**NLP Case Study**

Exploratory Data Analysis can be a great way to understand the data at hand. In this report, we discuss the insights found and results obtained on the task of predicting document tags.

# Data

## Details of available Data

We have 108 json files, each of which contain several articles and related metadata in the form of a nested dictionary.

We make a few assumptions and pre-process the data before it is deemed suitable for the prescribed tasks. The task already outlines the fact that we have Technology articles from the guardian newspaper, but the data doesn’t already have an obvious field/entry called ‘tags’ for each news article. Instead, ‘tags’ is a dictionary with a few keys of which ‘sectionName’ seems indicative of tags or related topics for the articles through some steps of analysis.

For the remaining tasks, I assume that the various ‘sectionName’ values in the ‘tags’ of each article are the topics or tags that it is associated with.

# Analysis

## Pre-processing and some feature engineering

* While reading the json files, we normalize the nested fields like ‘results’, ’tags’ and ‘fields’ using in-built pandas libraries and retain attributes from the nested dictionaries which are of importance to the task at hand such as ‘sectionName’ and ‘webTitle’.
* We create a field – ‘lengthoftopics’ to indicate how many tags are typically associated with each document. Articles with no tags and more than 5 tags are eliminated for the sake of simplicity. Post imposing such a filter, we are left with 20161 articles and 15 features.
* Additionally, the ‘body’ in ‘fields’ seems to contain special characters, paragraph separators, URLs etc. But we extract human-readable text from this field using Beautiful Soup into ‘Body\_cleaned’. This feature is what we will feed into TF-IDF vectorizer and our deep learning models.

## Insights found

### Number of Tags per document

To understand how many tags a document typically is associated with, we plot a bar chart showing the number of articles in the corpus with different number of tags as shown below.

[The plot](https://colab.research.google.com/drive/1_MESpgHdyxtmTeqk_RGeT81zJgcRd42u#scrollTo=xeg87Ozrvvoh&line=2&uniqifier=1) shows that most articles are associated with two tags. Data shows that there are 17 articles with no tags and a handful of articles with more than 5 tags. For the predictive tasks, we eliminate these samples.

### Understanding the ‘type’

We notice that each sample has a ‘type’ attribute and use a bar chart to see what values this column takes and in what numbers. The plot [here](https://colab.research.google.com/drive/1_MESpgHdyxtmTeqk_RGeT81zJgcRd42u#scrollTo=uRIkCRnZxllF&line=1&uniqifier=1) shows that the samples are predominantly ‘articles’, followed by some portions of ‘video’, ’audio’, ‘gallery’, ‘liveblog’ and others.

### Examining multiple proportions at a time

Using a [***Sunburst chart***](https://colab.research.google.com/drive/1_MESpgHdyxtmTeqk_RGeT81zJgcRd42u#scrollTo=XKeVqm-ox58x&line=3&uniqifier=1) ***,*** we analyze we attempt to analyze if the number of tags associated with a sample varies based on its type; i.e. do live blogs have a different number of tags typically associated with them? From the plot, it could be hard to see but the data shows that all types have generally 2 or more tags associated. None of them with any fixed number is observed. Admittedly, this could be better observed with a simple groupby and value\_counts commands.

### Most common topics in articles

[This word cloud](https://colab.research.google.com/drive/1_MESpgHdyxtmTeqk_RGeT81zJgcRd42u#scrollTo=scbsqdrh7vLX&line=1&uniqifier=1) shows that as expected (since the articles have been scraped from the technology section of a news source), the most common topic is Technology and other noticeable ones are Culture, World and Media amongst others.

We also attempt to check if this trend is any different based on the type of the article i.e. if video articles have a different common topic, etc. but Technology remains the most common topic across all article types. These additional word cloud plots are seen in the notebook.

To quantify this, we use a count vectorizer to find how many articles each tag/topic is associated with. The most common tag in all articles is again ‘technology’. This dominates all other tags by a very large margin and is shown in [this plot](https://colab.research.google.com/drive/1_MESpgHdyxtmTeqk_RGeT81zJgcRd42u#scrollTo=W1R-UTgjKZyq&line=1&uniqifier=1).

Because of this skewed ratio, predicting that a document is tagged ‘technology’ would be he easiest and a simple rule that predicts this tag for all documents would be right 99.866% of the time for the given data.

## Predicting Tags

We’ve previously seen the most common tags and the number of tags typically associated with each article. There are a total 41 different tags and for the sake of simplicity of the problem, we remove less frequent tags and keep only the top 3 most frequent tags. This decision was made after analyzing the percentage of articles that can be explained with different number of tags using [this plot.](https://colab.research.google.com/drive/1_MESpgHdyxtmTeqk_RGeT81zJgcRd42u#scrollTo=i80EeMpqhiBz&line=1&uniqifier=1)

Estimate made on our Data shows that with just 3 tags, 99.901% of the articles can be explained. We now showcase two methods employed to predict tags associated with a given news article.

### Machine Learning with TF-IDF features

Using the ‘Body\_cleaned’ and TfidfVectorizer, we get a sparse tf-idf matrix which we feed into a linear classifier and use OneVsRestClassifier to learn a discriminative model for each class.

|  |  |
| --- | --- |
| accuracy | 0.8118026283163898 |
| macro f1 score | 0.8204814807571276 |
| micro f1 score | 0.9268511861969806 |
| hamming loss | 0.0672782874617737 |

### Deep Learning with FastAI

Fastai methods internally convert data/words into embeddings. We use an architecture called AWD-LSTM [1]. Here we attempt to use a ULMFiT approach:

1. Fine tune our language model pretrained on Wikipedia (made available by fastai) to our corpus of news articles. Purpose of this activity is domain specific familiarization. Our model is now quite good at predicting the next word given some context of a news article.
2. Transfer learning to our multi-label classification task.

This method results in an accuracy of 91.3% and the accuracy variation with different threshold values is shown in [this plot.](https://colab.research.google.com/drive/1_MESpgHdyxtmTeqk_RGeT81zJgcRd42u#scrollTo=xmRQcjkqCttD&line=2&uniqifier=1)

# Hypothetical Scenario

Consider a remote camera trap that has collected hundreds of hours of video. You’ve been asked to produce a list of timestamps and their associated animals.

* Hundreds of hours of video could mean several image-frames; many with no changes at all. So many that serving predictions for each of them could be too much to handle for an ordinary computing system.
* We could write a program to read each image-frame and simultaneously start a time-counter is incremented as we read each frame. Also, we can discard image-frames where no blobs/connected components are detected, or if the change in intensity is lower than a threshold or if the difference between current frame and initial fixed frame (background with no animals) is greater than a threshold. Quick prototyping can be done using inbuilt matlab libraries
* Using Bing Image search API, we can get images and labels for different animal types to create an image classification dataset. We can then train a CNN to classify different animals based on their images. We can utilize multiple/single GPU for training based on how many animals we need to identify and the dataset size we’d be creating. Using transfer learning would reduce the need for large amount of training data.
* Post training of the above model, we can serve predictions for each image-frame we’ve selected using tensorflow serving/REST API by feeding the image-frames in a loop.

# CPU vs. GPU?

## Fundamental Difference

A central processing unit (CPU) has several simpler cores and does well for serial processing, but newer CPUs are beginning to have higher parallelism capabilities built in.

A graphics processing unit (GPU) is designed to process graphics/3d images with great efficiency; it has many complex cores. GPUs are great for parallel processing.

GPUs are designed with thousands of processor cores running simultaneously, GPUs can perform parallel operations on multiple sets of data.

## Latency

CPUs have low latency tolerance as execution will stall when there are L1/L2 cache misses. On the other hand, GPUs can switch to different thread until data returns from memory.

## Bandwidth

The high-end NVIDIA GPUs have much, much wider buses and higher memory clock rates than any CPU and computing huge and complex jobs takes up a lot of clock cycles. While the CPU has relatively good bandwidth to its local memory and the GPU has a massive link to its memory and could mean that interaction of GPU cores with memory would be slow.

## Data and Model Parallelism

Data Parallelism is when data is distributed across machines and each subset trains a replica of a model. Gradients of small batches are calculated on each machine and the final estimate of the gradients could be the weighted average of the gradients calculated from all the small batches.

Sometimes a model and the number/size of its parameters is so large that parts of it(some layers) are stored in different machines. Gradients computed on one machine are passed on as inputs to the next and so-on there by distributing the computing burden.

## Hardware for the future of AI

With advances in areas/techniques such as few-shot/zero-shot learning, popularity in adoption of transfer learning methods, and the constant simplification of things, in my opinion we may not need to be constantly thinking of using higher compute power or parallel processing capabilities provided by GPUs as they’re only faster for single instruction multiple data type of operations and cannot fully replace CPUs.

## New question for this task

How to go about clustering similar news articles?

* Input features could be TF-IDF features or embedding vectors.
* Since the dimensionality would be high, use TSNE to visualize clusters in lower dimension

# Further steps

* Analyze/perform EDA the so-far under-utilized attributes in the dataset
* Error analysis on specific classes/tags for both machine learning and deep learning approaches
* Hyperparameter tuning and optimization of models
* Choosing metrics for evaluation of the models – could’ve been a bit more thorough with additional time
* Theory answers – due to lack of time, answers were not detailed enough. With additional time, explanation could be strengthened. Especially the answer for the hypothetical scenario. Additional research on blob detection or any method to reduce the number of image frames could’ve been done for more confidence.
* Clustering analysis

# References

[1] <https://arxiv.org/pdf/1708.02182.pdf>