- **The Bundesliga is perhaps the most popular top-level professional association football league in Germany, and FC Bayern Munchen is one of the most popular clubsin this league with a global presence.**

```
# sample document
text = """
Bayern Munich, or FC Bayern, is a German sports club based in Munich,
Bavaria, Germany. It is best known for its professional football team,
which plays in the Bundesliga, the top tier of the German football
league system, and is the most successful club in German football
history, having won a record 26 national titles and 18 national cups.
FC Bayern was founded in 1900 by eleven football players led by Franz John.
Although Bayern won its first national championship in 1932, the club
was not selected for the Bundesliga at its inception in 1963. The club
had its period of greatest success in the middle of the 1970s when,
under the captaincy of Franz Beckenbauer, it won the European Cup three
times in a row (1974-76). Overall, Bayern has reached ten UEFA Champions
League finals, most recently winning their fifth title in 2013 as part
of a continental treble.
"""

import nltk
from normalization import parse_document
import pandas as pd

# tokenize sentences
sentences = parse_document(text)
tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in
sentences]

# tag sentences and use nltk's Named Entity Chunker
tagged_sentences = [nltk.pos_tag(sentence) for sentence in tokenized_
sentences]
ne_chunked_sents = [nltk.ne_chunk(tagged) for tagged in tagged_sentences]

# extract all named entities
named_entities = []
for ne_tagged_sentence in ne_chunked_sents:
    for tagged_tree in ne_tagged_sentence:
        # extract only chunks having NE labels
        if hasattr(tagged_tree, 'label'):
                entity_name = ' '.join(c[0] for c in tagged_tree.leaves()) #
                get NE name
                entity_type = tagged_tree.label() # get NE category
                named_entities.append((entity_name, entity_type))
# get unique named entities
named_entities = list(set(named_entities))
```

```
# store named entities in a data frame
entity_frame = pd.DataFrame(named_entities,
                            columns=['Entity Name', 'Entity Type'])

# display results
In [116]: print entity_frame
          Entity Name   Entity Type
0              Bayern        PERSON
1          Franz John        PERSON
2   Franz Beckenbauer        PERSON
3              Munich  ORGANIZATION
4            European  ORGANIZATION
5           Bundesliga  ORGANIZATION
6              German           GPE
7             Bavaria           GPE
8             Germany           GPE
9           FC Bayern  ORGANIZATION
10               UEFA  ORGANIZATION
11             Munich           GPE
12             Bayern           GPE
13            Overall           GPE
```

The Named Entity Chunker identifies named entities from the preceding text document, and we extract these named entities from the tagged annotated sentences and display them in the data frame as shown.

You can clearly see how it has correctly identified PERSON , ORGANIZATION , and GPE related named entities, although a few of them are incorrectly identified.

Exploring WordNet- Named Entity Recognition

- **Stanford NER tagger on the same text and compare the results.**

```
# tag sentences
ne_annotated_sentences = [sn.tag(sent) for sent in tokenized_sentences]

# extract named entities
named_entities = []
for sentence in ne_annotated_sentences:
    temp_entity_name = ''
    temp_named_entity = None
    for term, tag in sentence:
        # get terms with NE tags
        if tag != 'O':
            temp_entity_name = ' '.join([temp_entity_name, term]).strip() #
            get NE name
            temp_named_entity = (temp_entity_name, tag) # get NE and its
            category
        else:
            if temp_named_entity:
                named_entities.append(temp_named_entity)
                temp_entity_name = ''
                temp_named_entity = None

# get unique named entities
named_entities = list(set(named_entities))
# store named entities in a data frame
entity_frame = pd.DataFrame(named_entities,
                            columns=['Entity Name', 'Entity Type'])

# display results
In [118]: print entity_frame
          Entity Name   Entity Type
0         Franz John        PERSON
1  Franz Beckenbauer        PERSON
2            Germany      LOCATION
3             Bayern  ORGANIZATION
4             Bavaria      LOCATION
5             Munich      LOCATION
6          FC Bayern  ORGANIZATION
7               UEFA  ORGANIZATION
8      Bayern Munich  ORGANIZATION
```

```
from nltk.tag import StanfordNERTagger
import os

# set java path in environment variables
java_path = r'C:\Program Files\Java\jdk1.8.0_102\bin\java.exe'
os.environ['JAVAHOME'] = java_path

# load stanford NER
sn = StanfordNERTagger('E:/stanford/stanford-ner-2014-08-27/classifiers/
english.all.3class.distsim.crf.ser.gz',
                       path_to_jar='E:/stanford/stanford-ner-2014-08-27/
                       stanford-ner.jar')
```

The preceding output depicts various named entities obtained from our document.

You can compare this with the results obtained from nltk 's NER chunker.

The results here are definitely better—there are no misclassifications and each category is also assigned correctly.

Some really interesting points: It has correctly identified Munich as a LOCATION and Bayern Munich as an ORGANIZATION .

Does this mean the second NER tagger is better? Not really.

It depends on the type of corpus you are analyzing, and you can even build your own NER tagger using supervised learning by training on pre-tagged corpora

Exploring WordNet- Analyzing Semantic Representations

- Frameworks like propositional logic and first-order logic help us in representation of semantics.

PropositionalLogic

- A proposition is usually declarative, having a binary value of being either true or false.
- There also exist various logical operators like conjunction, disjunction, implication, and equivalence, and we also study the effects of applying these operators on multiple propositions to understand their behavior and outcome.
- Let us consider two propositions P and Q such that they can be represented as follows:
  - P : He is hungry
  - Q : He will eat a sandwich

Exploring WordNet- Analyzing Semantic Representations

```
import nltk
import pandas as pd
import os

# assign symbols and propositions
symbol_P = 'P'
symbol_Q = 'Q'
proposition_P = 'He is hungry'
propositon_Q = 'He will eat a sandwich'
# assign various truth values to the propositions
p_statuses = [False, False, True, True]
q_statuses = [False, True, False, True]
# assign the various expressions combining the logical operators
conjunction = '(P & Q)'
disjunction = '(P | Q)'
implication = '(P -> Q)'
```

The preceding output depicts the various truth values of the two propositions, and when we combine them with various logical operators,

For example, P & Q indicates He is hungry and he will eat a sandwich is True only when both of the individual propositions is True .

We use nltk 's Valuation class to create a dictionary of the propositions and their various outcome states.

We use the Model class to evaluate each expression, where the evaluate() function internally calls the recursive function satisfy() , which helps in evaluating the outcome of each expression with the propositions based on the assigned truth values.

```
equivalence = '(P <-> Q)'
expressions = [conjunction, disjunction, implication, equivalence]

# evaluate each expression using propositional logic
results = []
for status_p, status_q in zip(p_statuses, q_statuses):
    dom = set([])
    val = nltk.Valuation([(symbol_P, status_p),
                          (symbol_Q, status_q)])
    assignments = nltk.Assignment(dom)
    model = nltk.Model(dom, val)
    row = [status_p, status_q]
    for expression in expressions:
        # evaluate each expression based on proposition truth values
        result = model.evaluate(expression, assignments)
        row.append(result)
    results.append(row)
# build the result table
columns = [symbol_P, symbol_Q, conjunction,
           disjunction, implication, equivalence]
result_frame = pd.DataFrame(results, columns=columns)

# display results
In [125]: print 'P:', proposition_P
     ...: print 'Q:', propositon_Q
     ...: print
     ...: print 'Expression Outcomes:-'
     ...: print result_frame
P: He is hungry
Q: He will eat a sandwich

Expression Outcomes:-
       P      Q (P & Q) (P | Q) (P -> Q) (P <-> Q)
0  False  False   False   False     True      True
1  False   True   False    True     True     False
2   True  False   False    True    False     False
3   True   True    True    True     True      True
```

Exploring WordNet- Analyzing Semantic Representations

First Order Logic

- PL has several limitations, like the inability to represent facts or complex relationships and inferences.
- PL also has limited expressive power because for each new proposition we would need a unique symbolic representation, and it becomes very difficult to generalize facts.
- This is where first order logic (FOL) works really well with features like functions, quantifiers, relations, connectives, and symbols.
- It definitely provides a richer and more powerful representation for semantic information.
- How to represent FOL representations in Python and how to perform FOL inference using proofs based on some goal and predefined rules and events.
- There are several theorem provers you can use for evaluating expressions and proving theorems.
- The nltk package has three main different types of provers: Prover9 , TableauProver , and ResolutionProver .

Exploring WordNet- Analyzing Semantic Representations

First Order Logic

- let us evaluate a few FOL expressions.
- Consider a simple expression that *If an entity jumps over another entity, the reverse cannot happen* .
- Assuming the entities to be x and y , we can represent this is FOL as $\forall x \forall y$ (jumps_over(x, y) $\rightarrow \neg$ jumps_over(y, x)) which signifies that for all x and y , if x jumps over y , it implies that y cannot jump over x .
- Consider now that we have two entities fox and dog such that the fox jumps over the dog is an event which has taken place and can be represented by jumps_over(fox, dog) .
- Our end goal or objective is to evaluate the outcome of jumps_over(dog, fox) considering the preceding expression and the event that has occurred.

Exploring WordNet- Analyzing Semantic Representations

First Order Logic

```
import nltk
import os
# for reading FOL expressions
read_expr = nltk.sem.Expression.fromstring
# initialize theorem provers (you can choose any)
os.environ['PROVER9'] = r'E:/prover9/bin'
prover = nltk.Prover9()
# I use the following one for our examples
prover = nltk.ResolutionProver()
```

```
# set the rule expression
rule = read_expr('all x. all y. (jumps_over(x, y) -> -jumps_over(y, x))')
# set the event occured
```

```
event = read_expr('jumps_over(fox, dog)')
# set the outcome we want to evaluate -- the goal
test_outcome = read_expr('jumps_over(dog, fox)')

# get the result
In [132]: prover.prove(goal=test_outcome,
     ...:              assumptions=[event, rule],
     ...:              verbose=True)
[1] {-jumps_over(dog,fox)}                    A
[2] {jumps_over(fox,dog)}                     A
[3] {-jumps_over(z4,z3), -jumps_over(z3,z4)}  A
[4] {-jumps_over(dog,fox)}                    (2, 3)

Out[132]: False
```

The preceding output depicts the final result for our goal test_outcome is False , that is, the dog cannot jump over the fox if the fox has already jumped over the dog based on our rule expression and the events assigned to the assumptions parameter in the prover already given.

The sequence of steps that lead to the result is also shown in the output.

Exploring WordNet- Analyzing Semantic Representations

First Order Logic

- Let us now consider another FOL expression rule for all x studies(x, exam) → pass(x, exam) , which tells us that for all instances of x , if x studies for the exam, he/she will pass the exam. Let us represent this rule and consider two students, John and Pierre , such that John does not study for the exam and Pierre does.

```
# set the rule expression
rule = read_expr('all x. (studies(x, exam) -> pass(x, exam))')
# set the events and outcomes we want to determine
event1 = read_expr('-studies(John, exam)')
test_outcome1 = read_expr('pass(John, exam)')
event2 = read_expr('studies(Pierre, exam)')
test_outcome2 = read_expr('pass(Pierre, exam)')

# get results
In [134]: prover.prove(goal=test_outcome1,
     ...:              assumptions=[event1, rule],
     ...:              verbose=True)
[1] {-pass(John,exam)}                  A
[2] {-studies(John,exam)}               A
[3] {-studies(z6,exam), pass(z6,exam)}  A
[4] {-studies(John,exam)}               (1, 3)

Out[134]: False

In [135]: prover.prove(goal=test_outcome2,
     ...:              assumptions=[event2, rule],
     ...:              verbose=True)
[1] {-pass(Pierre,exam)}                A
[2] {studies(Pierre,exam)}              A
[3] {-studies(z8,exam), pass(z8,exam)}  A
[4] {-studies(Pierre,exam)}             (1, 3)
[5] {pass(Pierre,exam)}                 (2, 3)
[6] {}                                  (1, 5)

Out[135]: True
```

Exploring WordNet- Analyzing Semantic Representations

First Order Logic

- Let us consider a more complex example with several entities. They perform several actions as follows:
- There are two dogs rover (r) and alex (a)
- There is one cat garfield (g)
- There is one fox felix (f)
- Two animals, alex (a) and felix (f) run, denoted by function runs()
- Two animals rover (r) and garfield (g) sleep, denoted by function sleeps()
- Two animals, felix (f) and alex (a) can jump over the other two, denoted by function jumps_over()

Exploring WordNet- Analyzing Semantic Representations

First Order Logic

```
# define symbols (entities\functions) and their values
rules = """
    rover => r
    felix => f
    garfield => g
    alex => a
    dog => {r, a}
    cat => {g}
    fox => {f}
    runs => {a, f}
    sleeps => {r, g}
    jumps_over => {(f, g), (a, g), (f, r), (a, r)}
    """
val = nltk.Valuation.fromstring(rules)
# view the valuation object of symbols and their assigned values
(dictionary)
```

The preceding snippet depicts the evaluation of various expressions based on the valuation of different symbols based on the rules and domain.

```
In [143]: print val
{'rover': 'r', 'runs': set([('f',), ('a',)]), 'alex': 'a', 'sleeps':
set([('r',), ('g',)]), 'felix': 'f', 'fox': set([('f',)]), 'dog':
set([('a',), ('r',)]), 'jumps_over': set([('a', 'g'), ('f', 'g'), ('a',
'r'), ('f', 'r')]), 'cat': set([('g',)]), 'garfield': 'g'}

# define domain and build FOL based model
dom = {'r', 'f', 'g', 'a'}
m = nltk.Model(dom, val)

# evaluate various expressions
In [148]: print m.evaluate('jumps_over(felix, rover) & dog(rover) &
runs(rover)', None)
False

In [149]: print m.evaluate('jumps_over(felix, rover) & dog(rover) &
-runs(rover)', None)
True

In [150]: print m.evaluate('jumps_over(alex, garfield) & dog(alex) &
cat(garfield) & sleeps(garfield)', None)
True

# assign rover to x and felix to y in the domain
g = nltk.Assignment(dom, [('x', 'r'), ('y', 'f')])

# evaluate more expressions based on above assigned symbols
In [152]: print m.evaluate('runs(y) & jumps_over(y, x) & sleeps(x)', g)
True

In [153]: print m.evaluate('exists y. (fox(y) & runs(y))', g)
True
```

Sentiment Analysis

- Textual data , even though unstructured, mainly has two broad types of data points: factual based (objective) and opinion based (subjective).
- In general, social media, surveys, and feedback data all are heavily opinionated and express the beliefs, judgement, emotion, and feelings of human beings.
- Sentiment analysis, also popularly known as opinion analysis/mining , is defined as the process of using techniques like NLP, lexical resources, linguistics, and machine learning (ML) to extract subjective and opinion related information like emotions, attitude, mood, modality, and so on and try to use these to compute the polarity expressed by a text document.
- polarity - whether the document expresses a positive, negative, or a neutral sentiment.
- More advanced analysis involves trying to find out more complex emotions like sadness, happiness, anger, and sarcasm.

Sentiment Analysis

- Sentiment analysis for text data can be computed on several levels, including on an individual sentence level, paragraph level, or the entire document as a whole.
- Often sentiment is computed on the document as a whole or some aggregations are done after computing the sentiment for individual sentences.
- Polarity analysis usually involves trying to assign some scores contributing to the positive and negative emotions expressed in the document and then finally assigning a label to the document based on the aggregate score.
- We will depict two major techniques for sentiment analysis here:
- • Supervised machine learning
- • Unsupervised lexicon-based

Sentiment Analysis of IMDb Movie Reviews

- Dataset of movie reviews obtained from the Internet Movie Database (IMDb) for sentiment analysis.
- This dataset, containing over 50,000 movie reviews, can be obtained from http://ai.stanford.edu/~amaas/data/sentiment/
- 50,000 movie reviews from this dataset, which contain the review and a corresponding sentiment polarity label which is either positive or negative.
- A positive review is basically a movie review which was rated with more than six stars in IMDb, and a negative review was rated with less than five stars in IMDb

Sentiment Analysis of IMDb Movie Reviews- Setting Up Dependencies

- Some utility functions for text normalization, feature extracting, and model evaluation.

Getting and Formatting data

- We will use the IMDb movie review dataset officially available in raw text files for each set (training and testing) from http://ai.stanford.edu/~amaas/data/sentiment/.
- The data frame consists of two columns, review and sentiment , for each data point, which indicates the review for a movie and its corresponding sentiment (positive or negative).

Sentiment Analysis of IMDb Movie Reviews- Setting Up Dependencies

Text Normalization

- This mainly includes adding an HTML stripper to remove unnecessary HTML characters from text documents.
- We also add a new function to normalize special accented characters and convert them into regular ASCII characters so as to standardize the text across all documents.
- The overall text normalization function is depicted in the following snippet and it re-uses the expand contractions, lemmatization, HTML unescaping, special characters removal, and stopwords removal functions.

```
from HTMLParser import HTMLParser

class MLStripper(HTMLParser):
    def __init__(self):
        self.reset()
        self.fed = []
    def handle_data(self, d):
        self.fed.append(d)
    def get_data(self):
        return ' '.join(self.fed)

def strip_html(text):
    html_stripper = MLStripper()
    html_stripper.feed(text)
    return html_stripper.get_data()
```

```
def normalize_accented_characters(text):
    text = unicodedata.normalize('NFKD',
                                 text.decode('utf-8')
                                 ).encode('ascii', 'ignore')
    return text

def normalize_corpus(corpus, lemmatize=True,
                     only_text_chars=False,
                     tokenize=False):

    normalized_corpus = []
    for index, text in enumerate(corpus):
        text = normalize_accented_characters(text)
        text = html_parser.unescape(text)
        text = strip_html(text)
        text = expand_contractions(text, CONTRACTION_MAP)
        if lemmatize:
            text = lemmatize_text(text)
        else:
```

```
            text = text.lower()
        text = remove_special_characters(text)
        text = remove_stopwords(text)
        if only_text_chars:
            text = keep_text_characters(text)

        if tokenize:
            text = tokenize_text(text)
            normalized_corpus.append(text)
        else:
            normalized_corpus.append(text)

    return normalized_corpus
```

Sentiment Analysis of IMDb Movie Reviews- Setting Up Dependencies

Feature Extraction

- **Bag of Words-based frequencies, occurrences, and TF-IDF based features.**

```
from HTMLParser import HTMLParser

class MLStripper(HTMLParser):
    def __init__(self):
        self.reset()
        self.fed = []
    def handle_data(self, d):
        self.fed.append(d)
    def get_data(self):
        return ' '.join(self.fed)

def strip_html(text):
    html_stripper = MLStripper()
    html_stripper.feed(text)
    return html_stripper.get_data()
```

```
def normalize_accented_characters(text):
    text = unicodedata.normalize('NFKD',
                                 text.decode('utf-8')
                                 ).encode('ascii', 'ignore')
    return text
```

```
def normalize_corpus(corpus, lemmatize=True,
                     only_text_chars=False,
                     tokenize=False):

    normalized_corpus = []
    for index, text in enumerate(corpus):
        text = normalize_accented_characters(text)
        text = html_parser.unescape(text)
        text = strip_html(text)
        text = expand_contractions(text, CONTRACTION_MAP)
        if lemmatize:
            text = lemmatize_text(text)
        else:
            text = text.lower()
        text = remove_special_characters(text)
        text = remove_stopwords(text)
        if only_text_chars:
            text = keep_text_characters(text)

        if tokenize:
            text = tokenize_text(text)
            normalized_corpus.append(text)
        else:
            normalized_corpus.append(text)

    return normalized_corpus
```

Sentiment Analysis of IMDb Movie Reviews- Setting Up Dependencies

Model Performance Evaluation

- evaluating our models based on precision, recall, accuracy, and F1-score, similar to our evaluation methods in Chapter 4 for text classification.
- Additionally, confusion matrix and detailed classification reports for each class, that is, the positive and negative classes to evaluate model performance

```
from sklearn import metrics
import numpy as np
import pandas as pd

def display_evaluation_metrics(true_labels, predicted_labels, positive_
class=1):
    print 'Accuracy:', np.round(
                        metrics.accuracy_score(true_labels,
                                               predicted_labels),
                        2)
    print 'Precision:', np.round(
                        metrics.precision_score(true_labels,
                                               predicted_labels,
                                               pos_label=positive_class,
                                               average='binary'),
                        2)
    print 'Recall:', np.round(
                        metrics.recall_score(true_labels,
                                               predicted_labels,
                                               pos_label=positive_class,
                                               average='binary'),
                        2)
    print 'F1 Score:', np.round(
                        metrics.f1_score(true_labels,
                                               predicted_labels,
                                               pos_label=positive_class,
                                               average='binary'),
                        2)
```

```
def display_confusion_matrix(true_labels, predicted_labels, classes=[1,0]):
    cm = metrics.confusion_matrix(y_true=true_labels,
                                  y_pred=predicted_labels,
                                  labels=classes)
    cm_frame = pd.DataFrame(data=cm,
                            columns=pd.MultiIndex(levels=[['Predicted:'],
                            classes],
                                                  labels=[[0,0],[0,1]]),
                            index=pd.MultiIndex(levels=[['Actual:'],
                            classes],
                                                labels=[[0,0],[0,1]]))
    print cm_frame
```

Finally, we will define a function for getting a detailed classification report per sentiment category (positive and negative) by displaying the precision, recall, F1-score, and support (number of reviews) for each of the classes:

```
def display_classification_report(true_labels, predicted_labels,
classes=[1,0]):
    report = metrics.classification_report(y_true=true_labels,
                                           y_pred=predicted_labels,
                                           labels=classes)
    print report
```

Sentiment Analysis of IMDb Movie Reviews- Setting Up Dependencies

Preparing Datasets

- We will be loading our movie reviews data and preparing two datasets, namely training and testing.
- We will train our supervised model on the training data and evaluate model performance on the testing data.
- For unsupervised models, we will directly evaluate them on the testing data so as to compare their performance with the supervised model.
- Besides that, we will also pick some sample positive and negative reviews to see how the different models perfor

```
import pandas as pd
import numpy as np
# load movie reviews data
dataset = pd.read_csv(r'E:/aclImdb/movie_reviews.csv')
# print sample data
In [235]: print dataset.head()
                                              review sentiment
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive

# prepare training and testing datasets
train_data = dataset[:35000]
test_data = dataset[35000:]

train_reviews = np.array(train_data['review'])
train_sentiments = np.array(train_data['sentiment'])
test_reviews = np.array(test_data['review'])
test_sentiments = np.array(test_data['sentiment'])

# prepare sample dataset for experiments
sample_docs = [100, 5817, 7626, 7356, 1008, 7155, 3533, 13010]
sample_data = [(test_reviews[index],
                test_sentiments[index])
                  for index in sample_docs]
```

Supervised Machine Learning Technique

- This model will learn from past reviews and their corresponding sentiment from the training dataset so that it can predict the sentiment for new reviews from the test dataset.
- The basic principle here is to use the same concepts we used for text classification such that the classes to predict here are positive and negative sentiment corresponding to the movie reviews.
- The following points summarize these steps (Text classification Blue print):

1. Model training
- a. Normalize training data
- b. Extract features and build feature set and feature vectorizer
- c. Use supervised learning algorithm (SVM) to build a predictive model

2. Model testing
- a. Normalize testing data
- b. Extract features using training feature vectorizer
- c. Predict the sentiment for testing reviews using training model
- d. Evaluate model performance

Supervised Machine Learning Technique

- This model will learn from past reviews and their corresponding sentiment from the training dataset so that it can predict the sentiment for new reviews from the test dataset.
- The basic principle here is to use the same concepts we used for text classification such that the classes to predict here are positive and negative sentiment corresponding to the movie reviews.
- The following points summarize these steps (Text classification Blue print):

1. Model training
- a. Normalize training data
- b. Extract features and build feature set and feature vectorizer
- c. Use supervised learning algorithm (SVM) to build a predictive model

2. Model testing
- a. Normalize testing data
- b. Extract features using training feature vectorizer
- c. Predict the sentiment for testing reviews using training model
- d. Evaluate model performance

Supervised Machine Learning Technique

- **Normalization, Classification**

```
from normalization import normalize_corpus
from utils import build_feature_matrix

# normalization
norm_train_reviews = normalize_corpus(train_reviews, lemmatize=True, only_
text_chars=True)
# feature extraction
vectorizer, train_features = build_feature_matrix(documents=norm_train_
reviews,
                                                  feature_type='tfidf',
# normalize reviews
norm_test_reviews = normalize_corpus(test_reviews, lemmatize=True, only_
text_chars=True)
# extract features
test_features = vectorizer.transform(norm_test_reviews)
```

```
from sklearn.linear_model import SGDClassifier
# build the model
svm = SGDClassifier(loss='hinge', n_iter=200)
svm.fit(train_features, train_sentiments)
```

```
Actual Labeled Sentiment: positive
Predicted Sentiment: positive

Review:-
I don't care if some people voted this movie to be bad. If you want the
Truth this is a Very Good Movie! It has every thing a movie should have. You
really should Get this one.
Actual Labeled Sentiment: positive
Predicted Sentiment: negative

Review:-
Worst horror film ever but funniest film ever rolled in one you have got
to see this film it is so cheap it is unbeliaveble but you have to see it
really!!!! P.s watch the carrot
Actual Labeled Sentiment: positive
Predicted Sentiment: negative
```

```
# predict sentiment for sample docs from test data
In [253]: for doc_index in sample_docs:
     ...:     print 'Review:-'
     ...:     print test_reviews[doc_index]
     ...:     print 'Actual Labeled Sentiment:', test_sentiments[doc_index]
     ...:     doc_features = test_features[doc_index]
     ...:     predicted_sentiment = svm.predict(doc_features)[0]
     ...:     print 'Predicted Sentiment:', predicted_sentiment
     ...:     print
     ...:
     ...:
Review:-
Worst movie, (with the best reviews given it) I've ever seen. Over the top
dialog, acting, and direction. more slasher flick than thriller.With all the
great reviews this movie got I'm appalled that it turned out so silly. shame
on you martin scorsese
Actual Labeled Sentiment: negative
Predicted Sentiment: negative

Review:-
I hope this group of film-makers never re-unites.
Actual Labeled Sentiment: negative
Predicted Sentiment: negative

Review:-
no comment - stupid movie, acting average or worse... screenplay - no sense
at all... SKIP IT!
Actual Labeled Sentiment: negative
Predicted Sentiment: negative

Review:-
Add this little gem to your list of holiday regulars. It is<br /><br
/>sweet, funny, and endearing
Actual Labeled Sentiment: positive
Predicted Sentiment: positive

Review:-
a mesmerizing film that certainly keeps your attention... Ben Daniels is
fascinating (and courageous) to watch.
Actual Labeled Sentiment: positive
Predicted Sentiment: positive

Review:-
This movie is perfect for all the romantics in the world. John Ritter has
never been better and has the best line in the movie! "Sam" hits close to
home, is lovely to look at and so much fun to play along with. Ben Gazzara
was an excellent cast and easy to fall in love with. I'm sure I've met
Arthur in my travels somewhere. All around, an excellent choice to pick up
any evening.!:-)
```

Supervised Machine Learning Technique

- **Evaluation**

```
# predict the sentiment for test dataset movie reviews
predicted_sentiments = svm.predict(test_features)

# evaluate model prediction performance
from utils import display_evaluation_metrics, display_confusion_matrix,
display_classification_report

# show performance metrics
In [270]: display_evaluation_metrics(true_labels=test_sentiments,
     ...:                            predicted_labels=predicted_sentiments,
     ...:                            positive_class='positive')
Accuracy: 0.89
Precision: 0.88
Recall: 0.9
F1 Score: 0.89

# show confusion matrix
In [271]: display_confusion_matrix(true_labels=test_sentiments,
     ...:                           predicted_labels=predicted_sentiments,
     ...:                           classes=['positive', 'negative'])
                    Predicted:
                      positive negative
Actual: positive      6770     740
        negative      912      6578

# show detailed per-class classification report
In [272]: display_classification_report(true_labels=test_sentiments,
     ...:                                predicted_labels=predicted_
                                         sentiments,
     ...:                                classes=['positive', 'negative'])
              precision    recall  f1-score   support

    positive       0.88      0.90      0.89      7510
    negative       0.90      0.88      0.89      7490

avg / total        0.89      0.89      0.89     15000
```

Unsupervised Lexicon-based Techniques

- Sometimes, you may not have the convenience of a well-labeled training dataset.
- In those situations, you need to use unsupervised techniques for predicting the sentiment by using knowledgebases, ontologies, databases, and lexicons that have detailed information specially curated and prepared just for sentiment analysis.
- A lexicon is a dictionary, vocabulary, or a book of words. lexicons are special dictionaries or vocabularies that have been created for analyzing sentiment.
- Most of these lexicons have a list of positive and negative polar words with some score associated with them, and using various techniques like the position of words, surrounding words, context, parts of speech, phrases, and so on, scores are assigned to the text documents for which we want to compute the sentiment.
- After aggregating these scores, we get the final sentiment.
- More advanced analyses can also be done, including detecting the subjectivity, mood, and modality.

Unsupervised Lexicon-based Techniques

- AFINN lexicon
- Bing Liu's lexicon
- MPQA subjectivity lexicon
- SentiWordNet
- VADER lexicon
- Pattern lexicon

AFINN Lexicon

- AFINN-111, consists of a total of 2477 words and phrases with their own scores based on sentiment polarity.
- The polarity basically indicates how positive, negative, or neutral the term might be with some numerical score.
- The basic idea is to load the entire list of polar words and phrases in the lexicon along with their corresponding score (sample shown above) in memory and then find the same words/phrases and score them accordingly in a text document.
- Finally, these scores are aggregated, and the final sentiment and score can be obtained for a text document.

```
from afinn import Afinn
afn = Afinn(emoticons=True)

In [281]: print afn.score('I really hated the plot of this movie')
-3.0
In [282]: print afn.score('I really hated the plot of this movie :(')
-5.0
```

```
abandon       -2
abandoned     -2
abandons      -2
abducted      -2
abduction     -2
...
...
youthful       2
yucky         -2
yummy          3
zealot        -2
zealots       -2
zealous        2
```

Bing Liu's Lexicon

- Identifying Comparative Sentences in Text Documents.
- This lexicon consists of over 6800 words divided into two files named positive-words.txt , containing around 2000+ words/ phrases, and negative-words.txt , which contains around 4800+ words/phrases.
- The key idea is to leverage these words to contribute to the positive or negative polarity of any text document when they are identified in that document.
- This lexicon also includes many misspelled words, taking into account that words or terms are often misspelled on popular social media web sites.

MPQA Subjectivity Lexicon

- **MPQA stands for Multi-Perspective Question Answering, contains resources including opinion corpora, subjectivity lexicon, sense annotations, argument-based lexicon, and debate datasets.**
- **A lot of these can be leveraged for complex analysis of human emotions and sentiment.**

```
type=weaksubj len=1 word1=abandoned pos1=adj stemmed1=n
priorpolarity=negative
type=weaksubj len=1 word1=abandonment pos1=noun stemmed1=n
priorpolarity=negative
type=weaksubj len=1 word1=abandon pos1=verb stemmed1=y
priorpolarity=negative
type=strongsubj len=1 word1=abase pos1=verb stemmed1=y
priorpolarity=negative
...
...
type=strongsubj len=1 word1=zealously pos1=anypos stemmed1=n
priorpolarity=negative
type=strongsubj len=1 word1=zenith pos1=noun stemmed1=n
priorpolarity=positive
type=strongsubj len=1 word1=zest pos1=noun stemmed1=n priorpolarity=positive
```

MPQA Subjectivity Lexicon

- The various parameters mentioned above are explained briefly as follows:

- **type** : This has values that are either strongsubj indicating the presence of a strongly subjective context or weaksubj which indicates the presence of a weak/part subjective context.

- **len** : This points to the number of words in the term of the clue (all are single words of length 1 for now).

- **word1** : The actual term present as a token or a stem of the actual token.

- **pos1** : The part of speech for the term (clue) and it can be noun , verb , adj , adverb , or anypos .

- **stemmed1** : This indicates if the clue (term) is stemmed (y) or not stemmed (n). If it is stemmed, it can match all its other variants having the same pos1 tag.

- **priorpolarity** : This has values of negative, positive, both, or neutral, and indicates the polarity of the sentiment associated with this clue (term).

SentiWordNet

- The SentiWordNet lexicon is a lexical resource used for sentiment analysis and opinion mining.
- For each synset present in WordNet, the SentiWordNet lexicon assigns three sentiment scores to it, including a positive polarity score, a negative polarity score, and an objectivity score.:

```
import nltk
from nltk.corpus import sentiwordnet as swn
# get synset for 'good'
good = swn.senti_synsets('good', 'n')[0]
# print synset sentiment scores
In [287]: print 'Positive Polarity Score:', good.pos_score()
     ...: print 'Negative Polarity Score:', good.neg_score()
     ...: print 'Objective Score:', good.obj_score()
Positive Polarity Score: 0.5
Negative Polarity Score: 0.0
Objective Score: 0.5
```

Now that we know how to use the sentiwordnet interface, we define a function that can take in a body of text (movie review in our case) and analyze its sentiment by leveraging sentiwordnet:

```
from normalization import normalize_accented_characters, html_parser, strip_
html

def analyze_sentiment_sentiwordnet_lexicon(review,
                                           verbose=False):
    # pre-process text
    review = normalize_accented_characters(review)
    review = html_parser.unescape(review)
    review = strip_html(review)
    # tokenize and POS tag text tokens
    text_tokens = nltk.word_tokenize(review)
    tagged_text = nltk.pos_tag(text_tokens)
    pos_score = neg_score = token_count = obj_score = 0
    # get wordnet synsets based on POS tags
    # get sentiment scores if synsets are found
    for word, tag in tagged_text:
        ss_set = None
    if 'NN' in tag and swn.senti_synsets(word, 'n'):
        ss_set = swn.senti_synsets(word, 'n')[0]
    elif 'VB' in tag and swn.senti_synsets(word, 'v'):
        ss_set = swn.senti_synsets(word, 'v')[0]
    elif 'JJ' in tag and swn.senti_synsets(word, 'a'):
        ss_set = swn.senti_synsets(word, 'a')[0]
    elif 'RB' in tag and swn.senti_synsets(word, 'r'):
        ss_set = swn.senti_synsets(word, 'r')[0]
    # if senti-synset is found
    if ss_set:
        # add scores for all found synsets
        pos_score += ss_set.pos_score()
        neg_score += ss_set.neg_score()
        obj_score += ss_set.obj_score()
        token_count += 1

# aggregate final scores
final_score = pos_score - neg_score
norm_final_score = round(float(final_score) / token_count, 2)
final_sentiment = 'positive' if norm_final_score >= 0 else 'negative'
if verbose:
    norm_obj_score = round(float(obj_score) / token_count, 2)
    norm_pos_score = round(float(pos_score) / token_count, 2)
    norm_neg_score = round(float(neg_score) / token_count, 2)
    # to display results in a nice table
    sentiment_frame = pd.DataFrame([[final_sentiment, norm_obj_score,
                                     norm_pos_score, norm_neg_score,
                                     norm_final_score]],
                                     columns=pd.MultiIndex(levels
                                     =[['SENTIMENT STATS:'],
                                               ['Predicted Sentiment',
                                                'Objectivity',
                                                'Positive', 'Negative',
                                                'Overall']],
                                               labels=[[0,0,0,0,0],
                                               [0,1,2,3,4]]))
                                               print sentiment_frame

return final_sentiment
```

SentiWordNet

```
# detailed sentiment analysis for sample reviews
In [292]: for review, review_sentiment in sample_data:
     ...:     print 'Review:'
     ...:     print review
     ...:     print
     ...:     print 'Labeled Sentiment:', review_sentiment
     ...:     print
     ...:     final_sentiment = analyze_sentiment_sentiwordnet_
              lexicon(review,
     ...:
              verbose=True)
     ...:     print '-'*60
     ...:
     ...:
Review:
Worst movie, (with the best reviews given it) I've ever seen. Over the top
dialog, acting, and direction. more slasher flick than thriller.With all the
great reviews this movie got I'm appalled that it turned out so silly. shame
on you martin scorsese

Labeled Sentiment: negative

     SENTIMENT STATS:
  Predicted Sentiment Objectivity Positive Negative Overall
0            negative        0.83     0.08     0.09   -0.01
-----------------------------------------------------------
Review:
I hope this group of film-makers never re-unites.

Labeled Sentiment: negative

     SENTIMENT STATS:
  Predicted Sentiment Objectivity Positive Negative Overall
0            negative        0.71     0.04     0.25   -0.21
-----------------------------------------------------------
Review:
no comment - stupid movie, acting average or worse... screenplay - no sense
at all... SKIP IT!

Labeled Sentiment: negative

     SENTIMENT STATS:
  Predicted Sentiment Objectivity Positive Negative Overall
0            negative        0.81     0.04     0.15   -0.11
-----------------------------------------------------------
Review:
Add this little gem to your list of holiday regulars. It is<br /><br
/>sweet, funny, and endearing
```

```
# predict sentiment for test movie reviews dataset
sentiwordnet_predictions = [analyze_sentiment_sentiwordnet_lexicon(review)
                            for review in test_reviews]

from utils import display_evaluation_metrics, display_confusion_matrix,
display_classification_report

# get model performance statistics
In [295]: print 'Performance metrics:'
     ...: display_evaluation_metrics(true_labels=test_sentiments,
     ...:                            predicted_labels=sentiwordnet_
                                     predictions,
     ...:                            positive_class='positive')
     ...: print '\nConfusion Matrix:'
     ...: display_confusion_matrix(true_labels=test_sentiments,
     ...:                          predicted_labels=sentiwordnet_
                                   predictions,
     ...:                          classes=['positive', 'negative'])
     ...: print '\nClassification report:'
     ...: display_classification_report(true_labels=test_sentiments,
     ...:                               predicted_labels=sentiwordnet_
                                        predictions,
     ...:                               classes=['positive', 'negative'])
Performance metrics:
Accuracy: 0.59
Precision: 0.56
Recall: 0.92
F1 Score: 0.7

Confusion Matrix:
                Predicted:
                  positive negative
Actual: positive      6941      569
        negative      5510     1980

Classification report:

             precision    recall  f1-score   support

   positive       0.56      0.92      0.70      7510
   negative       0.78      0.26      0.39      7490

avg / total       0.67      0.59      0.55     15000
```

VADER Lexicon

- VADER stands for Valence Aware Dictionary and sEntiment Reasoner.
- It is a lexicon with a rule-based sentiment analysis framework that was specially built for analyzing sentiment from social media resources.
- The file vader_sentiment_lexicon.txt contains all the necessary sentiment scores associated with various terms, including words, emoticons, and even slang language based tokens (like lol , wtf , nah , and so on).
- There are over 9000 lexical features from which it was further curated to 7500 lexical features in this lexicon with proper validated valence scores.
- Each feature was rated on a scale from "[-4] Extremely Negative" to "[4] Extremely Positive" , with allowance for "[0] Neutral (or Neither, N/A)" .
- This curation was done by keeping all lexical features which had a non-zero mean rating and whose standard deviation was less than 2.5, which was determined by the aggregate of ten independent raters.

```
)-:<    -2.2   0.4     [-2, -2, -2, -2, -2, -2, -3, -3, -2, -2]
)-:{    -2.1   0.9434  [-1, -3, -2, -1, -2, -2, -3, -4, -1, -2]
):      -1.8   0.87178 [-1, -3, -1, -2, -1, -3, -1, -3, -1, -2]
...
...
resolved      0.7   0.78102  [1, 2, 0, 1, 1, 0, 2, 0, 0, 0]
resolvent     0.7   0.78102  [1, 0, 1, 2, 0, -1, 1, 1, 1, 1]
resolvents    0.4   0.66332  [2, 0, 0, 1, 0, 0, 1, 0, 0, 0]
...
...
}:-(    -2.1   0.7       [-2, -1, -2, -2, -2, -4, -2, -2, -2, -2]
}:-)     0.3   1.61555   [1, 1, -2, 1, -1, -3, 2, 2, 1, 1]
```

VADER Lexicon

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

def analyze_sentiment_vader_lexicon(review,
                                    threshold=0.1,
                                    verbose=False):
    # pre-process text
    review = normalize_accented_characters(review)
    review = html_parser.unescape(review)
    review = strip_html(review)
    # analyze the sentiment for review
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(review)
    # get aggregate scores and final sentiment
    agg_score = scores['compound']
    final_sentiment = 'positive' if agg_score >= threshold\
                                   else 'negative'
    if verbose:
        # display detailed sentiment statistics
        positive = str(round(scores['pos'], 2)*100)+'%'
        final = round(agg_score, 2)
        negative = str(round(scores['neg'], 2)*100)+'%'
        neutral = str(round(scores['neu'], 2)*100)+'%'
        sentiment_frame = pd.DataFrame([[final_sentiment, final, positive,
                                         negative, neutral]],
                columns=pd.MultiIndex(levels=[['SENTIMENT STATS:'],
                                              ['Predicted Sentiment',
                                               'Polarity Score',
                                               'Positive', 'Negative',
                                               'Neutral']],
                                      labels=[[0,0,0,0,0],[0,1,2,3,4]]))
        print sentiment_frame

    return final_sentiment
```

```
# get detailed sentiment statistics
In [301]: for review, review_sentiment in sample_data:
     ...:     print 'Review:'
     ...:     print review
     ...:     print
     ...:     print 'Labeled Sentiment:', review_sentiment
     ...:     print
     ...:     final_sentiment = analyze_sentiment_vader_lexicon(review,
     ...:                                                       threshold=0.1,
     ...:                                                       verbose=True)
     ...:     print '-'*60

Review:
Worst movie, (with the best reviews given it) I've ever seen. Over the top
dialog, acting, and direction. more slasher flick than thriller.With all the
great reviews this movie got I'm appalled that it turned out so silly. shame
on you martin scorsese

Labeled Sentiment: negative

    SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative Neutral
0            negative          0.03    20.0%    18.0%   62.0%
------------------------------------------------------------
Review:
I hope this group of film-makers never re-unites.

Labeled Sentiment: negative

    SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative Neutral
0            positive          0.44    33.0%     0.0%   67.0%
------------------------------------------------------------
Review:
no comment - stupid movie, acting average or worse... screenplay - no sense
at all... SKIP IT!

Labeled Sentiment: negative

    SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative Neutral
0            negative          -0.8     0.0%    40.0%   60.0%
------------------------------------------------------------
Review:
Add this little gem to your list of holiday regulars. It is<br /><br />sweet,
funny, and endearing
```

VADER Lexicon

```
Labeled Sentiment: positive

     SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative Neutral
0            positive           0.82    40.0%     0.0%   60.0%
-----------------------------------------------------------
Review:
a mesmerizing film that certainly keeps your attention... Ben Daniels is
fascinating (and courageous) to watch.

Labeled Sentiment: positive

     SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative Neutral
0            positive           0.71    31.0%     0.0%   69.0%
-----------------------------------------------------------
Review:
This movie is perfect for all the romantics in the world. John Ritter has
never been better and has the best line in the movie! "Sam" hits close to
home, is lovely to look at and so much fun to play along with. Ben Gazzara
was an excellent cast and easy to fall in love with. I'm sure I've met
Arthur in my travels somewhere. All around, an excellent choice to pick up
any evening.!:-)

Labeled Sentiment: positive

     SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative Neutral
0            positive           0.99    37.0%     2.0%   61.0%
-----------------------------------------------------------
Review:
I don't care if some people voted this movie to be bad. If you want the
Truth this is a Very Good Movie! It has every thing a movie should have. You
really should Get this one.

Labeled Sentiment: positive

     SENTIMENT STATS:
  Predicted Sentiment Polarity Score Positive Negative Neutral
0            negative          -0.16    17.0%    14.0%   69.0%
-----------------------------------------------------------
Review:
Worst horror film ever but funniest film ever rolled in one you have got
to see this film it is so cheap it is unbeliaveble but you have to see it
really!!!! P.s watch the carrot
```

```
# predict sentiment for test movie reviews dataset
vader_predictions = [analyze_sentiment_vader_lexicon(review, threshold=0.1)
                     for review in test_reviews]

# get model performance statistics
In [302]: print 'Performance metrics:'
     ...: display_evaluation_metrics(true_labels=test_sentiments,
     ...:                            predicted_labels=vader_predictions,
     ...:                            positive_class='positive')
     ...: print '\nConfusion Matrix:'
     ...: display_confusion_matrix(true_labels=test_sentiments,
     ...:                          predicted_labels=vader_predictions,
     ...:                          classes=['positive', 'negative'])
     ...: print '\nClassification report:'
     ...: display_classification_report(true_labels=test_sentiments,
     ...:                               predicted_labels=vader_predictions,
     ...:                               classes=['positive', 'negative'])
Performance metrics:
Accuracy: 0.7
Precision: 0.65
Recall: 0.86
F1 Score: 0.74

Confusion Matrix:
                Predicted:
                  positive negative
Actual: positive      6434     1076
        negative      3410     4080

Classification report:
             precision    recall  f1-score   support

   positive       0.65      0.86      0.74      7510
   negative       0.79      0.54      0.65      7490

avg / total       0.72      0.70      0.69     15000
```

Pattern Lexicon

- The pattern package is a complete package for NLP, text analytics, and information retrieval.
- For sentiment analysis, it analyzes any body of text by decomposing it into sentences and then tokenizing it and tagging the various tokens with necessary parts of speech.
- It contains scores like polarity, subjectivity, intensity, and confidence, along with other tags like the part of speech, WordNet identifier.
- It then leverages this lexicon to compute the overall polarity and subjectivity score associated with a text document.
- A threshold of 0.1 is recommended by pattern itself to compute the final sentiment of a document as positive, and anything below it as negative.
- You can also analyze the mood and modality of text documents by leveraging the mood and modality functions provided by the pattern package.
- The mood function helps in determining the mood expressed by a particular text document.
- This function returns INDICATIVE , IMPERATIVE , CONDITIONAL , or SUBJUNCTIVE for any text based on its content.

Pattern Lexicon

Mood	Form	Use	Example
INDICATIVE	none of the below	fact, belief	*It rains.*
IMPERATIVE	infinitive without *to*	command, warning	*Don't rain!*
CONDITIONAL	*would, could, should, may,* or *will, can* + *if*	conjecture	*It might rain.*
SUBJUNCTIVE	*wish, were,* or *it is* + infinitive	wish, opinion	*I hope it rains.*

- **Modality for any text represents the degree of certainty expressed by the text as a whole.**
- **This value is a number that ranges between 0 and 1. Values > 0.5 indicate factual texts having a high certainty, and < 0.5 indicate wishes and hopes and have a low certainty associated with them.**

```
from pattern.en import sentiment, mood, modality

def analyze_sentiment_pattern_lexicon(review, threshold=0.1,
                                      verbose=False):
    # pre-process text
    review = normalize_accented_characters(review)
    review = html_parser.unescape(review)
    review = strip_html(review)
    # analyze sentiment for the text document
    analysis = sentiment(review)
    sentiment_score = round(analysis[0], 2)
    sentiment_subjectivity = round(analysis[1], 2)
    # get final sentiment
    final_sentiment = 'positive' if sentiment_score >= threshold\
                                 else 'negative'
    if verbose:
        # display detailed sentiment statistics
        sentiment_frame = pd.DataFrame([[final_sentiment, sentiment_score,
                                         sentiment_subjectivity]],
                                         columns=pd.MultiIndex(levels
                                         =[['SENTIMENT STATS:'],
                                                   ['Predicted Sentiment',
                                                   'Polarity Score',
                                                    'Subjectivity Score']],
                                                    labels=[[0,0,0],
                                                    [0,1,2]]))

        print sentiment_frame
        assessment = analysis.assessments
        assessment_frame = pd.DataFrame(assessment,
                                  columns=pd.MultiIndex(levels=[['DETAILED
                                  ASSESSMENT STATS:'],
                                                 ['Key Terms', 'Polarity
                                                 Score',
                                                 'Subjectivity Score',
                                                 'Type']],
                                                 labels=[[0,0,0,0],
                                                 [0,1,2,3]]))
                                                 print assessment_frame
                                                 print


    return final_sentiment
```

```
Labeled Sentiment: negative

    SENTIMENT STATS:
  Predicted Sentiment Polarity Score Subjectivity Score
0            negative            0.0                0.0
Empty DataFrame
Columns: [(DETAILED ASSESSMENT STATS:, Key Terms), (DETAILED ASSESSMENT
STATS:, Polarity Score), (DETAILED ASSESSMENT STATS:, Subjectivity Score),
(DETAILED ASSESSMENT STATS:, Type)]
Index: []

------------------------------------------------------------
Review:
no comment - stupid movie, acting average or worse... screenplay - no sense
at all... SKIP IT!

Labeled Sentiment: negative

    SENTIMENT STATS:
  Predicted Sentiment Polarity Score Subjectivity Score
0            negative          -0.36                0.5
  DETAILED ASSESSMENT STATS:
                  Key Terms Polarity Score Subjectivity Score  Type
0                  [stupid]          -0.80                1.0  None
1                  [acting]           0.00                0.0  None
2                 [average]          -0.15                0.4  None
3                [worse, !]          -0.50                0.6  None

------------------------------------------------------------
Review:
Add this little gem to your list of holiday regulars. It is<br /><br
/>sweet, funny, and endearing

Labeled Sentiment: positive

    SENTIMENT STATS:
  Predicted Sentiment Polarity Score Subjectivity Score
0            positive          0.19                0.67
  DETAILED ASSESSMENT STATS:
                  Key Terms Polarity Score Subjectivity Score  Type
0                  [little]       -0.1875                0.5  None
1                   [funny]        0.2500                1.0  None
2               [endearing]        0.5000                0.5  None

------------------------------------------------------------
Review:
a mesmerizing film that certainly keeps your attention... Ben Daniels is
fascinating (and courageous) to watch.
```

Pattern Lexicon

Labeled Sentiment: positive

SENTIMENT STATS:

	Predicted Sentiment	Polarity Score	Subjectivity Score
0	positive	0.4	0.71

DETAILED ASSESSMENT STATS:

	Key Terms	Polarity Score	Subjectivity Score	Type
0	[mesmerizing]	0.300000	0.700000	None
1	[certainly]	0.214286	0.571429	None
2	[fascinating]	0.700000	0.850000	None

Review:
This movie is perfect for all the romantics in the world. John Ritter has never been better and has the best line in the movie! "Sam" hits close to home, is lovely to look at and so much fun to play along with. Ben Gazzara was an excellent cast and easy to fall in love with. I'm sure I've met Arthur in my travels somewhere. All around, an excellent choice to pick up any evening.!:-)

Labeled Sentiment: positive

SENTIMENT STATS:

	Predicted Sentiment	Polarity Score	Subjectivity Score
0	positive	0.66	0.73

DETAILED ASSESSMENT STATS:

	Key Terms	Polarity Score	Subjectivity Score	Type
0	[perfect]	1.000000	1.000000	None
1	[better]	0.500000	0.500000	None
2	[best, !]	1.000000	0.300000	None
3	[lovely]	0.500000	0.750000	None
4	[much, fun]	0.300000	0.200000	None
5	[excellent]	1.000000	1.000000	None
6	[easy]	0.433333	0.833333	None
7	[love]	0.500000	0.600000	None
8	[sure]	0.500000	0.888889	None
9	[excellent, !]	1.000000	1.000000	None
10	[:-)]	0.500000	1.000000	mood

Review:
I don't care if some people voted this movie to be bad. If you want the Truth this is a Very Good Movie! It has every thing a movie should have. You really should Get this one.

Review:
Worst movie, (with the best reviews given it) I've ever seen. Over the top dialog, acting, and direction. more slasher flick than thriller.With all the great reviews this movie got I'm appalled that it turned out so silly. shame on you martin scorsese
Labeled Sentiment: negative
Mood: indicative
Modality Score: 0.75
Certainty: Strong

Review:
I hope this group of film-makers never re-unites.
Labeled Sentiment: negative
Mood: subjunctive
Modality Score: -0.25
Certainty: Low

Review:
no comment - stupid movie, acting average or worse... screenplay - no sense at all... SKIP IT!
Labeled Sentiment: negative
Mood: indicative
Modality Score: 0.75
Certainty: Strong

Review:
Add this little gem to your list of holiday regulars. It is

sweet, funny, and endearing
Labeled Sentiment: positive
Mood: imperative
Modality Score: 1.0
Certainty: Strong

Review:
a mesmerizing film that certainly keeps your attention... Ben Daniels is fascinating (and courageous) to watch.
Labeled Sentiment: positive
Mood: indicative
Modality Score: 0.75
Certainty: Strong

Review:
This movie is perfect for all the romantics in the world. John Ritter has never been better and has the best line in the movie! "Sam" hits close to home, is lovely to look at and so much fun to play along with. Ben Gazzara was an excellent cast and easy to fall in love with. I'm sure I've met Arthur in my travels somewhere. All around, an excellent choice to pick up any evening.!:-)

Pattern Lexicon

```
# predict sentiment for test movie reviews dataset
pattern_predictions = [analyze_sentiment_pattern_lexicon(review,
                       threshold=0.1)
                        for review in test_reviews]

# get model performance statistics
In [307]: print 'Performance metrics:'
     ...: display_evaluation_metrics(true_labels=test_sentiments,
     ...:                            predicted_labels=pattern_predictions,
     ...:                            positive_class='positive')
     ...: print '\nConfusion Matrix:'
     ...: display_confusion_matrix(true_labels=test_sentiments,
     ...:                          predicted_labels=pattern_predictions,
     ...:                          classes=['positive', 'negative'])
```

```
     ...: print '\nClassification report:'
     ...: display_classification_report(true_labels=test_sentiments,
     ...:                               predicted_labels=pattern_
                                        predictions,
     ...:                               classes=['positive', 'negative'])
Performance metrics:
Accuracy: 0.77
Precision: 0.76
Recall: 0.79
F1 Score: 0.77

Confusion Matrix:
                Predicted:
                  positive negative
Actual: positive      5958     1552
        negative      1924     5566

Classification report:
             precision    recall  f1-score   support

   positive       0.76      0.79      0.77      7510
   negative       0.78      0.74      0.76      7490

avg / total       0.77      0.77      0.77     15000
```

TextBlob Lexicon

- Typically, specific adjectives have a polarity score (negative/positive, -1.0 to +1.0) and a subjectivity score (objective/subjective, +0.0 to +1.0).
- The reliability score specifies if an adjective was hand-tagged (1.0) or inferred (0.7).
- Words are tagged per sense, e.g., ridiculous (pitiful) = negative, ridiculous (humorous) = positive.
- The Cornetto id (lexical unit id) and Cornetto synset id refer to the Cornetto lexical database for Dutch.
- The WordNet id refers to the WordNet3 lexical database for English. The part-of-speech tags (POS) use the Penn Treebank convention.

```
<word form="abhorrent" wordnet_id="a-1625063" pos="JJ" sense="offensive
to the mind" polarity="-0.7" subjectivity="0.8" intensity="1.0"
reliability="0.9" />
<word form="able" cornetto_synset_id="n_a-534450" wordnet_id="a-01017439"
pos="JJ" sense="having a strong healthy body" polarity="0.5"
subjectivity="1.0" intensity="1.0" confidence="0.9" />
```

```
for review, sentiment in zip(test_reviews[sample_review_ids], test_
sentiments[sample_review_ids]):
    print('REVIEW:', review)
    print('Actual Sentiment:', sentiment)
    print('Predicted Sentiment polarity:', textblob.TextBlob(review).
    sentiment.polarity)
    print('-'*60)
```

```
REVIEW: no comment - stupid movie, acting average or worse... screenplay -
no sense at all... SKIP IT!
Actual Sentiment: negative
Predicted Sentiment polarity: -0.3625
------------------------------------------------------------
REVIEW: I don't care if some people voted this movie to be bad. If you want
the Truth this is a Very Good Movie! It has every thing a movie should
have. You really should Get this one.
Actual Sentiment: positive
Predicted Sentiment polarity: 0.16666666666666674
------------------------------------------------------------
```

TextBlob Lexicon

```
REVIEW: Worst horror film ever but funniest film ever rolled in one you
have got to see this film it is so cheap it is unbelievable but you have to
see it really!!!! P.s watch the carrot
Actual Sentiment: positive
Predicted Sentiment polarity: -0.037239583333333326
------------------------------------------------------------
```

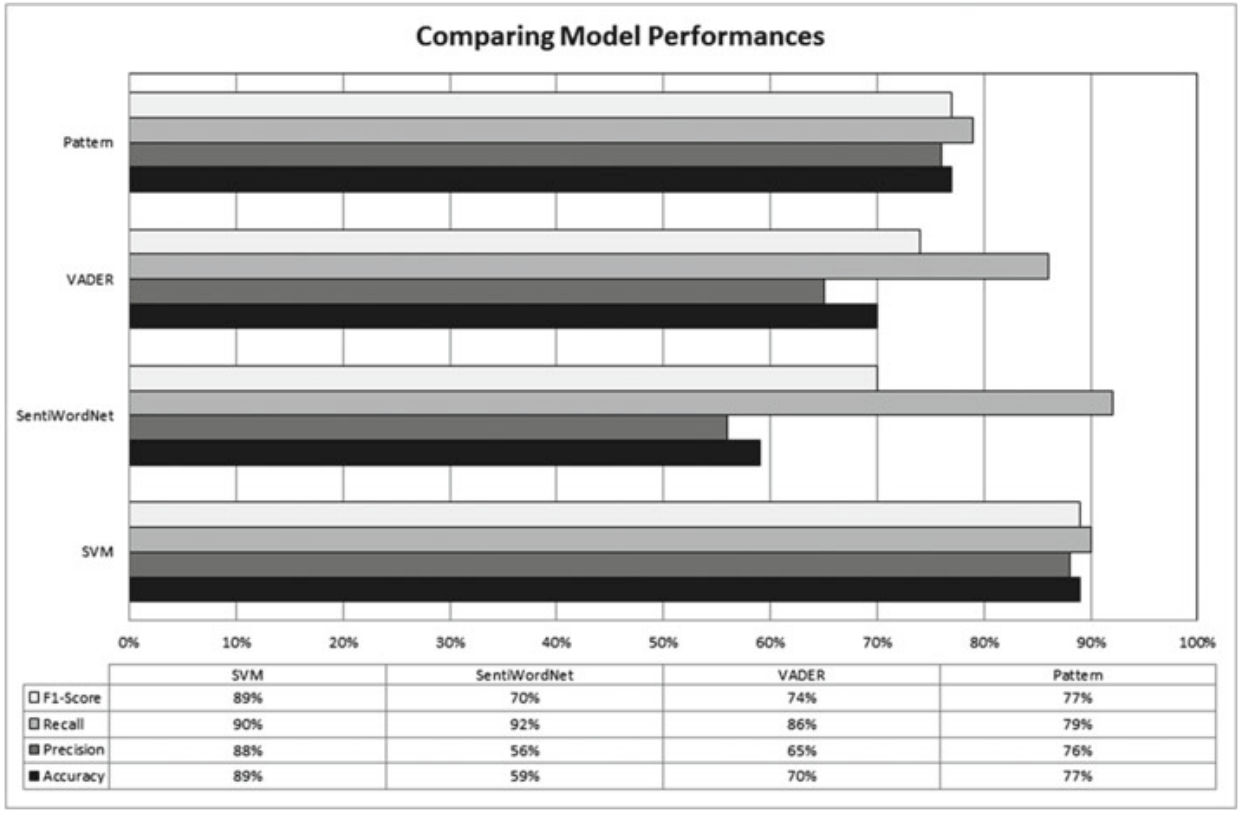
You can check the sentiment of some specific movie reviews and the sentiment polarity score as predicted by TextBlob. Typically, a positive score denotes positive sentiment and a negative score denotes negative sentiment. You can use a specific custom threshold to determine what should be positive or negative. We use a custom threshold of 0.1 based on multiple experiments. The following code computes the sentiment on the entire test data. See Figure 9-1.

```
sentiment_polarity = [textblob.TextBlob(review).sentiment.polarity for
                      review in test_reviews]
predicted_sentiments = ['positive' if score >= 0.1 else 'negative'
                                for score in sentiment_polarity]
meu.display_model_performance_metrics(true_labels=test_sentiments,
                                      predicted_labels=predicted_sentiments,
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.7669
Precision: 0.767
Recall: 0.7669
F1 Score: 0.7668

Model Classification report:
------------------------------
                precision    recall  f1-score   support

      positive       0.76      0.78      0.77      7510
      negative       0.77      0.76      0.76      7490

avg / total          0.77      0.77      0.77     15000

Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive       5835     1675
        negative       1822     5668
```

Figure 9-1. Model performance metrics for pattern lexicon based model

Comparing Sentiment analysis model performances



	SVM	SentiWordNet	VADER	Pattern
F1-Score	89%	70%	74%	77%
Recall	90%	92%	86%	79%
Precision	88%	56%	65%	76%
Accuracy	89%	59%	70%	77%

Thank You!!!