A

PROJECT REPORT

(Group - 8)

On

BBC news articles sorting

By

Sanket Andalkar

Rushikesh Pawar

Santosh Waghmode

PROJECT GUIDE

Ms.Ankita Kanchan

ML - 12

Symbiosis Skills

and

Professional University, Kiwale

2021

Abstract :

Text mining has gained quite a significant importance during the past few years. Data, now-a-days is available to users through many sources like electronic media, digital media and many more. This data is usually available in the most unstructured form and there exists a lot of ways in which this data may be converted to structured form. In many real-life scenarios, it is highly desirable to classify the information in an appropriate set of categories. News contents are one of the most important factors that have influence on various sections. In this project we have considered the problem of classification of news articles. This project presents algorithms for category identification of news and have analysed the shortcomings of a number of algorithm approaches.

❑Introduction :

In today’s world, data is power. With News companies having terabytes of data stored in servers, everyone is in the quest to discover insights that add value to the organization. With various examples to quote in which analytics is being used to drive actions, one that stands out is news article classification.

Nowadays on the Internet there are a lot of sources that generate immense amounts of daily news. In addition, the demand for information by users has been growing continuously, so it is crucial that the news is classified to allow users to access theinformation of interest quickly and effectively. This way, the machine learning model for automated news classification could be used to identify topics of untracked news and/or make individual suggestions based on the user’s prior interests.

❑Objectives

To build a machine learning and deep learning model that should be able to classify BBC news articles according to various categories like business , sports, technology, education, politics.

To check which feature extraction method among Bag of Word and TFIDF work precisely.

❑Brief terms :

What is Natural Language Processing (NLP)? The method of communication with the help of which humans can speak, read, and write, is language. In other words, we humans can think, make plans, make decisions in our natural language. Here the big question is, in the era of artificial intelligence, machine learning and deep learning, can humans communicate in natural language with computers/machines? Developing NLP applications is a huge challenge for us because computers require structured data, but on the other hand, human speech is unstructured and often ambiguous in nature. Natural language is that subfield of computer science, more specifically of AI, which enables computers/machines to understand, process and manipulate human language. In simple words, NLP is a way of machines to analyze, understand and derive meaning from human natural languages like Hindi, English, French, Dutch, etc.

Natural Language Processing :

Natural Language Processing is defined as understanding text or speech with the aid of any software program or machine.

An analogy is that humans can interact and understand each other views and reply with the perfect answer. In NLP, a computer made interactions, not a human.

Applications of NLP

- ➢ Sentiment Analysis
- ➢ Chatbots
- ➢ Machine Translation
- ➢ Auto-Correct
- ➢ Speech Recognition
- ➢ Keyword-Search
- ➢ Virtual Assistants

NLTK Library :

Among the above-mentioned NLP tool, NLTK scores very high when it comes to the ease of use and explanation of the concept. The learning curve of Python is very fast and NLTK is written in Python so NLTK is also having very good learning kit. NLTK has incorporated most of the tasks like tokenization, stemming, Lemmatization, Punctuation, Character Count, and Word count. It is very elegant and easy to work with.

NLTK stands for Natural Language Tool Kit and is one of the most powerful NLP libraries.

Natural Language Tool Kit is the mother of all NLP libraries.

Machine Learning projects and data pre-processing use NLTK.

It is used in data pre-processing.

- Data Source : The data of BBC news articles is collected from the Kaggle website.

```
1 df.head()
```

	ArticleId	Text	Category
0	1833	worldcom ex-boss launches defence lawyers defe...	business
1	154	german business confidence slides german busin...	business
2	1101	bbc poll indicates economic gloom citizens in ...	business
3	1976	lifestyle governs mobile choice faster bett...	tech
4	917	enron bosses in $168m payout eighteen former e...	business

Data contains 2225 BBC news Articles and their categories.

Exploratory Data Analysis :

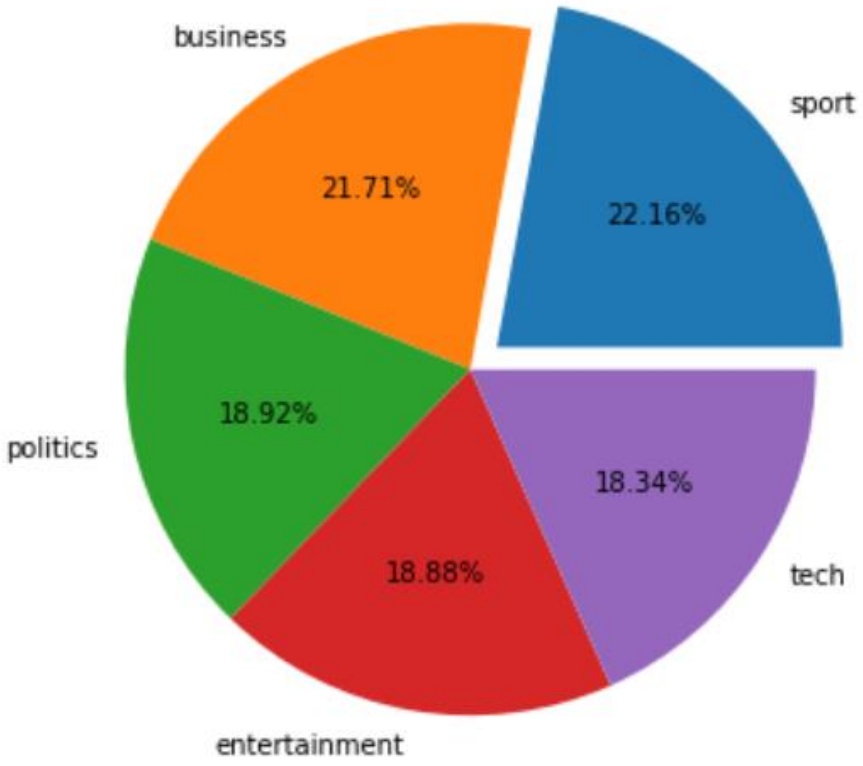
Checking null values

```
1 df.isna().sum()
```

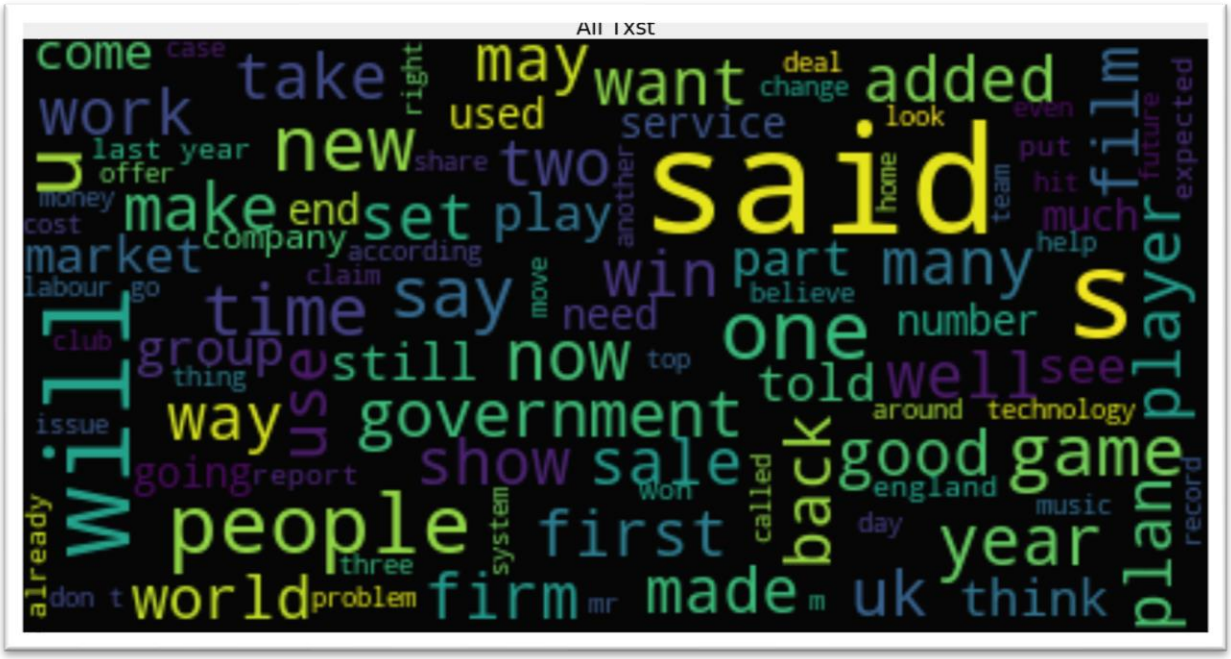
```
ArticleId    0
Text         0
Category     0
dtype: int64
```

Data do not contain any null values.

Pie chart of Categories in dataset



Word Cloud for Articles :



Text preprocessing in NLP :

In NLP, text preprocessing is the first step in the process of building a model.

The various text preprocessing steps are:

- ➢ Clean Text
- ➢ Lower casing
- ➢ Tokenization
- ➢ Stop words removal
- ➢ Stemming
- ➢ Lemmatization

Function to clean Text and lower case the text

```
def CleanText(text):
    text=re.sub('\d',' ',text) # Remove number
    text=re.sub(' +',' ',text) # Remove extra spaces
    text = "".join([char.lower() for char in text if char no
t in string.punctuation]) # Removing punctuation and n
ormalization
    return text
```

- Tokenization

What is Tokenizing?

It may be defined as the process of breaking up a piece of text into smaller parts, such as sentences and words. These smaller parts are called tokens. For example, a word is a token in a sentence, and a sentence is a token in a paragraph. As we know that NLP is used to build applications such as sentiment analysis, QA systems, language translation, smart chatbots, voice systems, etc., hence, in order to build them, it becomes vital to understand the pattern in the text. The tokens, mentioned above, are very useful in finding and understanding these patterns. We can consider tokenization as the base step for other recipes such as stemming and lemmatization.

Splitting the sentence into words.

Tokenization is the first step in text analytics. It is the process of breaking down a textual content paragraph into smaller chunks.

A token is a single entity which is building block for sentence or paragraph. For example, each word is a token if a sentence “tokenized” in words.

Every word is a token if a sentence is tokenized into a word.

```
# Tokenization
def tokenize(text):
    token=word_tokenize(text)
    return token

df["token_Text"]=df["clean_Text"].apply(tokenize)
df.head()
```

➢Stop words removal

What are stopwords?

Some common words that are present in text but do not contribute in the meaning of a sentence. Such words are not at all important for the purpose of information retrieval or natural language processing. The most common stopwords are ‘the’ and ‘a’

• Stop words are very commonly used words (a, an, the, etc.) in the documents.

• These words do not really signify any importance as they do not help in distinguishing two documents.

```
# Remove stop words
from nltk.corpus import stopwords
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words('english')
def remove_stopwords(text):
  txt_clean = [word for word in text if word not in stopwords]
  return txt_clean
df["stop_Text"]=df["token_Text"].apply(remove_stop words)
df.head()
```

➢ Stemming :

What is Stemming?

Stemming is a technique used to extract the base form of the words by removing affixes from them. It is just like cutting down the branches of a tree to its stems. For example, the stem of the words eating, eats, eaten is eat. Search engines use stemming for indexing the words. That’s why rather than storing all forms of a word, a search engine can store only the stems. In this way, stemming reduces the size of the index and increases retrieval accuracy.

Stemming is the process of removing a part of a word, or reducing a word to its stem or root.

A stemming algorithm might also reduce the words fishing, fished, and fisher to the stem fish.

The stem need not be a word, for example the Porter algorithm reduces, argue, argued, argues, arguing, and argus to the stem argu.

```
from nltk.stem import PorterStemmer
ps=PorterStemmer()
def stemming(text):
    stem_text=[ps.stem(word) for word in text]
    return stem_text
df["stem_Text"]=df["stop_Text"].apply(stemming)
df.head()
```

➢Bag of Word :

- A bag of words is a representation of text that describes the occurrence of words within a document.
- We just keep track of word counts and disregard the grammatical details and the word order.
- It is called a “bag” of words because any information about the order or structure of words in the document is discarded.
- A bag-of-words model is a way of extracting features from text for use in modelling, such as with Machine Learning algorithms. In this approach we look at the histogram of the words within the text. i.e. Considering each word count as a feature.

```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
CountVec=CountVectorizer(max_features=5000,ngram_range=(1,3))
x=CountVec.fit_transform(df.Fin_Text).toarray()
```

```
x_bow=pd.DataFrame(x, columns=CountVec.get_feature_names())
x_bow.head()
```

53]:

	aaa	abandon	abba	abc	abil	abl	abolish	abort	abroad	absenc	...	youth	yuan	yugansk	yuganskneftega	yuko
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 5000 columns

TF-IDF (Term Frequency - Inverse Document Frequency):

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

$$IDF = \log\left(\frac{\text{Number of sentences}}{\text{Number of sentences containing words}}\right)$$

$$\text{TF} = \frac{\text{Number of repetations of word in sentence}}{\text{Number of words in sentence}}$$

$$TF - IDF = TF * IDF$$

```
1 tfidf=TfidfVectorizer(max_features=5000,ngram_range=(1,3))
2 x=tfidf.fit_transform(df.Fin_Text).toarray()
```

```
1 x_tfidf=pd.DataFrame(x, columns=tfidf.get_feature_names())
2 x_tfidf.head()
```

]:

	aaa	abandon	abba	abc	abil	abl	abolish	abort	abroad	absenc	...	youth	yuan	yugansk	yuganskneftega
0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.022472	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

5 rows × 5000 columns

Fitting Machine Learning Algorithms.

➢ Naïve-Bayes :

- Naive Bayes is a kind of classifier which uses the Bayes Theorem.
- It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class.
- They calculate the probability of each category for a given text and then output the category with the highest one.

Advantage :

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction.
- When assumption of independence holds, with less training data a Naive Bayes classifier performs better compare to other models like logistic regression.

```
from sklearn.metrics import confusion_matrix,accuracy_score
print("Accuracy of Naive-Bayes model using BOW is : ",accuracy_score(ytest_bow,ypred_bow))
print("Accuracy of Naive-Bayes model using TF-IDF is : ",accuracy_score(ytest_tfidf,ypred_tfidf))
```

```
Accuracy of Naive-Bayes model using BOW is :  0.7095808383233533
Accuracy of Naive-Bayes model using TF-IDF is :  0.7125748502994012
```

```
from sklearn.naive_bayes import MultinomialNB
NB_model_bow=MultinomialNB()
NB_model_bow.fit(xtrain_bow,ytrain_bow)
ypred_bow=NB_model_bow.predict(xtest_bow)

NB_model_tfidf=MultinomialNB()
NB_model_tfidf.fit(xtrain_tfidf,ytrain_tfidf)
ypred_tfidf=NB_model_tfidf.predict(xtest_tfidf)
```

➢SVM :

- For classification problem statements SVM tries to differentiate data points of different classes by finding a hyperplane that maximize the margin between the classes in the training data.

```
from sklearn.svm import SVC
SV_model_bow=SVC(C=1)
SV_model_bow.fit(xtrain_bow,ytrain_bow)
ypred_bow=SV_model_bow.predict(xtest_bow)

SV_model_tfidf=SVC(C=1)
SV_model_tfidf.fit(xtrain_tfidf,ytrain_tfidf)
ypred_tfidf=SV_model_tfidf.predict(xtest_tfidf)
```

```
print("Accuracy of SVC model using BOW is : ",accuracy_score(ytest_bow,ypred_bow))
print("Accuracy of SVC model using TF-IDF is : ",accuracy_score(ytest_tfidf,ypred_tfidf))
```

```
Accuracy of SVC model using BOW is :  0.7065868263473054
Accuracy of SVC model using TF-IDF is :  0.6976047904191617
```

➢ Decision Tree:

- A Decision Tree is a supervised machine learning algorithm that can be used for both regression and classification problem .
- Decision Tree divides complete dataset into smaller subsets while at a same time associated decision tree is incrementally develop.
- Final output of the Decision tree is a tree having decision nodes and leaf nodes.

```
from sklearn.tree import DecisionTreeClassifier
DT_model_bow=DecisionTreeClassifier()
DT_model_bow.fit(xtrain_bow,ytrain_bow)
ypred_bow=DT_model_bow.predict(xtest_bow)

DT_model_tfidf=DecisionTreeClassifier()
DT_model_tfidf.fit(xtrain_tfidf,ytrain_tfidf)
ypred_tfidf=DT_model_tfidf.predict(xtest_tfidf)
```

```
print("Accuracy of Decision Tree  model using BOW is : ",accuracy_score(ytest_bow,ypred_bow))
print("Accuracy of Decision Tree model using TF-IDF is : ",accuracy_score(ytest_tfidf,ypred_tfidf))
```

```
Accuracy of Decision Tree  model using BOW is :  0.46407185628742514
Accuracy of Decision Tree model using TF-IDF is :  0.49251497005988026
```

➢Random Forest :

- Random forest is an ensemble algorithm that follows the bagging technique. It is an extension of bagging estimator algorithm the base estimators in random forest are decision tree.
- Random Forest randomly select set of features which are used to decide a best split at each node of the decision tree.
- The final prediction is taken by the majority of class from all decision trees.

```
from sklearn.ensemble import RandomForestClassifier
RF_model_bow=RandomForestClassifier(n_estimators=150)
RF_model_bow.fit(xtrain_bow,ytrain_bow)
ypred_bow=RF_model_bow.predict(xtest_bow)

RF_model_tfidf=RandomForestClassifier(n_estimators=150)
RF_model_tfidf.fit(xtrain_tfidf,ytrain_tfidf)
ypred_tfidf=RF_model_tfidf.predict(xtest_tfidf)
```

```
print("Accuracy of Random Forest model using BOW is : ",accuracy_score(ytest_bow,ypred_bow))
print("Accuracy of Random Forest model using TF-IDF is : ",accuracy_score(ytest_tfidf,ypred_tfidf))
```

```
Accuracy of Random Forest model using BOW is :  0.6916167664670658
Accuracy of Random Forest model using TF-IDF is :  0.6811377245508982
```

➢Word Embedding :

- Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.
- Loosely speaking, they are vector representations of a particular word.
- Why we use Word Embedding?
- Bag of word and TF-IDF has one disadvantage that it does not gives semantic information. To overcome the problem of BOW and TF-IDF we use word embedding technique.
- Word embedding techniques include two techniques
- 1) Word2Vec
- 2) Glove
- Word2Vec :
- Word2Vec algorithm uses neural network model to learn association from large corpus of test. Once trained, such a model can detect a synonymous words or suggest additional words for a partial sentence.

- In this specific model, each word is basically represented as vector of 32 or more dimension instead of single number.
- Here the semantic information and the relationship between different words is also preserved.

➢Onehot Representation :

- A one hot encoding allows the representation of Text data to be more expressive because machine learning algorithms cannot work with text data directly. The words must be converted into numbers.
- Padding :
- To get equal number of size of sentence we add some zero values to left side of sentence by using ‘padding’ technique.

Onehot Representation

```
[ ]  onehot_rep=[one_hot(words,voc_size)for words in Text]
     onehot_rep[0]
```

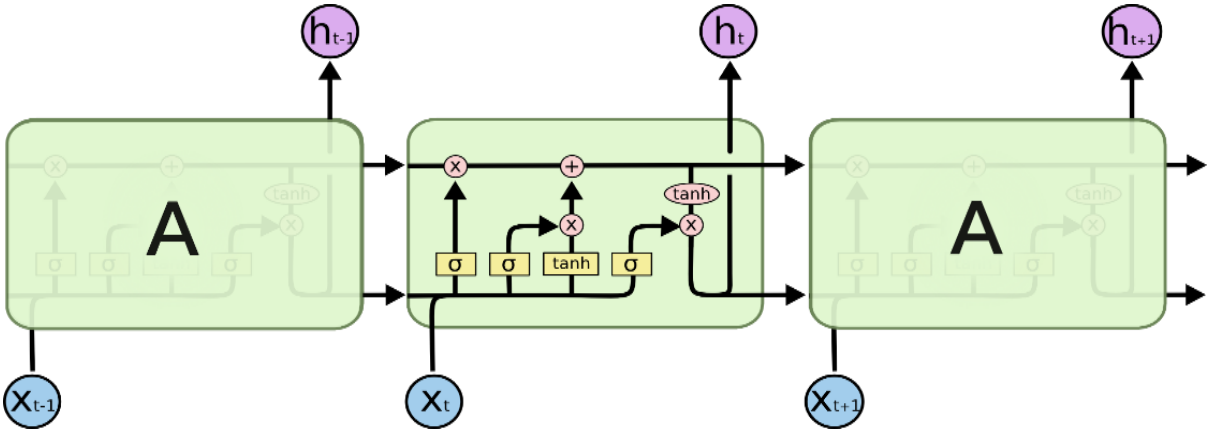
Embendding Representation

```
[ ]  sent_length=2200
     embedded_docs=pad_sequences(onehot_rep,padding='pre',maxlen=sent_length)
     print(embedded_docs)
```

```
[[   0    0    0 ... 2797 6872 5103]
 [   0    0    0 ... 3063 8884 7993]
 [   0    0    0 ... 1509 1558 2807]
 ...
 [   0    0    0 ...  702 7200  732]
 [   0    0    0 ... 4904 7928 5812]
 [   0    0    0 ... 1216  702 8117]]
```

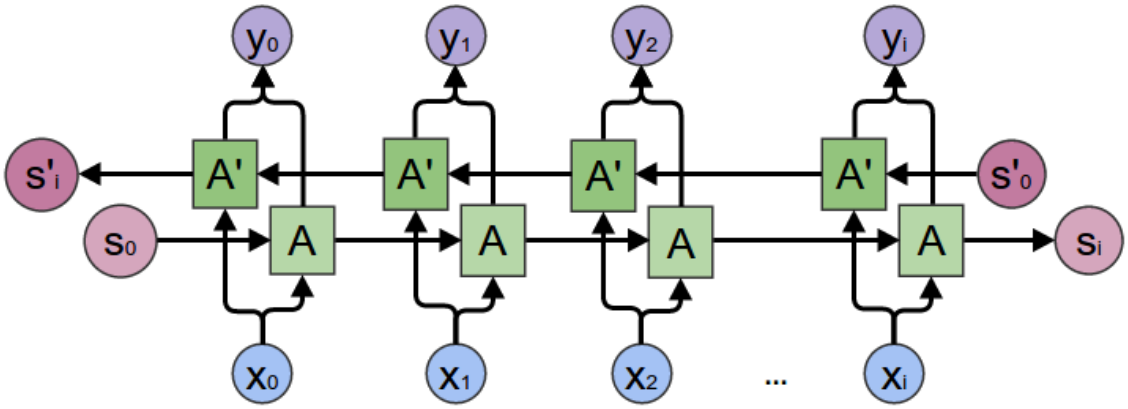
➢ LSTM Model :

- Humans don’t start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.
- Traditional neural networks can’t do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It’s unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.
- Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.
- Long Short-Term Memory networks usually just called “LSTMs” are a special kind of RNN, capable of learning long- term dependencies. They work tremendously well on a large variety of problems, and are now widely used.
- LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour , not something they struggle to learn!



➢Bidirectional LSTM :

- Bidirectional LSTMs are an extension of traditional LSTMs that can improve model performance on sequence classification problems. In problems where all time steps of the input sequence are available, Bidirectional LSTMs train two instead of one LSTMs on the input sequence.



```
### Creating model
embedded_vector_features=100
model=Sequential()
model.add(Embedding(voc_size,embedded_vector_features,input_length=sent_length))
model.add(Bidirectional(LSTM(300)))
model.add(Dropout(0.2))
model.add(Dense(5,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_18 (Embedding)     (None, 2200, 100)         1000000
_________________________________________________________________
bidirectional_23 (Bidirectio (None, 600)               962400
_________________________________________________________________
dropout_19 (Dropout)         (None, 600)               0
_________________________________________________________________
dense_13 (Dense)             (None, 5)                 3005
=================================================================
Total params: 1,965,405
Trainable params: 1,965,405
Non-trainable params: 0
_________________________________________________________________
None
```

```
model.fit(xtrain,ytrain,validation_data=(xtest,ytest),epochs=5,batch_size=32,verbose=1)
```

```
Epoch 1/5
49/49 [==============================] - 64s 1s/step - loss: 1.6141 - accuracy: 0.2408 - val_loss: 1.5596 - val_accuracy: 0.3308
Epoch 2/5
49/49 [==============================] - 59s 1s/step - loss: 1.4082 - accuracy: 0.4753 - val_loss: 1.3658 - val_accuracy: 0.5075
Epoch 3/5
49/49 [==============================] - 59s 1s/step - loss: 0.9747 - accuracy: 0.7148 - val_loss: 1.3371 - val_accuracy: 0.5195
Epoch 4/5
49/49 [==============================] - 59s 1s/step - loss: 0.5623 - accuracy: 0.8266 - val_loss: 1.4211 - val_accuracy: 0.5045
```

- Model Accuracy Table

:

ML Models	Accuracy with BOW	Accuracy with TF-IDF
Naïve Bayes	70.95%	71.25%
SVM	69.16%	68.11%
Decision Tree	46.40%	49.25%
Random Forest	69.16%	68.11%

Neural Network Model	Accuracy
Bidirectional LSTM	51.25%

- Conclusion :
- From the above models we conclude that for sorting of BBC's news article Naïve Bayes gives highest Accuracy i.e 71.25 % using TF-IDF feature extraction method.

Thank You..