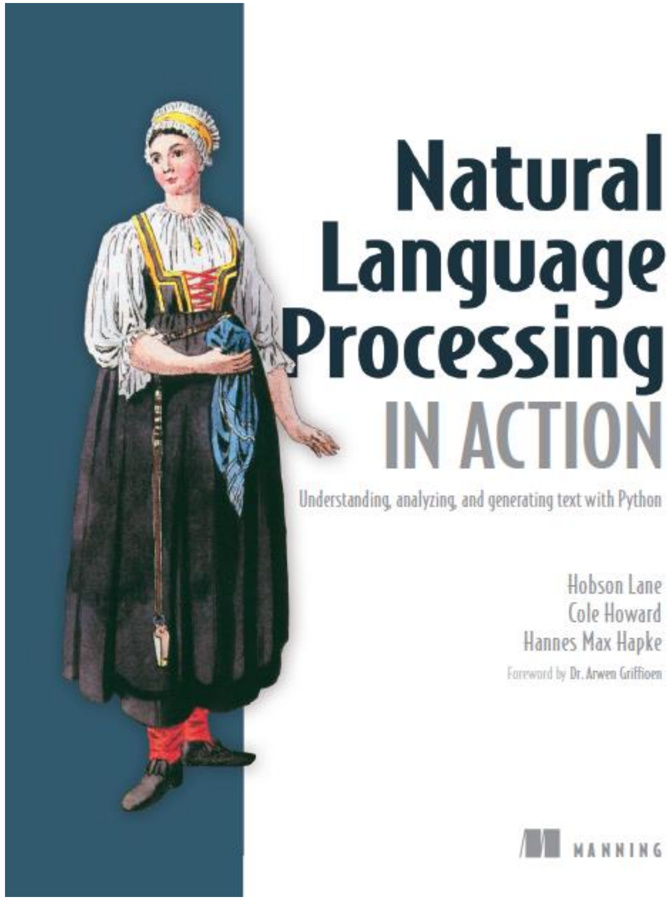
Semantic Analysis

Prof. Gheith Abandah

Reference 1

- Chapter 4: **Finding meaning in word counts (semantic analysis)**



- H. Lane, C. Howard, and H. Hapke, **Natural Language Processing in Action:** Understanding, analyzing, and generating text with Python, Manning, 2019.

Outline

- Limitations of TF-IDF Vectors
- Manual Creation of Topics
- Topic Modeling Algorithms
- Latent Semantic Analysis (LSA)
- LDA Classifier for Two Document Classes
- PCA for Finding Topics in Documents
- Summary

Limitations of TF-IDF Vectors

- TF-IDF treats words independently, ignoring synonyms and morphology.
- Example: "play" and "playing" are treated differently, even though they convey similar meaning.
- Lemmatization reduces words to their base form (lemma) - "play" and "playing" become "play".
- Topic vectors capture higher-level themes beyond individual words.

Manual Creation of Topics

- Select a subset of texts from a corpus.
- Identify common themes or subjects within these texts.
- Group related words under these common themes manually.
- Assign a label to each group, creating a 'topic'.
- Review and refine topics for consistency and relevance.

```
>>> topic['petness'] = (.3 * tfidf['cat'] +\
...                     .3 * tfidf['dog'] +\
...                      0 * tfidf['apple'] +\
...                      0 * tfidf['lion'] -\
...                     .2 * tfidf['NYC'] +\
...                     .2 * tfidf['love'])
>>> topic['animalness']  = (.1 * tfidf['cat']  +\
...                         .1 * tfidf['dog'] -\
...                         .1 * tfidf['apple'] +\
...                         .5 * tfidf['lion'] +\
...                         .1 * tfidf['NYC'] -\
...                         .1 * tfidf['love'])
>>> topic['cityness']    = ( 0 * tfidf['cat']  -\
...                         .1 * tfidf['dog'] +\
...                         .2 * tfidf['apple'] -\
...                         .1 * tfidf['lion'] +\
...                         .5 * tfidf['NYC'] +\
...                         .1 * tfidf['love'])
```

"Hand-crafted" weights (.3, .3, 0, 0, -.2, .2) are multiplied by imaginary tfidf values to create topic vectors for your imaginary random document. You'll compute real topic vectors later.

J. R. Firth Observation-based Topic Modeling

- Based on the observation by linguist J. R. Firth: "*You shall know a word by the company it keeps*“ (1957).
- Analyze word co-occurrence patterns to identify thematic clusters.
- Words that frequently appear together are likely related to a similar topic.
- Example: "dog" and "bone" might suggest a topic related to pets.

Algorithms

- Latent semantic analysis (LSA)
- Linear discriminant analysis (LDA)
- Principal component analysis (PCA)
- Latent Dirichlet allocation (LDiA)

Latent Semantic Analysis (LSA)

- Latent means present but not visible.
- LSA uses singular value decomposition (SVD) on the TF-IDF matrix.
- It identifies patterns in the relationships between the terms and concepts.
- Reduces the dimensionality of the text data.
- Helps in identifying synonymy (big and large) and polysemy (multiple meanings) within the corpus.
- Generates a latent structure of concepts.

LDA Classifier for Two Document Classes

- Linear Discriminant Analysis (LDA) is a supervised learning algorithm for classification.
- Assumes data is normally distributed.
- Seeks to reduce dimensions while preserving class separation.
- Maximizes the ratio of between-class variance to within-class variance in any particular dataset, thereby ensuring maximum separability.
- In NLP, LDA helps to classify documents by finding a linear combination of features that characterizes or separates two classes (e.g., spam vs. non-spam).
- The algorithm finds a decision boundary that best separates the two classes.
- Can be particularly effective when the number of features (words) is high.

The SMS Spam Dataset Example

```
import pandas as pd
from nlpia.data.loaders import get_data
pd.options.display.width = 120

sms = get_data('sms-spam')
index = ['sms{}{}'.format(i, '!'*j) for (i,j) in
         zip(range(len(sms)), sms.spam)]
sms = pd.DataFrame(sms.values, columns =
                   sms.columns, index=index)
mask = sms.spam.astype(bool).values
sms['spam'] = sms.spam.astype(int)
```

```
>>> sms.head(6)
       spam                                               text
sms0      0  Go until jurong point, crazy.. Available only ...
sms1      0                      Ok lar... Joking wif u oni...
sms2!     1  Free entry in 2 a wkly comp to win FA Cup fina...
sms3      0  U dun say so early hor... U c already then say...
sms4      0  Nah I don't think he goes to usf, he lives aro...
sms5!     1  FreeMsg Hey there darling it's been 3 week's n...
```

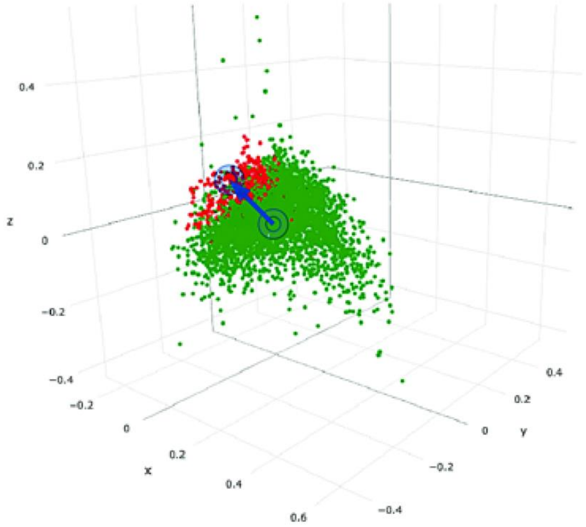
```python
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize.casual import casual_tokenize

tfidf_model = TfidfVectorizer(tokenizer=casual_tokenize)
tfidf_docs = tfidf_model.fit_transform(raw_documents=
                                        sms.text).toarray()
```

```python
>>> tfidf_docs.shape
(4837, 9232)                 # 4,837 documents and 9,232 words
>>> sms.spam.sum()
638
```

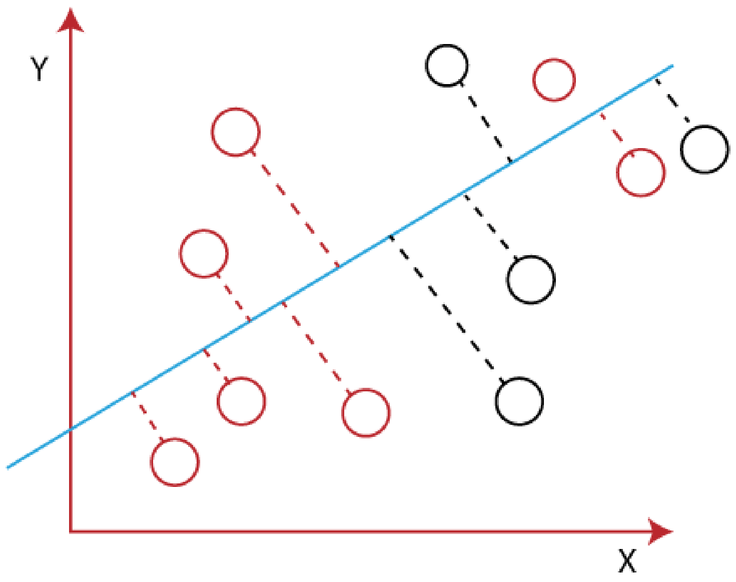
```python
mask = sms.spam.astype(bool).values
spam_centroid = tfidf_docs[mask].mean(axis=0)
ham_centroid = tfidf_docs[~mask].mean(axis=0)
# subtracting the centroids gives the line between them.
# This is the LDA model.

>>> spam_centroid.round(2)
array([0.06, 0.  , 0.  , ..., 0.  , 0.  , 0.  ])
>>> ham_centroid.round(2)
array([0.02, 0.01, 0.  , ..., 0.  , 0.  , 0.  ])
```



```
# The dot product computes the projection of each vector
#    on the line between the centroids.
spamminess_score = tfidf_docs.dot(spam_centroid - ham_centroid)

>>> spamminess_score
array([-0.01469806, -0.02007376,  0.03856095, ..., -0.01014774,
       -0.00344281,  0.00395752])
```



```
# To facilitate classification, scale scores [0, 1]
from sklearn.preprocessing import MinMaxScaler
sms['lda_score'] = MinMaxScaler().fit_transform(
                        spamminess_score.reshape(-1,1))
sms['lda_predict'] = (sms.lda_score > .5).astype(int)

>>> sms['spam lda_predict lda_score'.split()].round(2).head(6)
       spam  lda_predict  lda_score
sms0      0            0       0.23
sms1      0            0       0.18
sms2!     1            1       0.72
sms3      0            0       0.18
sms4      0            0       0.29
sms5!     1            1       0.55
# Classification accuracy of 97.7% with threshold 0.5
```

PCA for Finding Topics in Documents

- Principal component analysis (PCA) is a statistical technique to find patterns in data.
- It reduces the dimensionality while preserving variance.
- When applied to TF-IDF, it can reveal the underlying topics.
- PCA finds the principal components that can represent topics.

Finding SMS Spam Topics with PCA

```
from sklearn.decomposition import PCA

pca = PCA(n_components=16)
pca = pca.fit(tfidf_docs)
pca_topic_vectors = pca.transform(tfidf_docs)
pca_topic_vectors = pd.DataFrame(pca_topic_vectors,
                                 columns=['topic{}'.format(i) for i
                                          in range(16)])
>>> pca_topic_vectors.round(3).head()
       topic0  topic1  topic2   ...    topic13  topic14  topic15
sms0    0.201   0.003   0.037   ...     -0.026   -0.019    0.039
sms1    0.404  -0.094  -0.078   ...     -0.036    0.047   -0.036
sms2!  -0.030  -0.048   0.090   ...     -0.017   -0.045    0.057
sms3    0.329  -0.033  -0.035   ...     -0.065    0.022   -0.076
sms4    0.002   0.031   0.038   ...      0.031   -0.081   -0.020
```

Summary

- You can use semantic analysis to decompose and transform TF-IDF and BOW vectors into topic vectors.
- Use LDiA when you need to compute explainable topic vectors.
- No matter how you create your topic vectors, they can be used for semantic search to find documents based on their meaning.
- Topic vectors can be used to predict whether a social post is spam or is likely to be "liked."