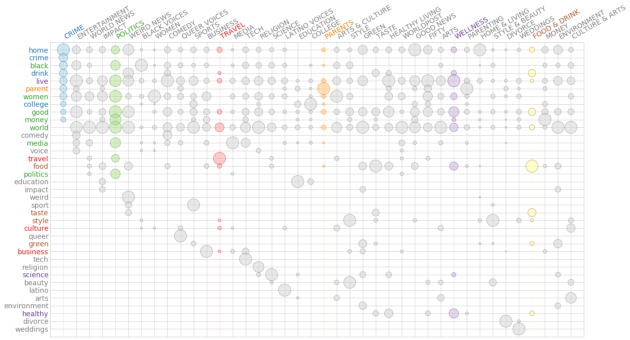
**A Basic NLP Tutorial for News Multiclass Categorization**

Natural Language Processing, Support Vector Machine, TF- IDF, deep learning, Spacy, Attention LSTM

[Armand Olivares](https://medium.com/@armandj.olivares?source=post_page-----82afa6d46aa5--------------------------------)

[Armand Olivares](https://medium.com/@armandj.olivares?source=post_page-----82afa6d46aa5--------------------------------)

[Jun 8, 2019·6 min read](https://medium.com/@armandj.olivares/a-basic-nlp-tutorial-for-news-multiclass-categorization-82afa6d46aa5?source=post_page-----82afa6d46aa5--------------------------------)



Let’s understand how to do an approach for multiclass classification for text data in Python through identify the type of news based on headlines and short descriptions.

Introduction

Text or document classification is a machine learning technique used to assigning text documents into one or more classes, among a predefined set of classes. A text classification system would successfully be able to classify each document to its correct class based on inherent properties of the text.

1. Getting Ready

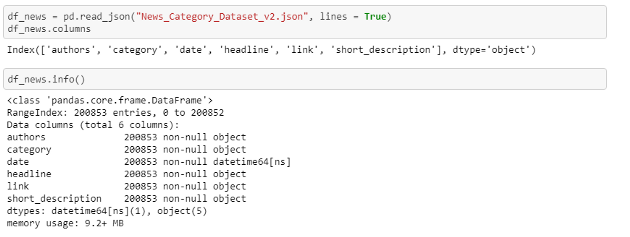
For this article we will need **Python 3.6, Spacy, NLTK, texblob**. If you do not have it yet, please install all of them.

2. Training a Custom Text Classifier

We will use [Kaggle’s News Category Dataset](https://www.kaggle.com/rmisra/news-category-dataset/kernels) to build a categories classifier with the libraries **sklearn** and **keras** for deep learning. This dataset contains around 200k news headlines from the year 2012 to 2018 obtained from [HuffPost](https://www.huffingtonpost.com/).

2.1 Preprocessing — Building the Dataset

We need to download the data from the [kaggle site](https://www.kaggle.com/rmisra/news-category-dataset), then we can use the following function to load the dataset and to check it.

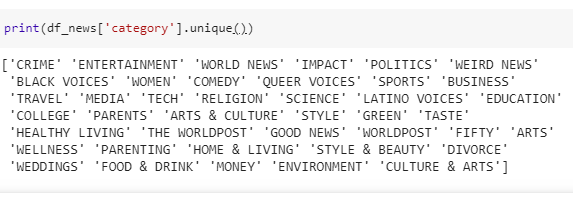


The first rows of the data:



There are about 200K rows and 6 columns, so for this exercise we will build a classifier with only the columns *headline*and *short\_description*, since our to predict variable is *category.*

Examining the categories we see that there are **41 categories**

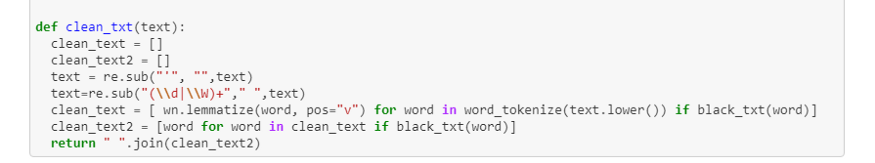


However we need to merge the categories *WORDPOST* with *THE WORDPOST*, as they are basically the same, next we will combine the columns *headline* with *short\_description* into a new column call *text;* this will be our predictor text.



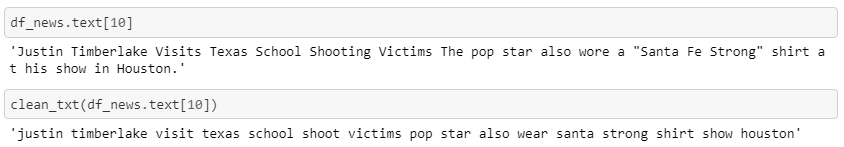
Now for this example as basic pre-processing:

* remove the punctuation from text (ex: .,:)
* make lowercase because we assume that punctuation and letter case don’t influence the meaning of words.
* use **NLTK**package remove the called stop\_word, i.e frecuent words that doesn’t add information to our classifiers, example of stop word are: *our, you, yourself, he, his, she,them*etc. you can review the complete list on this [link](https://gist.github.com/sebleier/554280).
* make lemmatization to words, lemmatization is a process of extracting a root word by considering the vocabulary. For example, “good”, “better”, or “best” is lemmatized (changed) into “good”.



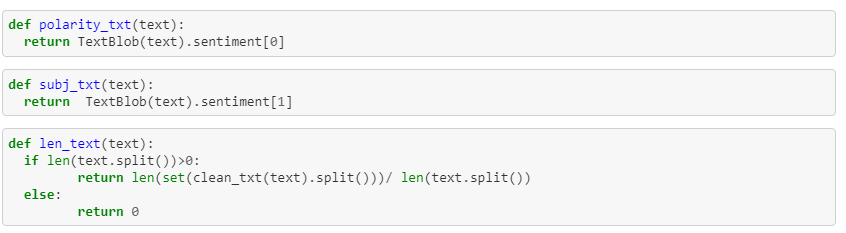
function to clean the text

Let’s check how our cleaning function is working by comparing a row from the data before and after applying the cleaner:



Next we are going to create some news variables columns (like metadata) to try to improve the quality of our classifier with the help of **textblob package**, we will create:[¶](https://render.githubusercontent.com/view/ipynb?commit=ccd97ff1559ec3f16fc1cbf1c510af8a89302730&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f41726d616e6444532f6e6577735f63617465676f72792f636364393766663135353965633366313666633163626631633531306166386138393330323733302f4e6577735f416e616c797369735f414f2e6970796e62&nwo=ArmandDS%2Fnews_category&path=News_Analysis_AO.ipynb&repository_id=189287566&repository_type=Repository#Next-we-are-going-to-create-some-news-variables-columns-to-try-to-improve-the-quaity-of-our-classifier,-we-will-create:)

* Polarity: to check the sentiment of the text
* Subjectivity: to check if text is objective or subjective
* Len: The number of word in the text



helper extracting metadata functions

2.2 Vectorization

Now, we need to find a way to transform these word sequences into numerical features: *vectorization*, in this article we will use the TF-IDF technique.

TF-IDF stands for Term Frequency-Inverse Document Frequency, a combination of two metrics: term frequency and inverse document frequency, and the idea is to weigh down the frequent terms while scaling up the rare or less frequent ones.

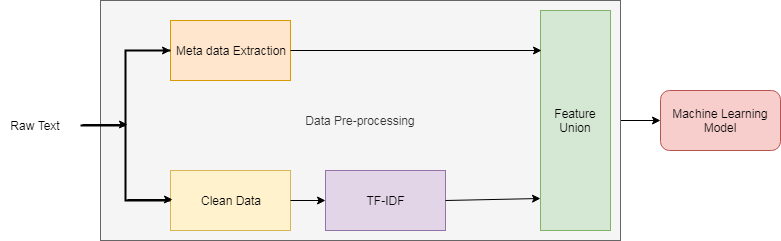
For Vectorization with TF-IDF we using the python package **sklearn.**

To know more about tf-idf please refer to [this wikipedia article](https://en.wikipedia.org/wiki/Tf%E2%80%93idf).

2.3 Features union

Since our dataset now consists of heterogeneous data types (text vector and metadata columns) that requires different feature extraction and processing pipelines, we must implement a custom pipeline with custom feature union.

The entire preprocessing pipeline is show below:



And the code for the pipeline:



2.4 Machine Learning Models

The final step in the text classification framework is to train a classifier using the features created previously. Our first approach is explore some “traditional” machine learning models like support vector machines (SVM) or Stochastic Gradient Classifier, both are implemented in the sklearn package.



We see that the best model is support vector classifier which score around **60%**of accuracy.

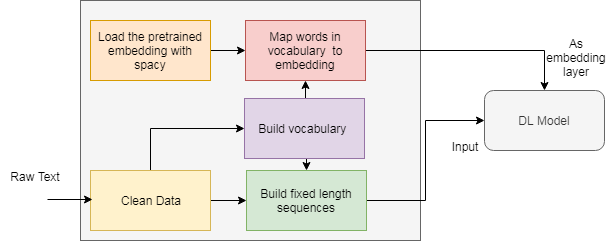
2.5 Deep Learning Models

Next step is explored some deep learning models looking for a better accuracy, but we need to modify our data to feed the models.

For this kind of models the data input will pass through a word embedding layer, a word embedding is a form of representing words and documents using a dense vector representation, so we need build the embedding using the following:

* Use tokenizer methods from the vectorizer tf-idf step
* make a vocabulary (limited to a number of words)
* make the text to sequence to convert words to numbers
* make fixed length sequences (for this particular exercise we selected a sequence length of 60).
* Load the pretrained word embeddings
* build the vector embedding with **spacy,**it meansmapping tokens to their respective embeddings

The processes to tranform the data to feed a DL model is rough summarized in the following diagram:



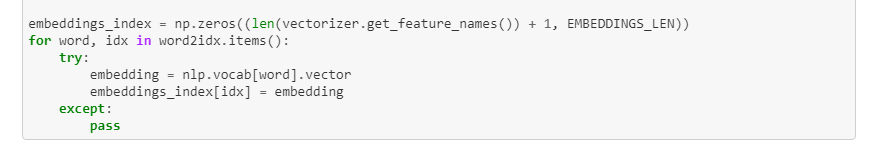
We are using the **spacy** pretained embedding, you can download the pre-trained word embeddings by executing:

!python -m spacy download en\_core\_web\_lg

And then load:

nlp = spacy.load('en\_core\_web\_lg')

Finally, the code to build the embedding is:

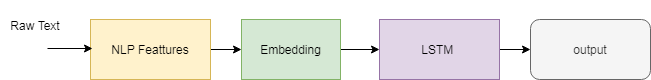


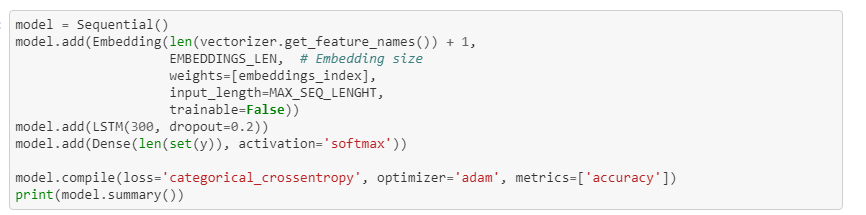
If you need to know more about word embeddings you can check this [article](https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa) and [spacy](https://spacy.io/usage/vectors-similarity/) .

Now we have the embedding is time to build/train the Deep learning models

**First model: simple LSTM**

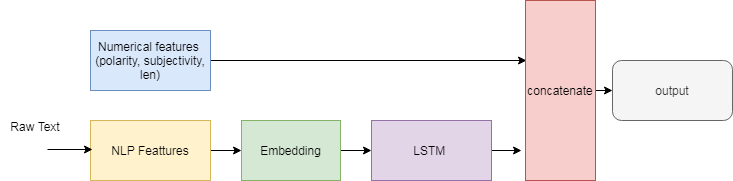
The structure of this model is:





**Second model: LSTM adding the metadata features**

In all the following models we will used the metadata columns to improve the models, so the architecture is as follow:



*note the above diagram is a general one, for simplicity we don’t show anothers layers like: dense, dropout, batchNormalization.*



**Third model: GRU with metadata features**

Just changed the LSTM layer by 2 GRU Layers



**Fourth model: LSTM with Attention NN and metadata features**

Added an [attention layer](https://gist.github.com/mbollmann/ccc735366221e4dba9f89d2aab86da1e) to the LSTM Network



3. Models Comparison

We must validate the models with the test dataset and compare them:



The best model is fourth model (LSTM with attention, the best on kaggle achive **65%**)

Final Thoughts

This article should give you a rough understanding of how to approach for text multiclass classification.

In order to improve the metric, you can make a better preprocessing and try more advanced techniques like BERT, ELMO, FastText, etc.

A more completed analysis and the code can be found on [this Jupyter notebook](https://github.com/ArmandDS/news_category/blob/master/News_Analysis_AO.ipynb), and you can browse for more projects on my [Github](https://armandds.github.io/).

If you need some help with Data Science related projects: <https://www.disruptio-analytics.com/>