Data Mining & Informatics Lecture #6

Amin Noroozi

University of Wolverhampton

✉ a.noroozifakhabi@wlv.ac.uk

https://www.linkedin.com/in/amin-n-148350218/

Reminder

- **Extra lessons**

- Extra lessons will be on Tuesday 5-7 pm

Natural Language Processing (NLP

NLP

- **What is NLP**

Natural Language Processing is a data science and data mining technique that allows computers to understand language naturally, as a person does

This means the laptop will comprehend sentiments, speech, answer questions, text summarisation, etc

In nearly all NLP applications we need Data cleaning and Data exploratory analysis

NLP – Data Cleaning

- **Data cleaning**

The raw text data comes directly after the various sources are not cleaned. The first step is to clean data.

```
import pandas as pd
text = [‘This is the NLP LECTURE by prof Amin** ‘,’IN this lecture I”ll be
explaining various DATA-CLEANING techniques’,
 ‘So stay tuned for FURther More &&’,'Nah I don"t think he goes to usf, he
lives around']
df = pd.DataFrame({'text':text})
```

NLP – Data Cleaning

- **Lowercasing**

The method lower()converts all uppercase characters into lowercase and returns

Example:

```
df['lower'] = df['text'].apply(lambda x: " ".join(x.lower()  for x in x.split()))
```

NLP – Data Cleaning

- **Punctuation Removal**

Removing punctuation(*,&,%#@#()) is a crucial step since punctuation doesn’t add any extra information or value to our data. Hence, removing punctuation reduces the data size; therefore, it improves computational efficiency.

Example:

```
# Rmoving punctuation using string module
df.text.apply(lambda x:''.join(i for i in x if i not in string.punctuation))

# Removing punctuation using regular expression
import re
df.text.apply(lambda x:''.join(re.findall(r'[a-zA-Z+" "]',x)))
```

NLP – Data Cleaning

- **Stop Words Removal**

Words that frequently occur in sentences and carry no significant meaning in sentences. These are not important for prediction, so we remove stopwords to reduce data size and prevent overfitting. Note: Before filtering stopwords, make sure you lowercase the data since our stopwords are lowercase.

Example:

```
# !pip install nltk
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
allstopwords = stopwords.words('english')
df.lower.apply(lambda x: " ".join(i for i in x.split() if i not in allstopwords))
```

NLP – Data Cleaning

- **Spelling Correction**

Most of the text data extracted from customer reviews, blogs, or tweets have some chances of spelling mistakes.

Correcting spelling mistakes improves model accuracy.

There are various libraries to fix spelling mistakes, but the most convenient method is to use a text blob.

The method correct() works on text blob objects and corrects spelling mistakes.

NLP – Data Cleaning

- **Spelling Correction**

Example:

```
from textblob import TextBlob
df['lower'].apply(lambda x: str(TextBlob(x).correct()))
```

NLP – Data Cleaning

- **Tokenization**

Tokenization means splitting text into meaningful unit words. There are sentence tokenizers as well as word tokenizers.

Sentence tokenizer splits a paragraph into meaningful sentences, while word tokenizer splits a sentence into unit meaningful words. Many libraries can perform tokenization like SpaCy, NLTK, and TextBlob

Example

```
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
mystring = "My favorite animal is cat"
nltk.word_tokenize(mystring)
mystring.split(" ")
```

NLP – Data Cleaning

- **Stemming**

Stemming is converting words into their root word using some set of rules irrespective of meaning. I.e.,

“fish,” “fishes,” and “fishing” are stemmed into “fish”.

“playing”, “played”,” plays” are stemmed into “play”.

Stemming helps to reduce the vocabulary hence improving the accuracy.

NLP – Data Cleaning

- **Stemming**

The simplest way to perform stemming is to use NLTK or a TextBlob library.

NLTK provides various stemming techniques, i.e. Snowball, PorterStemmer; different technique follows different sets of rules to convert words into their root word.

Example:

```
import nltk
from nltk.stem import PorterStemmer
st = PorterStemmer()
df['text'].apply(lambda x:" ".join([st.stem(word) for word in x.split()]))
```

NLP – Data Cleaning

- **Lemmatization**

Lemmatization is converting words into their root word using vocabulary mapping. Lemmatization is done with the help of part of speech and its meaning; hence it doesn’t generate meaningless root words. But lemmatization is slower than stemming

“good,” “better,” or “best” is lemmatized into “good“.

Lemmatization will convert all synonyms into a single root word. i.e. “automobile“, “car“,” truck“,” vehicles” are lemmatized into “automobile”.

Lemmatization usually gets better results than stemming

NLP – Data Cleaning

- **Lemmatization**

Example:

```
import textblob
from textblob import Word
df['text'].apply(lambda x: " ".join([Word(word).lemmatize() for word in x.split()]))
```

import nltk nltk.Freq_dist(all_words)

NLP – Exploratory Data Analysis

- **Word Frequency in Data**

Counting the unique words in our data gives an idea about our data’s most frequent, and least frequent terms. Often we drop the least frequent comments to make our model training more generalised.

Example:

```
all_words = []
for sentence in df['text']:
    all_words.extend(sentence.split())
import nltk
nltk.FreqDist(all_words)
```

import nltk nltk.Freq_dist(all_words)

NLP – Exploratory Data Analysis

- **Word Cloud**

Wordcloud is the pictorial representation of the word frequency of the dataset.WordCloud is easier to understand and gives a better idea about our textual data.

The library wordcloud Let us create a word cloud in a few lines of code.

import nltk nltk.Freq_dist(all_words)

NLP – Exploratory Data Analysis

- **Word Cloud**

Example:

```
from wordcloud import WordCloud
from wordcloud import STOPWORDS
import matplotlib.pyplot as plt
words = []
for message in df[text']:
    words.extend([word for word in message.split() if word not in STOPWORDS])

wordcloud = WordCloud(width = 1000, height = 500).generate(" ".join(words))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

import nltk nltk.Freq_dist(all_words)

Sentiment Analysis

- **What is Sentiment Analysis**

Sentiment Analysis is a use case of Natural Language Processing (NLP) and comes under the category of text classification.

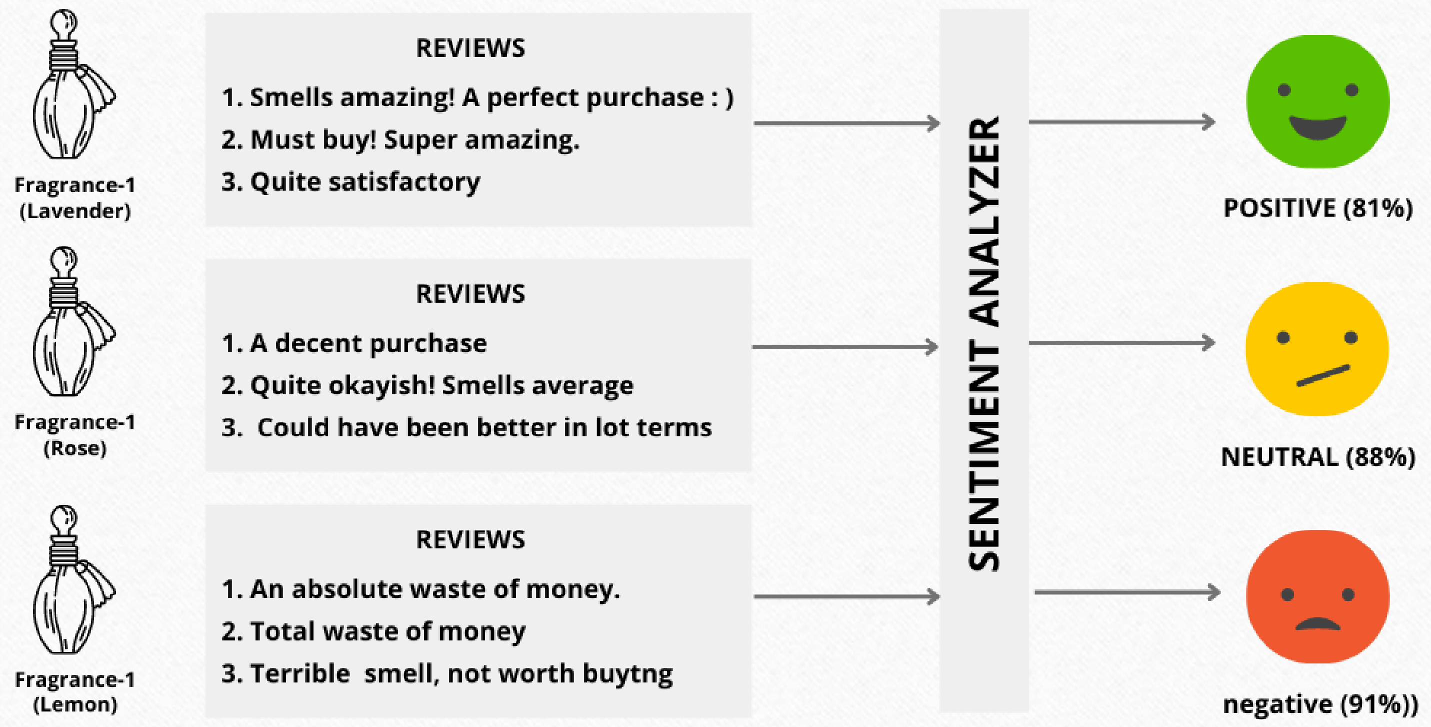
Sentiment analysis is a technique that detects the underlying sentiment in a piece of text.

It is the process of classifying text as either positive, negative, or neutral. Machine learning techniques are used to evaluate a piece of text and determine the sentiment behind it.

import nltk nltk.Freq_dist(all_words)

Sentiment Analysis

- **What is Sentiment Analysis**



REVIEWS

1. Smells amazing! A perfect purchase :)
2. Must buy! Super amazing.
3. Quite satisfactory

REVIEWS

1. A decent purchase
2. Quite okayish! Smells average
3. Could have been better in lot terms

REVIEWS

1. An absolute waste of money.
2. Total waste of money
3. Terrible smell, not worth buytng

import nltk nltk.Freq_dist(all_words)

Sentiment Analysis

- **Sentiment Analysis Use Cases**

Social Media Monitoring for Brand Management: Brands can use sentiment analysis to gauge their Brand's public outlook. For example, a company can gather all Tweets with the company's mention or tag and perform sentiment analysis to learn the company's public outlook.

Product/Service Analysis: Brands/Organizations can perform sentiment analysis on customer reviews to see how well a product or service is doing in the market and make future decisions accordingly.

Stock Price Prediction: Predicting whether the stocks of a company will go up or down is crucial for investors. One can determine the same by performing sentiment analysis on News Headlines of articles containing the company's name. If the news headlines pertaining to a particular organization happen to have a positive sentiment—its stock prices should go up and vice-versa.

import nltk nltk.Freq_dist(all_words)

Sentiment Analysis

- **Sentiment Analysis Steps**

Text Preprocessing: The text data is cleaned by removing irrelevant information, such as special characters, punctuation, and stopwords.

Tokenisation: The text is divided into individual words or tokens to facilitate analysis. Feature Extraction: Relevant features are extracted from the text, such as words or parts of speech.

Labelling the text: In this step, we determine whether a text is positive, negative or neutral

Sentiment Classification: Machine learning algorithms are used to classify the labelled text data.

Evaluation: The performance of the sentiment analysis model is assessed using evaluation metrics, such as accuracy, precision, recall, or F1 score.

import nltk nltk.Freq_dist(all_words)

Sentiment Analysis

- **How to label text data – TextBlob**

TextBlob is a Python library for Natural Language Processing. Using Text Blob for sentiment analysis is quite simple. It takes text as an input and can return polarity and subjectivity as outputs.

Polarity determines the sentiment of the text. Its values lie in [-1,1] where -1 denotes a highly negative sentiment and 1 denotes a highly positive sentiment.

Subjectivity determines whether a text input is a factual information or a personal opinion. Its value lies between [0,1] where a value closer to 0 denotes a piece of factual information and a value closer to 1 denotes a personal opinion.

import nltk nltk.Freq_dist(all_words)

Sentiment Analysis

- **How to label text data – text blob**

Example:

```
from textblob import TextBlob

text_1 = "The movie was so awesome."
text_2 = "The food here tastes terrible."

#Determining the Polarity
p_1 = TextBlob(text_1).sentiment.polarity
p_2 = TextBlob(text_2).sentiment.polarity

#Determining the Subjectivity
s_1 = TextBlob(text_1).sentiment.subjectivity
s_2 = TextBlob(text_2).sentiment.subjectivity

print("Polarity of Text 1 is", p_1)
print("Polarity of Text 2 is", p_2)
print("Subjectivity of Text 1 is", s_1)
print("Subjectivity of Text 2 is", s_2)
```

import nltk nltk.Freq_dist(all_words)

Sentiment Analysis

- **How to label text data – Vader**

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a rule-based sentiment analyzer that has been trained on social media text.

Its usage in Python is like TextBlob as it generates a polarity score for each text

import nltk nltk.Freq_dist(all_words)

Sentiment Analysis

- **How to label text data – Vader**

Example

```
# pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
sentiment = SentimentIntensityAnalyzer()
text_1 = "The book was a perfect balance between wrtiting style and plot."
text_2 =  "The pizza tastes terrible."
sent_1 = sentiment.polarity_scores(text_1)
sent_2 = sentiment.polarity_scores(text_2)
print("Sentiment of text 1:", sent_1)
print("Sentiment of text 2:", sent_2)
```