Summer of Science Mid-Term Report

Natural Language Processing

Author: Anoushka Dey

Mentor: Sneha Gaikwad

June 2023

1 Introduction to Natural Language Processing

Natural Language Processing (NLP) is a field derived from computer science and artificial intelligence that deals with the interaction between computers and human languages. The main goal of NLP is to enable computers to understand and interpret languages, in a way similar to humans. It involves understanding and generating human language and speech and is particularly concerned with how computers process and analyse large amounts of natural language data.

It has a wide range of applications like sentiment analysis, machine translation, text summarization, chatbots, email classification and filtering and so on.NLP involves the use of machine learning, deep learning and rule-based systems. Popular tools and libraries such as NLTK (Natural Language Toolkit), spaCy and Gensim are used in NLP.

It is a rapidly growing field and NLP techniques are used in a wide range of applications:

- **Text Summarization:** NLP techniques are used to summarize longer text documents. This is useful for document indexing and news indexing.
- **Sentiment Analysis:** NLP techniques are employed for determining the sentiment or emotion expressed in text. This is useful for customer feedback analysis and social media monitoring.
- **Speech Recognition and Transcription:** Speech is converted to text, which is useful for tasks such as dictation and voice controlled assistants.

2 NLP Pipeline

The NLP pipeline is a set of steps to be followed for an end-to-end implementation of NLP based software. Following are the steps:

- Data Acquisition
- Text Preprocessing
- Feature Engineering
- Modelling
- Model Evaluation
- Deployment

2.1 Data Acquisition

This step involves collating the data for the NLP software deployment. The data might already exist in a local machine or a database or it might have to be scraped from the web. In the former case, we are good to go. All we need to do is format the data in the manner in which we want to have it and then move over to the next step. In the latter, however, we have to first gather the data using APIs and other techniques. There might also be a problem of less data. In such a case, we use *data augmentation*.
Data augmentation involves making fake data from already pre-existing data using synonyms, bigram flip, back translation or noise addition. There may also be a case wherein there might be no data available at all. In that case, we have to manually collect the data ourselves and label it accordingly.
After the data has been acquired, we move on to *text preprocessing*.

2.2 Text Preprocessing

Text preprocessing is essential to NLP. It involves removing redundant data or noise by converting the text to lowercase and then removing all the stop words, punctuation marks and typographical errors.

The various text preprocessing steps are:

2.2.1 Tokenization

The sentence is split into a number of words to make the text preprocessing procedure less cumbersome.

```
from nltk.tokenize import word_tokenize
sentence= 'This is the tokenization subsection'
words=word_tokenize(sentence)
print(words)
```

Output: ['This', 'is', 'the', 'tokenization', 'subsection']

2.2.2 Lowercasing

Words like *Case* and *case* mean the same but appear to be different due to the different case structures of the words. Hence, in order to preserve the meaning of the words, they are converted to lowercase in order to render words like *Case* and *case* to be the same.

```
sentence='This is the Lowercase subsection.'
sentence=sentence.lower()
print(sentence)
```

Output: this is the lowercase subsection.

2.2.3 Stop Words Removal

Stop words such as 'a', 'an', 'the' and so on are very common in text documents. These words do not add any concrete meaning to the documents and do not help in distinguishing between two or more such documents. Hence, they can be removed as they are not significant.

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
sentence='Natural language processing is great!'
stop_words=set(stopwords.words('english')
word_tokens=word_tokenize(sentence)
filtered_sentence=[w for w in word_tokens if not w in stop_words]
print(filtered_sentence)
```

Output: ['Natural', 'language', 'processing', 'great', '!']
Here the stop word '*is*' has been removed.

2.2.4 Stemming

Stemming is a natural language processing technique wherein the word is reduced to its base or root form. This is done to normalize the text and is commonly used in retrieval and text mining applications. This technique reduces the number of words that serve as an input to the machine learning or deep learning model by reducing the number of variants in which a particular word exists.

```
import nltk
from nltk.stem import PorterStemmer
ps=PortetStemmer()
sentence="I am learning about stemming"
for word in sentence.split():
print(ps.stem(word))
```

Output: i am learn about stem

2.2.5 Lemmatization

This is similar to stemming. However, lemmatization reduces words to an already existing word in the language. This requires deep linguistic knowledge in constructing dictionaries to look up the lemma of the word. Stemming most commonly collapses derivationally related words whereas lemmatization collapses the

different inflectional forms of a lemma.

```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer=WordNetLemmatizer()
print(lemmatizer.lemmatize("Machine", pos='n')
print(lemmatizer.lemmatize("caring",pos='v'))
```

Output: machine, care
Machine is transformed to lowercase but it still retains its original form. Caring on the other hand, gets transformed to care as the pos variable specified is 'v' or verb.

2.3 Feature Engineering

Feature engineering involves converting text data to numerical data. There is need to do so because a machine learning model does not understand text data. There are multiple techniques to implement feature engineering.

2.3.1 One Hot Encoding

In one hot encoding, every word which part of the give text data is written in the form of a vector, constituting of only 0 and 1. Every word is written in the form of a one hot vector with each vector being unique. This enables the word to be uniquely identified by its vector and vice versa.
Example: The cat caught the rat

```
The --> [0 1 0 0 0 0]
cat --> [0 0 1 0 0 0]
caught --> [0 0 0 1 0 0]
the --> [0 0 0 0 1 0]
rat --> [0 0 0 0 0 1]
```

However, sparsity is one of the disadvantages of one hot encoding. Only a single sentence creates a single vector of size x*y where x is the length of the sentence and y is the number of unique words in the document. Even with relatively small eight dimensions, the text might require exponentially large memory space.
It is also difficult to extract meanings. Each word is embedded in isolation, and every word contains a single 1 and n 0s where n is the number of dimensions. Different and unrelated words present in the vocabulary, in this case, can be represented as equally spaced out vectors. The semantic meaning of the words are not retained. Hence, there is not much of a difference between a 'hello' and an 'apple' in this case!

2.3.2 Bag of Words

A bag of words is a text representation that is used to describe the occurrence of words within a document. Grammatical details are discarded and the order of words is also not taken into account. Only the word count is kept track of. It is called a 'bag of words' because any information about the order or structure of words in the document is discarded. This technique only checks whether known words occur in the document and not 'where'.

Without Preprocessing

Sentence 1: This is a report on Natural Language Processing

Sentence 2: It enables computers to understand the human language.

Sentence 1: [1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]

Word	**Encoding**
This	1
is	1
a	1
report	1
on	1
Natural	1
Language	1
Processing	1
It	0
enables	0
computers	0
to	0
understand	0
the	0
human	0
language	0

Table 1: Sentence 1

Sentence 2: [0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1]

With Preprocessing

These are two sentence after text preprocessing:

Sentence 1: welcome natural language processing start text processing

Sentence 2: learning good practice

Sentence 1: [1 1 1 2 1 1 0 0 0]

Word	Encoding
This	0
is	0
a	0
report	0
on	0
Natural	0
Language	0
Processing	0
It	1
enables	1
computers	1
to	1
understand	1
the	1
human	1
language	1

Table 2: Sentence 2

Sentence 2: [0 0 0 0 0 0 1 1 1]

There are two 'processing' words in the first sentence. Suppose one of them was 'Processing' and the other was 'processing'. Without text preprocessing, the bag of words technique would have differentiated between 'processing' and 'Processing' and would have counted them separately. However, after text preprocessing, both the words are rendered to be the same and hence the bag of words technique assigns a '2' to 'processing' for sentence 1.

There are some disadvantages of this technique as well. The size of the vector representing the sentence increases if new words are added to the vocabulary. The vectors may also contain many 0s, resulting in a sparse matrix. Also, neither is the grammar of the sentence taken into consideration while the vectors are being constructed nor is the ordering of the words in the text taken into account.

2.3.3 N-Gram

N-grams are contiguous sequences of words that are collected from a speech corpus or a text document or any other type of data. The 'n' specifies the number of words or items to club together. For example, n=1 for a unigram and n=2 for a bigram and so on.

N-grams are technically ordered groups of n tokens. N-gram models are based on statistical frequency of

Word	Encoding
welcome	1
natural	1
language	1
processing	2
start	1
text	1
learning	0
good	0
practice	0

Table 3: Sentence 1

Word	Encoding
welcome	0
natural	0
language	0
processing	0
start	0
text	0
learning	1
good	1
practice	1

Table 4: Sentence 2

the groups of tokens. The central concept of this theory is that the next word after a particular set of tokens is dependent on the previous n words. This conditional probability is obtained by *maximum likelihood estimation* on a set of training sequences.

Chain rule in general:

$\mathbb{P}(\omega_n, \omega_{n-1}, ..., \omega_1) = \mathbb{P}(\omega_1).\mathbb{P}(\omega_2|\omega_1).\mathbb{P}(\omega_3|\omega_2, \omega_1)...\mathbb{P}(\omega_n|\omega_{n-1}...\omega_1)$

Chain rule applied to compute the joint probability of words in a sentence:

$\mathbb{P}(\omega_1, ...\omega_n) = \prod_{i=1}^{n} \mathbb{P}(\omega_i|\omega_1...\omega_{i-1})$

Using the Markov assumption:

$\mathbb{P}(\omega_1, ...\omega_n) = \prod_i \mathbb{P}(\omega_i|\omega_{i-k}...\omega_{i-1})$, where $k = n - 1$

This can now be extended to unigrams (n=1), bigrams (n=2), trigrams (n=3) and so on.

One of the main challenges of n-grams is data sparsity, which means that some n-grams may not occur

frequently or at all in the training data, resulting in low or zero probabilities. Data sparsity can also affect the efficiency of n-grams, as storing and processing large n-gram models can be costly and time-consuming. One way to overcome data sparsity is to use smoothing techniques, which assign non-zero probabilities to unseen or rare n-grams by redistributing the probabilities of seen n-grams.

Another challenge of n-grams is context sensitivity, which means that the meaning and interpretation of an n-gram can depend on the surrounding words or the situation in which it is used. Context sensitivity can also affect the relevance and ranking of search results, as different users may have different intents and preferences when searching for the same n-gram. To deal with context sensitivity, n-gram models can use techniques such as word sense disambiguation, semantic analysis, and personalisation, to infer the meaning and intent of an n-gram based on the context and the user.

2.3.4 Tf-Idf

Tf-Idf or Term Frequency - Inverse Document Frequency is a natural language processing technique which measures how important a term is within a document, that is, relative to the corpus. The words within the document are transformed into number depicting their importance by a text vectorization process.

The Tf-Idf technique vectorizes or scores a word by multiplying the word's Term Frequency (TF) with the Inverse Document Frequency (IDF).

Term Frequency: It is the number of times a term appears in the given text document as compared to total words in the document.

Inverse Document Frequency: It depicts the proportion of documents in the corpus that contain the term under consideration. Words that are perceived to be more important are the ones that are unique to a small set of documents whereas words which are more frequently used across all documents are given a lower importance measure.

$$IDF = log\left(\frac{No.\ of\ documents\ in\ the\ corpus}{No.\ of\ documents\ in\ the\ corpus\ that\ contain\ the\ term}\right)$$

Now, Tf-Idf= TF*IDF. The resulting Tf-Idf value reflects the importance of a term for a document in the corpus. The importance of a ter, is high when it occurs a lot in the given document and rarely in others. Commonality within a document is measured by the Tf score and this balanced out by the rarity between documents as measured by the Idf score.

This technique is powerful but it can however, assign low values to words that are relatively important. There also arises the *zero value issue*. In this case, if a word appears across all the documents in the corpus, the Idf value would reduce to 0 and hence the Tf-Idf score would be 0, indicating that the word is of no consequence at all! This can be rectified by adding a 1 in the denominator of the Idf expression.

Another drawback is the extensive margin issue. this arises because Idf does not take into account how often a word appears in other documents (intensive margin) but just cares about whether it appears at all

(extensive margin). This causes it to be overly sensitive when there are changes on the extensive margin and overly resistant to change when there are changes in the intensive margin, even though the latter is arguably more important.

2.3.5 Word2Vec

The Word2Vec model is used for text representation in the vector space. This was founded by Tomas Mikolov and a group of research teams from Google in 2013. It is a neural network model that aims to explain word embedding based on the text corpus.

In this technique, to learn the embedding, the model looks at the nearby words. If a particular group of words is always found close to the same words, they will end up have the same embedding.

Before we can move on to label similar words, we must first fix the window size. This determines which nearby words we would want to pick. For example, a window size of 3 indicates that we shall pick up 3 words before and 3 words after the word under consideration.

Initially, a vector of random numbers ia assigned to each word in the corpus. We then iterate through each words of the document and take the vectors of the nearest n wrods on either side of the target word and concatenate all these vectors. These concatenated vectors are then forward propagated through a linear layer plus softmax function and we then try to predict what the target word was.

Next, we compute the error between the estimate and the actual target word and then backpropagate the error in order to modify the weights of the linear layer and also the vectors and embeddings of the neighbouring words. Finally, the weights are extracted from the hidden layer and by using these weights, we encode the meaning of the words in the vocabulary.

Word2Vec is composed of the following two preprocessing modules:**Continuous Bag of Words (CBOW)** and **Skip-Gram**

Without going much into the detail of both these modules, it is safe to say that as per observations, the skip gram model works best with a small amount of training datasets and can better represent rare words or phrases. However, the CBOW model is observed to train faster than the skip gram model. It can better represent more frequent words.

2.4 Modelling

In the modelling step, a model is made based on the data. Many approaches are used to build the model. Some of them are the *heuristic approach*,the *machine learning approach*, the *deep learning approach* and by using *cloud APIs*.

If there is very less data available, ML or DL approaches cannot be used. In that case, we have to use the heuristic approach. But if we have a good amount of data, we can use the machine learning approach and for even larger datasets, we can use the DL approach.

2.5 Model Evaluation

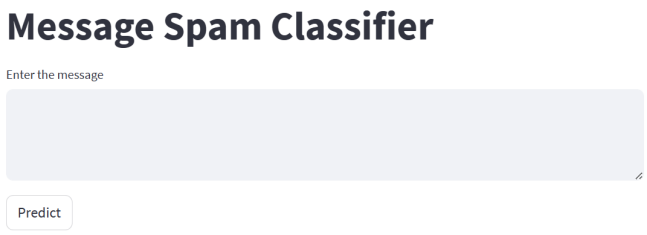
In the model evaluation, we can use two metrics Intrinsic evaluation and Extrinsic evaluation.
In intrinsic evaluation, we use multiple metrics to check our model such as Accuracy, Recall, Confusion Metrics, Perplexity, etc and extrinsic evaluation is done after deployment.

2.6 Deployment

In this step, the model is deployed on the cloud for users and they can then use this model. The model is first deployed on the cloud. This is followed by the monitoring phase in which we have to watch the model continuously and create a dashboard to show the evaluation metrics. Finally the model is re-trained based on the changed metrics and new data and is again deployed.

3 Project

I have succesfully completed the implementation of a message classifier using the nltk library as well as other machine learning modules offered by Scikit-learn.
The model evaluations have been documented as tables throughout the .ipynb notebook and I have also performed EDA on the sms-spam dataset downloaded from Kaggle.



You can find the project here: Github Repository
This is the link to the website: Message Classifier