Lecture 4-1: Text Preprocessing

Introduction

In any machine learning task, cleaning and pre-processing of the data is a very important step. The better we can represent our data, the better the model training and prediction can be expected.

Specially in the domain of Natural Language Processing (NLP) the data is unstructured. It become crucial to clean and properly format it based on the task at hand. There are various pre-processing steps that can be performed but not necessary to perform all. These steps should be applied based on the problem statement.

Example: Sentiment analysis on twitter data can required to remove hashtags, emoticons, etc. but this may not be the case if we are doing the same analysis on customer feedback data.

Common text preprocessing / cleaning steps

Lower casing	Removal of emoticons
Removal of Punctuations	Conversion of emoticons to words
Removal of Stopwords	Conversion of emojis to words
Removal of Frequent words	Removal of URLs
Removal of Rare words	Removal of HTML tags
Stemming	Chat words conversion
Lemmatization	Spelling correction
Removal of emojis	

Import libraries and load the data

```
import numpy as np
import pandas as pd
import re
import nltk
import string
import demoji
import contractions
import unidecode
from nltk.tokenize import word_tokenize
from collections import Counter
from num2words import num2words
from nltk.corpus import twitter_samples
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from bs4 import BeautifulSoup

pd.options.display.max_columns=None
pd.options.display.max_rows=None
pd.options.display.max_colwidth=None

nltk.download('twitter_samples') # Download the dataset

# We are going to use the Negative and Positive Tweets file which each contains 5000 tweets.
for name in twitter_samples.fileids():
    print(f' - {name}')
```

```python
# Load the negative tweets file and assign label as 0 for negative
negative_tweets = twitter_samples.strings("negative_tweets.json")
df_neg = pd.DataFrame(negative_tweets, columns=['text'])
df_neg['label'] = 0

# Load the positive tweets file and assign label as 1 for positive
positive_tweets = twitter_samples.strings("positive_tweets.json")
df_pos = pd.DataFrame(positive_tweets, columns=['text'])
df_pos['label'] = 1

df = pd.concat([df_pos, df_neg]) # Concatenate both the files
# Shuffle the data to mix negative and positive
df = df.sample(frac=1).reset_index(drop=True) tweets

print(f'Shape of the whole data is: {df.shape[0]} rows and {df.shape[1]} columns')
```

Shape of the whole data is: 10000 rows and 2 columns

Look at the head of the dataframe

index	text	label
0	@AlyssaC_HK I use Chrome :)	1
1	This manga is just too cute and yet made me cry.. :(http://t.co/KB6GswBxMT	0
2	@joohyunvrl you are a goddess :D	1
3	& why tf can't i find subtitles for OITNB ? I need to understand Changs backstory :(	0
4	But you know I do na... :(We can negotiate the Bride Price Advance Payment https://t.co/2uRT9ae8Px	0
5	@Jaaeeeeee lucky :(I was feeling hungry, but then I didn't wanna get out of bed either, so I gotta wait until mañana	0
6	@DiongzonS follow @jnlazts & http://t.co/RCvcYYO0lq follow u back :)	1
7	i miss watching anna akana videos :(	0
8	@SupportOrganize Hey Lynne, on FB the Buffer button will bring up your composer and you can Buffer an FB share. :) -Mary	1
9	@nicoleezzy halaaaang :(just done crying.	0
10	Why am I not tierd :(	0

Lower Casing

Lowercasing is a common text preprocessing technique. It helps to transform all the text in same case.

Examples 'The', 'the', 'ThE' -> 'the'

This is also useful to find all the duplicates since words in different cases are treated as separate words and becomes difficult for us to remove redundant words in all different case combination.

This may not be helpful when we do tasks like Part of Speech tagging (where proper casing gives some information about Nouns and so on) and Sentiment Analysis (where upper casing refers to anger and so on)

```
df.text = df.text.str.lower()
df.head(2)
```

Remove URL's

URL stands for Uniform Resource Locator. If present in a text, it represents the location of another website.

If we are performing any websites backlink analysis, in that case URL's are useful to keep. Otherwise, they don't provide any information. So we can remove them from our text.

```
df.text = df.text.str.replace(r'https?://\S+|www\.\S+', '', regex=True)
df.head()
```

Remove E-mail

E-mail id's are common in customer feedback data and they do not provide any useful information. So we remove them from the text.

Twitter data that we are using does not contain any email id's. Hence, please find the code snipper with an dummy example to remove e-mail id's.

```
text = 'I have being trying to contact xyz via email to xyz@abc.co.in but there is no response.'
re.sub(r'\S+@\S+', '', text)
```

> 'I have being trying to contact xyz via email to but there is no response.'

Reomove Date

Dates can be represented in various formats and can be difficult at times to remove them. They are unlikely to contain any useful information for predicting the labels.

Below I have used dummy text to showcase the following task.

```
text = "Today is 22/12/2020 and after two days on 24-12-2020 our vacation starts until 25th.09.2021"

# 1. Remove date formats like: dd/mm/yy(yy), dd-mm-yy(yy), dd(st|nd|rd).mm/yy(yy)
re.sub(r'\d{1,2}(st|nd|rd|th)?[-./]\d{1,2}[-./]\d{2,4}', '', text)
```

> 'Today is and after two days on our vacation starts until '

```
text = "Today is 11th of January, 2021 when I am writing this post. I hope to post this by
February 15th or max to max by 20 may 21 or 20th-December-21"

# 2. Remove date formats like: 20 apr 21, April 15th, 11th of April, 2021
pattern = re.compile(r'(\d{1,2})?(st|nd|rd|th)?[-./,]?\s?(of)?\s?([J|j]an(uary)?|[F|f]eb(ruary)?|
[Mm]ar(ch)?|[Aa]pr(il)?|[Mm]ay|[Jj]un(e)?|[Jj]ul(y)?|[Aa]ug(ust)?|[Ss]ep(tember)?|[Oo]ct(ober)?|
[Nn]ov(ember)?|[Dd]ec(ember)?)\s?(\d{1,2})?(st|nd|rd|th)?\s?[-./,]?\s?(\d{2,4})?')
pattern.sub(r'', text)
```

> 'Today is when I am writing this post. I hope to post this byor max to max by or '

HTML Tags

If we are extracting data from various websites, it is possible that the data also contains HTML tags. These tags does not provide any information and should be removed. These tags can be removed using regex or by using BeautifulSoup library.

```
<title>Below is a dummy html code.</title>
<body>
  <p>All the html opening and closing brackets should be remove.</p>
  <a href="https://www.abc.com">Company Site</a>
</body>
```

```
# Using regex to remove html tags
pattern = re.compile('<.*?>')
pattern.sub('', text)
```

> '\nBelow is a dummy html code.\n\n All the html opening and closing brackets should be remove.\n Company Site\n\n'

Using Beautiful Soup

```
def remove_html(text):
    clean_text = BeautifulSoup(text).get_text()
    return clean_text

remove_html(text)
```

> 'Below is a dummy html code.\n\nAll the html opening and closing brackets should be remove.\nCompany Site\n\n'

Emojis

As more and more people have started using social media emoji's play a very crucial role. Emoji's are used to express emotions that are universally understood.

In some analysis such as sentiment analysis emoji's can be useful. We can convert them to words or create some new features based on them. For some analysis we need to remove them. Find the below code snippet used to remove the emoji's.

```python
# Reference: https://gist.github.com/slowkow/7a7f61f495e3dbb7e3d767f97bd7304b

def remove_emoji(text):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002500-\U00002BEF"  # chinese char
                           u"\U00002702-\U000027B0"
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           u"\U0001f926-\U0001f937"
                           u"\U00010000-\U0010ffff"
                           u"\u2640-\u2642"
                           u"\u2600-\u2B55"
                           u"\u200d"
                           u"\u23cf"
                           u"\u23e9"
                           u"\u231a"
                           u"\ufe0f"  # dingbats
                           u"\u3030"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

```python
text = "game is on 🔥🔥. Hilarious😂"
remove_emoji(text)
```

> 'game is on . Hilarious'

```python
# Remove emoji's from text
df.text = df.text.apply(lambda x: remove_emoji(x))
```

Emoticons

Emoji's and Emoticons are different. Yes!!

Emoticons are used to express facial expressions using keyboard characters such as letters, numbers, and pucntuation marks. Where emjoi's are small images.

Thanks to Neel Shah for curating a dictionary of emoticons and their description. We shall use this dictionary and remove the emoticons from our text.

```
EMOTICONS = {
    u":-\)":"Happy face or smiley",
    u":\)":"Happy face or smiley",
    u":-\]":"Happy face or smiley",
    u":\]":"Happy face or smiley",
    u":-3":"Happy face smiley",
    u":3":"Happy face smiley",
    u":->":"Happy face smiley",
    u":>":"Happy face smiley",
    u"8-\)":"Happy face smiley",
    ...
    u"\(\^\)o\(\^\)":"Happy",
    u"\(\^O\^\)":"Happy",
    u"\(\^o\^\)":"Happy",
    u"\)\^o\^\(":"Happy",
    u":O o_O":"Surprised",
    u"o_0":"Surprised",
    u"o\.O":"Surpised",
    u"\(o\.o\)":"Surprised",
    u"oO":"Surprised",
    u"\(\*￣m￣\)":"Dissatisfied",
    u"\(‘A`\)":"Snubbed or Deflated"
}
```

```python
def remove_emoticons(text):
    emoticons_pattern = re.compile(u'(' + u'|'.join(emo for emo in EMOTICONS) + u')')
    return emoticons_pattern.sub(r'', text)

remove_emoticons("Hello :->")
```

> 'Hello '

```python
# Remove emoticons from text
df.text = df.text.apply(lambda x: remove_emoticons(x))
```

Hashtags and Mentions

We are habituated to use hashtags and mentions in our tweet either to indicate the context or bring attention to an individual. Hashtags can be used to extract features, to see what's trending and in various other applications.

Since, we don't require them we'll remove them.

```
def remove_tags_mentions(text):
    pattern = re.compile(r'(@\S+|#\S+)')
    return pattern.sub('', text)

text = "live @flippinginja on #younow - jonah and jaredddddd"
remove_tags_mentions(text)
```

> 'live on - jonah and jaredddddd'

```
# Remove hashtags and mentions
df.text = df.text.apply(lambda x: remove_tags_mentions(x))
```

Punctuations

Punctuations are character other than alphaters and digits. These include `[!"#$%&\'()*+,-./:;<=>?@\\^_`{|}~]`

It is better remove or convert emoticons before removing the punctuations, since if we do the other we around we might loose the emoticons from the text. Another example, if the text contains $10.50 then we'll remove the .(dot) and the value will loose it's meaning.

```
PUNCTUATIONS = string.punctuation

def remove_punctuation(text):
    return text.translate(str.maketrans('', '', PUNCTUATIONS))

df.text = df["text"].apply(lambda text: remove_punctuation(text))
df.text[:2].tolist()
```

> [' i use chrome ',
>
> > 'this manga is just too cute and yet made me cry ']

Stopwords

Stopwords are commonly occuring words in any language. Such as, in english these words are 'the', 'a', 'an', & many more. They are in most cases not useful and should be removed.

There are certain tasks in which these words are useful such as Part-of-Speech(POS) tagging, language translation. Stopwords are compiled for many languages, for english language we can use the list from the nltk package.

```
nltk.download('stopwords')
STOPWORDS = set(stopwords.words('english'))

def remove_stopwords(text):
    return ' '.join([word for word in text.split() if word not in STOPWORDS])

# Remove stopwords
df.text = df.text.apply(lambda text: remove_stopwords(text))
df.text[:3].tolist()
```

> ['use chrome', 'manga cute yet made cry', 'goddess']

Numbers

We may remove numbers if they are not useful in our analysis. But analysis in the financial domain, numbers are very useful.

```
df.text = df.text.str.replace(r'\d+', '', regex=True)
```

Extra whitespaces

After usually after preprocessing the text there might be extra whitespaces that might be created after transforming, removing various characters. Also, there is a need to remove all the new line, tab characters as well from our text.

```
def remove_whitespaces(text):
    return " ".join(text.split())

text = "  Whitespaces in the beginning are removed  \t as well \n  as in between  the text    "

clean_text = " ".join(text.split())
clean_text
```

> 'Whitespaces in the beginning are removed as well as in between the text'

```
df.text = df.text.apply(lambda x: remove_whitespaces(x))
```

Frequent words

we can remove the common words which don't provide us with much information.

```
def freq_words(text):
    tokens = word_tokenize(text)
    FrequentWords = []
    for word in tokens:
        counter[word] += 1
    for (word, word_count) in counter.most_common(10):
        FrequentWords.append(word)
    return FrequentWords

def remove_fw(text, FrequentWords):
    tokens = word_tokenize(text)
    without_fw = []
    for word in tokens:
        if word not in FrequentWords:
            without_fw.append(word)
    without_fw = ' '.join(without_fw)
    return without_fw

counter = Counter()
```

```
text = """
Natural Language Processing is the technology used to aid computers to understand the human’s natural language. It’s not an easy task teaching machines to understand how we communicate. Leand Romaf, an experienced software engineer who is passionate at teaching people how artificial intelligence systems work, says that “in recent years, there have been significant breakthroughs in empowering computers to understand language just as we do.” This article will give a simple introduction to Natural Language Processing and how it can be achieved. Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. Most NLP techniques rely on machine learning to derive meaning from human languages.
"""
```

```
FrequentWords = freq_words(text)
print(FrequentWords)
```

> ',', 'to', '.', 'is', 'the', 'understand', 'Natural', 'Language', 'Processing', 'computers']

```
fw_result = remove_fw(text, FrequentWords)
fw_result
```

> 'technology used aid human ’ s natural language It ’ s not an easy task teaching machines how we communicate Leand Romaf an experienced software engineer who passionate at teaching people how artificial intelligence systems work says that “ in recent years there have been significant breakthroughs in empowering language just as we do. ” This article will give a simple introduction and how it can be achieved usually shortened as NLP a branch of artificial intelligence that deals with interaction between and humans using natural language The ultimate objective of NLP read decipher and make sense of human languages in a manner that valuable Most NLP techniques rely on machine learning derive meaning from human languages'

Rare words

Rare words are similar to frequent words. We can remove them because they are so less that they cannot add any value to the purpose.

```
def rare_words(text):
    tokens = word_tokenize(text)     # tokenization
    for word in tokens:
        counter[word]= +1
    RareWords = []
    number_rare_words = 10
    frequentWords = counter.most_common()
    for (word, word_count) in frequentWords[:-number_rare_words:-1]: # take top 10 frequent words
        RareWords.append(word)
    return RareWords

def remove_rw(text, RareWords):
    tokens = word_tokenize(text)
    without_rw = []
    for word in tokens:
        if word not in RareWords:
            without_rw.append(word)
    without_rw = ' '.join(without_rw)
    return without_rw

counter = Counter()
```

```
text = """
Natural Language Processing is the technology used to aid computers to understand the human’s natural language. It’s not an easy task teaching machines to understand how we communicate. Leand Romaf, an experienced software engineer who is passionate at teaching people how artificial intelligence systems work, says that “in recent years, there have been significant breakthroughs in empowering computers to understand language just as we do.” This article will give a simple introduction to Natural Language Processing and how it can be achieved. Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. Most NLP techniques rely on machine learning to derive meaning from human languages.
"""
```

```
RareWords = rare_words(text)
print(RareWords)
```

> ['from', 'meaning', 'derive', 'learning', 'machine', 'on', 'rely', 'techniques', 'Most']

```
rw_result = remove_fw(text, RareWords)
rw_result
```

Conversion of Emoji to Words

To remove or not is done based on the purpose of the application. Example if we are building a sentiment analysis system emoji's can be useful.

"The movie was 🔥"
or
"The movie was 💩"

If we remove the emoji's the meaning of the sentence changes completely. In these cases we can convert emoji's to words.

demoji requires an initial data download from the Unicode Consortium's emoji code repository.

On first use of the package, call download_codes().

This will store the Unicode hex-notated symbols at ~/.demoji/codes.json for future use.

Read more about demoji on pypi.org

```
demoji.download_codes()
```

```python
def emoji_to_words(text):
    return demoji.replace_with_desc(text, sep="__")

text = "game is on 🔥 🚣🏼"
emoji_to_words(text)
```

```
game is on __fire__ __person rowing boat: medium-light skin tone__'
```

Conversion of Emoticons to Words

As we did for emoji's, we convert emoticons to words for the same purpose.

```
def emoticons_to_words(text):
    for emot in EMOTICONS:
        text = re.sub(u'('+emot+')', "_".join(EMOTICONS[emot].replace(",","").replace(":","").split()), text)
    return text

text = "Hey there!! :-)"
emoticons_to_words(text)
```

> 'Hey there!! Happy_face_smiley'

Converting Numbers to Words

If our analysis require us to use information based on the numbers in the text, we can convert them to words.

Read more about num2words on github

```python
def nums_to_words(text):
    new_text = []
    for word in text.split():
        if word.isdigit():
            new_text.append(num2words(word))
        else:
            new_text.append(word)
    return " ".join(new_text)

text = "I ran this track 30 times"
nums_to_words(text)
```

> 'I ran this track thirty times'

Chat words Conversion

The more we use social media, we have become lazy to type the whole phrase or word. Due to which slang words came into existance such as "omg" which represents "Oh my god". Such slang words don't provide much information and if we need to use them we have to convert them.

Thank you: GitHub repo for the list of slang words

```
chat_words = """
AFAIK=As Far As I Know
AFK=Away From Keyboard
ASAP=As Soon As Possible
ATK=At The Keyboard
...
WUF=Where Are You From?
W8=Wait...
7K=Sick:-D Laugher
OMG=Oh my god"""
```

```python
chat_words_dict = dict()
chat_words_set = set()

def cw_conversion(text):
    new_text = []
    for word in text.split():
        if word.upper() in chat_words_set:
            new_text.append(chat_words_dict[word.upper()])
        else:
            new_text.append(word)
    return " ".join(new_text)

for line in chat_words.split('\n'):
    if line != '':
        cw, cw_expanded = line.split('=')[0], line.split('=')[1]

        chat_words_set.add(cw)
        chat_words_dict[cw] = cw_expanded

text = "omg that's awesome."
cw_conversion(text)
```

> "Oh my god that's awesome."

Expanding Contractions

Contractions are words or combinations of words created by dropping a few letters and replacing those letters by an apostrophe.

Example:

- don't: do not
- we'll: we will

Our nlp model don't understand these contractions i.e. they don't understand that "don't" and "do not" are the same thing. If our problem statement requires them then we can expand them or else leave it as it is.

```python
def expand_contractions(text):
    expanded_text = []
    for line in text:
        expanded_text.append(contractions.fix(line))
    return expanded_text

text = ["I'll be there within 15 minutes.", "It's awesome to meet your new friends."]
expand_contractions(text)
```

> ['I will be there within 15 minutes.',
>
> > 'It is awesome to meet your new friends.']

In stemming we reduce the word to it's base or root form by removing the suffix characters from the word. It is one of the technique to normalize text.

Stemming for root word "like" include:

- "likes"
- "liked"
- "likely"
- "liking"

Stemmed word doesn't always match the words in our dictionary such as:

- console -> consol
- company -> compani
- welcome -> welcom

Due to which stemming is not performed in all nlp tasks.

There are various algorithms used for stemming but the most widely used is

```
stemmer = PorterStemmer()

def stem_words(text):
    return ' '.join([stemmer.stem(word) for word in text.split()])


df['text_stemmed'] = df.text.apply(lambda text: stem_words(text))
df[['text', 'text_stemmed']].head()
```

text	text_stemmed
use chrome	use chrome
manga cute yet made cry	manga cute yet made cri
goddess	goddess
amp tf cant find subtitles oitnb need understand changs backstory	amp tf cant find subtitl oitnb need understand chang backstori
know na negotiate bride price advance payment	know na negoti bride price advanc payment

PorterStemmer can be used only for english. If we are working with other than english then we can use SnowballStemmer.

```
SnowballStemmer.languages
```

```
('arabic',
 'danish',
 'dutch',
 'english',
 'finnish',
 'french',
 'german',
 'hungarian',
 'italian',
 'norwegian',
 'porter',
 'portuguese',
 'romanian',
 'russian',
 'spanish',
 'swedish')
```

Lemmatization

Lemmatization tried to perform the similar task as that of stemming i.e. trying to reduce the inflection words to it's base form. But lemmatization does it by using a different approach.

Lemmatizations takes into consideration of the morphological analysis of the word. It tries to reduce to words to it's dictionary form which is known as lemma.

```python
nltk.download('wordnet')
nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()

def text_lemmatize(text):
    return ' '.join([lemmatizer.lemmatize(word) for word in text.split()])

df['text_lemmatized'] = df.text.apply(lambda text: text_lemmatize(text))
df[['text', 'text_stemmed', 'text_lemmatized']].head()
```

text	text_stemmed	text_lemmatized
use chrome	use chrome	use chrome
manga cute yet made cry	manga cute yet made cri	manga cute yet made cry
goddess	goddess	goddess
amp tf cant find subtitles oitnb need understand changs backstory	amp tf cant find subtitl oitnb need understand chang backstori	amp tf cant find subtitle oitnb need understand chang backstory
know na negotiate bride price advance payment	know na negoti bride price advanc payment	know na negotiate bride price advance payment

Difference between Stemming and Lemmatization:

Stemming	Lemmatization
Fast compared to lemmatization	Slow compared to stemming
Reduces the word to it's base form by removing the suffix	Uses lexical knowledge to get the base form of the word
Does not always provide meaning or dictionary form of the original word	Resulting words are always meaningful and dictionary words

Spelling Correction

We as human always make mistake. Normally incorrect spelling in text are know as typos.

Since the NLP model doesn't know the difference between a correct and an incorrect word. For the model "thanks" and "thnks" are two different words. Therefore, spelling correction is an important step to bring the incorrect words in the correct format.

```
spell = SpellChecker()

def correct_spelling(text):
    correct_text = []
    misspelled_words = spell.unknown(text.split())
    for word in text.split():
        if word in misspelled_words:
            correct_text.append(spell.correction(word))
        else:
            correct_text.append(word)
    return " ".join(correct_text)
```

```
text = "Hi, hwo are you doin? I'm good thnks for asking"
correct_spelling(text)
```

> "Hi, how are you doing I'm good thanks for asking"

```
text = "hw are you doin? I'm god thnks"
correct_spelling(text)
```

> "he are you doing I'm god thanks"

Convert accented characters to ASCII characters

Accent marks (also referred to as diacritics or diacriticals) usually appear above a character when we press the character for a long time. These need to be remove cause the model cannot distinguish between "dèèp" and "deep". It will consider them as two different words.

```
def accented_to_ascii(text):
    return unidecode.unidecode(text)

text = "This is an example text with accented characters like dèèp lèarning ánd cömputer vísíön etc."
accented_to_ascii(text)
```

> 'This is an example text with accented characters like deep learning and computer vision etc.'