DEEP LEARNING – NLP PROJECT
NEWS ARTICLE CLASSIFICATION

ROLL NO: 2018103592 **DATE: 21 May 2021**
NAME: SHRUTHI M

PROBLEM STATEMENT

Classifying the semantic content of text is one of the challenging problems in natural language processing and machine learning. This includes classifying emotions, email spam detection, movie review analysis, news article tagging and so on. Newspaper article topic classification can enable automatic tagging of articles that helps to categorize different articles. One of the problem here is an article can come under multiple tags and putting them under one category is difficult. Text classification is not easy unlike other classification problems. This is because there could be human errors while typing, words can be in upper as well as lower case along with different punctuations. Also, there could be 2 different words that mean the same. Humans can identify all these but it is quite complicated for machines to recognise. Here, we use Tf-Idf vectorization followed by 2 approaches: One is the logistic regression model and the other is a deep neural network CNN model. We observe that both these models provide similar accuracy results.

DATASET DETAILS

https://www.kaggle.com/rmisra/news-category-dataset

The dataset contains around 200k news headlines with a short description from the year 2012 to 2018 obtained from HuffPost. The features present are news headline, short description, names of the reporter, URL, and date of the incident. Each news headline has a corresponding category. There are around 30 categories including but not limited to Politics, Business, Entertainment, Women, Religion, Science, Tech etc. The dataset is skewed as some of the categories have more examples than the others. Also, we do not use all the 200k news articles. We limit to 20k articles from 5 different categories with a balanced dataset.

MODULES

EXPLORING AND VISUALISING DATASET

The dataset contains 200k news articles as JSON response. The first task is to convert JSON entries to a data frame. Following this, we balance the skewed dataset by taking 4000 articles of 5 categories. The 5 categories are: POLITICS, ENTERTAINMENT, SPORTS, FOOD&DRINK, and BUSINESS. There are few features that are not required for classification. Hence, we drop those features and use the necessary ones alone.

TEXT PRE-PROCESSING

- **REMOVING STOP WORDS AND NOISE**

  Here, we convert the words in text to lower case and remove punctuations as they add noise to the text. Words are also tokenized so that they can be used in the following steps.

- **STEMMING**

  Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. The most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective, is Porter's algorithm. Porter's algorithm consists of 5 phases of word reductions, applied sequentially. Within each phase there are various conventions to select rules, such as selecting the rule from each rule group that applies to the longest suffix.

  Sses → ss
  Ies → i
  Ss → ss
  S →

- **LEMMATIZATION**

  Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. Doing full morphological analysis produces at most very modest benefits for retrieval.
  Example: operating, operator, operational all are reduced to their root word – operate

LABEL ENCODING AND ONE HOT ENCODING

Since, we have 5 different categories, we prefer label encoding for logistic regression and one hot encoding for CNN model.

In **label encoding**: Business category belongs to class 0, Entertainment to class 1, Food and Drink to class 2, Politics to class 3, and Sports to class 4.

In **one hot encoding:** Business is one-hot encoded as: [1,0,0,0,0], Entertainment as [0,1,0,0,0], Food and Drink as [0,0,1,0,0], Politics as [0,0,0,1,0], and Sports as [0,0,0,0,1].

SPLIT TRAINING AND VALIDATION SET

20k text entries are split with a percentage of 0.1. Hence, 18k samples for training and 2k for testing purposes.

TF-IDF (TERM FREQUENCY- INVERSE DOCUMENT FREQUENCY)

Not all words are equally important to a particular document / category. For example, while words like ‘murder’, ‘knife’ and ‘abduction’ are important to a crime related document, words like ‘news’ and ‘reporter’ may not be quite as important. Here, we use Tf-Idf as feature weighting model. The more common the word across documents, the lower its score and the more unique a word is, the higher the score.

For a term **t** in document **d**, the weight **Wtd** of term t in document d is given by:

$$Wtd = TFtd \log (N/DFt)$$

- TFtd is the number of occurrences of t in document d.
- DFt is the number of documents containing the term t.
- N is the total number of documents in the corpus.

Here, we perform Tf-Idf with logistic regression and CNN

TF-IDF WITH LOGISTIC REGRESSION

Once, the words are transformed and assigned scores based on tf-idf vectorizer, we perform logistic regression. Here, we set the solver to **‘lbfgs’** as we deal with multi class labelling. **N-grams** range is set to (1,2) and hence 1 or 2 words will be extracted at a time.

STRATIFIED K-FOLD CROSS VALIDATION

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. K value of 5 indicates that the sample will be split into groups of 5.

PREDICTIONS

As we know, a news article may belong to multiple categories. Hence, here we take top 3 predictions.

EVALUATION METRICS

- **CONFUSION MATRIX**
  Confusion matrix is plotted for the 5 different classes.

- **CLASSIFICATION REPORT**
  Accuracy, Precision, Recall, F1 scores are plotted for each.

TF-IDF WITH CNN

BUILDING THE MODEL

Here, we come up with our own model for the problem. There are 5 different layers and a final layer with 5 nodes. The optimizer we use here is **'Adam'** and since it is multi-class classification **'categorical cross entropy'** is used as loss function.

VALIDATION SPLIT

Of the 18k training sample, we use 0.1% (1800) for validation and the remaining for training.

FITTING THE MODEL

The vectorized words are now fit into the model

PLOTTING GRAPHS

We plot training and validation errors and accuracy graph

EVALUATION METRICS

- **CONFUSION MATRIX**
  Confusion matrix is plotted for the 5 different classes.

- **CLASSIFICATION REPORT**
  Accuracy, Precision, Recall, F1 scores are plotted for each.

CODING SNAPHOTS

MODULE I – EXPLORING AND VISUALISING DATASET

Importing Packages

'nltk' (Natural Language ToolKit) is the library package that contains the corpus for stop words and for lemmatization.

Importing packages

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import json
import re
import itertools
import nltk
import tensorflow as tf

nltk.download('stopwords') # for stopwords
nltk.download('punkt')    # for tokenizing
nltk.download('wordnet')  # for lemmatizing
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_fscore_support
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense, BatchNormalization
from keras.utils import to_categorical

import warnings
warnings.filterwarnings("ignore")

from tensorflow.keras.callbacks import EarlyStopping
tf.config.run_functions_eagerly(True)
```

Reading JSON file

Reading JSON file

In [2]:
```
data = []
with open('News Dataset.json', mode='r', errors='ignore') as json_file:
    for dict in json_file:
        data.append( json.loads(dict) )
```

In [3]:
```
print("No of news articles: ", len(data))
print("Random data: " , data[50])
```

```
No of news articles:  200853
Random data:  {'category': 'ENTERTAINMENT', 'headline': "James Corden And Adam Levine Get Pulled Over During 'Carpool Karaoke'", 'authors': 'Ed Mazza', 'link': 'https://www.huffingtonpost.com/entry/james-corden-adam-levine-carpool-karaoke-pulled-over_us_5b07b025e4b0fdb2aa5206b9', 'short_description': 'It was bound to happen.', 'date': '2018-05-25'}
```

Out of the 5 features: Headline, authors, link, short description, date we use only headline and short description as other features are not important for semantic understanding of the text.

Only category, headline and short description is relevant. Hence, we use only these 3 columns

In [4]:
```
df = pd.DataFrame(data)
df = df[['category', 'headline', 'short_description']]
df
```

Out[4]:

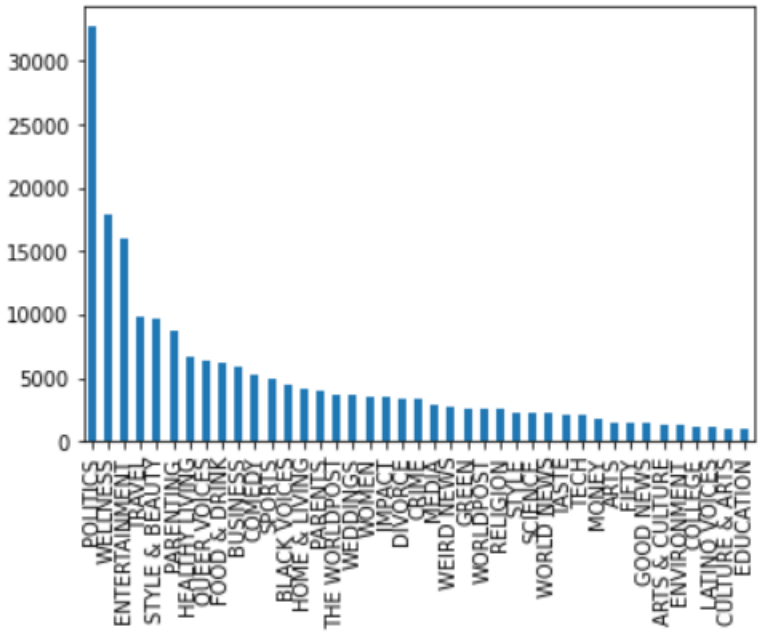
	category	headline	short_description
0	CRIME	There Were 2 Mass Shootings In Texas Last Week...	She left her husband. He killed their children...
1	ENTERTAINMENT	Will Smith Joins Diplo And Nicky Jam For The 2...	Of course it has a song.
2	ENTERTAINMENT	Hugh Grant Marries For The First Time At Age 57	The actor and his longtime girlfriend Anna Ebe...
3	ENTERTAINMENT	Jim Carrey Blasts 'Castrato' Adam Schiff And D...	The actor gives Dems an ass-kicking for not fi...
4	ENTERTAINMENT	Julianna Margulies Uses Donald Trump Poop Bags...	The "Dietland" actress said using the bags is ...
...	...	...	...
200848	TECH	RIM CEO Thorsten Heins' 'Significant' Plans Fo...	Verizon Wireless and AT&T are already promotin...
200849	SPORTS	Maria Sharapova Stunned By Victoria Azarenka I...	Afterward, Azarenka, more effusive with the pr...
200850	SPORTS	Giants Over Patriots, Jets Over Colts Among M...	Leading up to Super Bowl XLVI, the most talked...
200851	SPORTS	Aldon Smith Arrested: 49ers Linebacker Busted ...	CORRECTION: An earlier version of this story i...
200852	SPORTS	Dwight Howard Rips Teammates After Magic Loss ...	The five-time all-star center tore into his te...

200853 rows × 3 columns

Bar plot of 30 categories

In [5]:
```
df['category'].value_counts().plot(kind='bar')
```

Out[5]: <AxesSubplot:>



From the above bar plot, we infer that data is skewed. To maintain balance, we choose only 5 categories and 4000 samples from each category.

There are 30 categories and since the count is skewed we take only 5 categories and 4000 values from each category ¶

In [6]:
```
#  politics, entertainment, sports, business, food & drink
df = df[df['category'].isin(['POLITICS', 'ENTERTAINMENT', 'SPORTS', 'BUSINESS', 'FOOD & DRINK'])]
df.shape
```

Out[6]: (65844, 3)

In [7]:
```
df_politics = df[df['category'].isin(['POLITICS'])].sample(n=4000)
df_entertainment = df[df['category'].isin(['ENTERTAINMENT'])].sample(n=4000)
df_sports = df[df['category'].isin(['SPORTS'])].sample(n=4000)
df_business = df[df['category'].isin(['BUSINESS'])].sample(n=4000)
df_foodanddrink = df[df['category'].isin(['FOOD & DRINK'])].sample(n=4000)
```

In [8]:
```
print(df_politics.shape, df_entertainment.shape, df_sports.shape, df_business.shape, df_foodanddrink.shape)
```

```
(4000, 3) (4000, 3) (4000, 3) (4000, 3) (4000, 3)
```

In [9]:
```
new_df = df_politics
new_df = new_df.append(df_entertainment, ignore_index = True)
new_df = new_df.append(df_sports, ignore_index = True)
new_df = new_df.append(df_business, ignore_index = True)
new_df = new_df.append(df_foodanddrink, ignore_index = True)
# + df_entertainment + df_sports + df_business + df_foodanddrink
```

Headline and short description are combined to form another column

In [10]:
```
new_df["headline and short_description combined"] = new_df["headline"] + new_df["short_description"]
```

In [11]:
```
print('Shape of redefined data: ', new_df.shape)
new_df
```

```
Shape of redefined data:  (20000, 4)
```

Out[11]:

	category	headline	short_description	headline and short_description combined
0	POLITICS	California Governor Says Trump Is Courting The...	Jerry Brown rips the president's "reckless di...	California Governor Says Trump Is Courting The...
1	POLITICS	The Wars Come Home	Unlike in the Vietnam era, the U.S. has been f...	The Wars Come HomeUnlike in the Vietnam era, t...
2	POLITICS	Ted Cruz Thrills Crowd With Spot-On 'Princess ...	"I will confess to knowing an awful lot of tha...	Ted Cruz Thrills Crowd With Spot-On 'Princess ...
3	POLITICS	Thursday's Morning Email: Trump Administration...	Education Secretary Betsy DeVos initially oppo...	Thursday's Morning Email: Trump Administration...
4	POLITICS	Suit Against D.A. Who Used Fake Subpoenas To P...	Prosecutors are generally immune from accounta...	Suit Against D.A. Who Used Fake Subpoenas To P...
...	...	...	...	...
19995	FOOD & DRINK	The Gratuity Gulf	The average restaurant-goer in the U.S. tips t...	The Gratuity GulfThe average restaurant-goer i...
19996	FOOD & DRINK	The Recipes That'll Finally Make You Love Gril...	You won't miss the burgers for a second.	The Recipes That'll Finally Make You Love Gril...
19997	FOOD & DRINK	Where to Eat in Austin During South by Southwest	Franklin Barbecue, one of the most popular bar...	Where to Eat in Austin During South by Southwe...
19998	FOOD & DRINK	5 Fish Should You Be Eating Now and Why	The healthiest picks of fish to eat.	5 Fish Should You Be Eating Now and WhyThe hea...
19999	FOOD & DRINK	Best Of The Oscars Cocktail Buffet	We're skipping the uncertainty of cooking for ...	Best Of The Oscars Cocktail BuffetWe're skippi...

20000 rows × 4 columns

MODULE II – TEXT PRE-PROCESSING

➢ REMOVING STOP WORDS AND NOISE

Text Pre-processing

Removing stop words and noise

Convert to lower case. Remove punctuations, tokenize and remove sto words

In [13]:
```
stop_words = set(stopwords.words('english'))
text = "This project is based on text processing. Hope the accuracy will be good."
text = re.sub(r'[^\w\s]', '', str(text).lower().strip())
word_tokens = word_tokenize(text)
filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print("Filtered sentence after removing stop words: " , filtered_sentence)
```

```
Filtered sentence after removing stop words:  ['project', 'based', 'text', 'processing', 'hope', 'accuracy', 'good']
```

➢ STEMMING

Stemming

Remove suffixes like -ed, -ing, -ly, -tion

In [14]:
```
ps = PorterStemmer()
stemmed_word = []
for w in filtered_sentence:
    stemmed_word.append(ps.stem(w))

print("Stemmed words: " , stemmed_word)
```

```
Stemmed words:  ['project', 'base', 'text', 'process', 'hope', 'accuraci', 'good']
```

➢ LEMMATIZATION

Lemmatization

Eliminate redundant prefix or suffix and get the root word

In [15]:
```
lemmatizer = WordNetLemmatizer()
lemmatized_word = []
for w in stemmed_word:
    lemmatized_word.append(lemmatizer.lemmatize(w))

print("Lemmatized words: ", lemmatized_word)

# Example
print("Example: ", lemmatizer.lemmatize("playing"))
```

```
Lemmatized words:  ['project', 'base', 'text', 'process', 'hope', 'accuraci', 'good']
Example:  playing
```

Combining all the pre-processing functions

Combining all preprocessing functions

In [16]:
```
def pre_process(text):
    text = re.sub(r'[^\w\s]', '', str(text).lower().strip())
    word_tokens = word_tokenize(text)
    filtered_sentence = []

    for w in word_tokens:
        if w not in stop_words:
            filtered_sentence.append(w)

    stemmed_word = []
    for w in filtered_sentence:
        stemmed_word.append(ps.stem(w))

    lemmatized_word = []
    for w in filtered_sentence:
        lemmatized_word.append(lemmatizer.lemmatize(w))

    text = " ".join(lemmatized_word)
    return text
```

In [17]:
```
new_df["headline_cleaned"] = new_df["headline"].apply(lambda x: pre_process(x))
new_df["short_description_cleaned"] = new_df["short_description"].apply(lambda x: pre_process(x))
new_df["headline_and_short_description_combined_cleaned"] = new_df["headline and short_description combined"].apply(lambda x: pr
```

MODULE III - LABEL ENCODING AND ONE HOT ENCODING

Label Encoding

Label Encoding

business - 0; entertainment - 1; food and drink - 2; politics - 3; sports - 4

In [20]:
```
label_encoder = preprocessing.LabelEncoder()
new_df['category'] = label_encoder.fit_transform(new_df['category'])
```

In [21]:
```
new_df
```

Out[21]:

	category	headline_cleaned	short_description_cleaned	headline_and_short_description_combined_cleaned
0	3	atlanta mayor kasim reed urge fellow dems reac...		atlanta mayor kasim reed urge fellow dems reac...
1	1	ceelo green say he alive well exploding cellph...	im ok want thank everybody love concern	ceelo green say he alive well exploding cellph...
2	3	gobsmacked londoner bash trump complains uk vi...	french also furious trump obscene acting paris...	gobsmacked londoner bash trump complains uk vi...
3	3	benghazi committee chair staffer fired classif...	staffer say fired sufficiently focusing hillar...	benghazi committee chair staffer fired classif...
4	4	mason madness inside unlikely run ncaa tournam...	there fear team trepidation thats thing like h...	mason madness inside unlikely run ncaa tournam...
...	...	...	...	...
19995	0	ficos new credit score impact consumer	fair issac corp creator widely used fico score...	ficos new credit score impact consumersfair is...
19996	2	oscar party recipe year academy award	long night prepare appropriately	oscar party recipe year academy awardsits long...
19997	0	creating new normal white men diversity table	white men could important group ensuring diver...	creating new normal white men diversity tablew...
19998	3	marco rubio sued give back pay harry reid say	floridian missed hearing announce reelection p...	marco rubio sued give back pay harry reid says...
19999	3	love high deductible youll love senate health ...	talk lowering outofpocket cost yeah didnt mean	love high deductible youll love senate health ...

20000 rows × 4 columns

One hot encoding

One hot encoding and validation split

In [49]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.1)
```

In [50]:
```
y_train_onehot = to_categorical(y_train, num_classes=5)
y_val_onehot = to_categorical(y_val, num_classes=5)
y_test_onehot =  to_categorical(y_test, num_classes = 5)
```

In [51]:
```
y_train_onehot
```

Out[51]:
```
array([[0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.]], dtype=float32)
```

MODULE IV - SPLIT TRAINING AND VALIDATION SET

Train and test split

```
In [22]: Y = new_df['category']
         new_df = new_df.drop(['category'], axis=1)
         X = new_df
```

```
In [23]: X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1, random_state = 100)
```

```
In [49]: X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.1)
```

MODULE V – TF-IDF

Here, we will have sample text documents and let's see how scores are obtained for each.

Tf-Idf

```
In [26]: docs = ["the house had a tiny little mouse",
         "the cat saw the mouse",
         "the mouse ran away from the house",
         "the cat finally ate the mouse",
         "the end of the mouse story"
         ]
```

```
In [27]: tfidf_vectorizer = TfidfVectorizer(use_idf=True)
         fitted_vectorizer = tfidf_vectorizer.fit(docs)
         tfidf_vectorizer_vectors = fitted_vectorizer.transform(docs)
```

```
In [28]: first_vector_tfidfvectorizer = tfidf_vectorizer_vectors[0]
         first_vector_tfidfvectorizer_score = pd.DataFrame(first_vector_tfidfvectorizer.T.todense(), index=tfidf_vectorizer.get_feature_na
         first_vector_tfidfvectorizer_score.sort_values(by=["tfidf"], ascending=False)
         first_vector_tfidfvectorizer_score
```

Out[28]:

	tfidf
ate	0.000000
away	0.000000
cat	0.000000
end	0.000000
finally	0.000000
from	0.000000
had	0.493562
house	0.398203
little	0.493562
mouse	0.235185
of	0.000000
ran	0.000000
saw	0.000000
story	0.000000
the	0.235185
tiny	0.493562

We observe that words like ‘mouse’ that occurs frequently has less score compared to words like ‘little’, ‘tiny’ which occurred only once.

MODULE V1 – TF-IDF WITH LOGISTIC REGRESSION

The maximum features hyper parameter is set to 1,50,000.

For logistic regression: we use ‘lbfgs’ solver as that is used for multi class classification. The weights are penalised by ‘l2’ norm. We observe that testing accuracy is around 87%

Tf-Idf with Logistic Regression

In [29]:
```
logit = LogisticRegression(C = 5e1, solver='lbfgs', multi_class='multinomial', penalty = 'l2', max_iter = 150)
```

In [30]:
```
text_transformer = TfidfVectorizer(ngram_range=(1, 2), max_features = 150000)
X_train_text = text_transformer.fit_transform(X_train['headline_and_short_description_combined_cleaned'])
X_test_text = text_transformer.transform(X_test['headline_and_short_description_combined_cleaned'])
```

In [31]:
```
logistic_model_with_tfidf = logit.fit(X_train_text, y_train['category'])
test_score = logistic_model_with_tfidf.score(X_test_text, y_test['category'])
print('Testing accuracy: ', test_score)
```

```
Testing accuracy:  0.8725
```

STRATIFIED K-FOLD CROSS VALIDATION

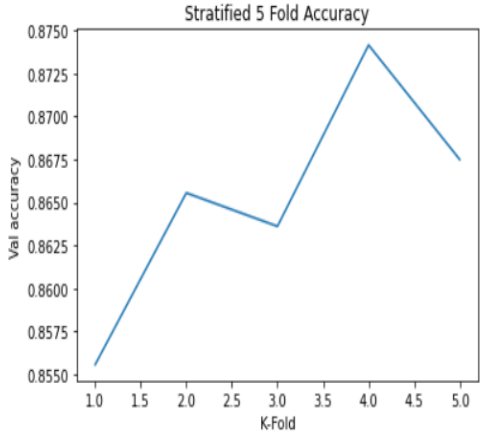
5 – fold cross validation is performed and the mean value achieved here is 86.52% which is similar to our testing accuracy. Hence, there is no over fitting in the model.

Stratified K-fold validation

In [32]:
```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=17)
cv_results = cross_val_score(logit, X_train_text, y_train['category'], cv=skf, scoring='f1_micro')
cv_results, cv_results.mean()
```

Out[32]:
```
(array([0.85555556, 0.86555556, 0.86361111, 0.87416667, 0.8675    ]),
 0.8652777777777777)
```

In [33]:
```
index = [1, 2, 3, 4, 5]
plt.plot(index, cv_results)
plt.title('Stratified 5 Fold Accuracy')
plt.xlabel('K-Fold')
plt.ylabel('Val accuracy')
plt.show()
```



PREDICTIONS

Predictions

In [34]:
```
def get_top_k_predictions(X_test_text, k):

    # get probabilities instead of predicted labels, since we want to collect top 3
    probs = logistic_model_with_tfidf.predict_proba(X_test_text)

    # GET TOP K PREDICTIONS BY PROB - note these are just index
    best_3 = np.argsort(probs, axis=1)[:,-k:]

    # GET CATEGORY OF PREDICTIONS
    preds=[[logistic_model_with_tfidf.classes_[predicted_category] for predicted_category in prediction] for prediction in best_
    preds =[ item[::-1] for item in preds]

    return preds
```

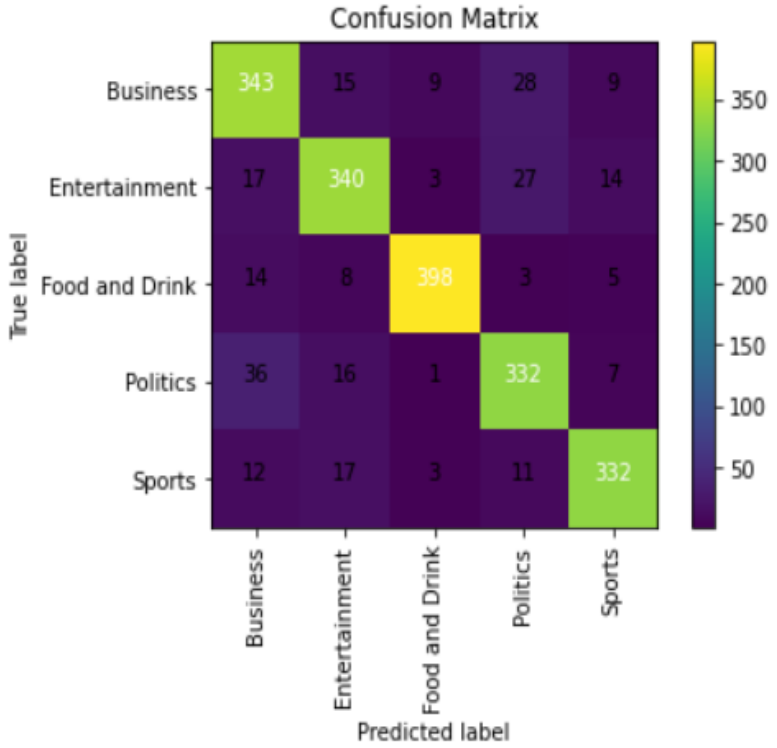
In [62]:
```
predicted_results = get_top_k_predictions(X_test_text, 3)

count = 0
y_predicted = []
y_test_array = y_test['category'].to_numpy()
for i in range(len(y_test_array)):
    if y_test_array[i] == predicted_results[i][0]:
        count += 1
    y_predicted.append(predicted_results[i][0])
```

EVALUATION METRICS

➢ CONFUSION MATRIX

```
Out[36]: Text(0.5, 0, 'Predicted label')
```



From the plot, we can infer that news articles belonging to food and drink category are identified most accurately compared to other categories.

➢ CLASSIFICATION REPORT

Classification Report

```
In [37]: report = classification_report(y_test_array, y_predicted, target_names=['0','1', '2', '3', '4'])
         print(report)
```

	precision	recall	f1-score	support
0	0.81	0.85	0.83	404
1	0.86	0.85	0.85	401
2	0.96	0.93	0.95	428
3	0.83	0.85	0.84	392
4	0.90	0.89	0.89	375
accuracy			0.87	2000
macro avg	0.87	0.87	0.87	2000
weighted avg	0.87	0.87	0.87	2000

MODULE VII – TF-IDF WITH CNN

BUILDING MODEL

```
model = Sequential()
model.add(Dense(256, input_dim=15000, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(160, activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())

model.add(Dense(120, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.5))

model.add(Dense(80, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(5, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

MODEL SUMMARY

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 256)               3840256
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024
_________________________________________________________________
dense_13 (Dense)             (None, 200)               51400
_________________________________________________________________
dropout_10 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_14 (Dense)             (None, 160)               32160
_________________________________________________________________
dropout_11 (Dropout)         (None, 160)               0
_________________________________________________________________
batch_normalization_7 (Batch (None, 160)               640
_________________________________________________________________
dense_15 (Dense)             (None, 120)               19320
_________________________________________________________________
dropout_12 (Dropout)         (None, 120)               0
_________________________________________________________________
dense_16 (Dense)             (None, 80)                9680
_________________________________________________________________
dropout_13 (Dropout)         (None, 80)                0
_________________________________________________________________
dense_17 (Dense)             (None, 5)                 405
=================================================================
Total params: 3,954,885
Trainable params: 3,954,053
Non-trainable params: 832
_________________________________________________________________
```

FITTING THE MODEL

The vectorized words are converted to ‘float’ data type and stored as numpy array. Early Stopping with a patience level of 5 is set to prevent over fitting. No of epochs is 25 with a batch size of 80.

Running the model

In [52]:
```
text_transformer = TfidfVectorizer(ngram_range=(1, 2), max_features = 15000)
X_train_text = text_transformer.fit_transform(X_train['headline_and_short_description_combined_cleaned']).astype('float16')
X_test_text = text_transformer.transform(X_test['headline_and_short_description_combined_cleaned']).astype('float16')
X_val_text = text_transformer.transform(X_val['headline_and_short_description_combined_cleaned']).astype('float16')
```

In [53]:
```
early_stopping = EarlyStopping(monitor='loss', patience=5, verbose=1)
history = model.fit(X_train_text, y_train_onehot, validation_data = (X_val_text, y_val_onehot), batch_size = 80, epochs = 25, ver
```

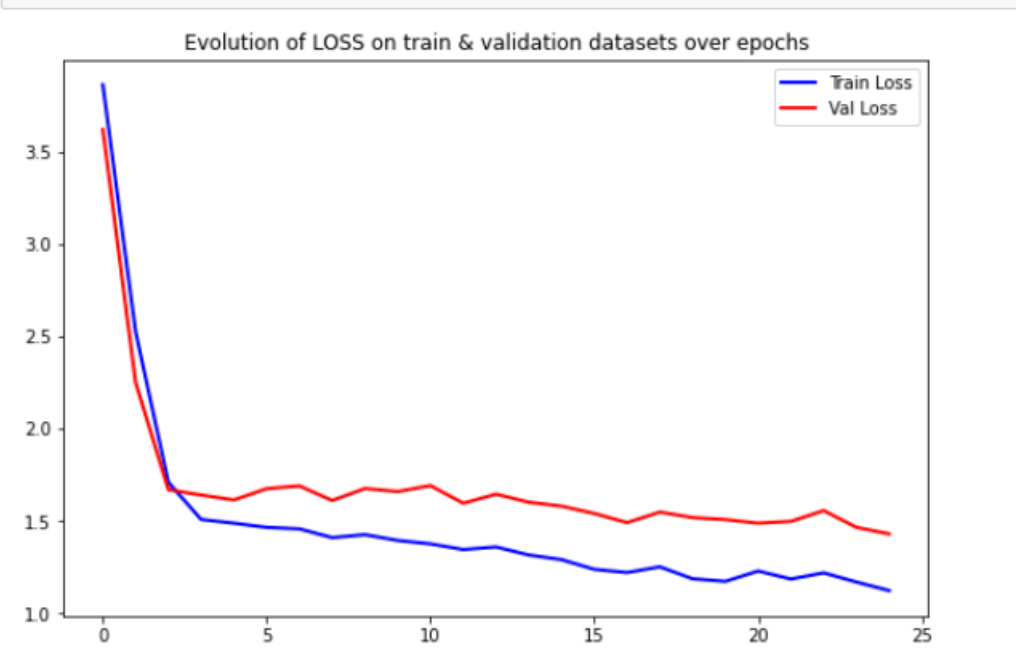
```
Epoch 1/25
203/203 [==============================] - 21s 103ms/step - loss: 4.3473 - accuracy: 0.2274 - val_loss: 3.6186 - val_accuracy:
0.2311
Epoch 2/25
203/203 [==============================] - 22s 110ms/step - loss: 2.8673 - accuracy: 0.5697 - val_loss: 2.2484 - val_accuracy:
0.8139
Epoch 3/25
203/203 [==============================] - 24s 118ms/step - loss: 1.7371 - accuracy: 0.8510 - val_loss: 1.6680 - val_accuracy:
0.8300
Epoch 4/25
203/203 [==============================] - 24s 117ms/step - loss: 1.4470 - accuracy: 0.8839 - val_loss: 1.6389 - val_accuracy:
0.8328
Epoch 5/25
203/203 [==============================] - 23s 113ms/step - loss: 1.4444 - accuracy: 0.8923 - val_loss: 1.6122 - val_accuracy:
0.8456
Epoch 6/25
203/203 [==============================] - 23s 115ms/step - loss: 1.4117 - accuracy: 0.9032 - val_loss: 1.6732 - val_accuracy:
0.8367
Epoch 7/25
203/203 [==============================] - 22s 107ms/step - loss: 1.4188 - accuracy: 0.9067 - val_loss: 1.6882 - val_accuracy:
0.8322
```

```
Epoch 20/25
203/203 [==============================] - 21s 102ms/step - loss: 1.1375 - accuracy: 0.9346 - val_loss: 1.5059 - val_accuracy:
0.8339
Epoch 21/25
203/203 [==============================] - 21s 101ms/step - loss: 1.2101 - accuracy: 0.9198 - val_loss: 1.4864 - val_accuracy:
0.8406
Epoch 22/25
203/203 [==============================] - 21s 105ms/step - loss: 1.1497 - accuracy: 0.9326 - val_loss: 1.4962 - val_accuracy:
0.8350
Epoch 23/25
203/203 [==============================] - 24s 120ms/step - loss: 1.2201 - accuracy: 0.9194 - val_loss: 1.5545 - val_accuracy:
0.8261
Epoch 24/25
203/203 [==============================] - 24s 116ms/step - loss: 1.1642 - accuracy: 0.9273 - val_loss: 1.4640 - val_accuracy:
0.8350
Epoch 25/25
203/203 [==============================] - 22s 106ms/step - loss: 1.0965 - accuracy: 0.9347 - val_loss: 1.4283 - val_accuracy:
0.8400
```

The initial training accuracy at first epoch is only around 22%. After 25 epochs, training accuracy is 93.5%

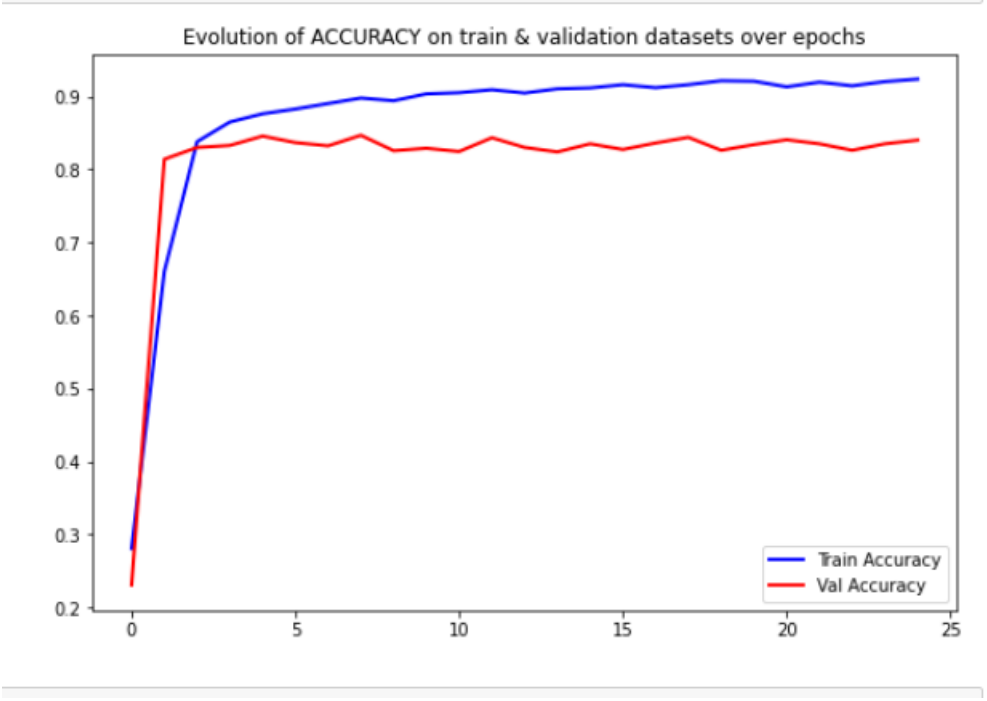
PLOTTING GRAPHS

Training and Validation Loss over 25 epochs



We observe that there is not much generalization gap and hence not overfitting.

Training and Validation accuracy

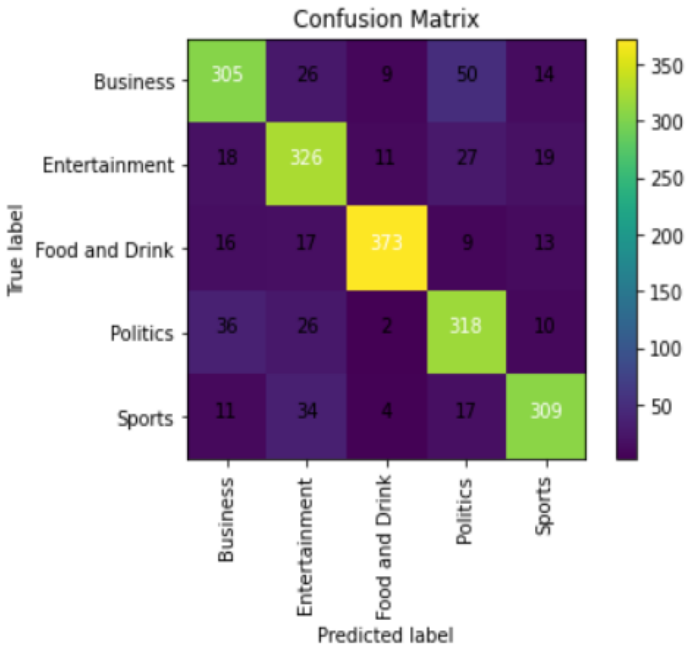


Training accuracy ends at 94% while validation accuracy comes around 85%

EVALUATION METRICS

- **CONFUSION MATRIX**

```
Out[88]: Text(0.5, 0, 'Predicted label')
```



➢ CLASSIFICATION REPORT

Classification report

```
In [89]: report = classification_report(y_true_label, y_pred_label, target_names=['0','1', '2', '3', '4'])
         print(report)
```

	precision	recall	f1-score	support
0	0.79	0.75	0.77	404
1	0.76	0.81	0.79	401
2	0.93	0.87	0.90	428
3	0.76	0.81	0.78	392
4	0.85	0.82	0.84	375
accuracy			0.82	2000
macro avg	0.82	0.81	0.82	2000
weighted avg	0.82	0.82	0.82	2000

CONCLUSION

Though Tf-Idf feature weighing model does not give importance to the position of a word in a text, from the results we have obtained we can conclude that for news article classification tf-idf has provided a significant accuracy of 87% with logistic regression and 82% with CNN. We can further enhance the model by increasing the o of epochs or tweaking the parameters. Other NLP techniques such as Word2Vec, GloVe can also be used.

REFERENCES

https://kavita-ganesan.com/news-classifier-with-logistic-regression-in-python/#.YKZPb6gzZPY

https://medium.com/analytics-vidhya/applying-text-classification-using-logistic-regression-a-comparison-between-bow-and-tf-idf-1f1ed1b83640