Article Class Classification

Task:

1. Develop a train and test set from the given dataset, perform EDA and build a document classification model to correctly classify a given document.
2. Project the document to an embedding space using any embedding models like a glove, bert etc.. and perform visualization.
3. Quantity and visualize the results using appropriate evaluation metrics.

Dataset:

The dataset consists of 2225 documents from the BBC news website comprising stories in five topical areas for the annual year 2004-2005.
Each document corresponds to news and we have 5 classes - business, entertainment, politics, sports, tech.
One can find the document related to a specific class in their respective folders. i.e, documents related to entertainment class can be found in the entertainment folder and likewise for the other classes.
Here is the link to download the dataset.

Approach :

Initially, it was a challenging task for me to figure out where I should start. Subsequently after scrupulously analyzing my last project "Fake_News_Detector using LIAR PLUS dataset", it gave me a starting approach i.e. to tokenize the text and using TF-IDF vector representation.

Data Exploration

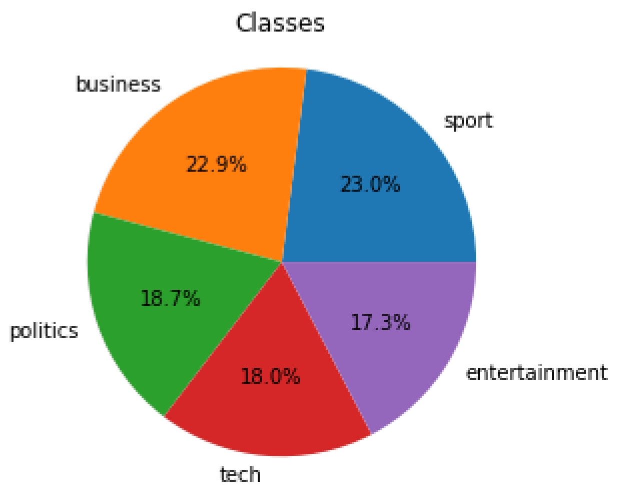
"Initial data analysis is the process of data inspection steps after data collection."

- After downloading the whole dataset, I made the systematic arrangement of the five folders provided to further facilitate the formal statistical analysis. Moreover, I came across that the whole dataset comprises of five classes with one article embedded in each text file.

- Afterward, looking into how an article is saved in each text file, I concluded the following:
    1. Text is saved in bytes format and hence to be decoded into string type.
    2. There are unnecessary characters and tags like \n,' ' and \, thus we need to remove them.
    3. There are also unnecessary symbols which don't have any significance in our article classification like ?, : , ; , etc.
    4. Also, there are lots of stopping words like you, ourselves, ours, it, it's, etc will be further removed.

    I solved these later on in the cleaning data section.

- Next step in consideration is the dataset creation, which gives an overview of the whole data. I created a pandas dataframe containing columns as File_Name, Text, Raw_Text_Length, and Class_name.
- To calculate how the articles are distributed across each class/category, I plotted a Pie Chart for the same.



- From the Pie Chart above, I concluded that the data is almost equally distributed across each category.

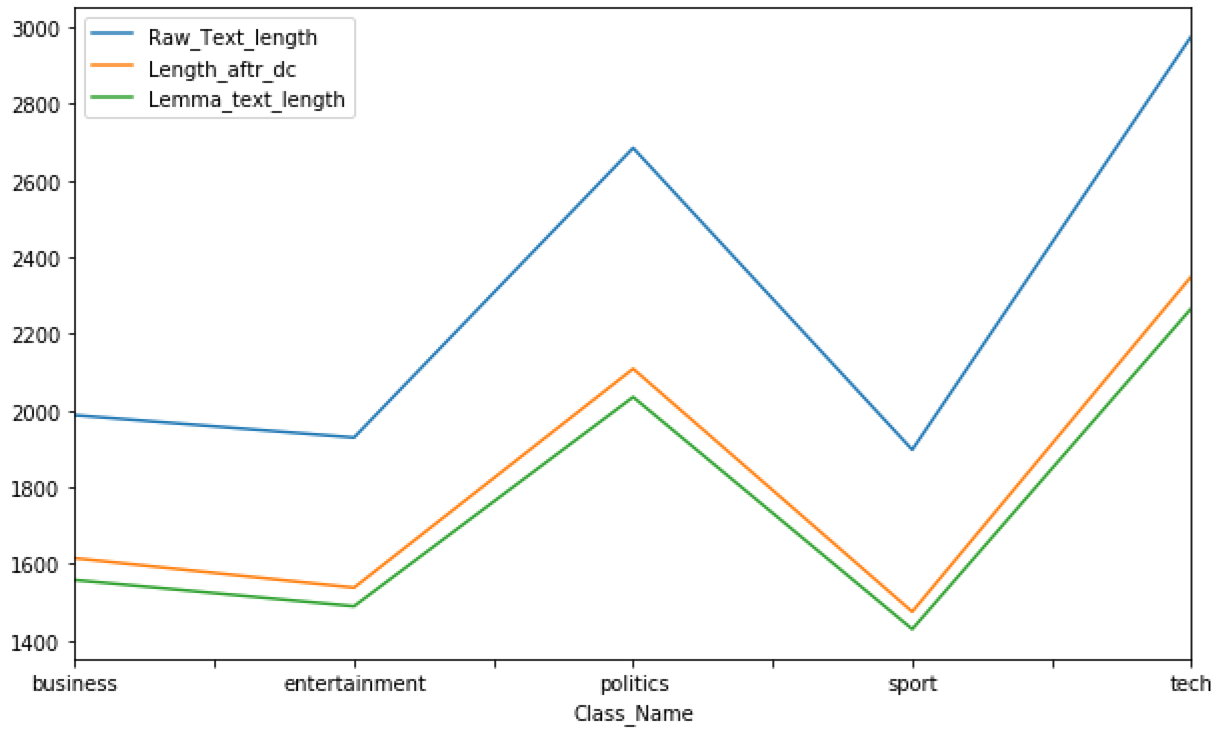
Data Cleaning

- Decoded the text data which was in byte form into string type by using ancient MS-DOS cp437 encoding.
- Removed '\n' and 'double space'.
- Replaced unnecessary symbols like ' 's','?', '!',';',':','.' and ','.

- Since, to treat the words like ‘My’ and ‘my’ equally, I lowercase the whole text data.
- Using regular expressions and ‘english’ stop-words database, I identified the stop-words present in my text data and hence removed them.
- Created two new columns, containing updated_text and updated_length.

Text Normalization

- To keep the dimensionality of vector representation of our words low, I used the Text Normalization technique of NLP called *Lemmatization* because using Stemming may result in words that are not actual words. Hence, Lemmatizaton would reduce our words or tokens into their Lemmas keeping the dimensionality of the vector representation low.
- I used a pre-defined dictionary called *WordNet* for extracting lemmas.
- Created two new columns, containing Lemma_text and Lemma_text_length.
- Further, to compare between how our data of article texts have been changed after data cleaning and text Normalization process I created three lists named as ‘Average article length of each class before data cleaning and lemmatization’, ‘Average article length of each class after data cleaning’ and ‘Average article length of each class after lemmatization’ ; and plotted a graph for the same:



- From the above graph, we concluded that how the average length of each class' article has been changed after Data cleaning and Text Normalization process.
- Created final data-frame containing only updated columns.

Feature Engineering

- Encoded class_Name column of dataframe into One-Hot encoding representation and saved a new column 'class_id' containing the same.
- Split our dataset into training and test dataset with a division of 75% and 25% respectively.

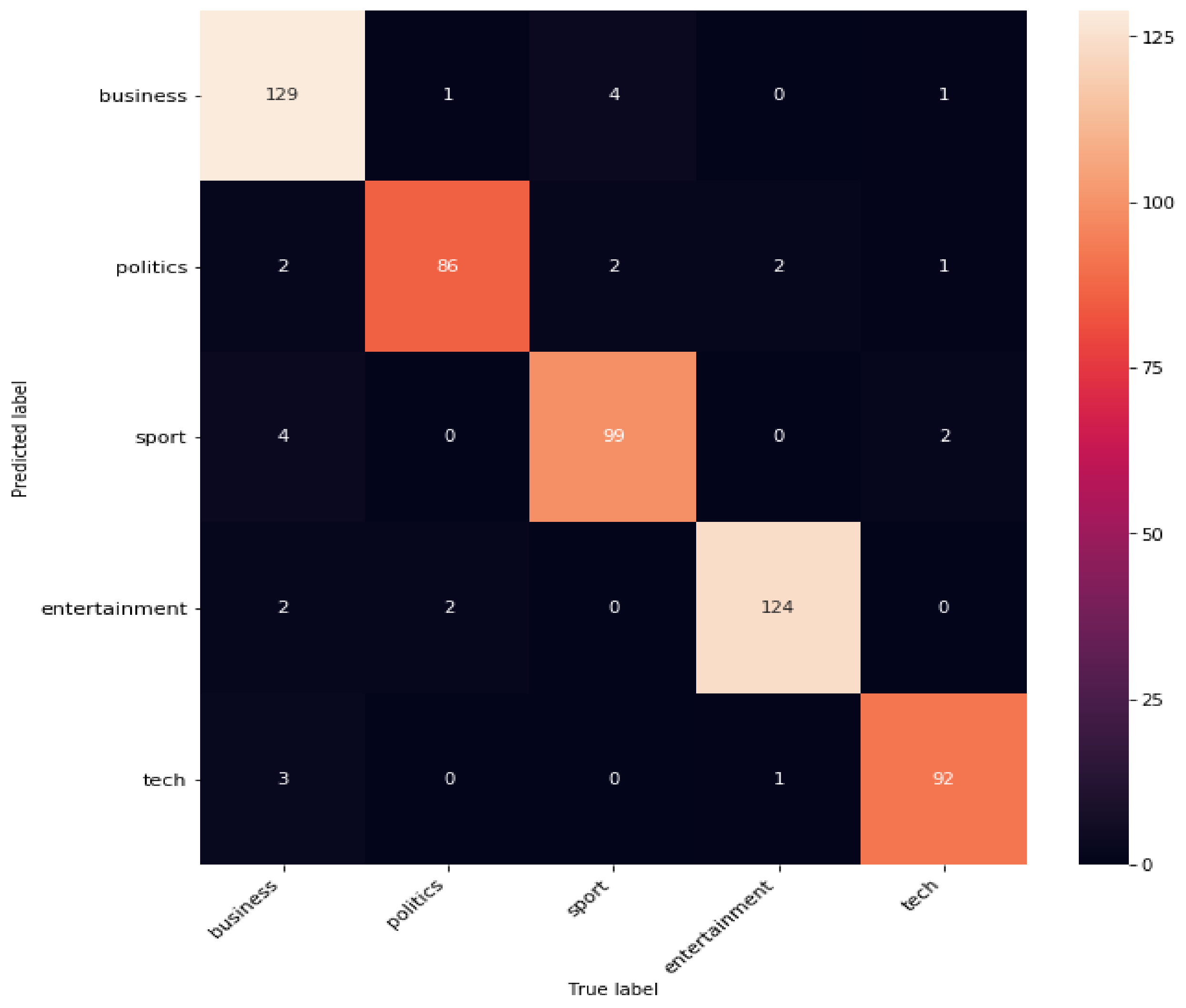
Text Representation

- To transform our training and test dataset into a matrix of TF-IDF features, I used the vectorizer method of TfidfVectorizer class, passing appropriate parameters values.

Building Classifier

- To build up a perfect classifier for our article class classification task, I used **Random Forest Classifier.**
- I selected few hyper-parameters like :
  n_estimators (The number of trees in the forest),
  max_depth (The maximum depth of the tree),
  min_samples_split (The minimum number of samples required to split an internal node),
  min_samples_leaf (The minimum number of samples required to be at a leaf node),
  bootstrap (Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree),
  max_features (The number of features to consider when looking for the best split),
- And to find the best values of these hyper-parameters I used Randomized Search Cross Validation method.
- I passed the obtained best values of hyper-parameters into my RandomForest Classifier and trained it with our training dataset and got the classification accuracy of **95.15%** on the test dataset.

- Since the classification accuracy alone can be misleading as we have an unequal number of text files in each class, I used Confusion Matrix for further visualization and evaluation of the performance of our model.



- The above heat map of confusion matrix reveals the insight of the errors that are being made by our classifier and the types of errors that are being made (like true-positive, false-positive, etc).

<u>Using the embedding Model GloVe</u>

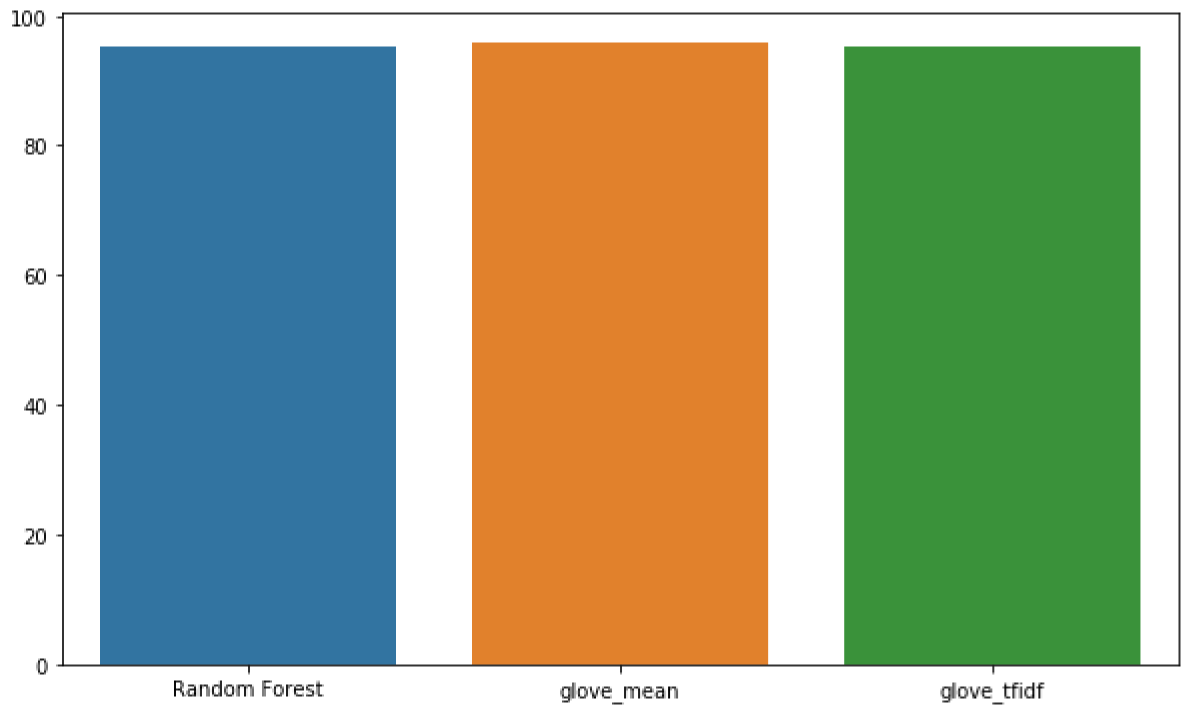
- Initially, I split the text and saved them into a list after removing the duplicates. This gave me a total of 53195 words in our whole dataset.
- I loaded the embedding model and prepared the word-embeddings from GloVe Model.

<u>Building Features</u>

- First by averaging word vectors from GloVe Model of all words in a text file and by using these features I got a classification accuracy of **95.77%**.
- Then by using TF-IDF weighting method where firstly the vector of a word is multiplied by it's IDF and then averaging these word vectors from GloVe Model of all words in a text file, I got an accuracy of **95.27%**.

Results:

- It is clear from the classification accuracy scores that all three models are giving almost equal accuracy.



<u>Further improvements:</u>

- We can also use other models of machine learning, and train them to get more accuracy.
- Certain other embedding models like BERT can also be used instead of GloVe Model.
- By varying the number of hyper-parameters to be tuned in a machine learning model, we can also increase the classification accuracy.
- Concluding more on this, if there is being provided a larger dataset, we can also use Neural Networks and Deep Learning methods to get maximum classification accuracy.

Resources:

- **NLP with PyTorch** *by* Delip Rao & Brian McMahan.
- Confusion Matrix in Machine Learning: https://machinelearningmastery.com/confusion-matrix-machine-learning
- StackExchange Thread: https://datascience.stackexchange.com/questions/42247/how-can-i-parallelize-glove-reverse-lookups-in-pytorch
- Randomize Search Cross-Validation: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
- RandomForestClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html