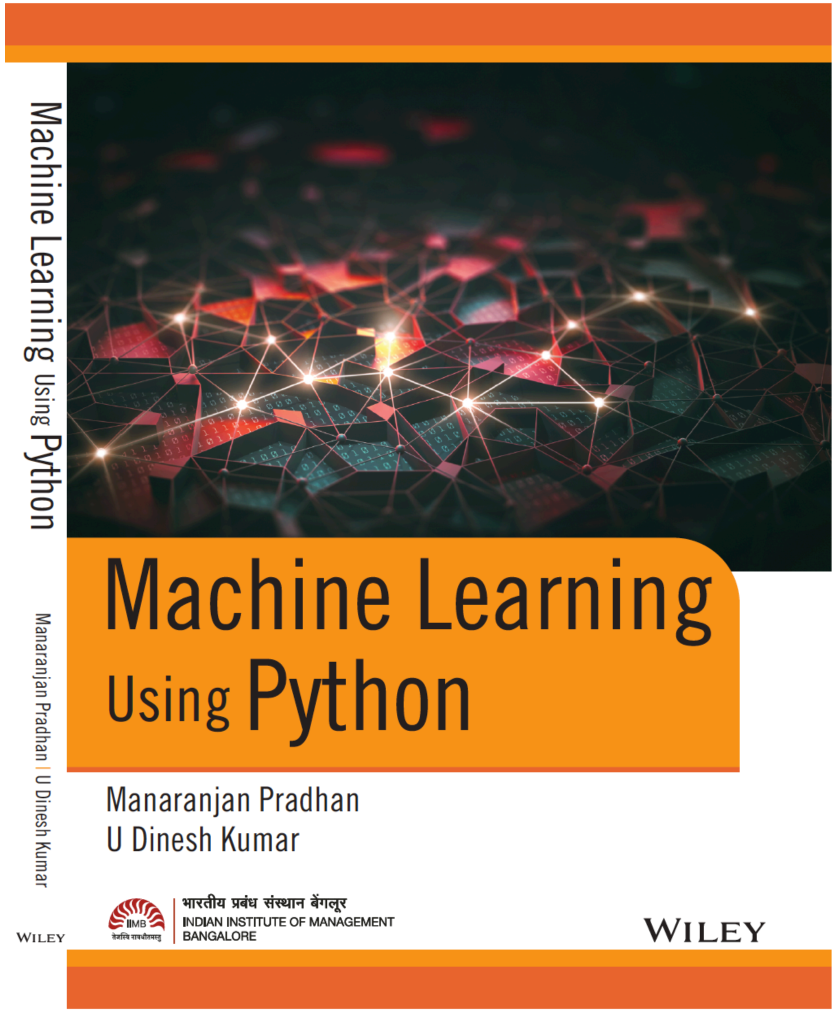


Chapter 10: Text Analytics

Learning Objectives

- Understand the challenges associated with the handling of text data.
- Learn various pre-processing steps to prepare text data for modelling.
- Learn Naïve–Bayes classification algorithm.
- Learn to develop model for sentiment classification.

SENTIMENT CLASSIFICATION

Dataset - *https://www.kaggle.com/c/si650winter11/* data (the original data was contributed by the University of Michigan). The data consists of sentiments expressed by users on various movies

```
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings('ignore')

train_ds = pd.read_csv("sentiment_train", delimiter="\t")
train_ds.head(5)
```

	Sentiment	Text
0	1	The Da Vinci Code book is just awesome.
1	1	this was the first clive cussler i've ever read, but even books like Relic, and Da Vinci code were more plausible than this.
2	1	i liked the Da Vinci Code a lot.
3	1	i liked the Da Vinci Code a lot.
4	1	I liked the Da Vinci Code but it ultimatly didn't seem to hold it's own.

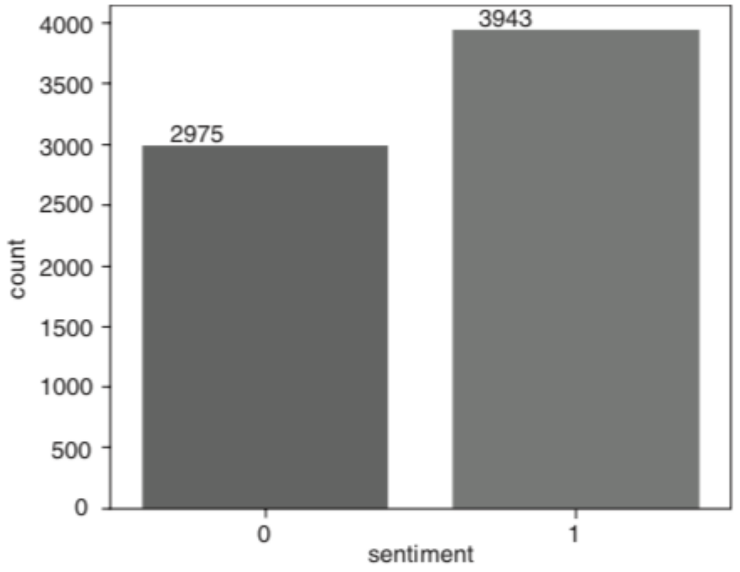
EXPLORING THE DATASET

```
train_ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6918 entries, 0 to 6917
Data columns (total 2 columns):
sentiment    6918 non-null int64
text         6918 non-null object
dtypes: int64(1), object(1)
memory usage: 108.2+ KB
```

```
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline

plt.figure(figsize=(6,5))
# Create count plot
ax = sn.countplot(x='sentiment', data=train_ds)
# Annotate
for p in ax.patches:
        ax.annotate(p.get_height(), (p.get_x()+0.1,
        p.get_height()+50))
```



BAG-OF-WORDS (BOW) MODEL

Count Vector Model

1. **Document 1 (positive sentiment):** I really really like IPL.
2. **Document 2 (negative sentiment):** I never like IPL.

Documents	x1	x2	x3	x4	x5	y
	I	really	never	like	ipl	
I really really like ipl	1	2	0	1	1	1
I never like ipl	1	0	1	1	1	0

BAG-OF-WORDS (BOW) MODEL

<u>Term Frequency Vector Model</u>

1. Document 1 (positive sentiment): I really really like IPL.
2. Document 2 (negative sentiment): I never like IPL.

$$\text{Term Frequency } (TF_i) = \frac{\text{Number of occurrences of word } i \text{ in the document}}{\text{Total number of words in the document}}$$

	x1	x2	x3	x4	x5	y
	I	really	never	like	ipl	
I really really like ipl	0.2	0.4	0	0.2	0.2	1
I never like ipl	0.25	0	0.25	0.25	0.25	0

BAG-OF-WORDS (BOW) MODEL

Term Frequency-Inverse Document Frequency (TF-IDF)

1. Document 1 (positive sentiment): I really really like IPL.
2. Document 2 (negative sentiment): I never like IPL.

$$TF-IDF_i = TF_i \times \ln\left(1+\frac{N}{N_i}\right)$$

	x1	x2	x3	x4	x5
IDF Values	0.693	1.098	1.098	0.693	0.693

	x1	x2	x3	x4	x5	y
	1	really	never	like	ipl	
I really really like ipl	0.1386	0.4394	0.0	0.1386	0.1386	1
I never like ipl	0.1732	0.0	0.2746	0.1732	0.1732	0

COUNT VECTORS - SENTIMENT_TRAIN DATASET

```
from sklearn.feature_extraction.text import CountVectorizer
# Initialize the CountVectorizer
count_vectorizer = CountVectorizer()
# Create the dictionary from the corpus
feature_vector = count_vectorizer.fit(train_ds.text)
# Get the feature names features = feature_vector.get_feature_names()
print("Total number of features: ", len(features))
```

```
Total number of features: 2132
```

```
train_ds_features = count_vectorizer.transform(train_ds.text)
train_ds_features.shape
```

```
(6918, 2132)
```

```
train_ds_features.getnnz()
```

```
65398
```

DISPLAY DOCUMENT VECTORS

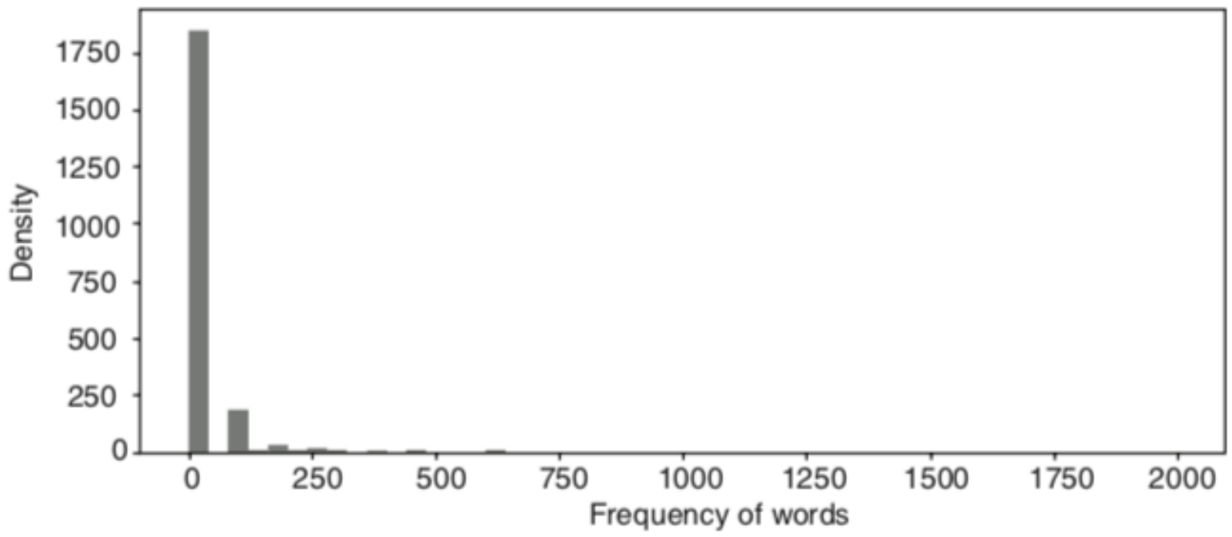
Converting the matrix to a dataframe
```
train_ds_df = pd.DataFrame(train_ds_features.todense())
```
Setting the column names to the features i.e. words
```
train_ds_df.columns = features
```
train_ds[0:1]

	Sentiment	Text
0	1	The Da Vinci Code book is just awesome.

train_ds_df[[‘the’, ‘da’, “vinci”, “code”, “book”, ‘is’, ‘just’, ‘awesome’]][0:1]

	the	da	vinci	code	book	is	Just	awesome
0	1	1	1	1	1	1	1	1

REMOVING LOW-FREQUENCY WORDS



- Count Number of features by count equal to 1

```
1228
```

- Restrict features by setting max_features = 1000

	Counts	Features
866	3306	The
37	2154	And
358	2093	Harry
675	2093	potter
138	2002	Code
934	2001	Vinci
178	2001	Da
528	2000	mountain
104	2000	brokeback
488	1624	Love
423	1520	Is
941	1176	Was
60	1127	awesome
521	1094	mission
413	1093	impossible

REMOVING STOP WORDS

```
from sklearn.feature_extraction import
text my_stop_words = text.ENGLISH_STOP_WORDS

# Adding custom words to the list of stop
words
my_stop_words =
text.ENGLISH_STOP_WORDS.union([‘harry’,
‘potter’, ‘code’, ‘vinci’, ‘da’,’harry’,
‘mountain’, ‘movie’, ‘movies’])

# Setting stop words list
count_vectorizer = CountVectorizer(stop_words
= my_stop_words, max_features = 1000)
```

	Counts	Features
73	2000	brokeback
408	1624	love
39	1127	awesome
436	1094	mission
341	1093	impossible
390	974	like
745	602	sucks
743	600	sucked
297	578	hate
652	374	really
741	365	stupid
362	287	just
374	276	know
742	276	suck
409	256	loved

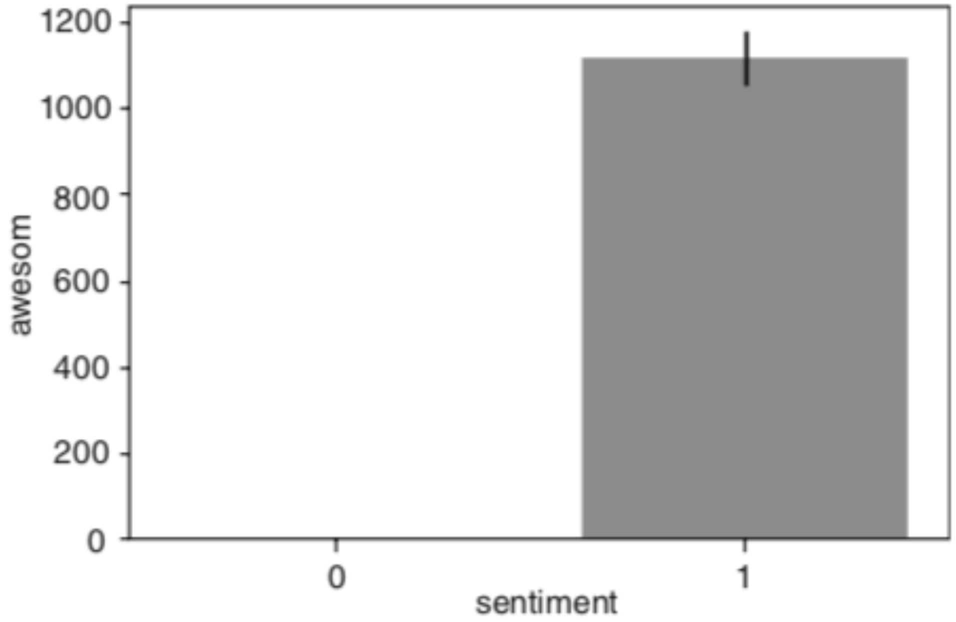
STEMMING & LEMMATIZATION

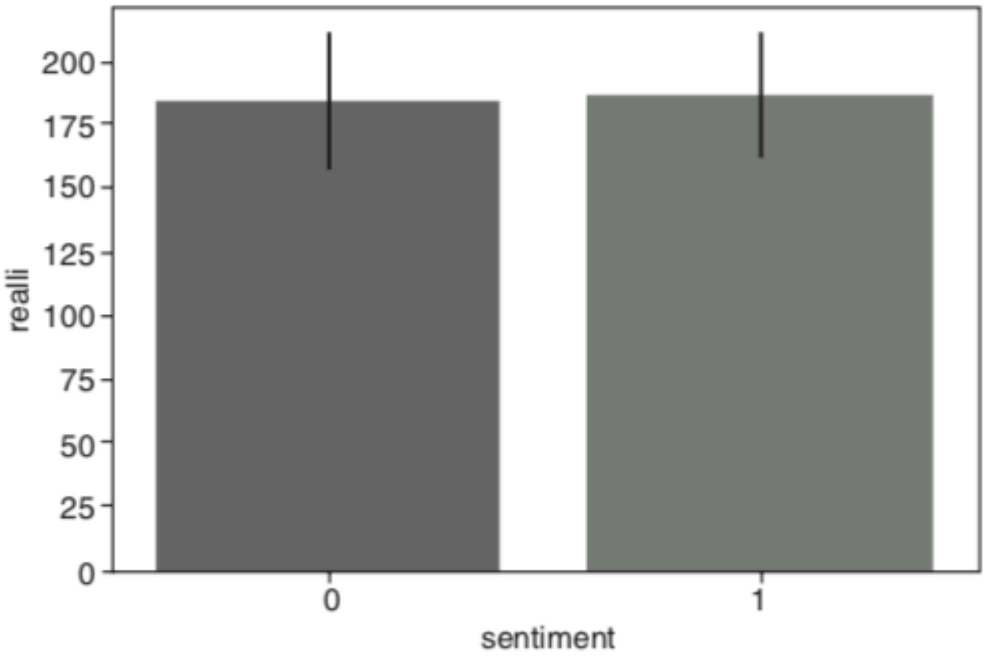
- **Stemming:** Remove the differences between inflected forms of a word to reduce each word to its root form
  - PorterStemmer
  - LancasterStemmer

- **Lemmatization:** Takes the morphological analysis of the words, It uses a language dictionary to convert the words to the root word
  - WordNetLemmatizer

	Counts	Features
80	1930	brokeback
297	1916	harri
407	1837	love
803	1378	suck
922	1142	wa
43	1116	awesom
345	1090	imposs
433	1090	mission
439	1052	movi
393	823	like
299	636	hate
54	524	becaus
604	370	realli
796	364	stupid
379	354	know

DISTRIBUTION OF WORDS ACROSS DIFFERENT SENTIMENT





NAÏVE–BAYES MODEL FOR SENTIMENT CLASSIFICATION

$$P(doc = +ve \mid word = awesome) \propto P(word = awesome \mid doc = +ve) * P(doc = +ve)$$

$$P(doc = +ve \mid word = W_1, W_2, \ldots W_N) \propto \prod_{i=1}^{N} P_i(word = W_i \mid doc = +ve) * P(doc = +ve)$$

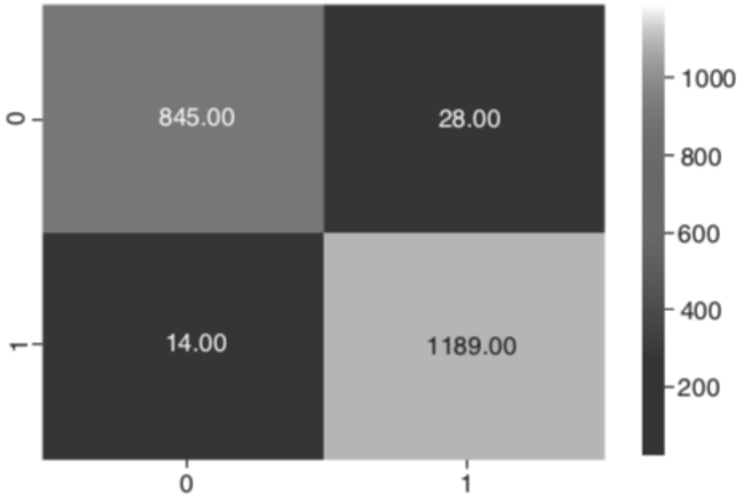
```
from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(train_ds_
features, train_ds.sentiment,
test_size = 0.3, random_state = 42)
from sklearn.naive_bayes import BernoulliNB nb_clf = BernoulliNB()

test_ds_predicted = nb_clf.predict(test_X.toarray())
```

MODEL ACCURACY

from sklearn import `metrics print(metrics.classification_report(test_y, test_ds_predicted))`

	Precision	recall	f1-score	support
0	0.98	0.97	0.98	873
1	0.98	0.99	0.98	1203
avg / total	0.98	0.98	0.98	2076



USING TF-IDF VECTORIZER

```
tfidf_vectorizer = TfidfVectorizer(analyzer=stemmed_words, max_features = 1000)
feature_vector = tfidf_vectorizer.fit(train_ds.text)
train_ds_features = tfidf_vectorizer.transform(train_ds.text)
features = feature_vector.get_feature_names()

from sklearn.naive_bayes import GaussianNB
train_X, test_X, train_y, test_y = train_test_split(train_ds_features,
train_ds.sentiment,
test_size = 0.3, random_state = 42)
```

	Precision	Recall	f1-score	support
0	0.96	0.96	0.96	873
1	0.97	0.97	0.97	1203
avg/total	0.97	0.97	0.97	2076

CHALLENGES OF TEXT ANALYTICS

- **Building model using N-Grams**

```
tfidf_vectorizer = TfidfVectorizer(max_features=500,
stop_words='english', tokenizer=get_stemmed_okens, ngram_range=(1,2))
```

```
train_X, test_X, train_y, test_y = train_test_split(train_ds_features,
train_ds.sentiment, test_size = 0.3, random_state = 42)
nb_clf = BernoulliNB()
nb_clf.fit(train_X.toarray(), train_y)
test_ds_predicted = nb_clf.predict(test_X.toarray())
```

	precision	recall	f1-score	support
0	1.00	0.94	0.97	873
1	0.96	1.00	0.98	1203
avg/total	0.97	0.97	0.97	2076

Thank You!