DTSA 5798 - Supervised Text Classification for Marketing Analytics

University of Colorado, Boulder

Master of Science in Data Science

News Category Dataset

1 Introduction

This project involves analyzing a dataset of news articles from the HuffPost, categorized into various topics. The primary objective is to employ supervised learning techniques to predict whether the article is about health and wellness.
This dataset contains around 210k news headlines from 2012 to 2022 from HuffPost. This is one of the biggest news datasets and can serve as a benchmark for a variety of computational linguistic tasks. HuffPost stopped maintaining an extensive archive of news articles sometime after this dataset was first collected in 2018, so it is not possible to collect such a dataset in the present day. Due to changes in the website, there are about 200k headlines between 2012 and May 2018 and 10k headlines between May 2018 and 2022.[1-3]
Each record in the dataset consists of the following attributes:

- category: category in which the article was published.
- headline: the headline of the news article.
- authors: list of authors who contributed to the article.
- link: link to the original news article.
- short_description: Abstract of the news article.
- date: publication date of the article.

There are a total of 42 news categories in the dataset.[1-3]

2 Exploratory Data Analysis(EDA)

2.1 Data Loading

We start by loading our training dataset, which consists of Huffington Post news articles, each labeled with a corresponding category.
Note: We'll be mounting our Google Drive on the collab environment and will access the training dataset there. We also import all the necessary libraries and check whether we have access to gpu which will be needed to facilitate our learning process.

```
from google.colab import drive
drive.mount('/content/drive')

import os
try:
  import ktrain
```

```
except:
  !pip install ktrain
  os.kill(os.getpid(), 9)
import ktrain
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud


gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)

reviews = pd.read_json("drive/MyDrive/
   news_category_trainingdata.json")

reviews.head()
```

	category	headline	authors	link	short_description	date
0	CRIME	There Were 2 Mass Shootings In Texas Last Week...	Melissa Jeltsen	https://www.huffingtonpost.com/entry/texas-ama...	She left her husband. He killed their children...	2018-05-26
1	ENTERTAINMENT	Will Smith Joins Diplo And Nicky Jam For The 2...	Andy McDonald	https://www.huffingtonpost.com/entry/will-smit...	Of course it has a song.	2018-05-26
2	ENTERTAINMENT	Hugh Grant Marries For The First Time At Age 57	Ron Dicker	https://www.huffingtonpost.com/entry/hugh-gran...	The actor and his longtime girlfriend Anna Ebe...	2018-05-26
3	ENTERTAINMENT	Jim Carrey Blasts 'Castrato' Adam Schiff And D...	Ron Dicker	https://www.huffingtonpost.com/entry/jim-carre...	The actor gives Dems an ass-kicking for not fi...	2018-05-26
4	ENTERTAINMENT	Julianna Margulies Uses Donald Trump Poop Bags...	Ron Dicker	https://www.huffingtonpost.com/entry/julianna-...	The "Dietland" actress said using the bags is ...	2018-05-26

2.2 EDA

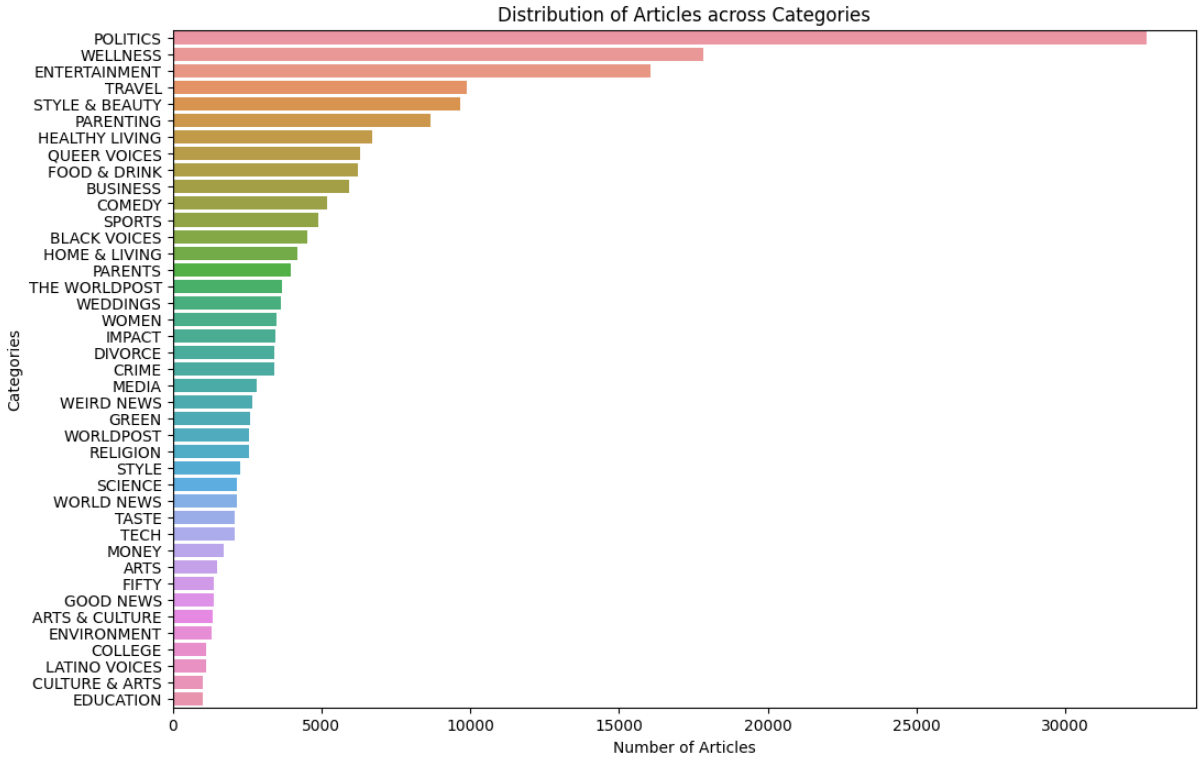
Let's start by looking at our dataset info() and missing values:

```
print(reviews.info())
print(reviews.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200853 entries, 0 to 200852
Data columns (total 6 columns):
 #   Column             Non-Null Count   Dtype         
---  ------             --------------   -----         
 0   category           200853 non-null  object        
 1   headline           200853 non-null  object        
 2   authors            200853 non-null  object        
 3   link               200853 non-null  object        
 4   short_description  200853 non-null  object        
 5   date               200853 non-null  datetime64[ns]
dtypes: datetime64[ns](1), object(5)
memory usage: 10.7+ MB
None
category             0
headline             0
authors              0
link                 0
short_description    0
date                 0
dtype: int64
```

we see that we don't have any missing values. We can look at the distribution of articles across different categories.

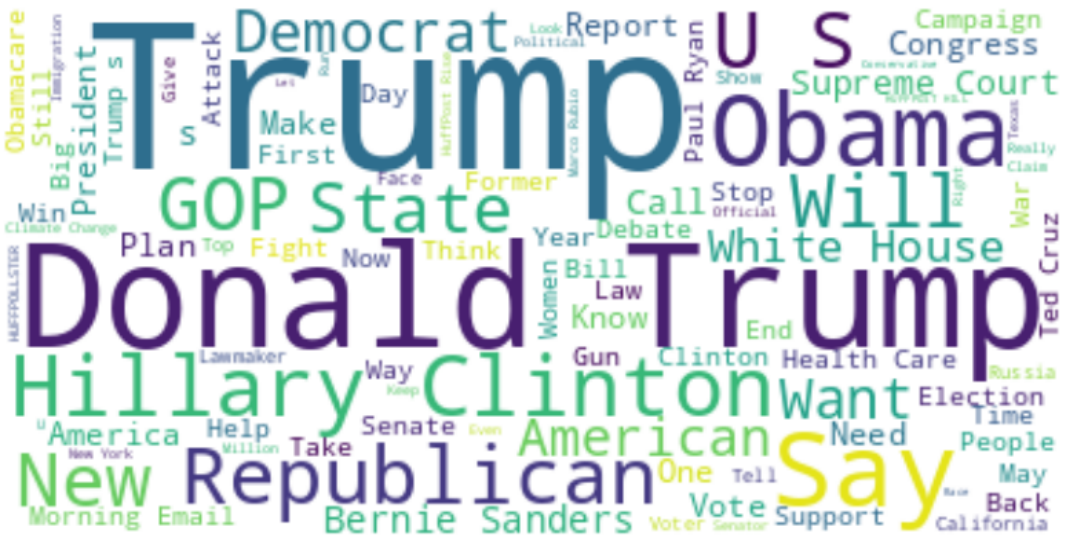
```
# Count the number of articles in each category
category_counts = reviews['category'].value_counts()

# Plot the distribution
plt.figure(figsize=(12, 8))
sns.barplot(x=category_counts, y=category_counts.index)
plt.xlabel('Number of Articles')
plt.ylabel('Categories')
plt.title('Distribution of Articles across Categories')
plt.show()
```



We see some imbalance in our dataset regarding different categories. We'll have this on mind for later when it comes to sampling. We can also do a sanity check by looking at the word cloud for different categories to see if the top words make sense regarding the category. For example, we can look at the word cloud for articles with politics as its category.

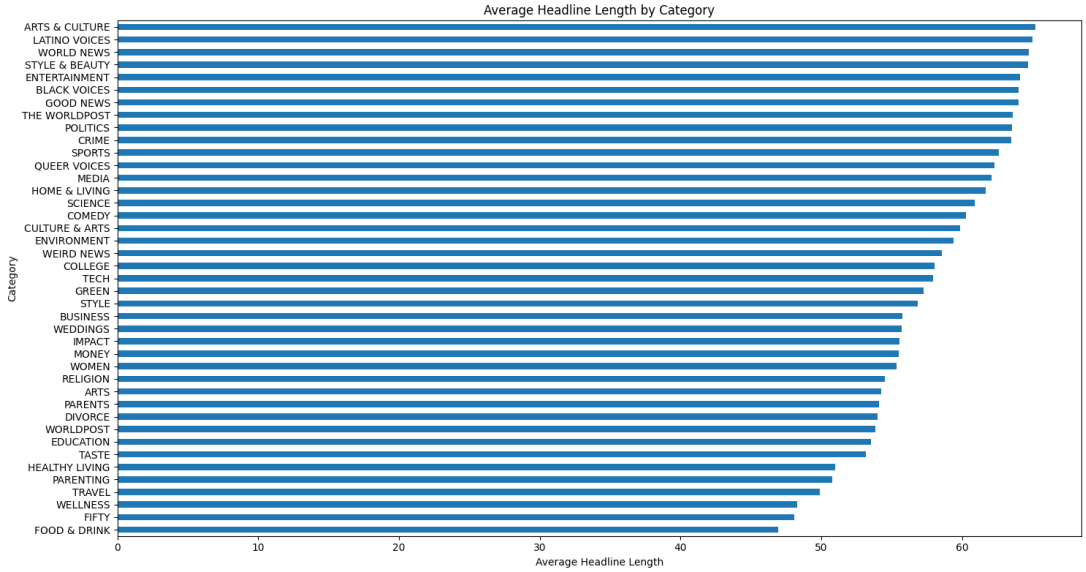
```
from wordcloud import WordCloud

# Join all headlines into a single string
politics_headlines = reviews[reviews['category'] == 'POLITICS'
    ]['headline']

# Join all headlines in the 'politics' category into a single
    string
politics_text = ' '.join(politics_headlines)

# Create and generate a word cloud image
wordcloud = WordCloud(background_color='white').generate(
    politics_text)

# Display the generated WordCloud
```

```
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

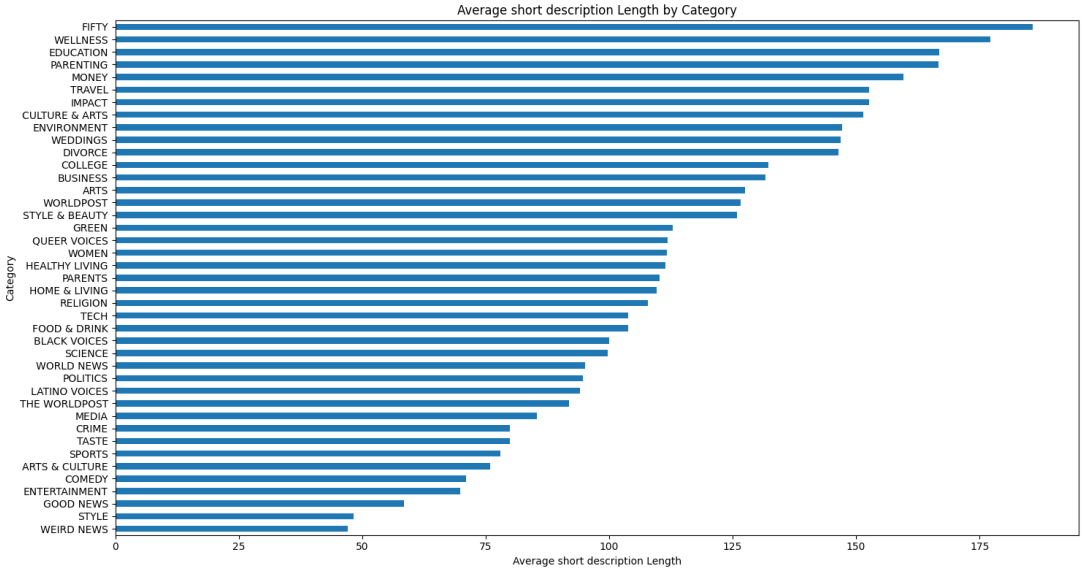


We also can look at the average headline and short_description length across different categories which later can be helpful when we are adjusting our maxlength parameter for our models.

```
# Calculate the length of each headline
reviews['headline_length'] = reviews['headline'].apply(len)

# Group by category and describe headline lengths
category_headline_length_stats = reviews.groupby('category')['
    headline_length'].describe()

# Plotting the average headline length for each category
plt.figure(figsize=(15, 8))
category_headline_length_stats['mean'].sort_values().plot(kind=
    'barh')
plt.xlabel('Average Headline Length')
plt.ylabel('Category')
plt.title('Average Headline Length by Category')
plt.tight_layout()
plt.show()
```




```
# Calculate the length of each headline
reviews['short_description_length'] = reviews['
    short_description'].apply(len)

# Group by category and describe headline lengths
category_short_description_length_stats = reviews.groupby('
    category')['short_description_length'].describe()

# Plotting the average headline length for each category
plt.figure(figsize=(15, 8))
category_short_description_length_stats['mean'].sort_values().
    plot(kind='barh')
plt.xlabel('Average short description Length')
plt.ylabel('Category')
plt.title('Average short description Length by Category')
plt.tight_layout()
plt.show()
```



3 Data Preparation

As mentioned in our lab work, in most machine learning models we need to have one column of type string; thus, we are going to merge the shortdescription and headline columns together, separated by a space.Also,note that since we are trying to zero in on only healthy and living categories, we need an extra binary variable to tell us whether the article is about health and wellness or not.

```
reviews['combined_text'] = reviews['headline'] + ' ' + reviews[
    'short_description']

reviews['healthy'] = np.where((reviews['category'] == 'HEALTHY
    LIVING'), 1, 0)
```

3.1 Balancing the data

For balancing our dataset, we sample the same amount of healthy and not healthy.

```
sample_amount = len(reviews[reviews["healthy"] == 1])

healthy = reviews[reviews['healthy'] == 1].sample(n=
    sample_amount)
not_healthy = reviews[reviews['healthy'] == 0].sample(n=
    sample_amount)
```

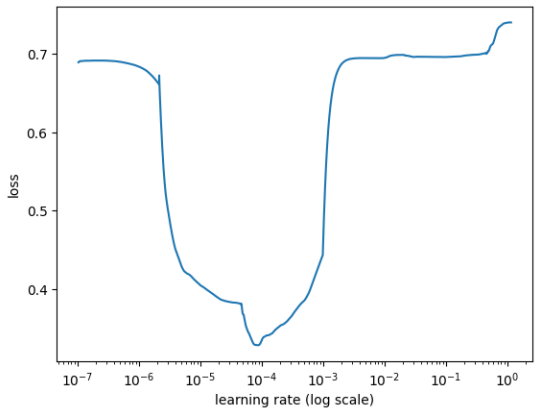
```
review_sample = pd.concat([healthy,not_healthy])
```

4 Model Selection, Training and parameter optimization

For our project,we decided to choose two different models: ”distilbertbaseuncased” and ”bertbaseuncased” or distilbert and bert for short.

There are many parameters that can be adjusted through ktrain such as val_df, max-feautures, maxlen, ngramrange and preprocessmode. The last one refers to the model that we’ll be using bert and distilbert. Since bert is a much larger model, we also adjust the maxlen parameter that helps us run the training with our gpu in the Google Collab. (high maxlen such as 512 will make our environment to run out of allocated memory but we know we don’t need much length from our EDA that showed 256 or 128 is more than enough.)
Regarding hyperparamter(learning rate) for which we’ll use learner.lr_find to optimize.We also need to note the batchsize in ktrain.getlearner method which also can lead into memory allocation failure if it is large. We’ll start off by setting batch size to 16 and maxlen to 128 for our distilbert model to see how it does against the benchmarks.

```
train, val, preprocess = ktrain.text.texts_from_df(
    review_sample,
    "combined_text",
    label_columns=["healthy"],
    val_df=None,
    max_features=20000,
    maxlen=128,
    val_pct=0.1,
    ngram_range=1,
    preprocess_mode="distilbert",
    verbose=1
)

model = preprocess.get_classifier()
learner = ktrain.get_learner(model, train_data=train, val_data=
   val, batch_size=16)

learner.lr_find(max_epochs=6)

learner.lr_plot()
```



Now we can set our learning rate to 1e-4 and use the learner.autofit method.(Note that the actual minimum of loss happened at approximately 8e-5 but we choose 1e-4 for the simplicity and it's close enough)

4.1 Validation Metrics

```
history=learner.autofit(
    1e-4,
    checkpoint_folder='checkpoint',
    epochs=10,
    early_stopping=True
)


predictor = ktrain.get_predictor(learner.model, preproc=
   preprocess)

validation = learner.validate(val_data=val, print_report=True)
```

```
begin training using triangular learning rate policy with max lr of 0.0001...
Epoch 1/10
754/754 [==============================] - 186s 231ms/step - loss: 0.3902 - accuracy: 0.8344 - val_loss: 0.3118 - val_accuracy: 0.8723
Epoch 2/10
754/754 [==============================] - ETA: 0s - loss: 0.2359 - accuracy: 0.9098Restoring model weights from the end of the best epoch: 1.
754/754 [==============================] - 171s 226ms/step - loss: 0.2359 - accuracy: 0.9098 - val_loss: 0.3345 - val_accuracy: 0.8745
Epoch 2: early stopping
Weights from best epoch have been loaded into model.
```

```
42/42 [==============================] - 7s 143ms/step
              precision    recall  f1-score   support

           0       0.93      0.82      0.87       692
           1       0.83      0.93      0.88       647

    accuracy                           0.87      1339
   macro avg       0.88      0.87      0.87      1339
weighted avg       0.88      0.87      0.87      1339
```

We can follow the same steps for our bert model as well.Training and validation metrics are as follow:(note that we had to change the batch size to 8 for bert since it's a larger model and it runs into memory allocation problems if we use the same 16 we used for our distilbert model)

```
begin training using triangular learning rate policy with max lr of 5e-05...
Epoch 1/10
1507/1507 [==============================] - 425s 266ms/step - loss: 0.3689 - accuracy: 0.8387 - val_loss: 0.2920 - val_accuracy: 0.8835
Epoch 2/10
1507/1507 [==============================] - ETA: 0s - loss: 0.2218 - accuracy: 0.9134Restoring model weights from the end of the best epoch: 1.
1507/1507 [==============================] - 403s 267ms/step - loss: 0.2218 - accuracy: 0.9134 - val_loss: 0.3021 - val_accuracy: 0.8813
Epoch 2: early stopping
Weights from best epoch have been loaded into model.
```

```
42/42 [==============================] - 17s 269ms/step
              precision    recall  f1-score   support

           0       0.93      0.82      0.87       648
           1       0.85      0.94      0.89       691

    accuracy                           0.88      1339
   macro avg       0.89      0.88      0.88      1339
weighted avg       0.89      0.88      0.88      1339
```

5 Conclusion

Below we can see our distilbert,bert and benchmark metrics, respectively.

5.1 Precision

- The precision for class 0 (negative class) is the highest in the BERT model (0.93) and the benchmark (0.88), indicating that BERT is better at correctly identifying the negative class as negative.
- The precision for class 1 (positive class) is the highest in the DistilBERT model (0.85), suggesting that DistilBERT is slightly more precise in identifying the positive class correctly.

5.2 Recall

- The recall for class 0 is the highest in the DistilBERT model (0.94), showing it is best at identifying all relevant instances of the negative class.
- The recall for class 1 is the highest in the DistilBERT model (0.93), indicating that DistilBERT is also better at capturing all relevant instances of the positive class.

5.3 F1-Score

- The F1-score, which balances precision and recall, is highest for both classes in the DistilBERT model (0.87 for class 0, 0.89 for class 1), showing that it has the best overall balance of precision and recall for both classes.

5.4 Support

- The support, which is the number of true instances for each class, shows that the classes are quite balanced across the models, which helps in making a fair comparison of the metrics.

5.5 Accuracy

- The overall accuracy is slightly higher for the DistilBERT model (0.88) compared to the BERT (0.87) and the benchmark (0.86). This means that DistilBERT correctly predicts a higher percentage of the total number of instances.

5.6 Macro and Weighted Averages

- The macro average treats all classes equally, and the weighted average takes the support of each class into account. DistilBERT has the highest macro and weighted averages, indicating it performs well across all classes and also when the class imbalance is considered.

In summary, the bert model outperforms the benchmark and distilbert model across almost all metrics, suggesting that it is the best performing model among the three for this specific text classification task. It is particularly better at identifying the correct instances for both classes, as evidenced by its higher recall and F1-scores. Distilbert excels in precisely identifying the negative class but falls slightly behind in other metrics. The benchmark model, while decent, is surpassed by the capabilities of the more advanced BERT and DistilBERT models.

```
42/42 [==============================] - 7s 143ms/step
              precision    recall  f1-score   support

           0       0.93      0.82      0.87       692
           1       0.83      0.93      0.88       647

    accuracy                           0.87      1339
   macro avg       0.88      0.87      0.87      1339
weighted avg       0.88      0.87      0.87      1339
```

```
42/42 [==============================] - 17s 269ms/step
              precision    recall  f1-score   support

           0       0.93      0.82      0.87       648
           1       0.85      0.94      0.89       691

    accuracy                           0.88      1339
   macro avg       0.89      0.88      0.88      1339
weighted avg       0.89      0.88      0.88      1339
```

	precision	recall	f1-score	support
0	0.88	0.84	0.86	669
1	0.85	0.89	0.87	670
accuracy			0.86	1339
macro avg	0.86	0.86	0.86	1339
weighted avg	0.86	0.86	0.86	1339

6 Future Work

In this project,we focused mainly on bert and distilbert models.One may wish to explore more regarding other model structures such as recurrent neural networks(RNN) or other transformer models.We also restricted our parameter optimization mainly to our batch size,maxlen and learning rate. There are more ktrain parameters that can be adjusted to see their effects on the validation metrics.

7 References

[1] Misra, Rishabh. ”News Category Dataset.” arXiv preprint arXiv:2209.11429 (2022).
[2] Misra, Rishabh and Jigyasa Grover. ”Sculpting Data for ML: The first act of Machine Learning.” ISBN 9798585463570 (2021).
[3] Dataset source at : rishabhmisra.github.io/publications